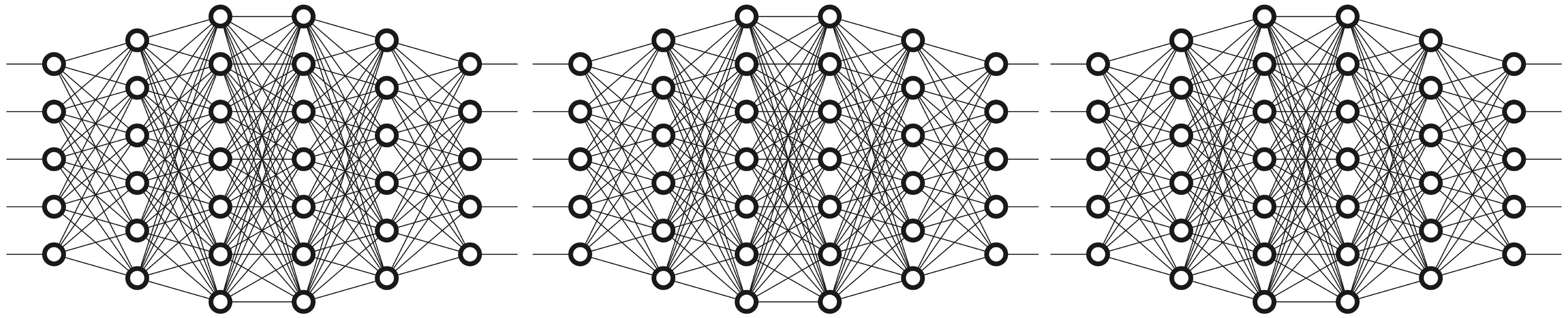
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 21: Neural Networks (cont), CNNs

Menu for Today

Topics:

- Backpropagation
- Convolutional Layers
- Pooling Layer?

Redings:

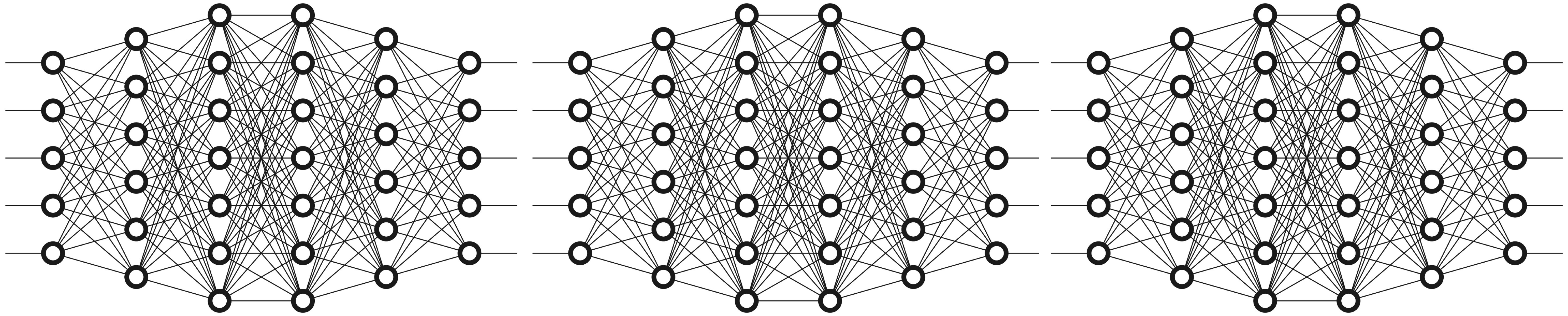
- **Today's** Lecture: N/A
- **Next** Lecture: N/A

Reminders:

- **Assignment 6**: Deep Learning is **out**
- **Material** for **Final Prep** will make available on Canvas this weekend

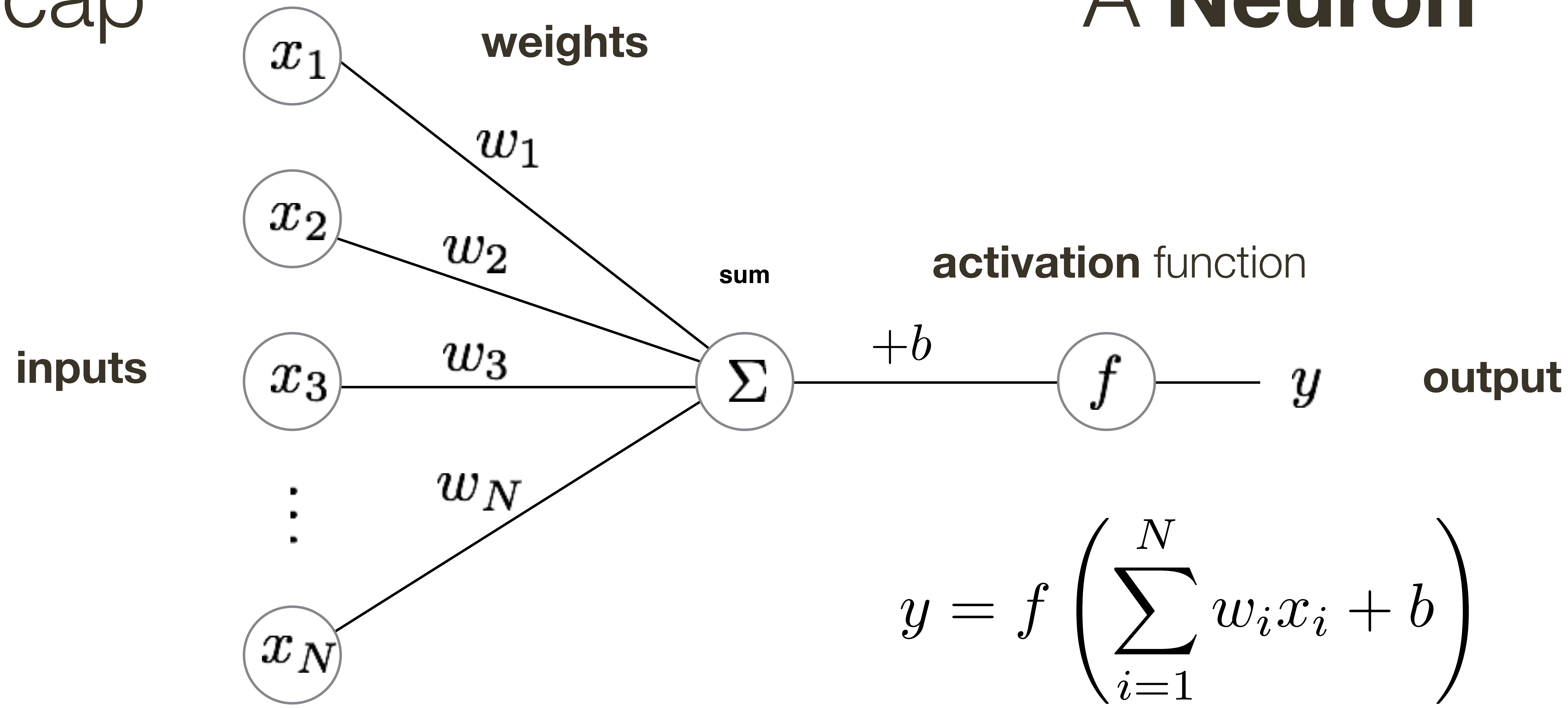
What we **have seen** so far …

- Started from talking about **linear classification** / **regression**
- Defined a **neuron**
- Defined **neural network** (how to build one from neurons) and terminology
- Discussed **properties** of neural networks (light theory and universality)
- **Learning parameters** of neural network
  - Stochastic Gradient Discent
  - Computational Graph and Gradients



Lecture 20: Re-cap

A **Neuron**



$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

— The basic unit of computation in a neural network is a neuron.

— A neuron accepts some number of input signals, computes their weighted sum, and applies an **activation function** (or **non-linearity**) to the sum.

— Common activation functions include sigmoid and rectified linear unit (ReLU)

A neural network comprises neurons connected in an acyclic graph

The outputs of neurons can become inputs to other neurons

Neural networks typically contain multiple layers of neurons

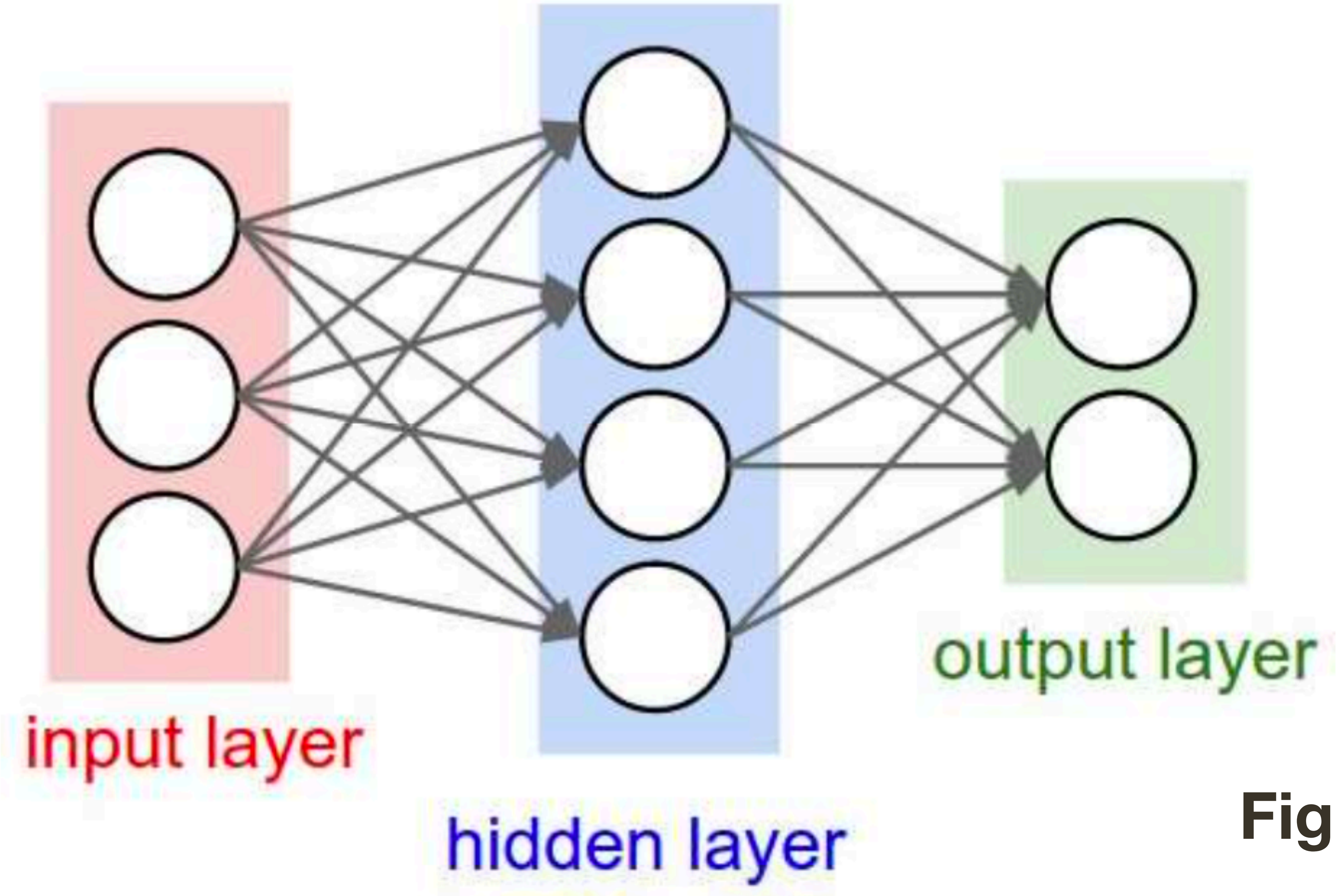


Figure credit: Fei-Fei and Karpathy

Example of a neural network with three inputs, a single hidden layer of four neurons, and an output layer of two neurons

Lecture 20: Re-cap

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

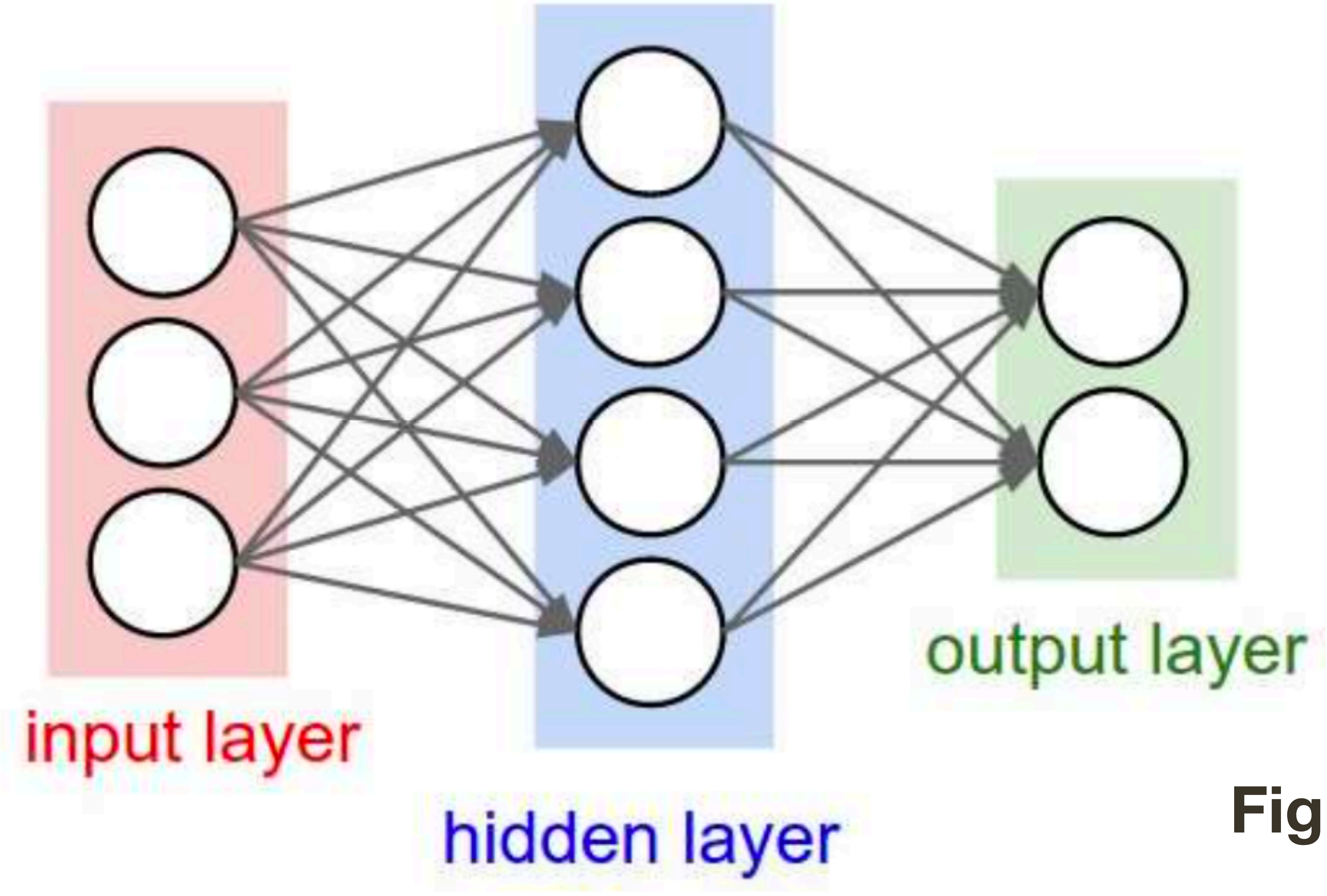


Figure credit: Fei-Fei and Karpathy

Lecture 20: Re-cap — **Neural** Network

Note: each neuron will have its own vector of weights and a bias, its easier to think of all neurons in a layer as a single entity with a matrix of weights (size = number of inputs x number of neurons) and a vector of biases (size = number of neurons)

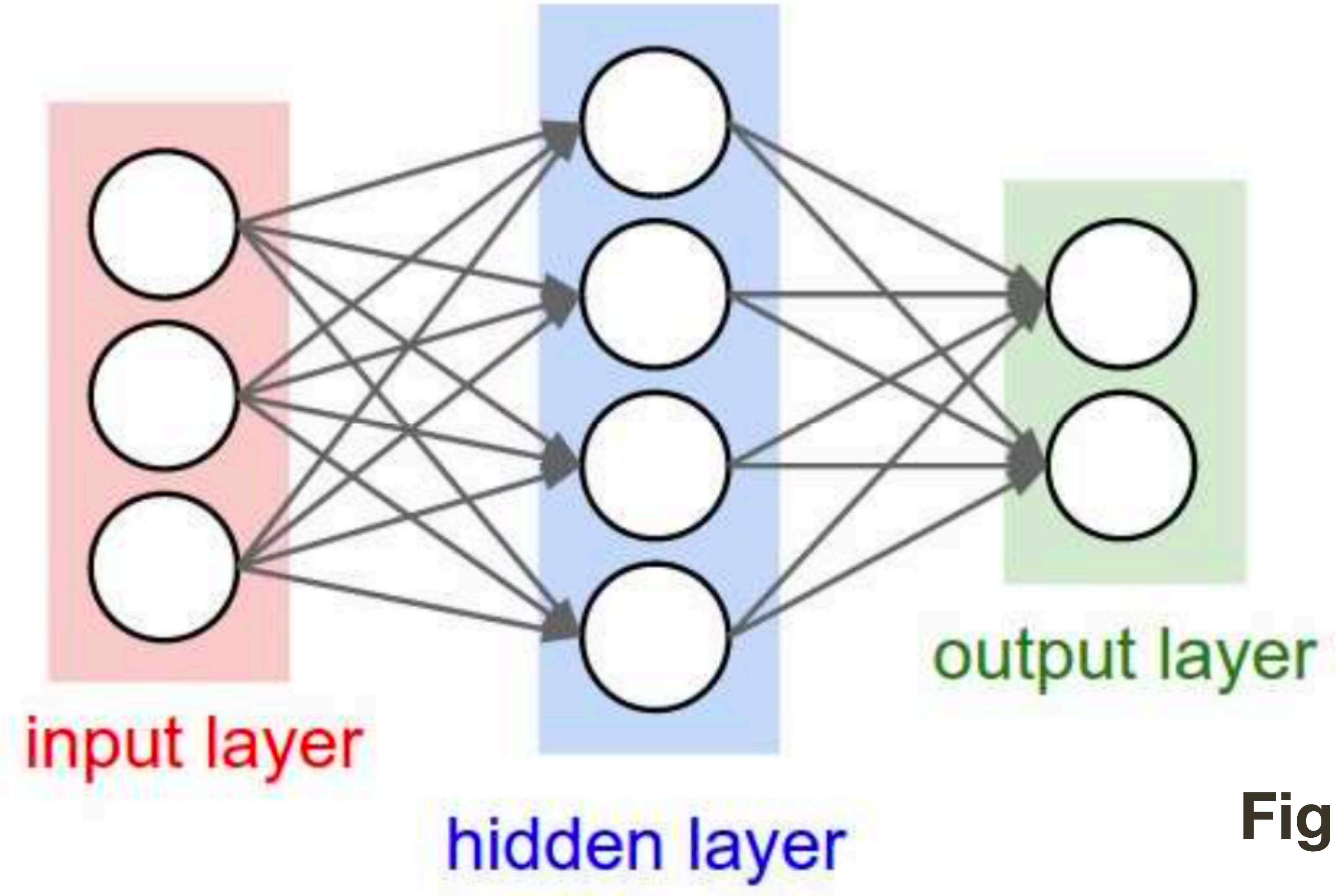


Figure credit: Fei-Fei and Karpathy

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

Activation Function

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \sigma\left(\mathbf{W}_2^{(2\times4)}\sigma\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}\right)$$

$$= \mathbf{W}_2^{(2\times4)}\left(\mathbf{W}_1^{(4\times3)}\mathbf{x} + \mathbf{b}_1^{(4)}\right) + \mathbf{b}_2^{(2)}$$

$$= \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{W}_1^{(4\times3)}}_{\mathbf{W}_*^{(2\times3)}}\mathbf{x} + \underbrace{\mathbf{W}_2^{(2\times4)}\mathbf{b}_1^{(4)} + \mathbf{b}_2^{(2)}}_{\mathbf{b}^{(2)}}$$

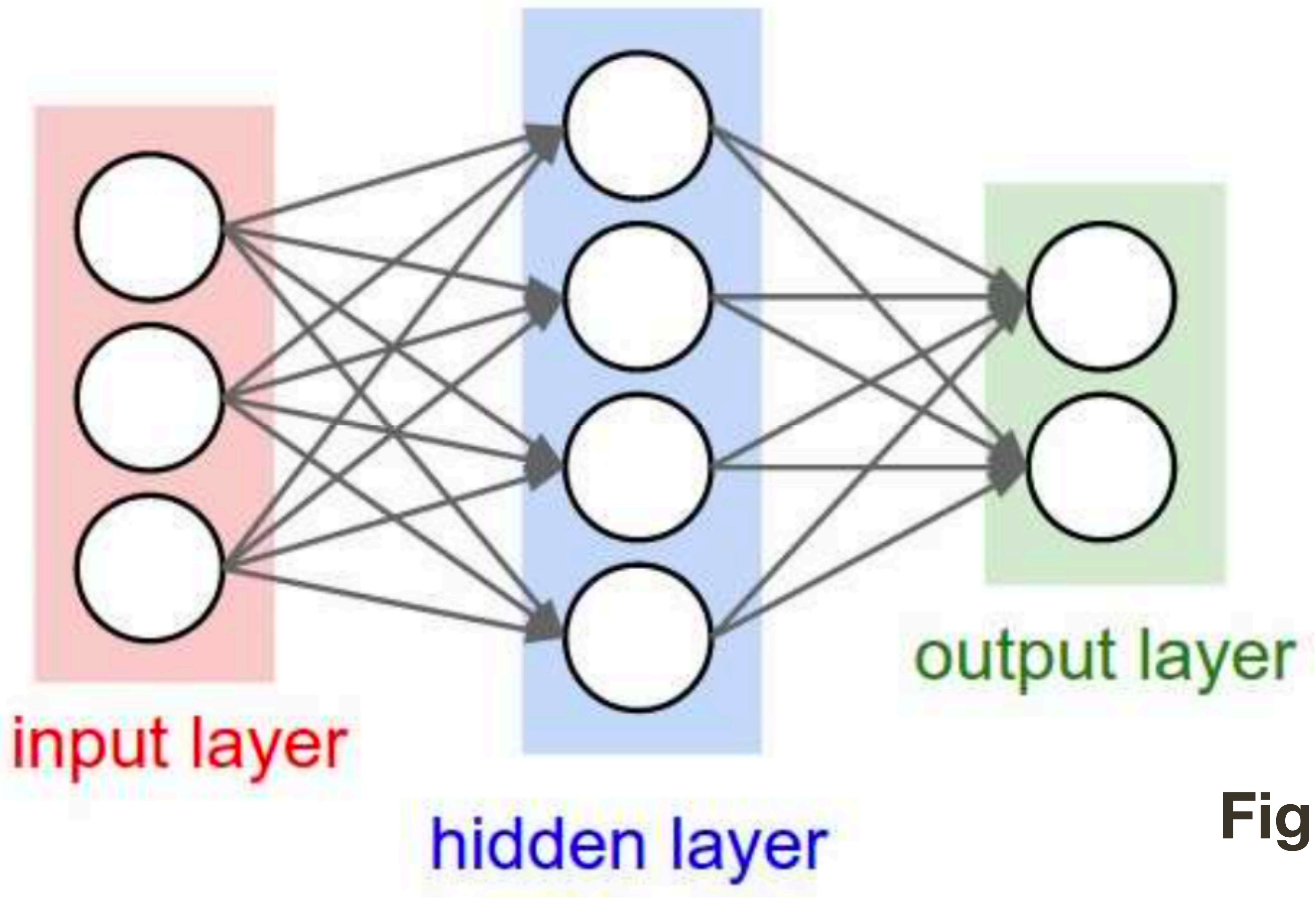


Figure credit: Fei-Fei and Karpathy

Light Theory: Neural Network as Universal Approximator

Conditions needed for proof to hold: Activation function needs to be well defined

$$\lim_{x \to \infty} a(x) = A$$

$$\lim_{x \to -\infty} a(x) = B$$

$$A \neq B$$

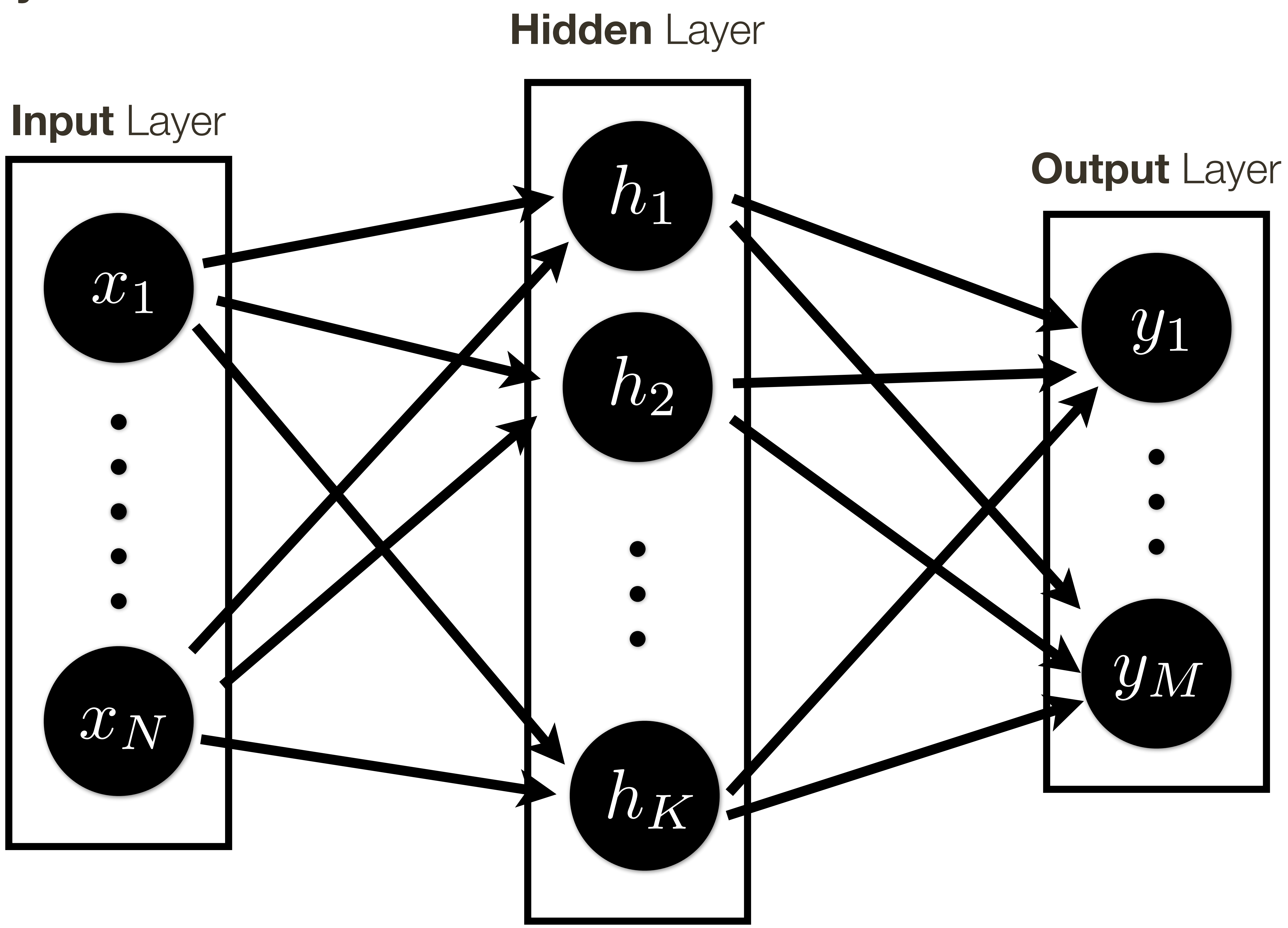
Note: This gives us another way to provably say that linear activation function cannot produce a neural network which is an universal approximator.

*slide adopted from http://neuralnetworksanddeeplearning.com/chap4.html

Light Theory: Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]



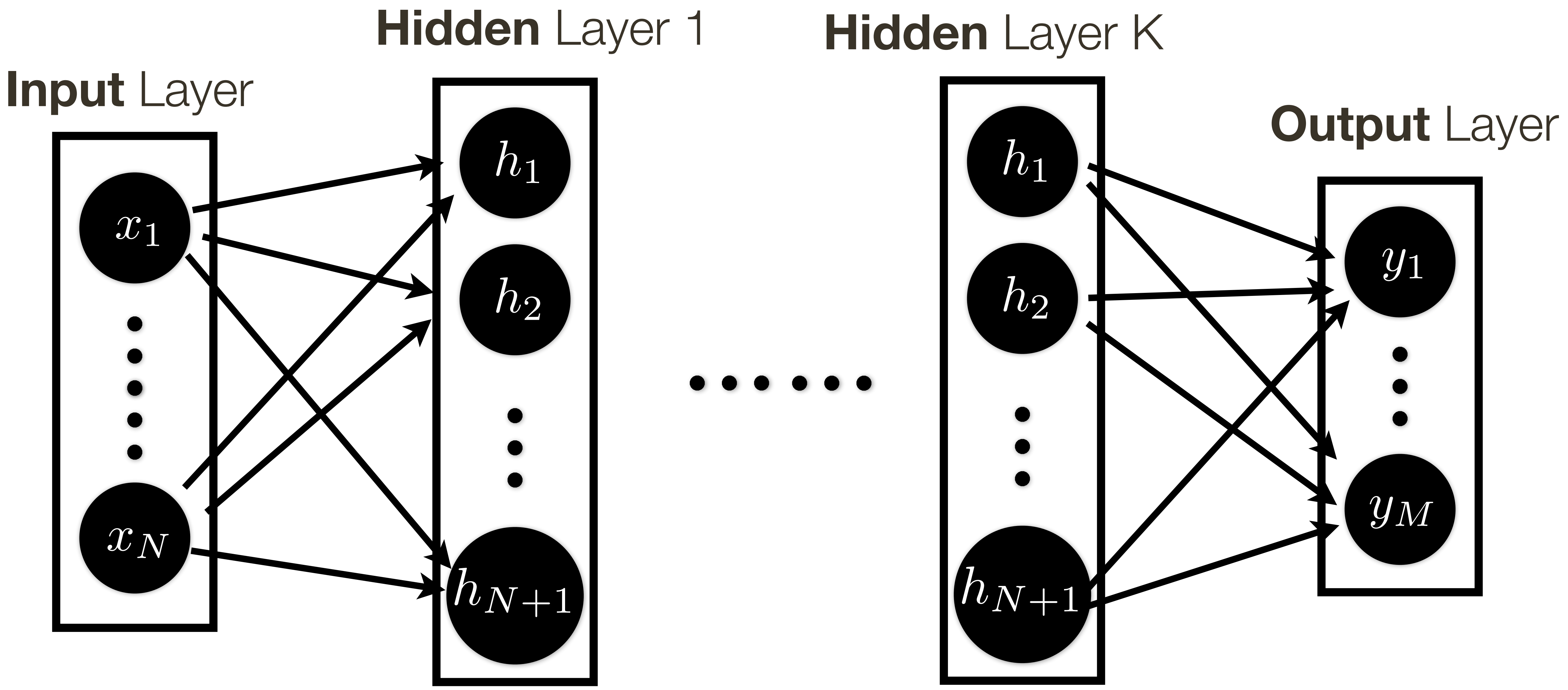
Light Theory: Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]

Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size $d + 1$ neurons, where d is the dimension of the input space, can approximate any continuous function.

[Lu *et al*., NIPS 2017]



Light Theory: Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]

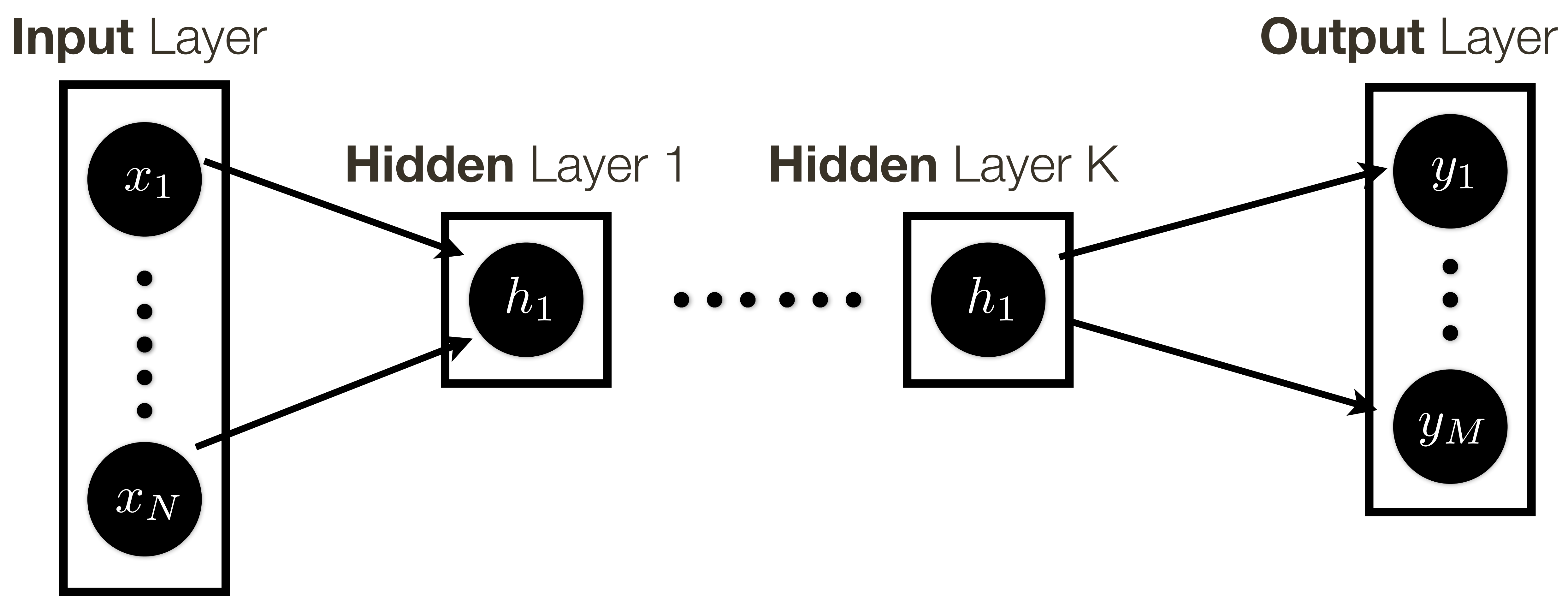
Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size $d+1$ neurons, where d is the dimension of the input space, can approximate any continuous function.

[Lu *et al*., NIPS 2017]

Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function.

[Lin and Jegelka, NIPS 2018]

Light Theory: Neural Network as Universal Approximator



Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function.

[Lin and Jegelka, NIPS 2018]

Neural Networks

Modern **convolutional neural networks** contain 10-20 layers and on the order of 100 million parameters

Training a neural network requires estimating a large number of parameters

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network
(hidden layers)

Prediction
(score)

Training a Neural Network

Output layer (prob)

$$\hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$$

$o(\mathbf{x})$

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network (hidden layers)

Prediction (score)

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

Training a Neural Network

Output layer
(prob)

$$\hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$$

$$o(\mathbf{x})$$

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized **Input**

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

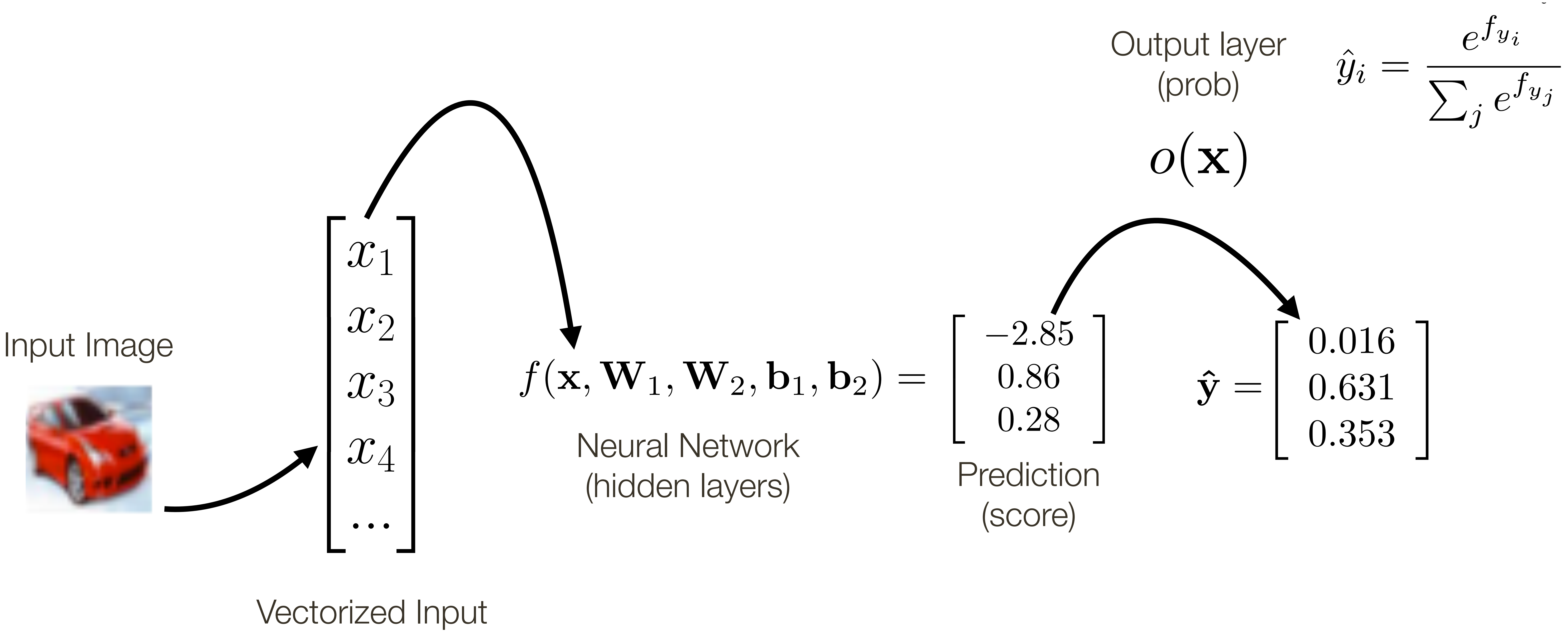
Neural Network
(hidden layers)

Prediction
(score)

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

Input and **output** layers (size and form) are dictated by the problem, intermediate hidden layers have few constraints and can be *anything*

Training a Neural Network



Inference: $o(f(\mathbf{x}, \cdots))$

Training a Neural Network

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix}$$

Vectorized Input

$$f(\mathbf{x}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}$$

Neural Network
(hidden layers)

Prediction
(score)

Output layer
(prob)

$$\hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$$

$$o(\mathbf{x})$$

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

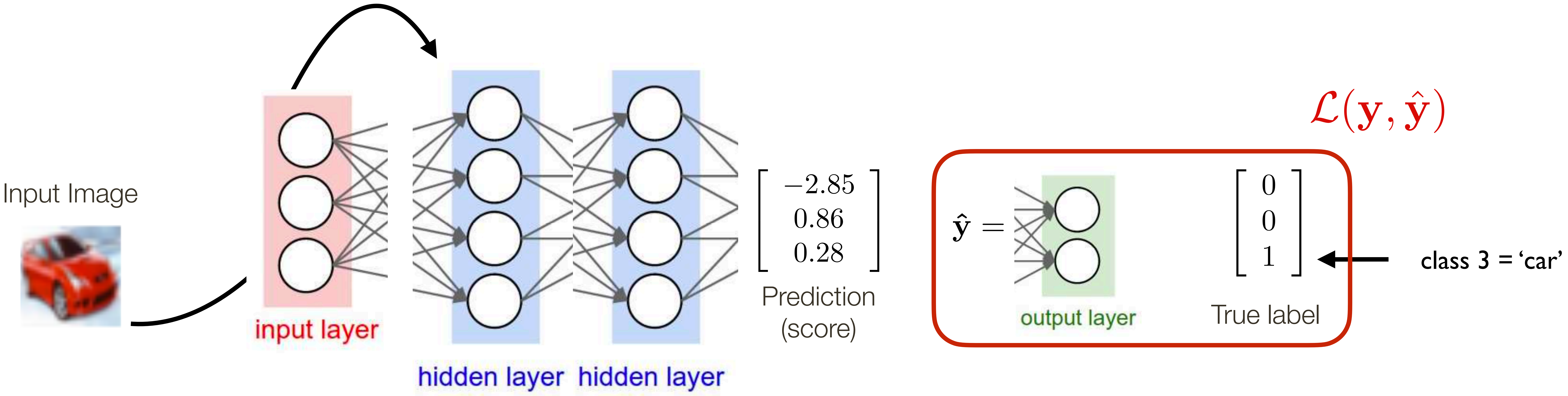
True label

class 3 = ‘car’

Inference: $o(f(\mathbf{x}, \cdots))$

Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

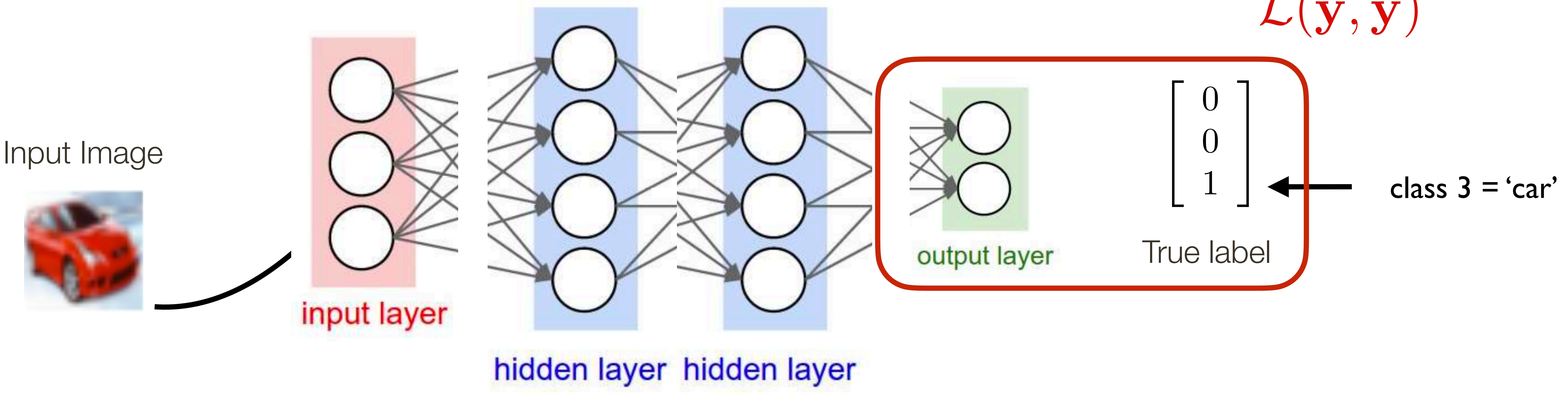
Training a Neural Network



Inference: $o(f(\mathbf{x}, \cdots))$

Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

Training a Neural Network



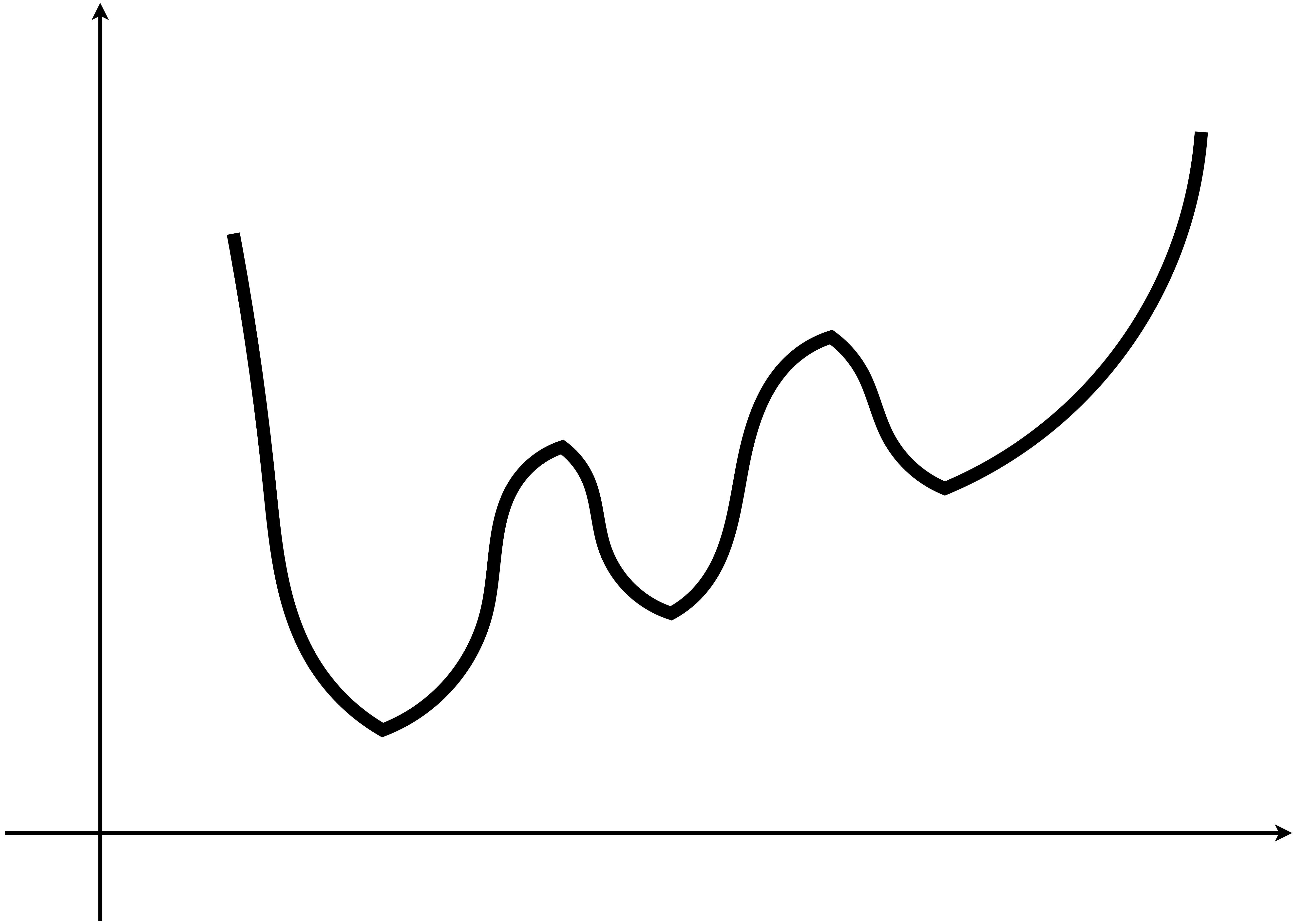
$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Inference: $o(f(\mathbf{x}, \cdots))$

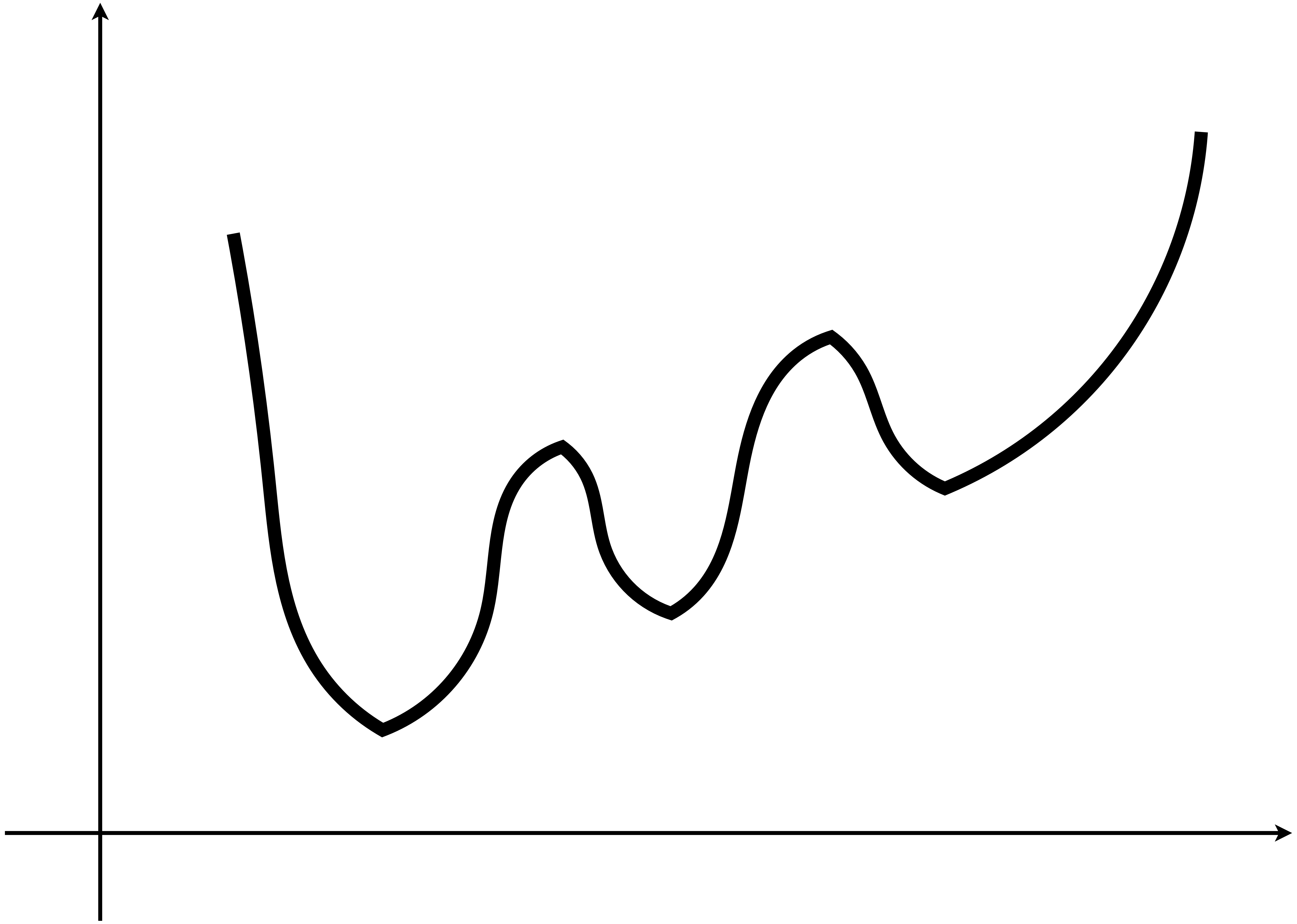
Learning: $\mathcal{L}(\mathbf{y}, o(f(\mathbf{x}, \cdots)))$

Gradient Descent



*slide adopted from V. Ordonex

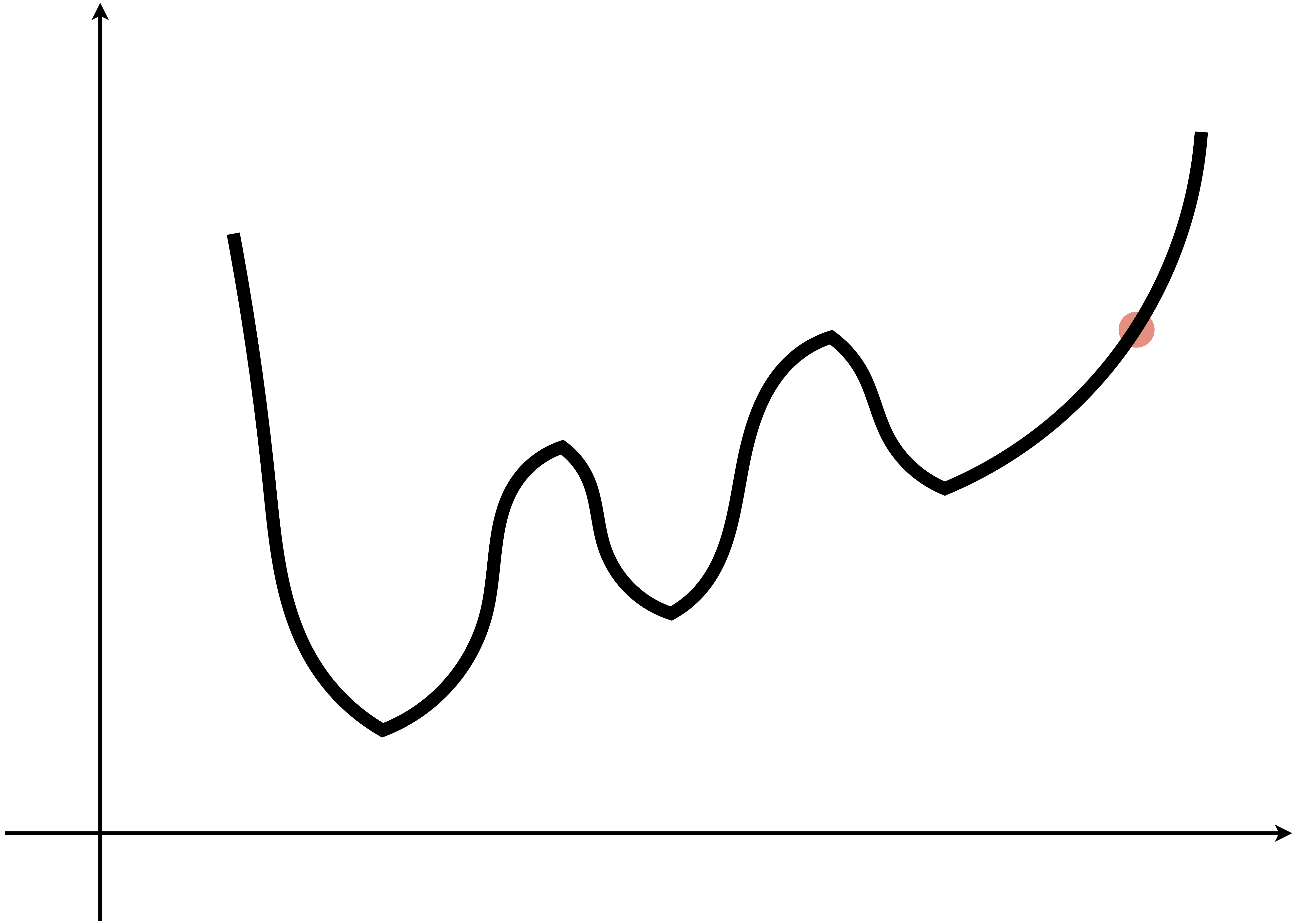
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

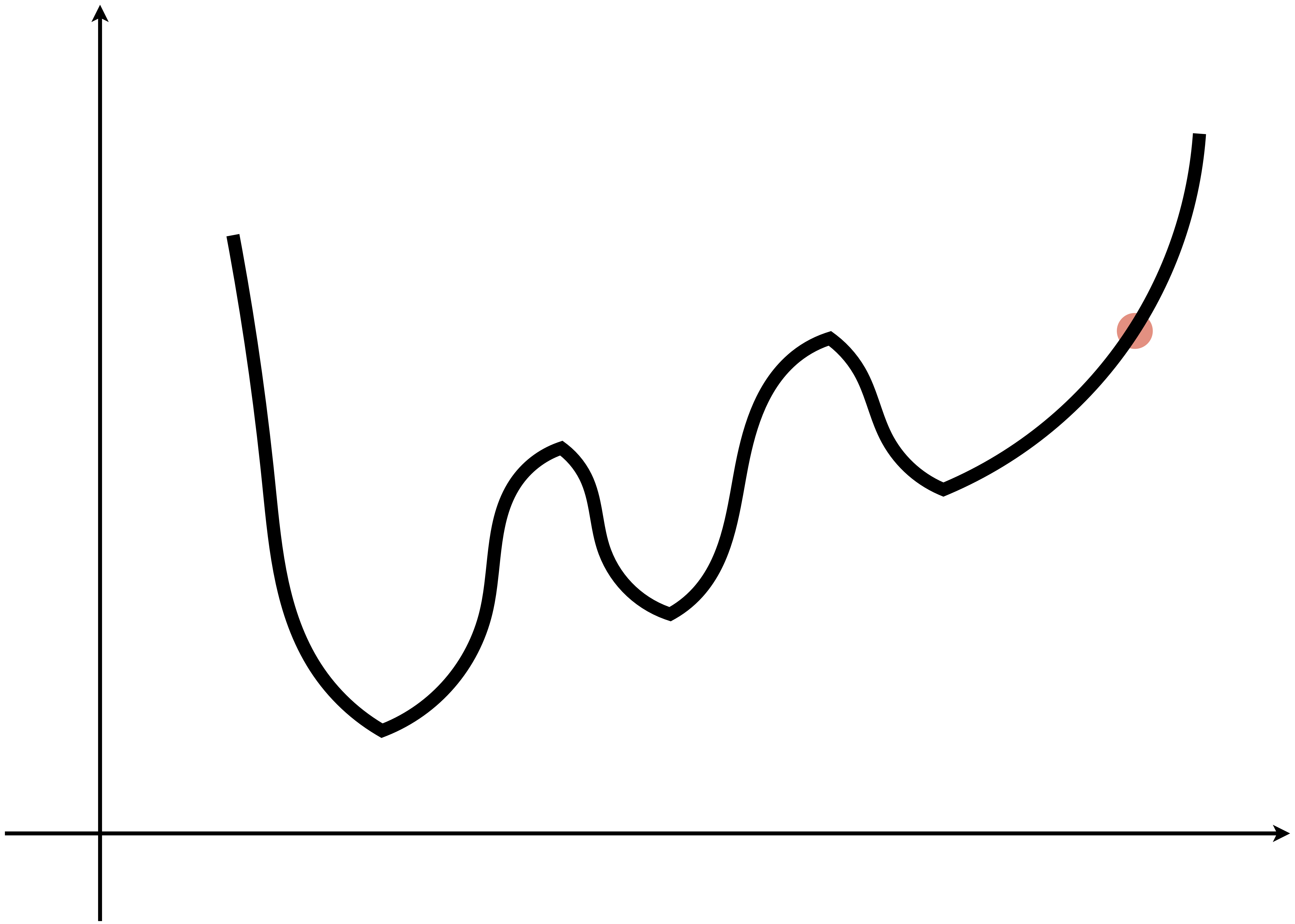
Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

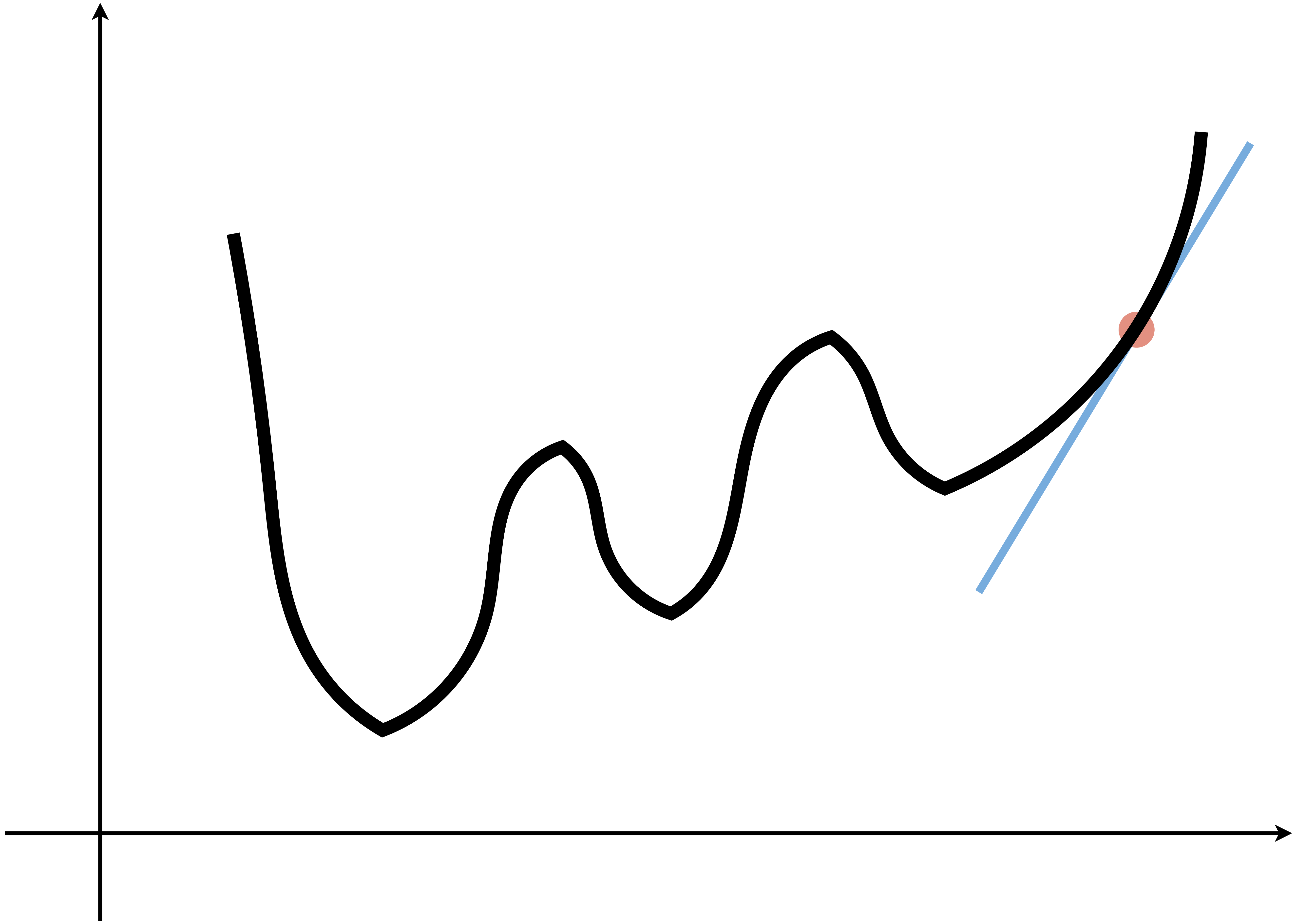
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

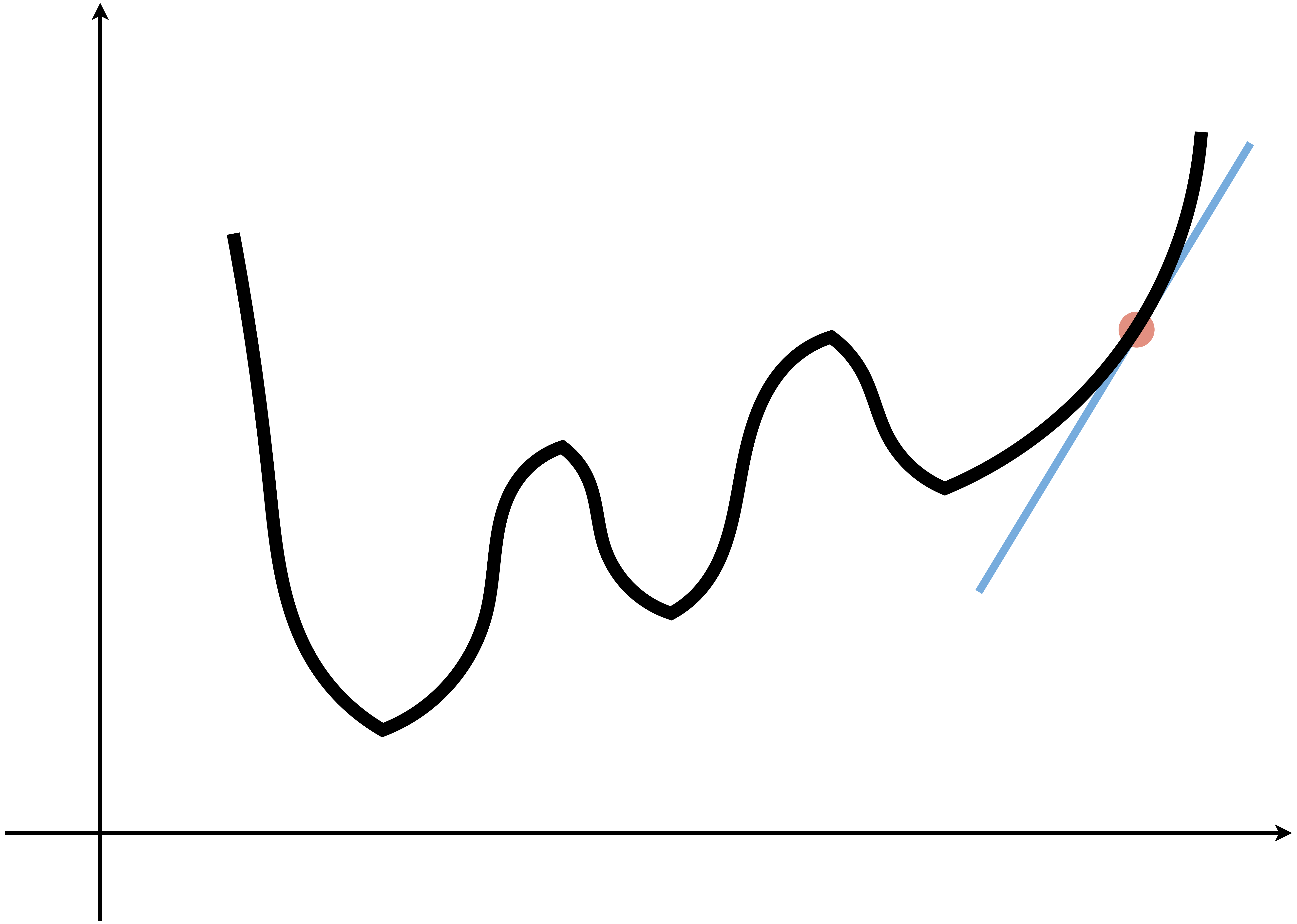
For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

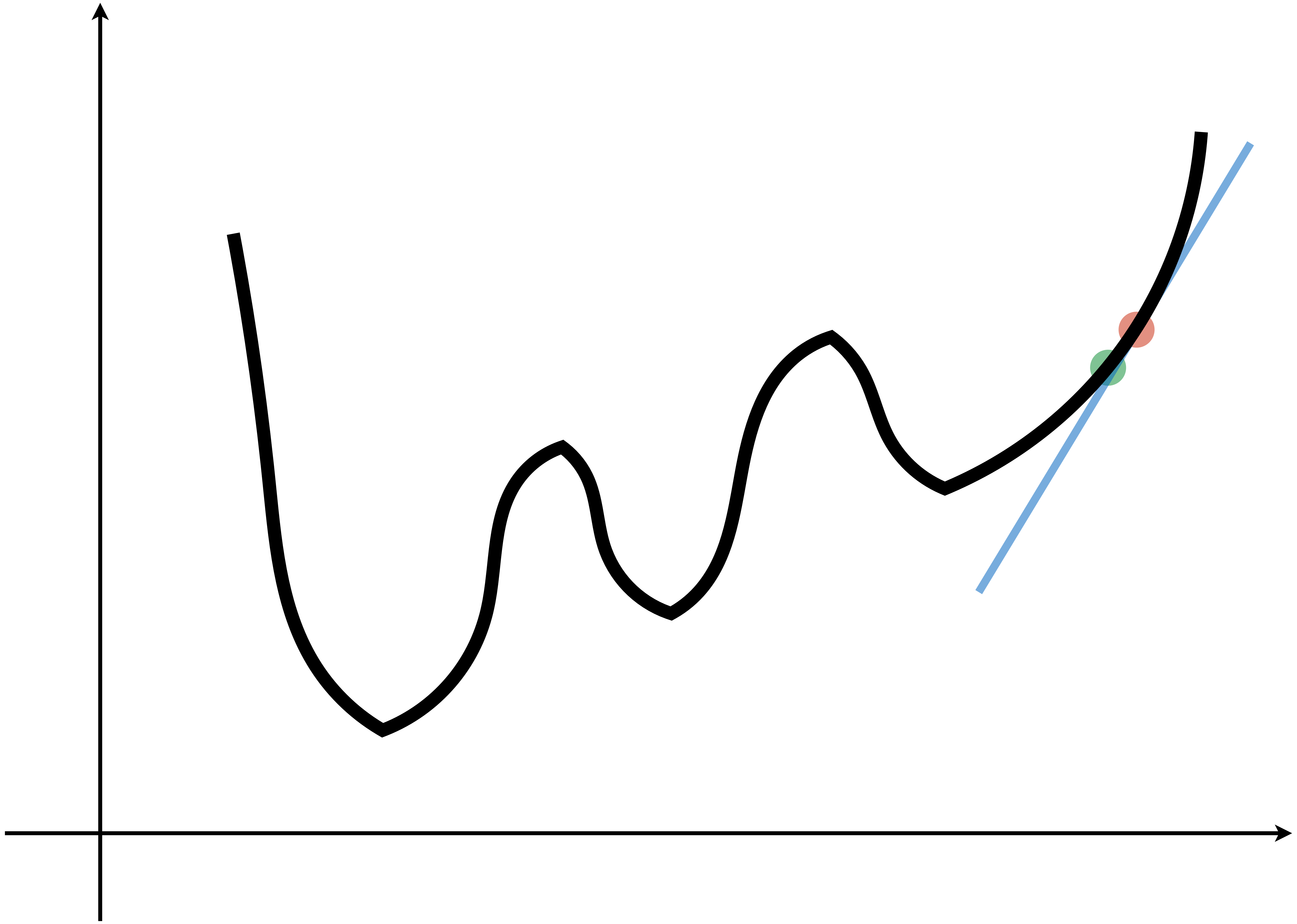
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left.\frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})\big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

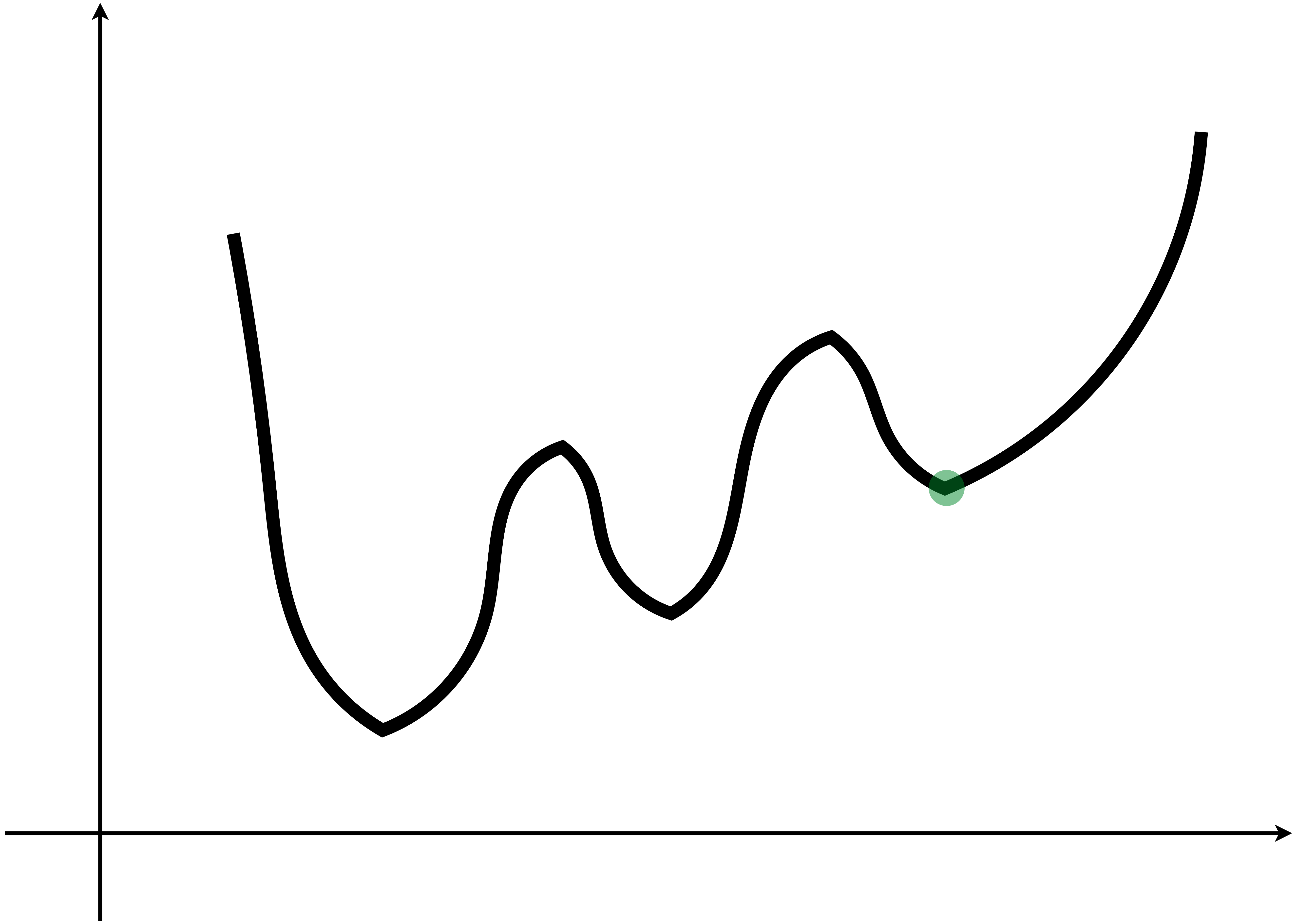
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla\, \mathcal{L}(\mathbf{W}, \mathbf{b})|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

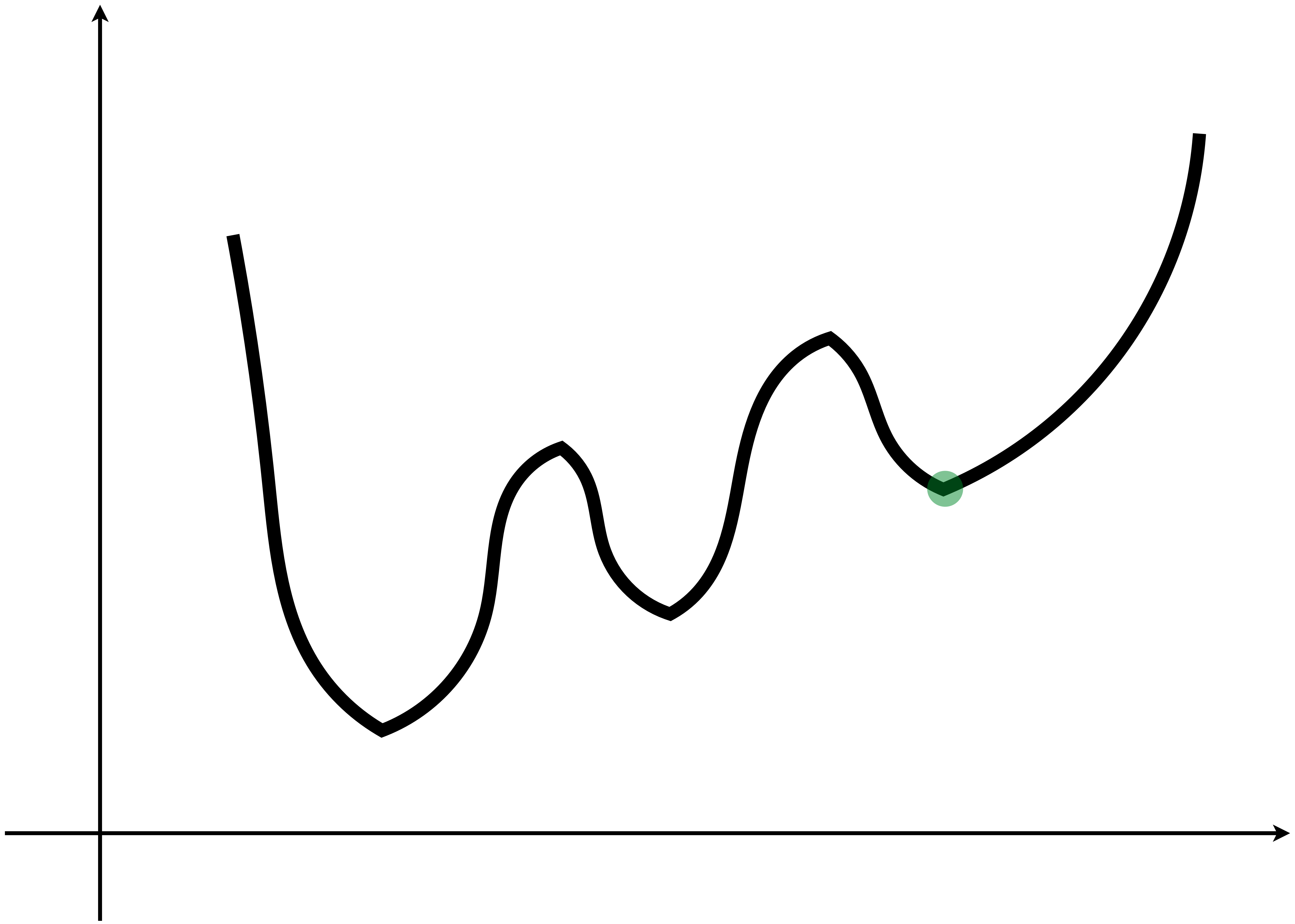
3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \left. \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}} \right|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Gradient Descent



λ - is the learning rate

1. Start from random value of $\mathbf{W}_0, \mathbf{b}_0$

For $k = 0$ to max number of iterations

2. Compute gradient of the loss with respect to previous (initial) parameters:

$$\nabla \mathcal{L}(\mathbf{W}, \mathbf{b})\Big|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

3. Re-estimate the parameters

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}}\Bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \lambda \frac{\partial \mathcal{L}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}}\Bigg|_{\mathbf{W}=\mathbf{W}_k, \mathbf{b}=\mathbf{b}_k}$$

*slide adopted from V. Ordonex

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Solution: Compute approximate gradient with mini-batches of much smaller size (as little as 1-example sometimes)

Stochastic Gradient Descent

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{1,i,j}} = \frac{\partial}{\partial \mathbf{W}_{1,i,j}} \sum_{i=1}^{|\mathcal{D}_{train}|} [\mathbf{y}_i - f(\mathbf{x}_i, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2)]^2$$

Problem: For large datasets computing sum is expensive

Solution: Compute approximate gradient with mini-batches of much smaller size (as little as 1-example sometimes)

Problem: How do we compute the actual gradient?

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Numerical Differentiation

$\mathbf{1}_i$ - Vector of all zeros, except for one 1 in i-th location

We can approximate the gradient numerically, using:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x})}{h}$$

Even better, we can use central differencing:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{1}_i) - f(\mathbf{x} - h\mathbf{1}_i)}{2h}$$

However, both of theses suffer from rounding errors and are not good enough for learning.

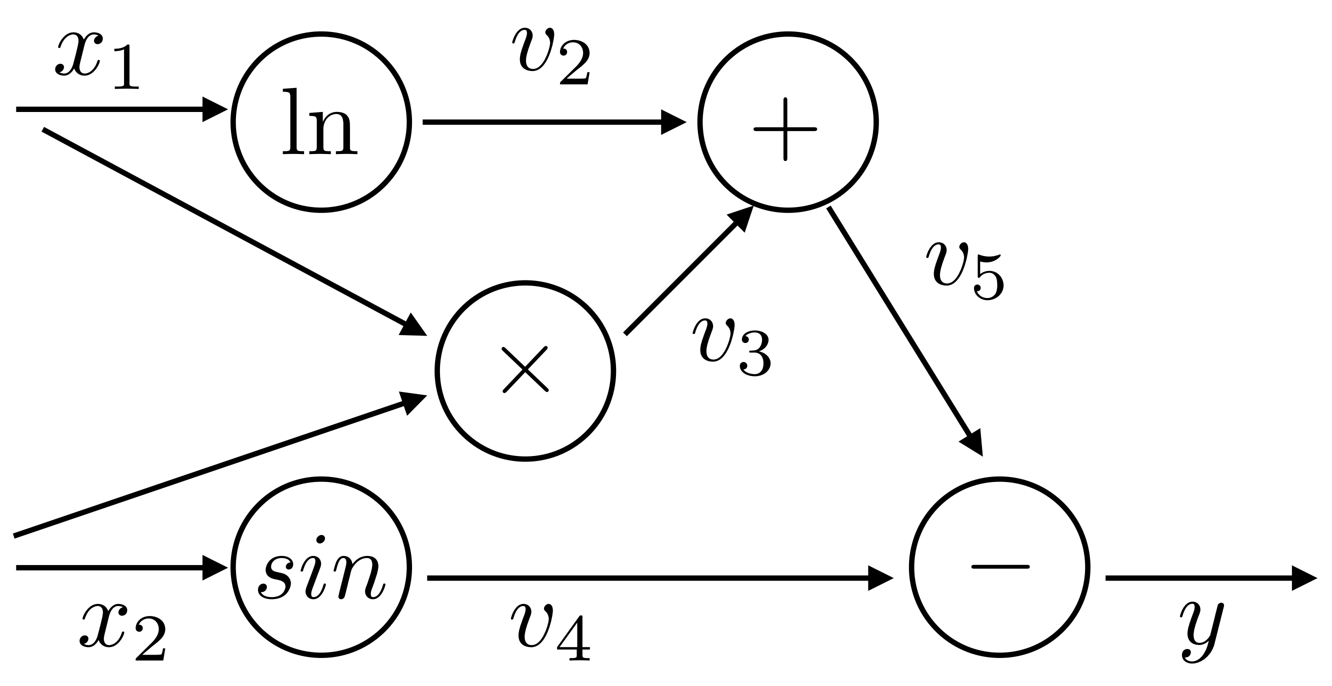
$$h = 0.000001$$

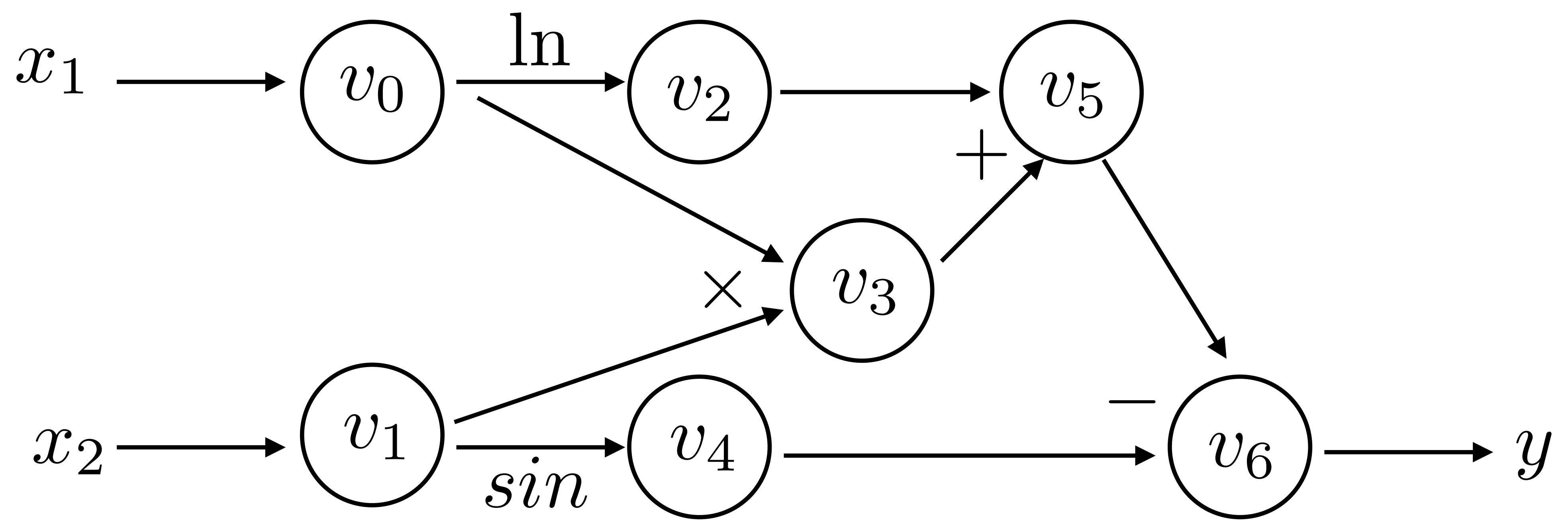
*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Symbolic Differentiation

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Input function is represented as **computational graph** (a symbolic tree)





Implements differentiation rules for composite functions:

Sum Rule

$$\frac{\mathrm{d}\,(f(x)+g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x} + \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Product Rule

$$\frac{\mathrm{d}\,(f(x)\cdot g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x}g(x) + f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Chain Rule

$$\frac{\mathrm{d}(f(g(x)))}{\mathrm{d}x} = \frac{\mathrm{d}f(g(x))}{\mathrm{d}g(x)} \cdot \frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

Problem: For complex functions, expressions can be exponentially large; also difficult to deal with piece-wise functions (creates many symbolic cases)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Intuition: Interleave symbolic differentiation and simplification

Key Idea: apply symbolic differentiation at the elementary operation level, evaluate and keep intermediate results

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Intuition: Interleave symbolic differentiation and simplification

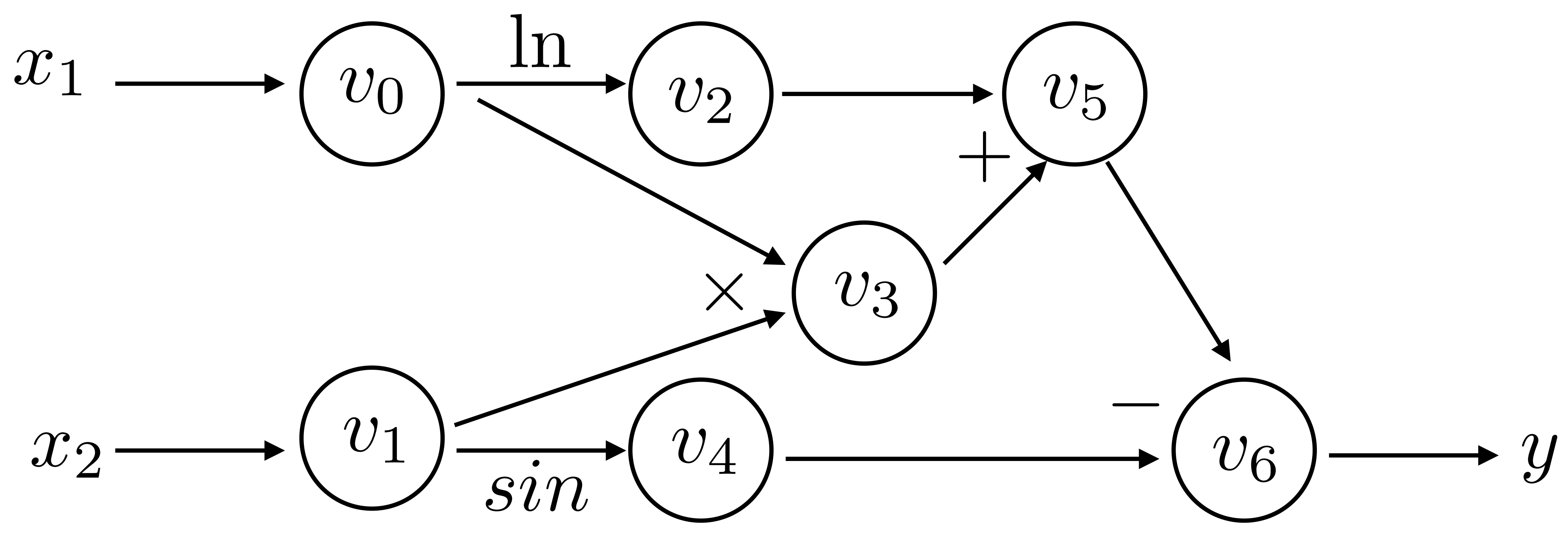
Key Idea: apply symbolic differentiation at the elementary operation level, evaluate and keep intermediate results

Success of **deep learning** owes A LOT to success of AutoDiff algorithms
(also to advances in parallel architectures, and large datasets, …)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



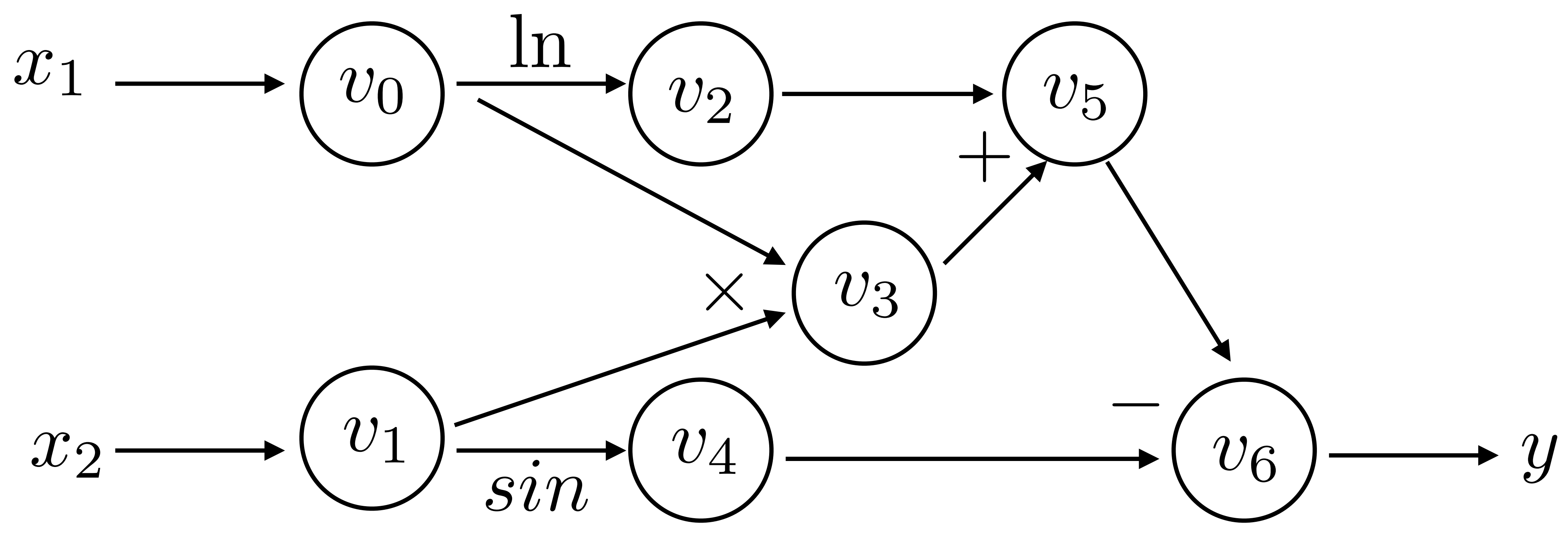
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Computational graph is governed by these equations

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

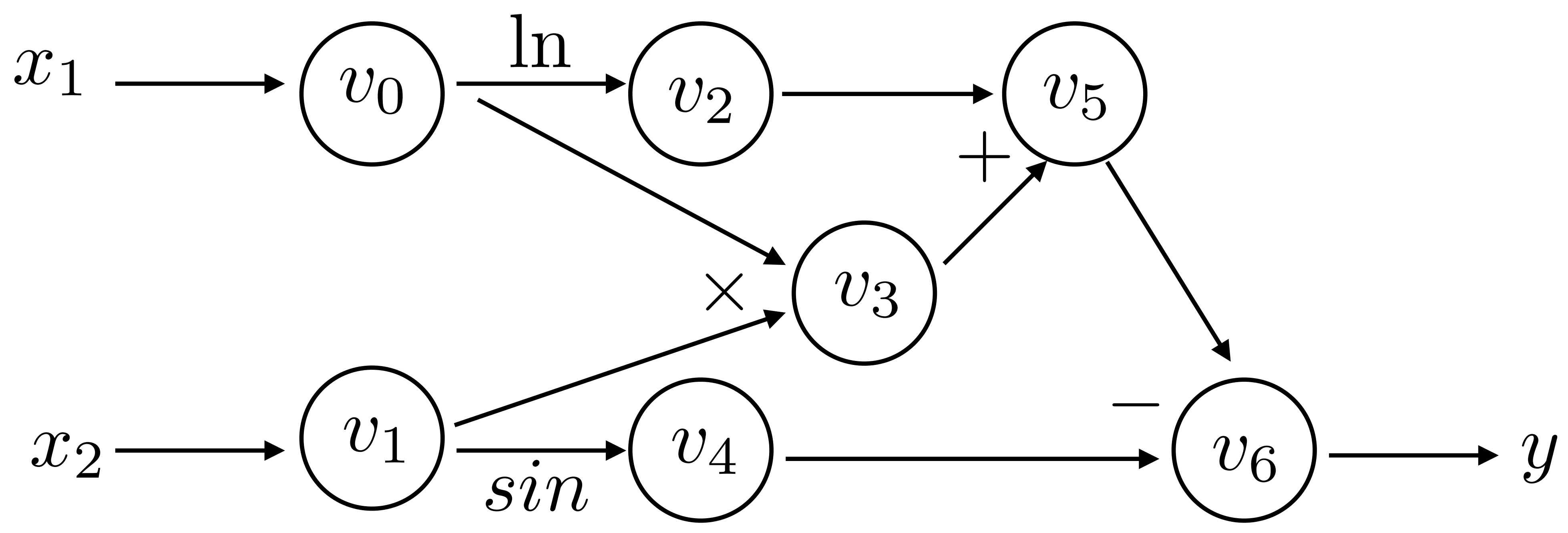
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Computational graph is governed by these equations

$$v_0 = x_1$$
$$v_1 = x_2$$
$$v_2 = \ln(v_0)$$
$$v_3 = v_0 \cdot v_1$$
$$v_4 = sin(v_1)$$
$$v_5 = v_2 + v_3$$
$$v_6 = v_5 - v_4$$
$$y = v_6$$

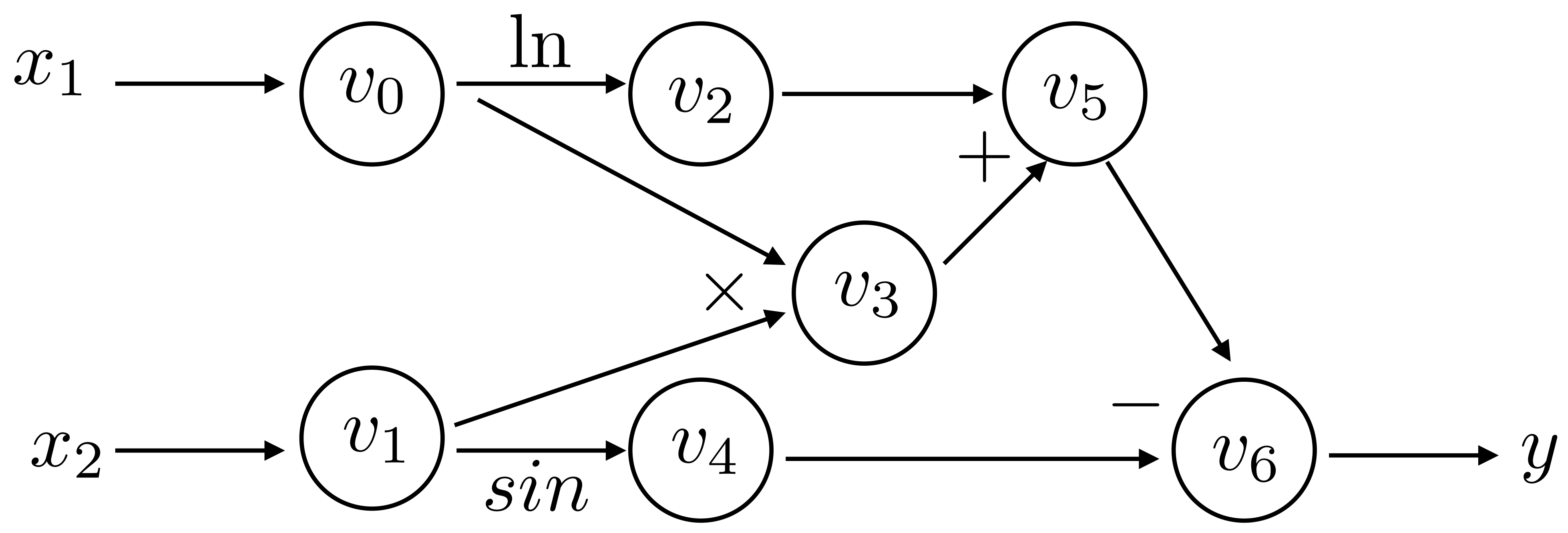
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Lets see how we can **evaluate a function** using computational graph (DNN inferences)

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	
$v_1 = x_2$	
$v_2 = \ln(v_0)$	
$v_3 = v_0 \cdot v_1$	
$v_4 = sin(v_1)$	
$v_5 = v_2 + v_3$	
$v_6 = v_5 - v_4$	
$y = v_6$	

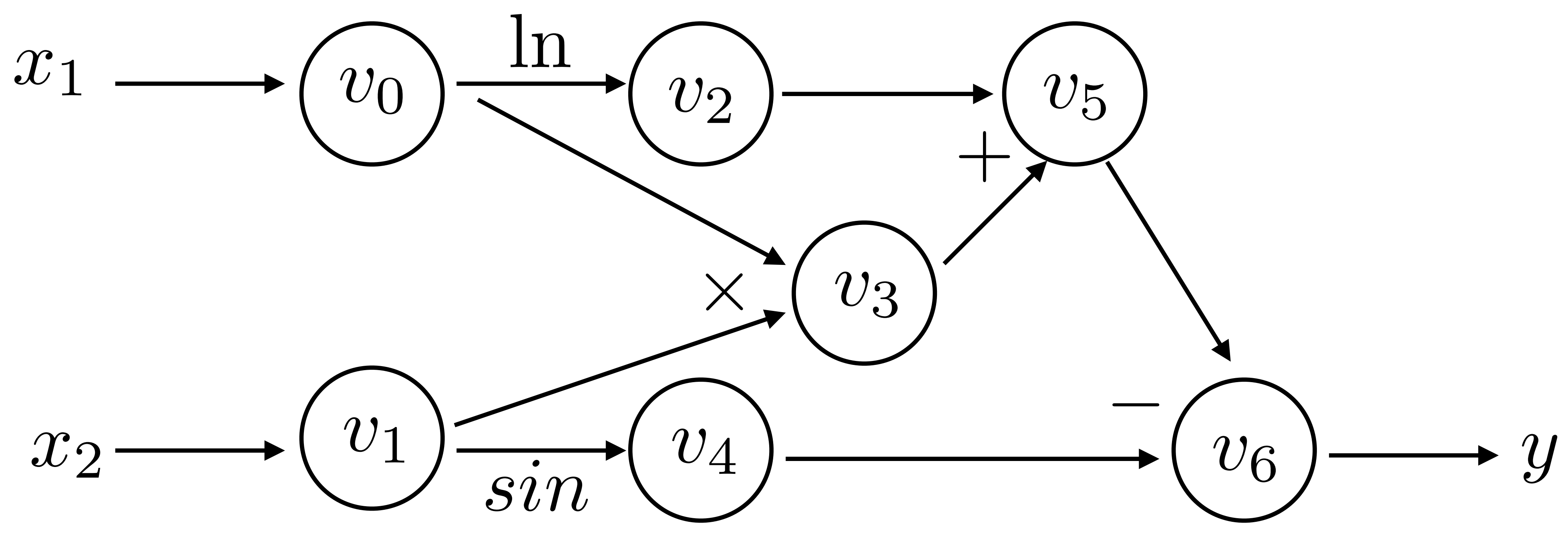
Each **node** is an input, intermediate, or output variable

Computational graph (a DAG) with variable ordering from topological sort.

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

Lets see how we can **evaluate a function** using computational graph (DNN inferences)



Each **node** is an input, intermediate, or output variable

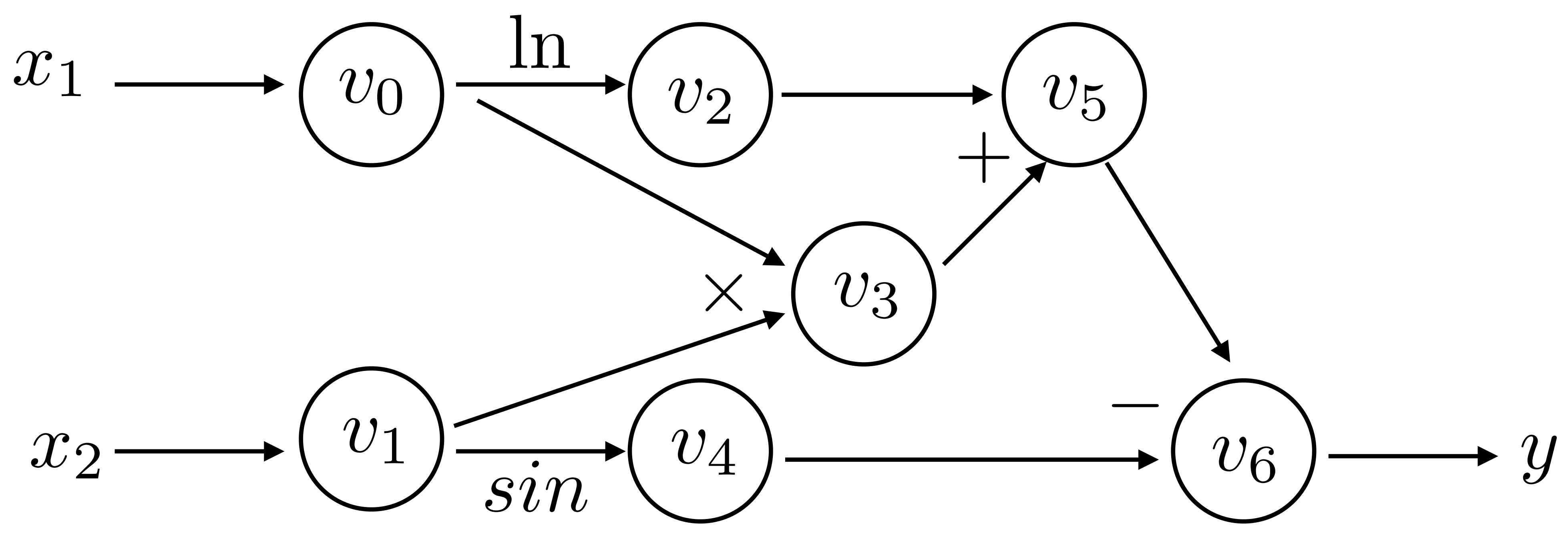
Computational graph (a DAG) with variable ordering from topological sort.

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Automatic Differentiation (AutoDiff)

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

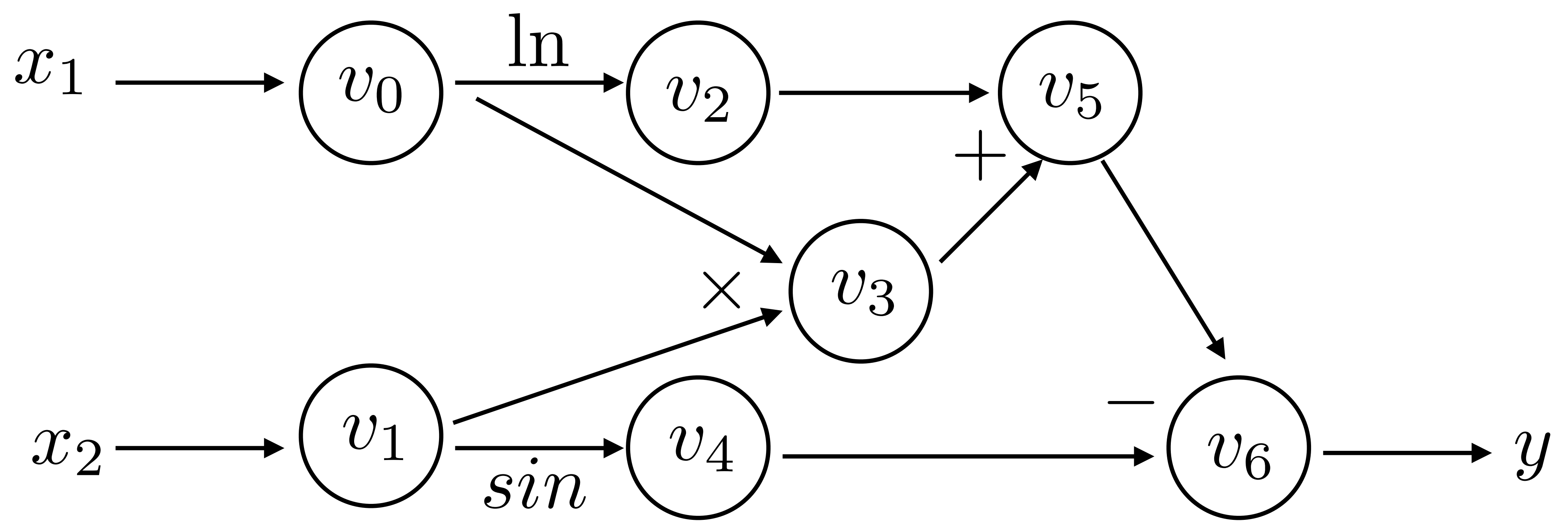


Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

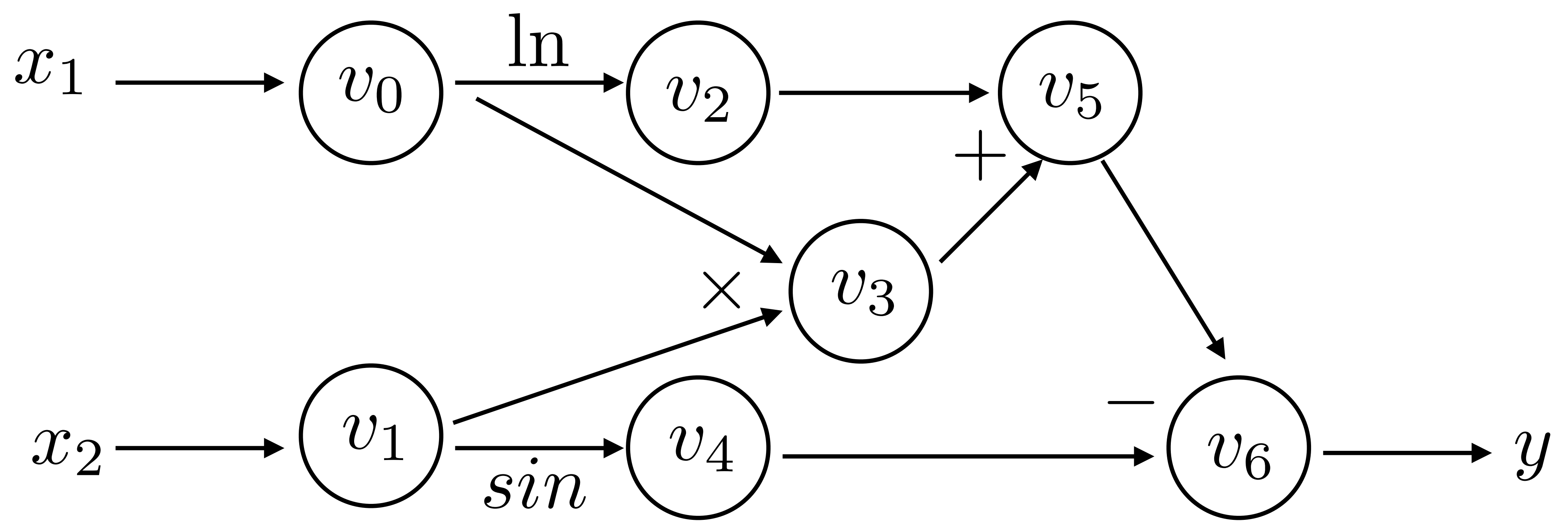
Lets see how we can **evaluate a derivative** using computational graph (DNN learning)

$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)}$$

We will do this with **forward mode** first, by introducing a derivative of each variable node with respect to the input variable.

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Evaluation Trace:

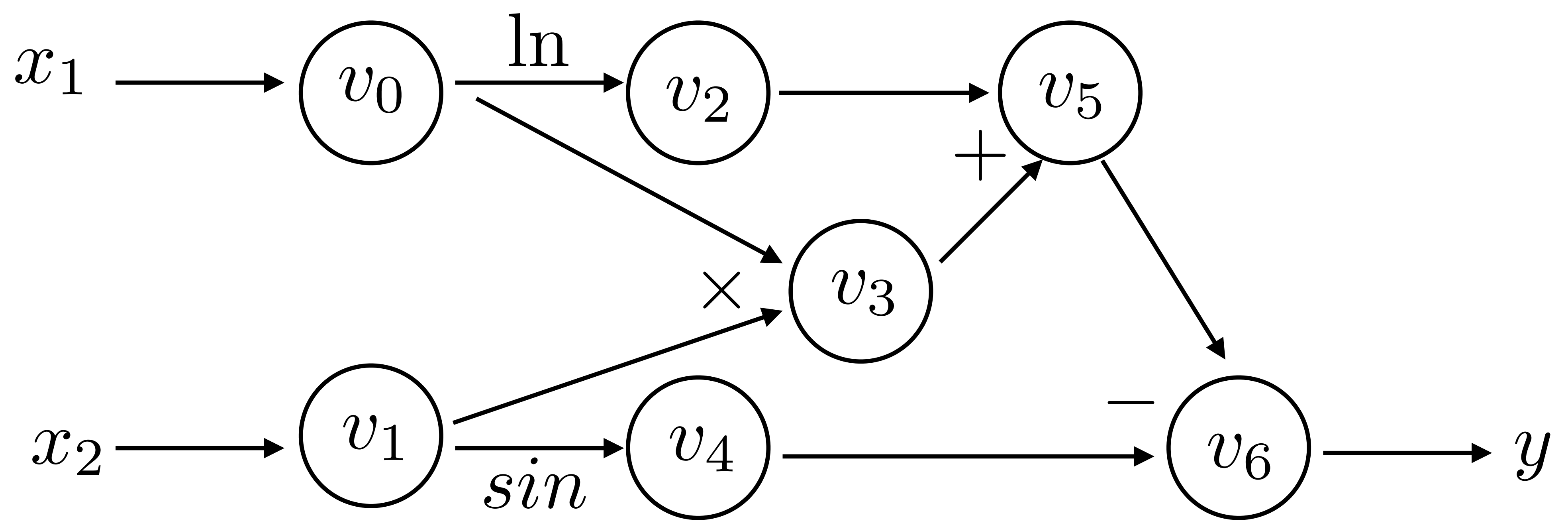
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\|_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



We now have:

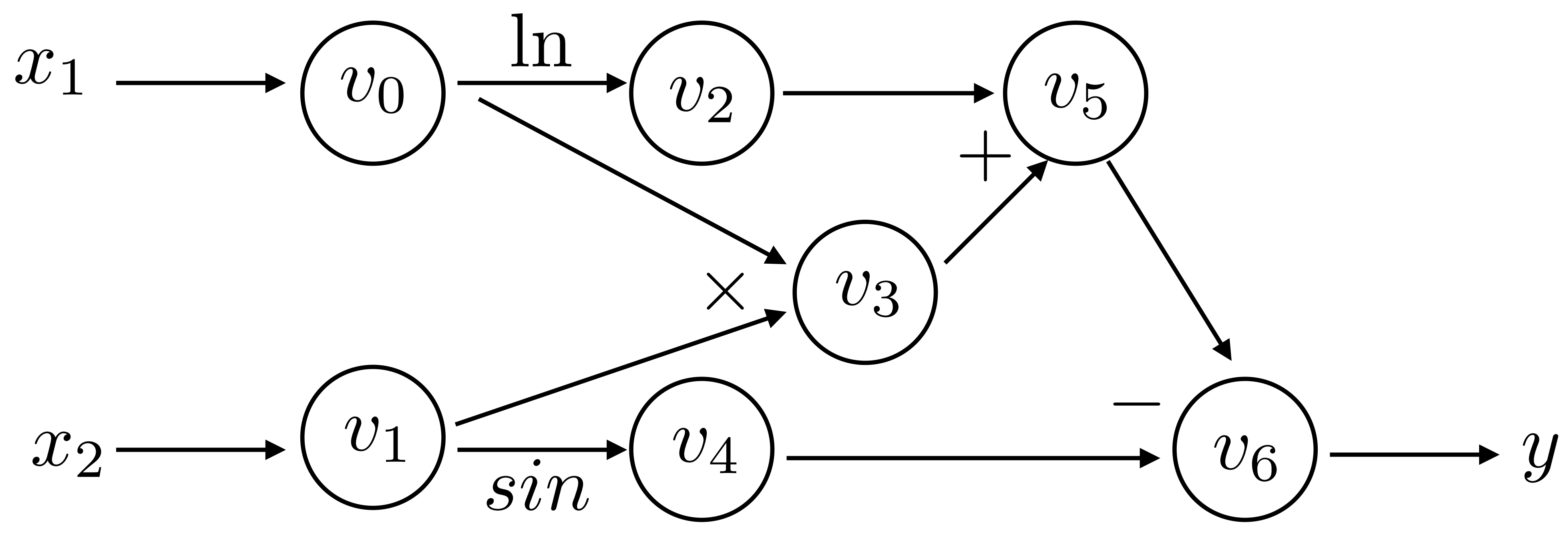
$$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right|_{(x_1=2, x_2=5)} = 5.5$$

Forward Derivative Trace:

	$\left.\frac{\partial f(x_1, x_2)}{\partial x_1}\right\|_{(x_1=2, x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0} \frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

AutoDiff - **Forward Mode**

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$



Forward Derivative Trace:

	$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right\|_{(x_1=2,x_2=5)}$
$\frac{\partial v_0}{\partial x_1}$	1
$\frac{\partial v_1}{\partial x_1}$	0
$\frac{\partial v_2}{\partial x_1} = \frac{1}{v_0}\frac{\partial v_0}{\partial x_1}$	1/2 * 1 = 0.5
$\frac{\partial v_3}{\partial x_1} = \frac{\partial v_0}{\partial x_1} \cdot v_1 + v_0 \cdot \frac{\partial v_1}{\partial x_1}$	1*5 + 2*0 = 5
$\frac{\partial v_4}{\partial x_1} = \frac{\partial v_1}{\partial x_1} cos(v_1)$	0 * cos(5) = 0
$\frac{\partial v_5}{\partial x_1} = \frac{\partial v_2}{\partial x_1} + \frac{\partial v_3}{\partial x_1}$	0.5 + 5 = 5.5
$\frac{\partial v_6}{\partial x_1} = \frac{\partial v_5}{\partial x_1} - \frac{\partial v_4}{\partial x_1}$	5.5 - 0 = 5.5
$\frac{\partial y}{\partial x_1} = \frac{\partial v_6}{\partial x_1}$	5.5

We now have:

$$\left.\frac{\partial f(x_1,x_2)}{\partial x_1}\right|_{(x_1=2,x_2=5)} = 5.5$$

Still need:

$$\left.\frac{\partial f(x_1,x_2)}{\partial x_2}\right|_{(x_1=2,x_2=5)}$$

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

image as an input, plus all the weights and biases of layers = millions of inputs!

and very few outputs (many DNNs have $n = 1$) **Why?**

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Forward Mode**

Forward mode needs m forward passes to get a full Jacobian (all gradients of output with respect to each input), where m is the number of inputs

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$

Problem: DNN typically has large number of inputs:

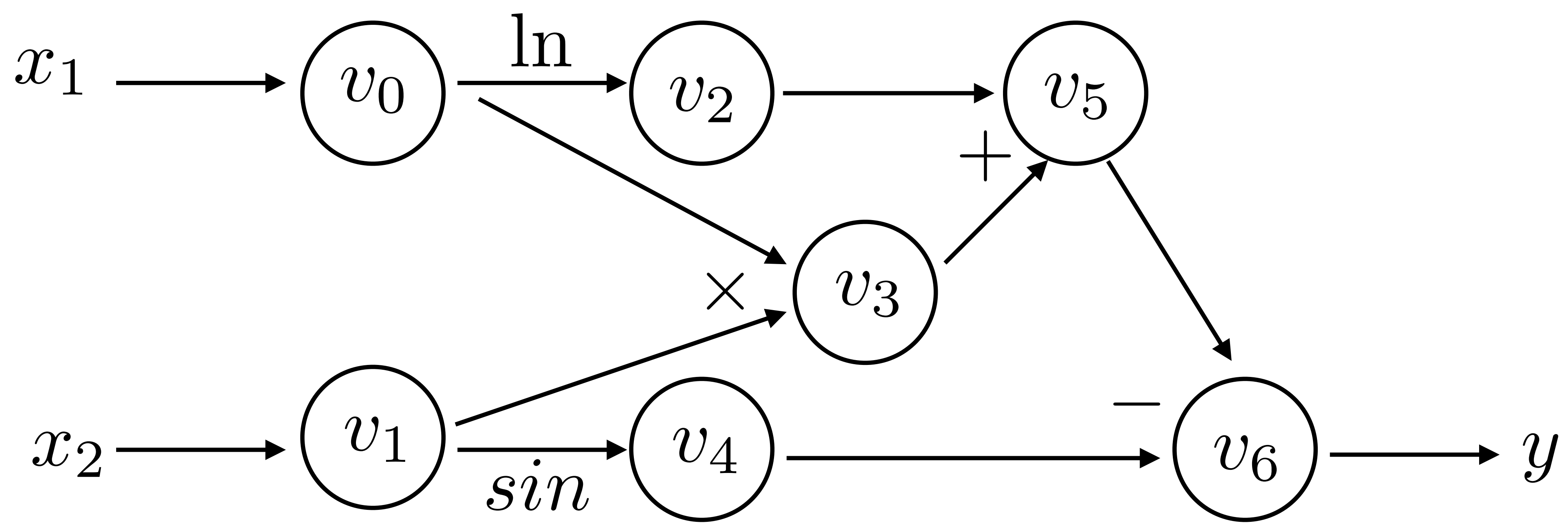
image as an input, plus all the weights and biases of layers = millions of inputs!

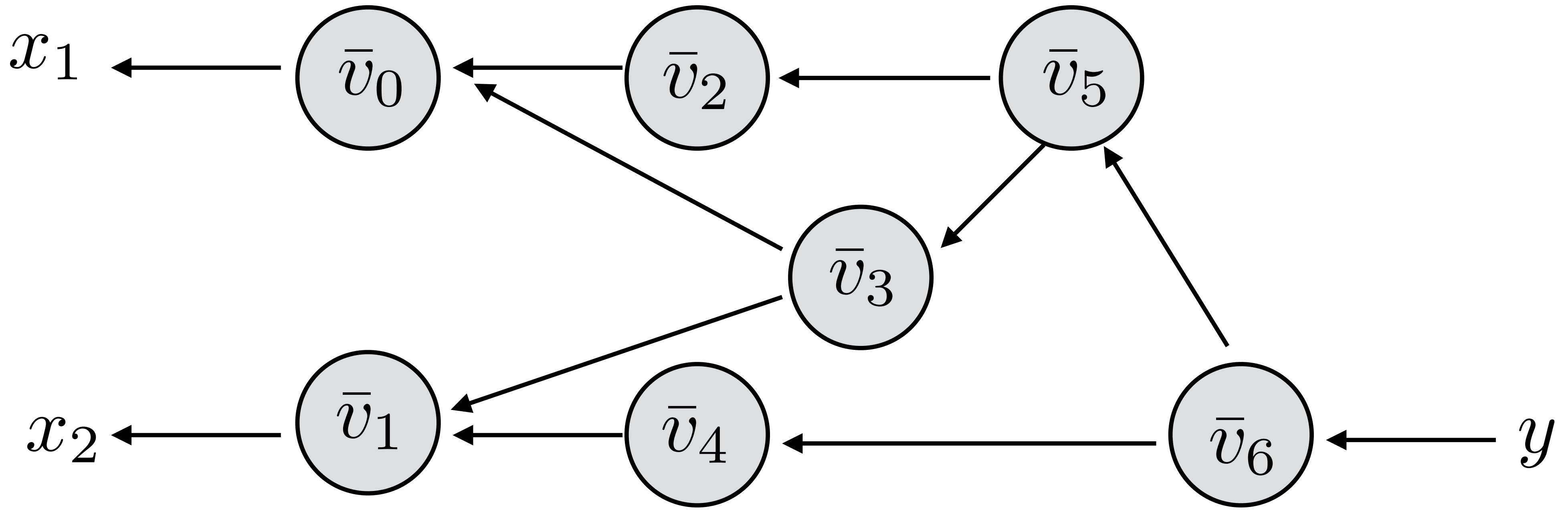
and very few outputs (many DNNs have $n = 1$)

Automatic differentiation in **reverse mode** computes all gradients in n backwards passes (so for most DNNs in a single back pass — **back propagation**)

*slide adopted from T. Chen, H. Shen, A. Krishnamurthy CSE 599G1 lecture at UWashington

AutoDiff - **Reverse Mode**





Forward Evaluation Trace:

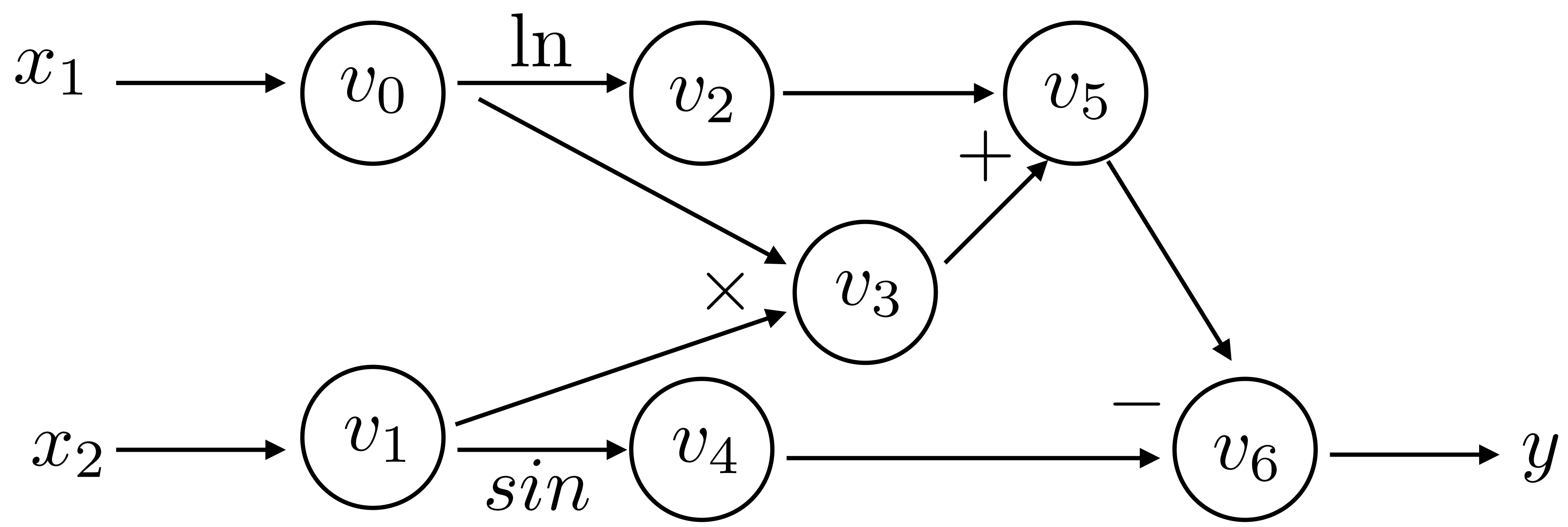
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Traverse the original graph in the *reverse* topological order and for each node in the original graph introduce an **adjoint node**, which computes derivative of the output with respect to the local node (using Chain rule):

$$\bar{v}_i = \frac{\partial y_j}{\partial v_i} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \frac{\partial y_j}{\partial v_k} = \sum_{k \in \mathrm{pa}(i)} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

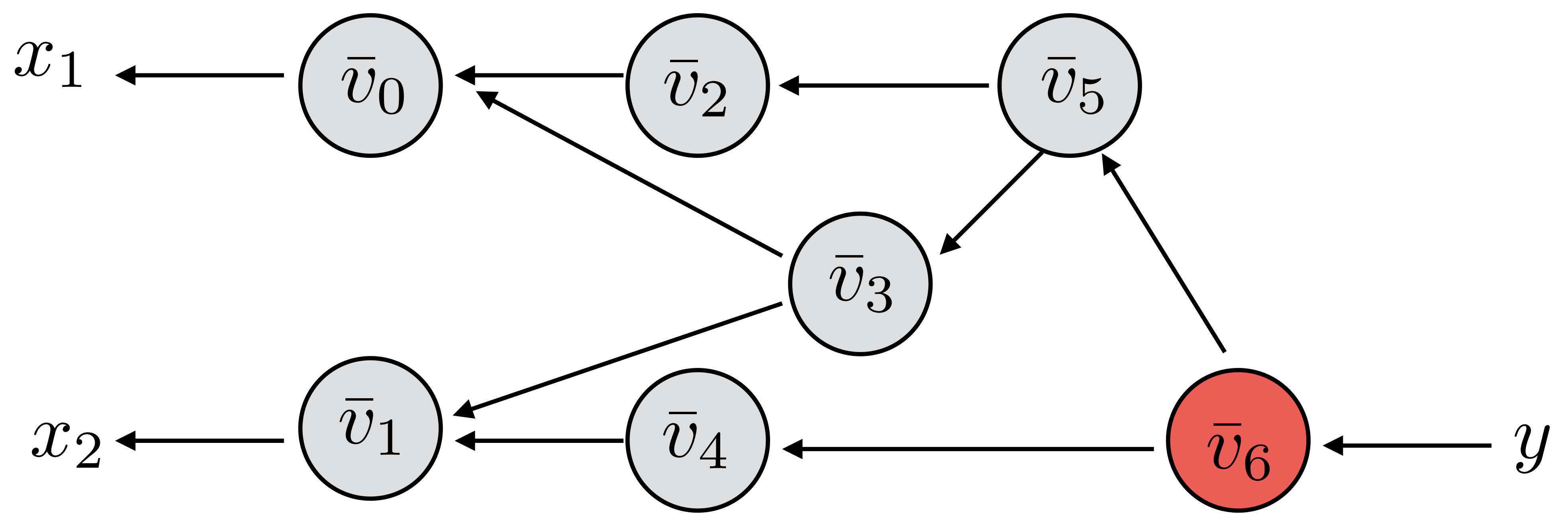
“**local**” derivative

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

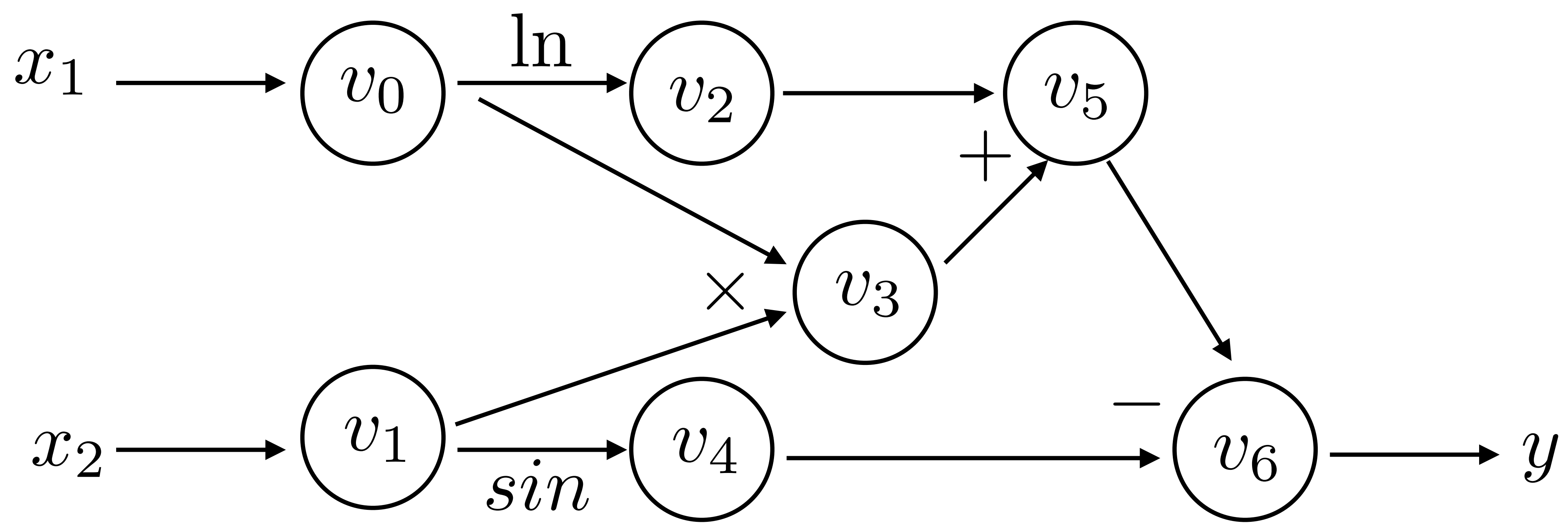
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

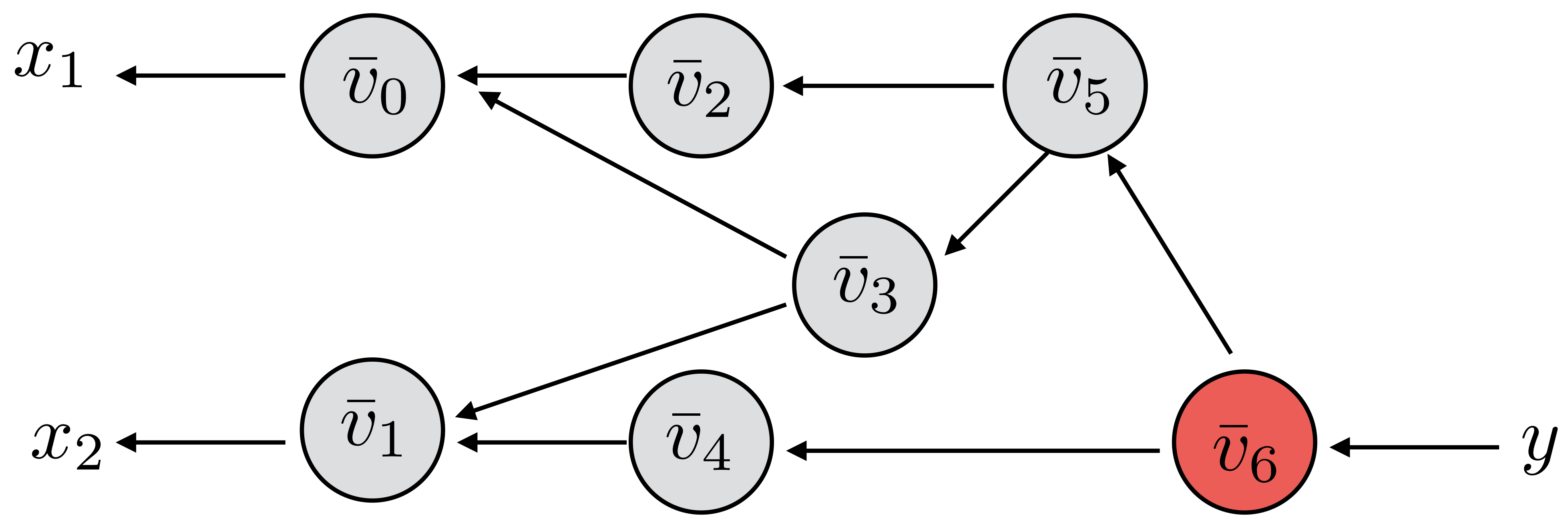


Backwards Derivative Trace:

$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff - **Reverse Mode**





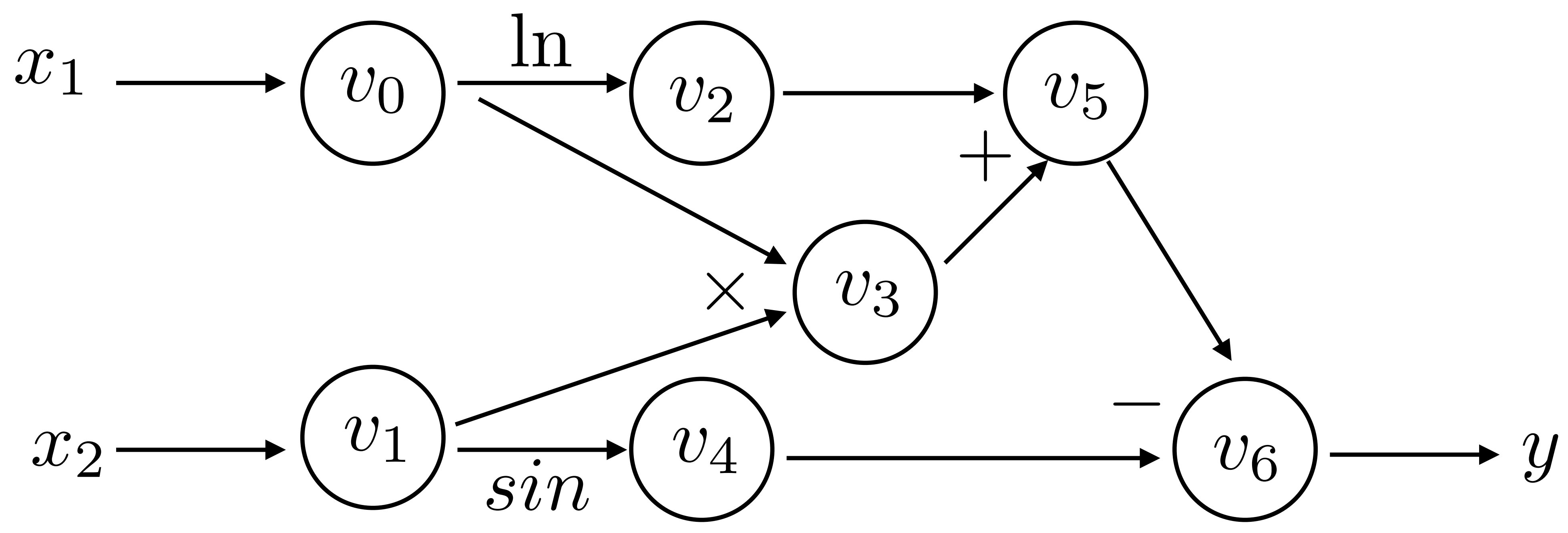
Backwards Derivative Trace:

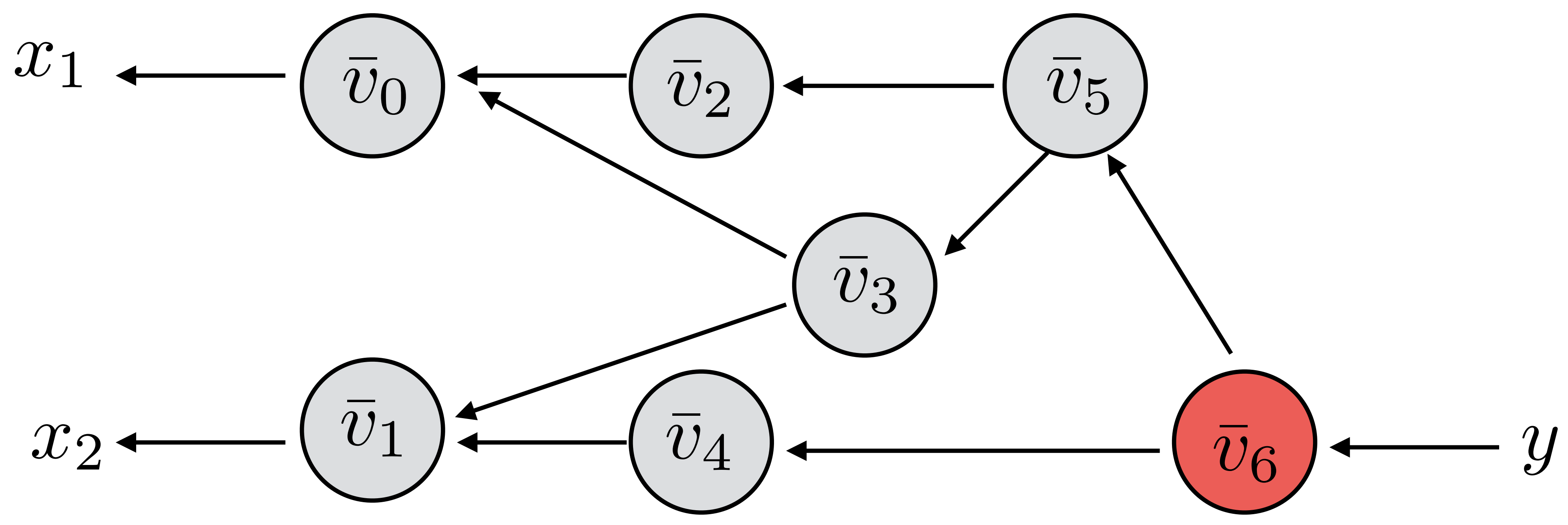
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

$$\bar{v}_6 = \frac{\partial y}{\partial v_6}$$

AutoDiff - **Reverse Mode**





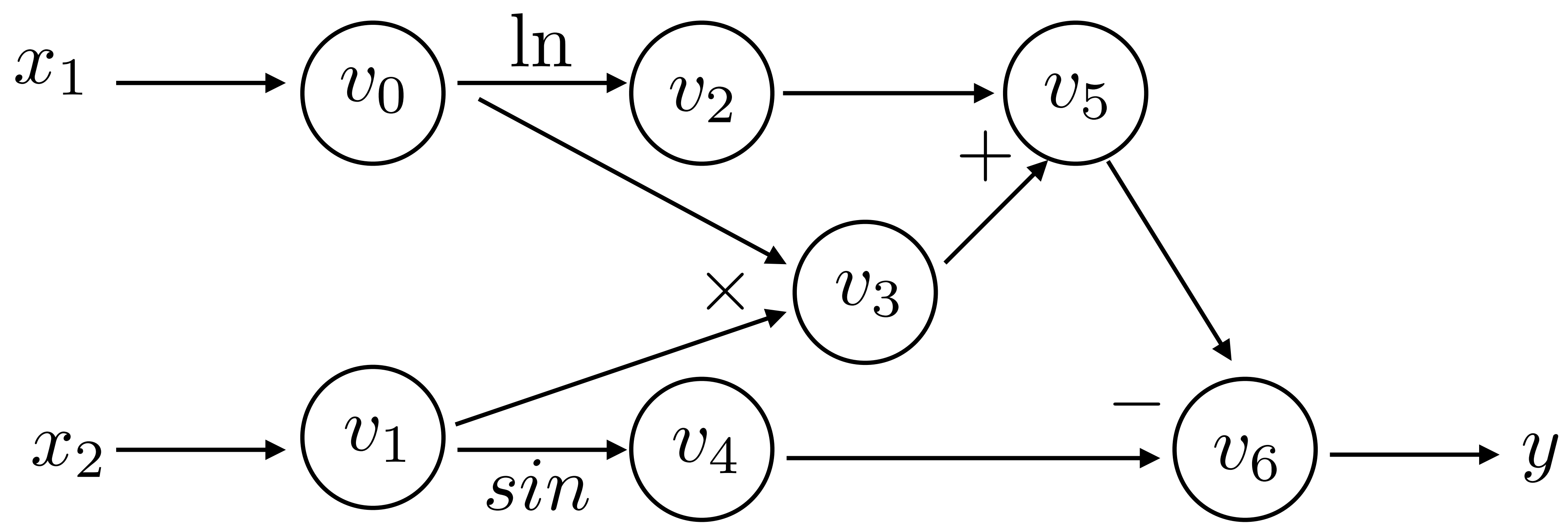
Forward Evaluation Trace:

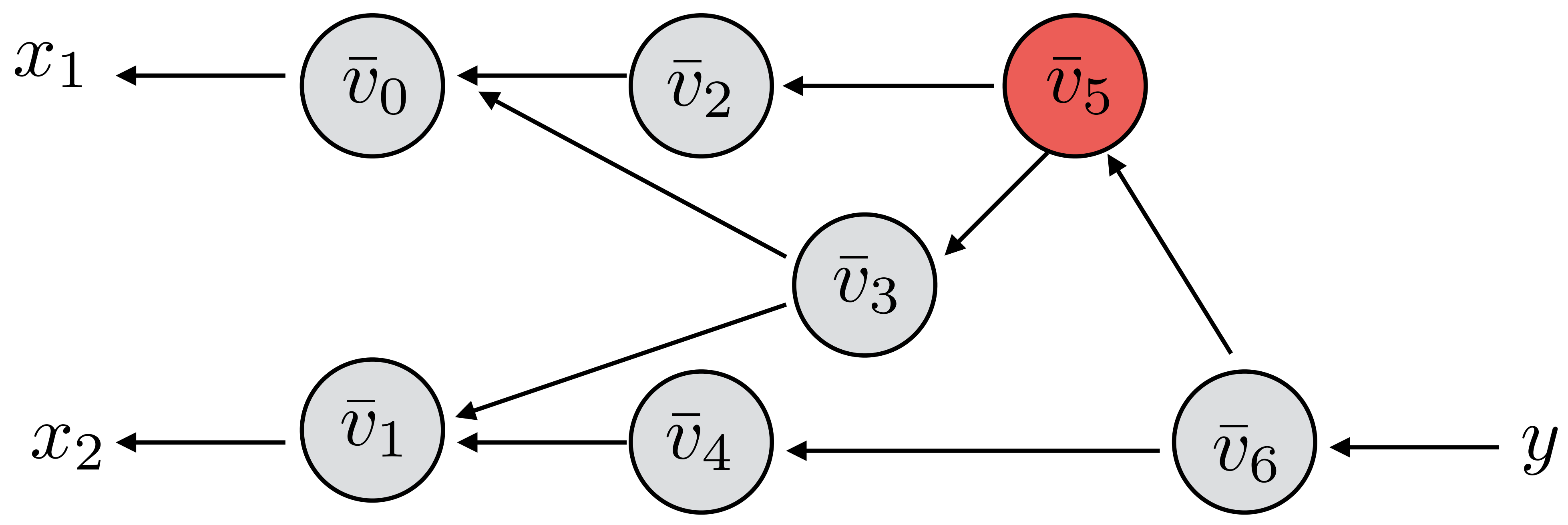
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





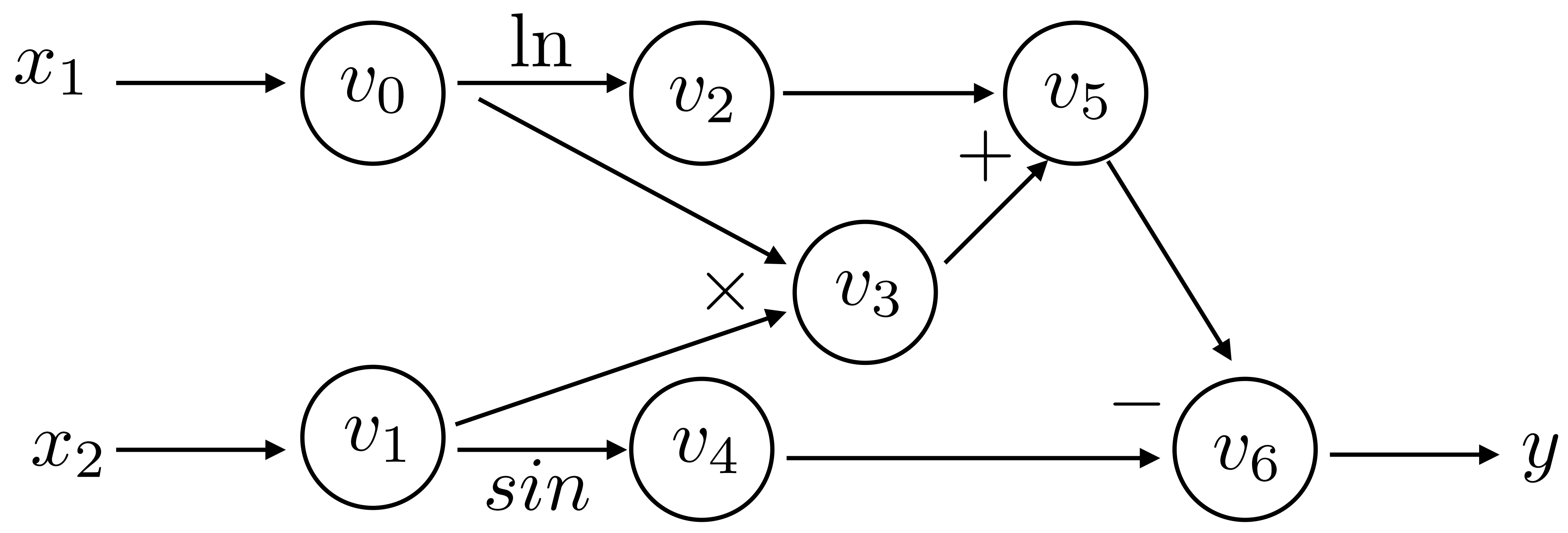
Forward Evaluation Trace:

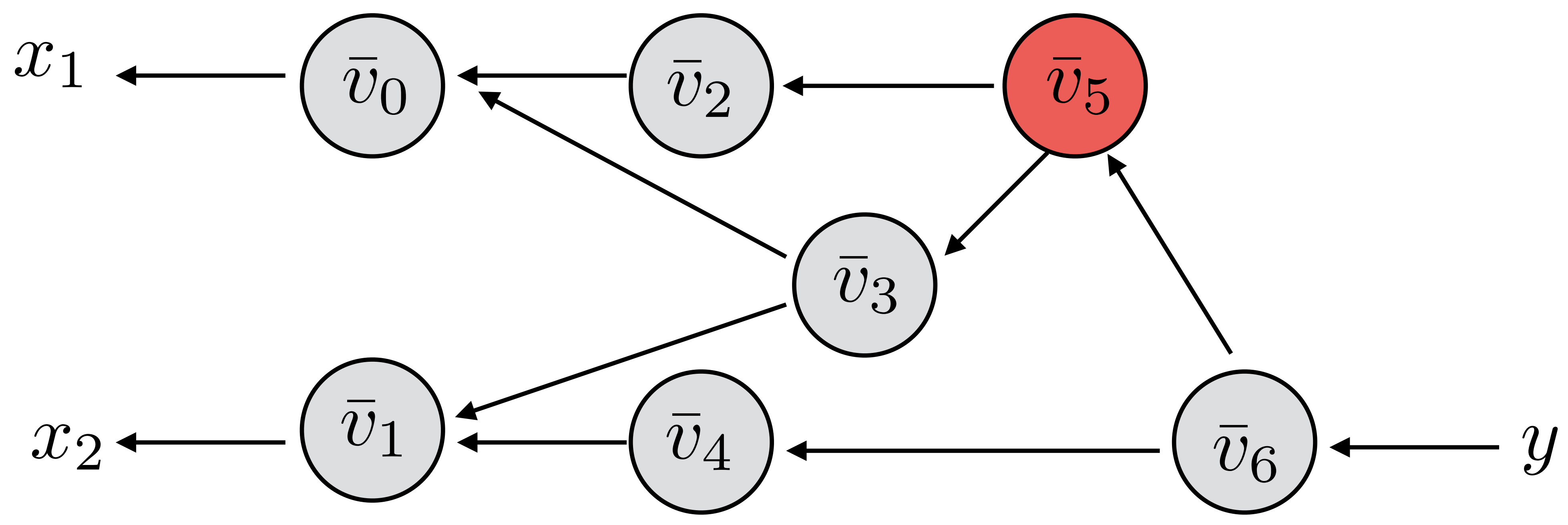
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5}$	
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





Forward Evaluation Trace:

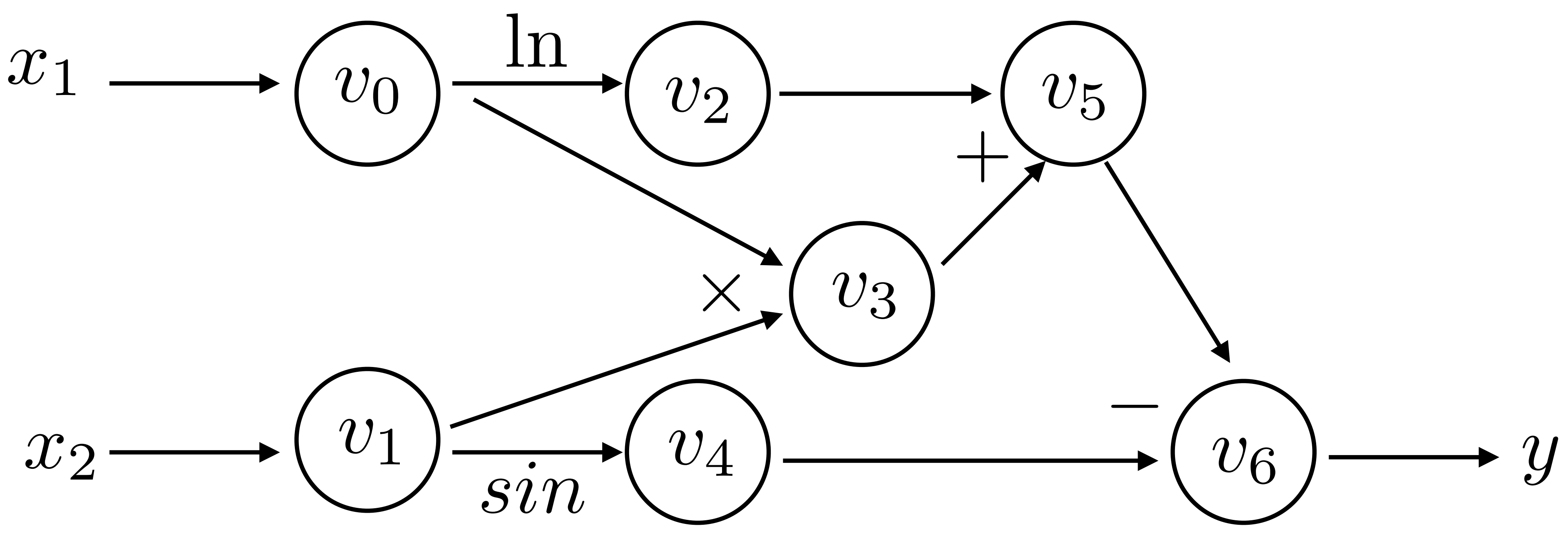
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

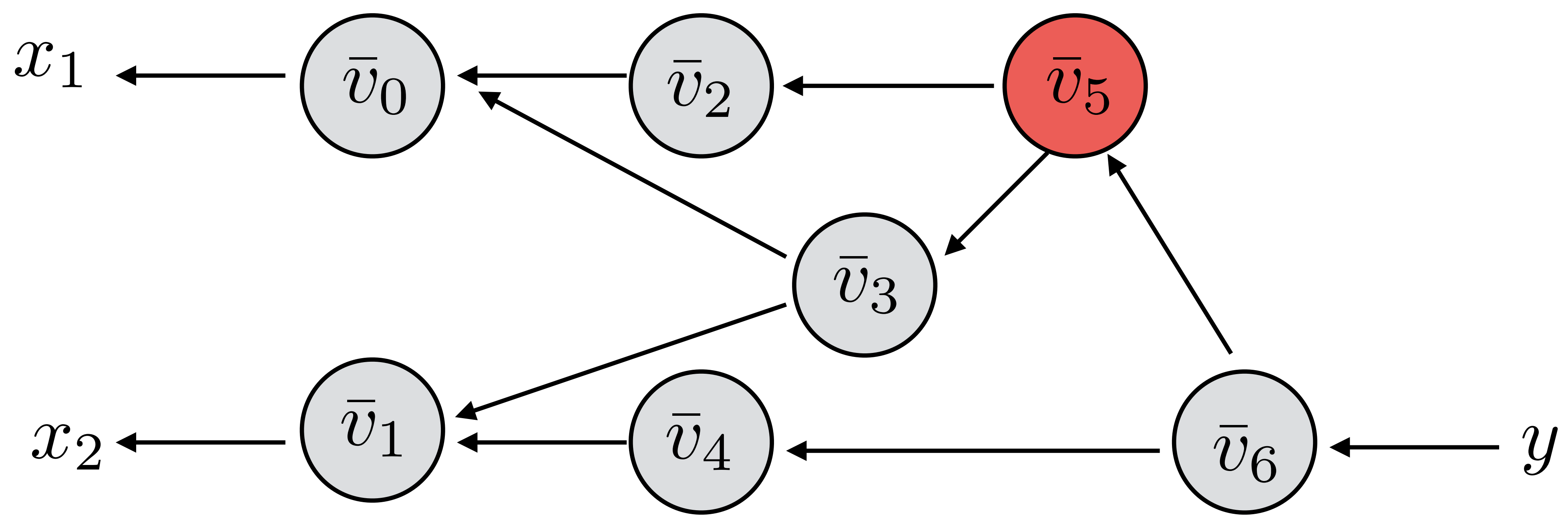
Backwards Derivative Trace:

$$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5}$$

$$\bar{v}_6 = \frac{\partial y}{\partial v_6} \quad 1$$

AutoDiff - **Reverse Mode**





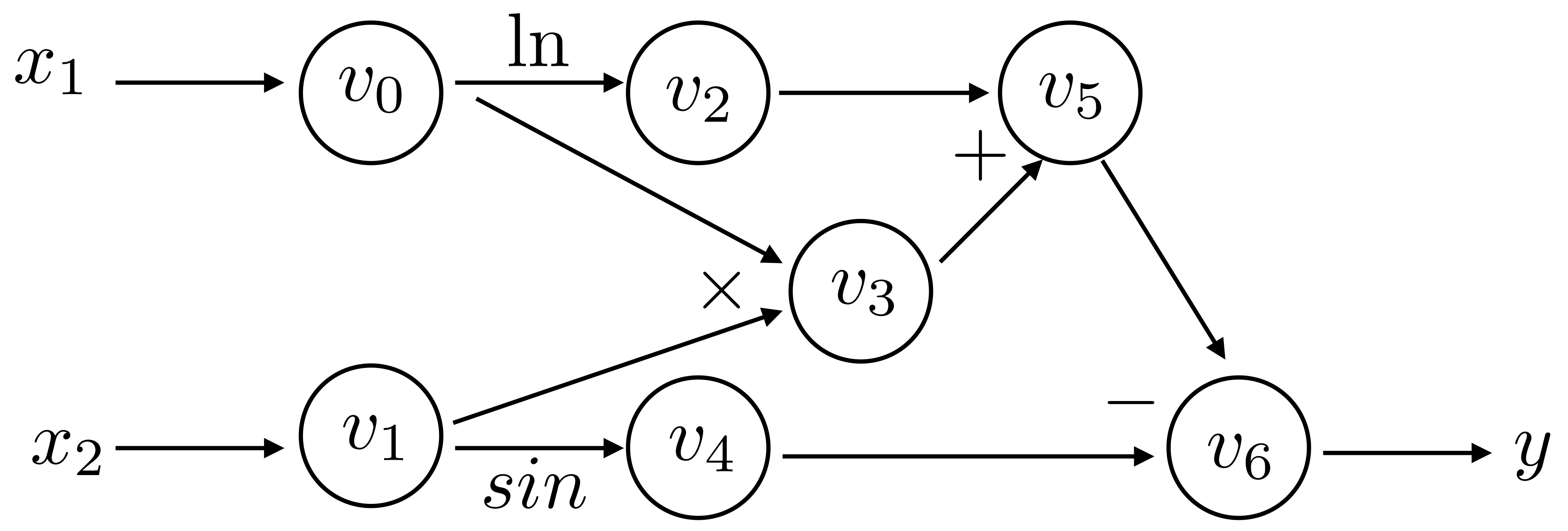
Forward Evaluation Trace:

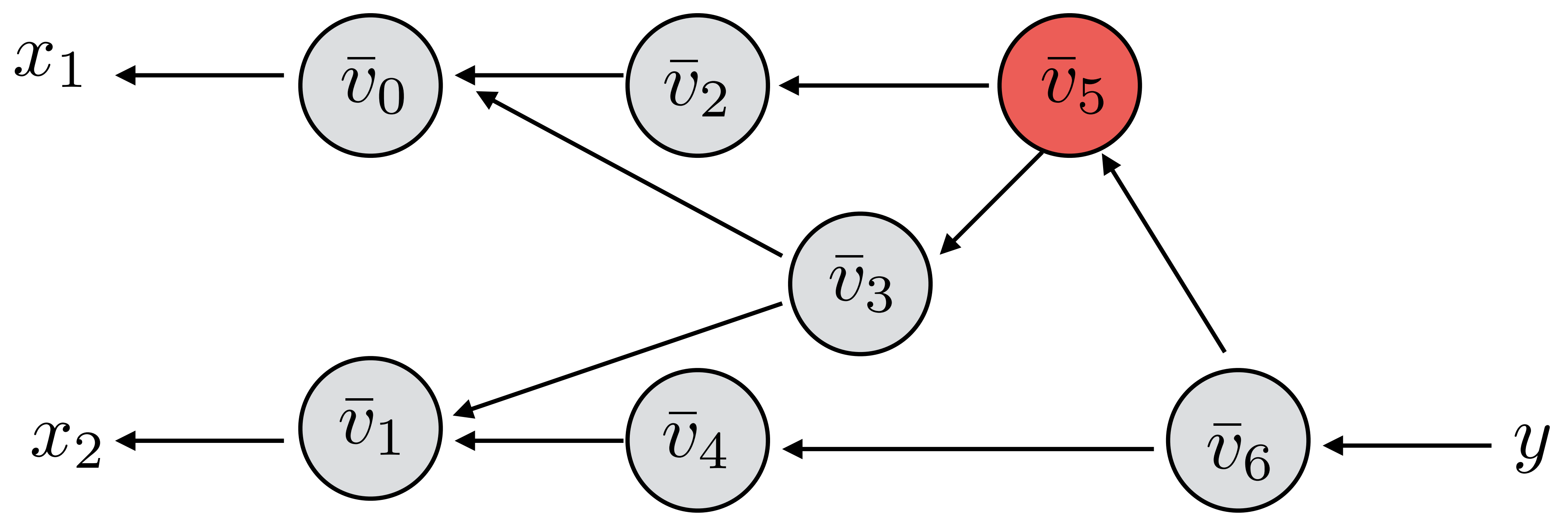
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





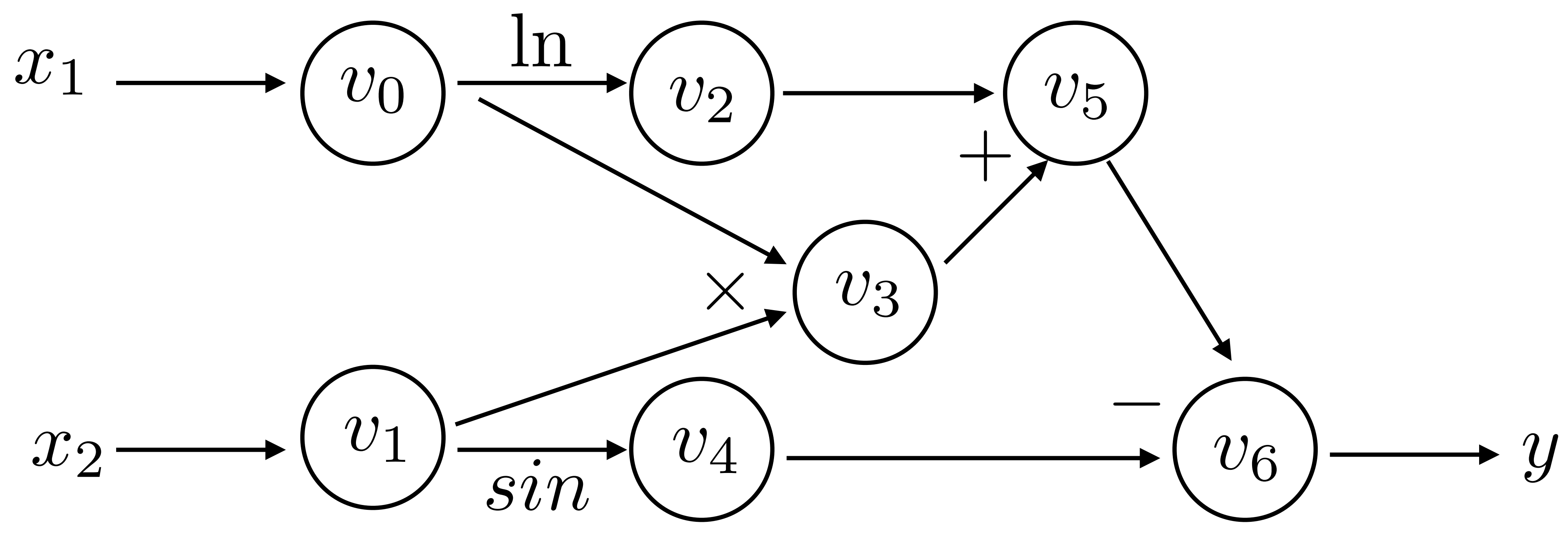
Forward Evaluation Trace:

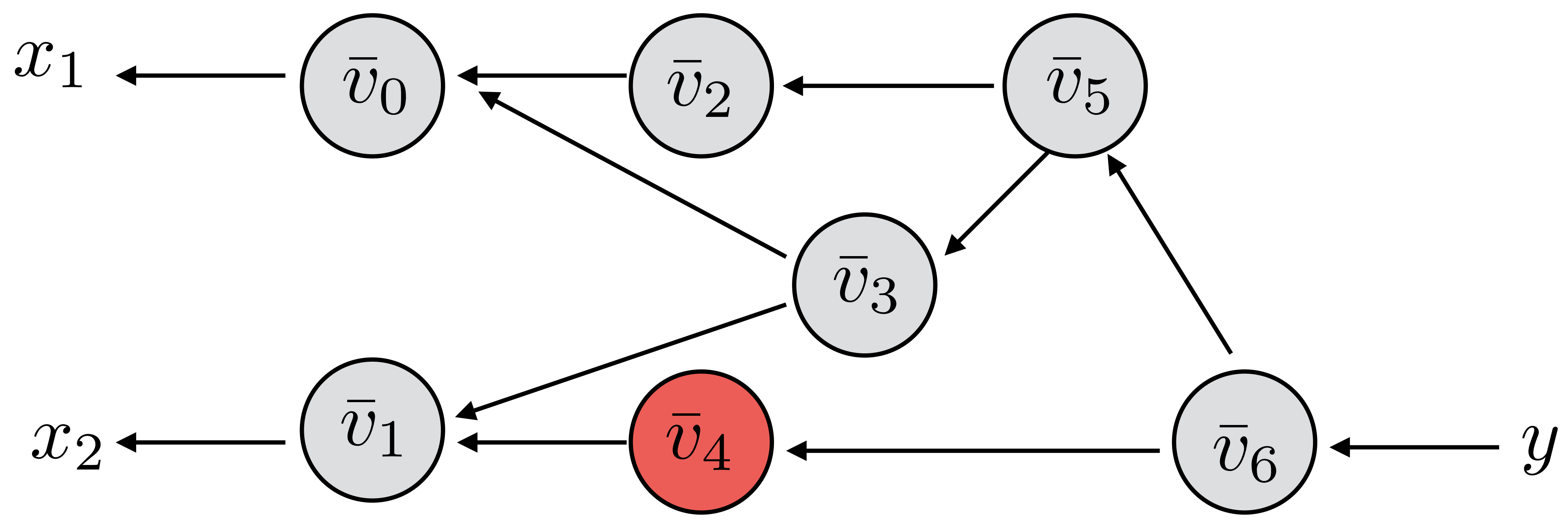
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





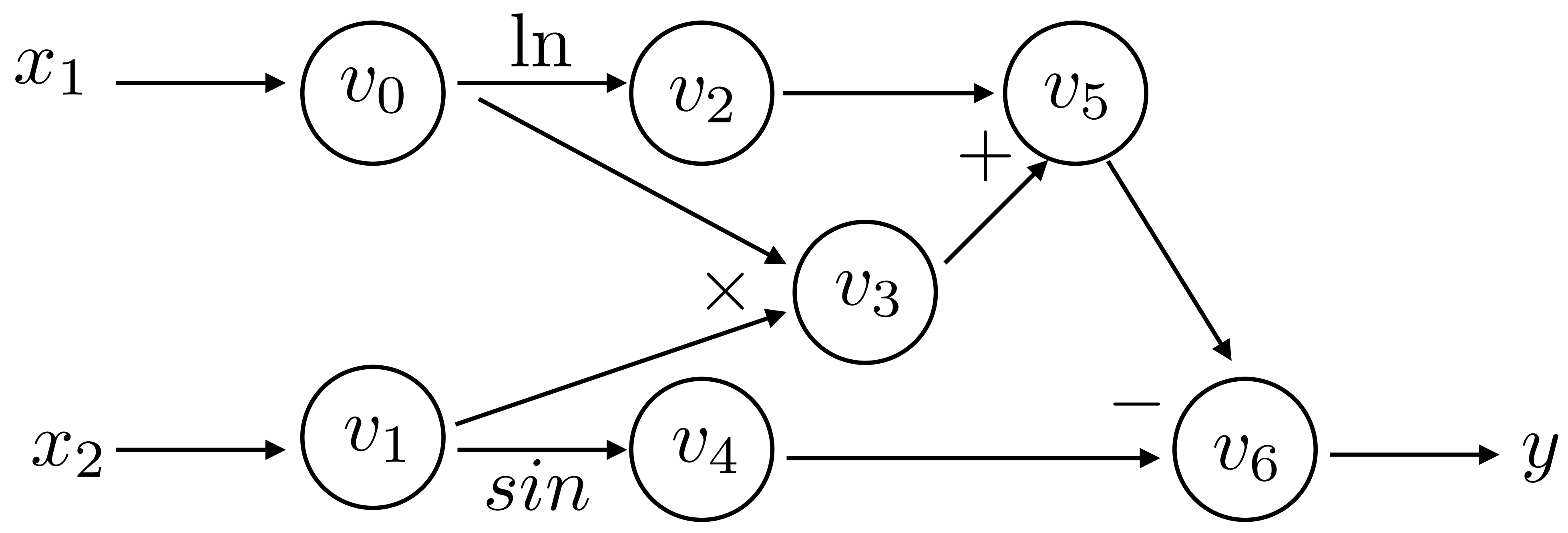
Forward Evaluation Trace:

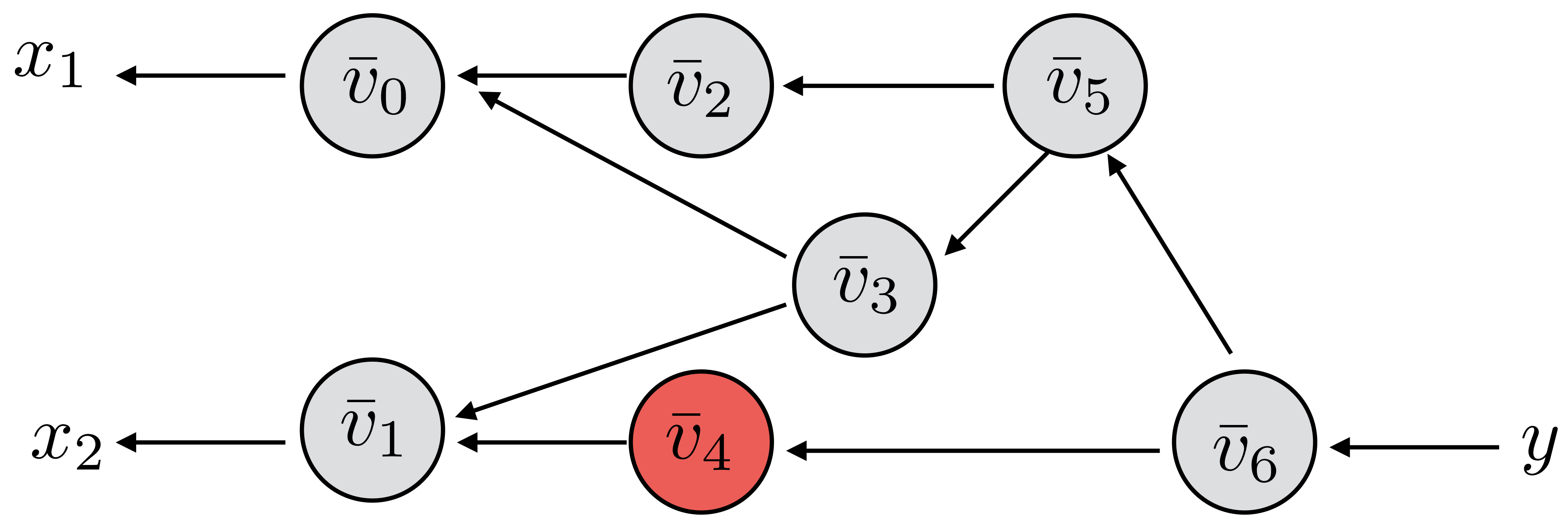
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





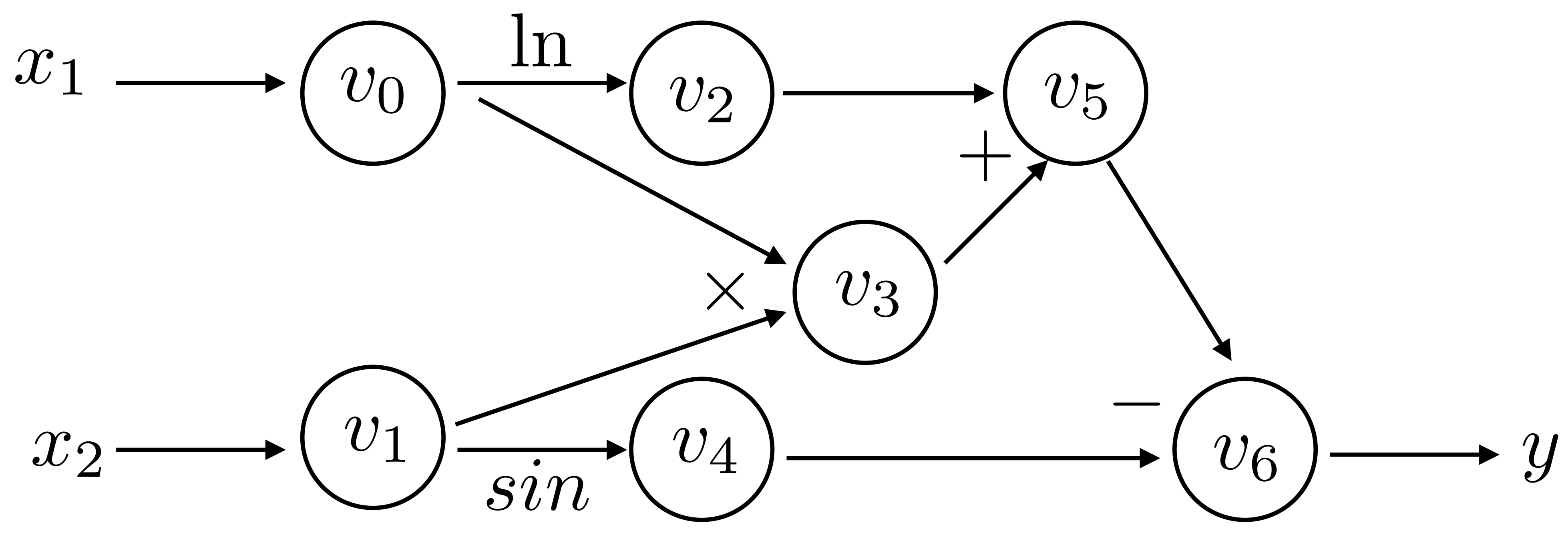
Backwards Derivative Trace:

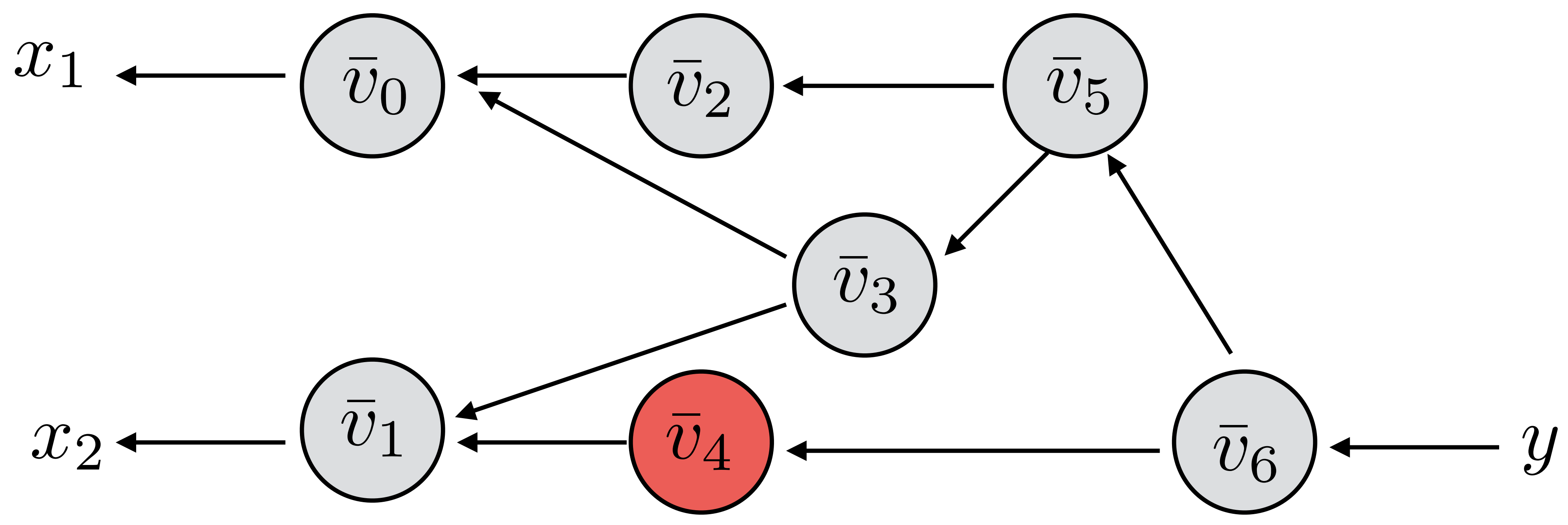
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4}$	
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





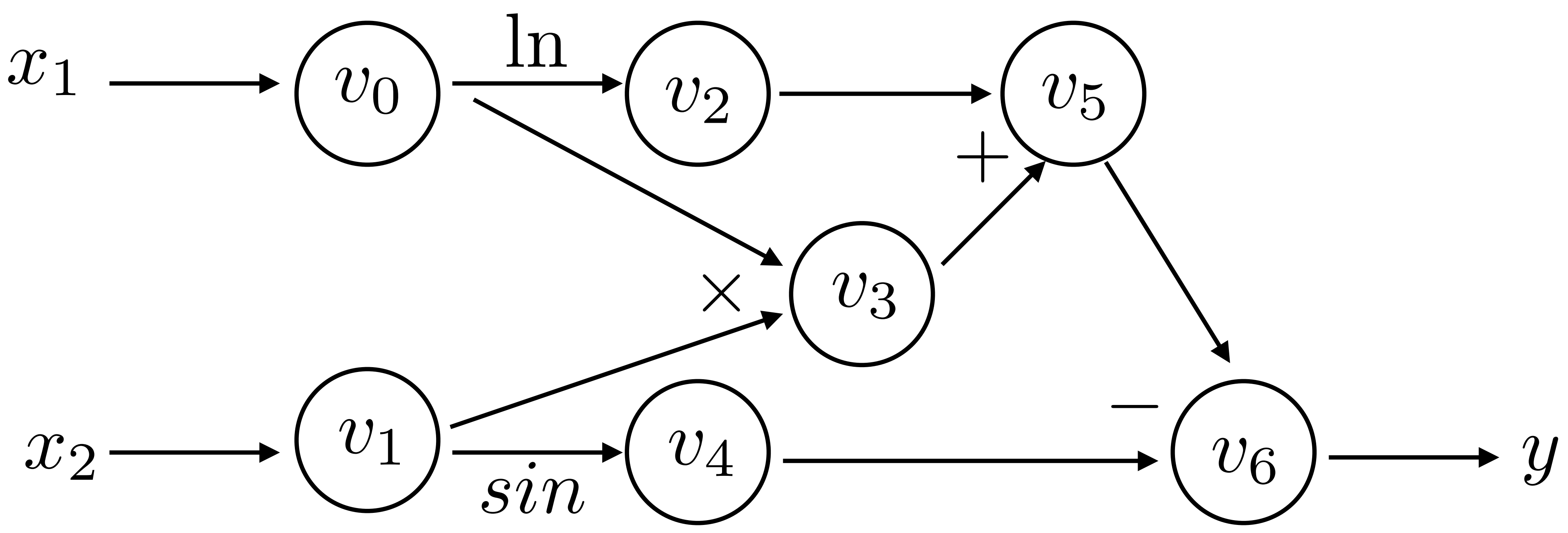
Backwards Derivative Trace:

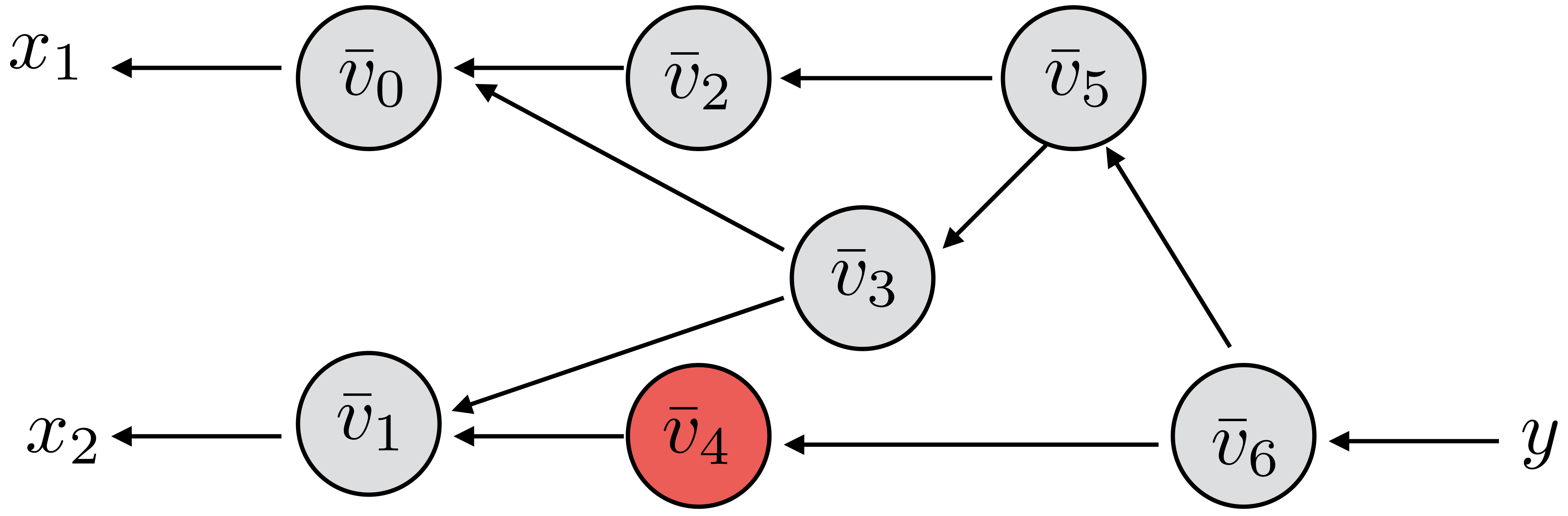
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





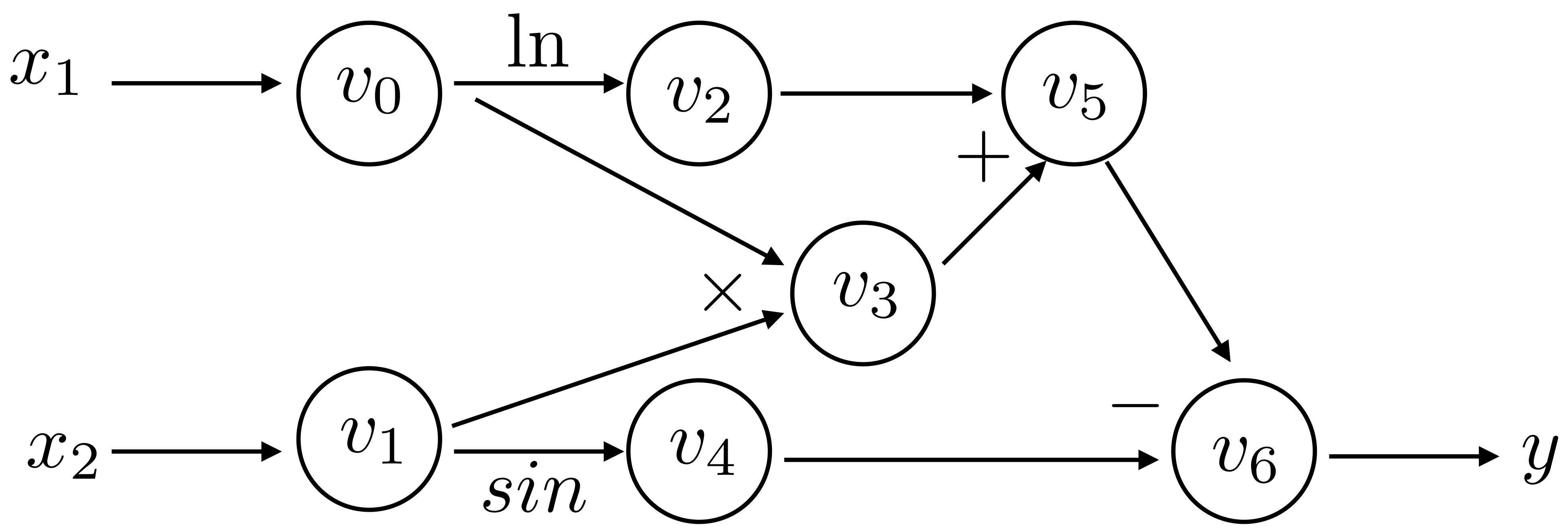
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

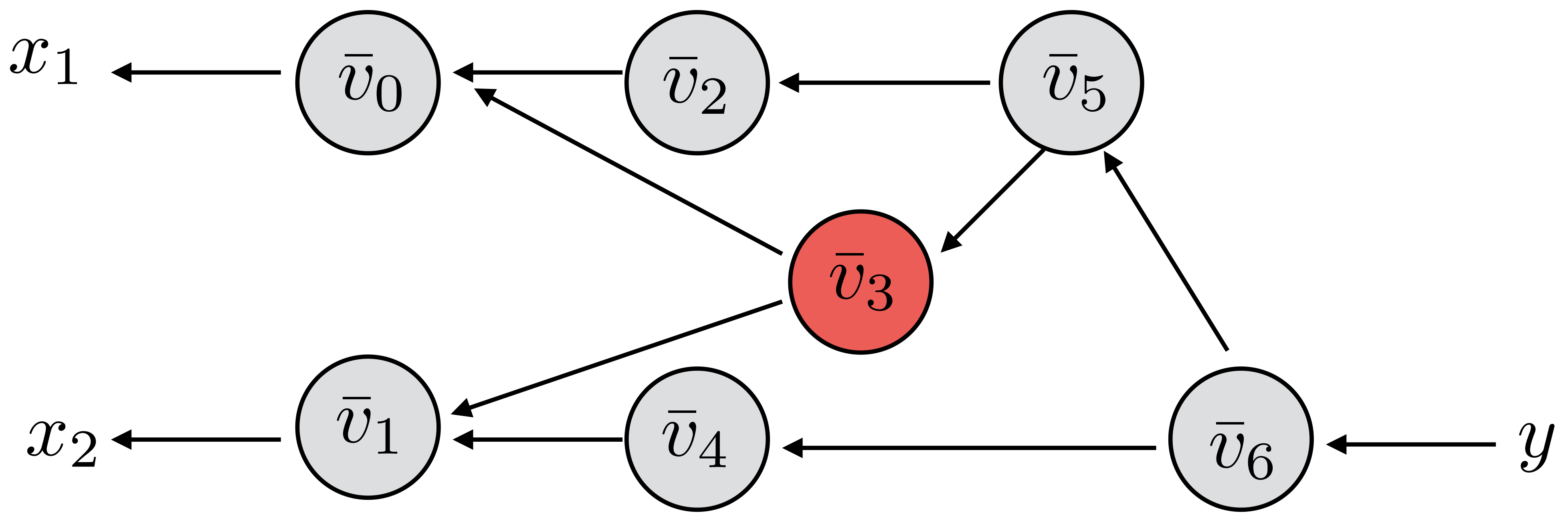
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

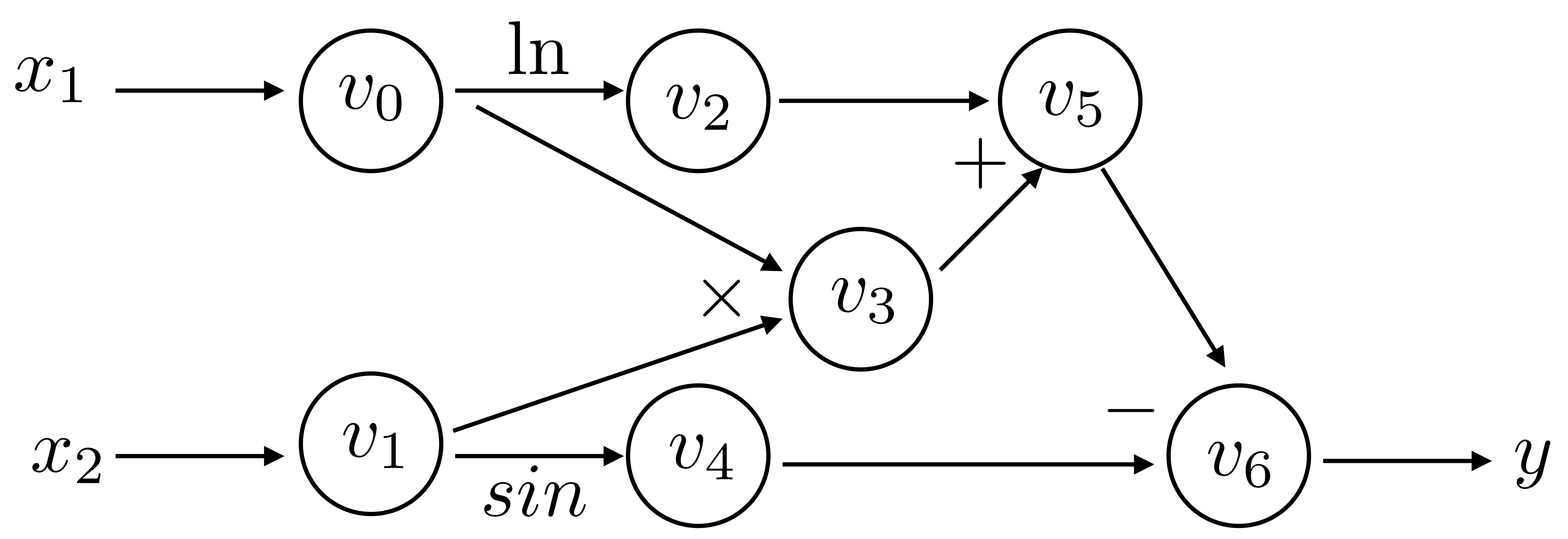
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

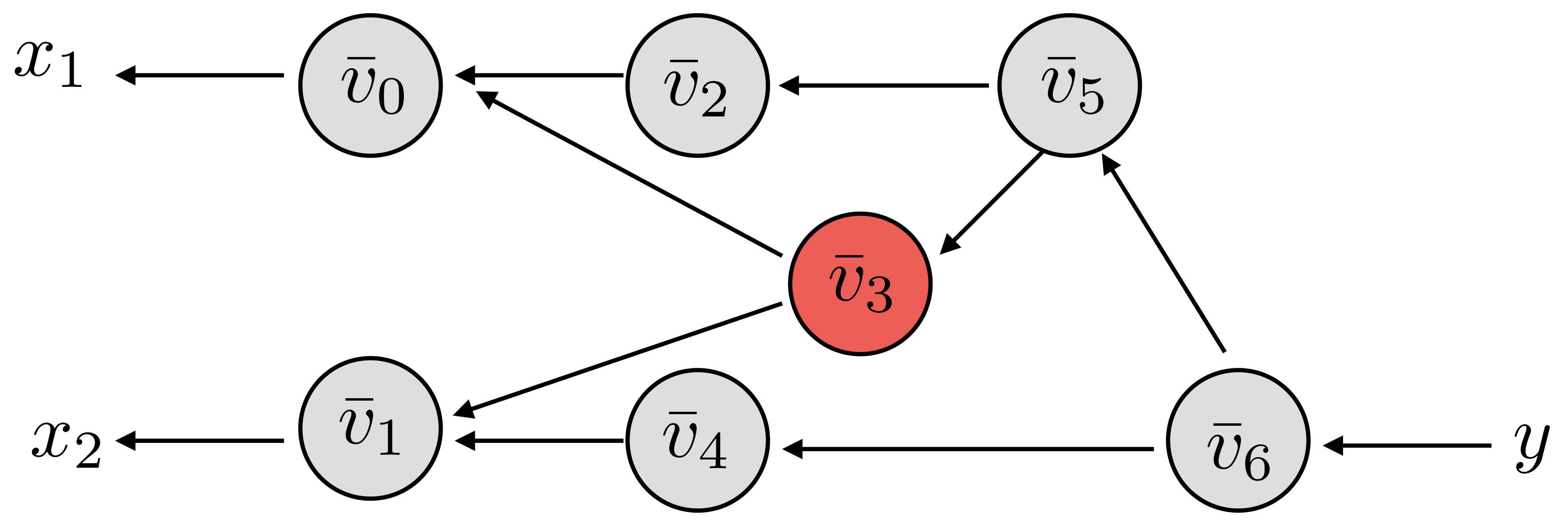


Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





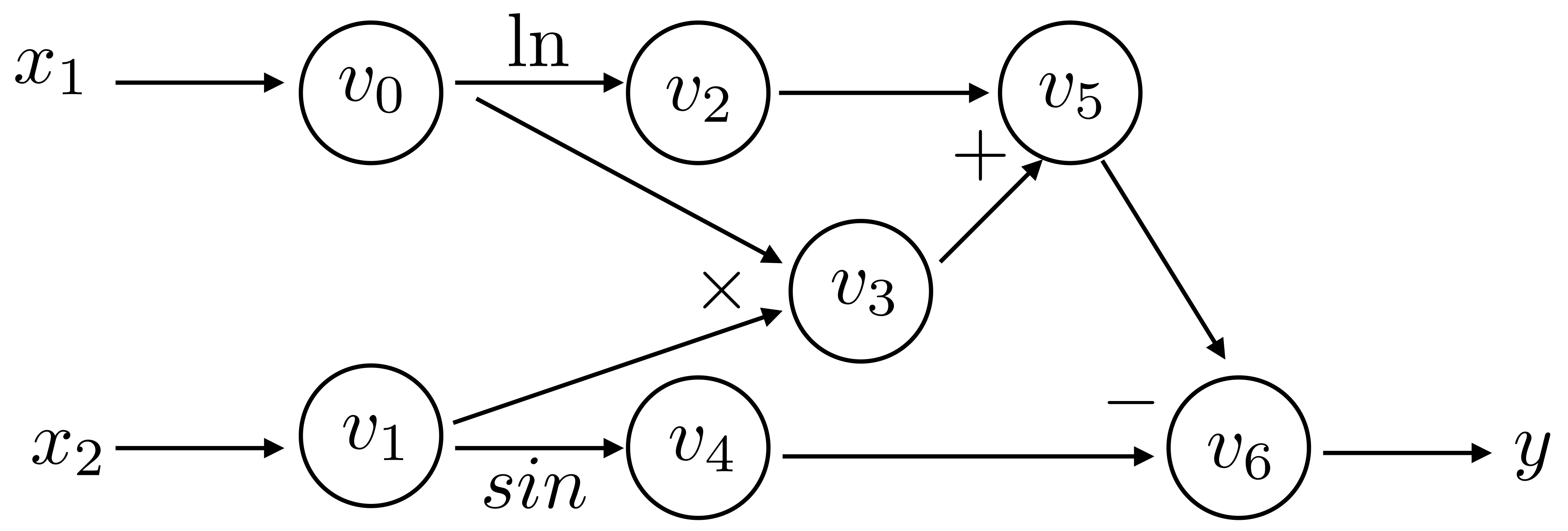
Forward Evaluation Trace:

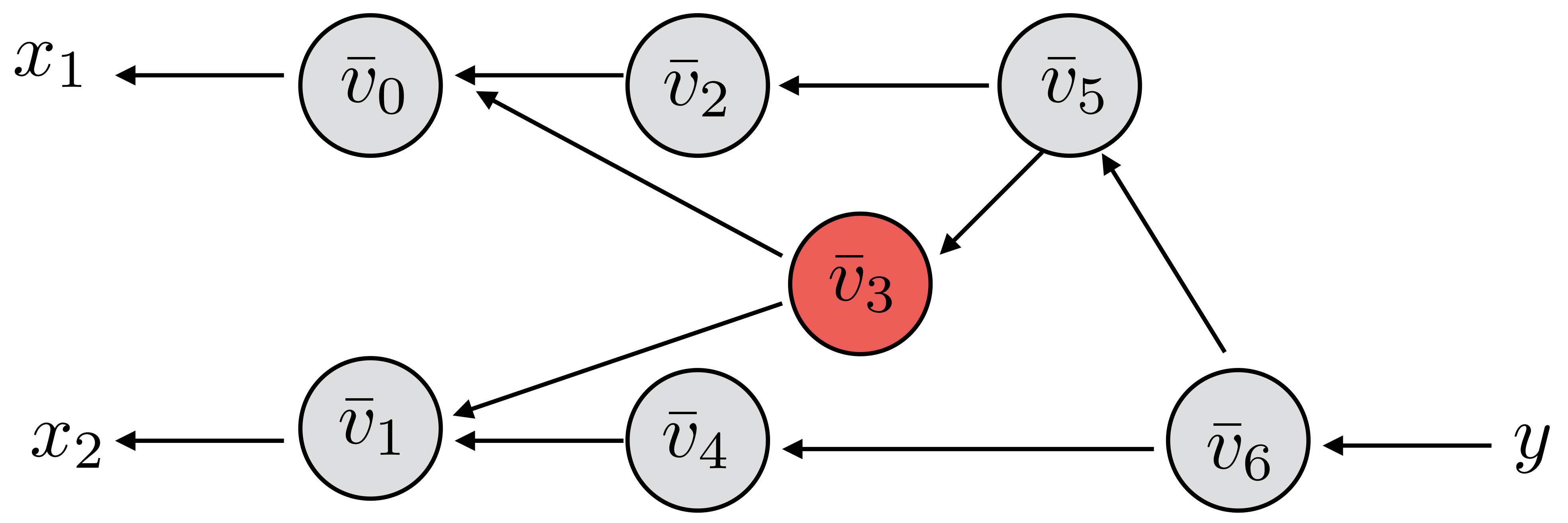
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3}$	
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





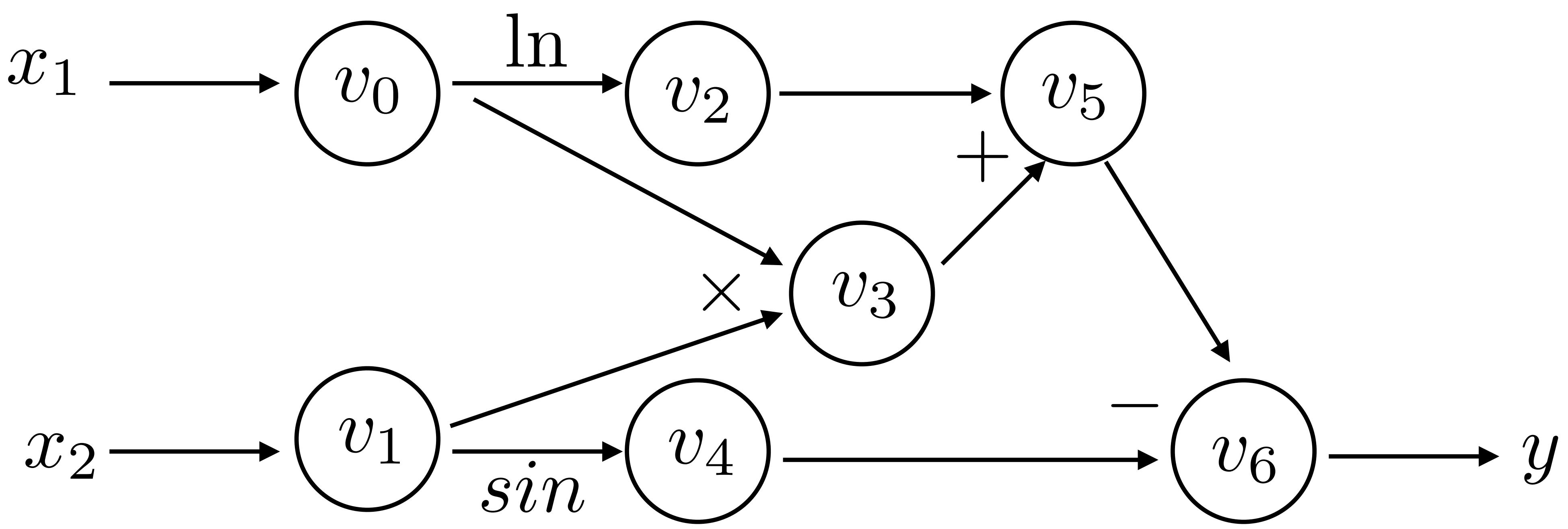
Forward Evaluation Trace:

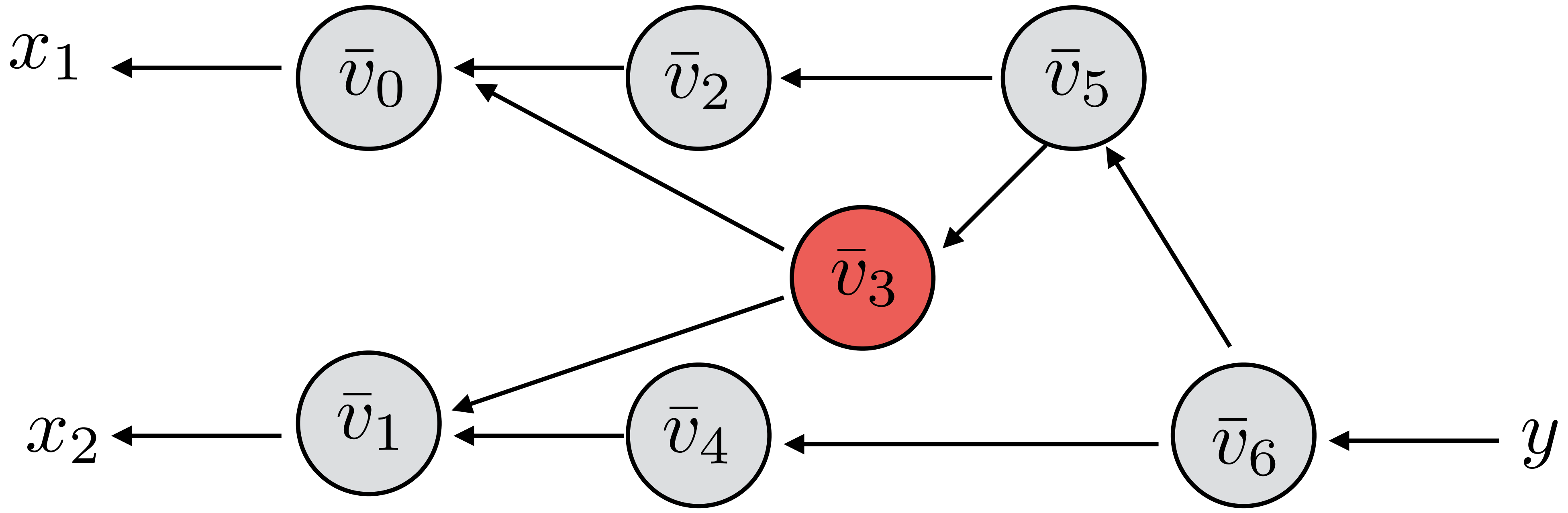
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





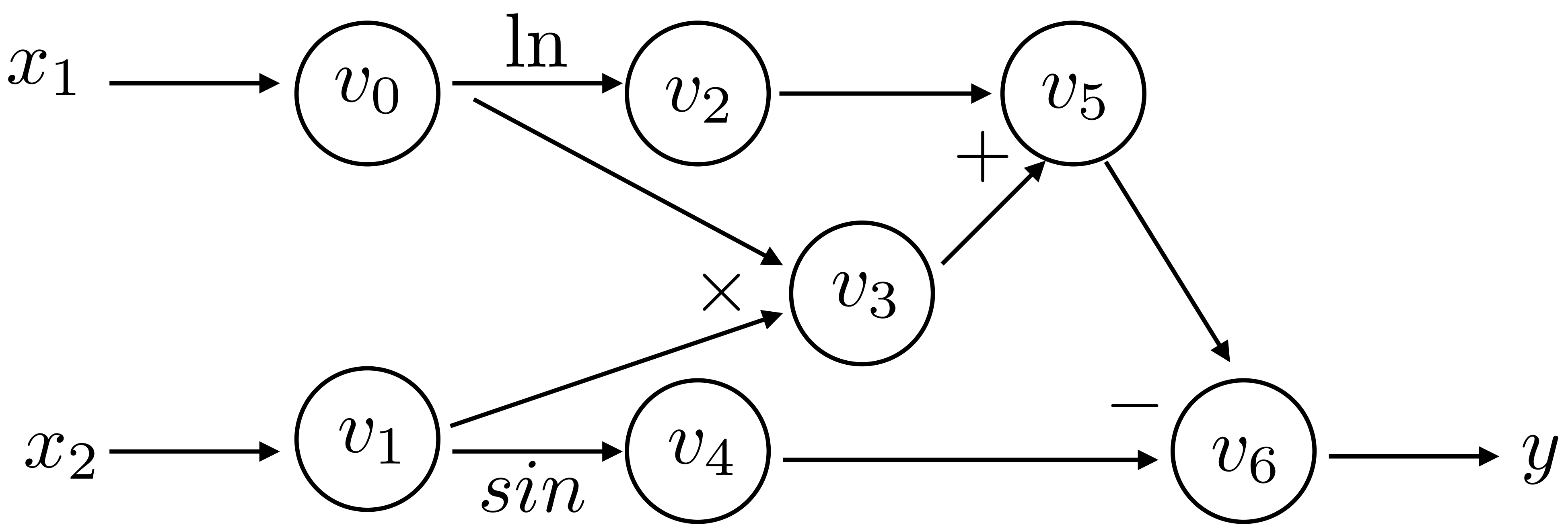
Forward Evaluation Trace:

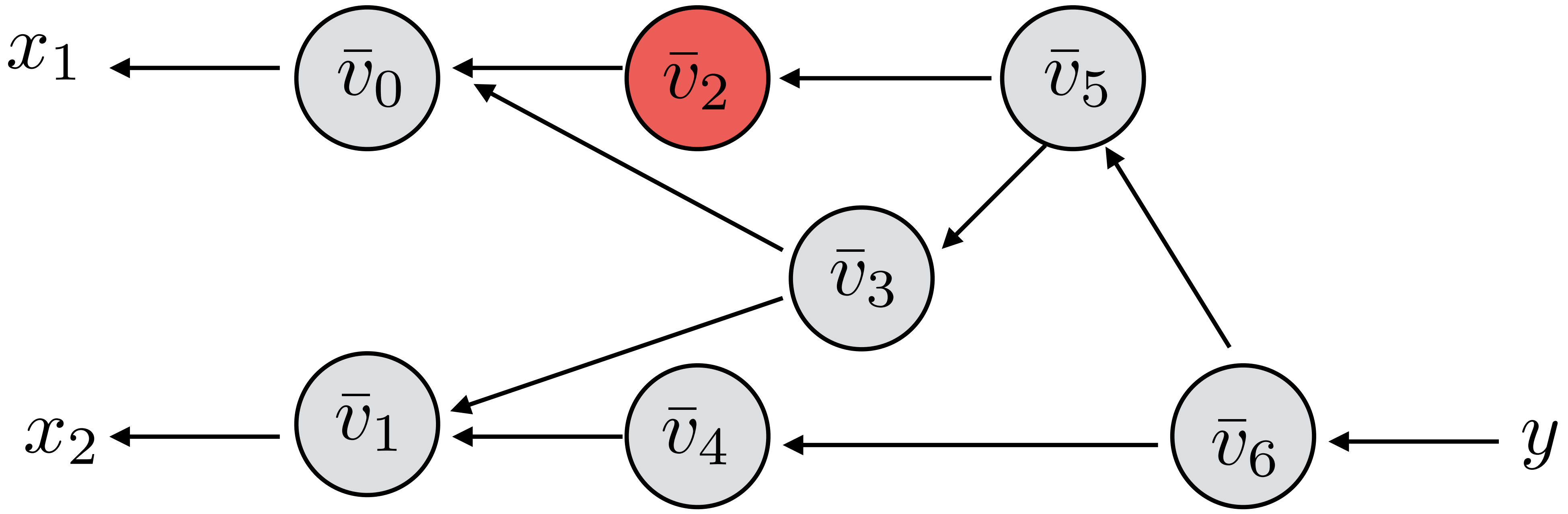
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





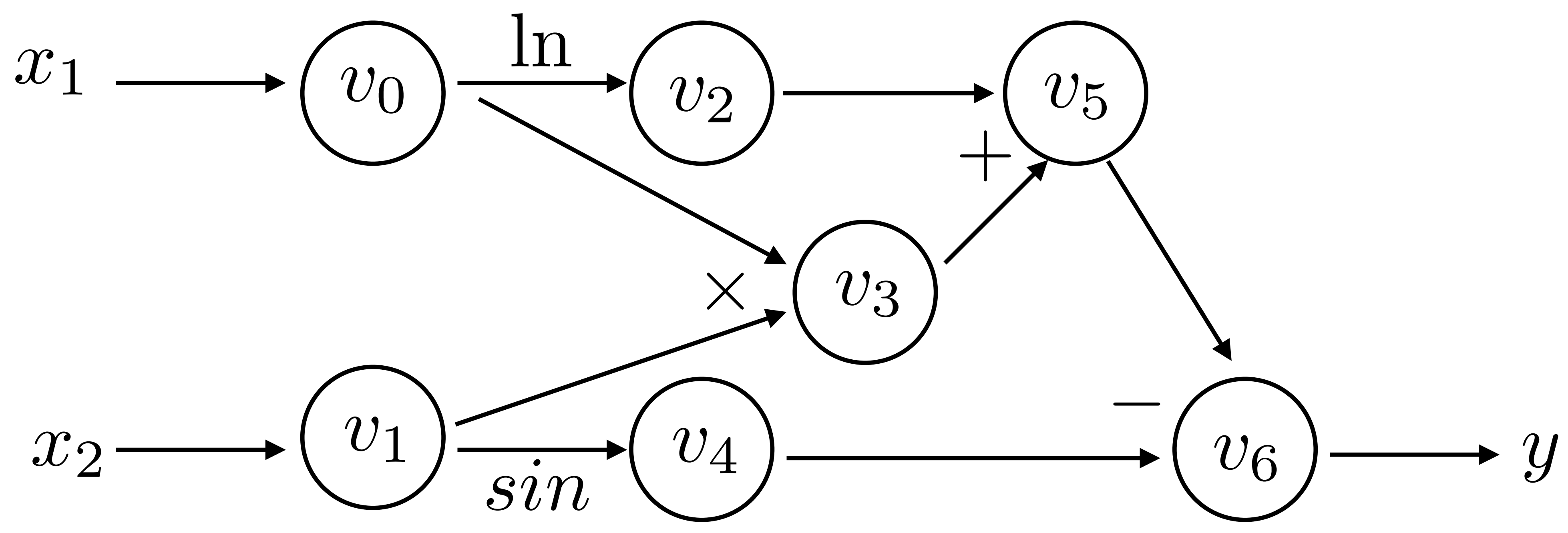
Forward Evaluation Trace:

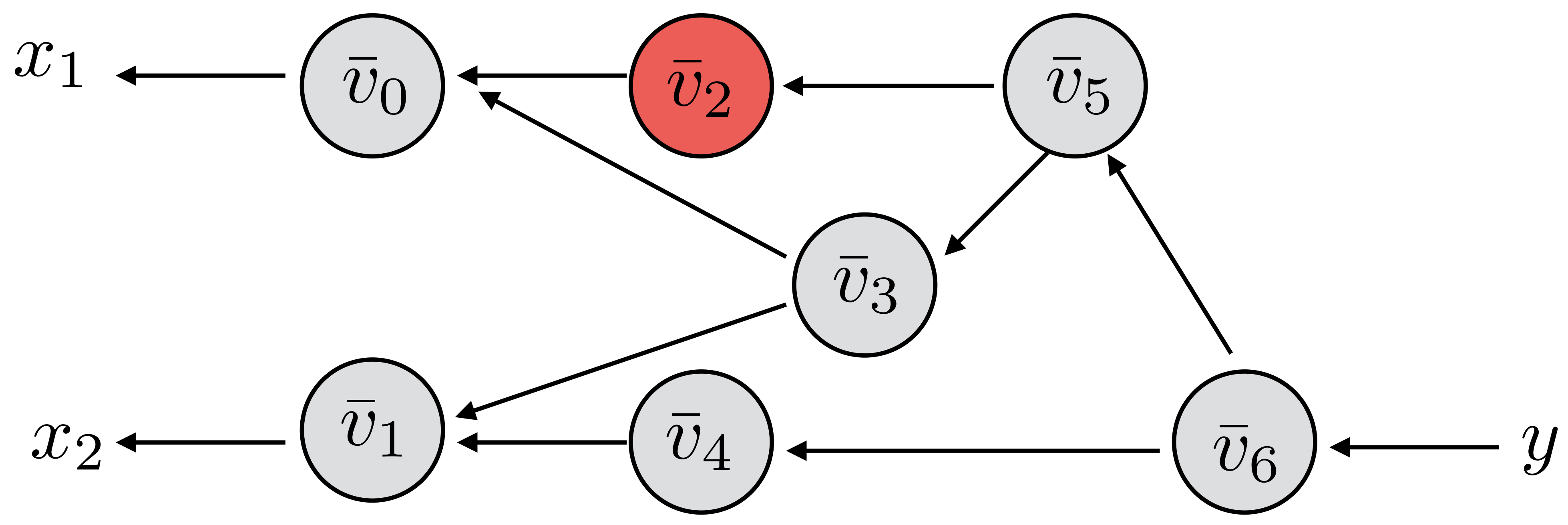
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2}$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





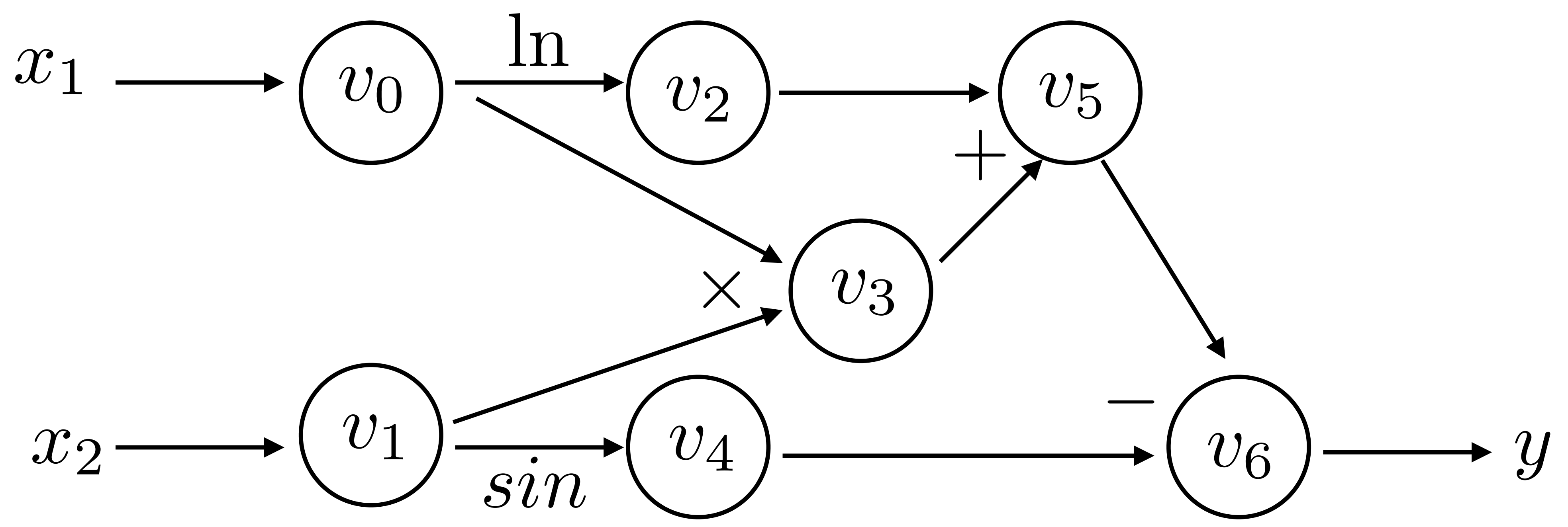
Forward Evaluation Trace:

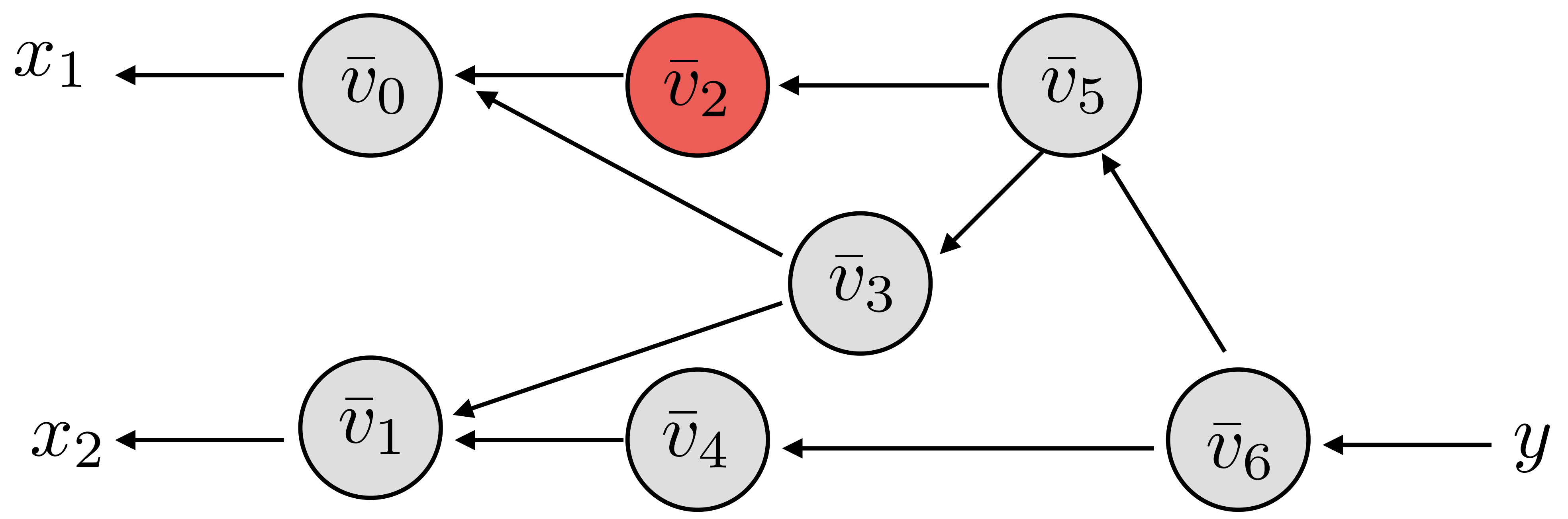
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_2 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_2}$	
$\bar{v}_3 = \bar{v}_5 \dfrac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \dfrac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \dfrac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





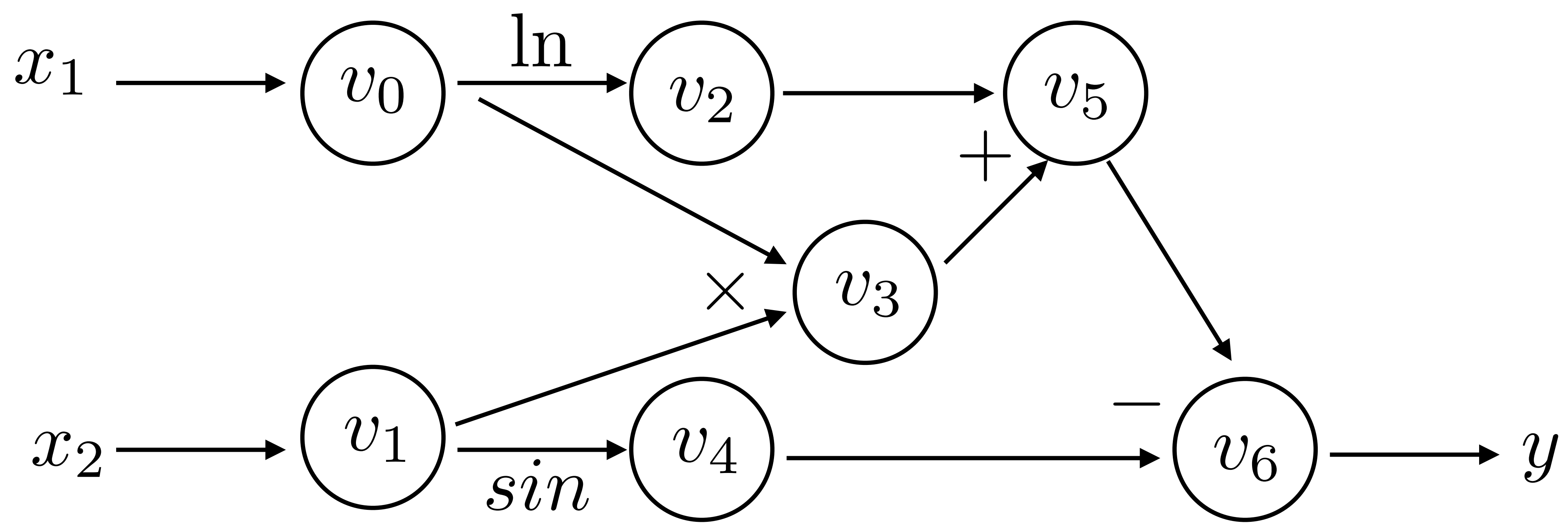
Forward Evaluation Trace:

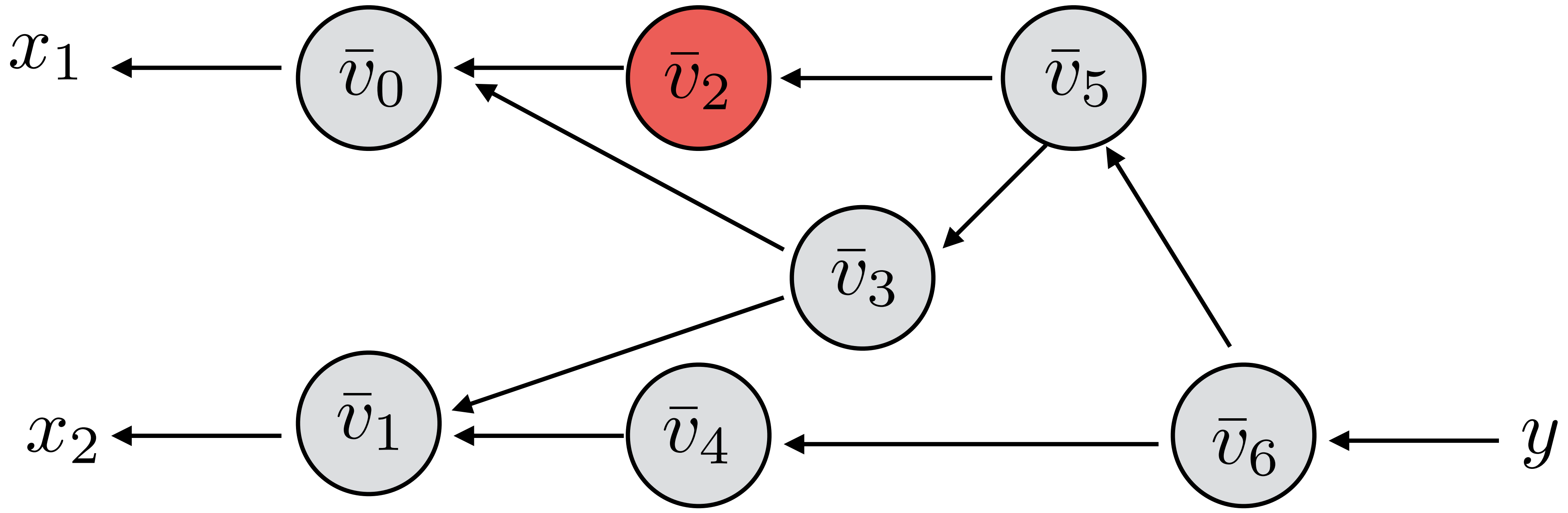
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





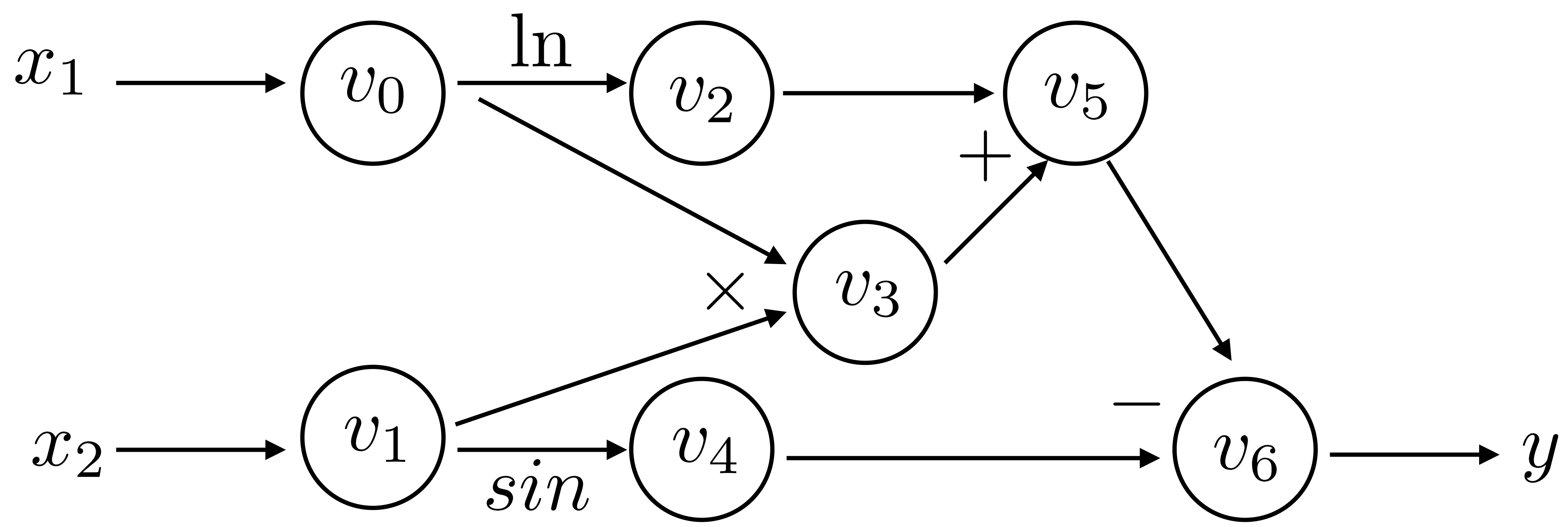
Backwards Derivative Trace:

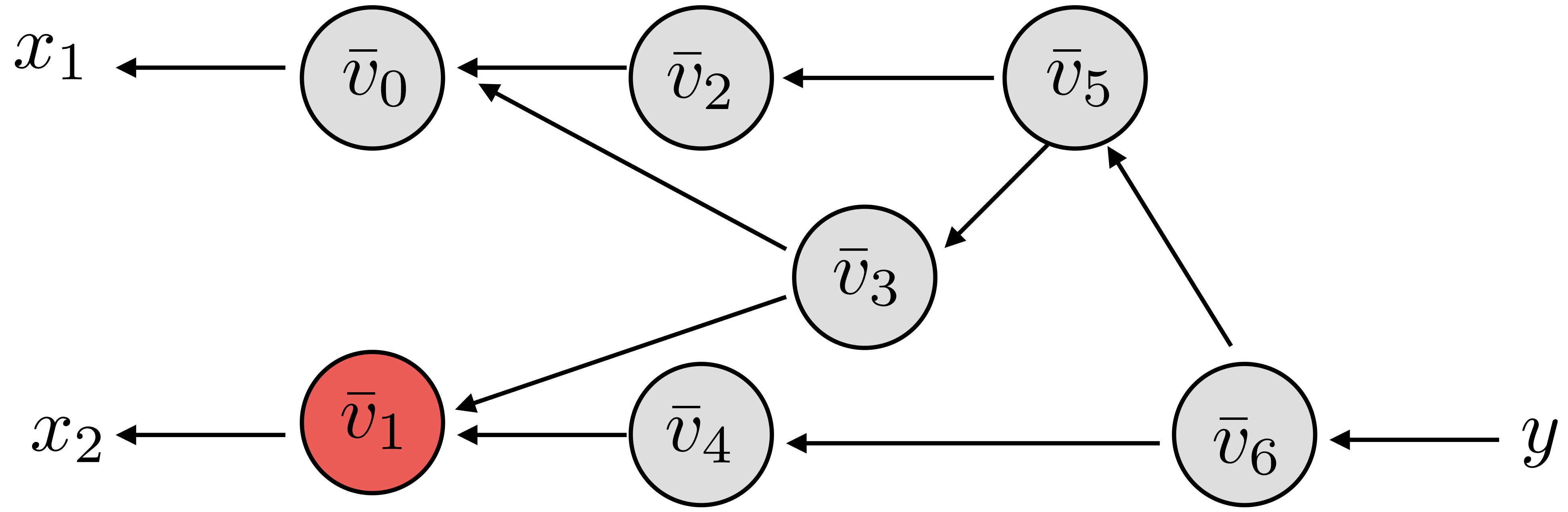
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





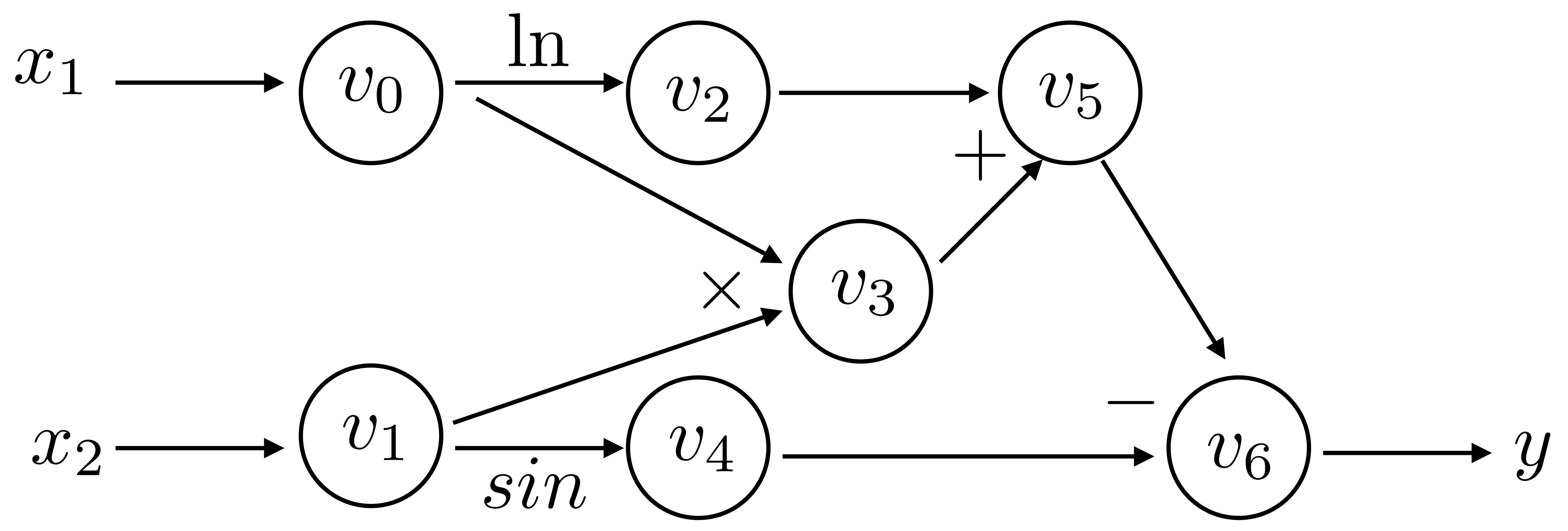
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

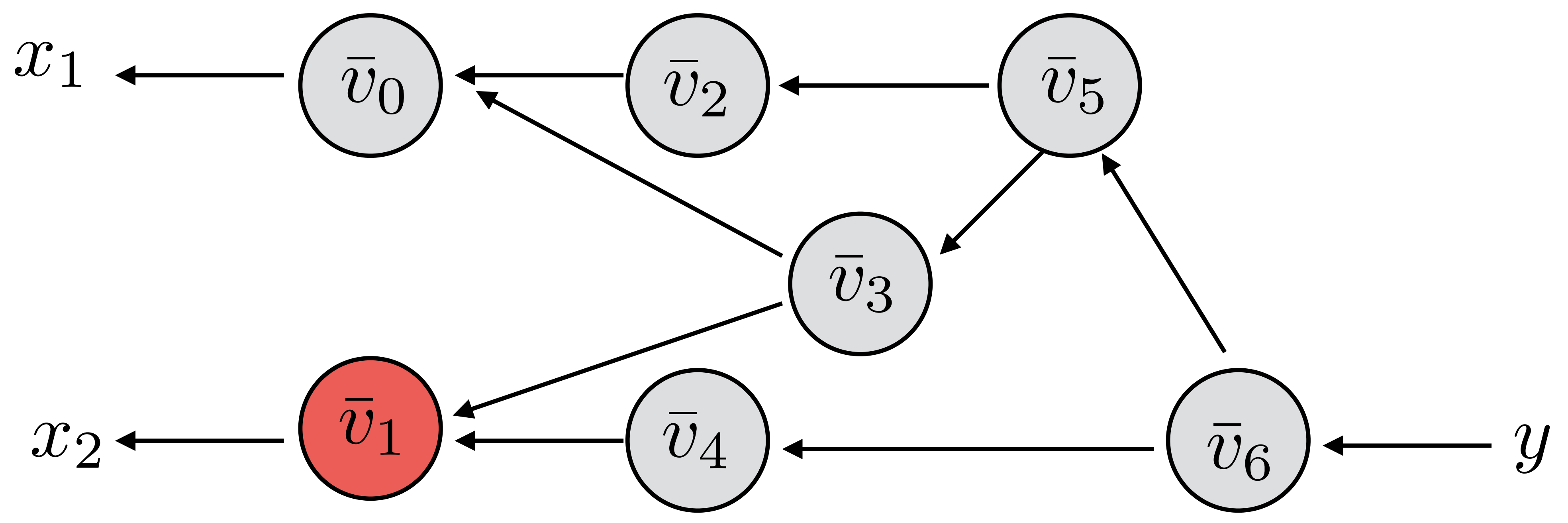
$\bar{v}_1$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

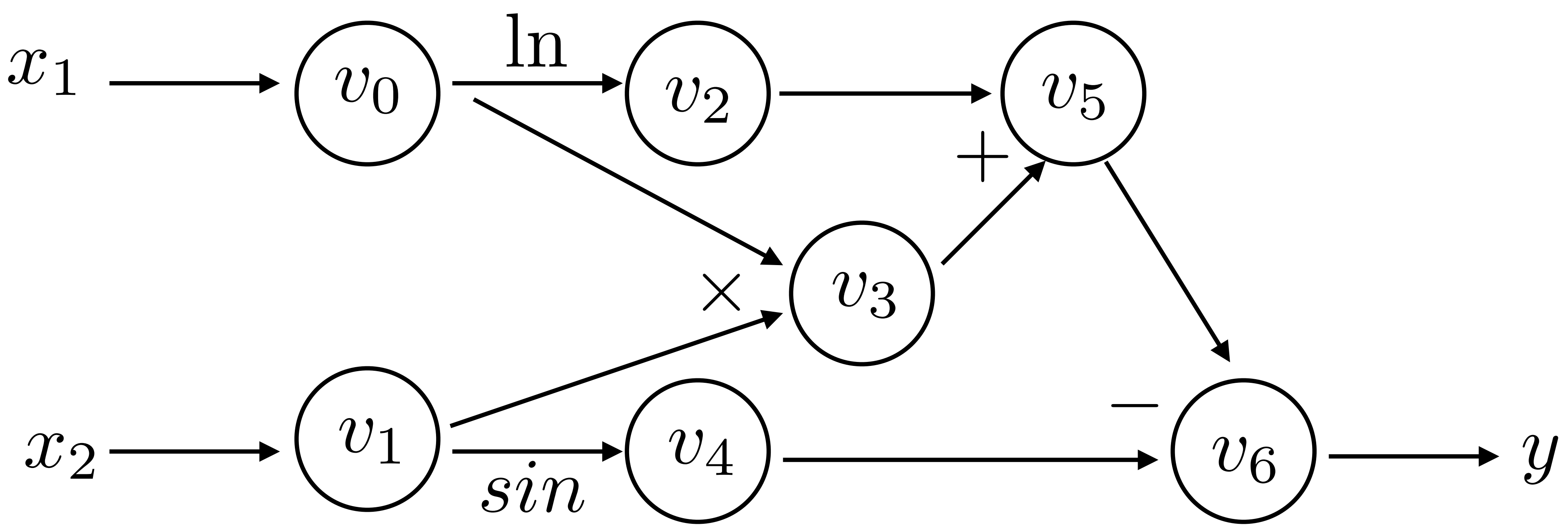
	$f(2, 5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

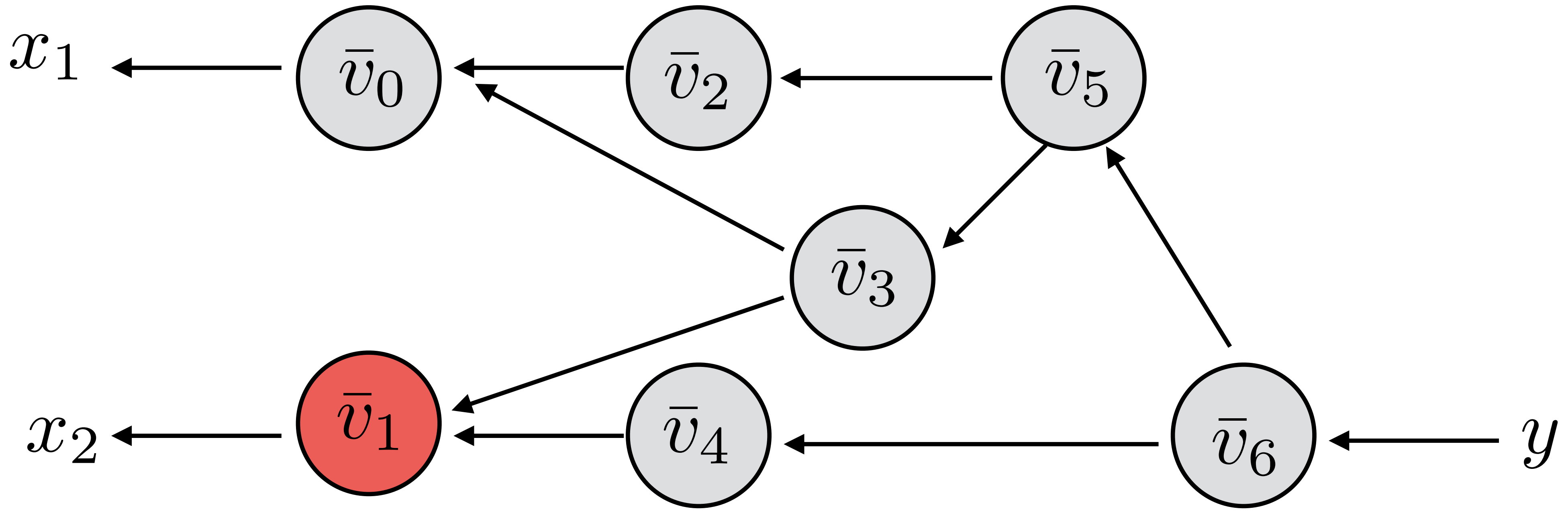


Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





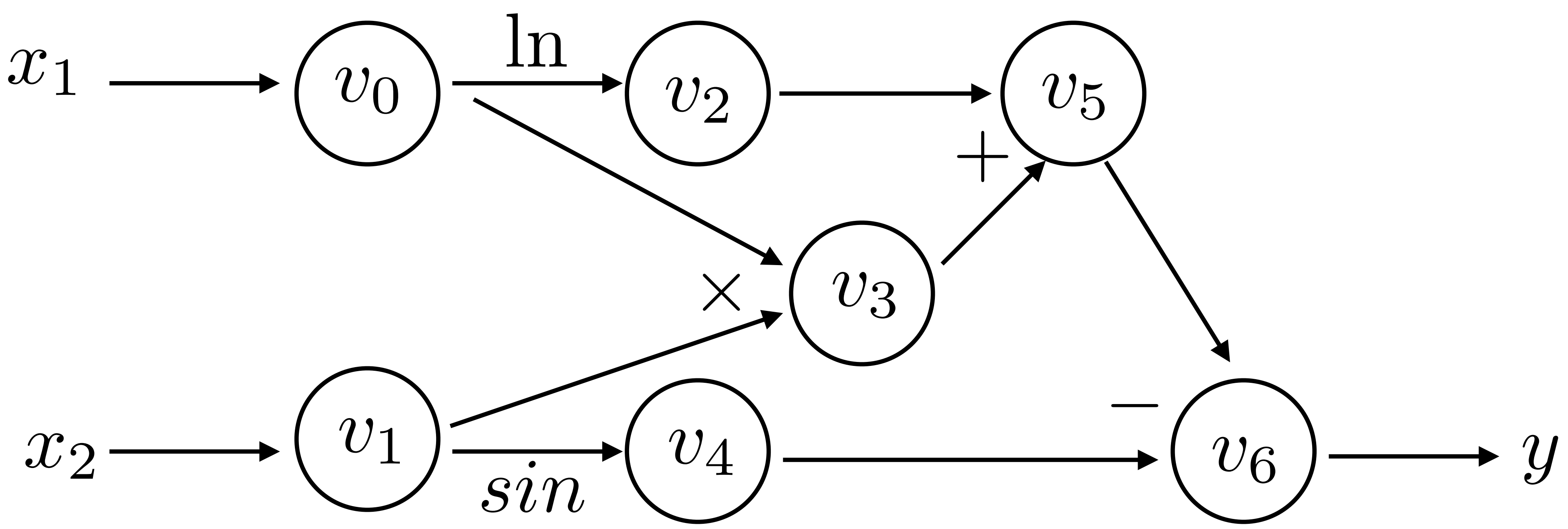
Forward Evaluation Trace:

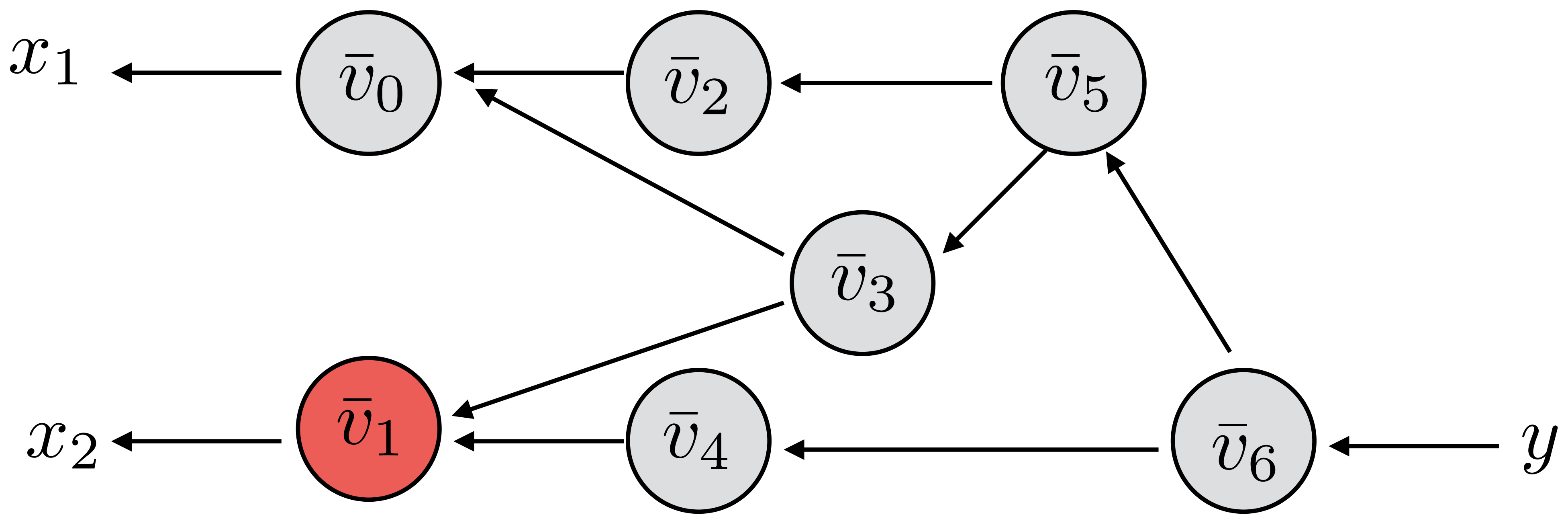
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





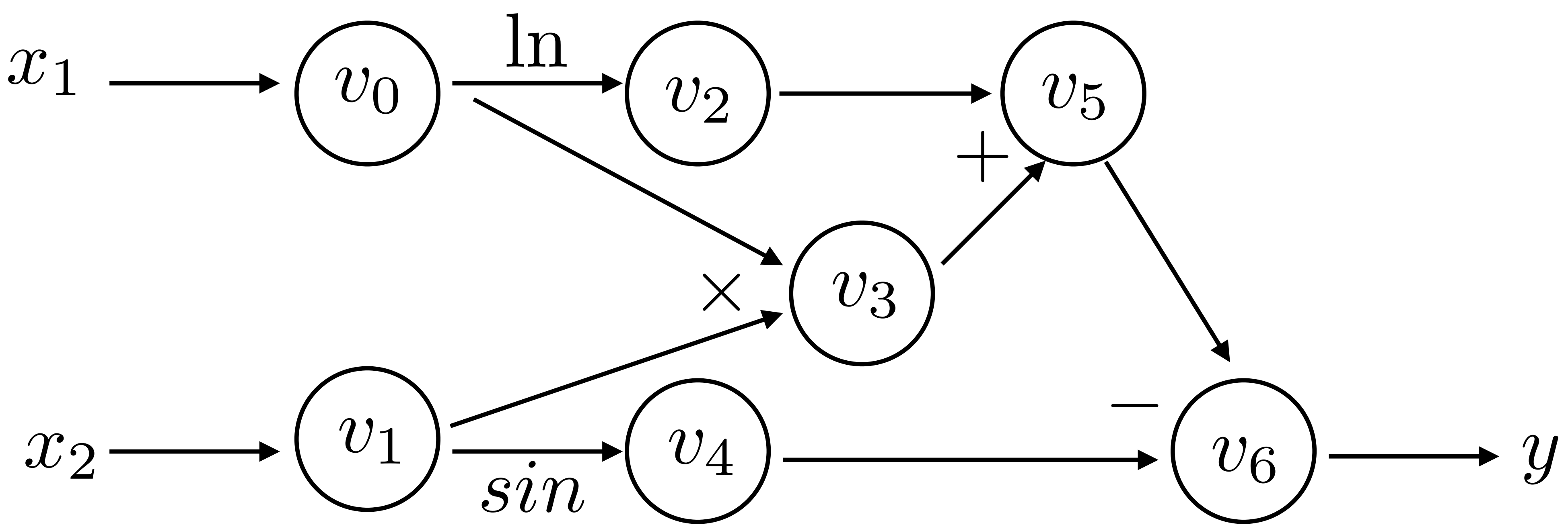
Backwards Derivative Trace:

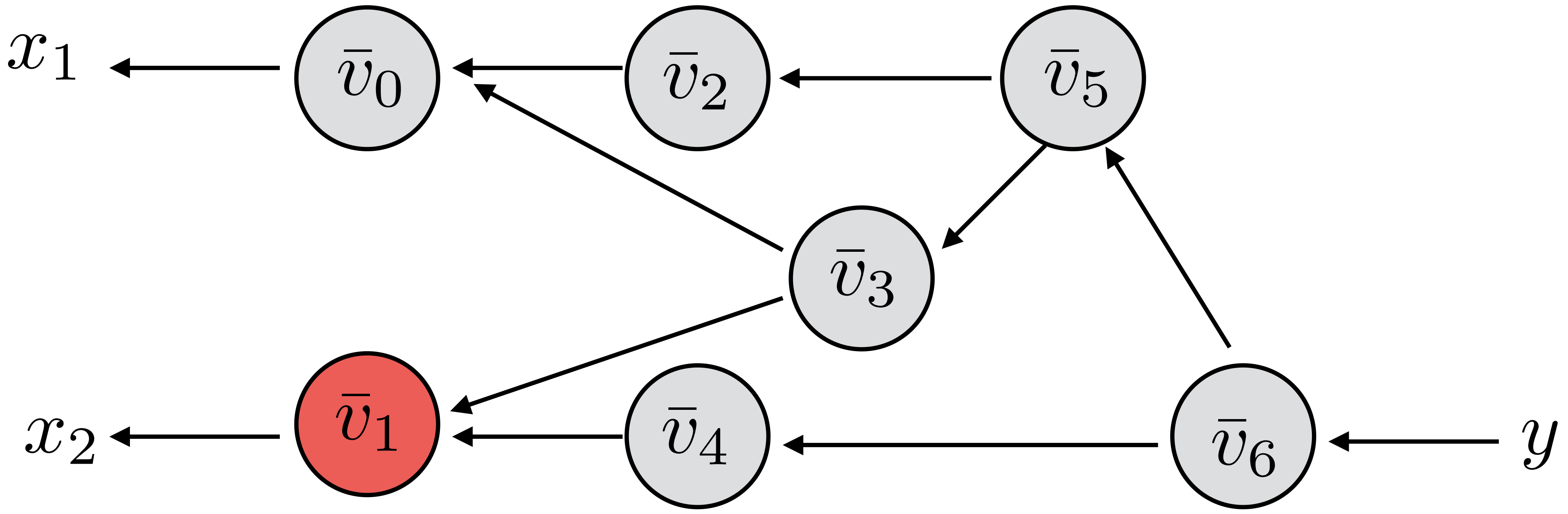
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

AutoDiff - **Reverse Mode**





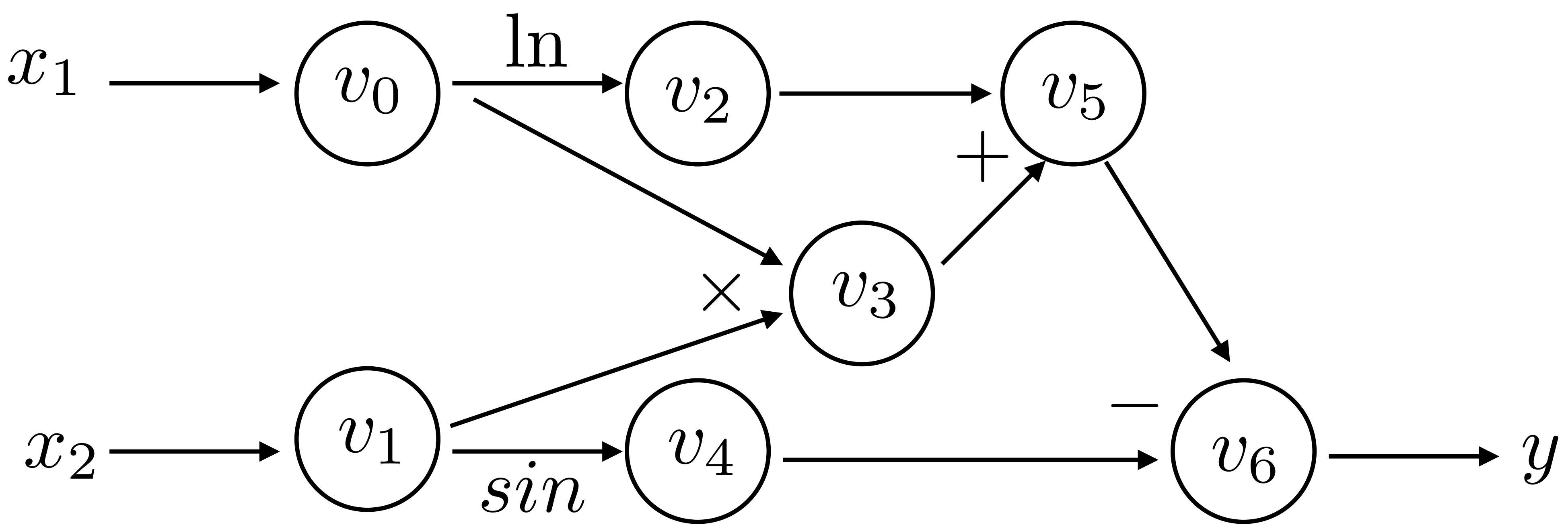
Forward Evaluation Trace:

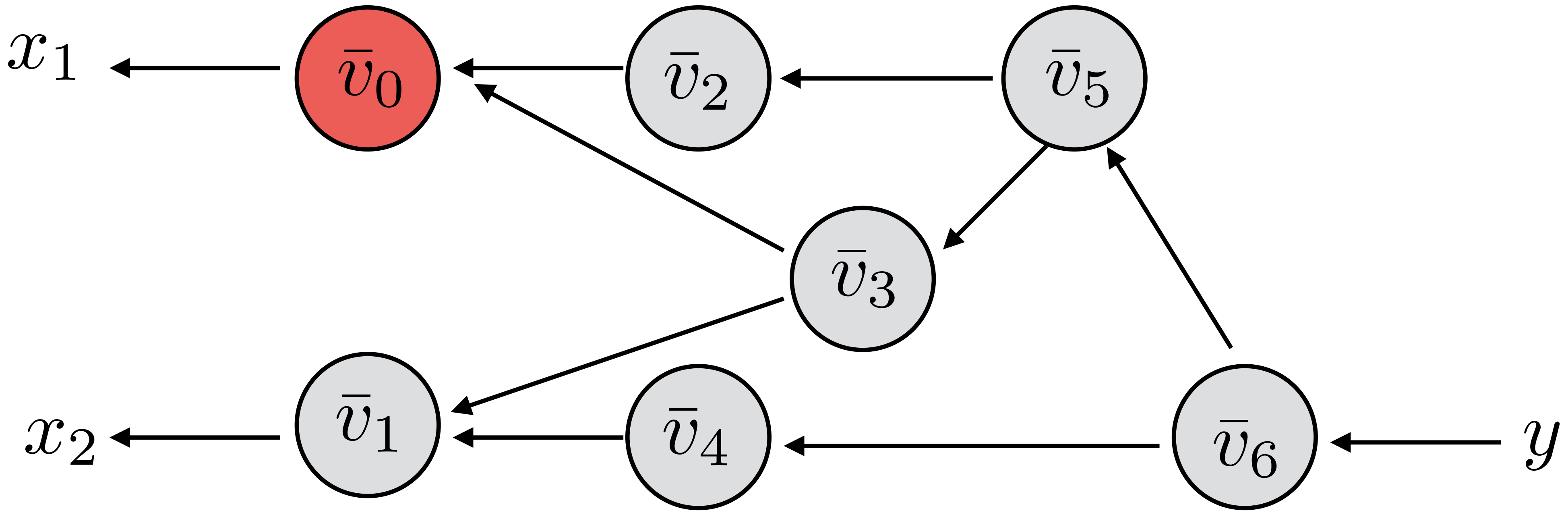
	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**





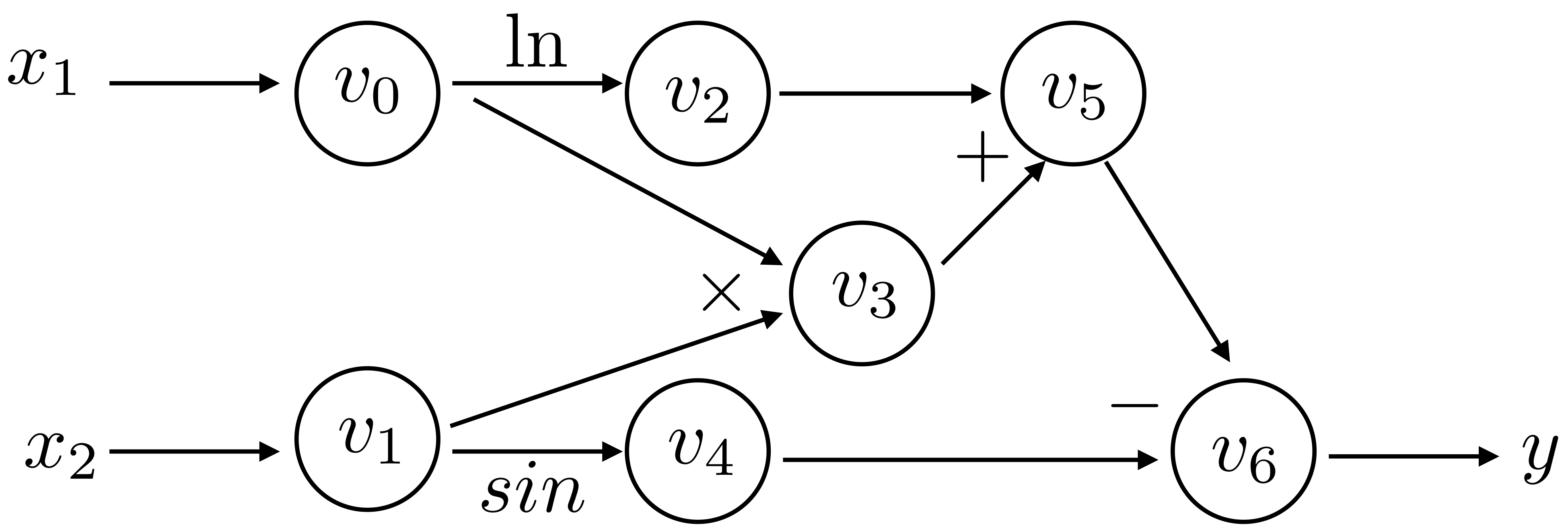
Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652

Backwards Derivative Trace:

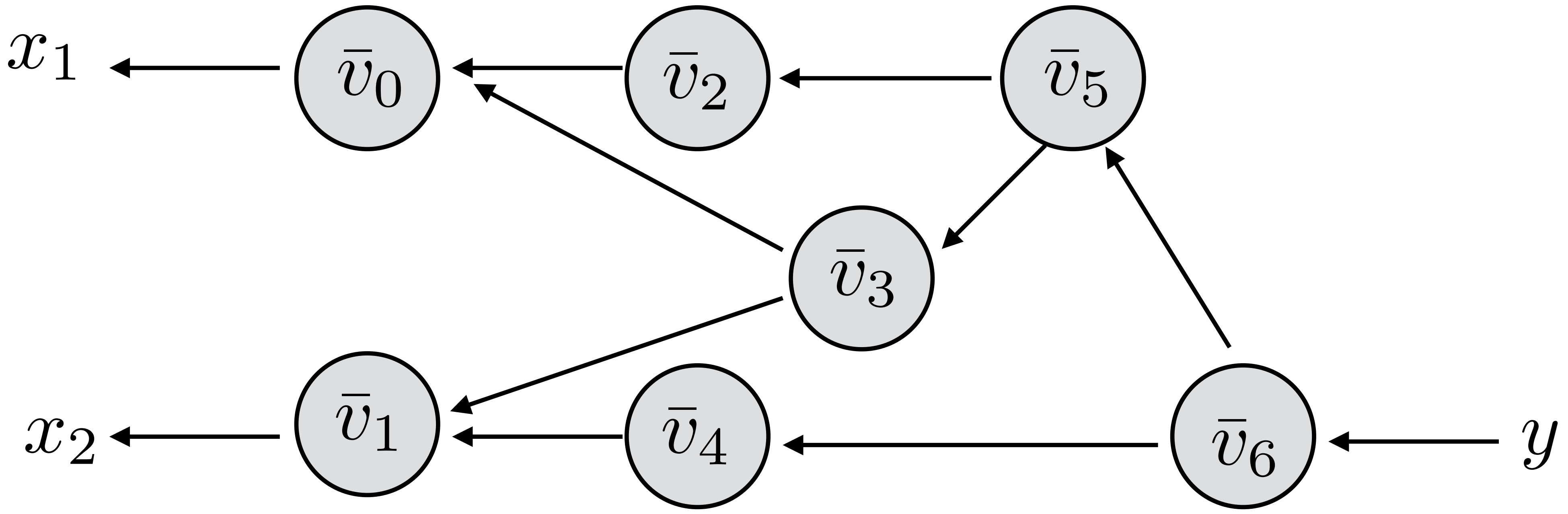
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_3 v_1 + \bar{v}_2 \frac{1}{v_0}$	5.5
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	1.716
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

AutoDiff - **Reverse Mode**



Forward Evaluation Trace:

	$f(2,5)$
$v_0 = x_1$	2
$v_1 = x_2$	5
$v_2 = \ln(v_0)$	ln(2) = 0.693
$v_3 = v_0 \cdot v_1$	2 x 5 = 10
$v_4 = sin(v_1)$	sin(5) = 0.959
$v_5 = v_2 + v_3$	0.693 + 10 = 10.693
$v_6 = v_5 - v_4$	10.693 + 0.959 = 11.652
$y = v_6$	11.652



Backwards Derivative Trace:

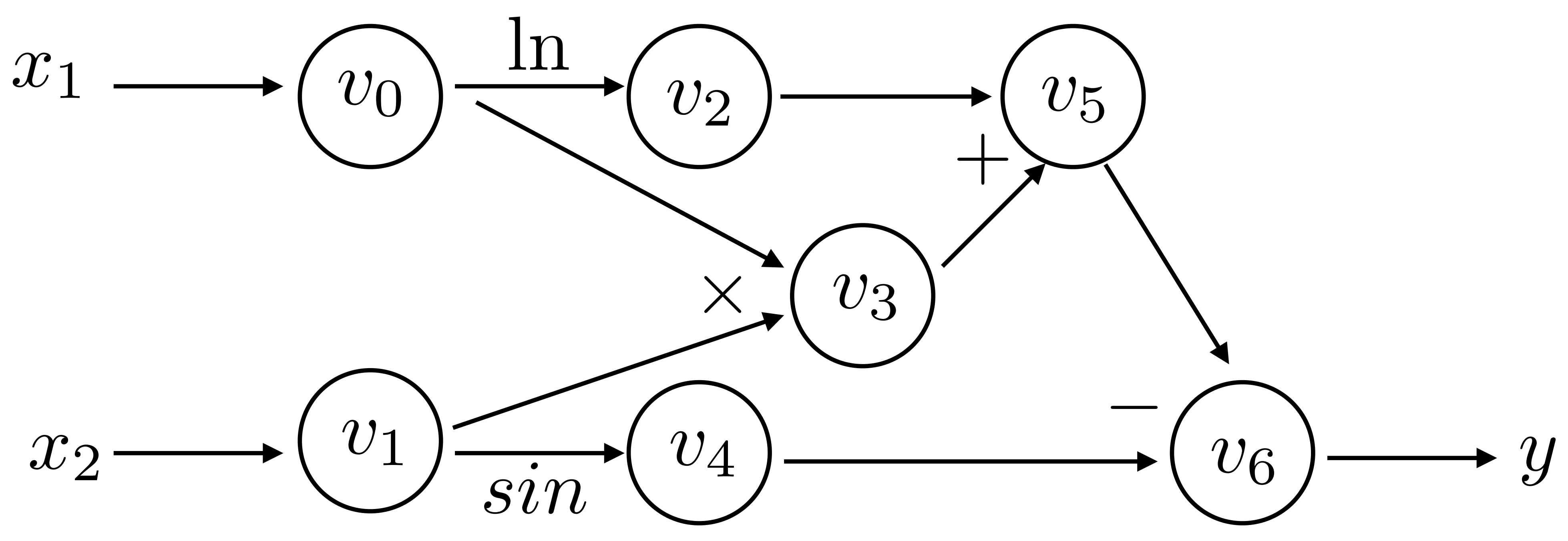
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_3 v_1 + \bar{v}_2 \frac{1}{v_0}$	**5.5**
$\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1} + \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_3 v_0 + \bar{v}_4 cos(v_1)$	**1.716**
$\bar{v}_2 = \bar{v}_5 \frac{\partial v_5}{\partial v_2} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \cdot (1)$	1x1 = 1
$\bar{v}_4 = \bar{v}_6 \frac{\partial v_6}{\partial v_4} = \bar{v}_6 \cdot (-1)$	1x-1 = -1
$\bar{v}_5 = \bar{v}_6 \frac{\partial v_6}{\partial v_5} = \bar{v}_6 \cdot 1$	1x1 = 1
$\bar{v}_6 = \frac{\partial y}{\partial v_6}$	1

Automatic Differentiation (AutoDiff)

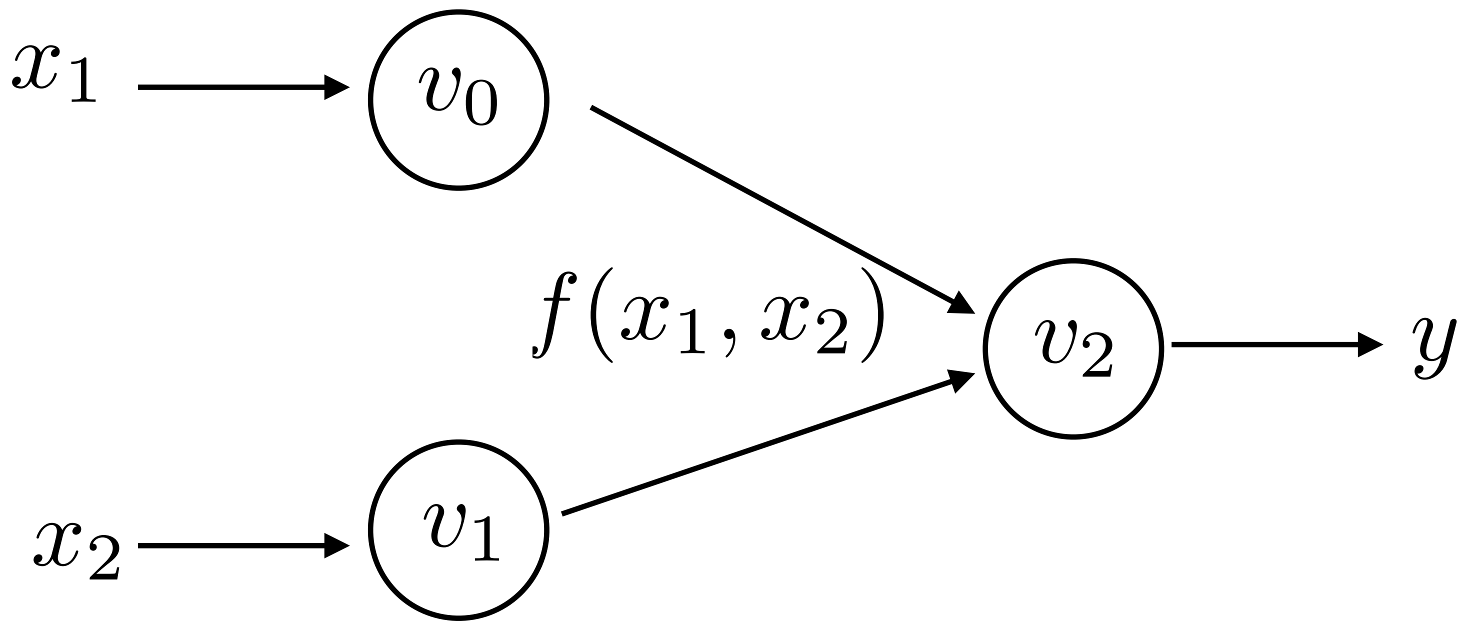
$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - sin(x_2)$$

AutoDiff can be done at various **granularities**

Elementary function granularity:

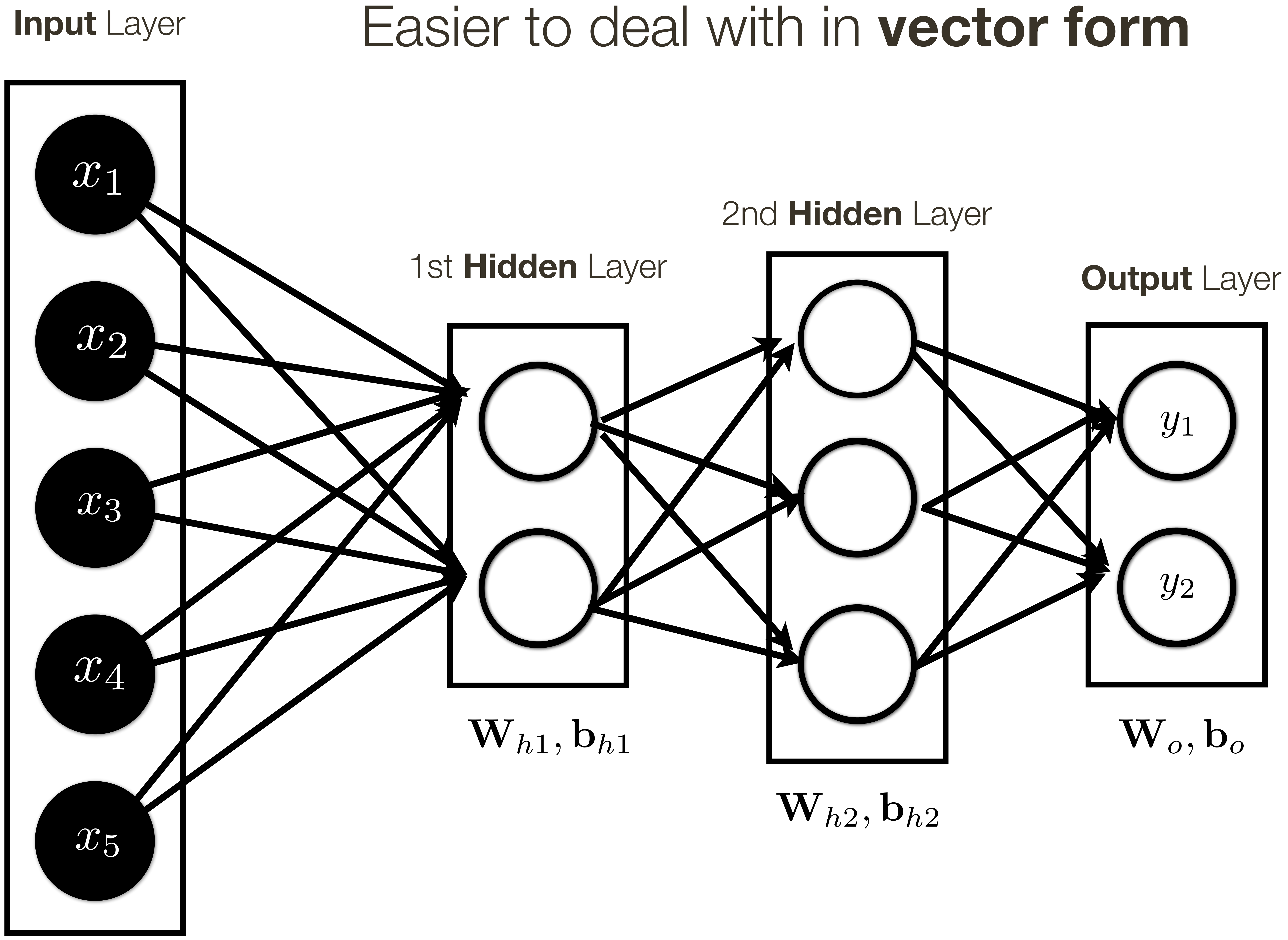


Complex function granularity:



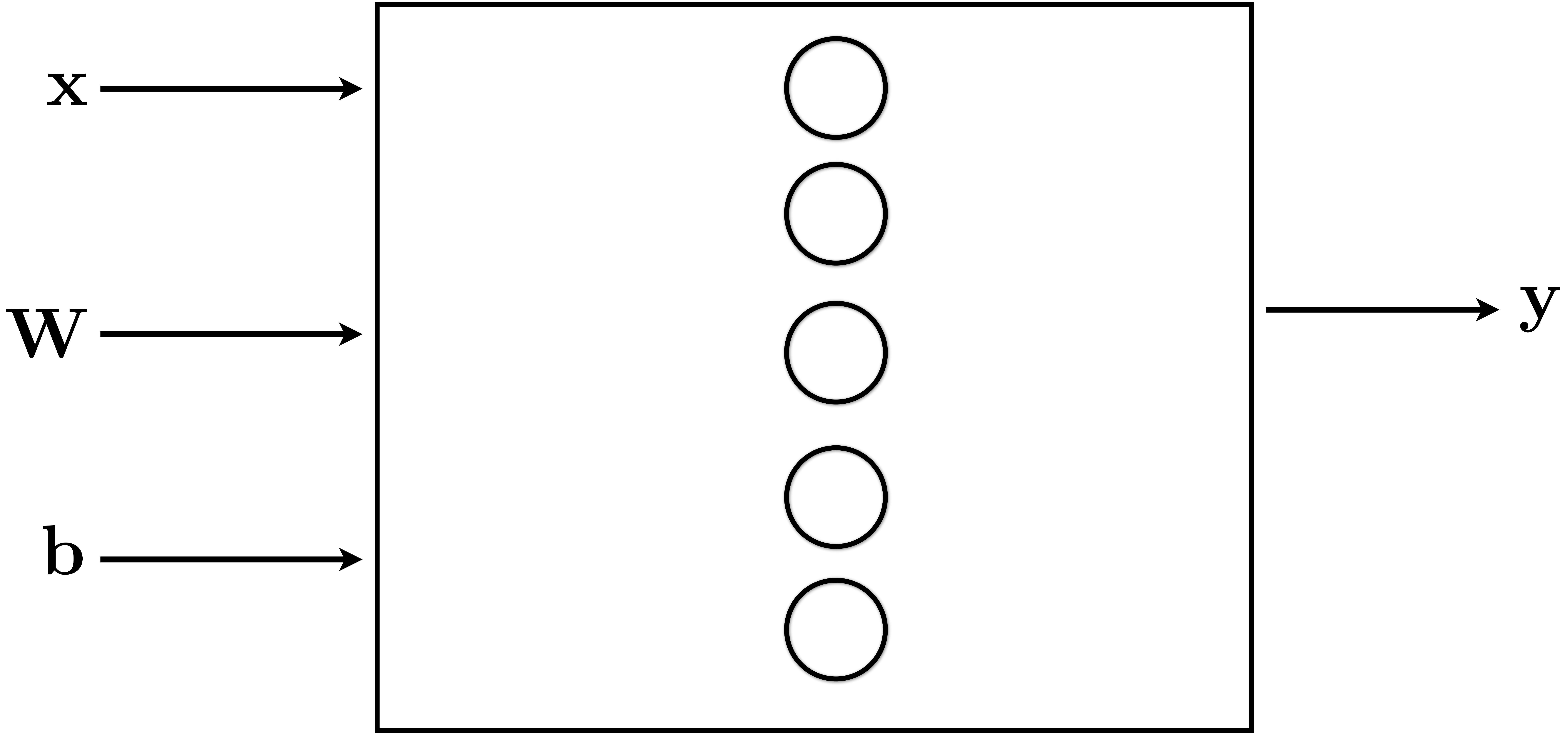
Backpropagation Practical Issues

Easier to deal with in **vector form**



Backpropagation Practical Issues

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

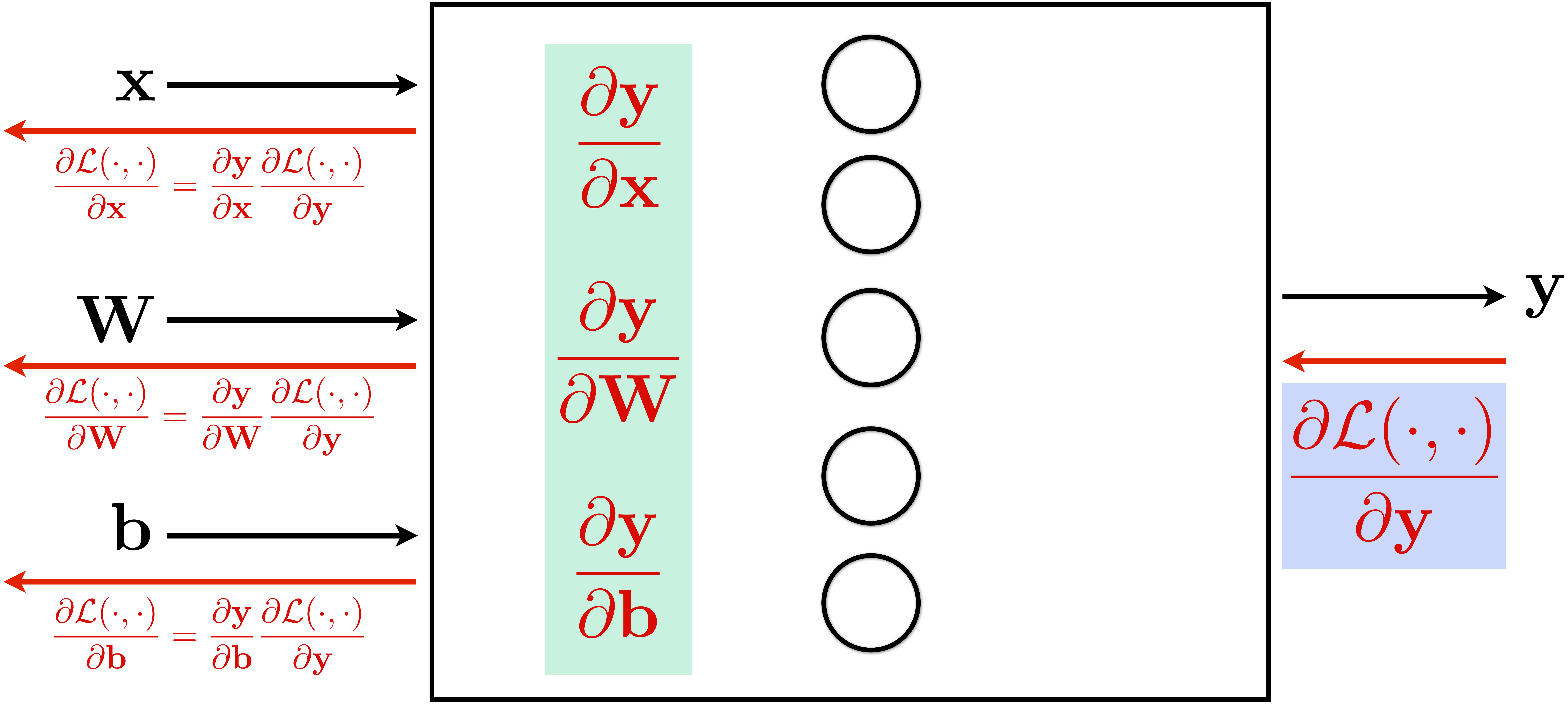


Backpropagation Practical Issues

"**local**" Jacobians
(matrix of partial derivatives, e.g. size |x| x |y|)

"**backprop**" Gradient

$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{sigmoid}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

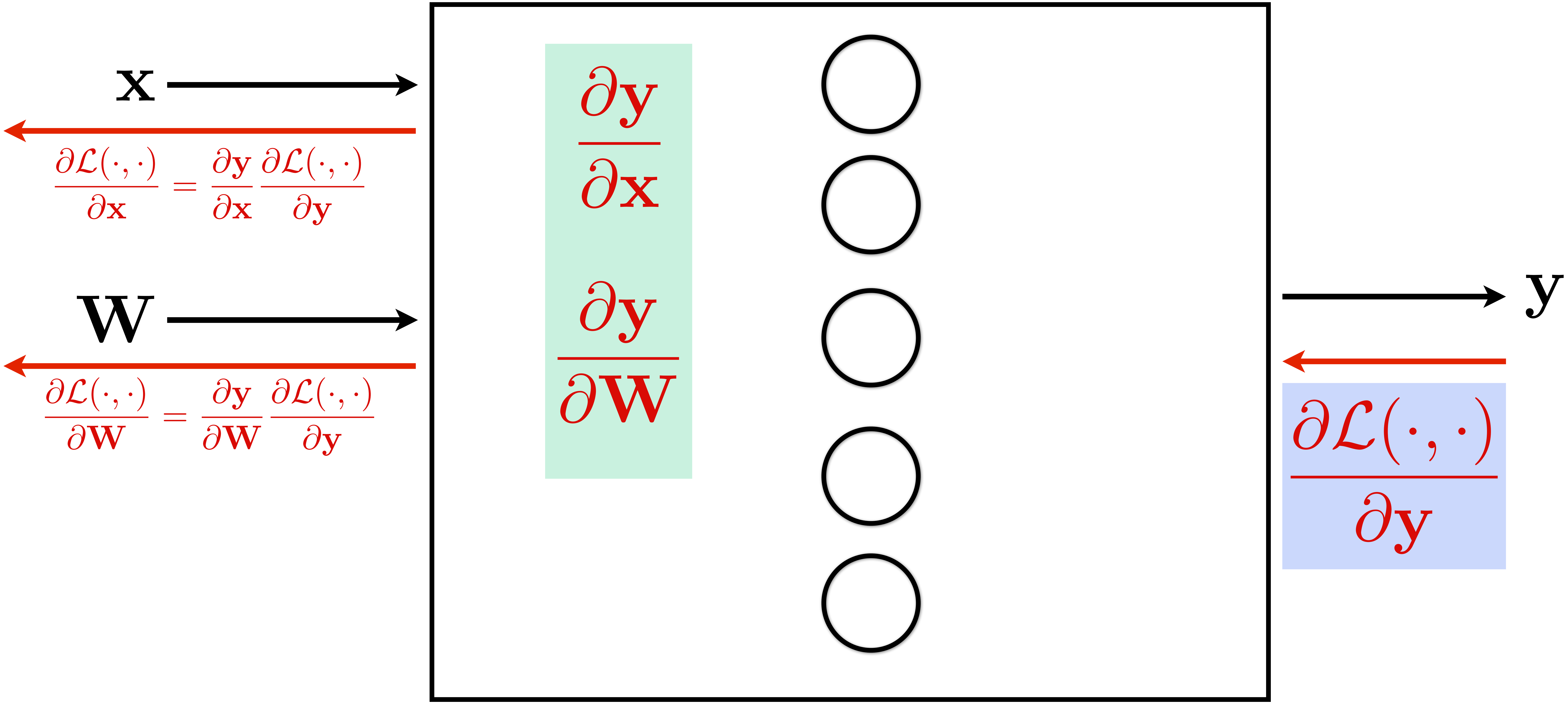


Backpropagation Practical Issues

"**local**" Jacobians
(matrix of partial derivatives, e.g. size |x| x |y|)

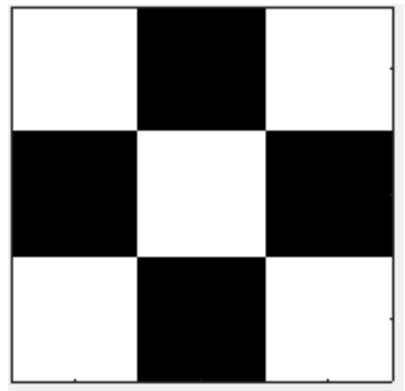
"**backprop**" Gradient

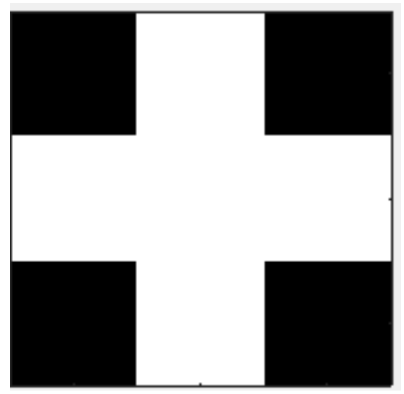
$$\mathbf{y} = f(\mathbf{W}, \mathbf{b}, \mathbf{x}) = \mathbf{W} \cdot \mathbf{x}$$

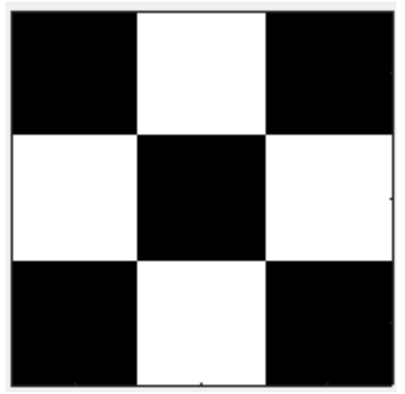


Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

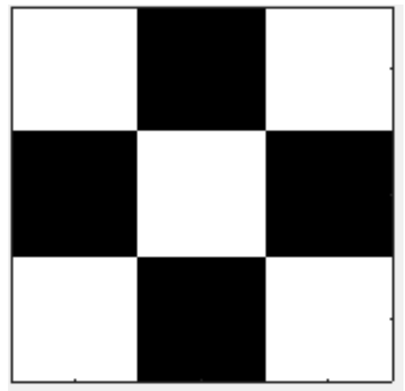


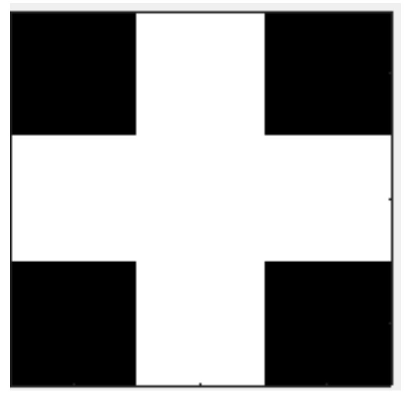


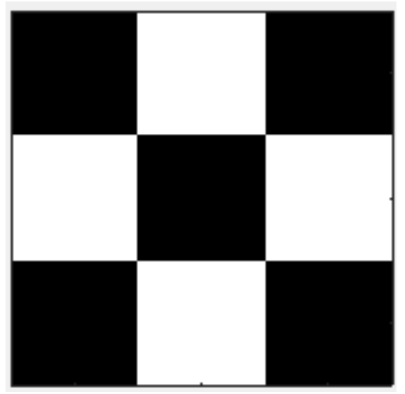


Example: Let's Build (world smallest) Neural Network

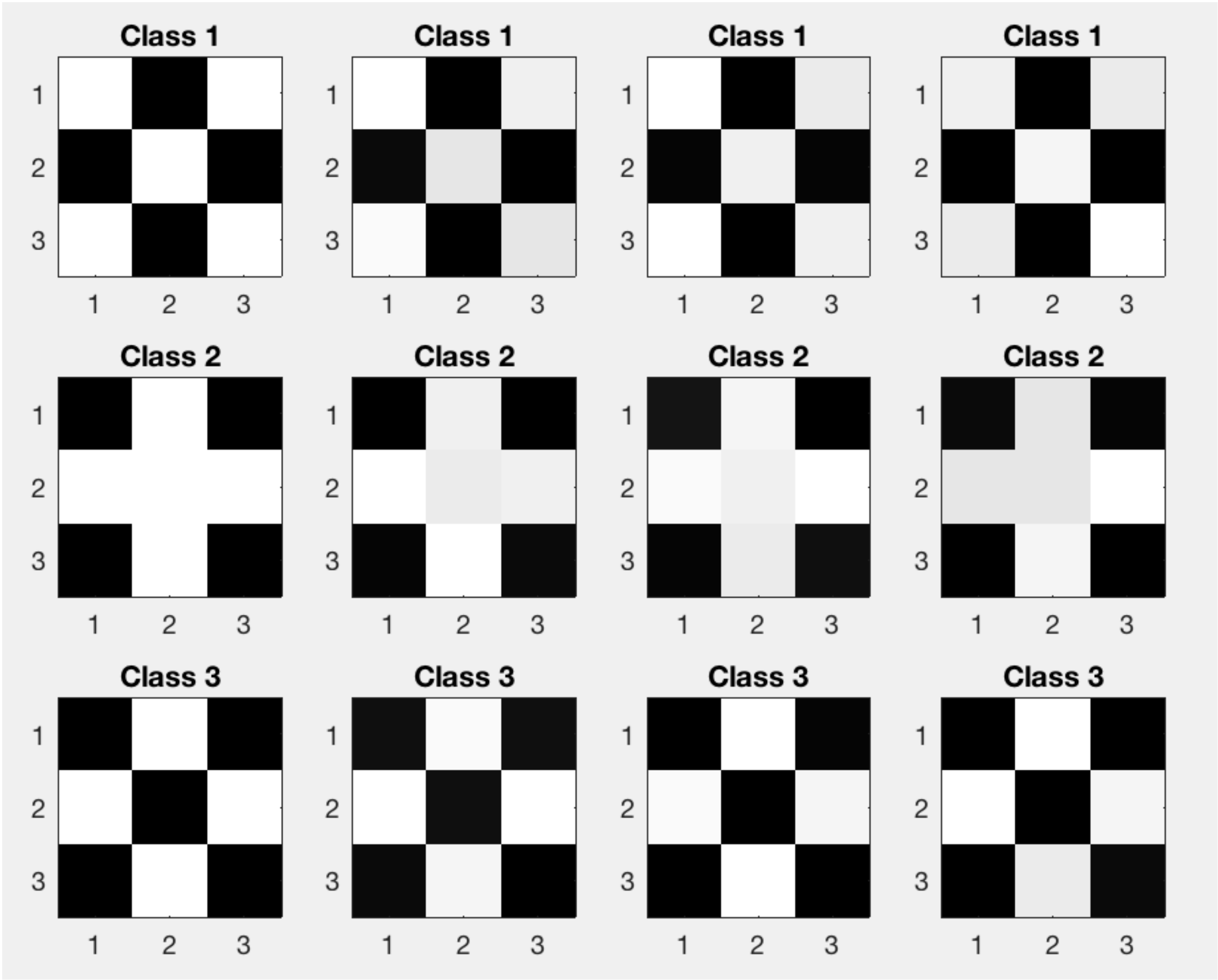
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images





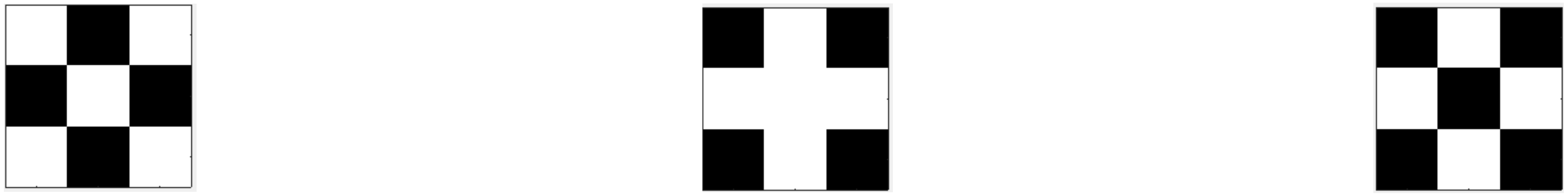


We will need some labeled data



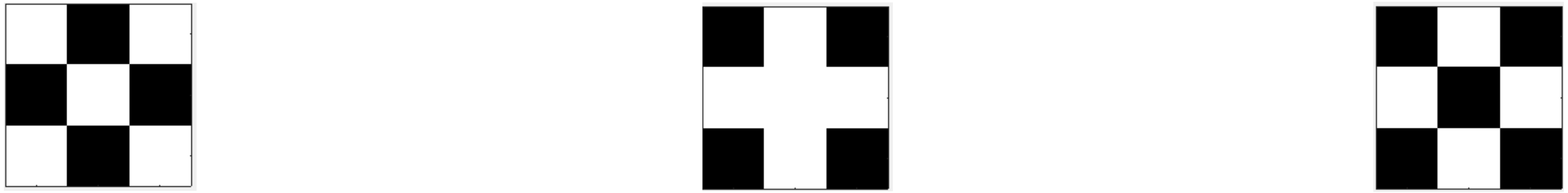
Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



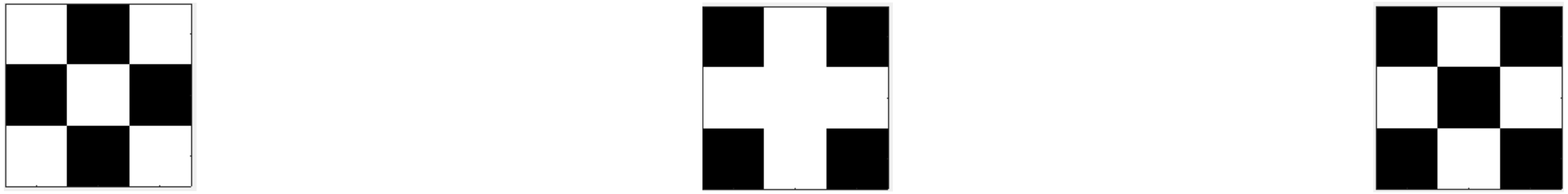
Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



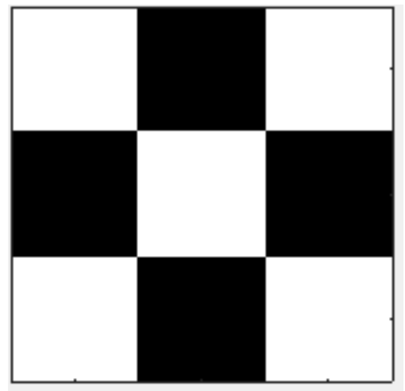
Example: Let’s Build (world smallest) Neural Network

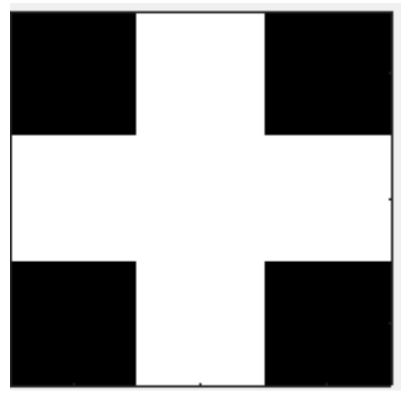
Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

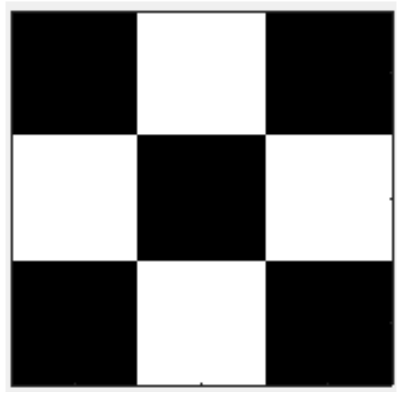


Example: Let’s Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images





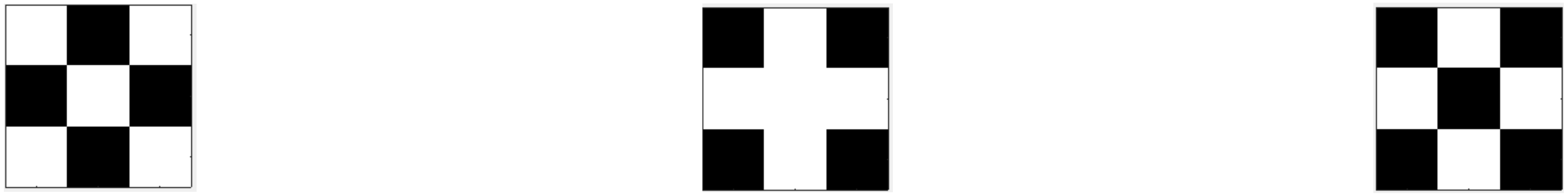


What do we need to do?

First, lets re-formulate the problem

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



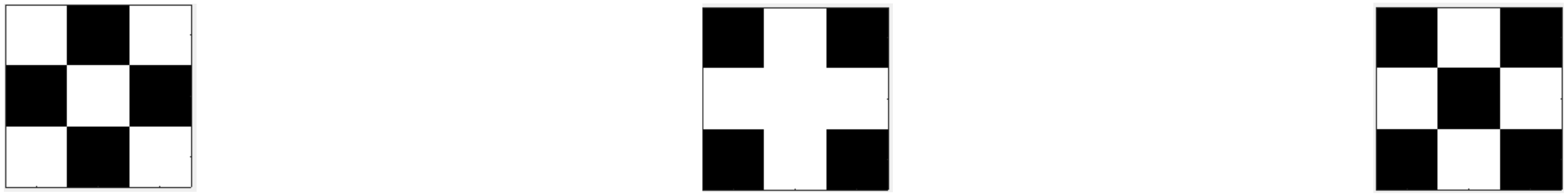
What do we need to do?

p(Class **1**)
p(Class **2**)
p(Class **3**)

First, lets re-formulate the problem

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

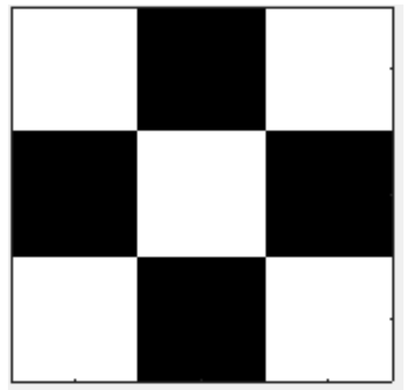
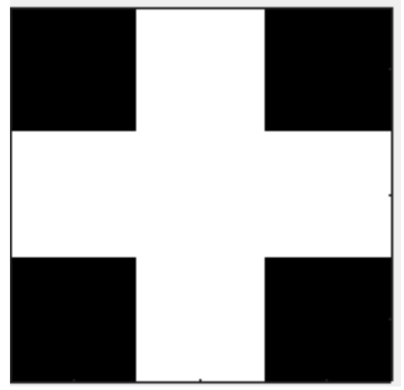
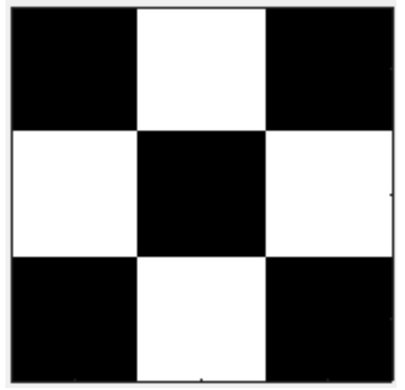


Now, lets build a **network**!

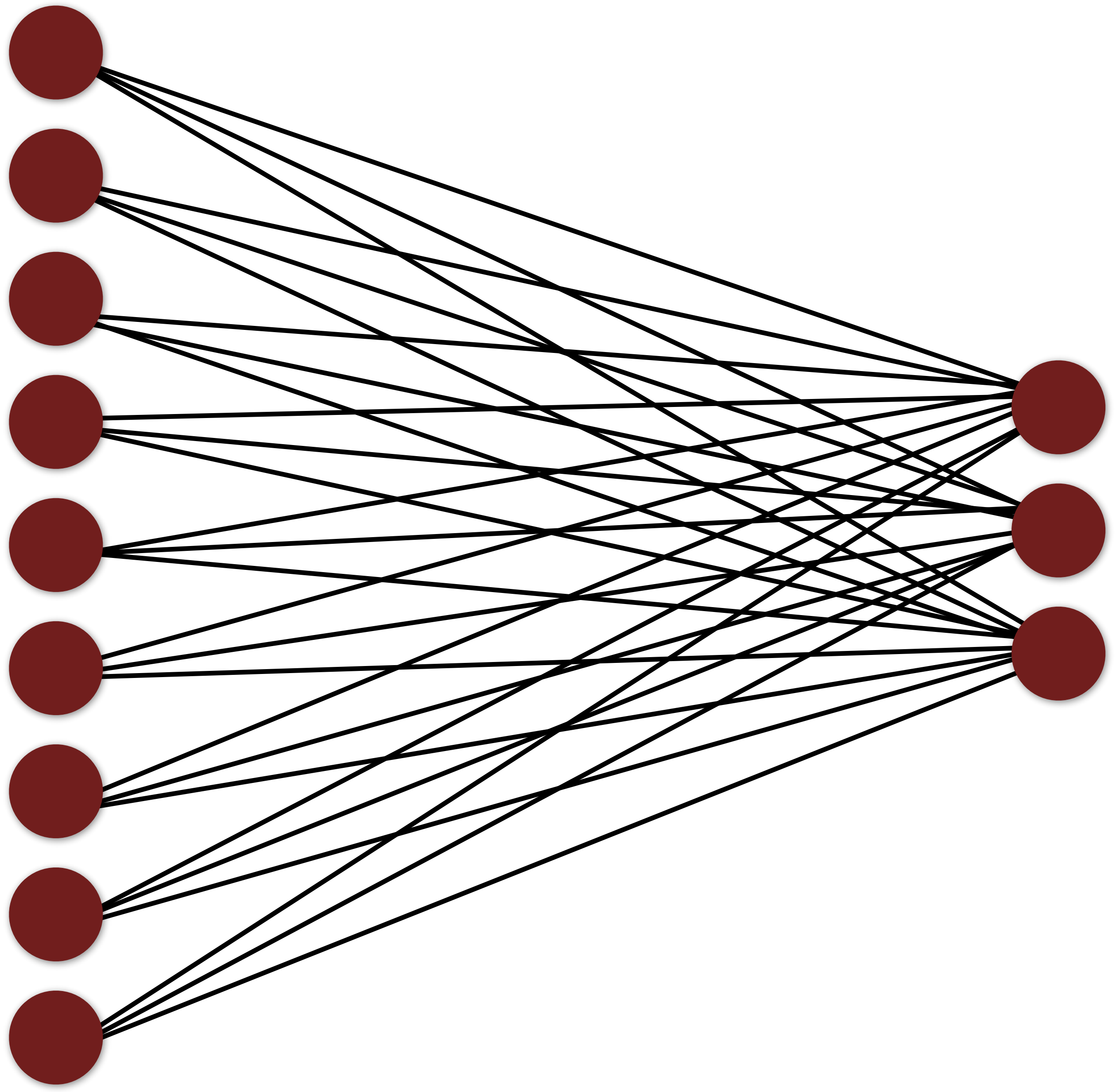
How many inputs should the network have? How neuron outputs?

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images

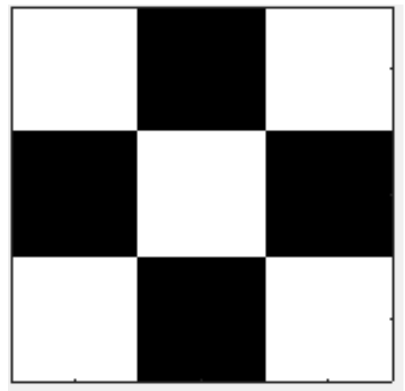
Input Layer **Output** Layer

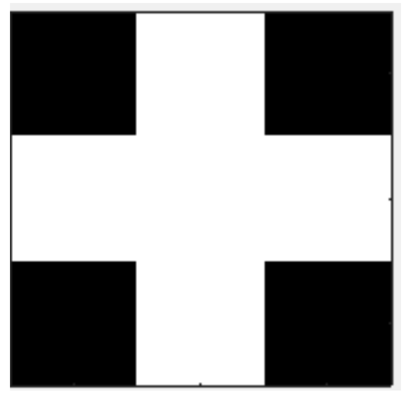


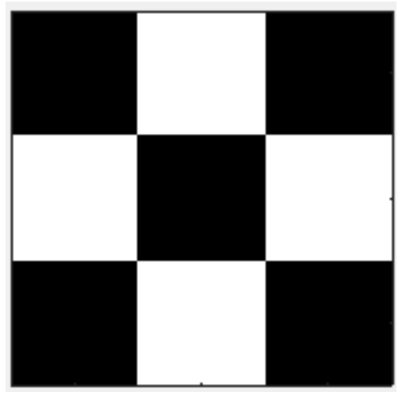
What else is missing for us to train it?

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



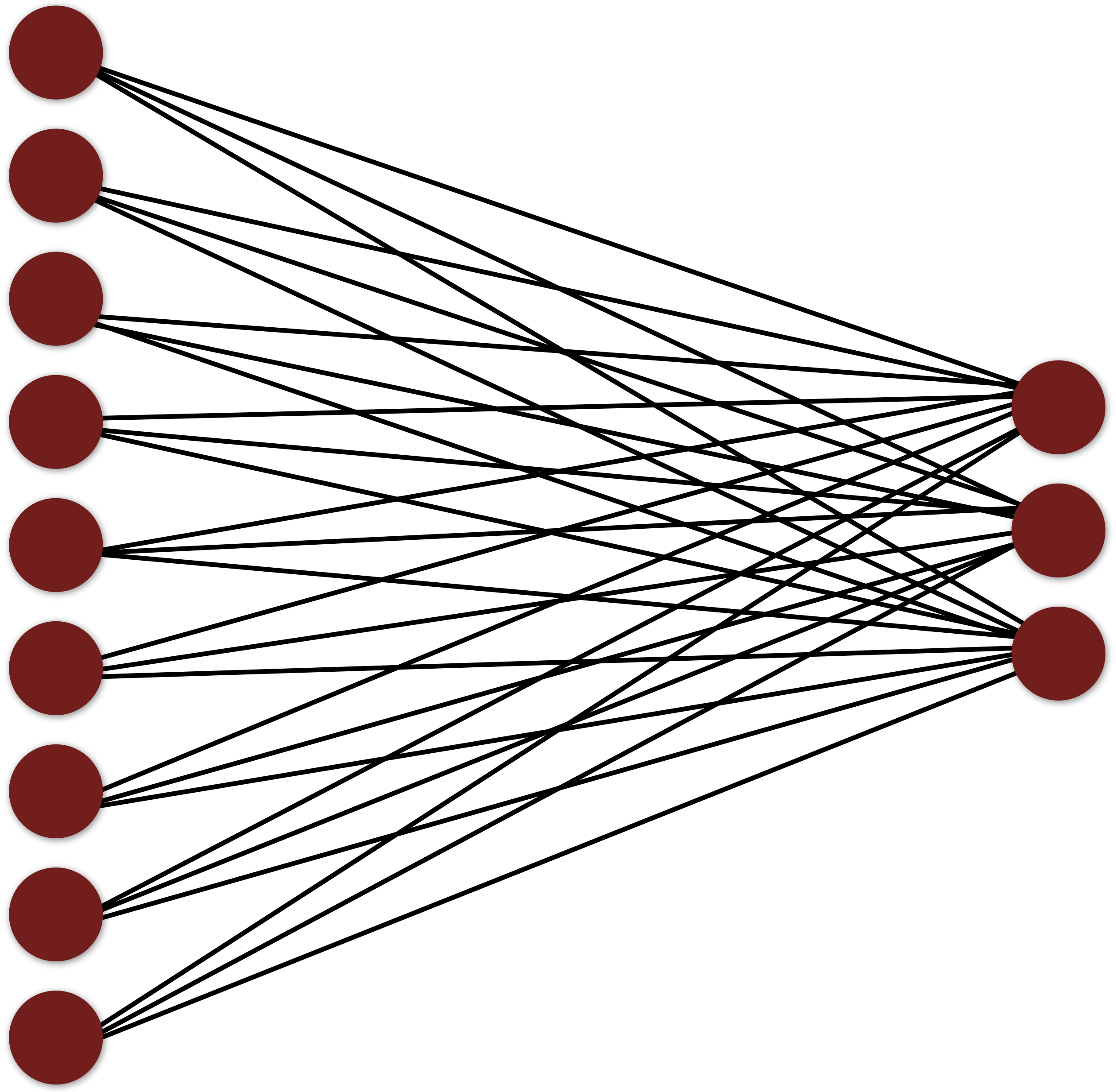




Input Layer

Output Layer

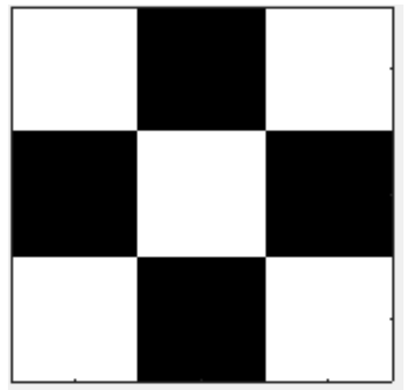
Loss

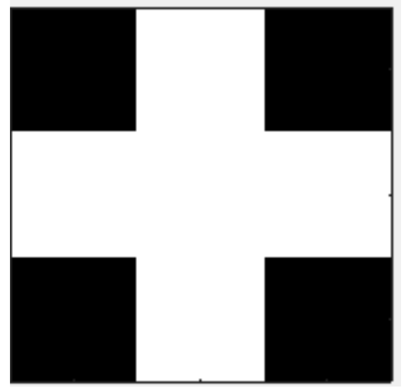


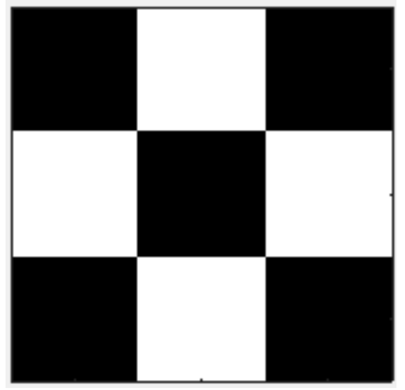
$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}\right)$$

Example: Let's Build (world smallest) Neural Network

Lets create a neural network that will be able to differentiate (classify) these patterns of simple 3x3 pixel images



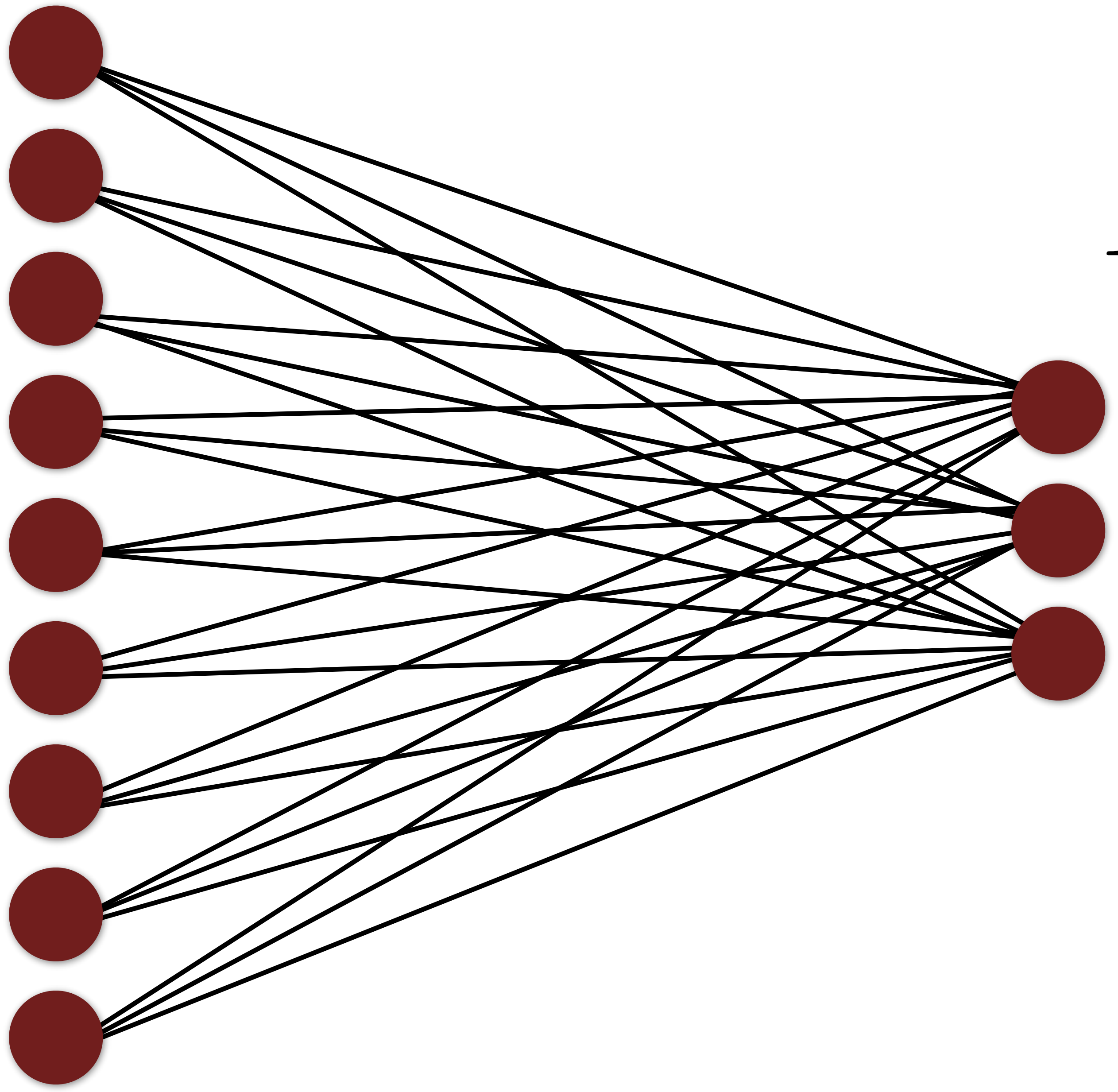




Input Layer

Output Layer

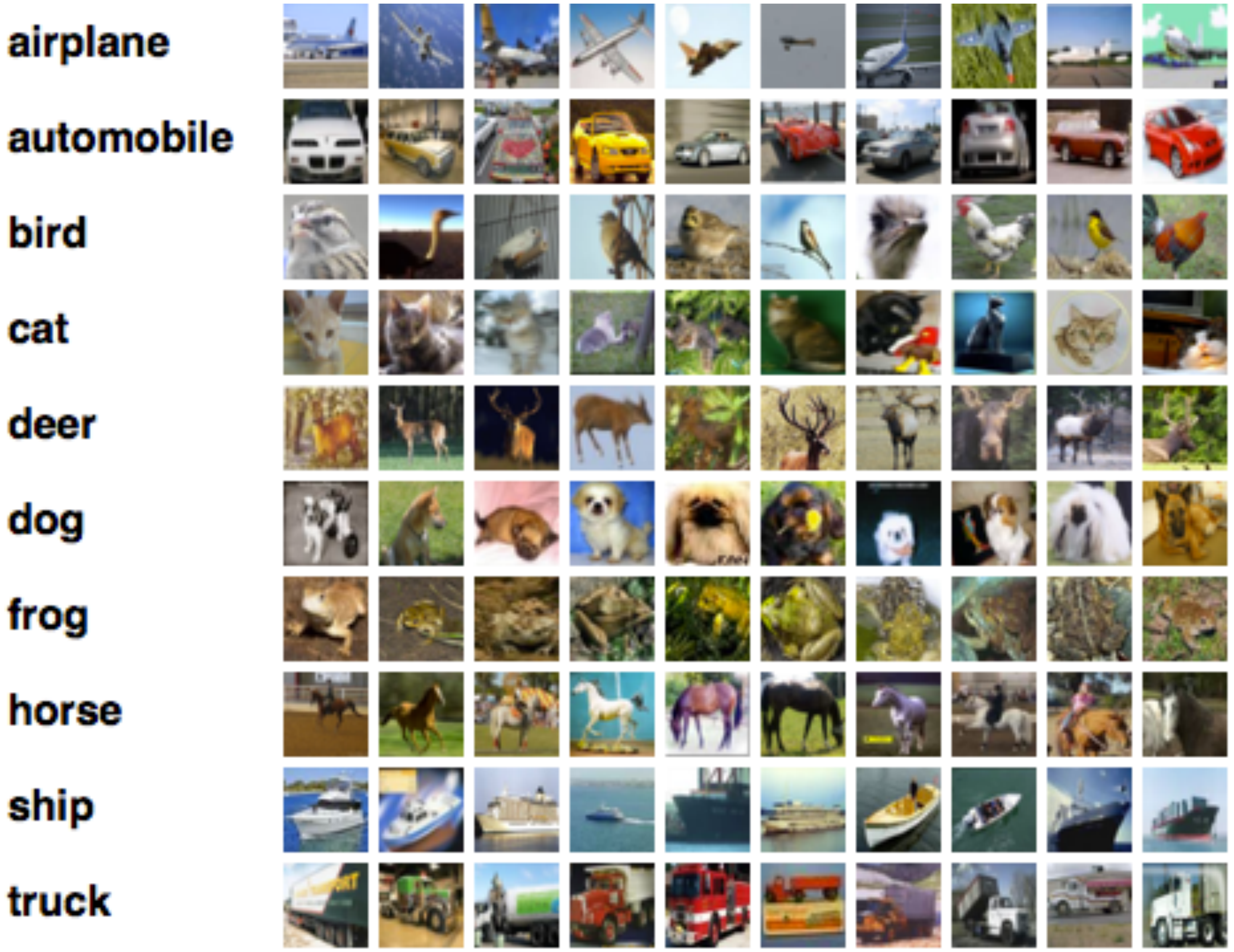
Loss



$$L_1 = -log\left(\frac{e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}{\sum_{j=1}^{3} e^{\sum_{i=1}^{9}\sigma(w_{1,i}x_i+b_1)}}\right)$$

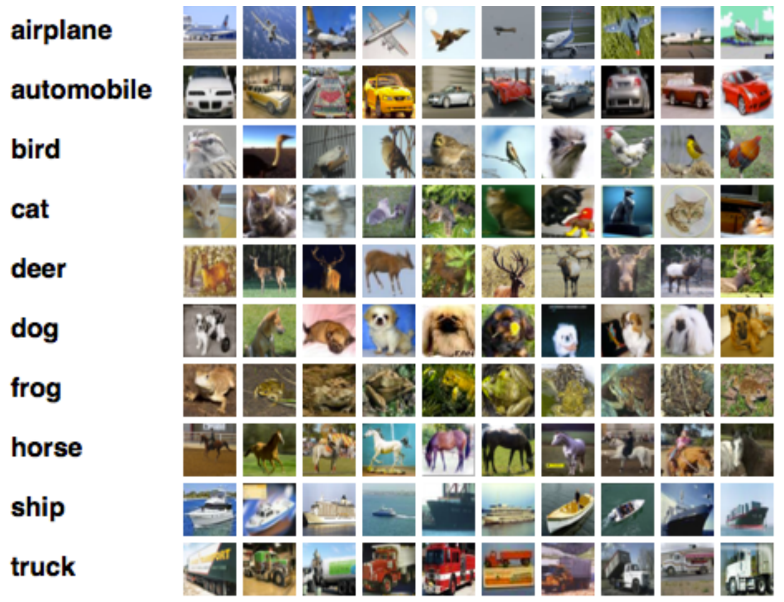
CIFAR10 Dataset

- Hand labelled set of 10 categories from Tiny Images dataset
- 60,000 32x32 images in 10 classes (50k train, 10k test)

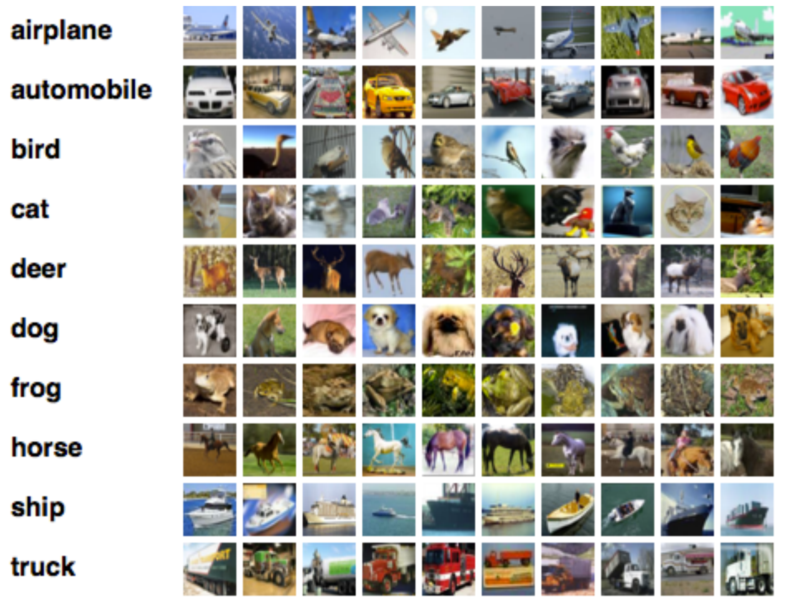


Good test set for visual recognition problems

Neural Network: Short Review



Neural Network: Short Review

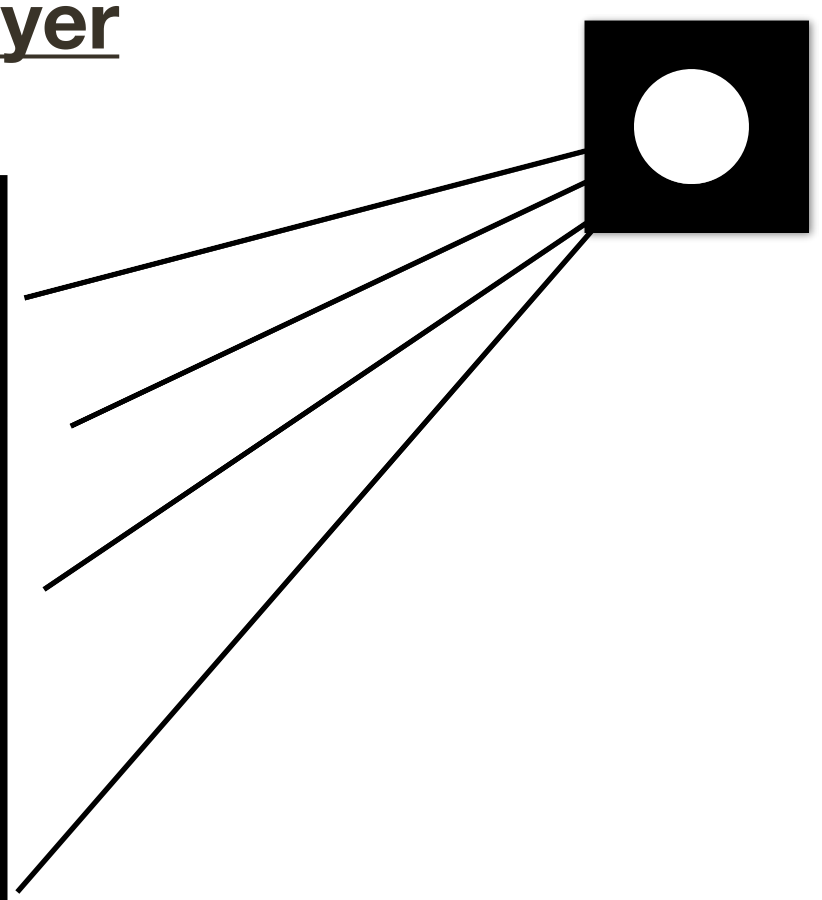


$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

Input Layer

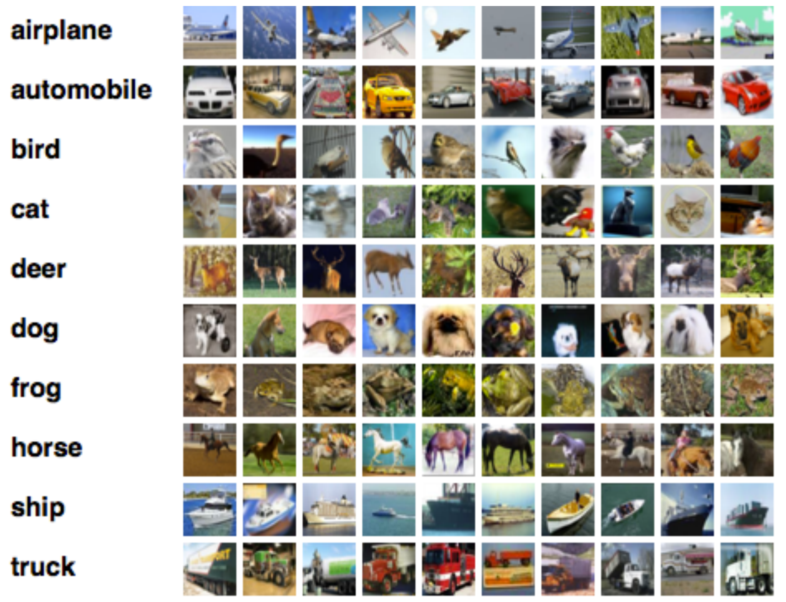
Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$



Vectorized Input
(32 x 32 x 3) = 3072

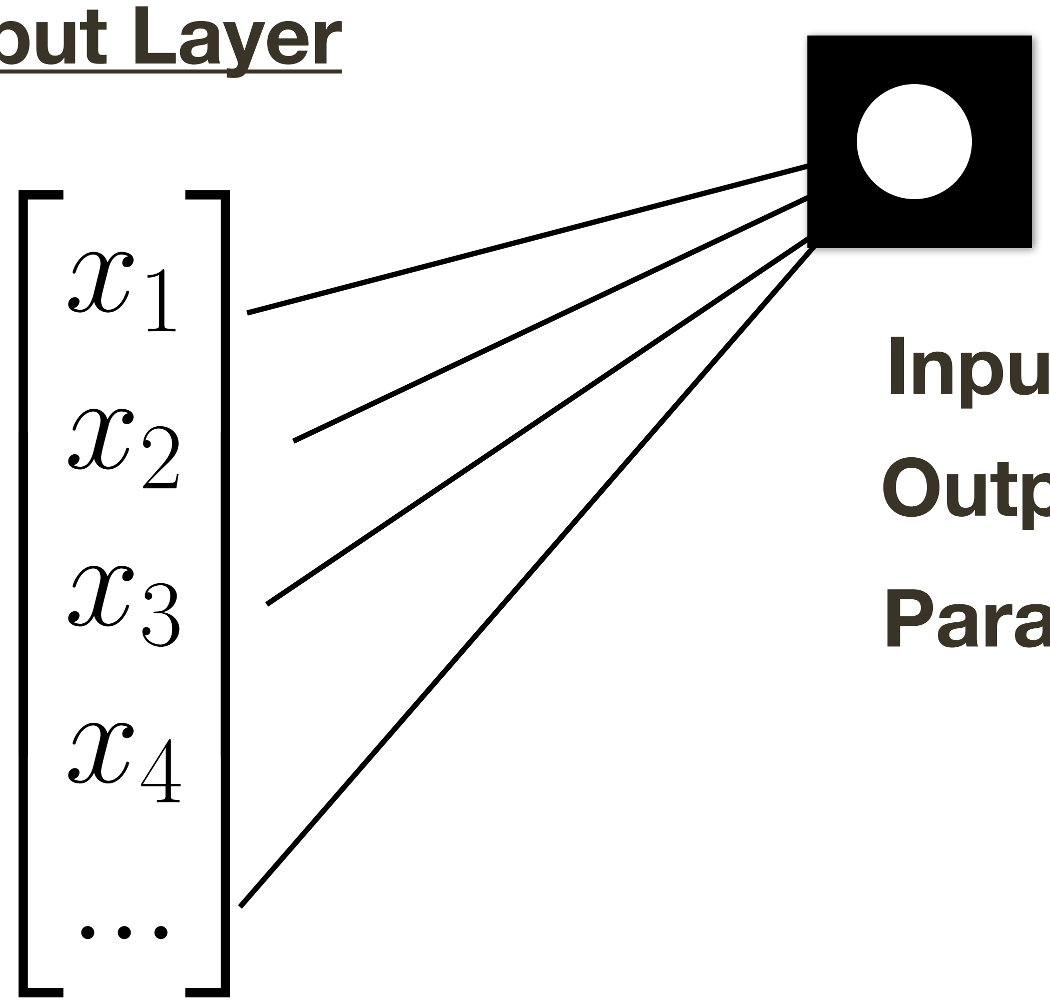
Neural Network: Short Review



$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

Input Layer

Input Image



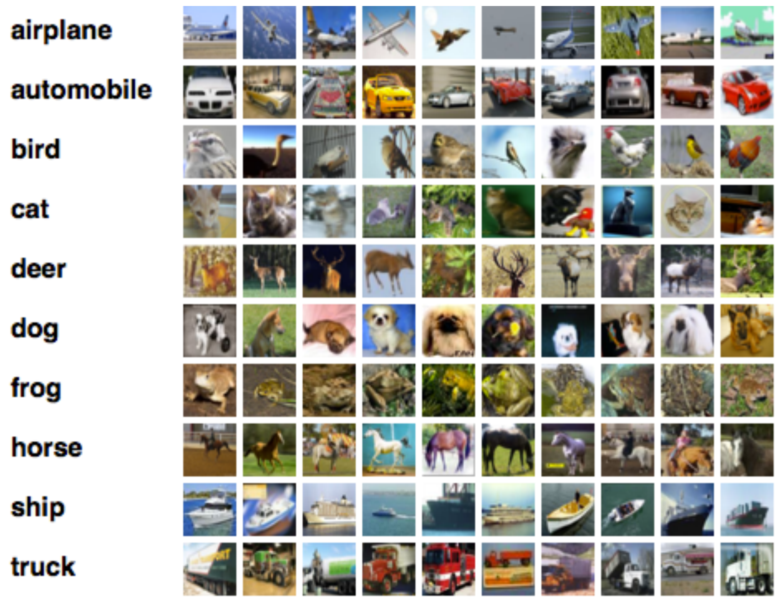
Inputs: 3072

Outputs: 1

Parameters: 3072 + 1

Vectorized Input
(32 x 32 x 3) = 3072

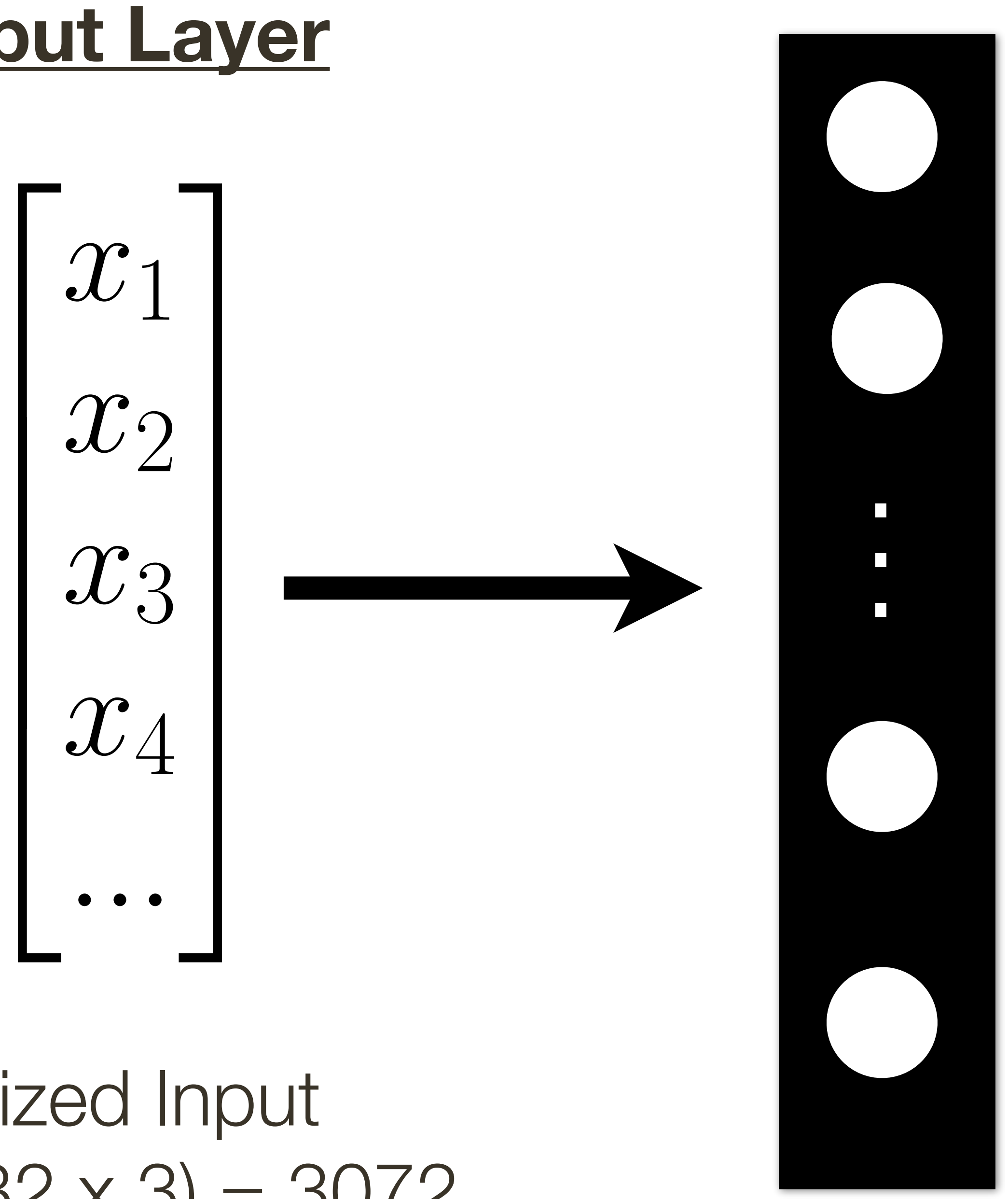
Neural Network: Short Review



Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

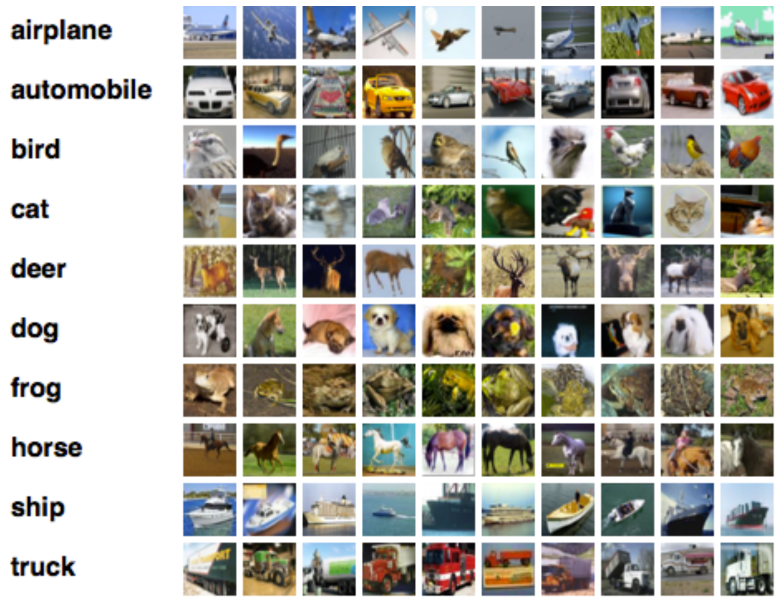
Input Image



Vectorized Input
(32 x 32 x 3) = 3072

$\mathbf{W}_1, \mathbf{b}_1$

Neural Network: Short Review

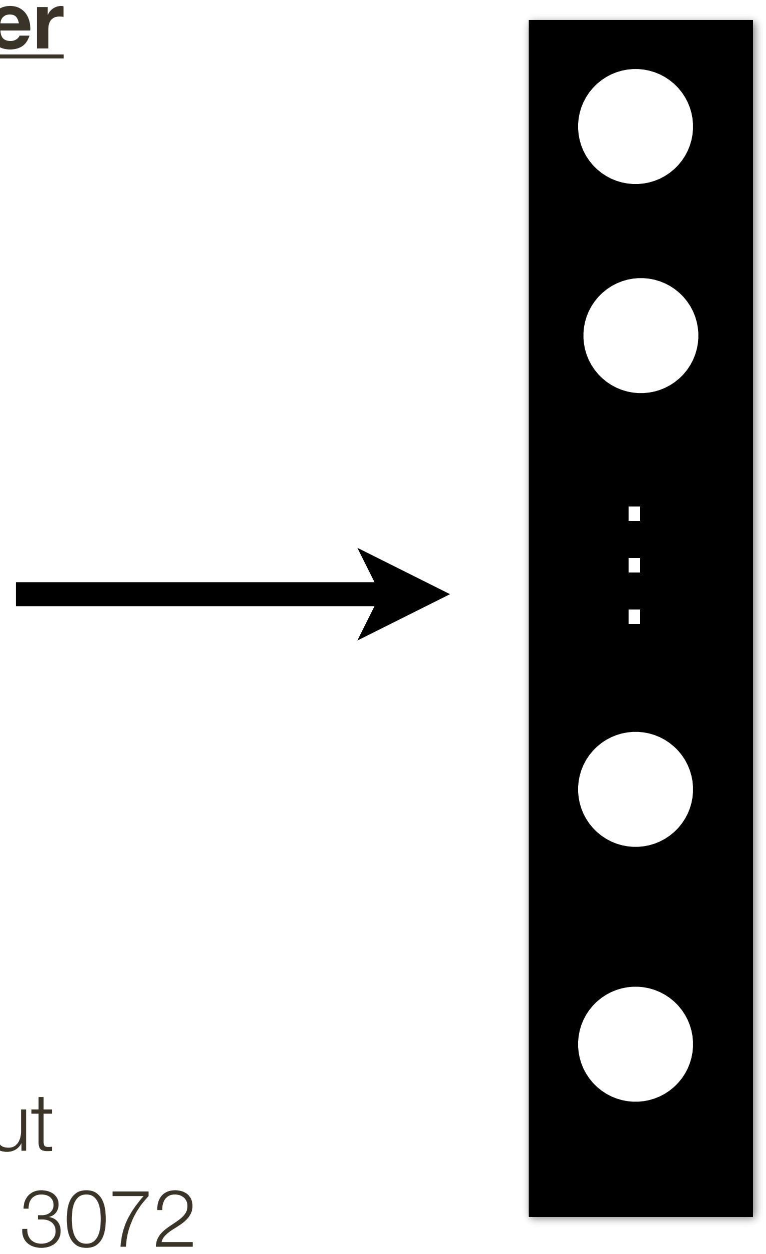


Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix}$$



$\mathbf{W}_1, \mathbf{b}_1$

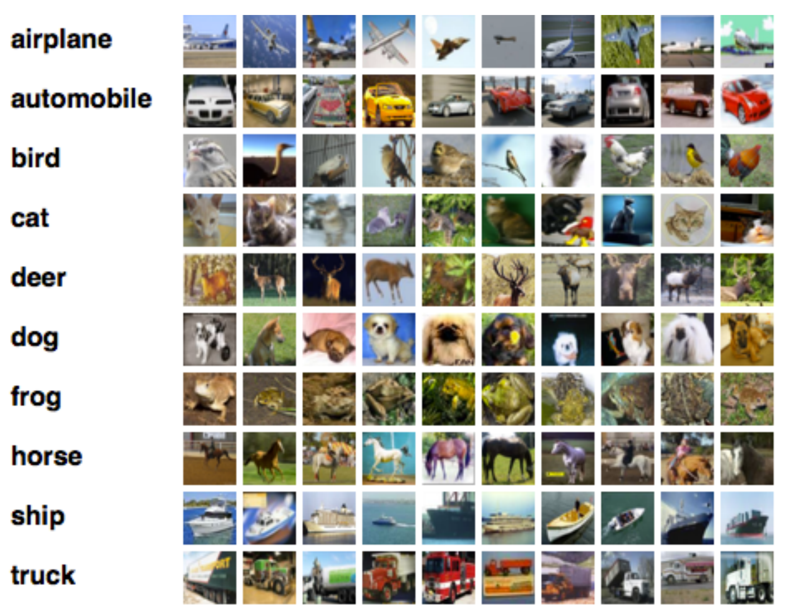
Vectorized Input
(32 x 32 x 3) = 3072

Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

Neural Network: Short Review

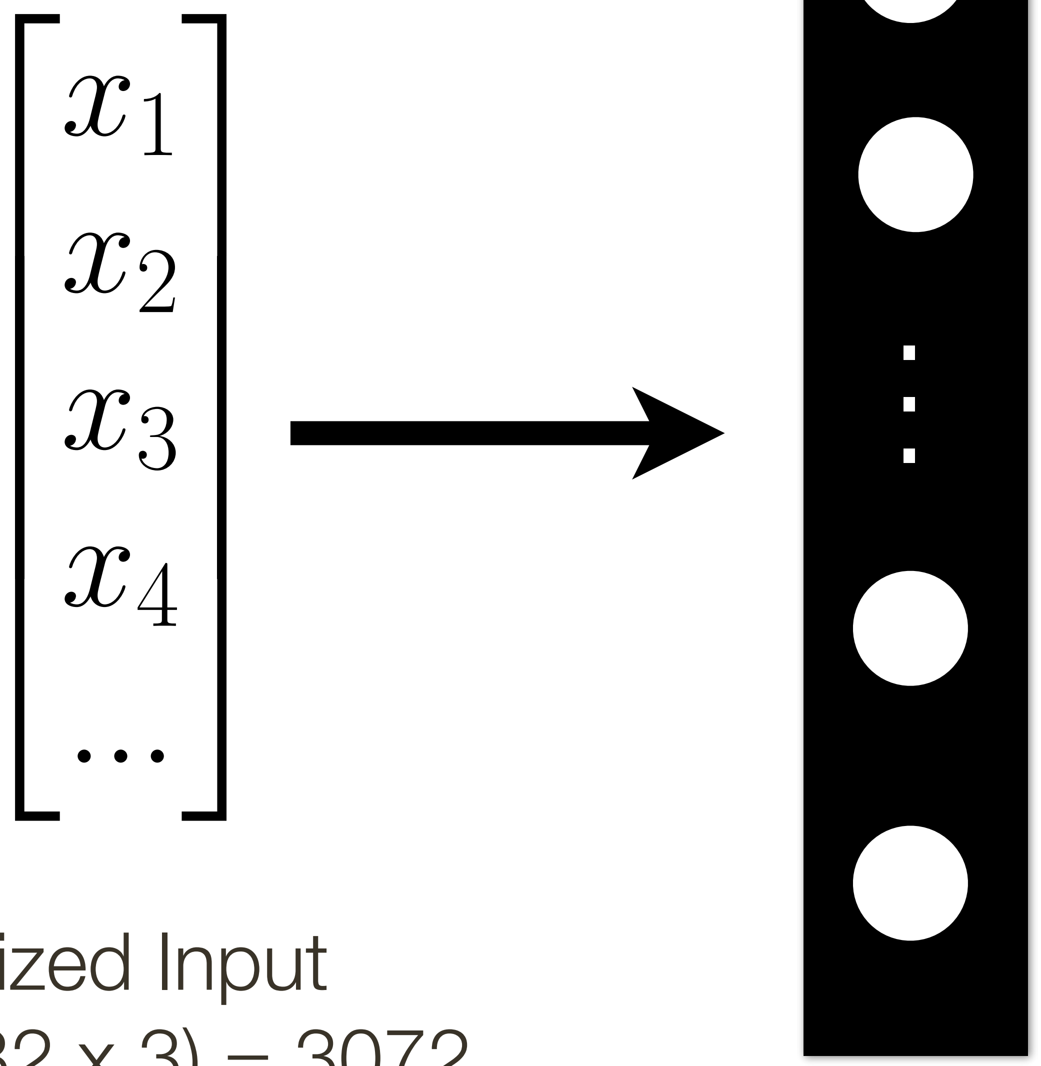


Hidden Layer 1

* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

Input Image



$\mathbf{W}_1, \mathbf{b}_1$

Vectorized Input
(32 x 32 x 3) = 3072

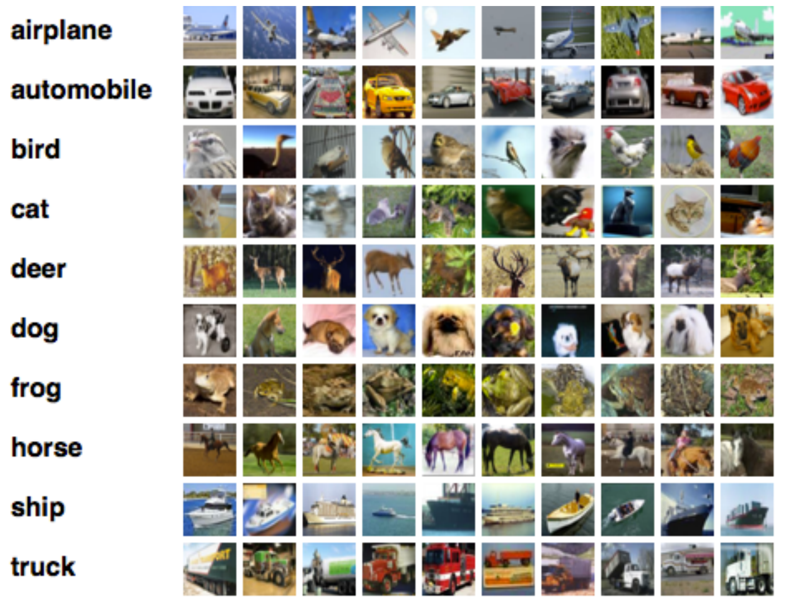
Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

Note: All neurons within a layer can be computed in parallel, making computations very efficient (especially on GPUs!, which are designed for parallelism)

Neural Network: Short Review

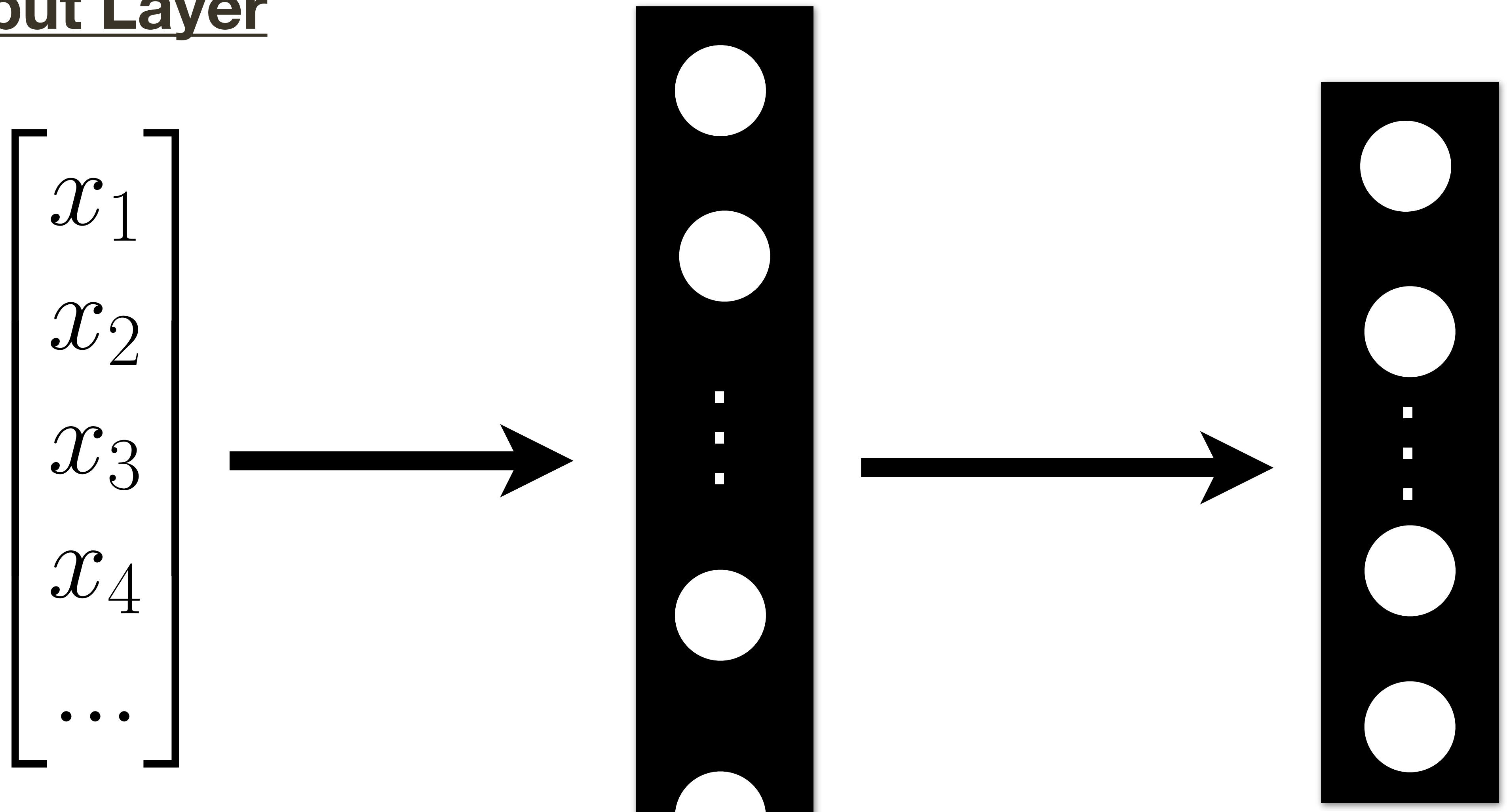


Input Image

Input Layer

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ



$\mathbf{W}_1, \mathbf{b}_1$

$\mathbf{W}_2, \mathbf{b}_2$

Vectorized Input
(32 x 32 x 3) = 3072

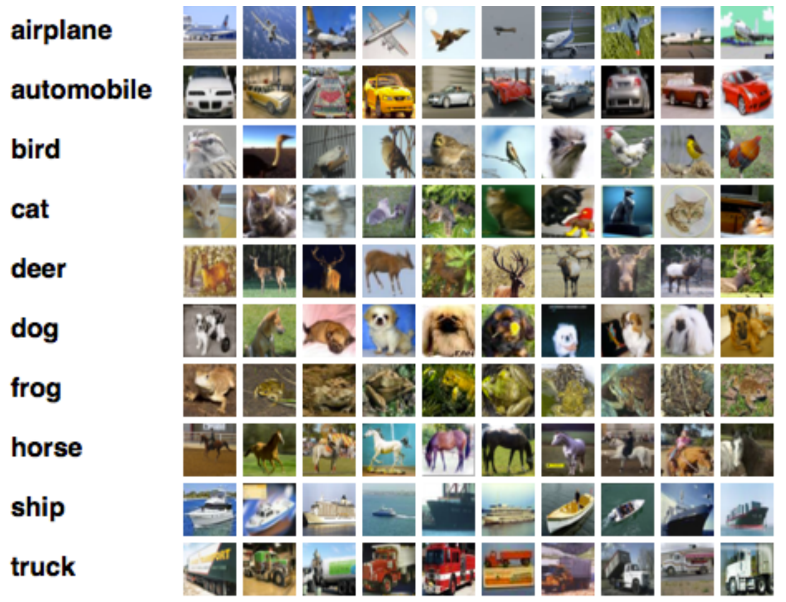
Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

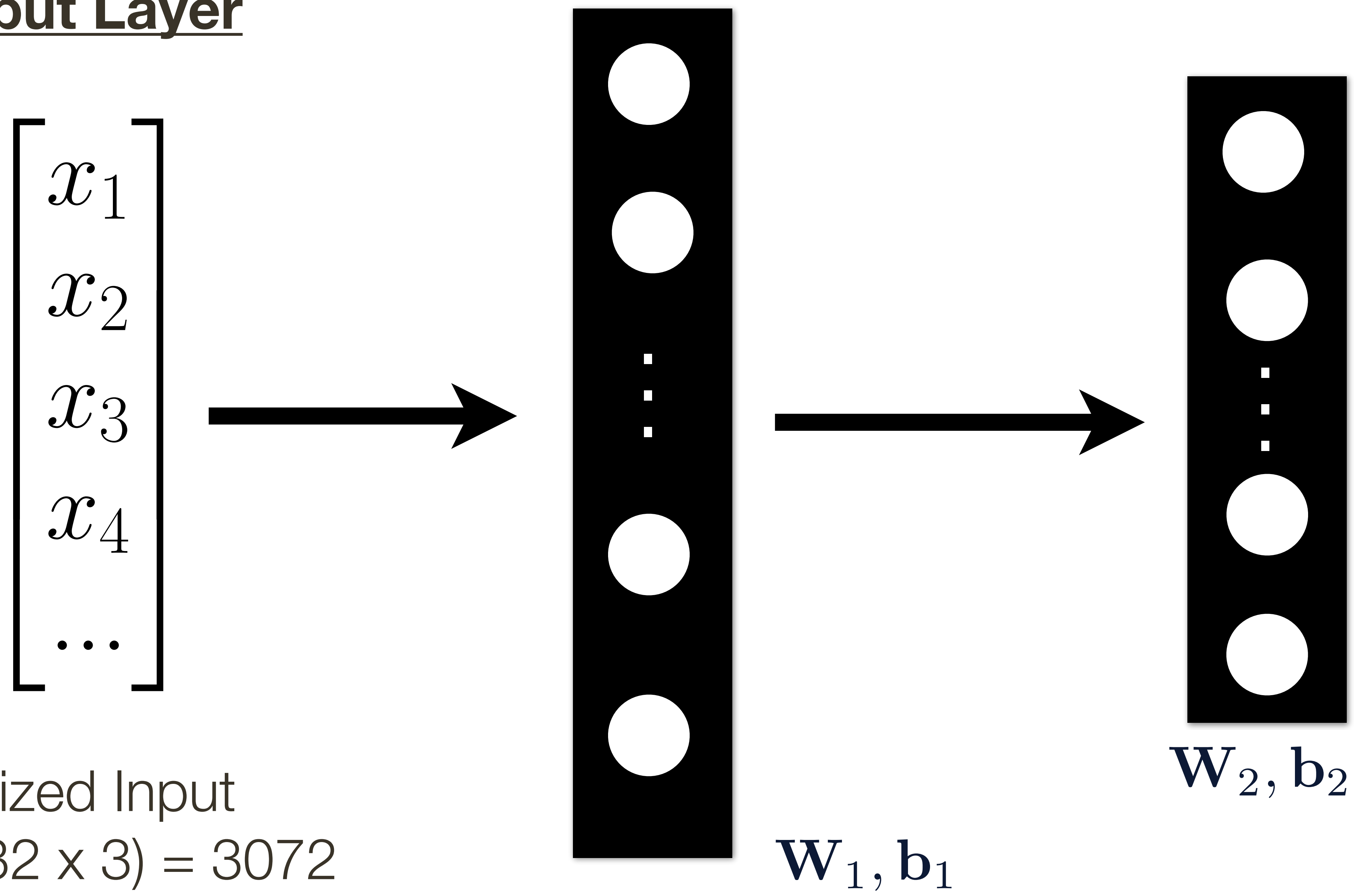
Note: Across layers computations are sequential

Neural Network: Short Review



Input Layer

Input Image



Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

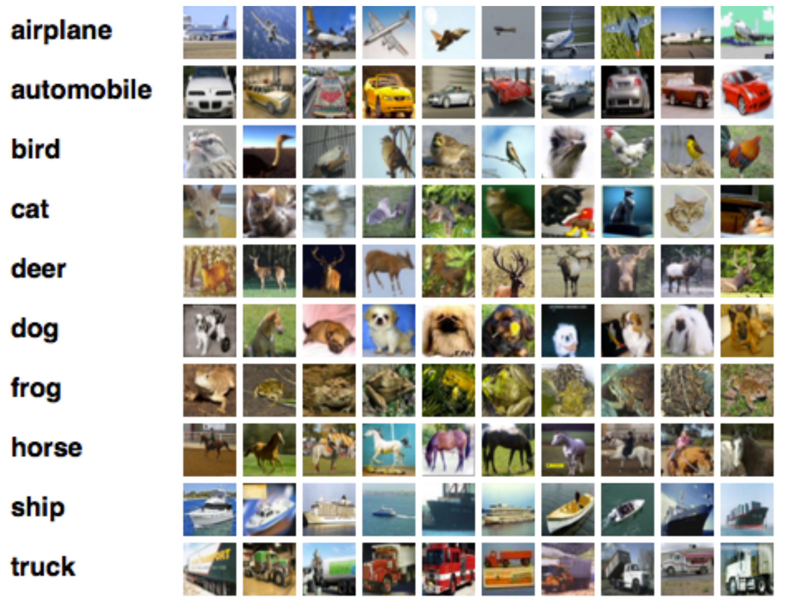
Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

Inputs: 400

Outputs: 100

Parameters:
400 x 100 + 100

Neural Network: Short Review



Input Image

Input Layer

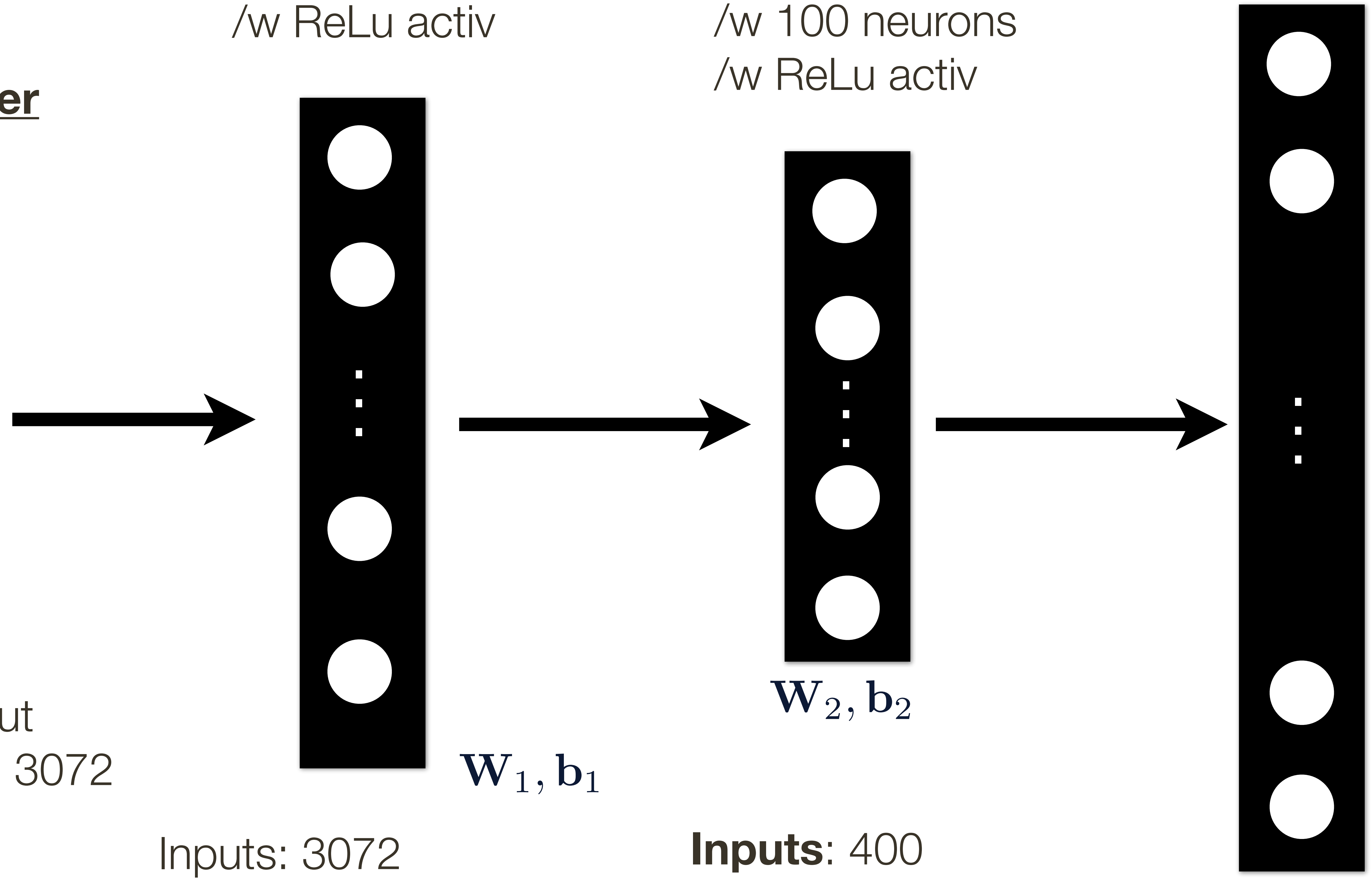
$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \ldots \end{bmatrix}$$

Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ



$\mathbf{W}_1, \mathbf{b}_1$

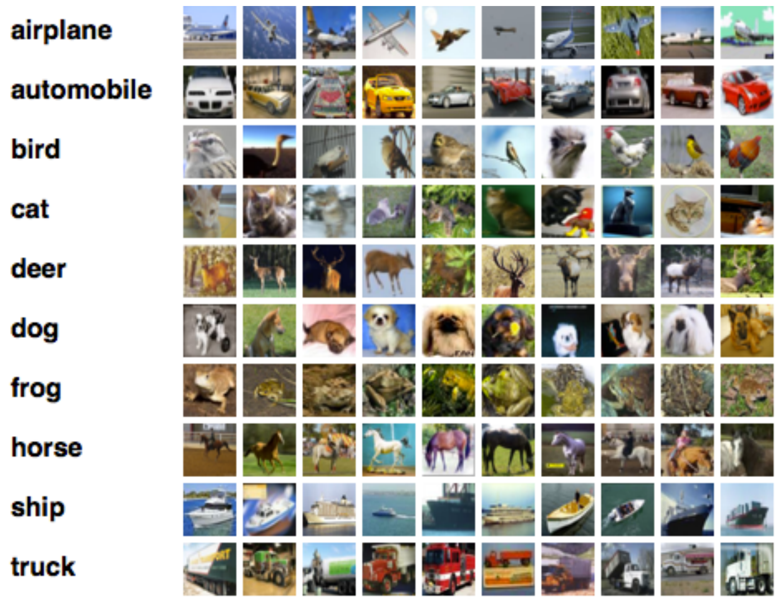
$\mathbf{W}_2, \mathbf{b}_2$

$\mathbf{W}_3, \mathbf{b}_3$

Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Neural Network: Short Review



Input Layer

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$$

Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

$\mathbf{W}_1, \mathbf{b}_1$

Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

$\mathbf{W}_2, \mathbf{b}_2$

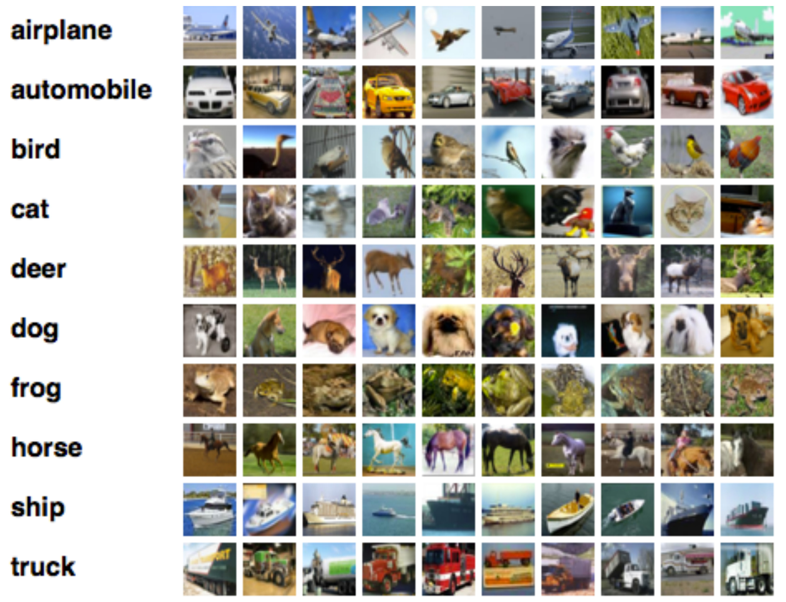
Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ

$\mathbf{W}_3, \mathbf{b}_3$

Inputs: 100
Outputs: 700
Parameters:
100 x 700 + 700

Neural Network: Short Review



Input Image

Input Layer

$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix}$

Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

$\mathbf{W}_1, \mathbf{b}_1$

Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

$\mathbf{W}_2, \mathbf{b}_2$

Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ

$\mathbf{W}_3, \mathbf{b}_3$

Inputs: 100
Outputs: 700
Parameters:
100 x 700 + 700

Output Layer
* Fully Connected
/w 10 neurons
/w ReLu activ

$\mathbf{W}_4, \mathbf{b}_4$

Inputs: 700
Outputs: 10
Parameters:
700 x 10 + 10

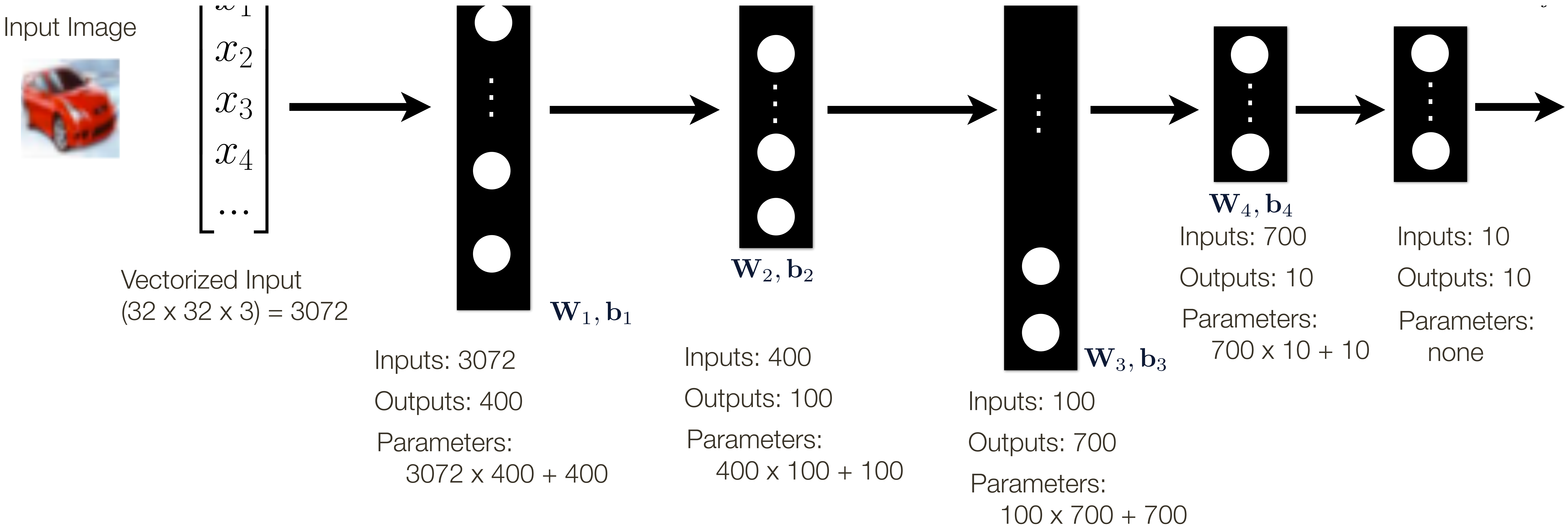
+ sigmoid

$$\hat{y}_i = \frac{e^{f_{y_i}}}{\sum_j e^{f_{y_j}}}$$

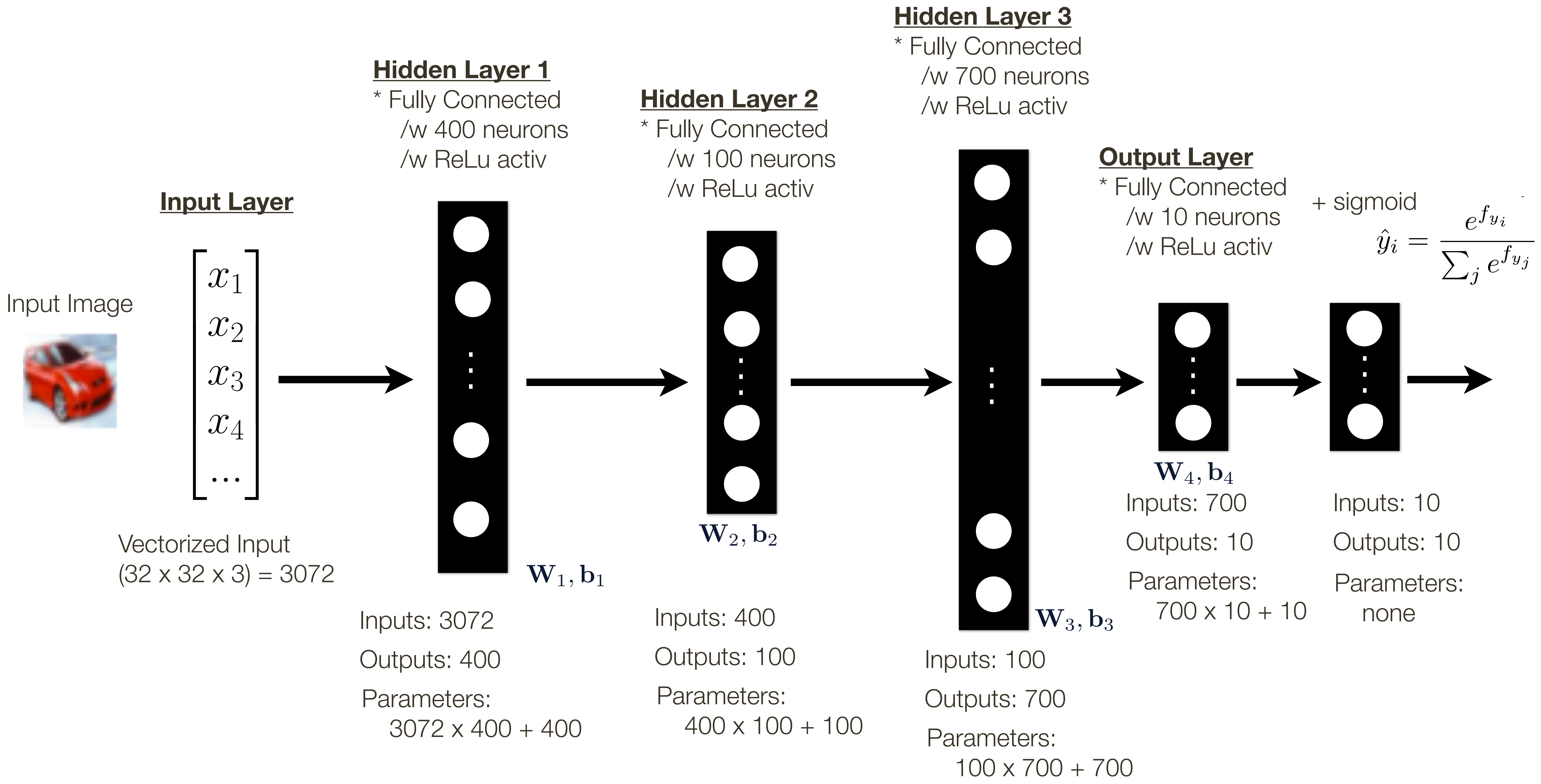
Inputs: 10
Outputs: 10
Parameters:
none

Neural Network: Short Review

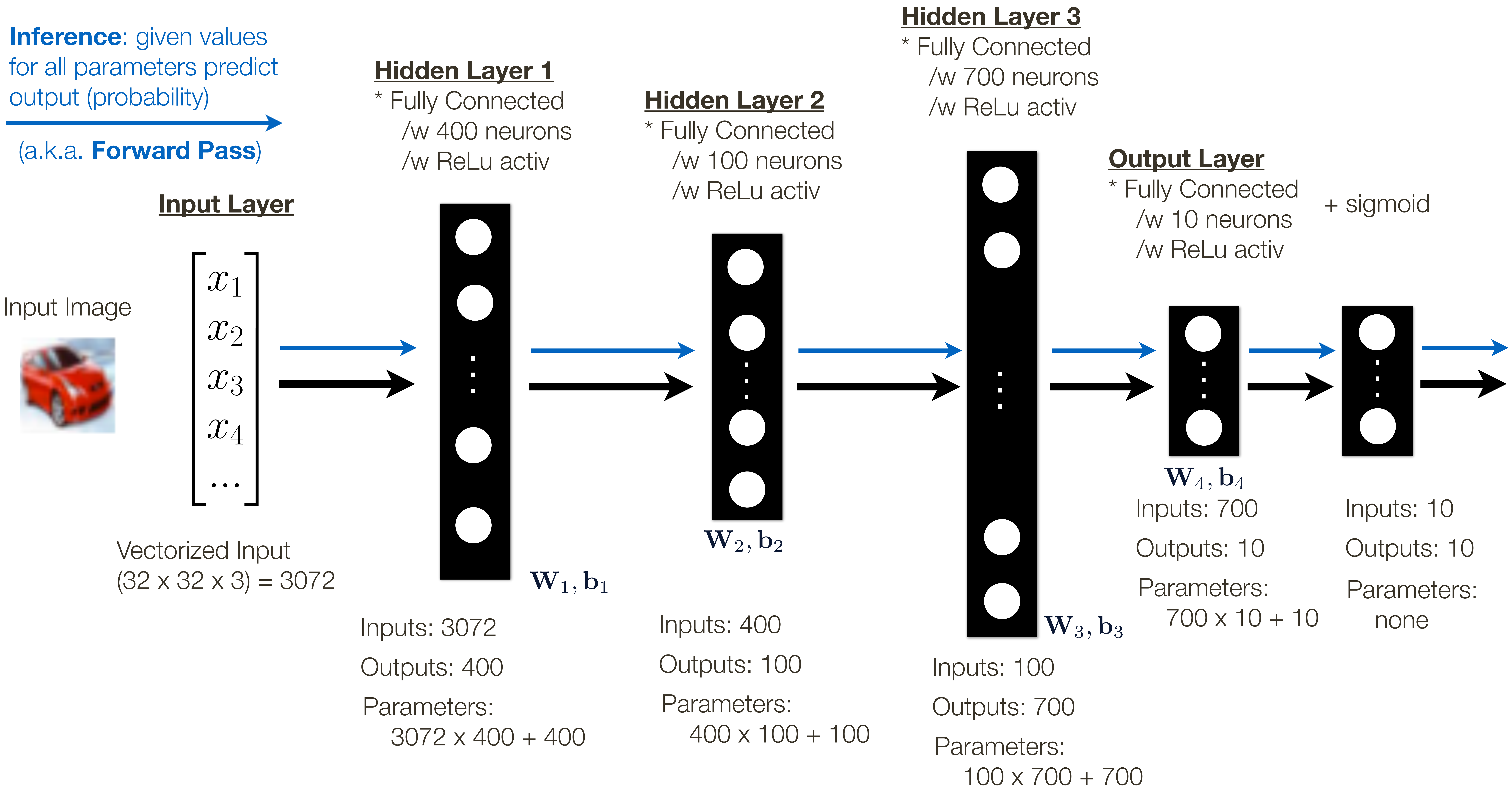
This simple neural network has nearly 1.35 million parameters



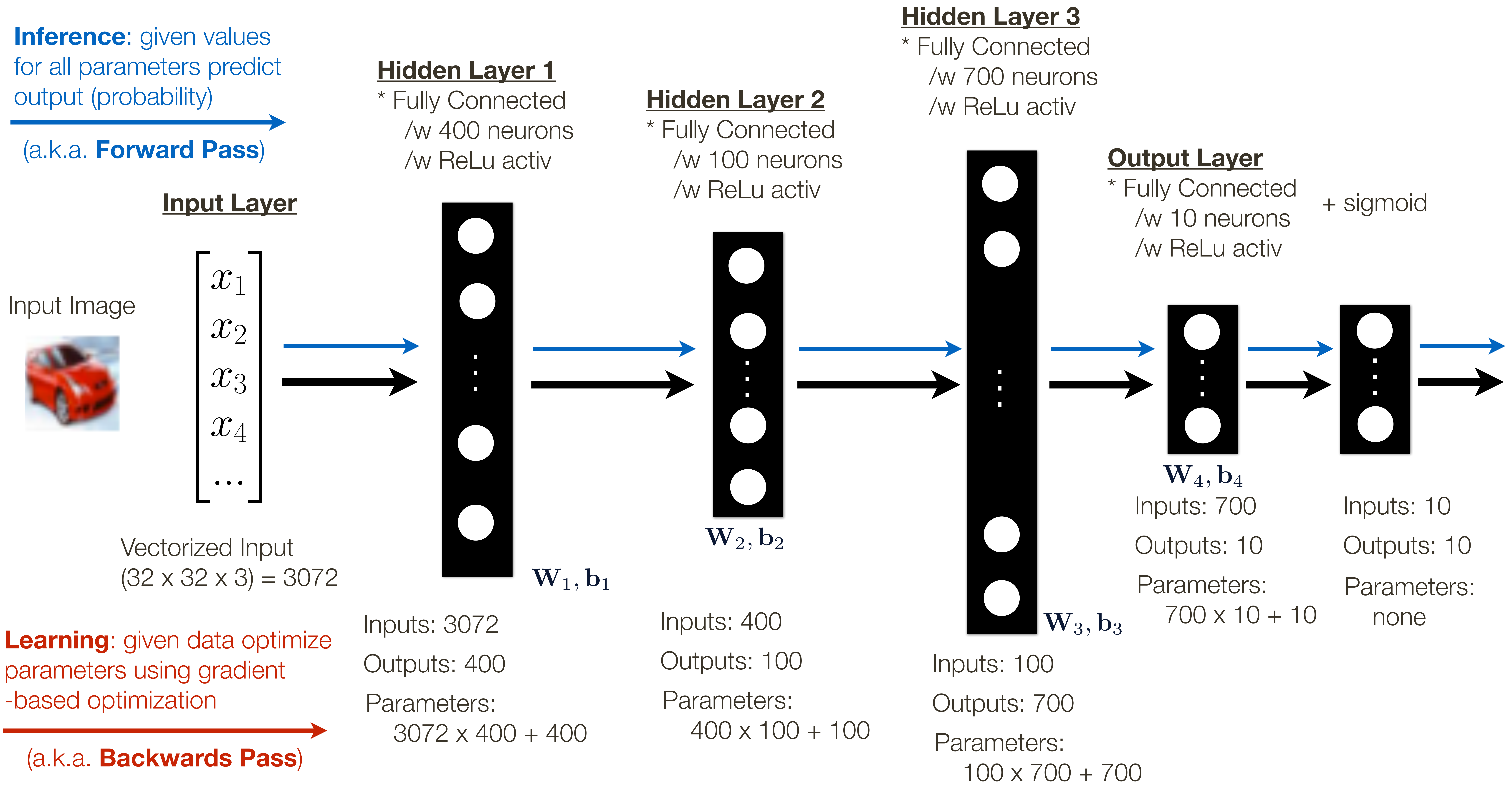
Neural Network: Short Review



Neural Network: Short Review



Neural Network: Short Review



Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

Learning: given data optimize parameters using gradient -based optimization

(a.k.a. **Backwards Pass**)

Input Image

Input Layer

Vectorized Input
(32 x 32 x 3) = 3072

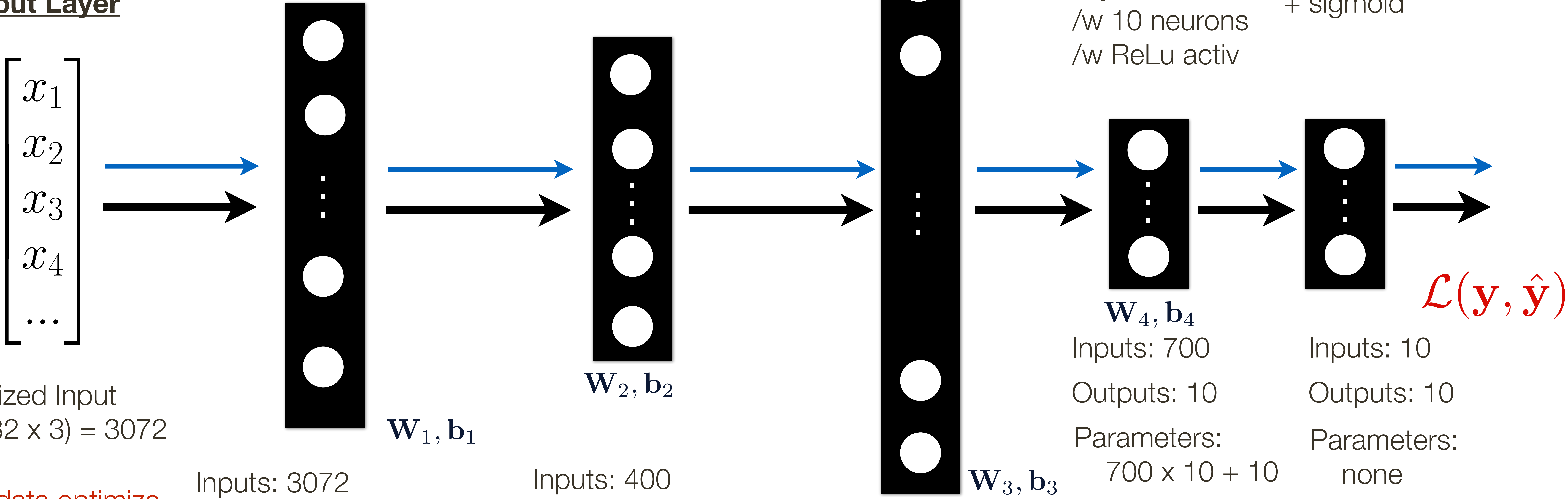
Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ

Output Layer
* Fully Connected
/w 10 neurons
/w ReLu activ

+ sigmoid



Hidden Layer 1:
Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Hidden Layer 2:
Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Hidden Layer 3:
Inputs: 100
Outputs: 700
Parameters:
100 x 700 + 700

Output Layer:
Inputs: 700
Outputs: 10
Parameters:
700 x 10 + 10

Sigmoid:
Inputs: 10
Outputs: 10
Parameters:
none

Neural Network: Short Review

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

Learning: given data optimize parameters using gradient-based optimization

(a.k.a. **Backwards Pass**)

Input Image

Input Layer

Vectorized Input
(32 x 32 x 3) = 3072

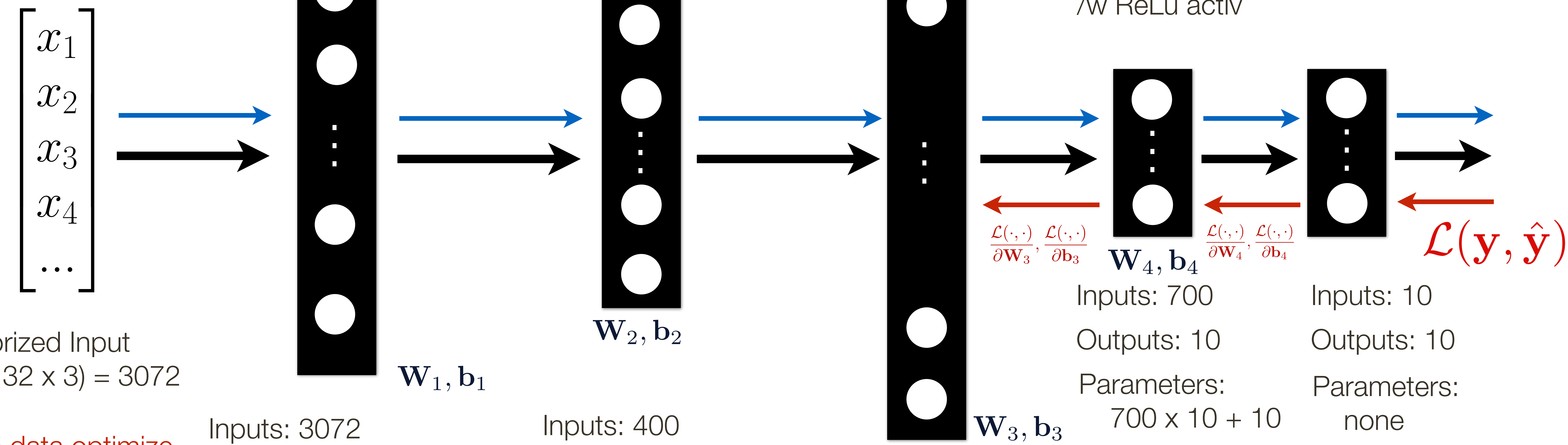
Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ

Output Layer
* Fully Connected
/w 10 neurons
/w ReLu activ

+ sigmoid



Hidden Layer 1:
Inputs: 3072
Outputs: 400
Parameters:
3072 x 400 + 400

Hidden Layer 2:
Inputs: 400
Outputs: 100
Parameters:
400 x 100 + 100

Hidden Layer 3:
Inputs: 100
Outputs: 700
Parameters:
100 x 700 + 700

Output Layer:
Inputs: 700
Outputs: 10
Parameters:
700 x 10 + 10

Sigmoid:
Inputs: 10
Outputs: 10
Parameters:
none

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

Learning: given data optimize parameters using gradient -based optimization

(a.k.a. **Backwards Pass**)

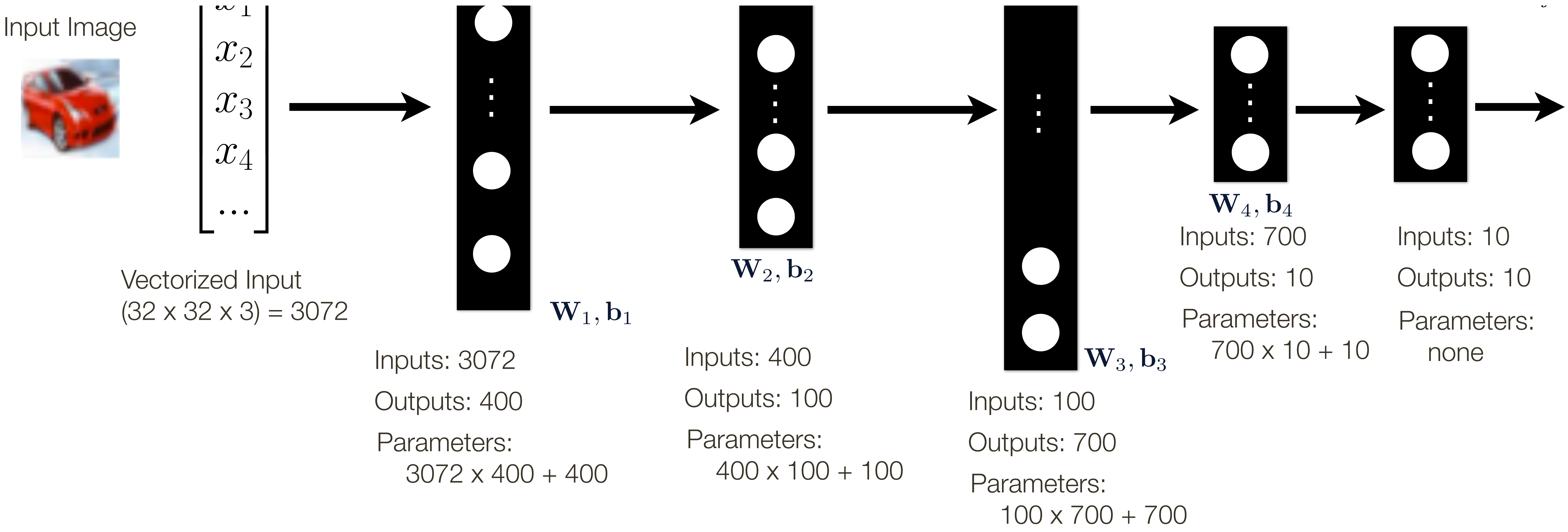
Neural Network: Short Review

Neural Network: Short Review

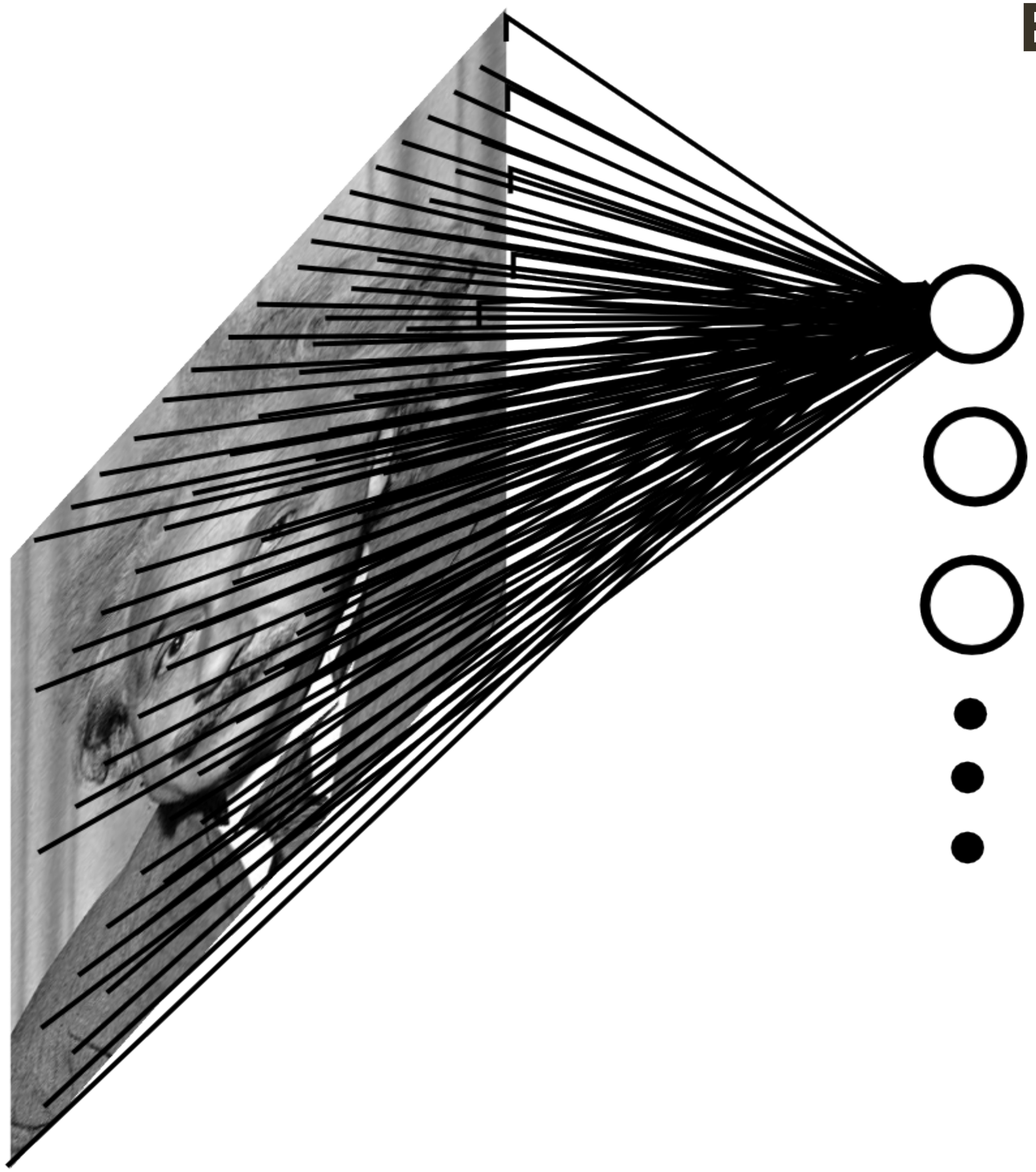
Neural Network: Short Review

Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



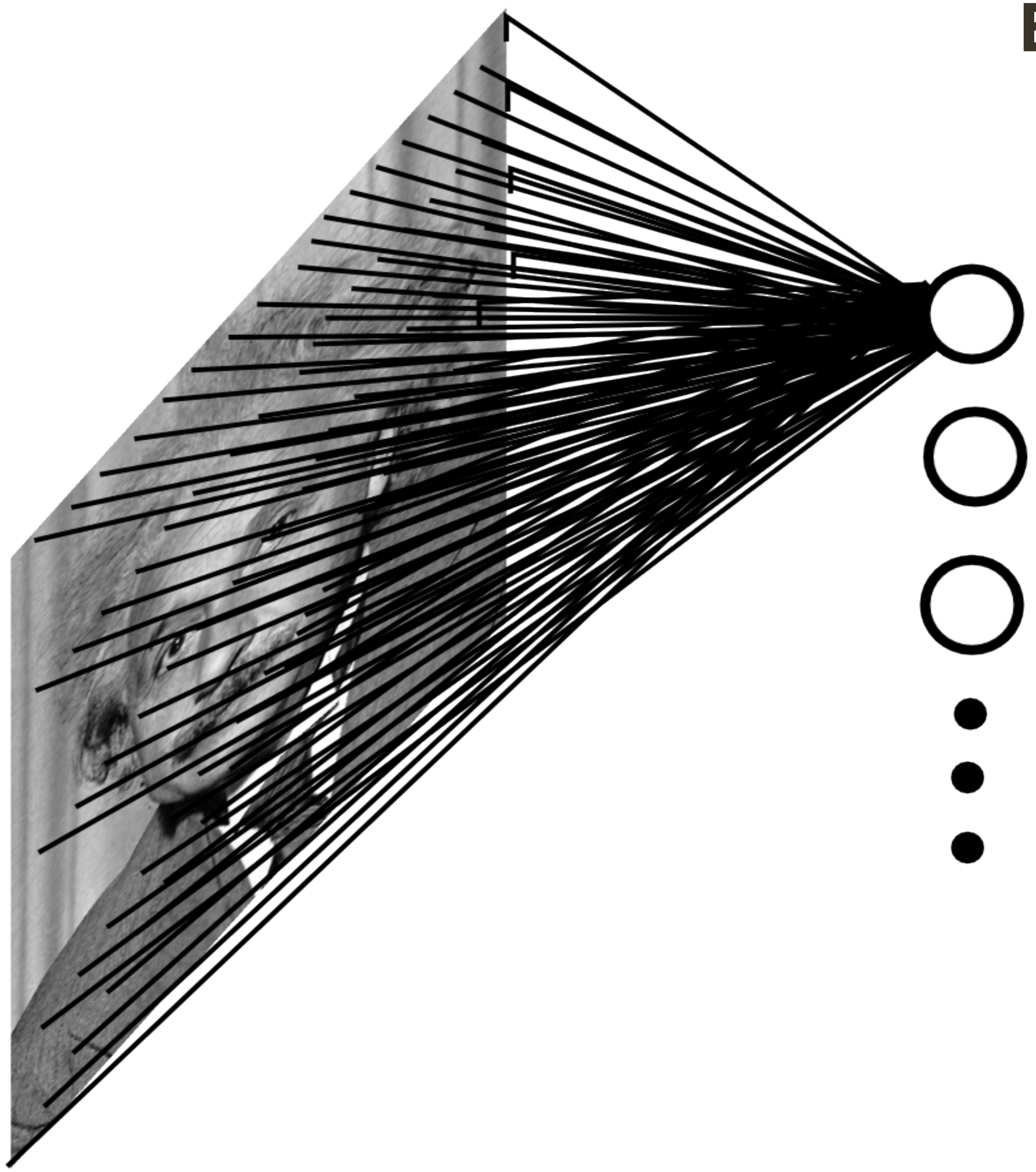
Fully Connected Layer



Example: 200 x 200 image (small)
x 40K hidden units

* slide from Marc'Aurelio Renzato

Fully Connected Layer

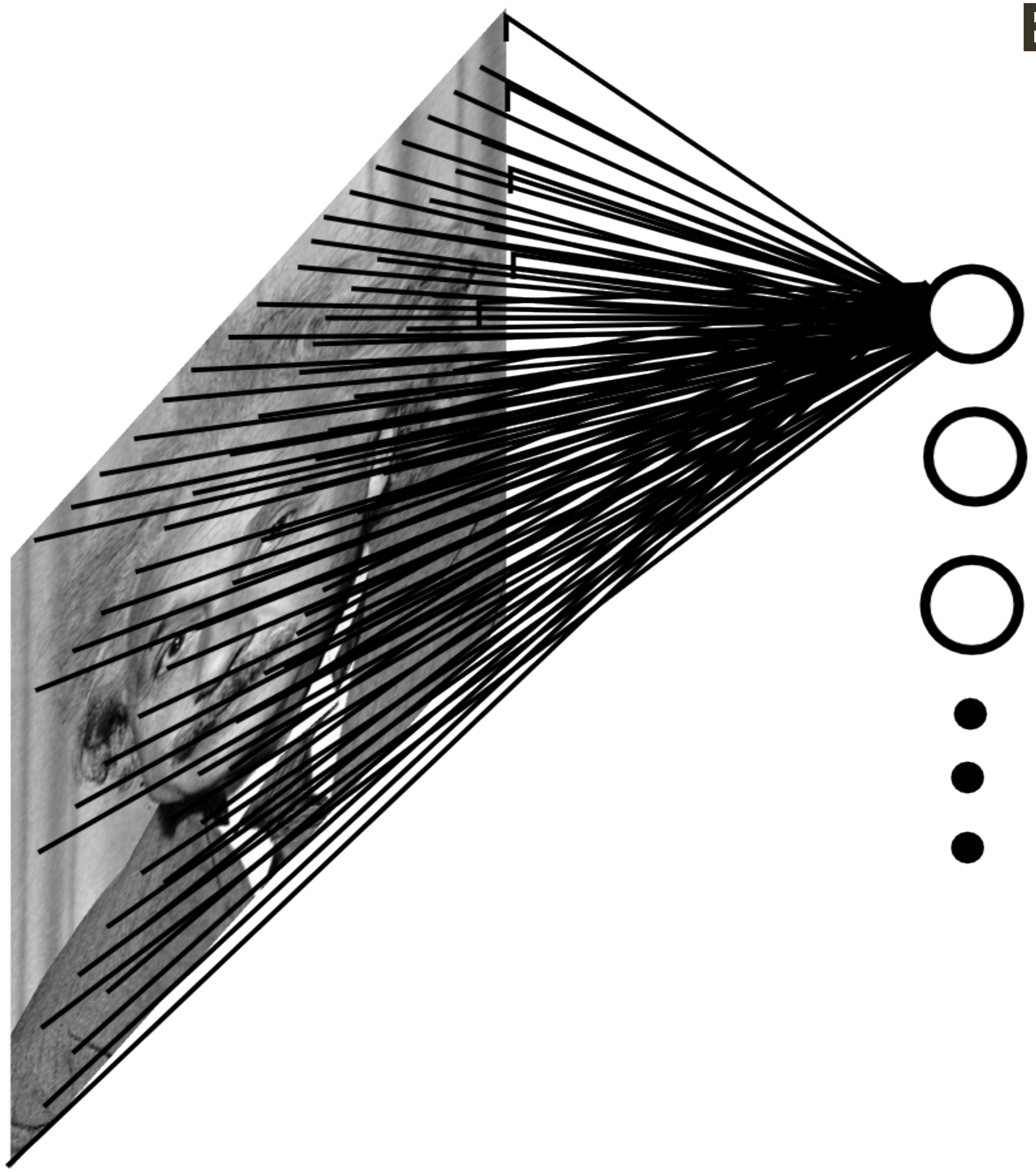


Example: 200 x 200 image (small)
x 40K hidden units

= ~ **2 Billion** parameters (for one layer!)

* slide from Marc'Aurelio Renzato

Fully Connected Layer



Example: 200 x 200 image (small)
x 40K hidden units

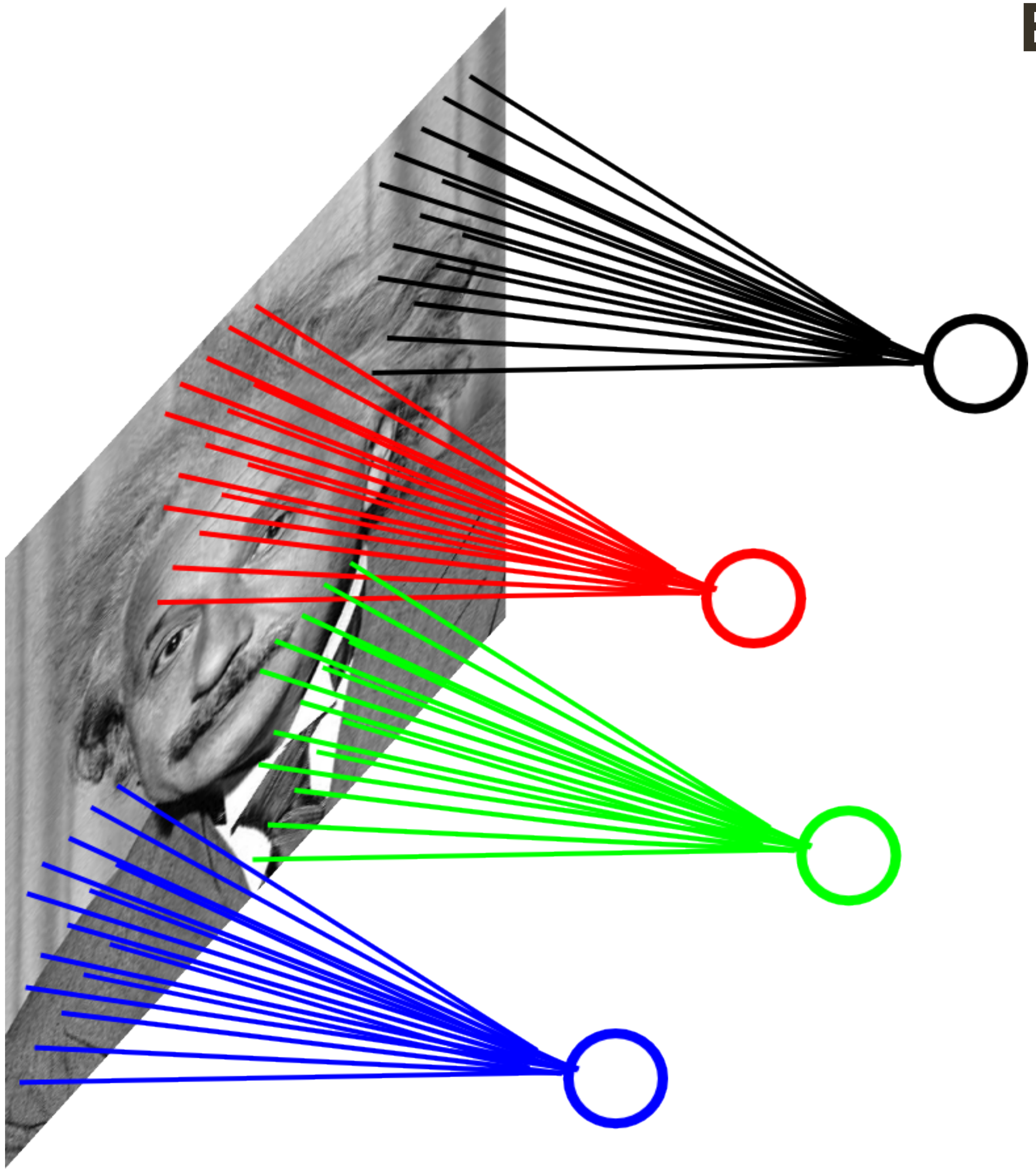
= ~ **2 Billion** parameters (for one layer!)

Spatial correlations are generally local

Waste of resources + we don't have enough data to train networks this large

* slide from Marc'Aurelio Renzato

Locally Connected Layer



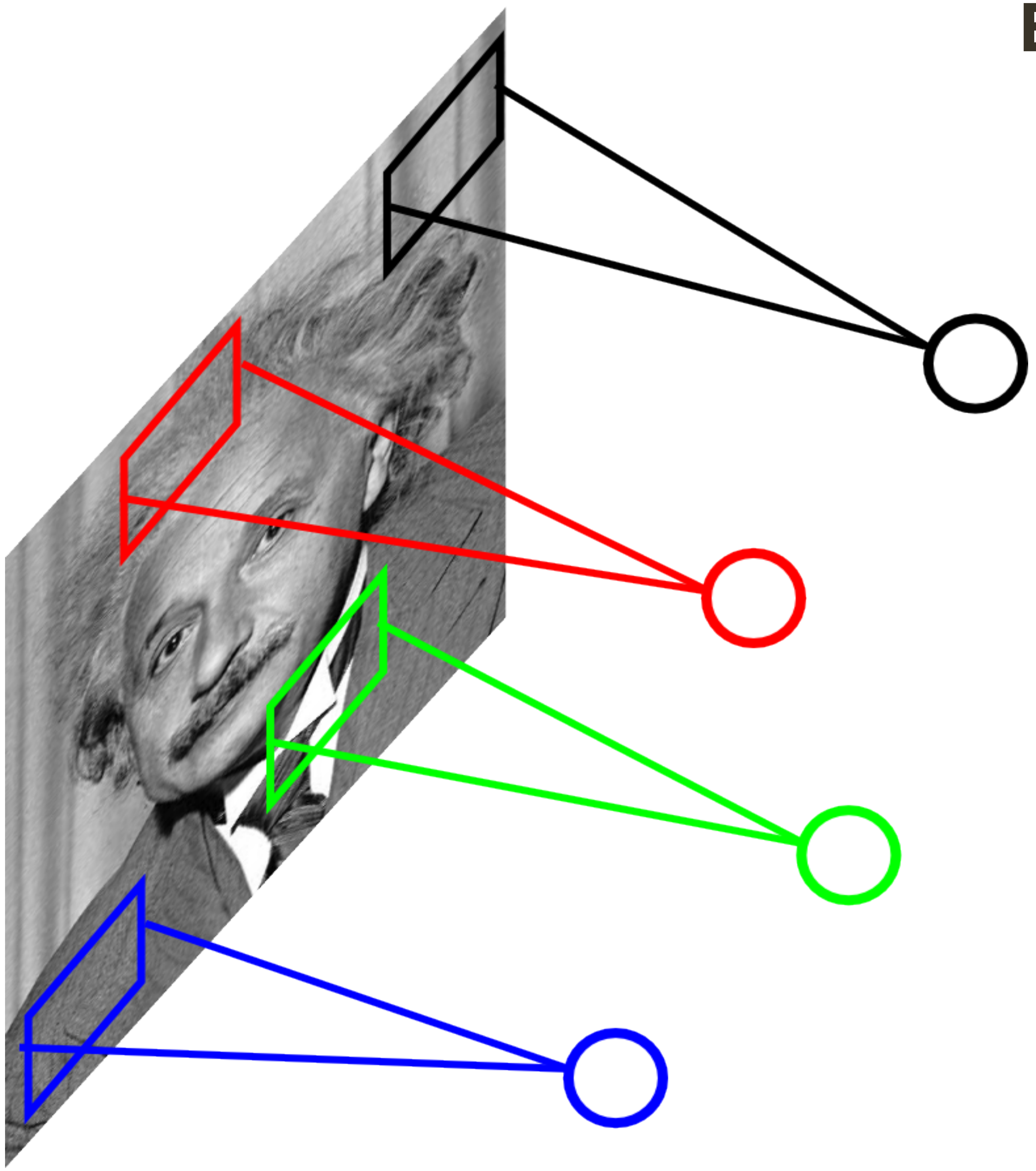
Example: 200 x 200 image (small)
x 40K hidden units

Filter size: 10 x 10

= ~ **4 Million** parameters

* slide from Marc'Aurelio Renzato

Locally Connected Layer



Example: 200 x 200 image (small)
x 40K hidden units

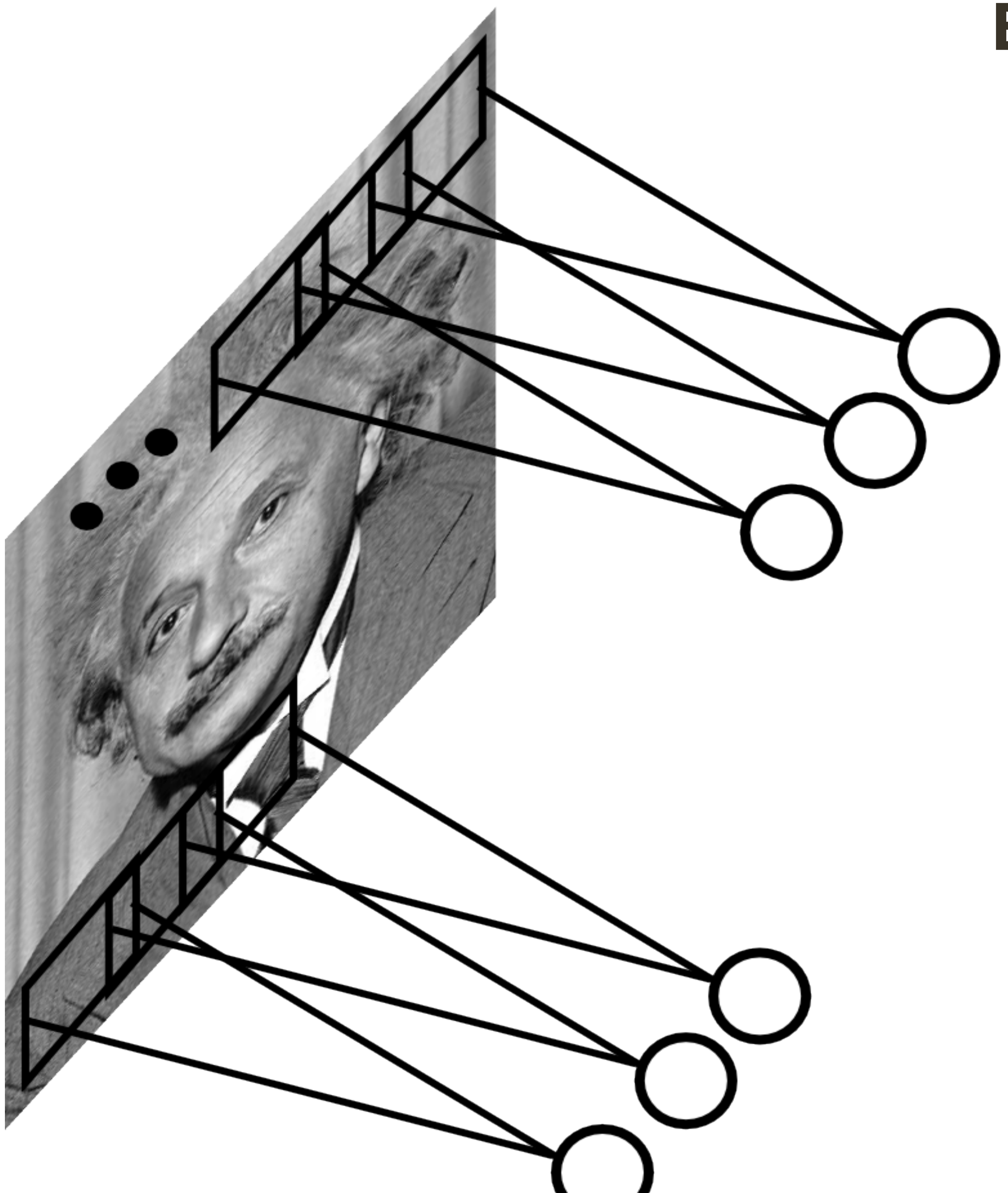
Filter size: 10 x 10

= ~ **4 Million** parameters

Stationarity — statistics is similar at different locations

* slide from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

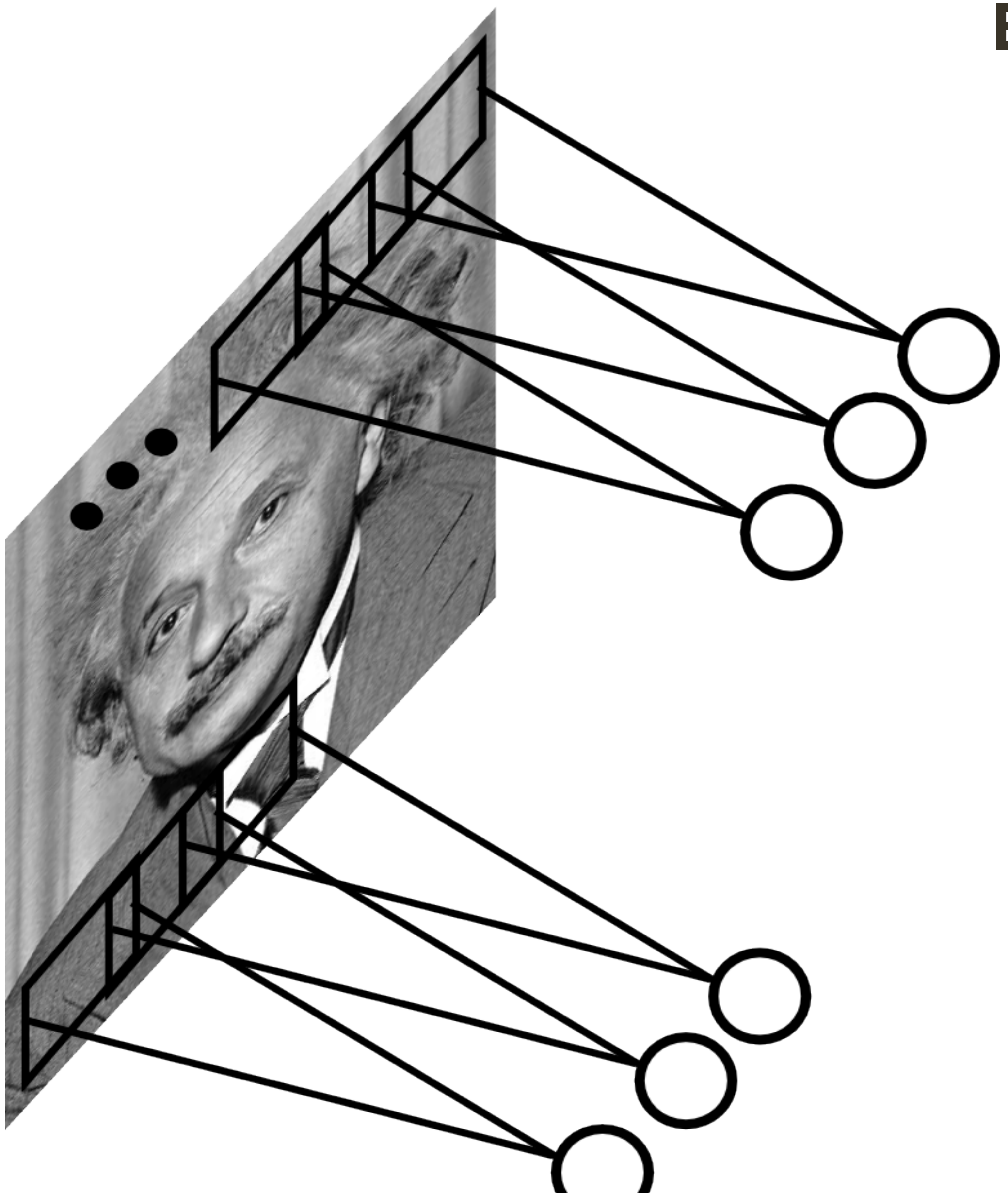
Filter size: 10 x 10

= ~ **4 Million** parameters

Share the same parameters across the locations (assuming input is stationary)

* slide adopted from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

Filter size: 10 x 10

= ~ ~~**4 Million**~~ parameters

= 100+1 parameters

Share the same parameters across the locations (assuming input is stationary)

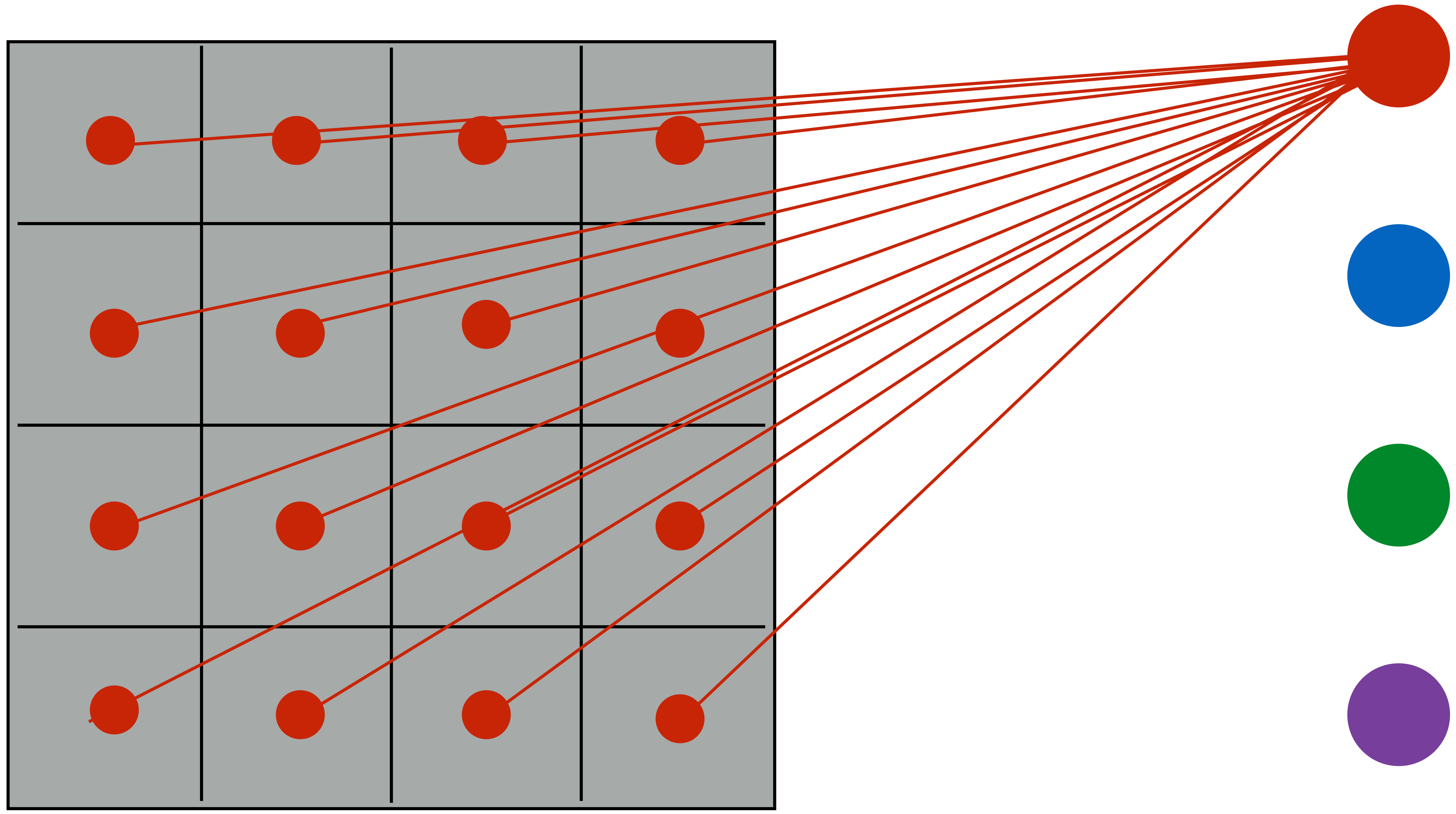
* slide adopted from Marc'Aurelio Renzato

Fully Connected Layer

Fully Connected Layer

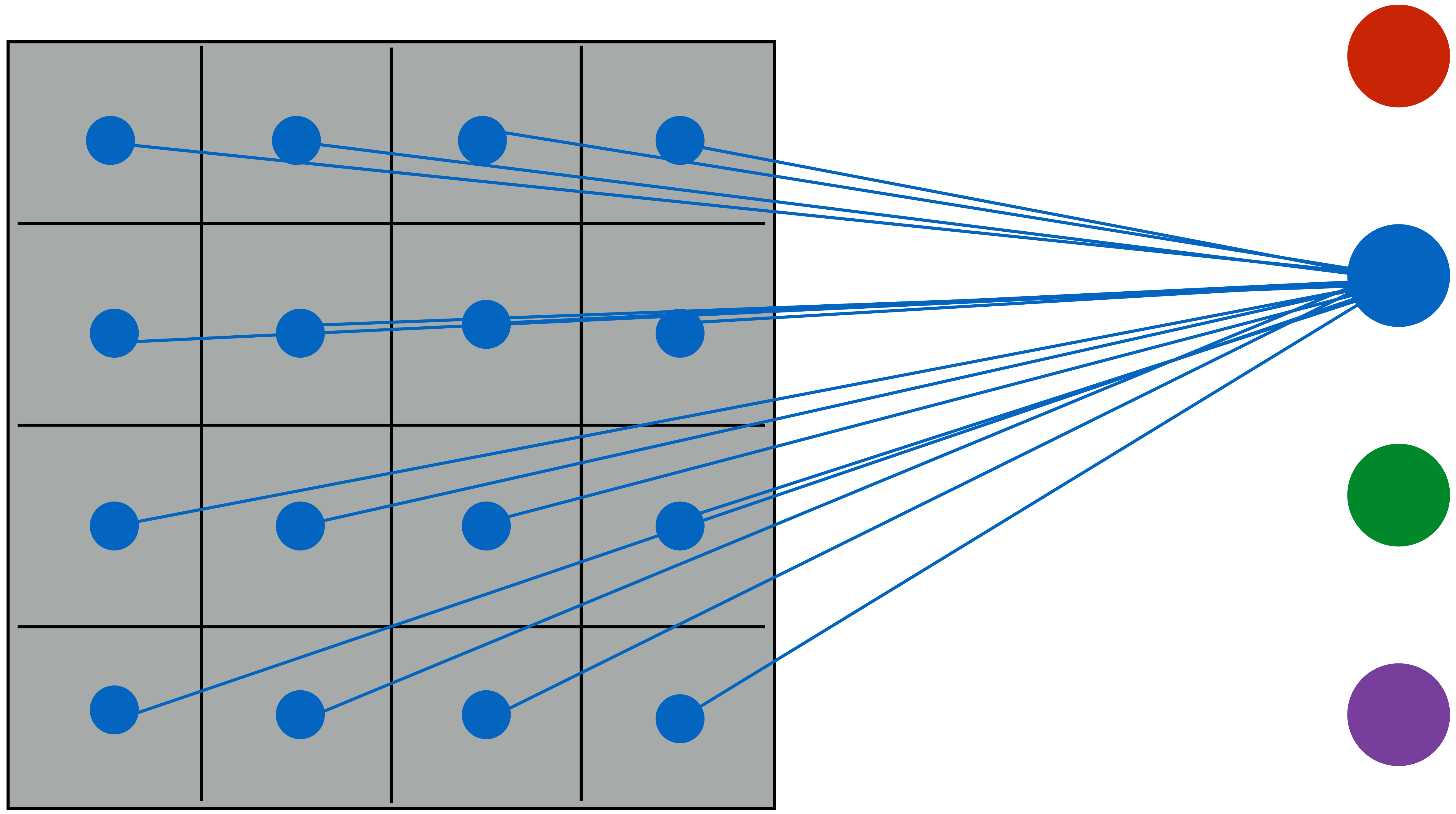
$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

Linear Layer



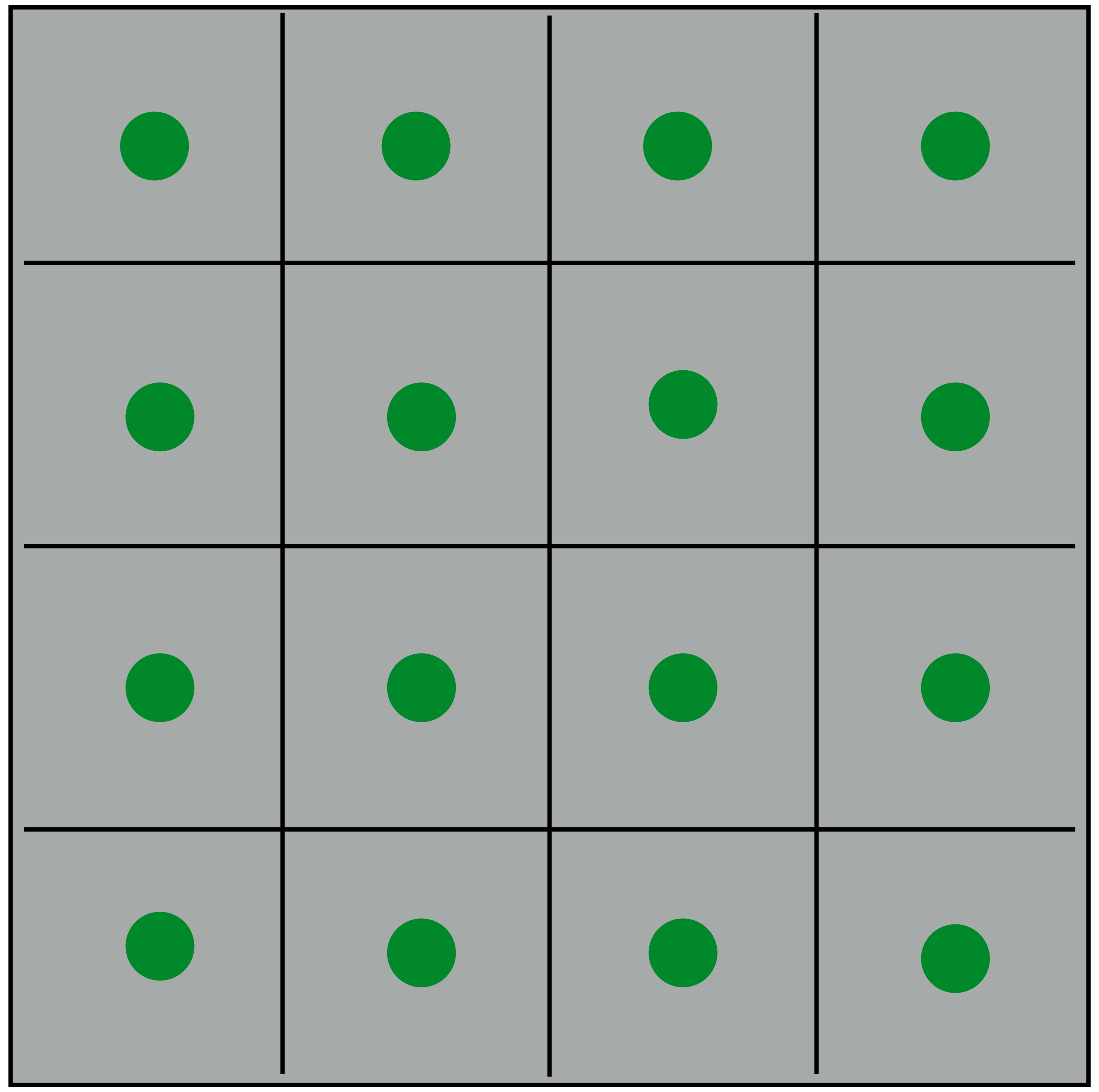
Fully Connected Layer

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$



$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

Fully Connected Layer

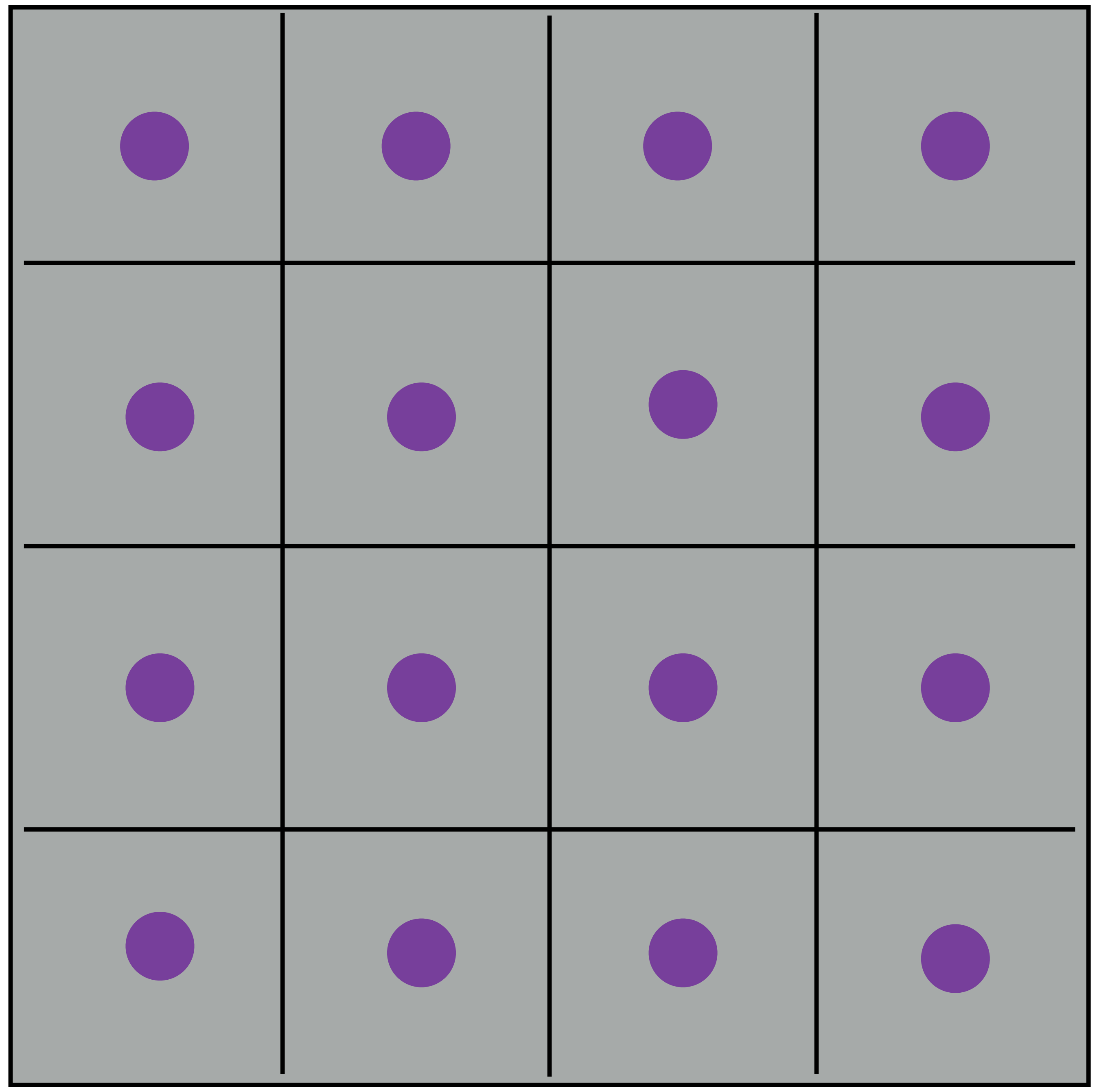


$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

Fully Connected Layer



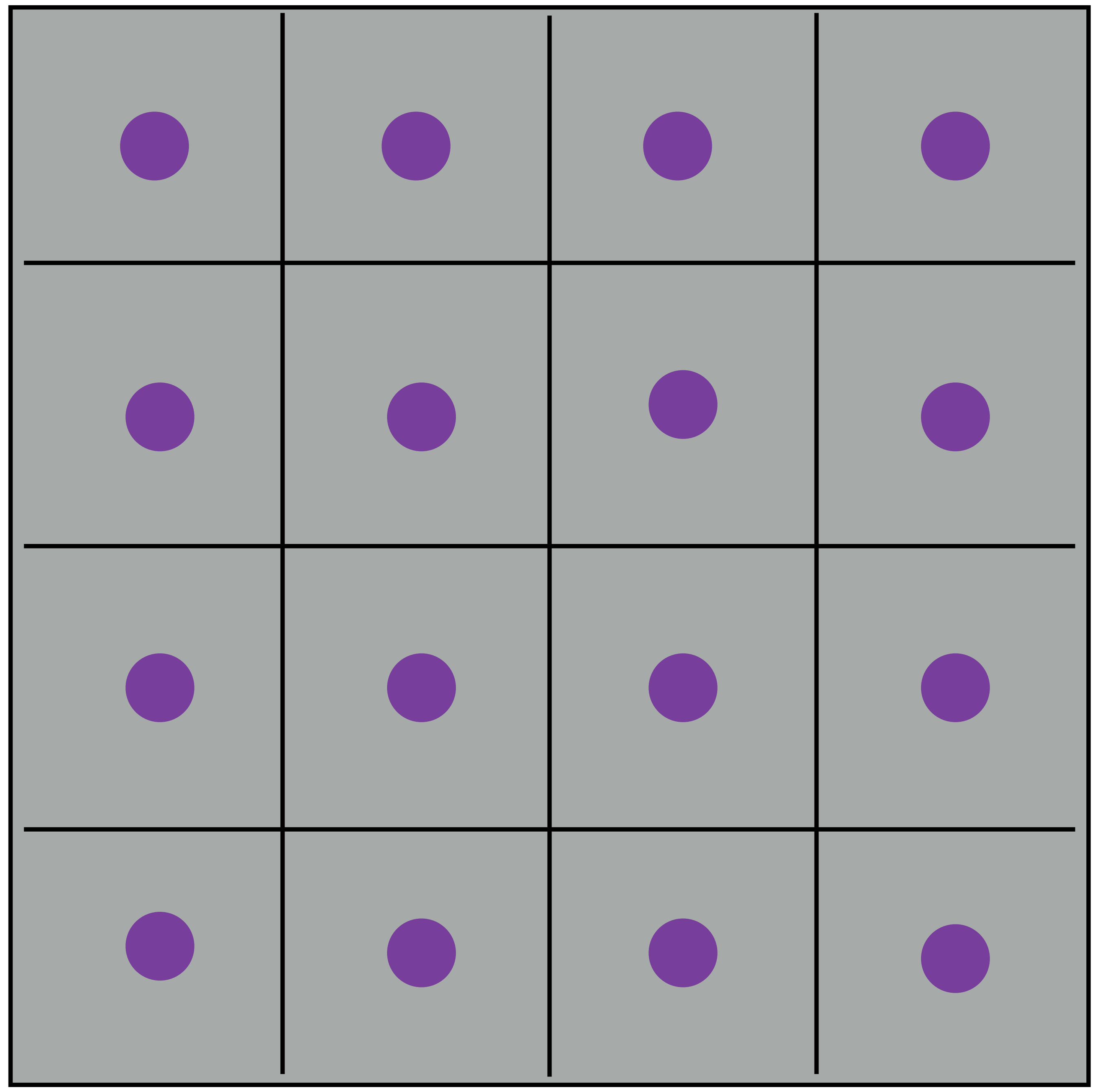
$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{4,i,j}\mathcal{I}(i,j)+b_4\right)$$

Fully Connected Layer



$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

4 x 4 + 1 = 17

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{2,i,j}\mathcal{I}(i,j)+b_2\right)$$

4 x 4 + 1 = 17

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{3,i,j}\mathcal{I}(i,j)+b_3\right)$$

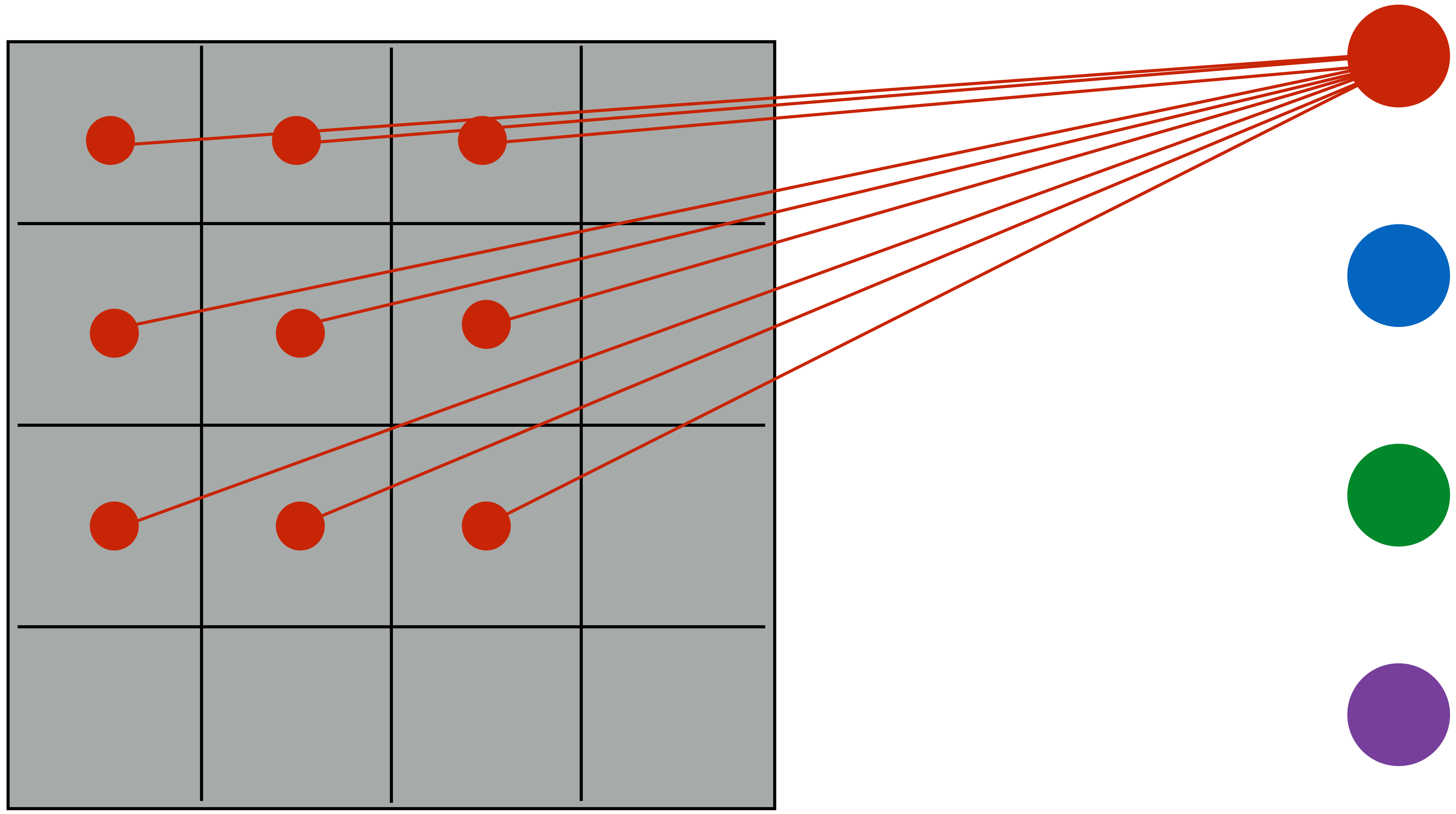
4 x 4 + 1 = 17

$$\sigma\left(\sum_{i=1}^{4}\sum_{j=1}^{4}\mathbf{W}_{4,i,j}\mathcal{I}(i,j)+b_4\right)$$

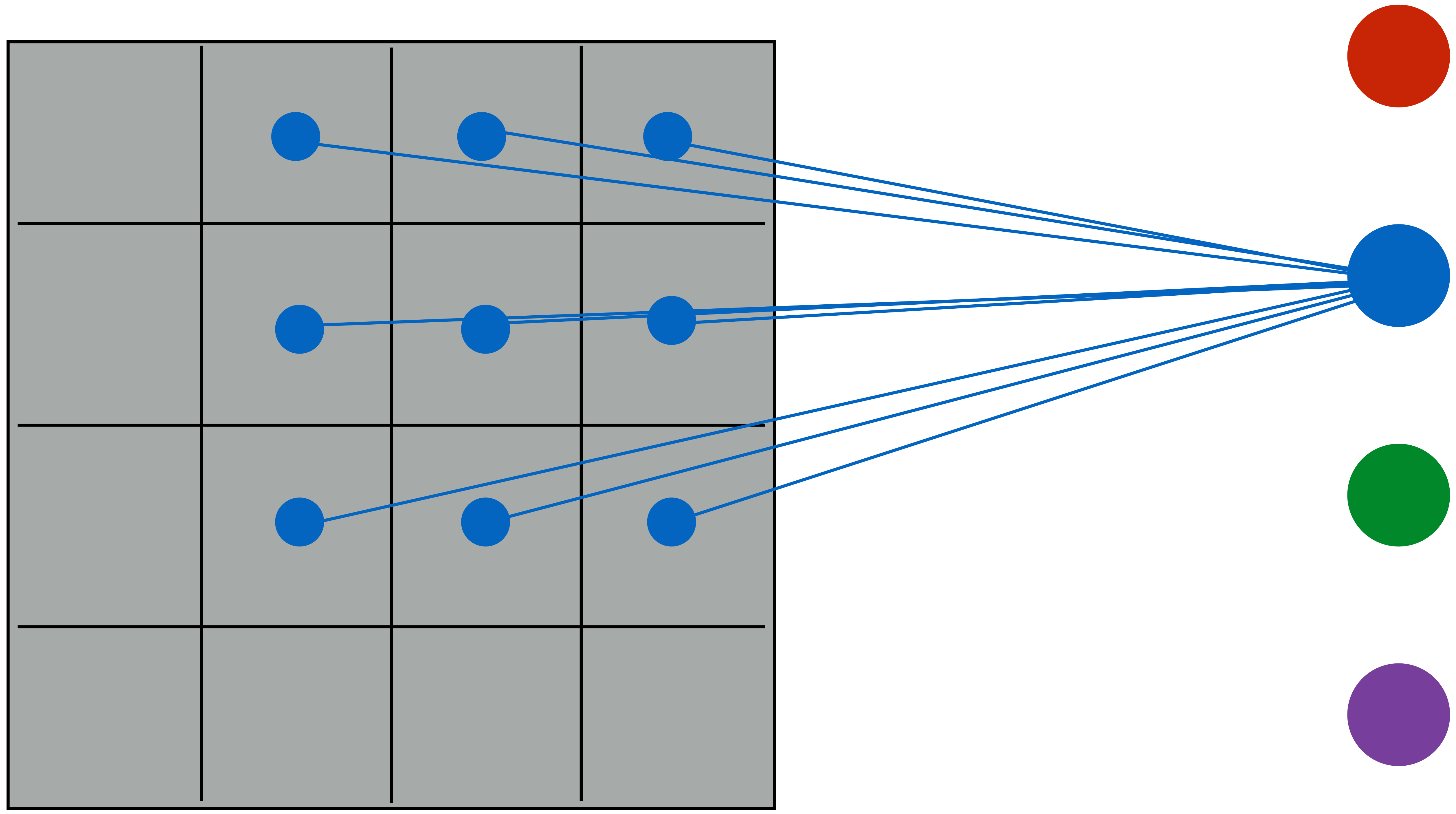
4 x 4 + 1 = 17

Locally Connected Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$



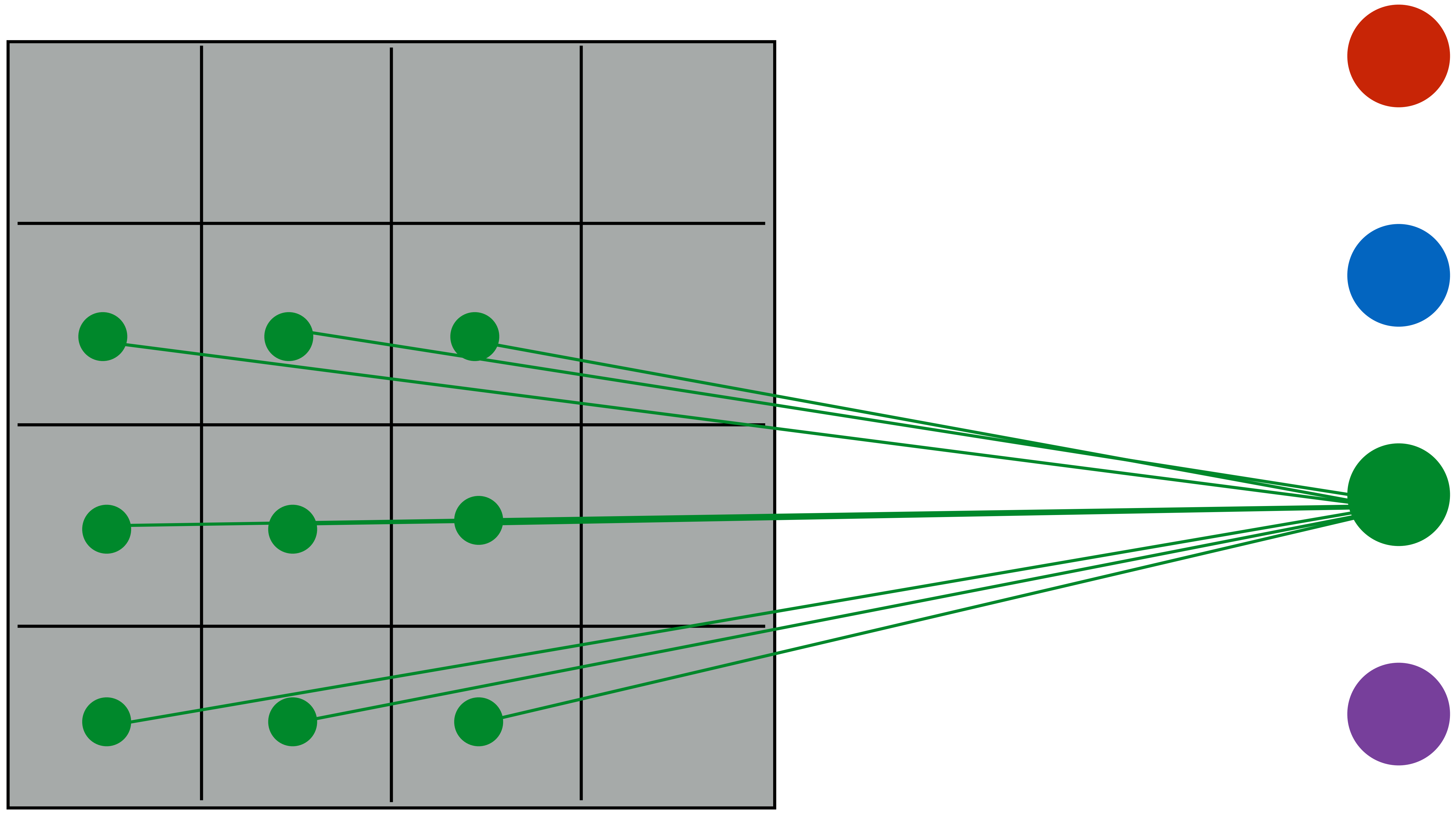
Locally Connected Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

Locally Connected Layer



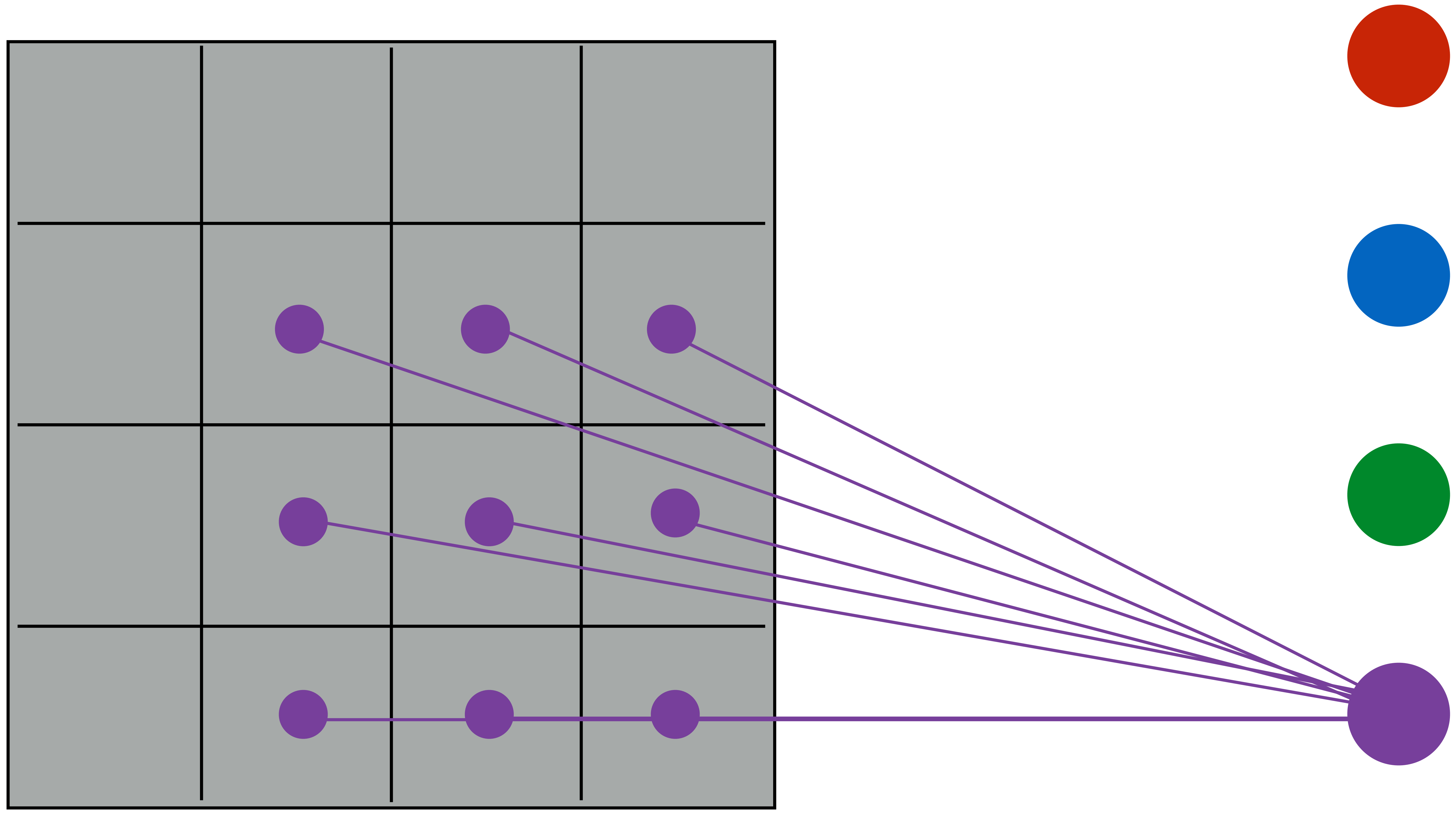
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

Locally Connected Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

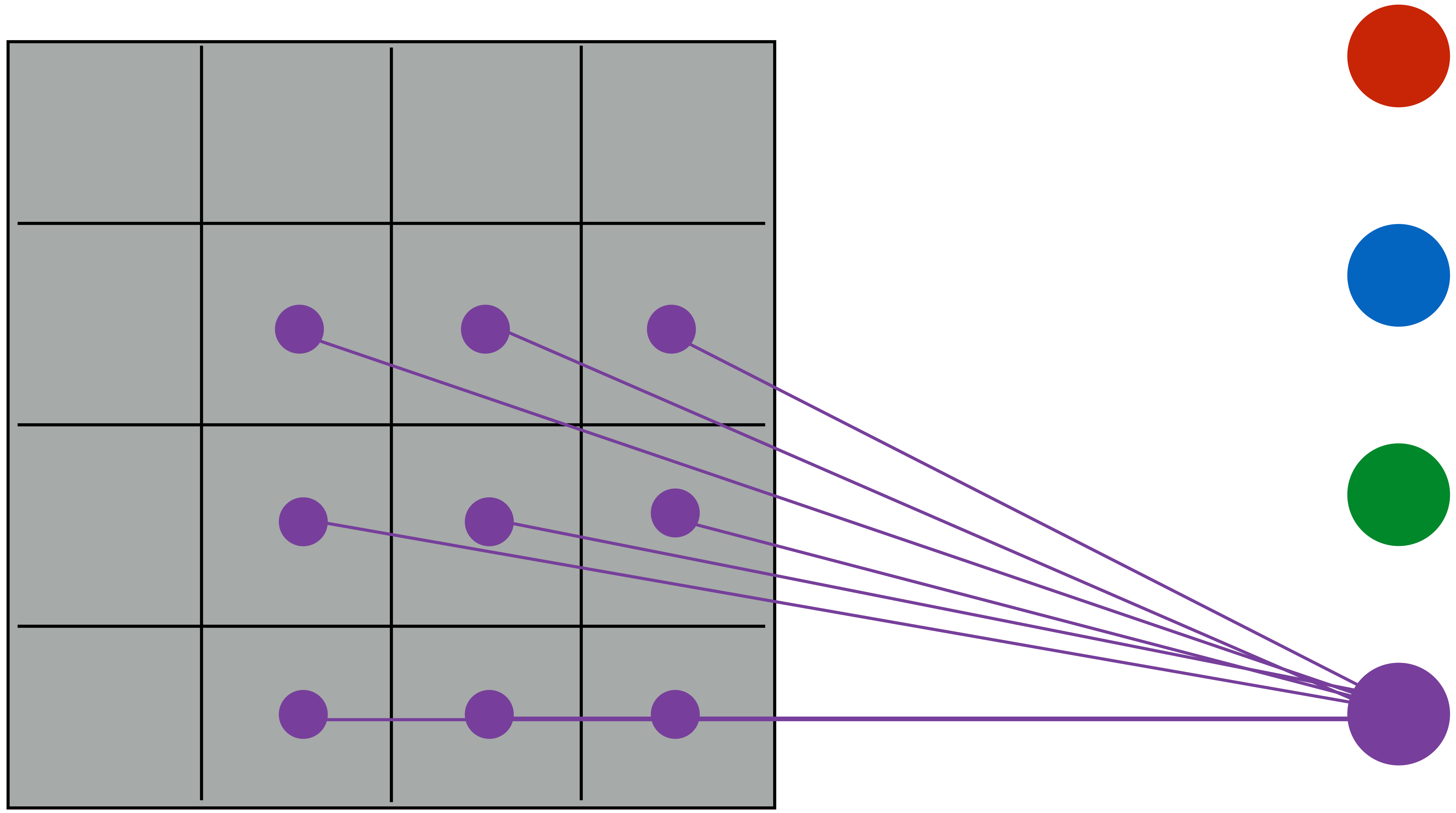


$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{4,i,j}\mathcal{I}(i+1,j+1)+b_4\right)$$

Locally Connected Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{1,i,j}\mathcal{I}(i,j)+b_1\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{2,i,j}\mathcal{I}(i+1,j)+b_2\right)$$

3 x 3 + 1 = 10

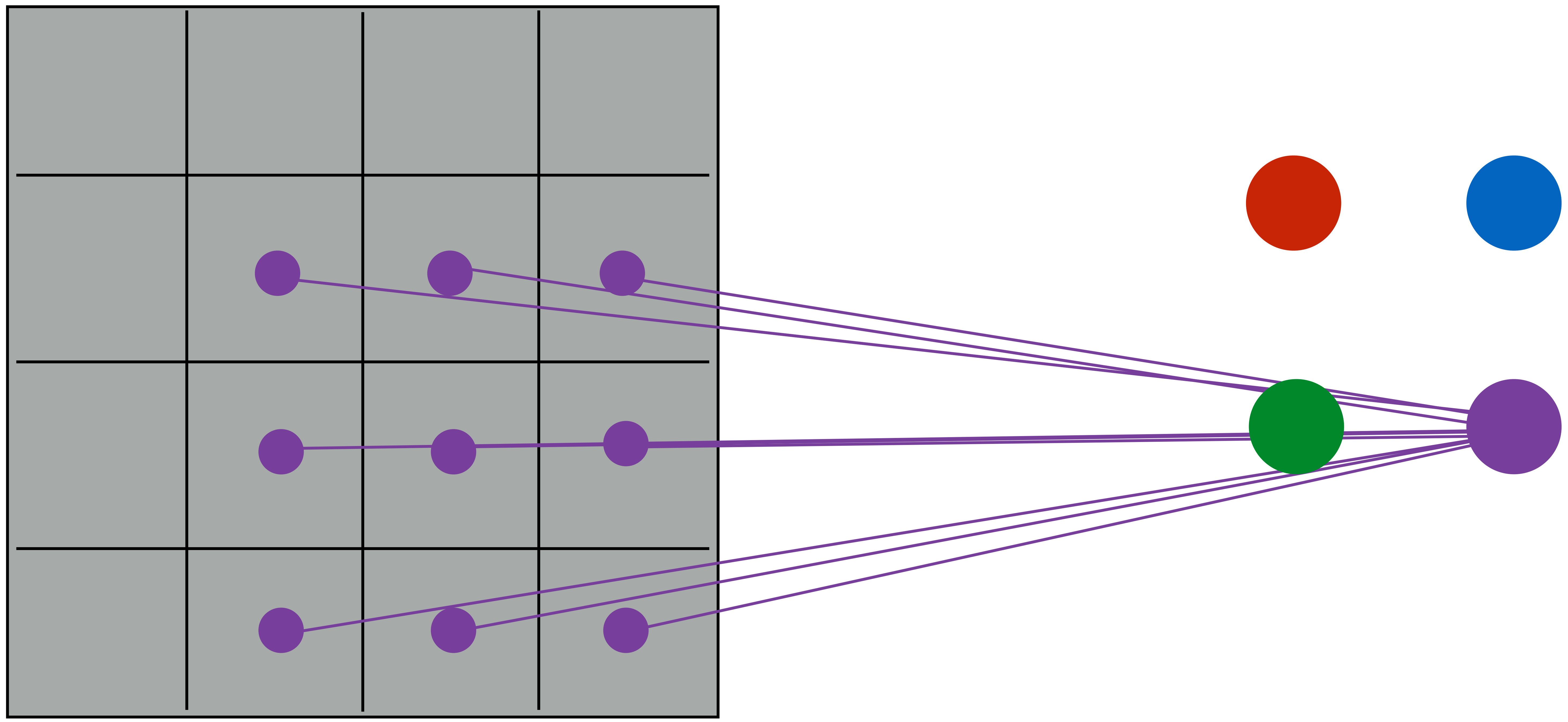
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{3,i,j}\mathcal{I}(i,j+1)+b_3\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{4,i,j}\mathcal{I}(i+1,j+1)+b_4\right)$$

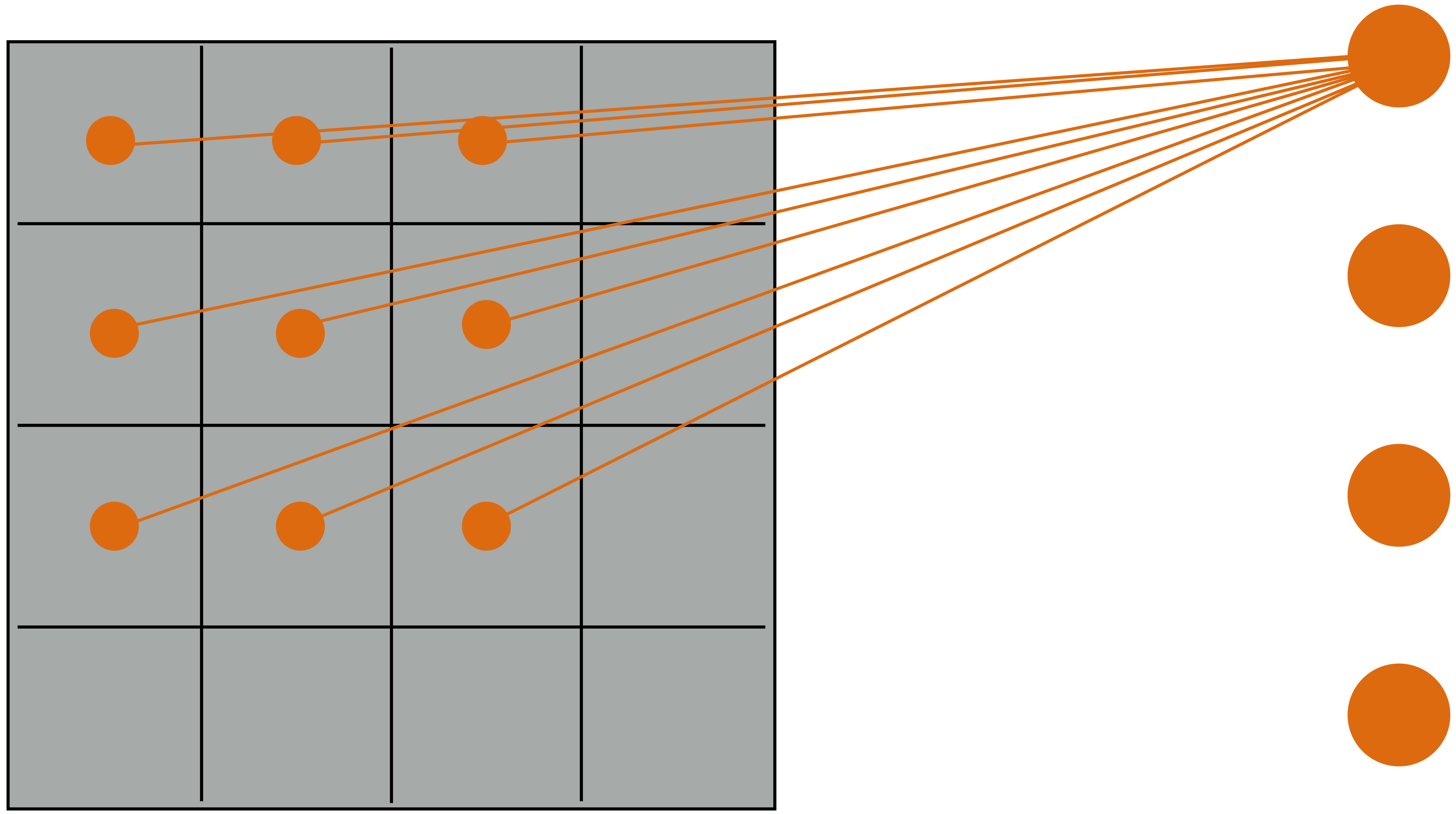
3 x 3 + 1 = 10

Locally Connected Layer

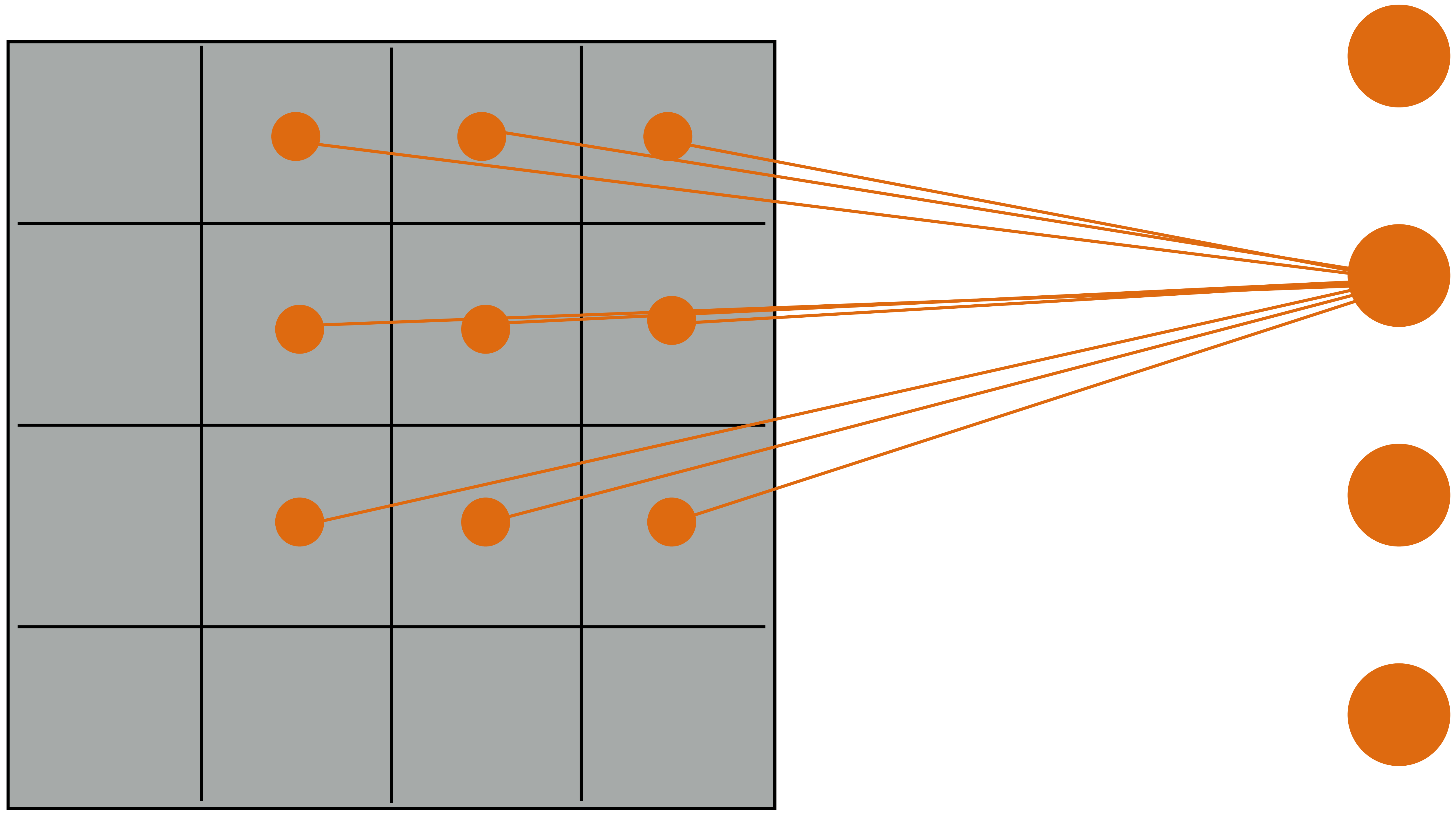


Convolutional Layer

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$



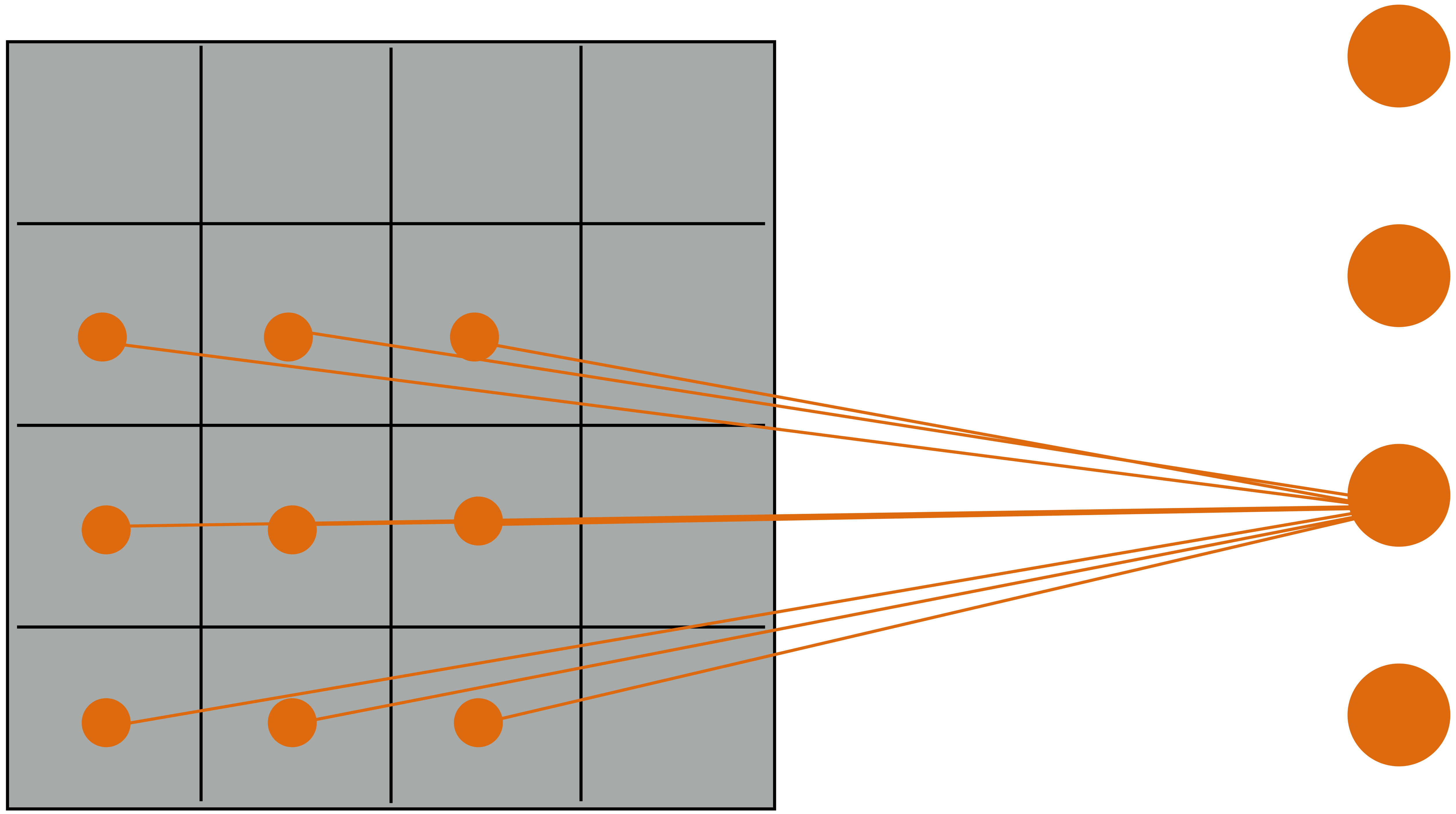
Convolutional Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{w}_{i,j}\mathcal{I}(i,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{w}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

Convolutional Layer

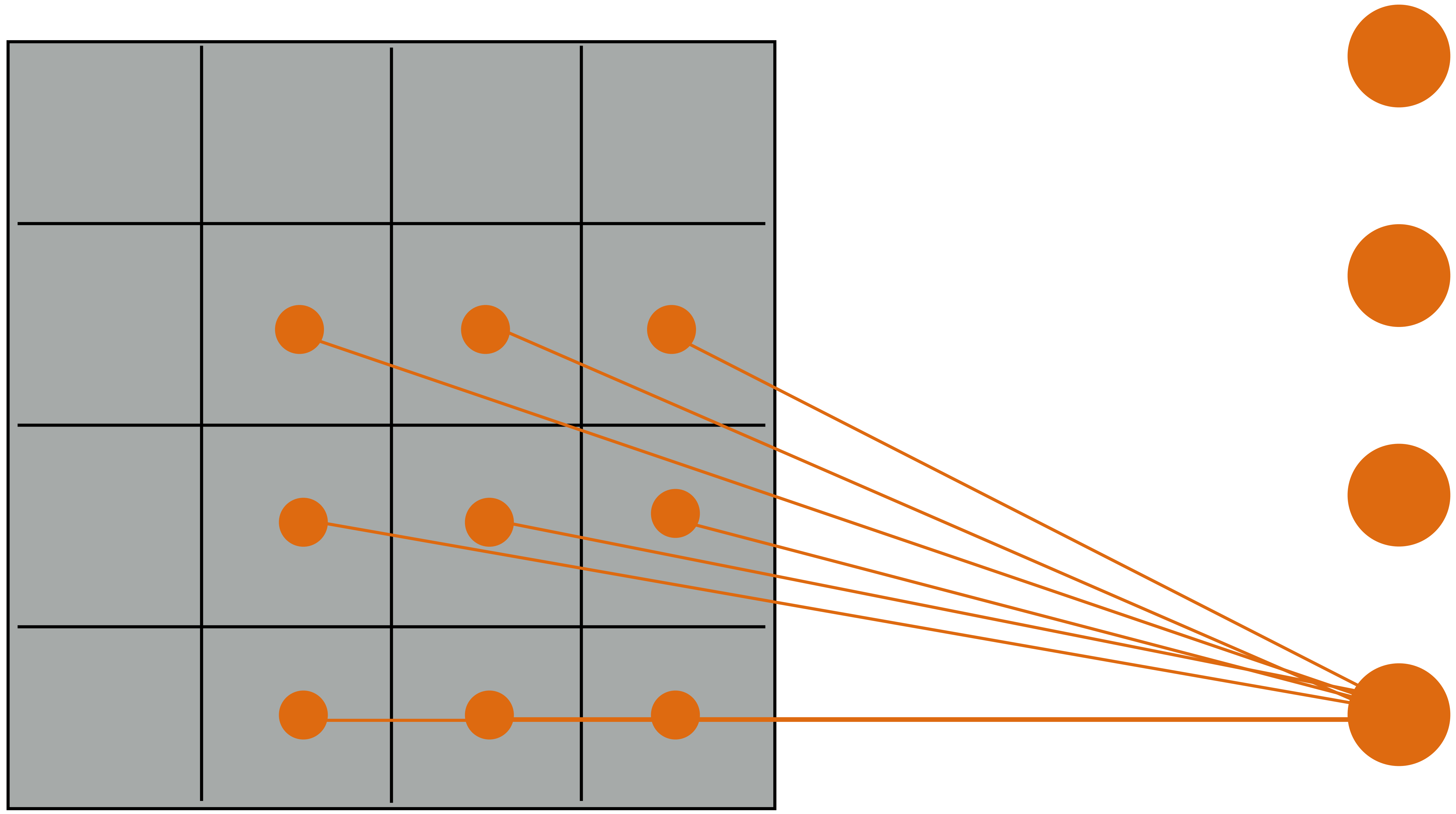


$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$

Convolutional Layer



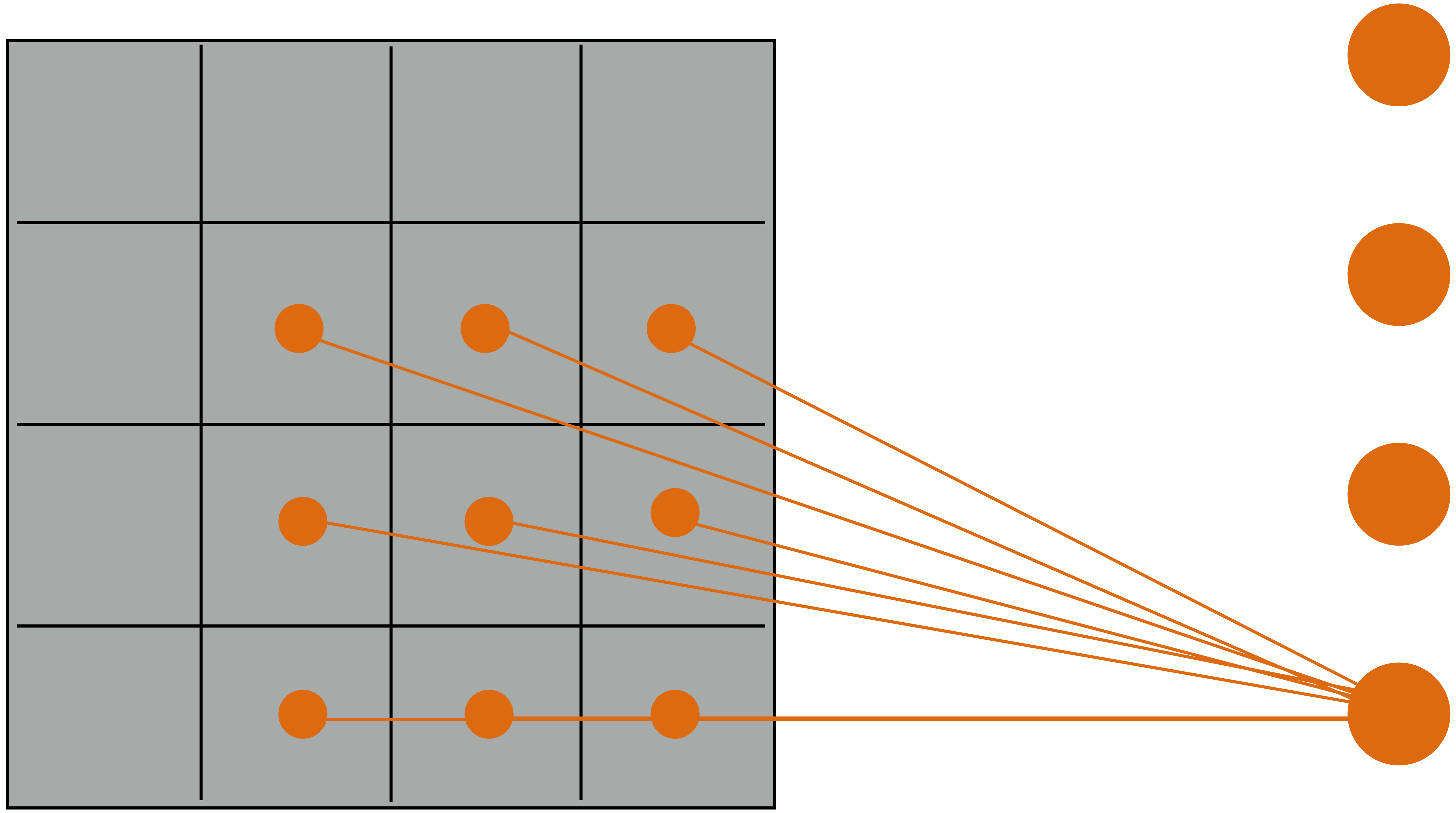
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j+1)+b\right)$$

Convolutional Layer



$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

3 x 3 + 1 = 10

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j)+b\right)$$

0 x 0 + 0 = 0

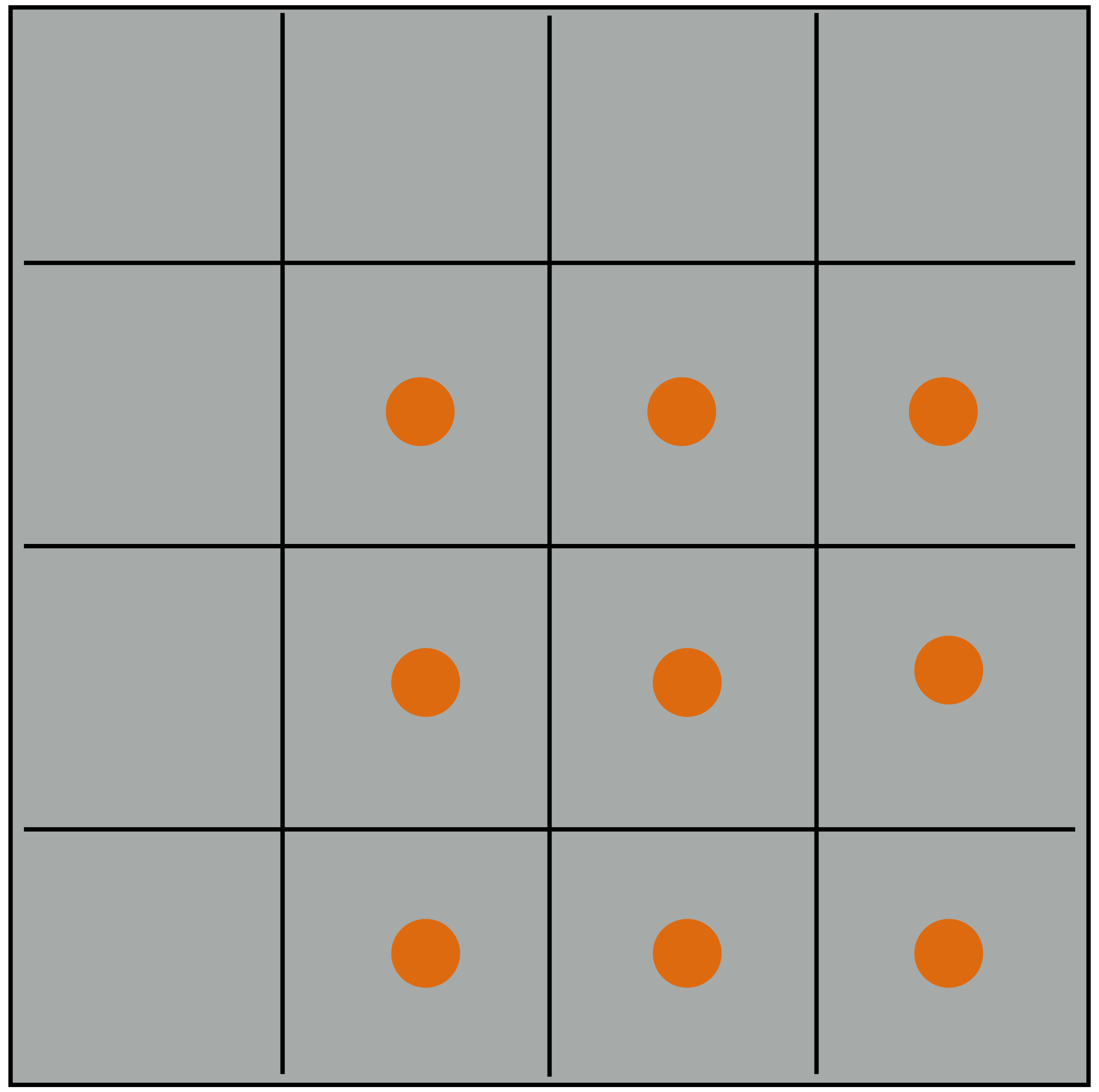
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j+1)+b\right)$$

0 x 0 + 0 = 0

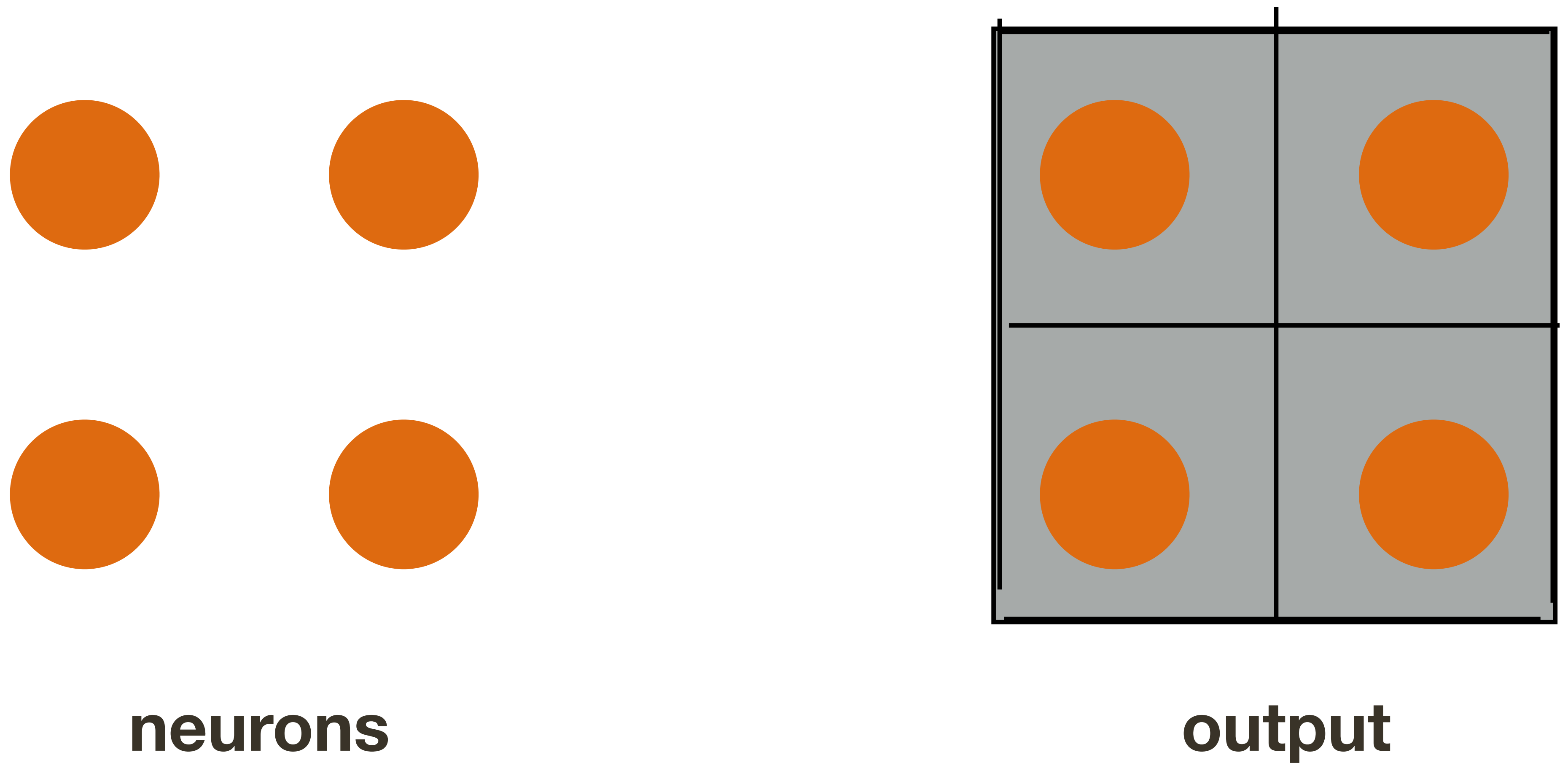
$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i+1,j+1)+b\right)$$

0 x 0 + 0 = 0

Convolutional Layer: Interpretation #1

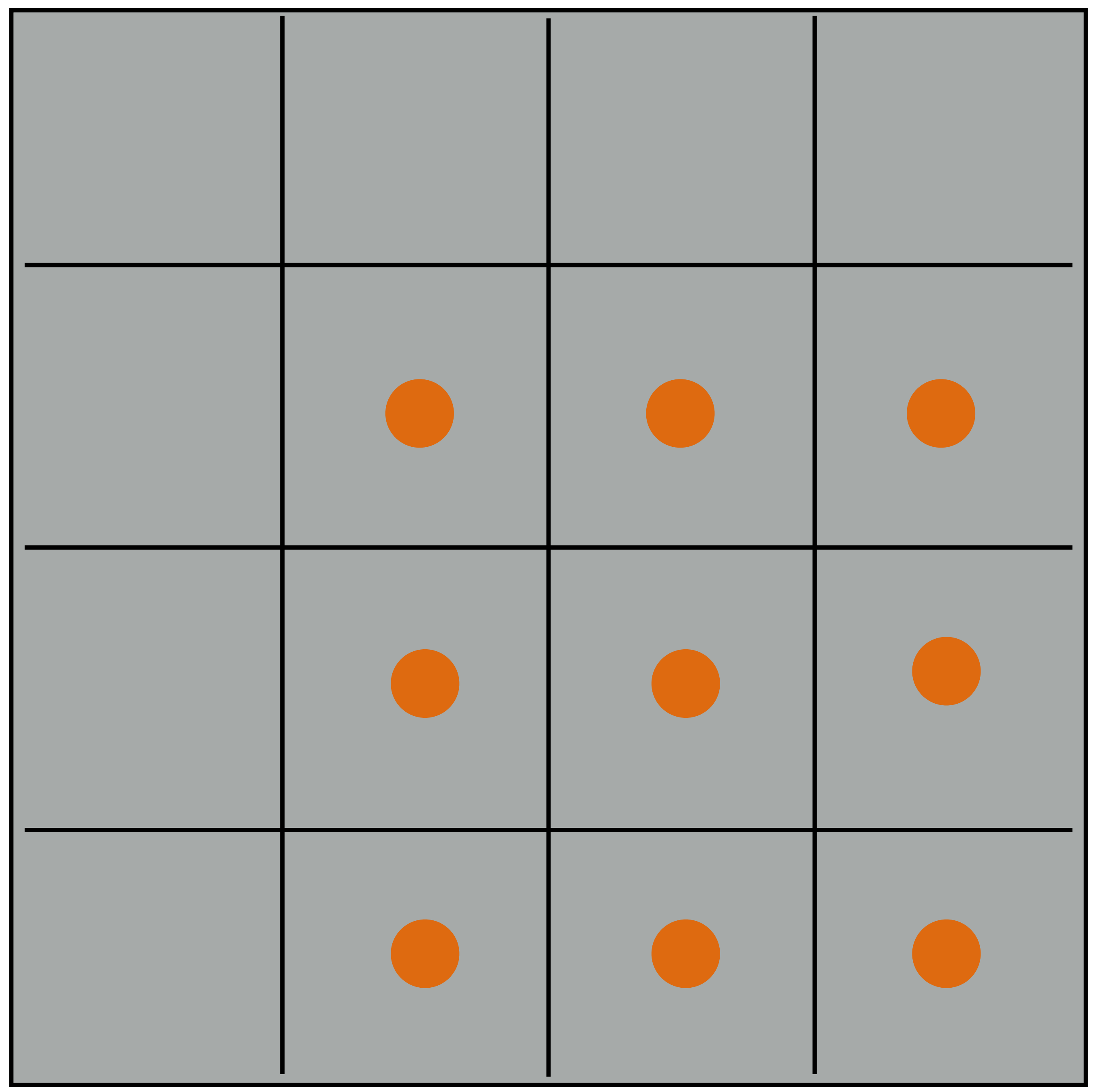


Multiple neurons that share weights

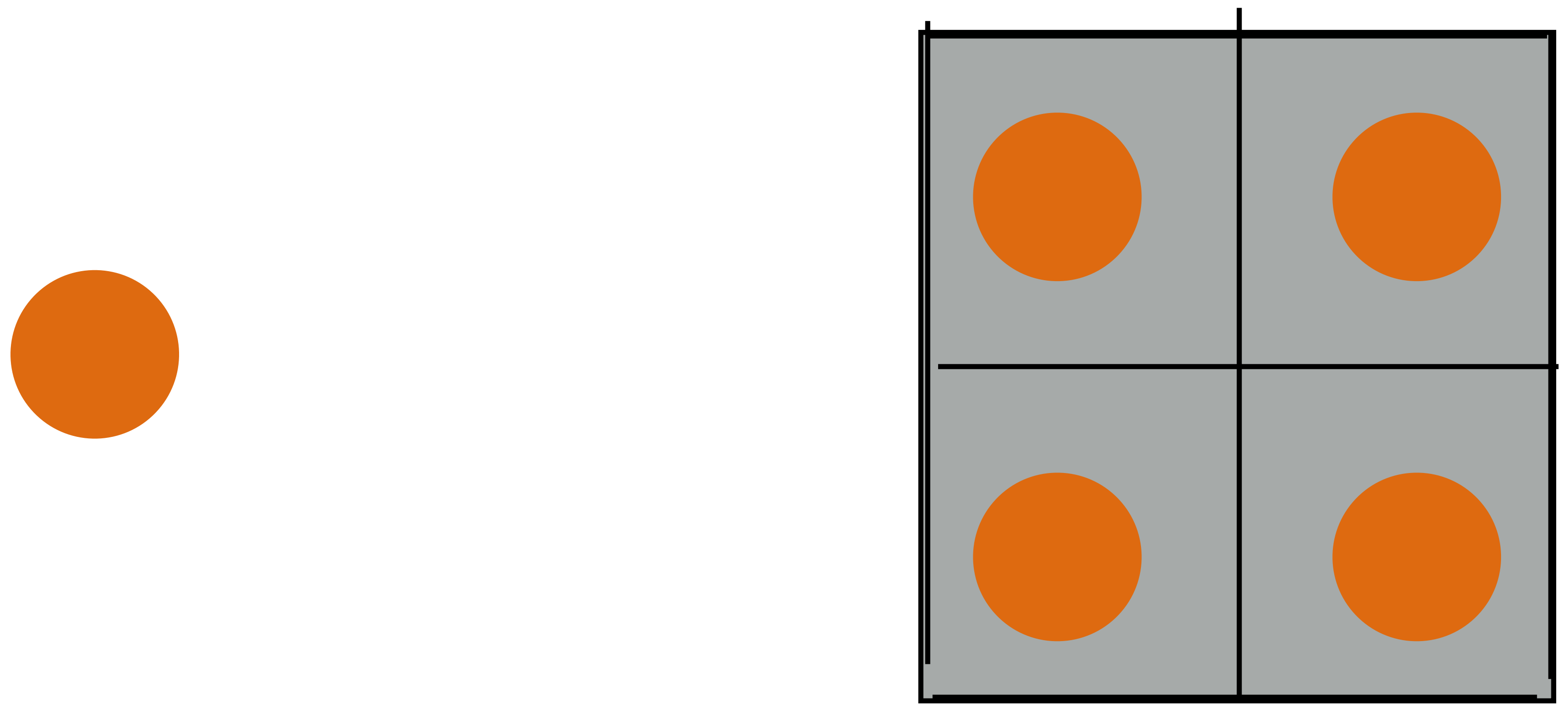


Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)



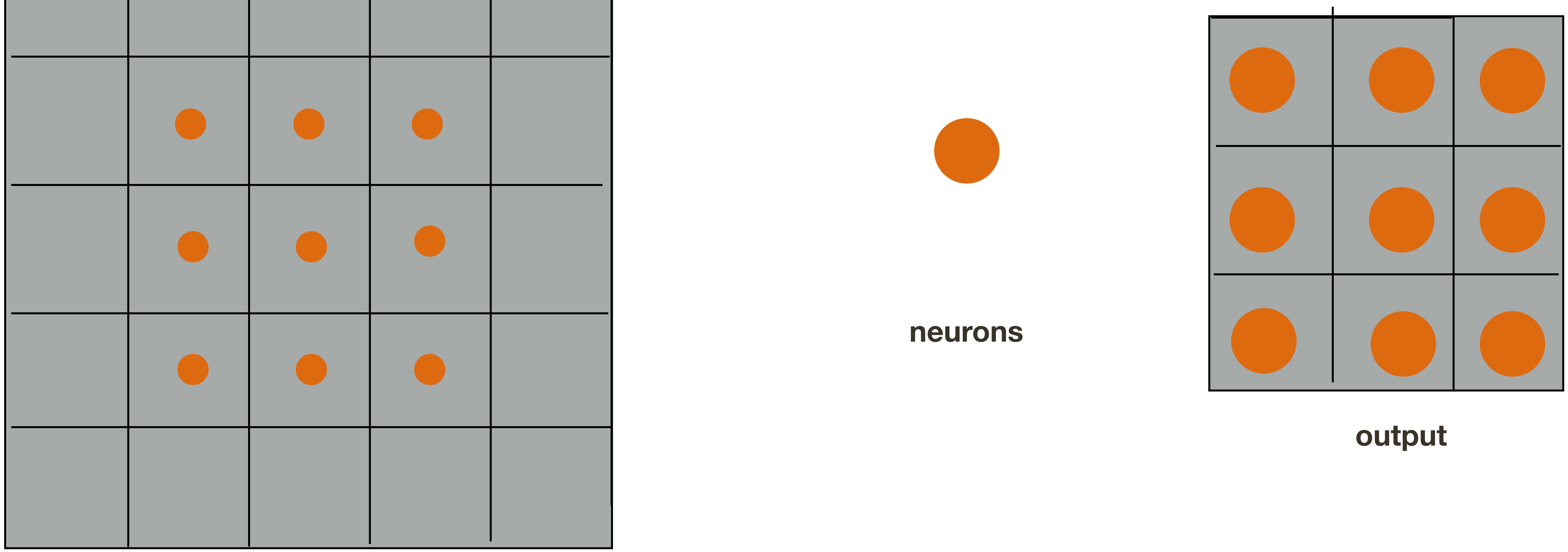
neurons



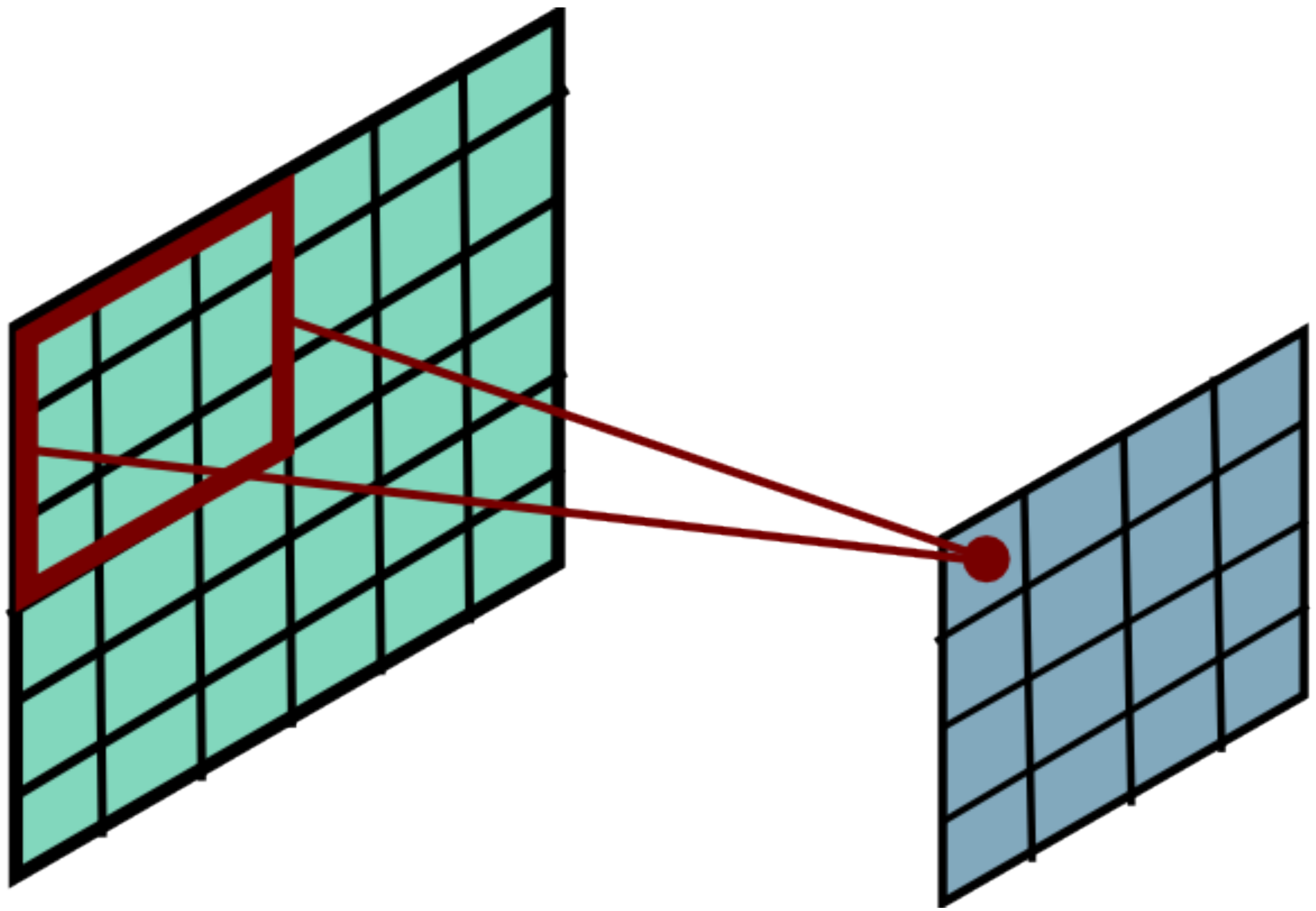
output

Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)

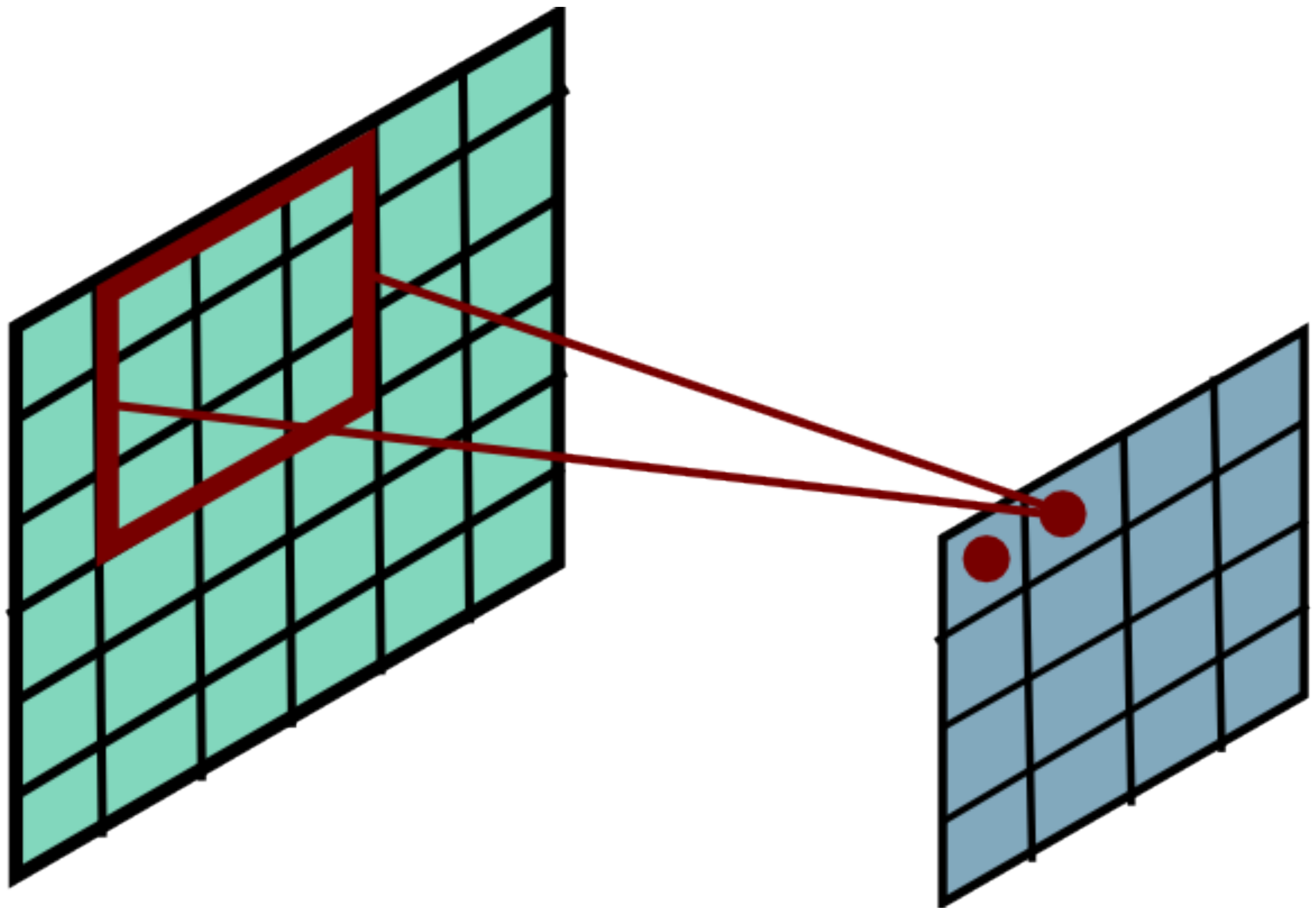


Convolutional Layer



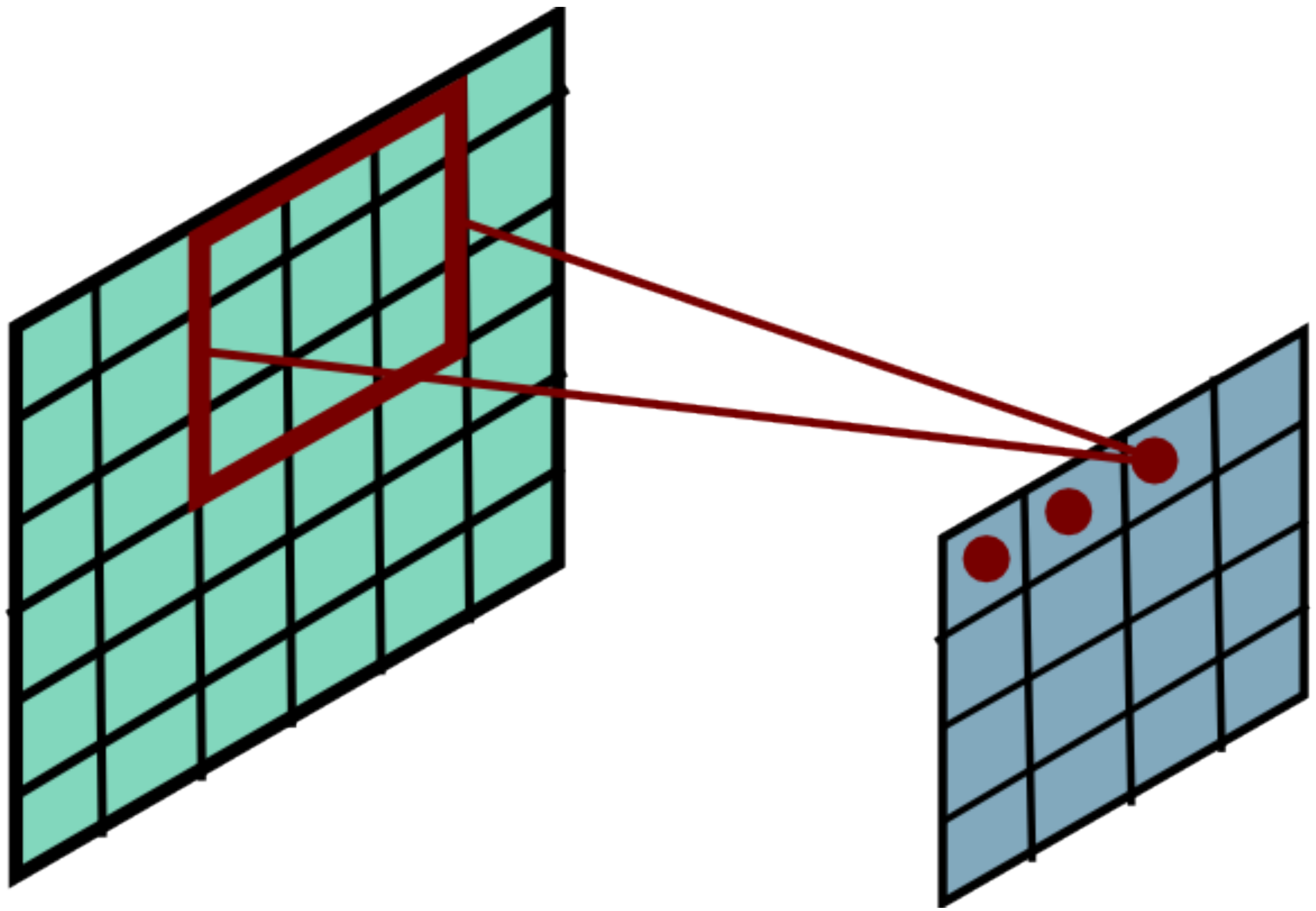
* slide from Marc'Aurelio Renzato

Convolutional Layer



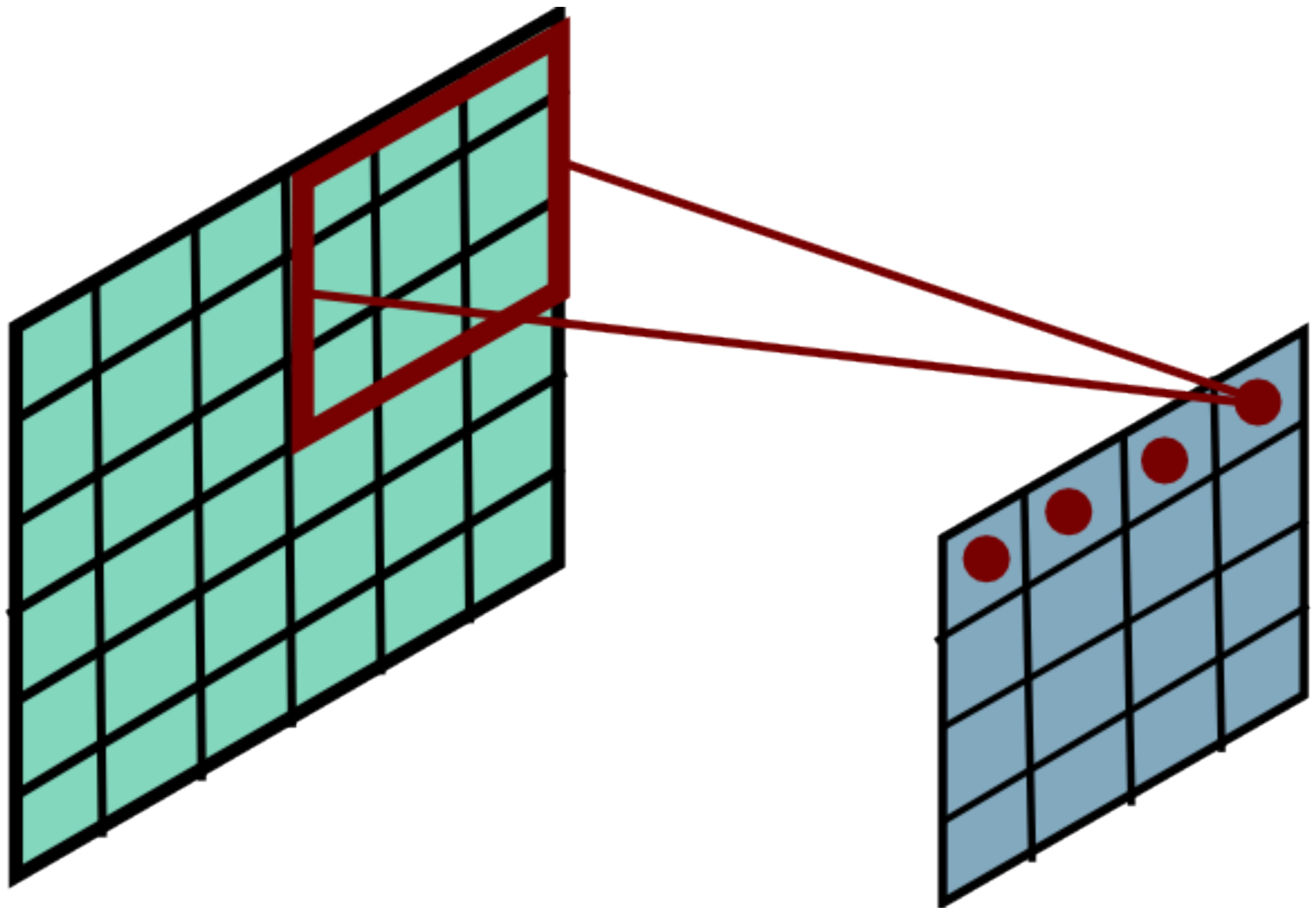
* slide from Marc'Aurelio Renzato

Convolutional Layer



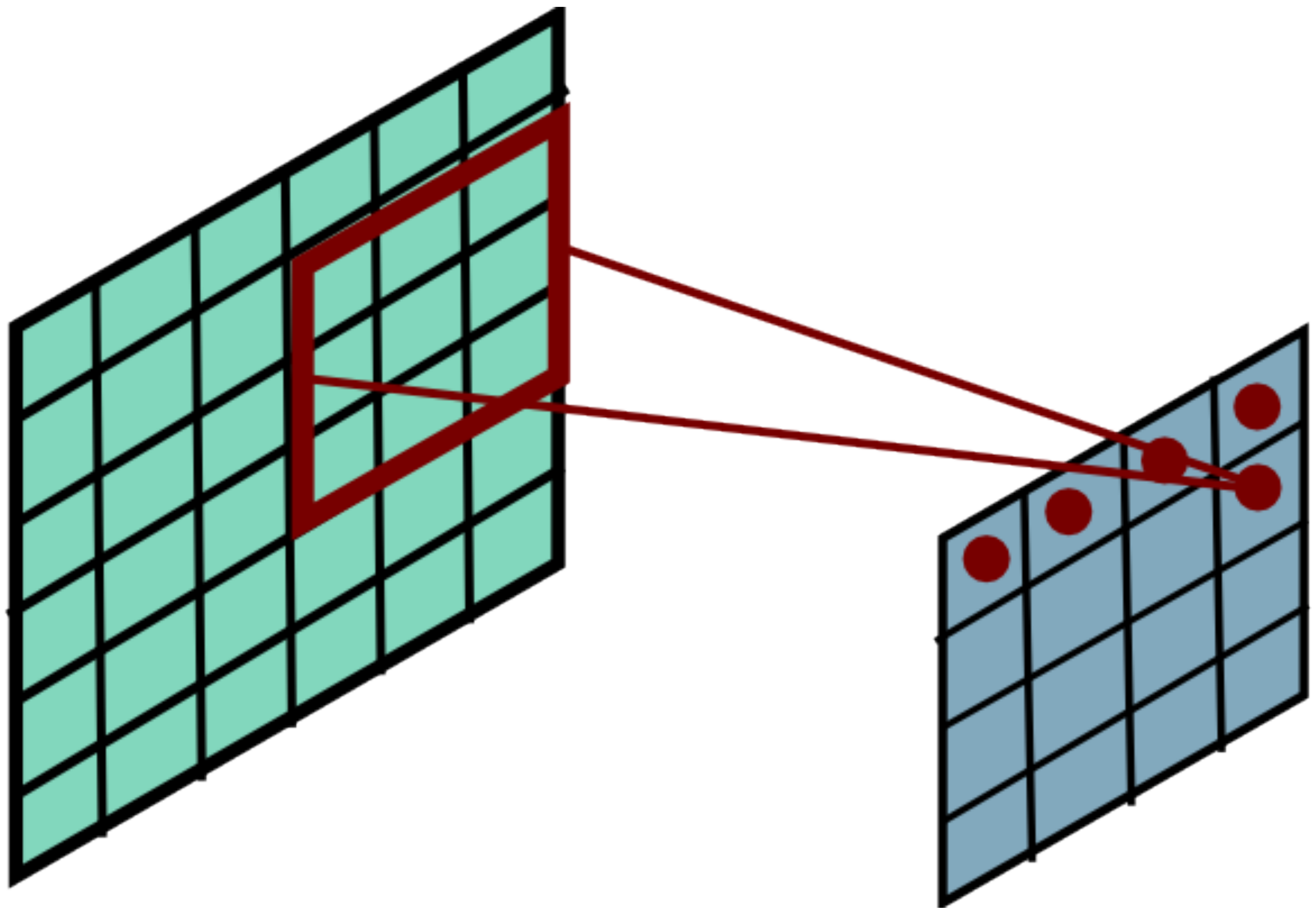
* slide from Marc'Aurelio Renzato

Convolutional Layer



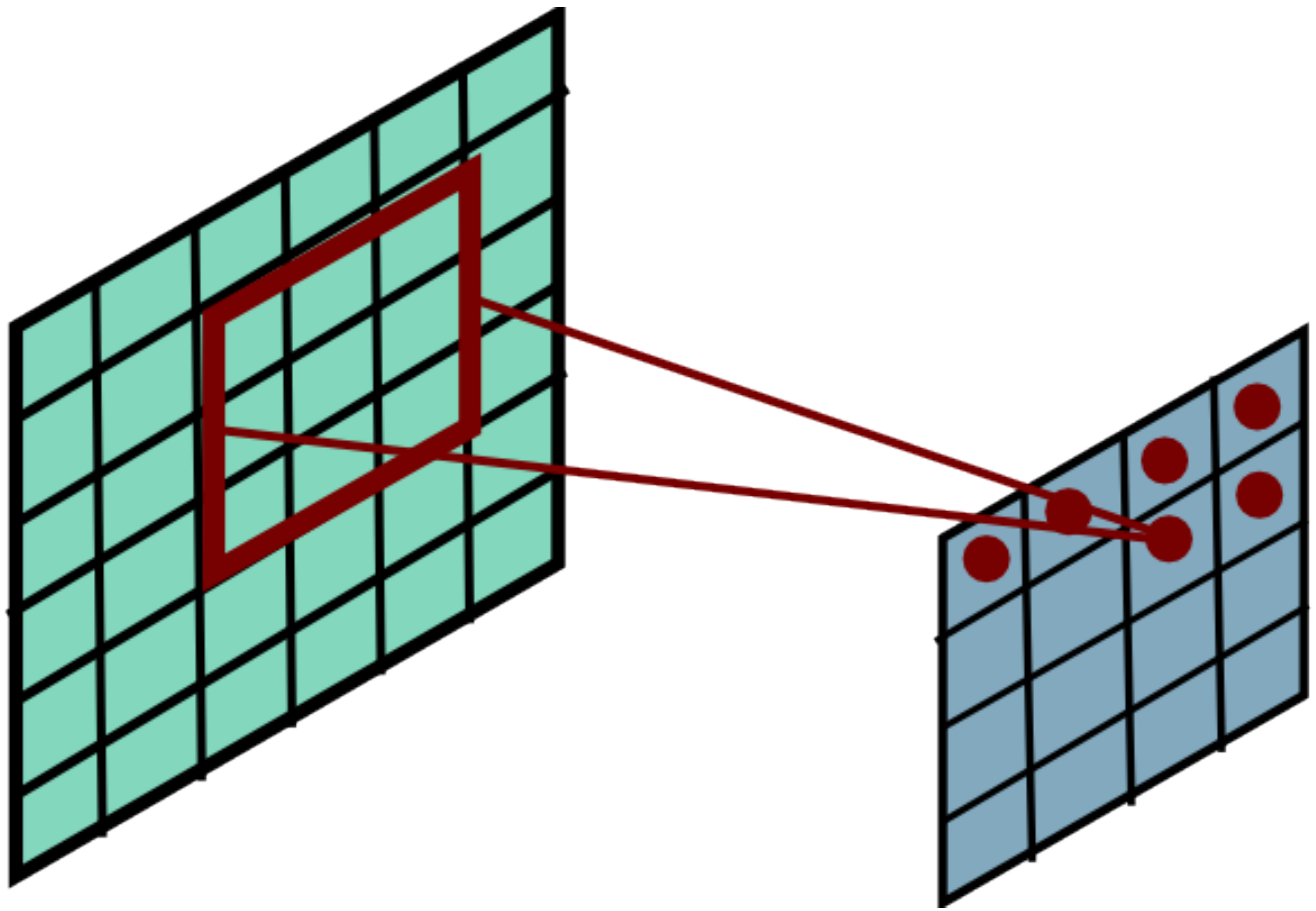
* slide from Marc'Aurelio Renzato

Convolutional Layer



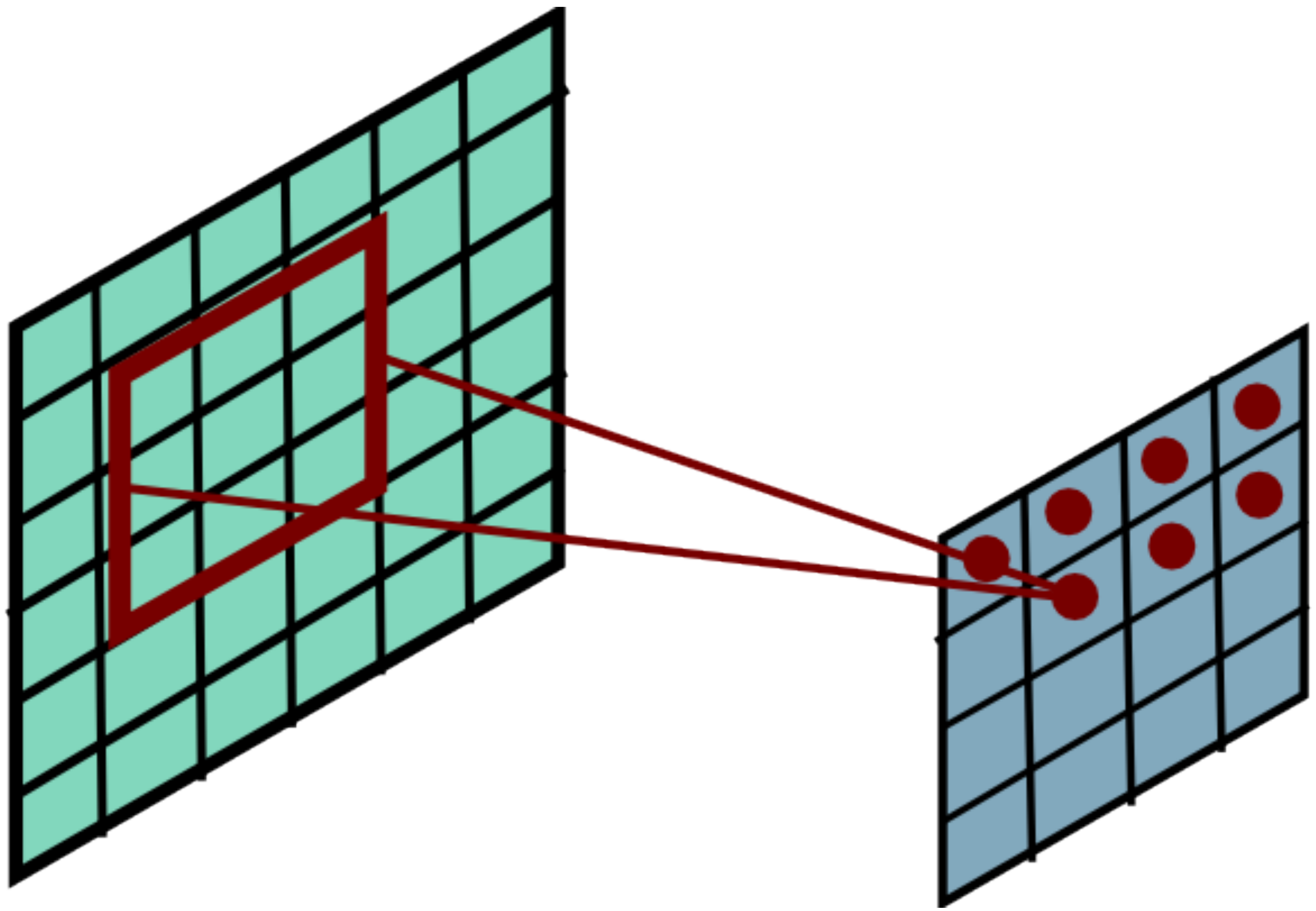
* slide from Marc’Aurelio Renzato

Convolutional Layer



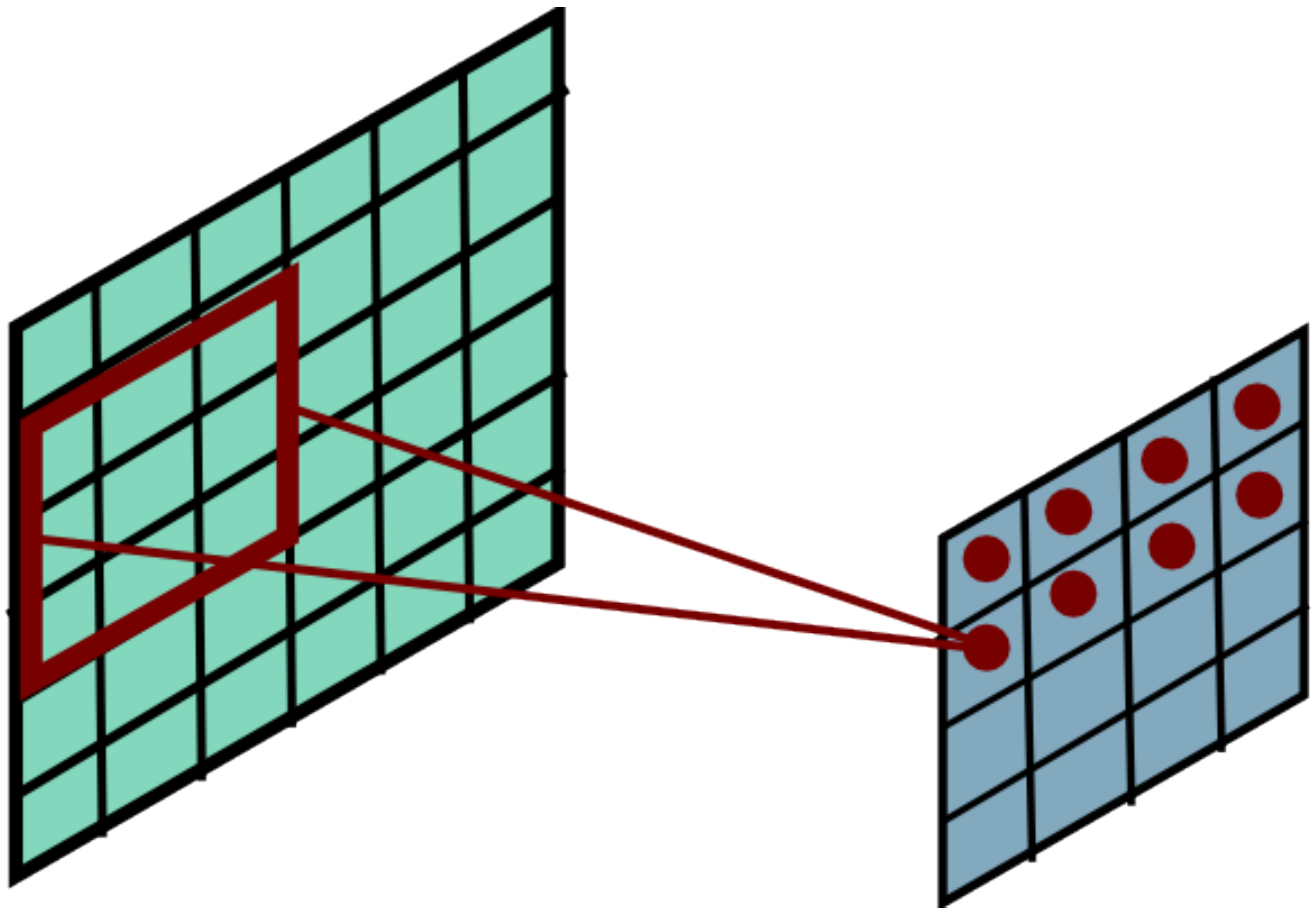
* slide from Marc'Aurelio Renzato

Convolutional Layer



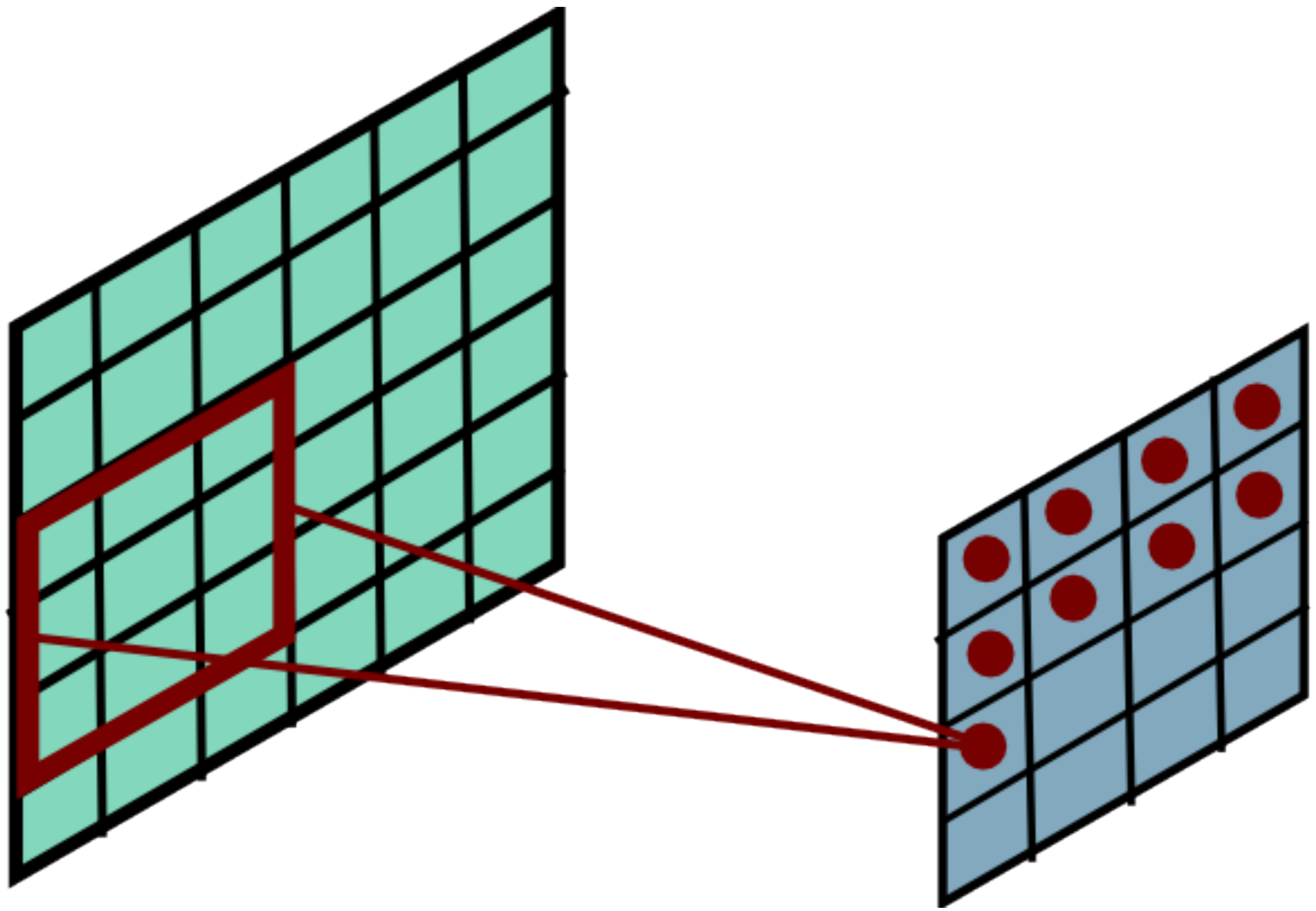
* slide from Marc'Aurelio Renzato

Convolutional Layer



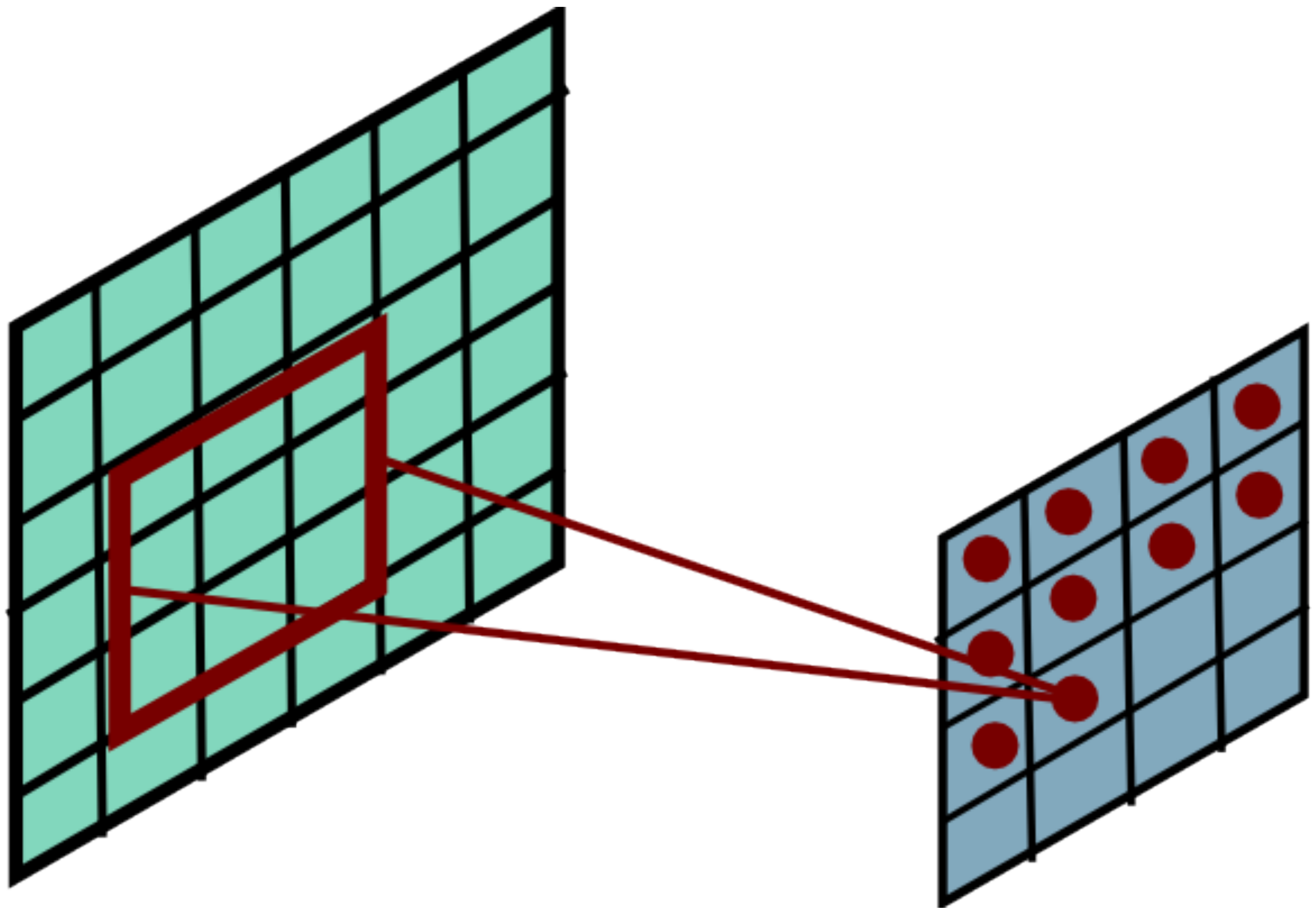
* slide from Marc'Aurelio Renzato

Convolutional Layer



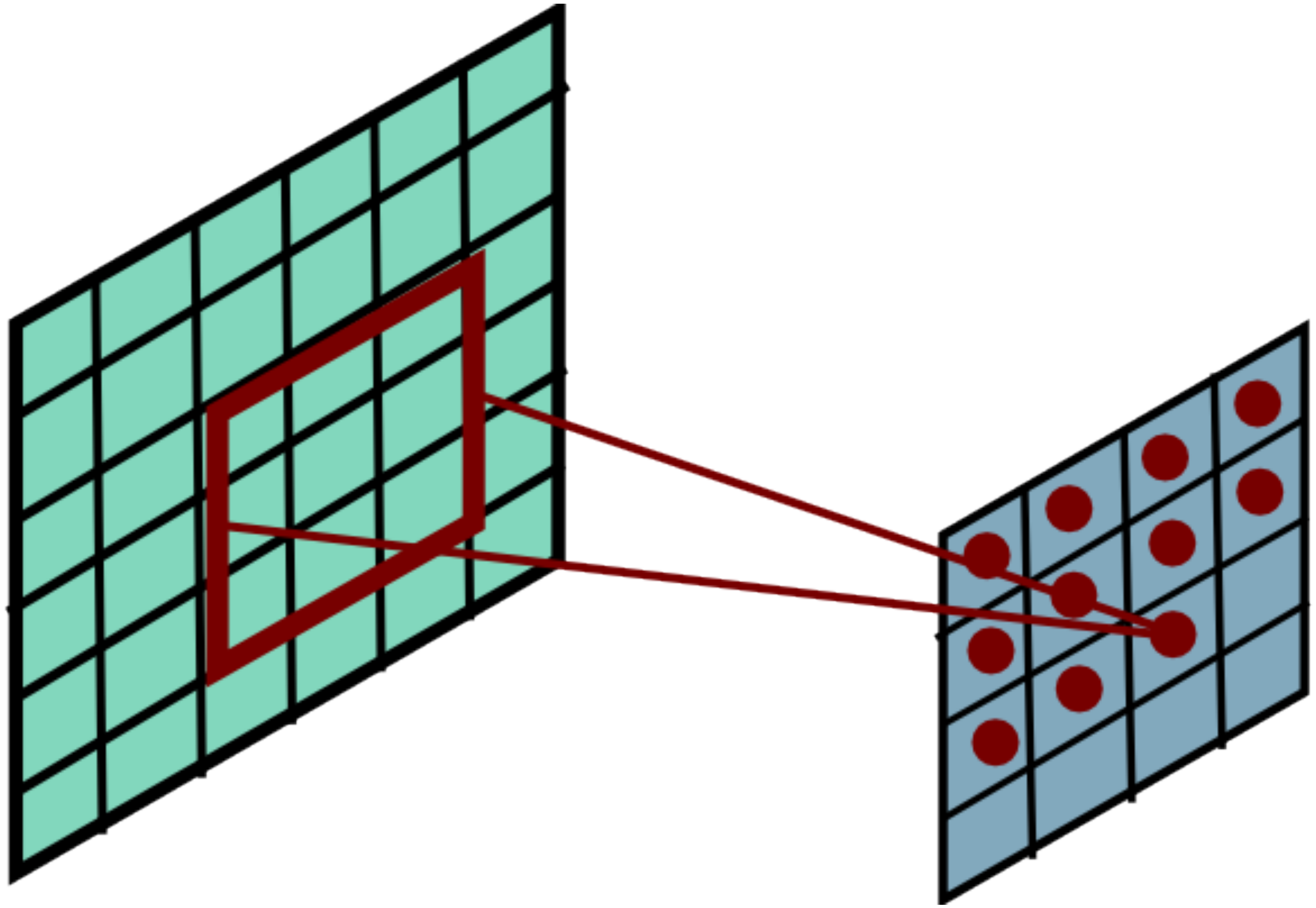
* slide from Marc'Aurelio Renzato

Convolutional Layer



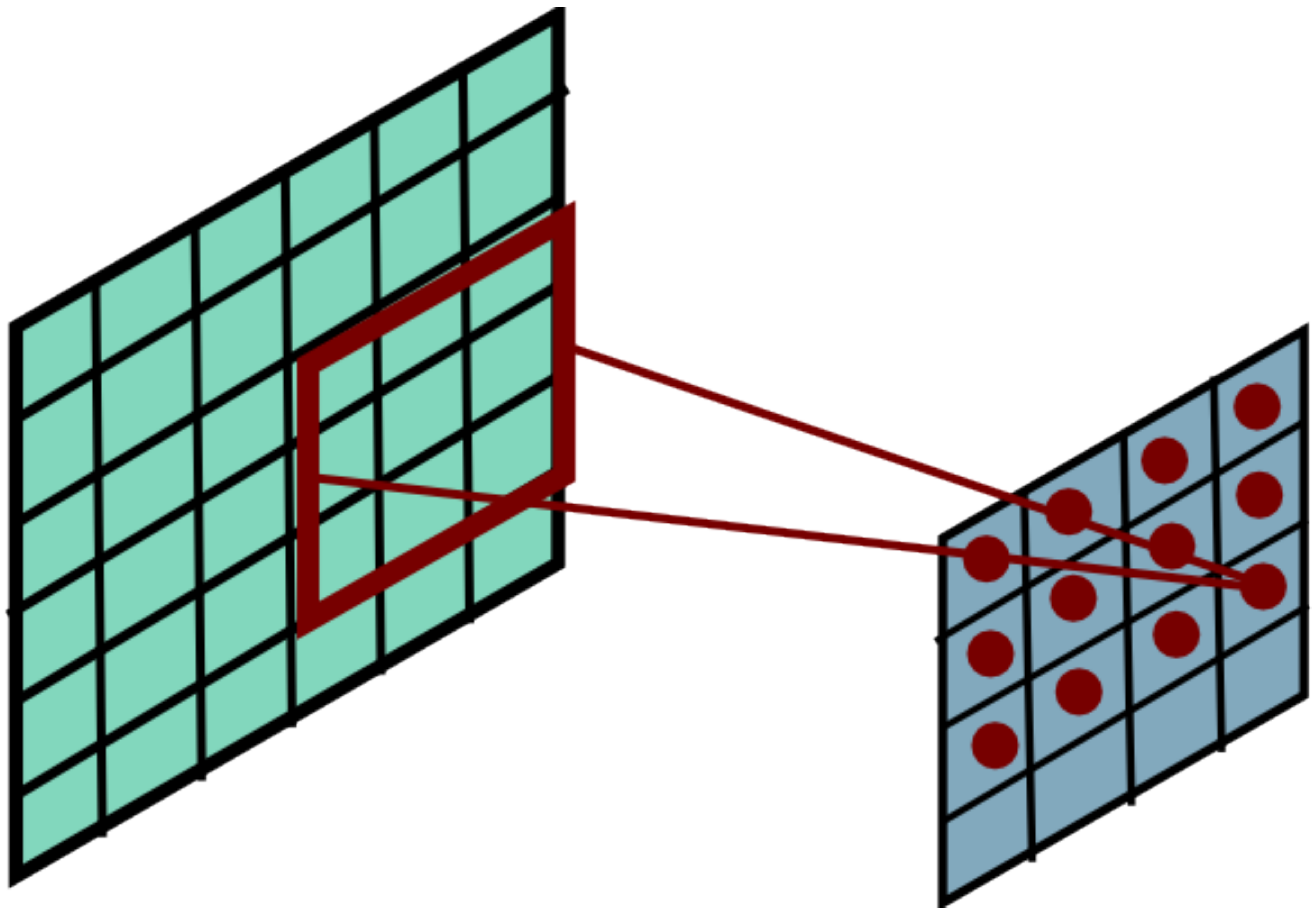
* slide from Marc'Aurelio Renzato

Convolutional Layer



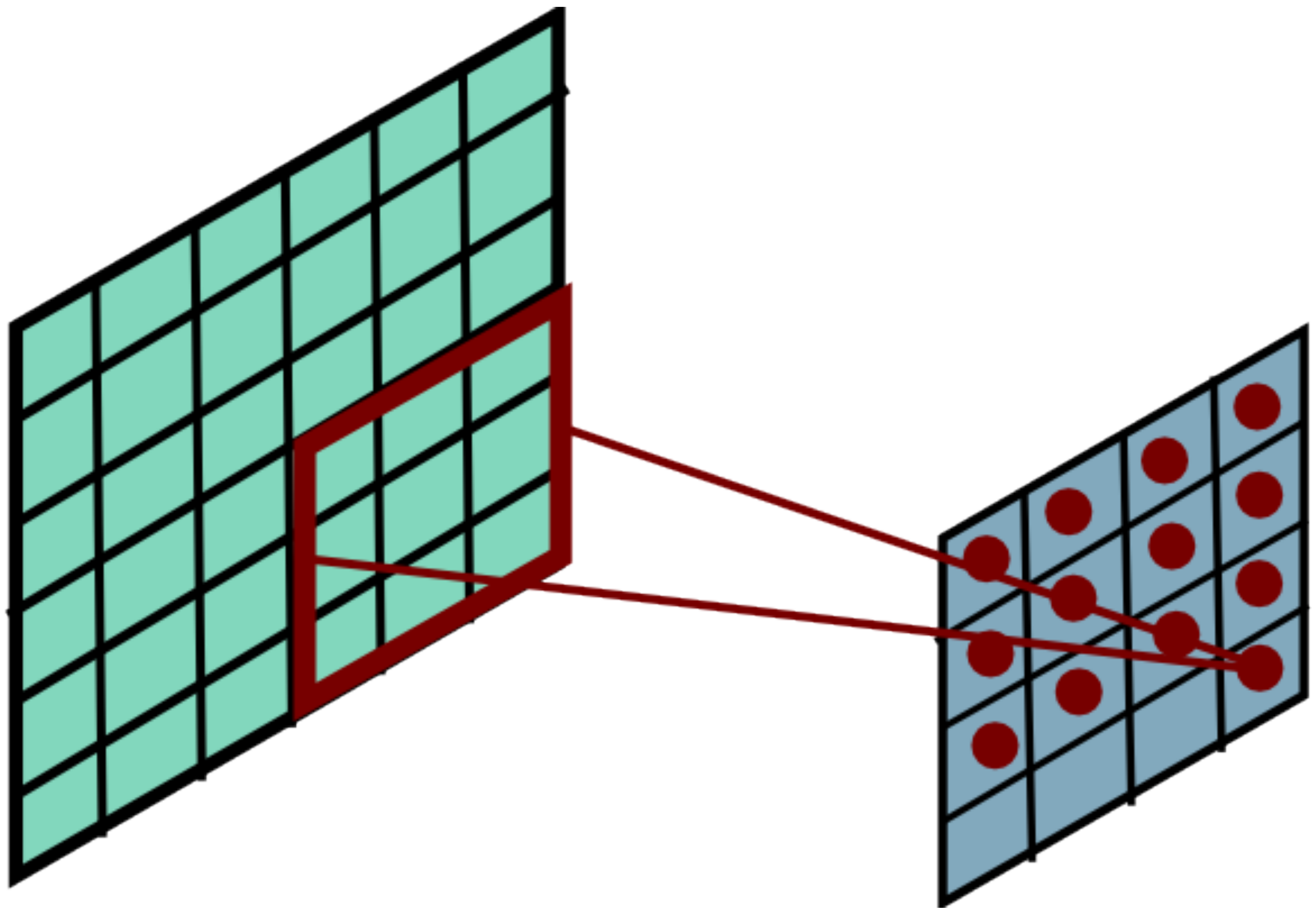
* slide from Marc’Aurelio Renzato

Convolutional Layer



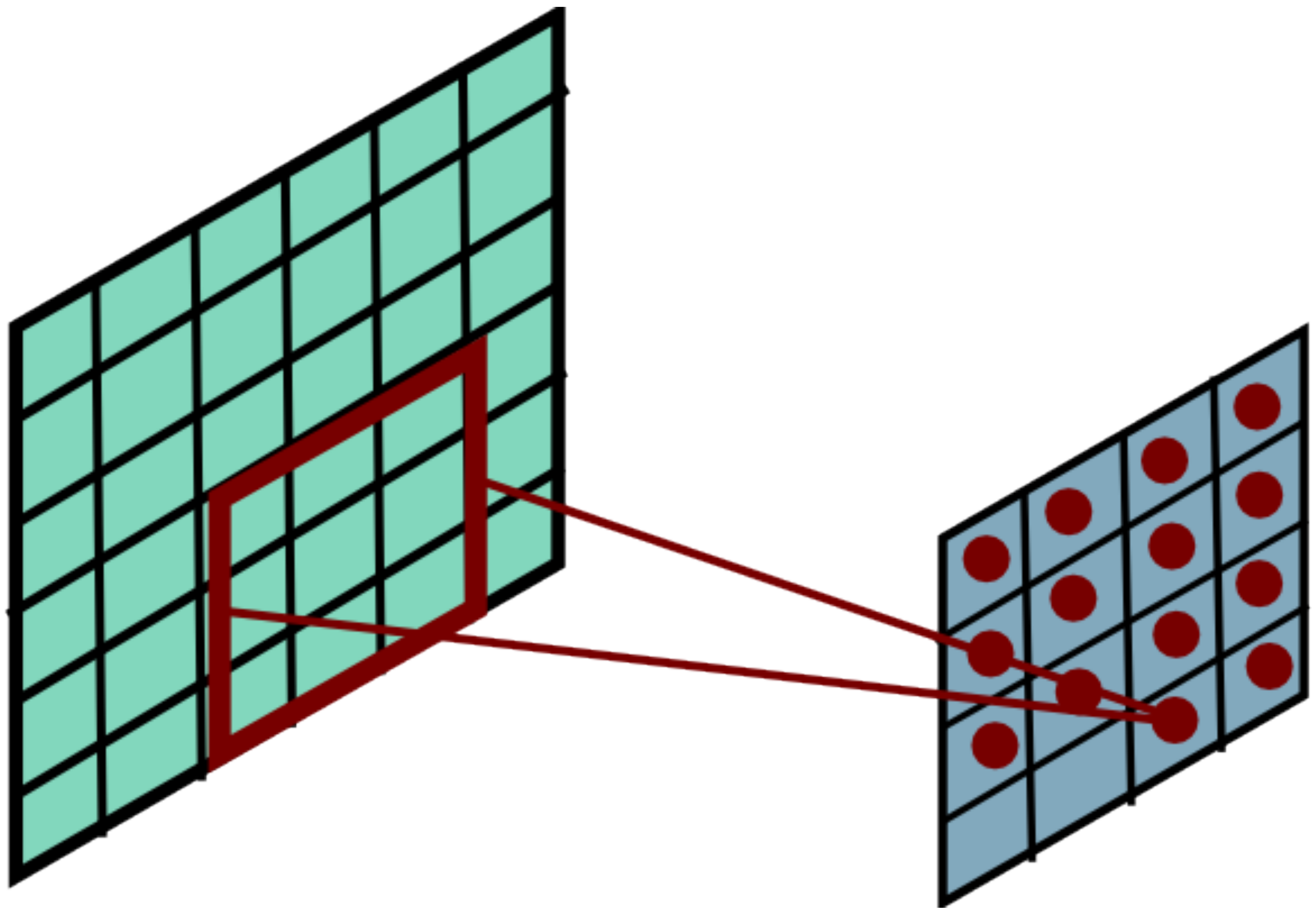
* slide from Marc'Aurelio Renzato

Convolutional Layer



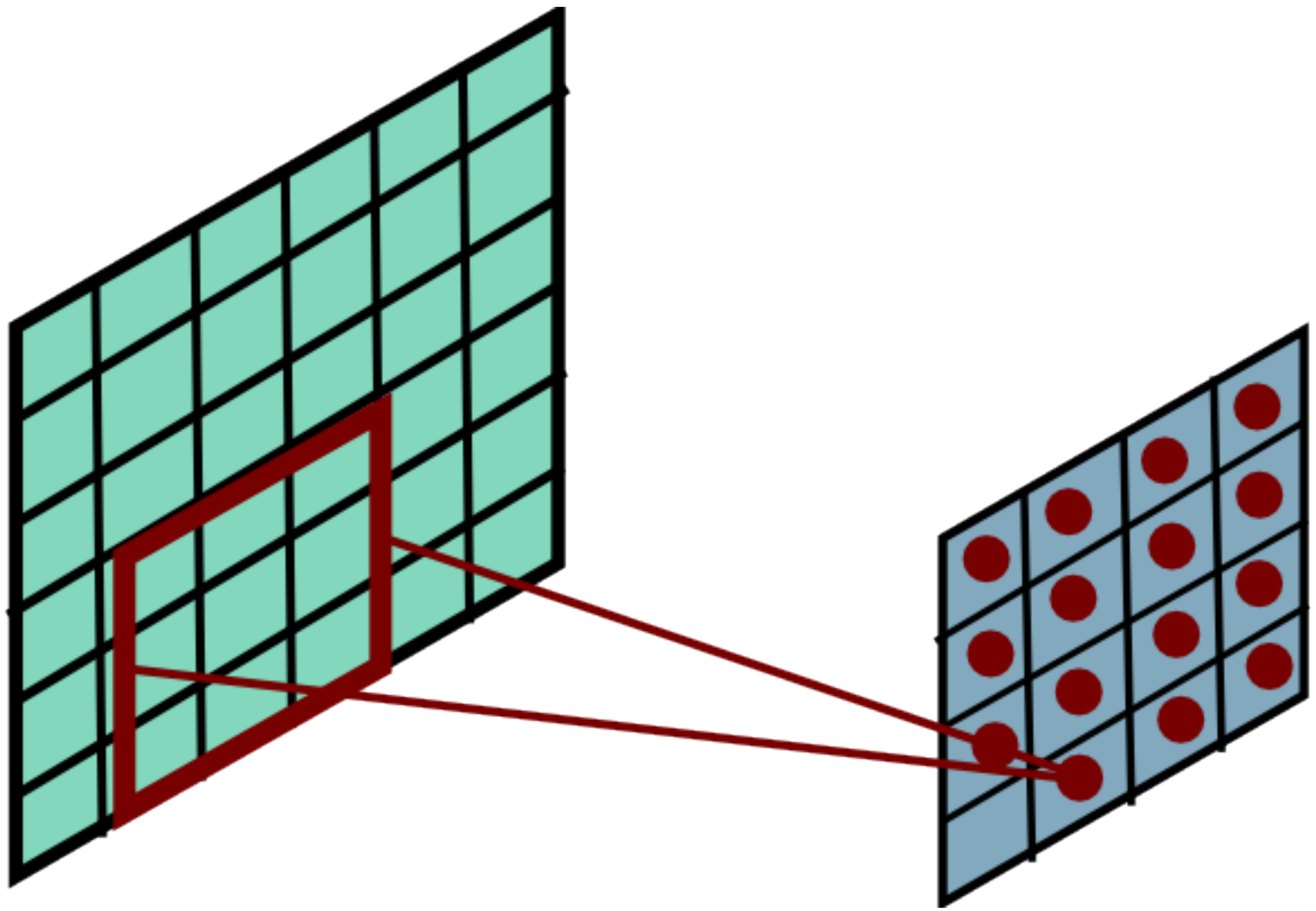
* slide from Marc'Aurelio Renzato

Convolutional Layer



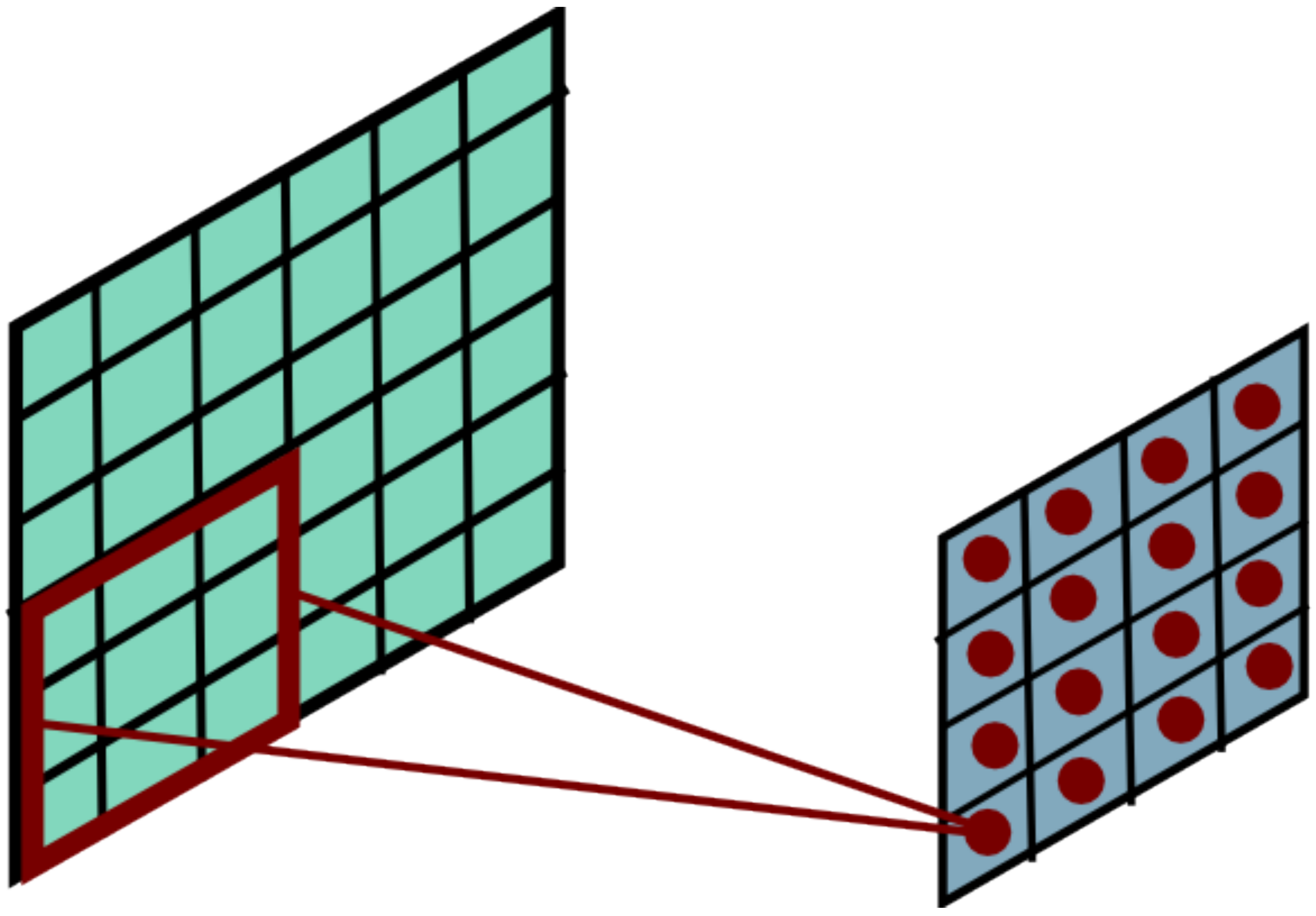
* slide from Marc'Aurelio Renzato

Convolutional Layer



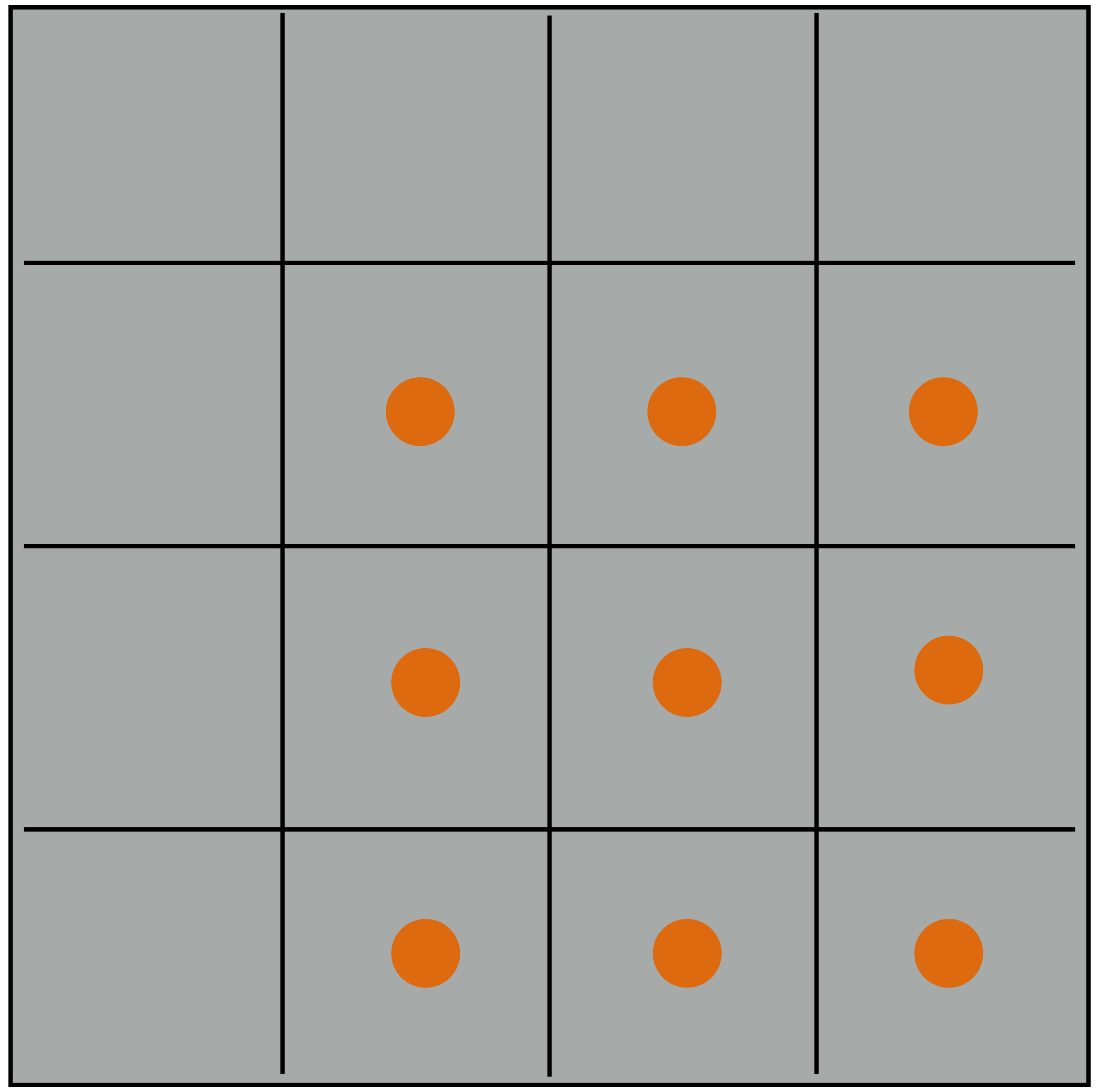
* slide from Marc'Aurelio Renzato

Convolutional Layer

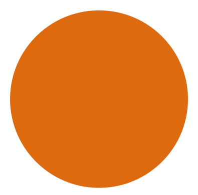


* slide from Marc'Aurelio Renzato

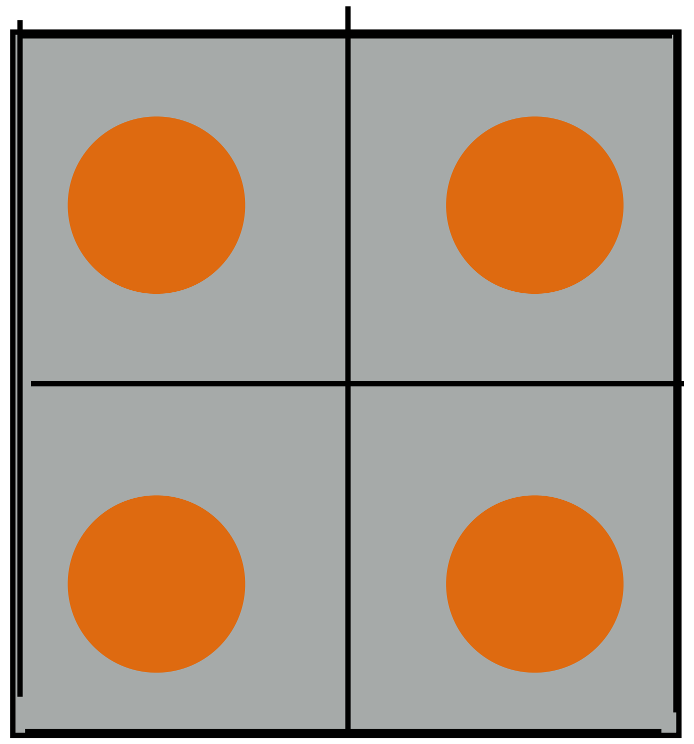
Convolutional Layer: Interpretation #2



One neuron applied as convolution (by shifting)



neurons



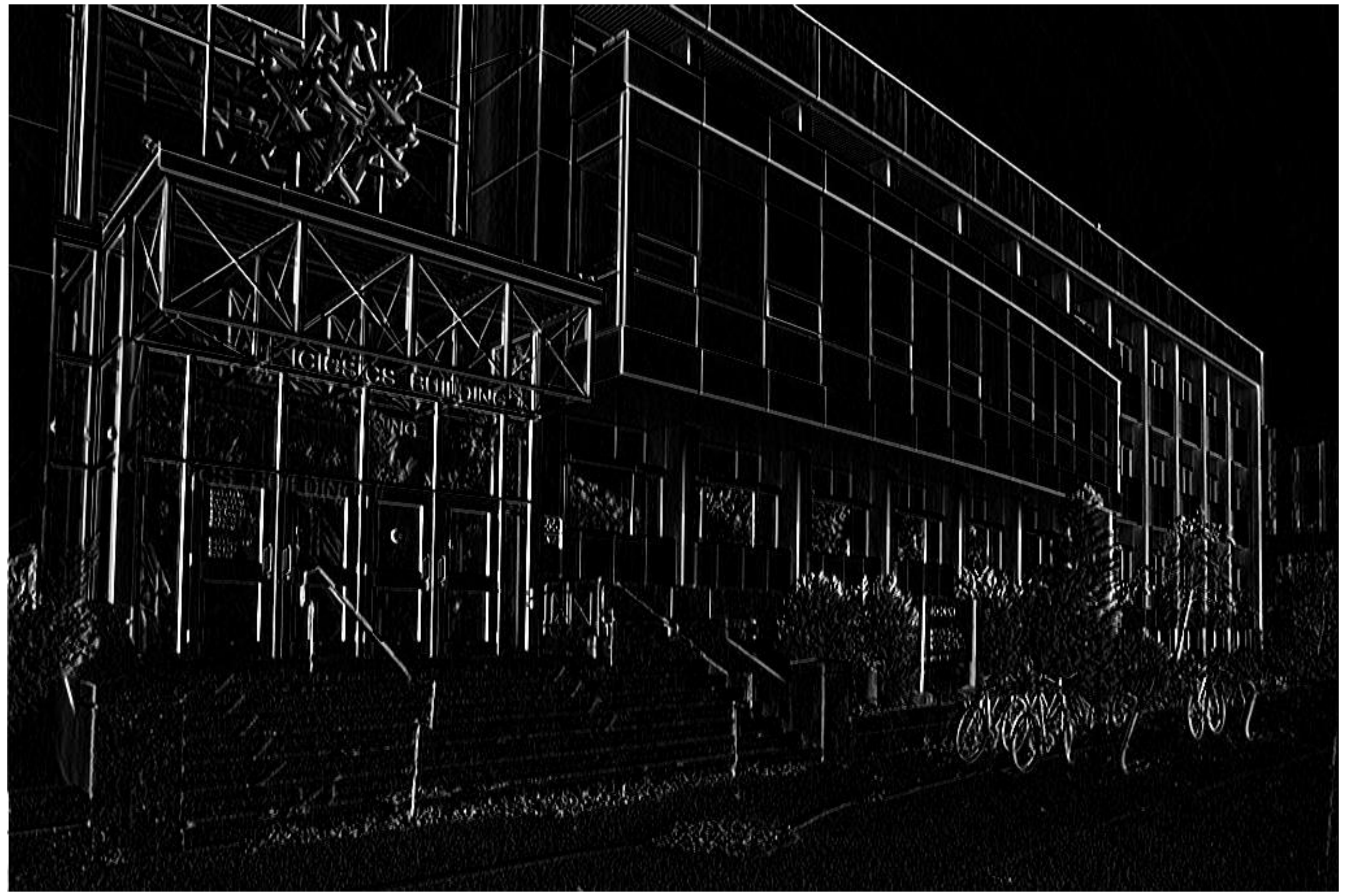
output

$$\sigma\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\mathbf{W}_{i,j}\mathcal{I}(i,j)+b\right)$$

Similar to Filter in Normalized Correlation

Convolution Layer

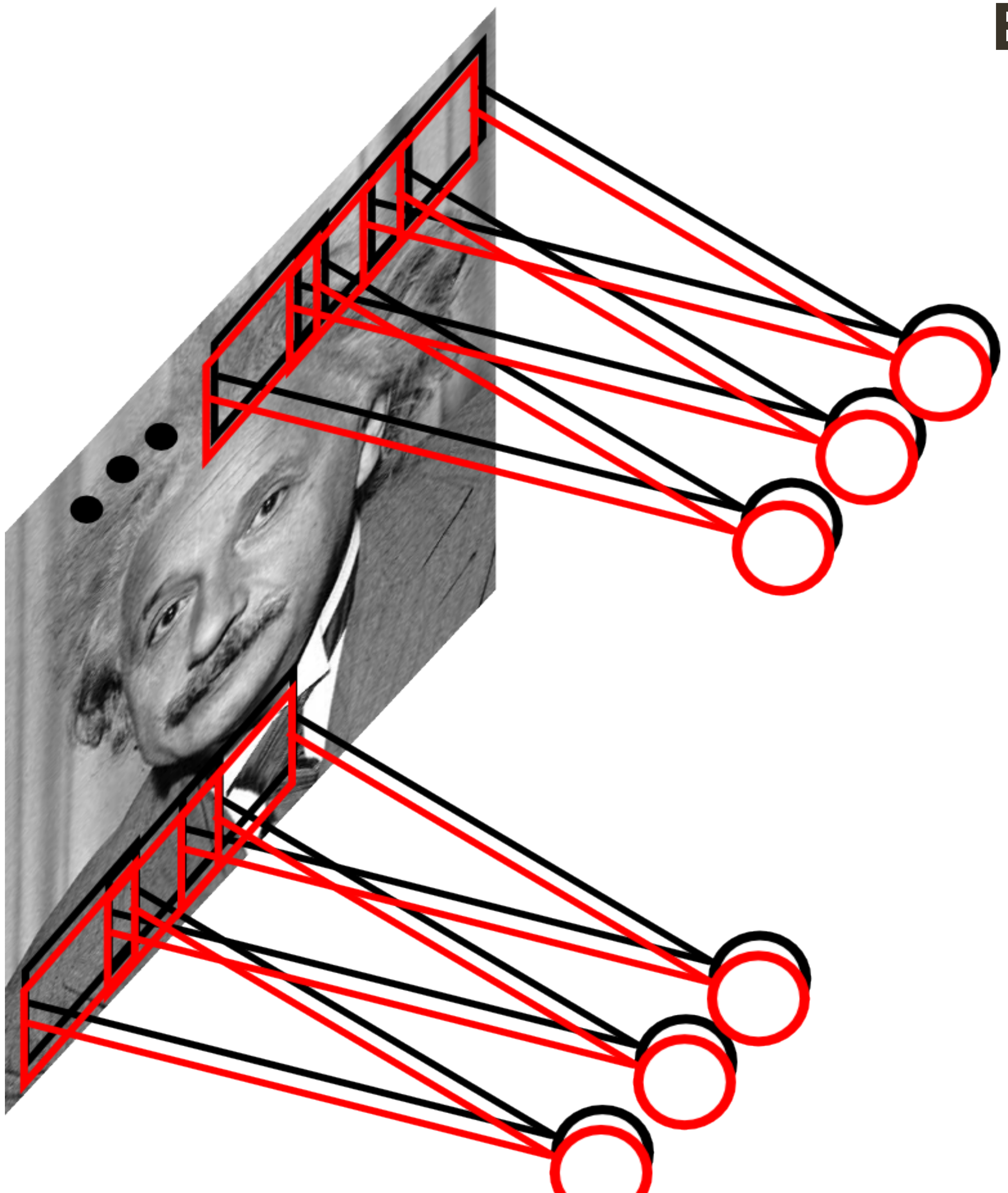
$$\star \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \longrightarrow$$



Convolution Layer

$$\star \begin{bmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{bmatrix} \rightarrow$$

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

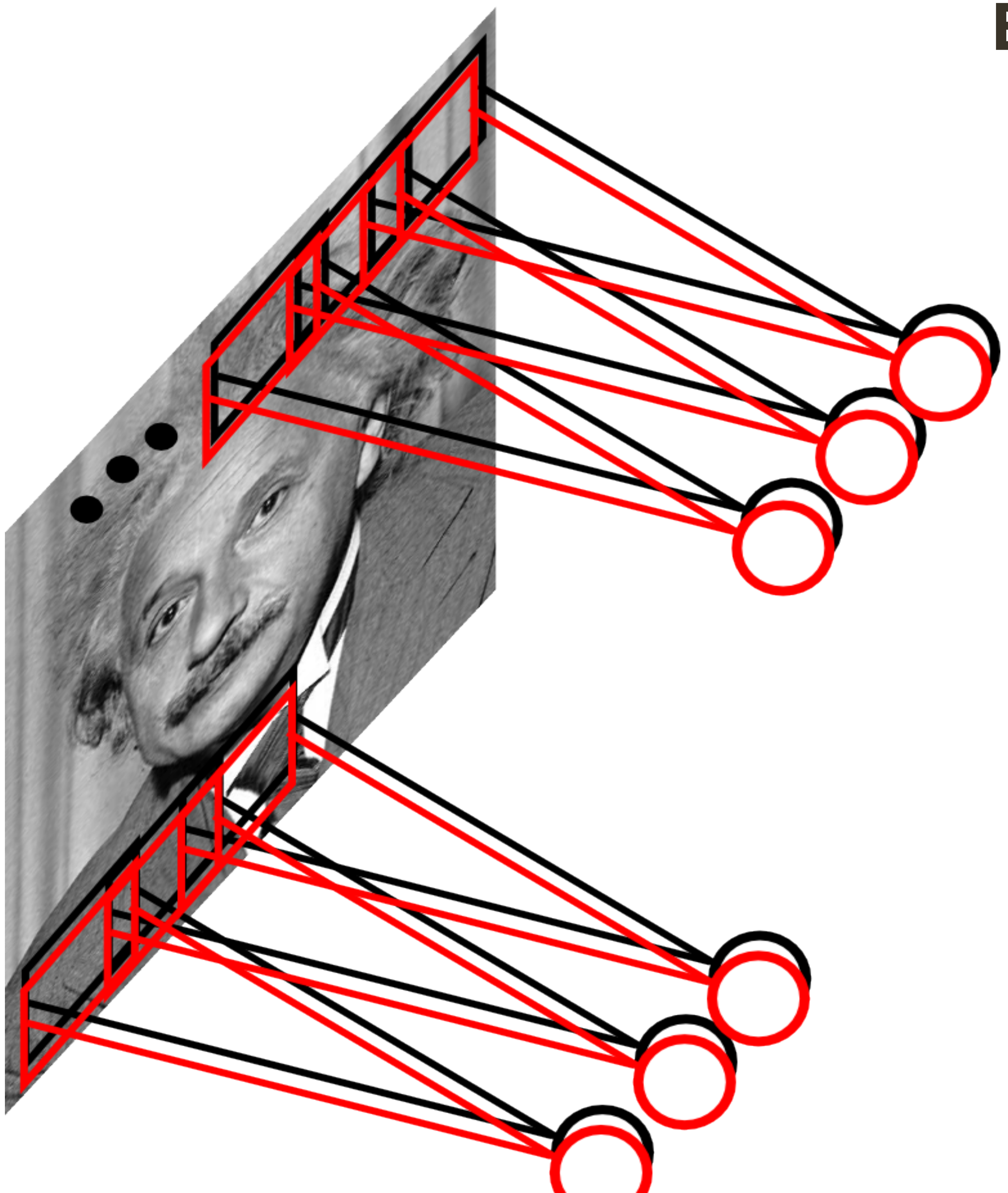
Filter size: 10 x 10

# of filters: 20

Learn **multiple filters**

* slide from Marc'Aurelio Renzato

Convolutional Layer



Example: 200 x 200 image (small)
x 40K hidden units

Filter size: 10 x 10

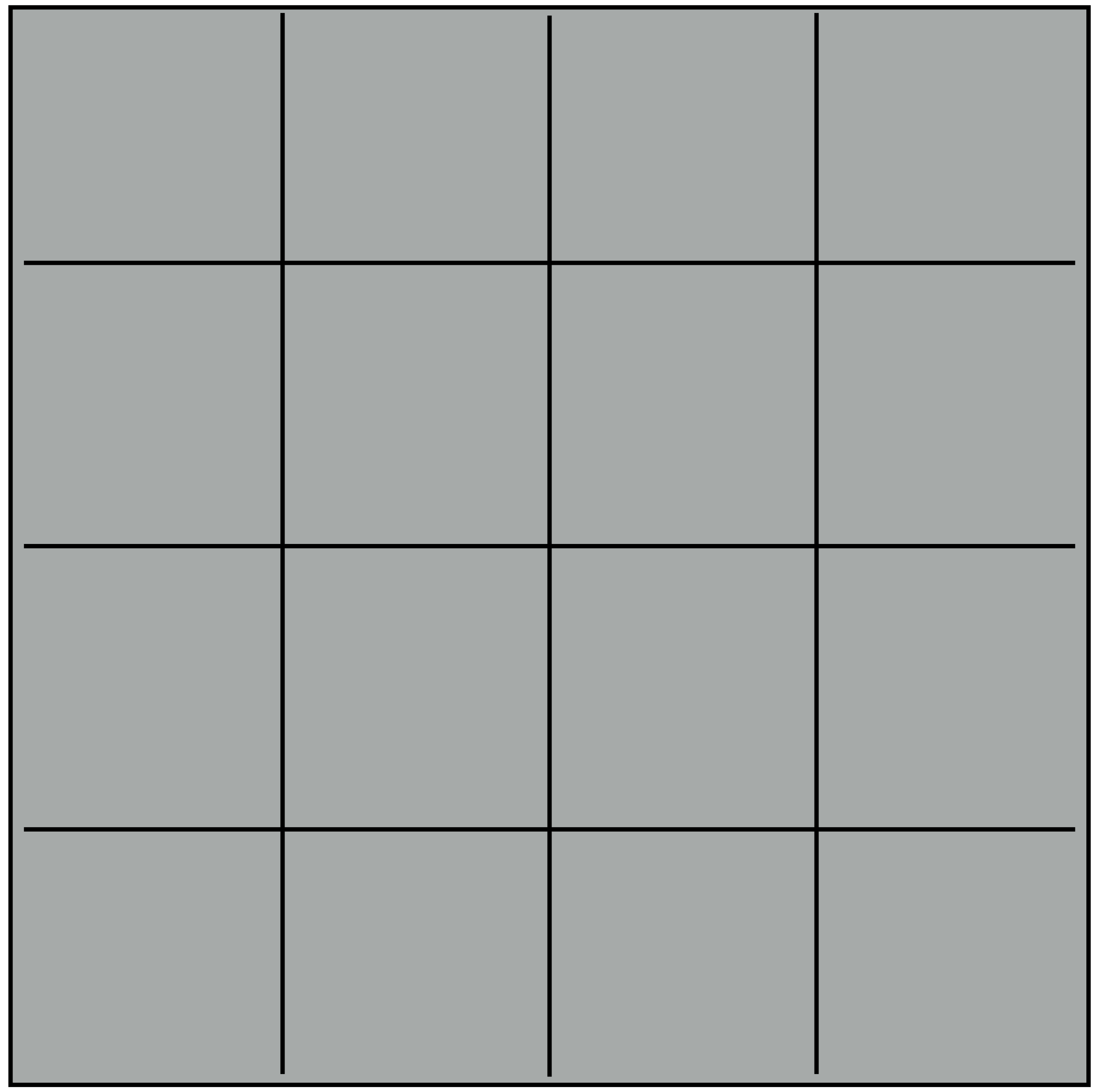
# of filters: 20

= 2000 parameters

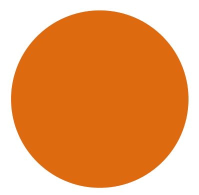
Learn **multiple filters**

* slide from Marc'Aurelio Renzato

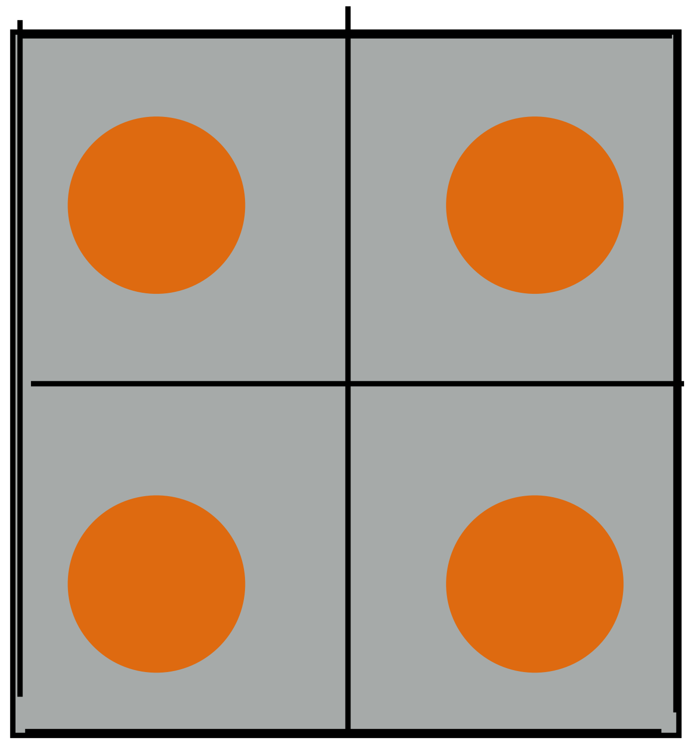
Convolutional Layer: Interpretation #2



One neuron applied as convolution (by shifting)



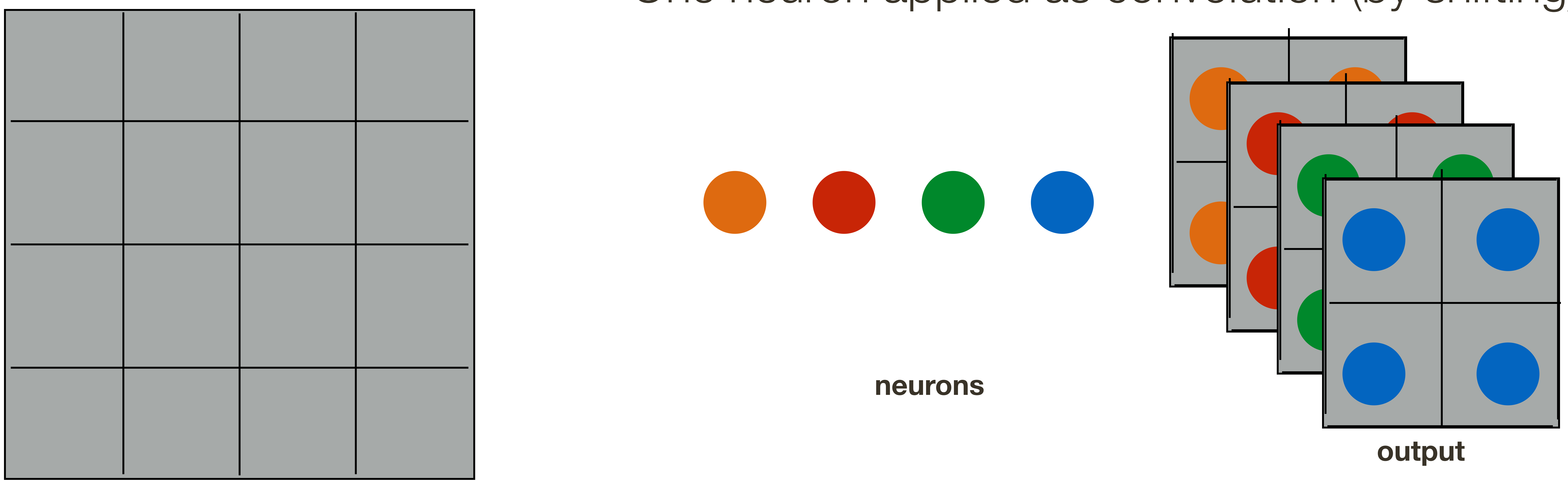
neurons



output

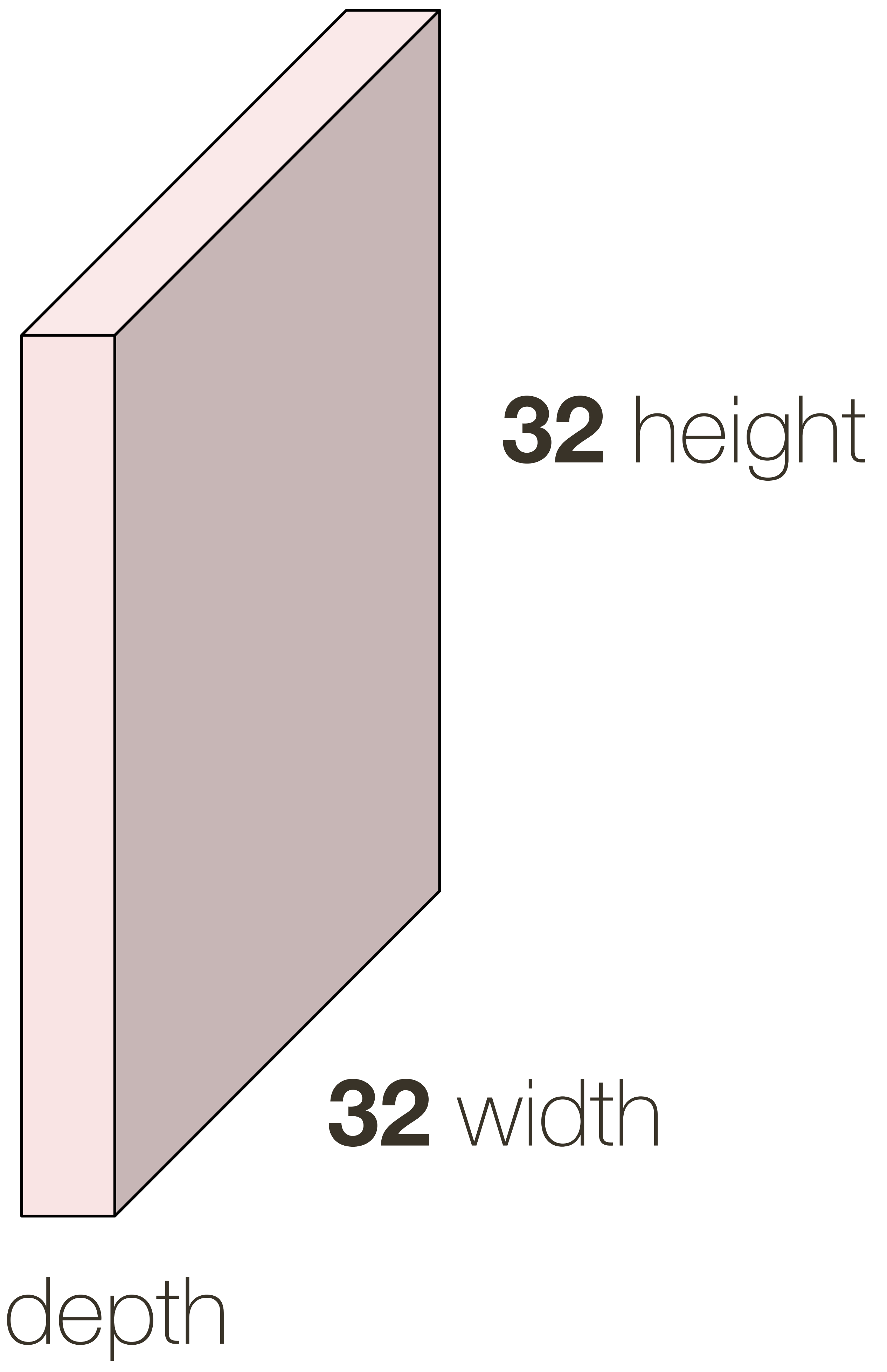
Convolutional Layer: Interpretation #2

One neuron applied as convolution (by shifting)

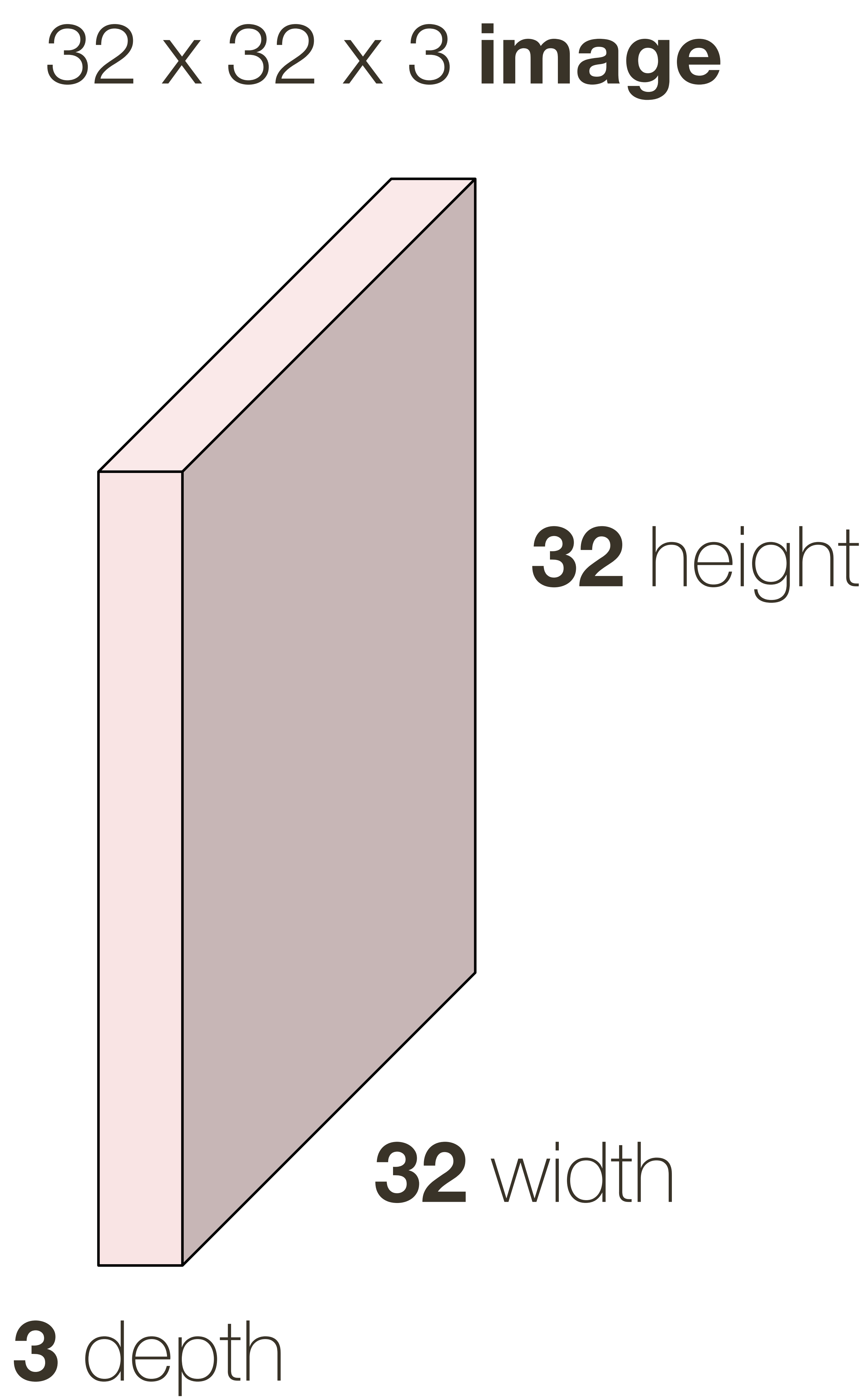


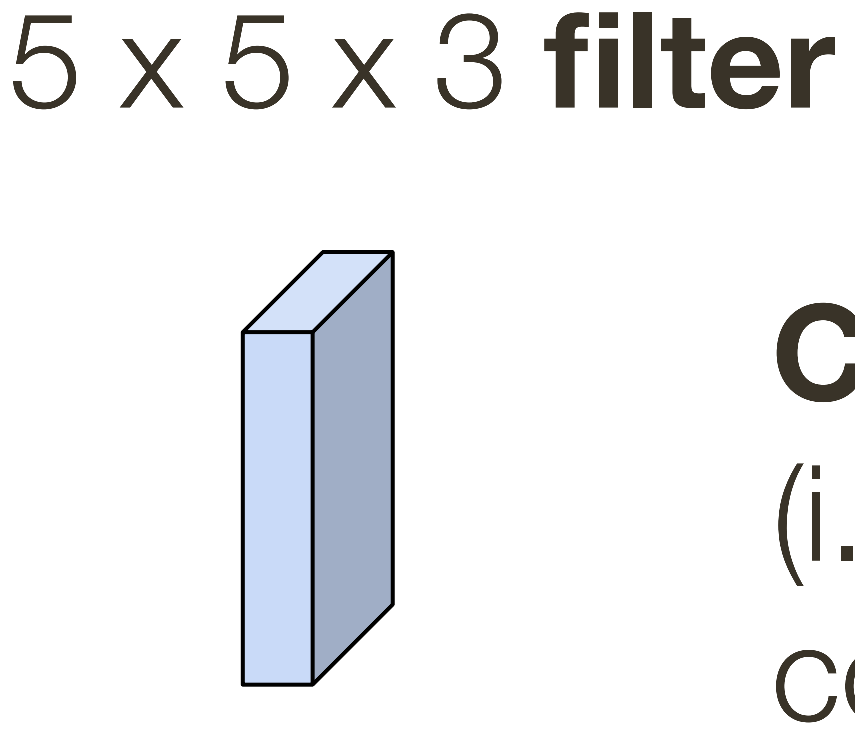
Convolutional Layer

32 x 32 x 3 **image** (note the image preserves spatial structure)



Convolutional Layer



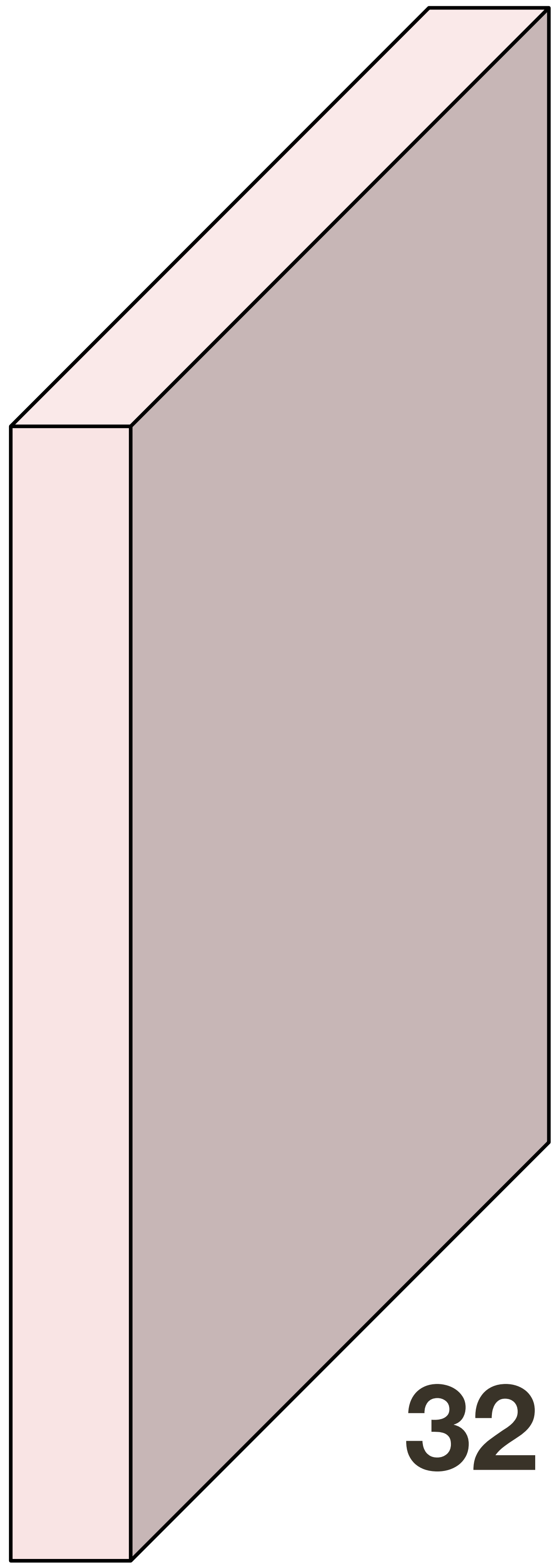


Convolve the filter with the image (i.e., "slide over the image spatially, computing dot products")

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer

32 x 32 x 3 **image**



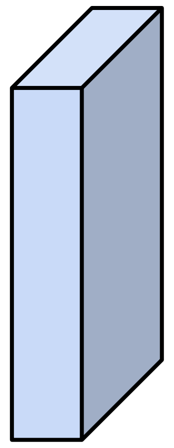
32 height

32 width

3 depth

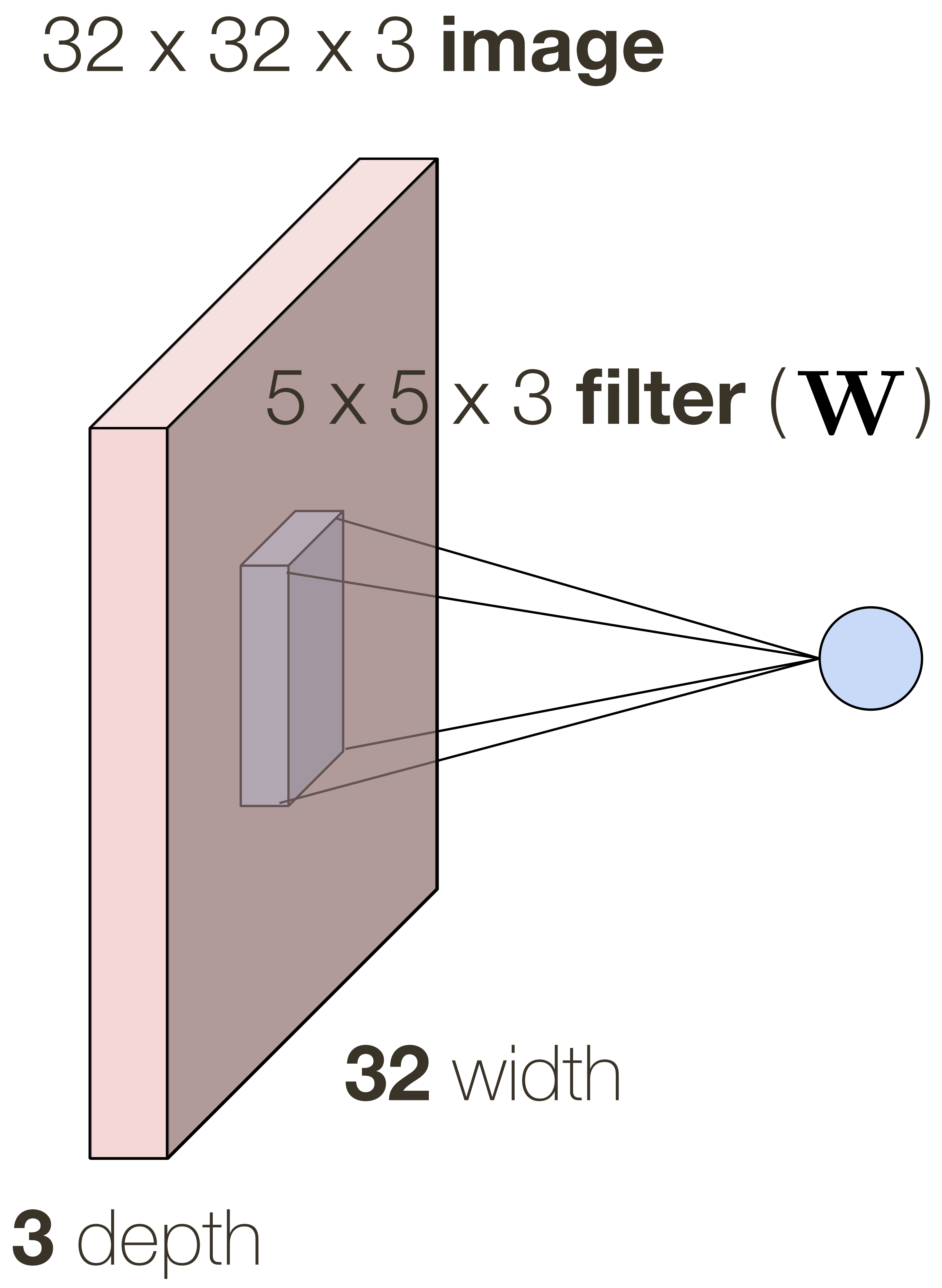
Filters always extend the full depth of the input volume

5 x 5 x 3 **filter**



Convolve the filter with the image (i.e., “slide over the image spatially, computing dot products”)

Convolutional Layer

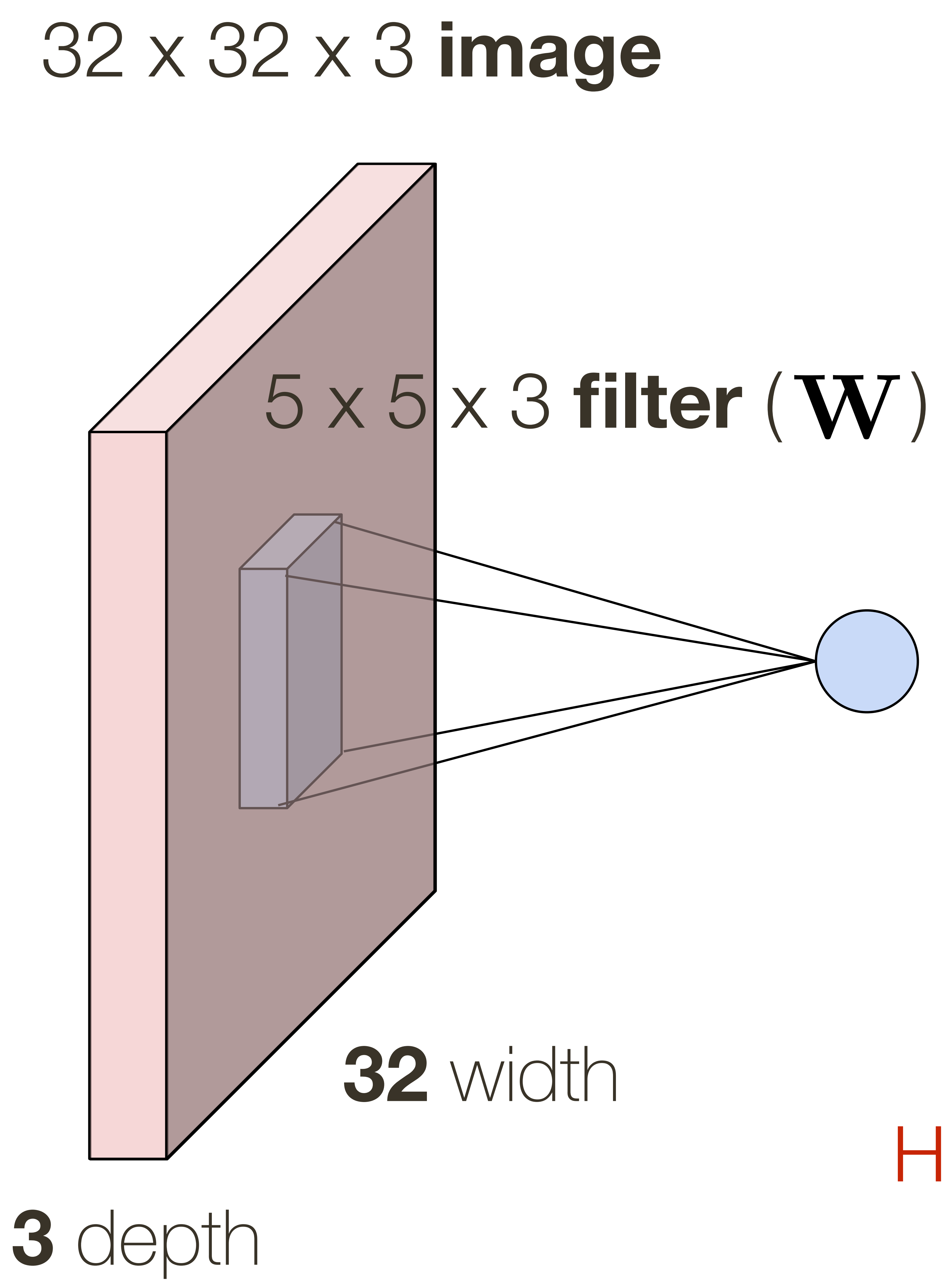


1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer



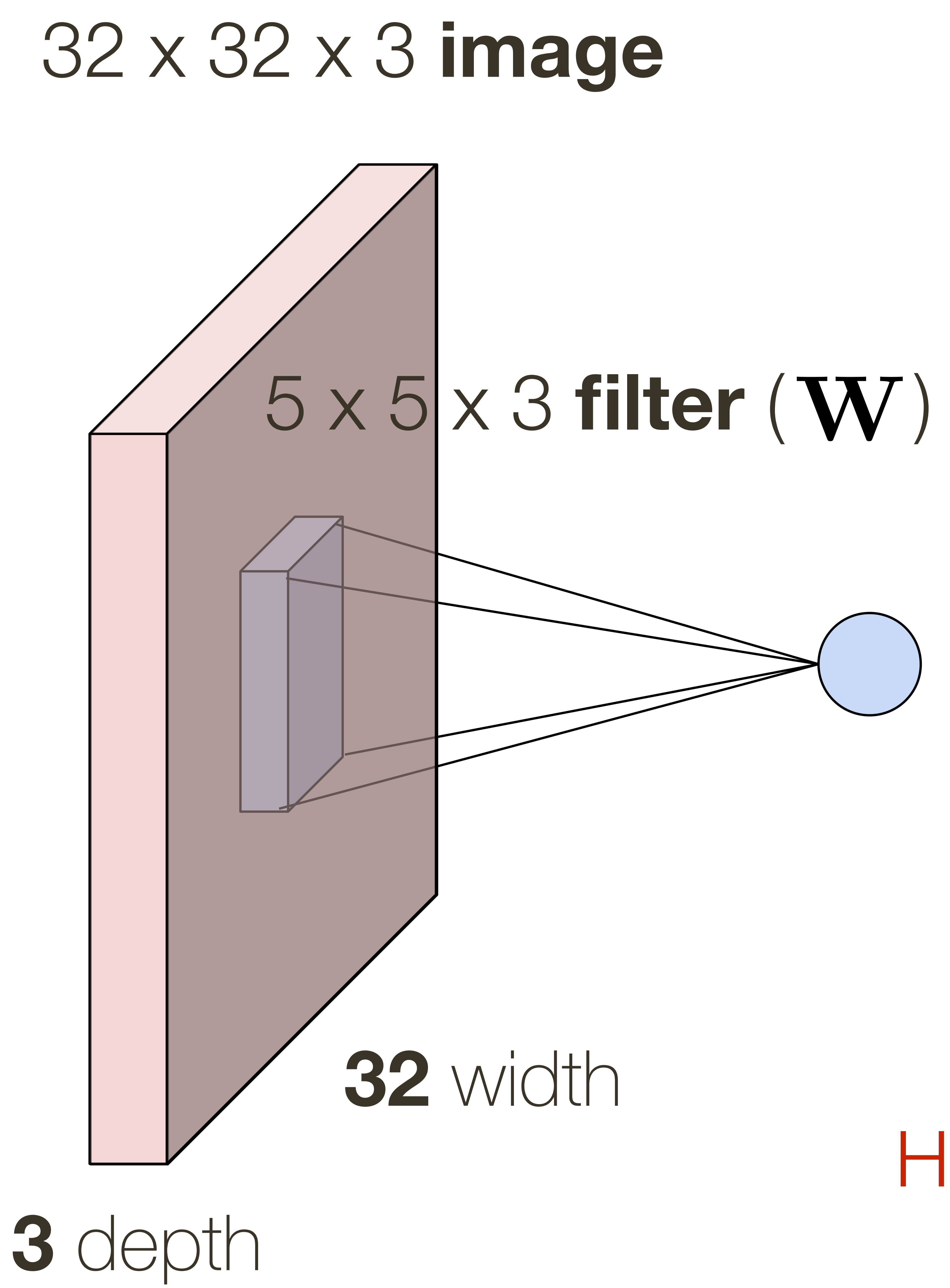
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

How many **parameters** does the layer have?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer



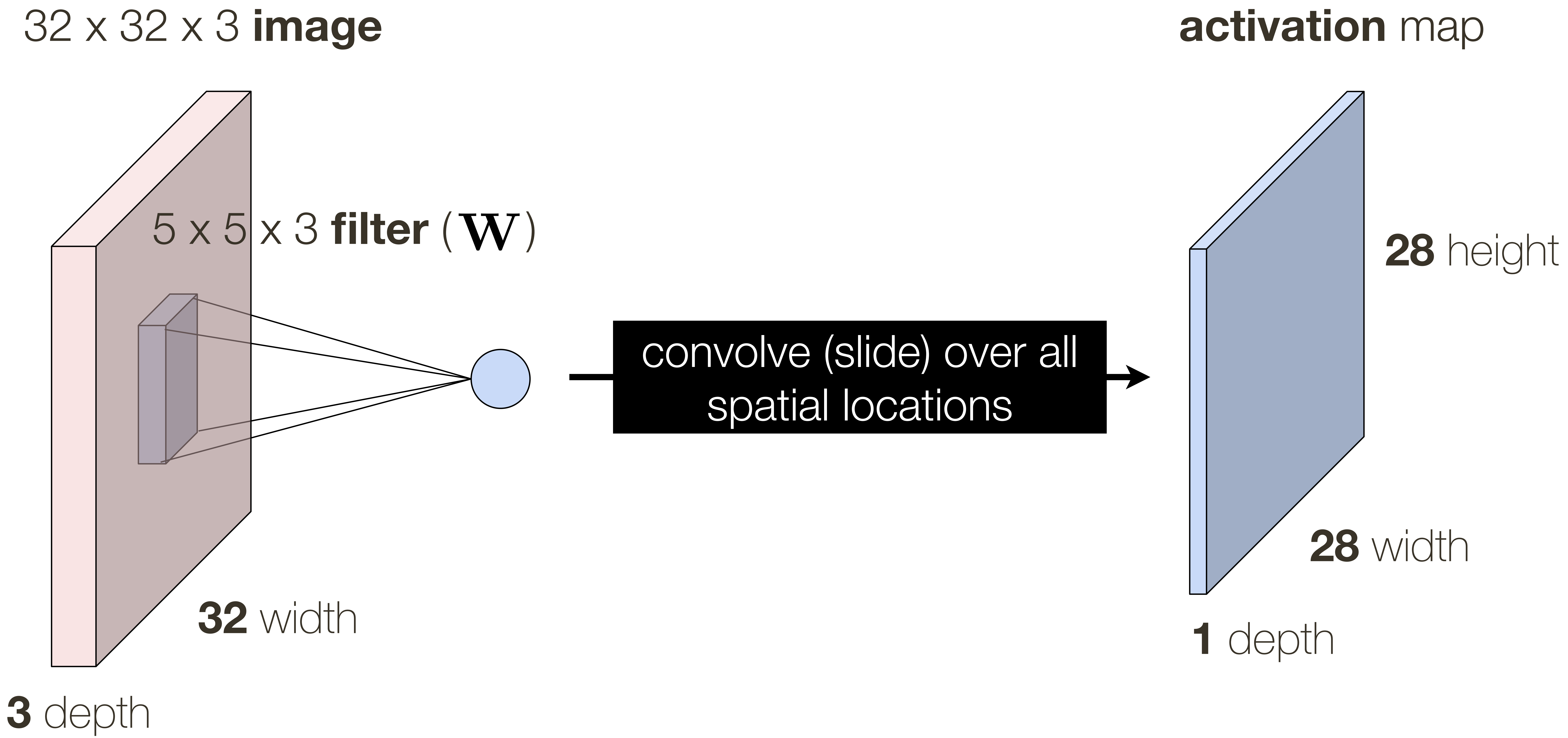
1 number: the result of taking a dot product between the filter and a small 5 x 5 x 3 part of the image

$$\mathbf{W}^T\mathbf{x} + b, \text{where } \mathbf{W}, \mathbf{x} \in \mathbb{R}^{75}$$

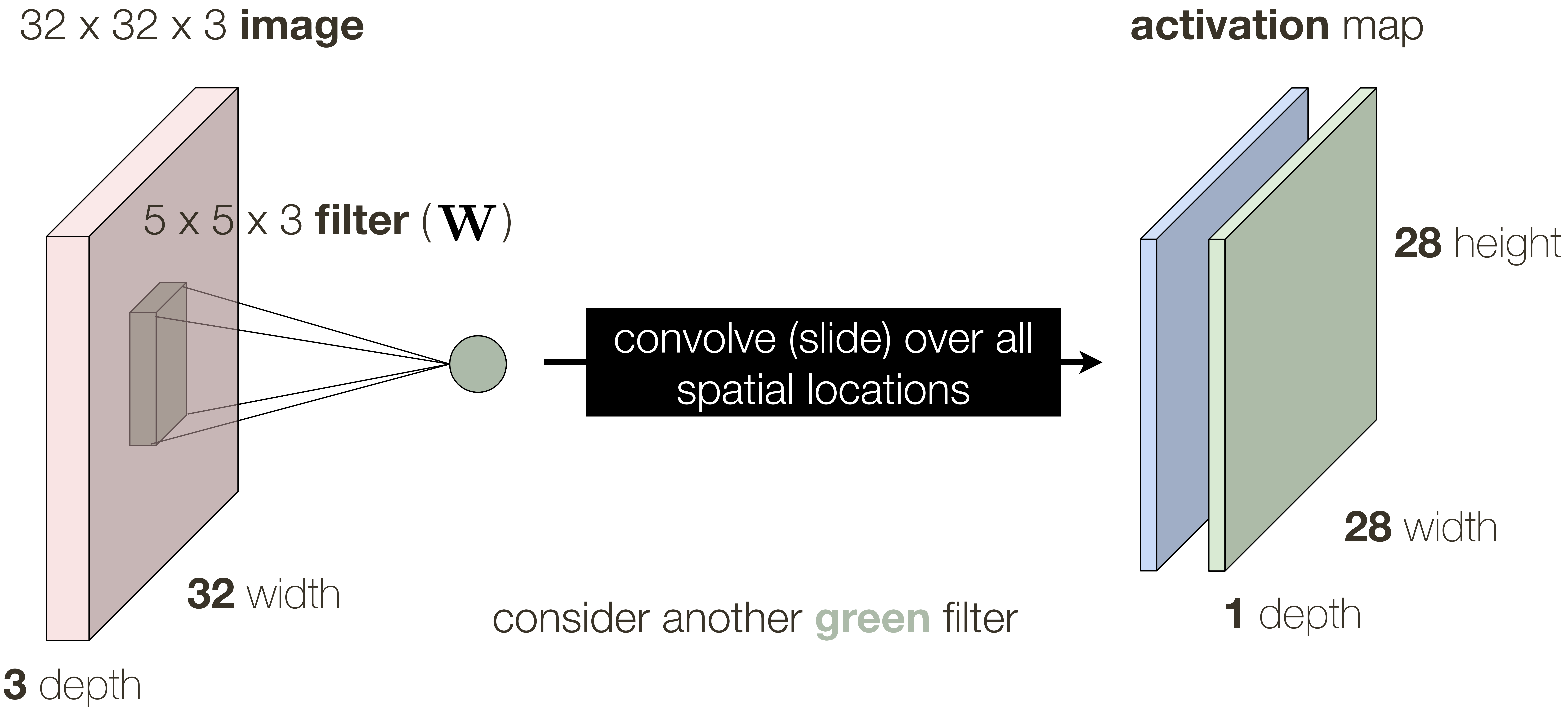
How many **parameters** does the layer have? **76**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer



Convolutional Layer

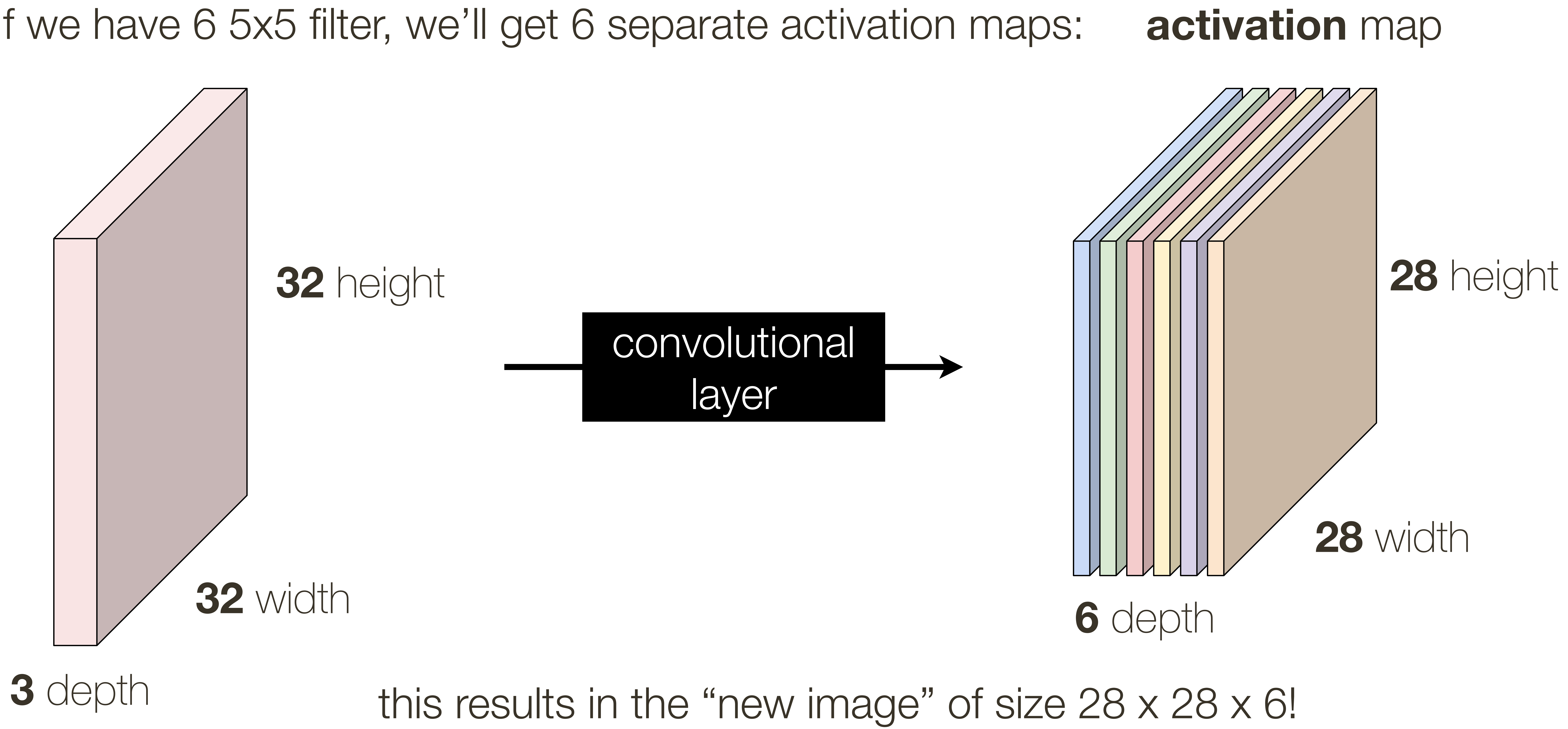


consider another **green** filter

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

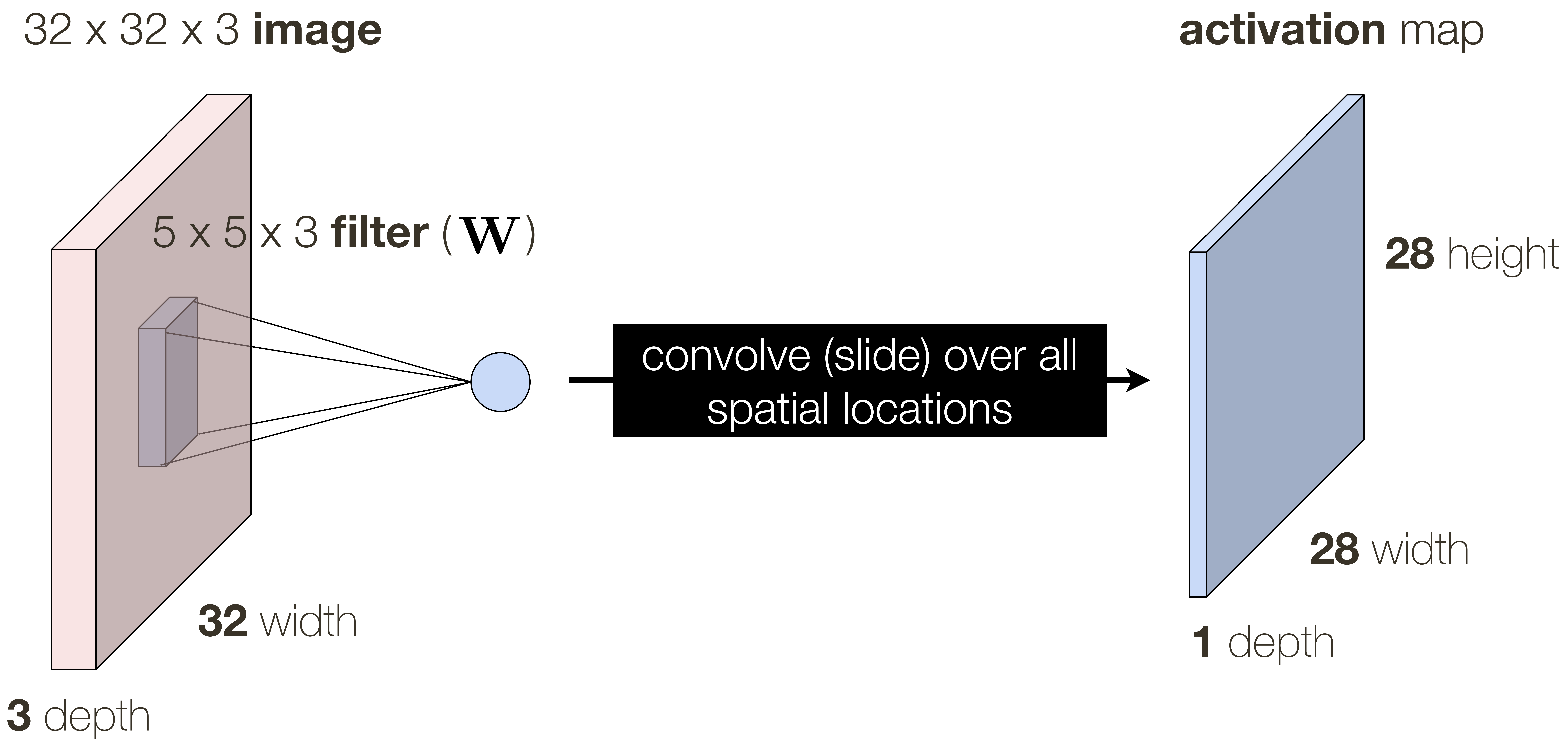
Convolutional Layer

If we have 6 5x5 filter, we'll get 6 separate activation maps:



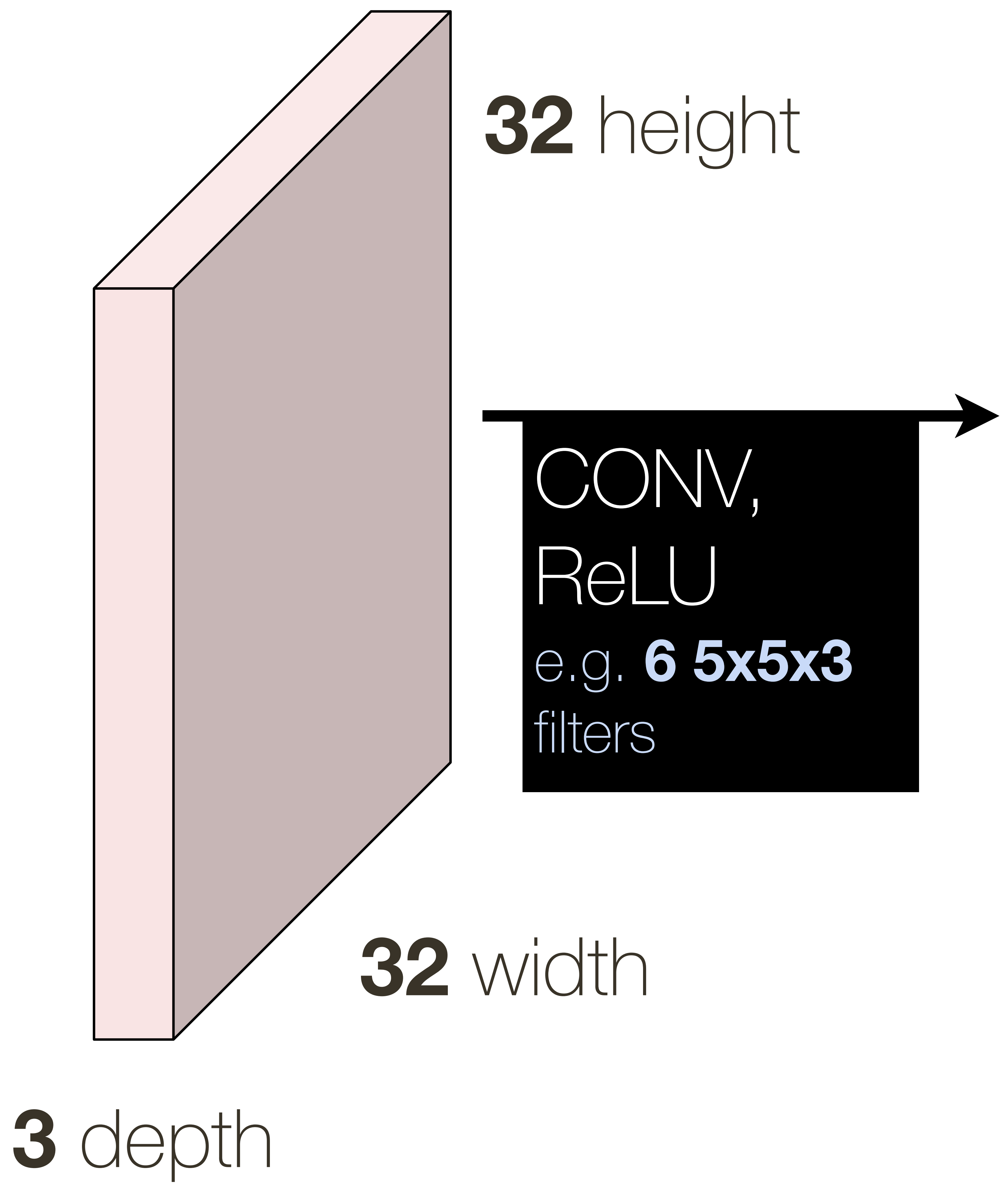
this results in the "new image" of size 28 x 28 x 6!

Convolutional Layer: Closer Look at **Spatial Dimensions**



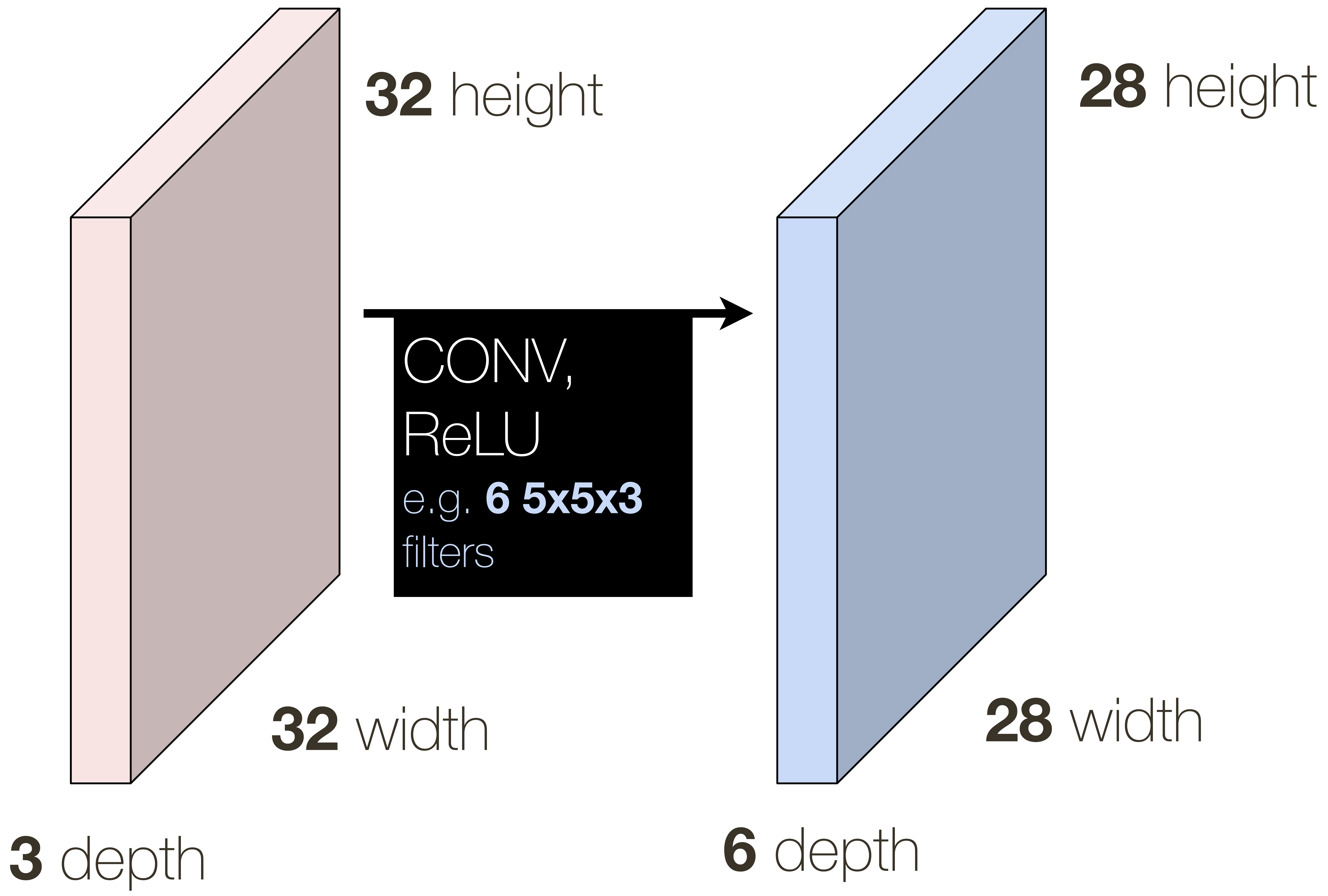
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



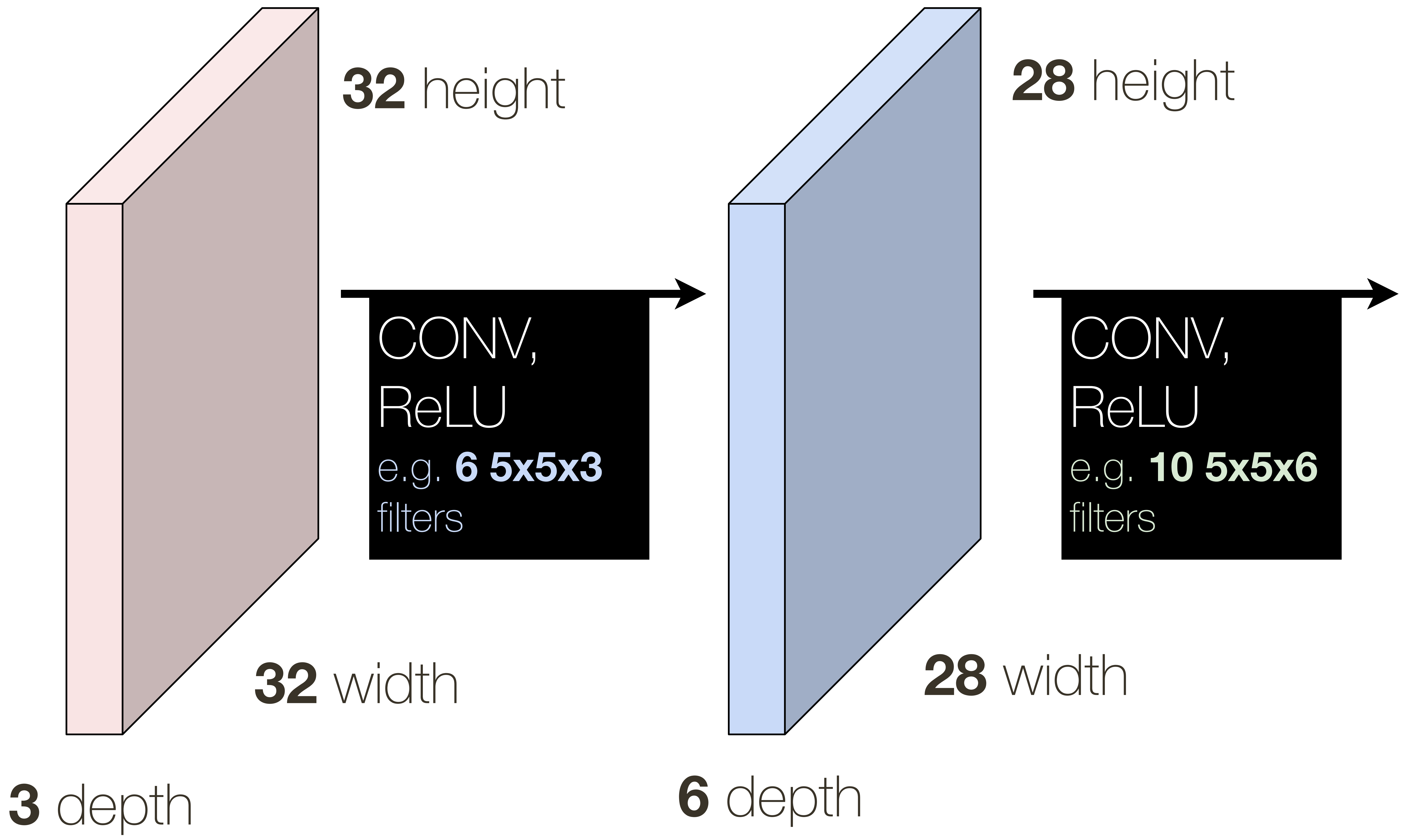
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



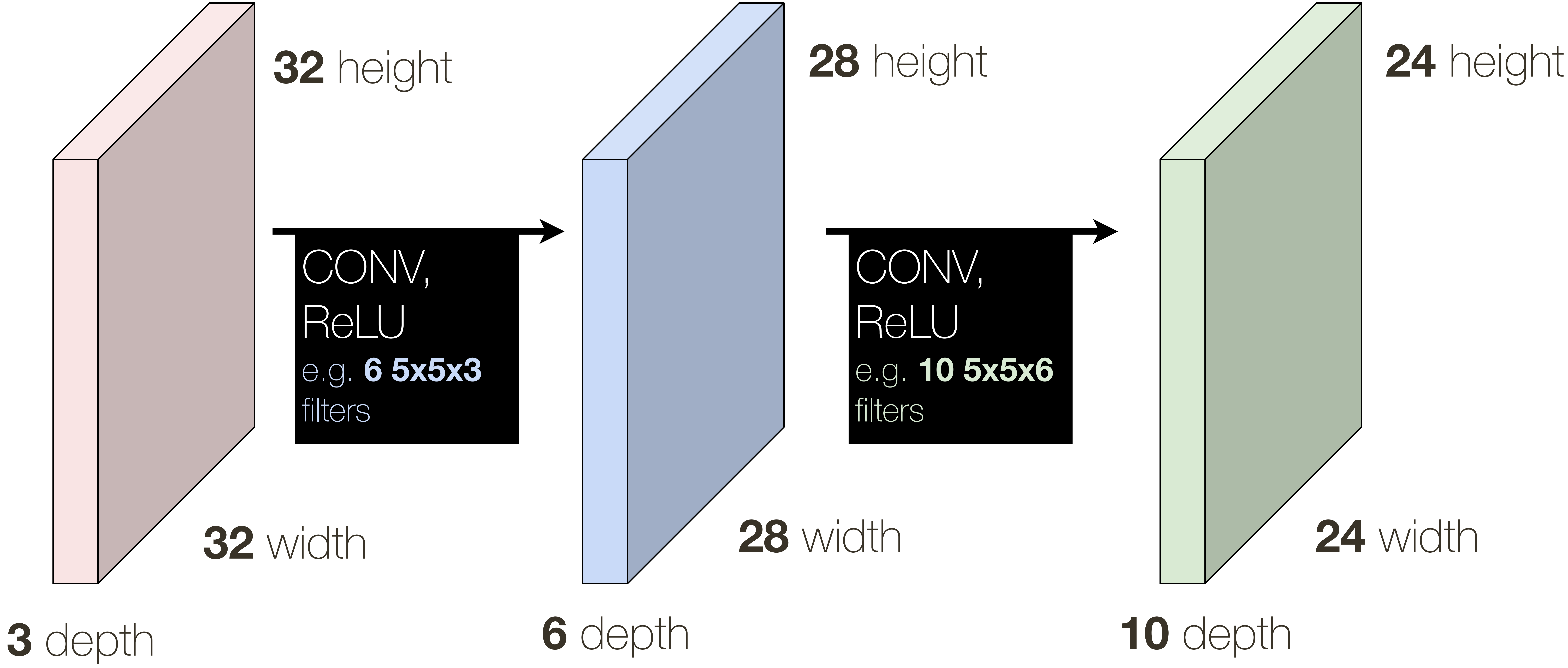
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



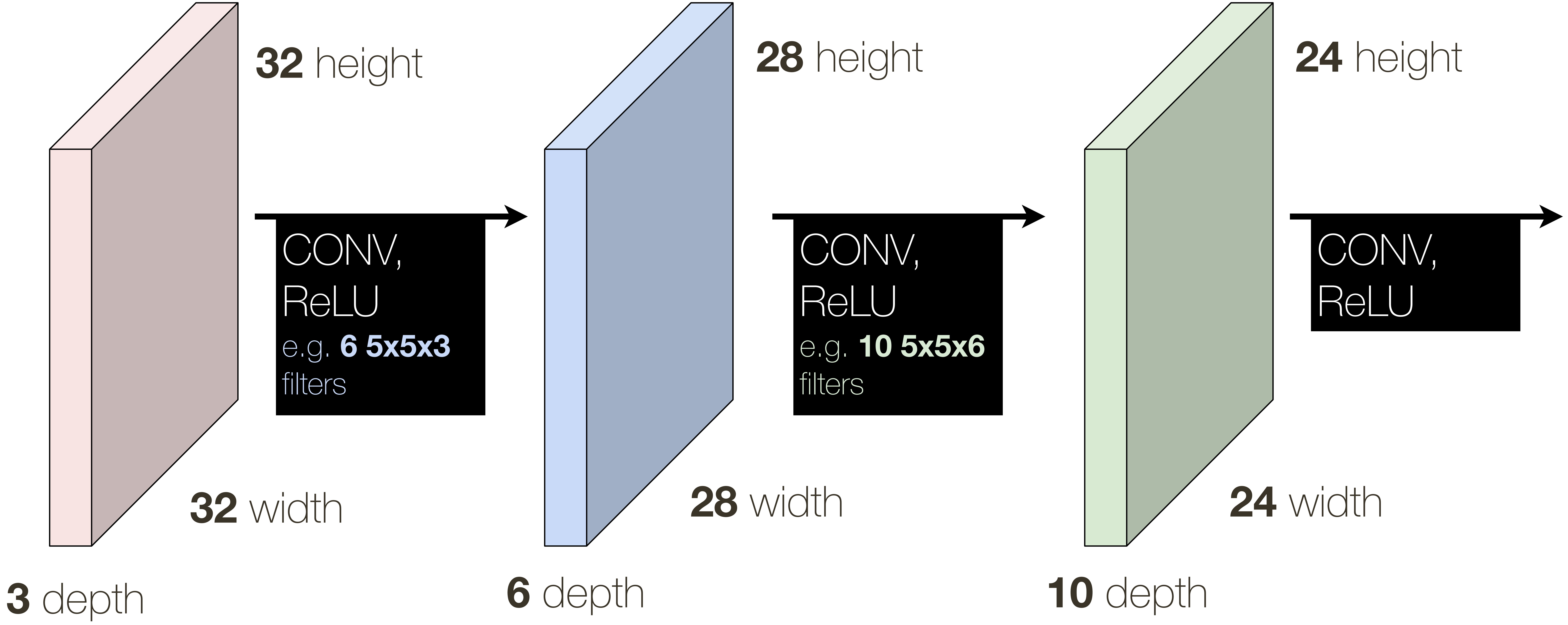
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

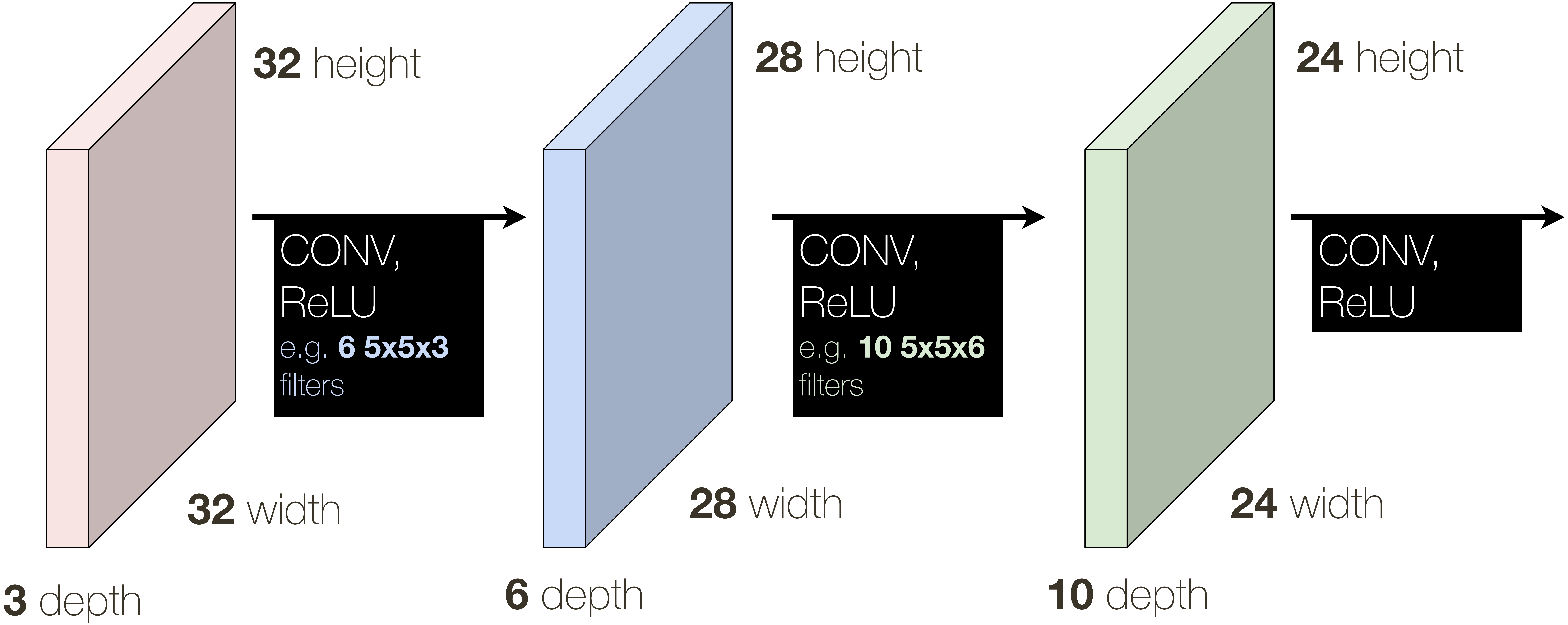
Convolutional Neural Network (ConvNet)



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

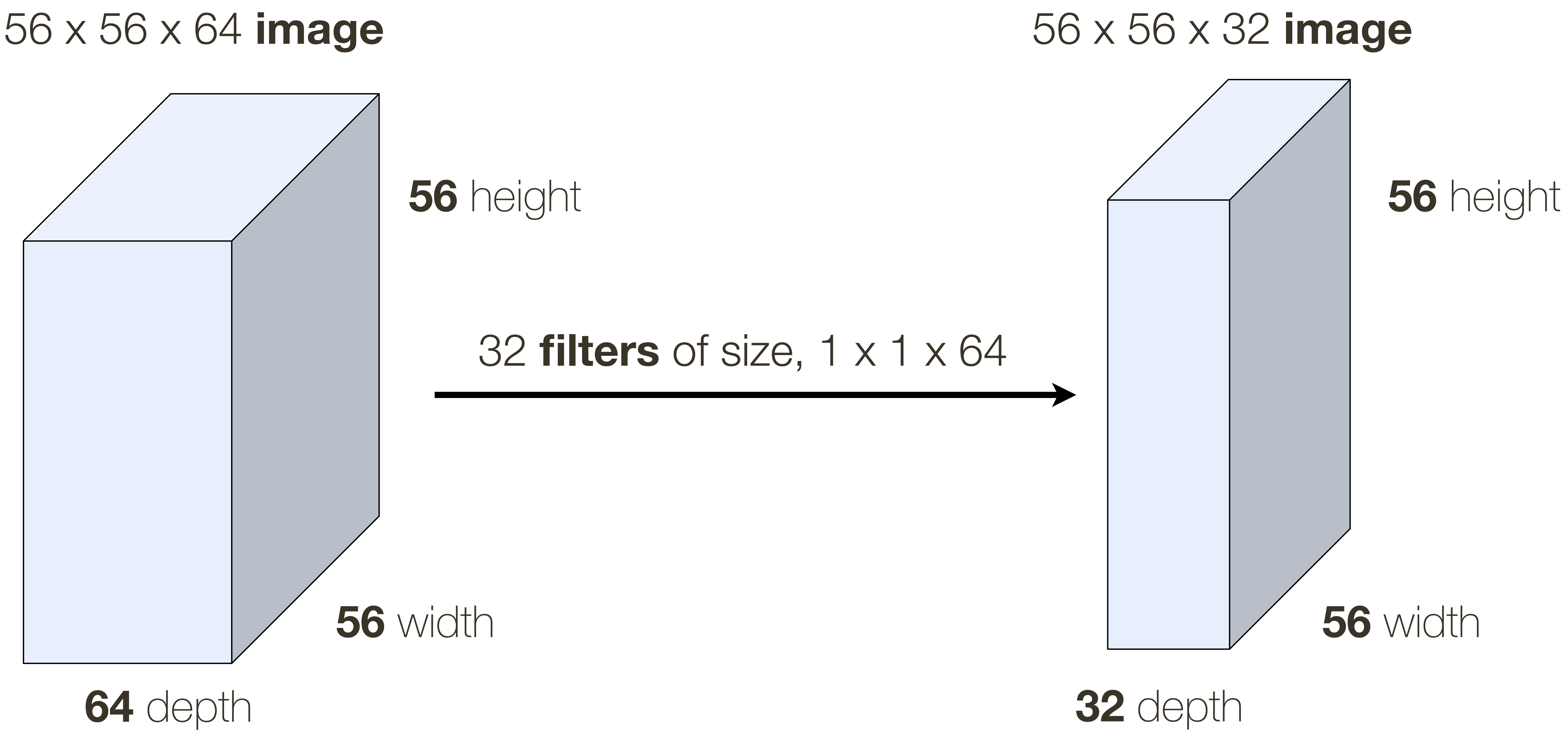
Convolutional Neural Network (ConvNet)

With padding we can achieve no shrinking (32 -> 28 -> 24); shrinking quickly (which happens with larger filters) doesn't work well in practice



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **1x1 convolutions**



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Neural Network (ConvNet)

Convolutional neural networks can be seen as learning a hierarchy of filters.

As we go deeper in the network, filters learn and respond to increasingly specialized structures
— The first layers may contain simple orientation filters, middle layers may respond to common substructures, and final layers may respond to entire objects

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

— Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)

— Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)

— Stride of application: S (for a typical network $S \in \{1, 2\}$)

— Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \qquad H_o = (H_i - F + 2P)/S + 1 \qquad D_o = K$$

Convolutional Layer **Summary**

Accepts a volume of size: $W_i \times H_i \times D_i$

Requires hyperparameters:

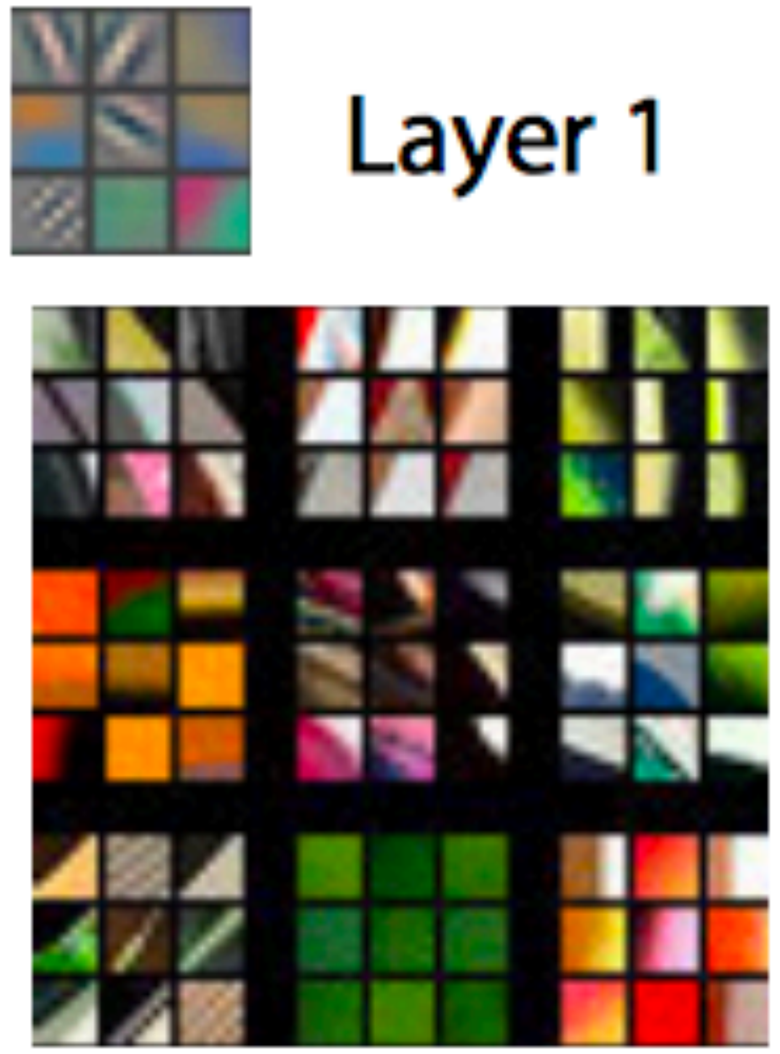
- Number of filters: K (for typical networks $K \in \{32, 64, 128, 256, 512\}$)
- Spatial extent of filters: F (for a typical networks $F \in \{1, 3, 5, ...\}$)
- Stride of application: S (for a typical network $S \in \{1, 2\}$)
- Zero padding: P (for a typical network $P \in \{0, 1, 2\}$)

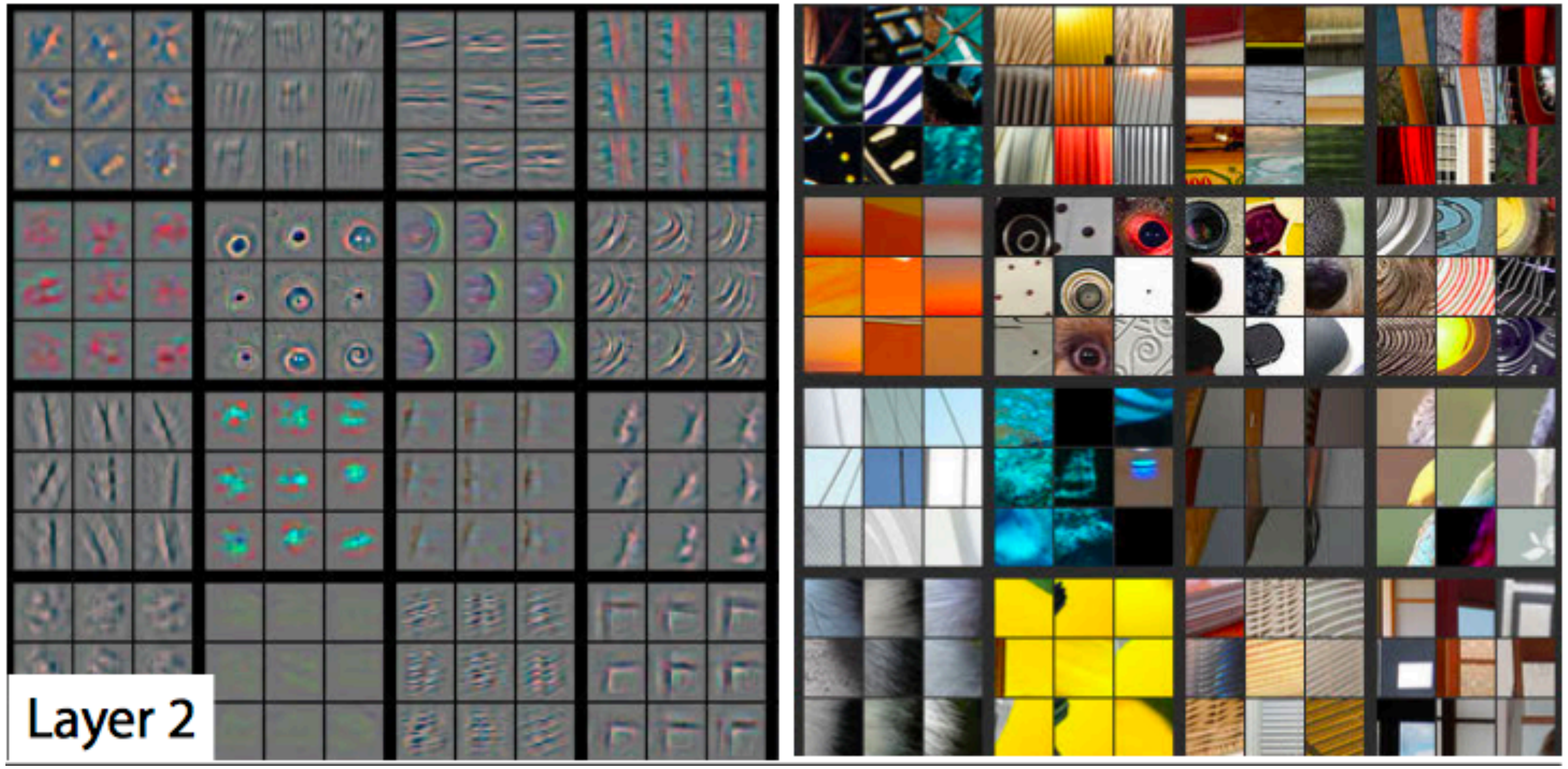
Produces a volume of size: $W_o \times H_o \times D_o$

$$W_o = (W_i - F + 2P)/S + 1 \qquad H_o = (H_i - F + 2P)/S + 1 \qquad D_o = K$$

Number of total learnable parameters: $(F \times F \times D_i) \times K + K$

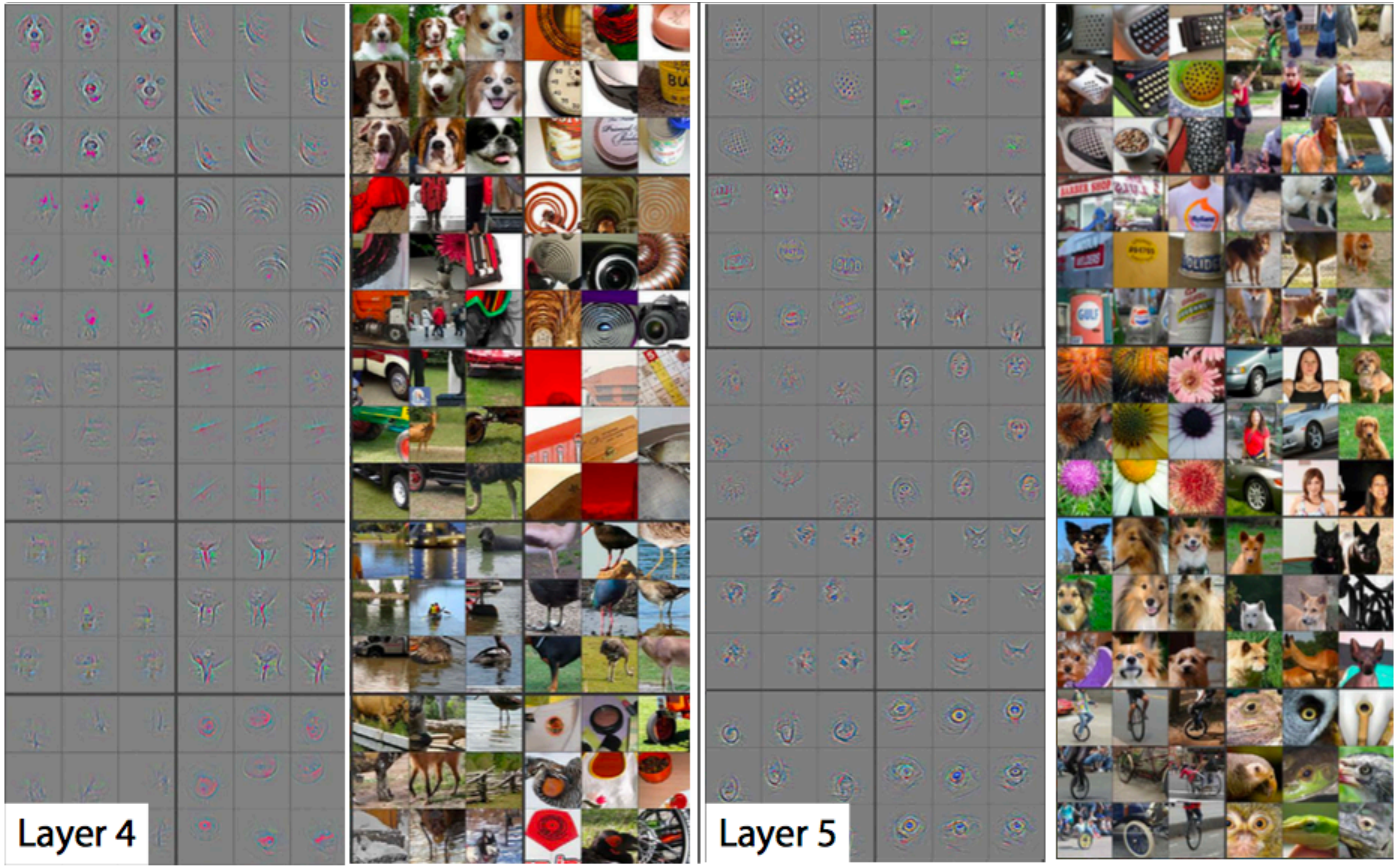
What **filters** do networks learn?





[Zeiler and Fergus, 2013]

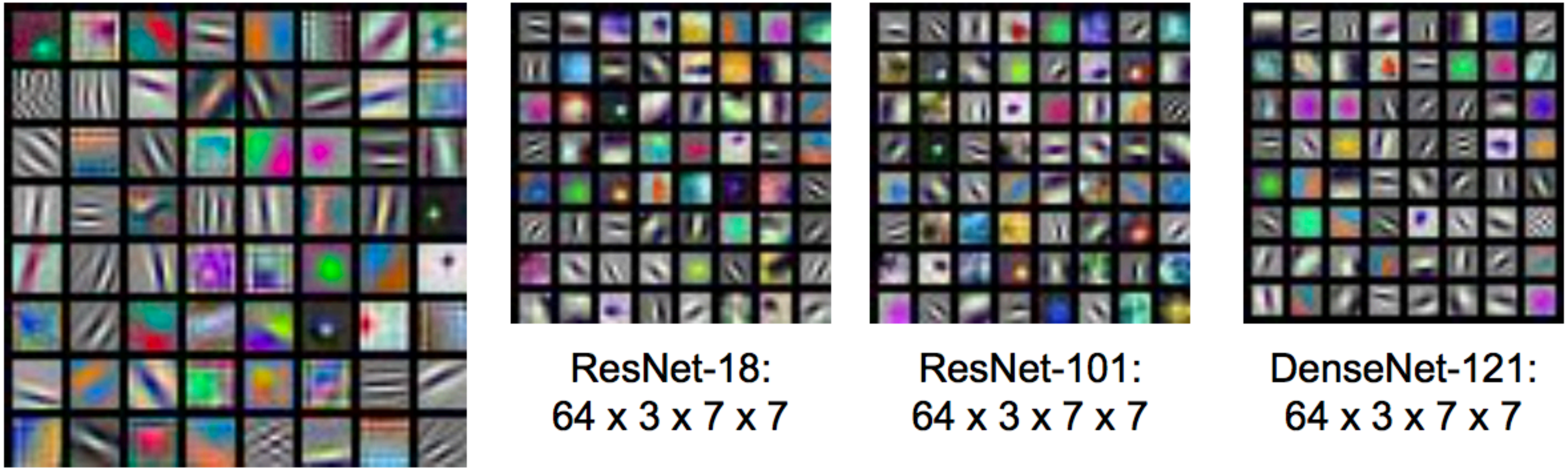
What **filters** do networks learn?



[Zeiler and Fergus, 2013]

First Layer Filters …

Directly **visualize filters** (only works for the first layer)



AlexNet:
64 x 3 x 11 x 11

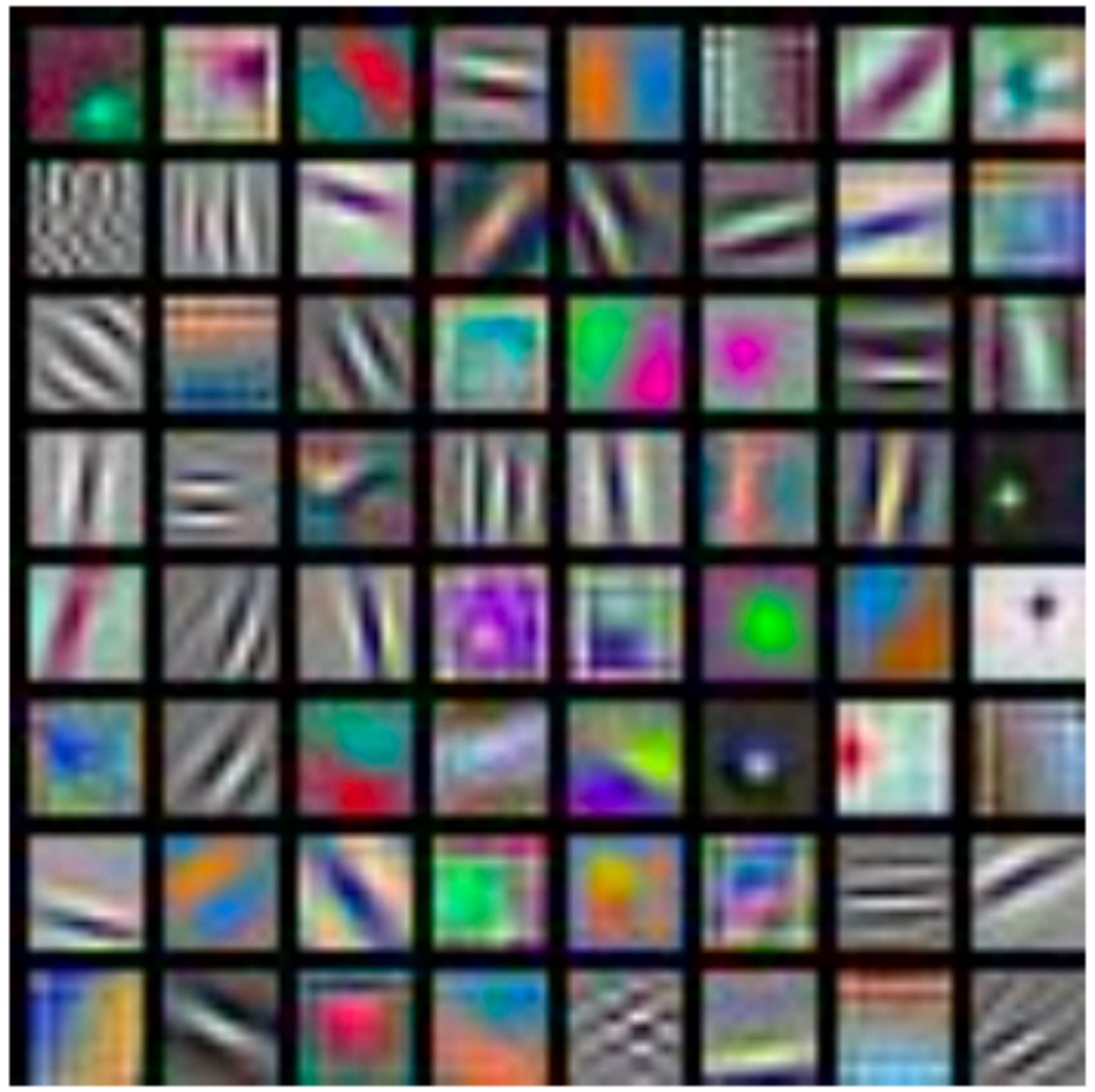
ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

First Layer Filters …

Directly **visualize filters** (only works for the first layer)



AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

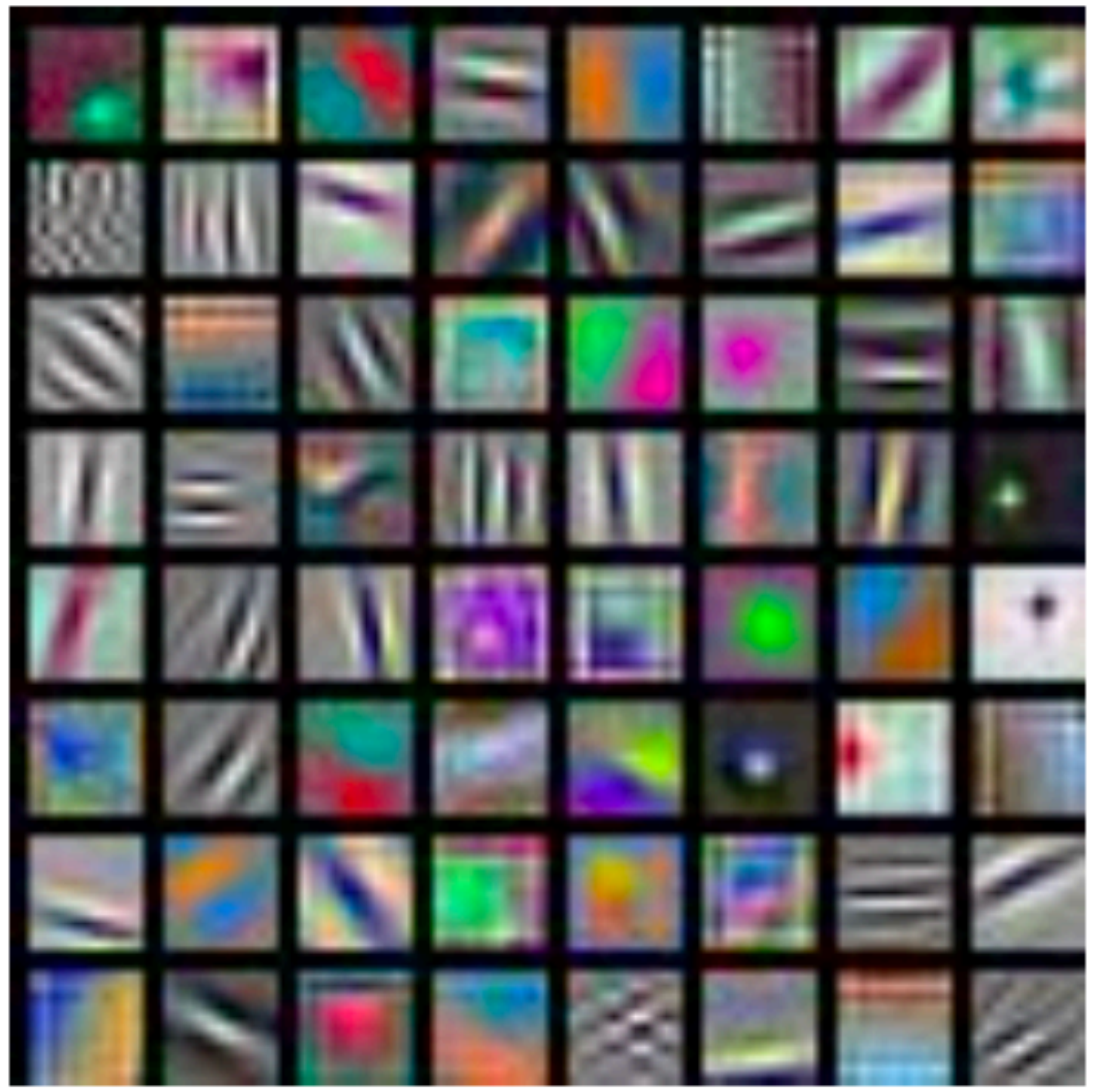
DenseNet-121:
64 x 3 x 7 x 7

… surprisingly similar across variety of networks

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

First Layer Filters …

Directly **visualize filters** (only works for the first layer)



AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

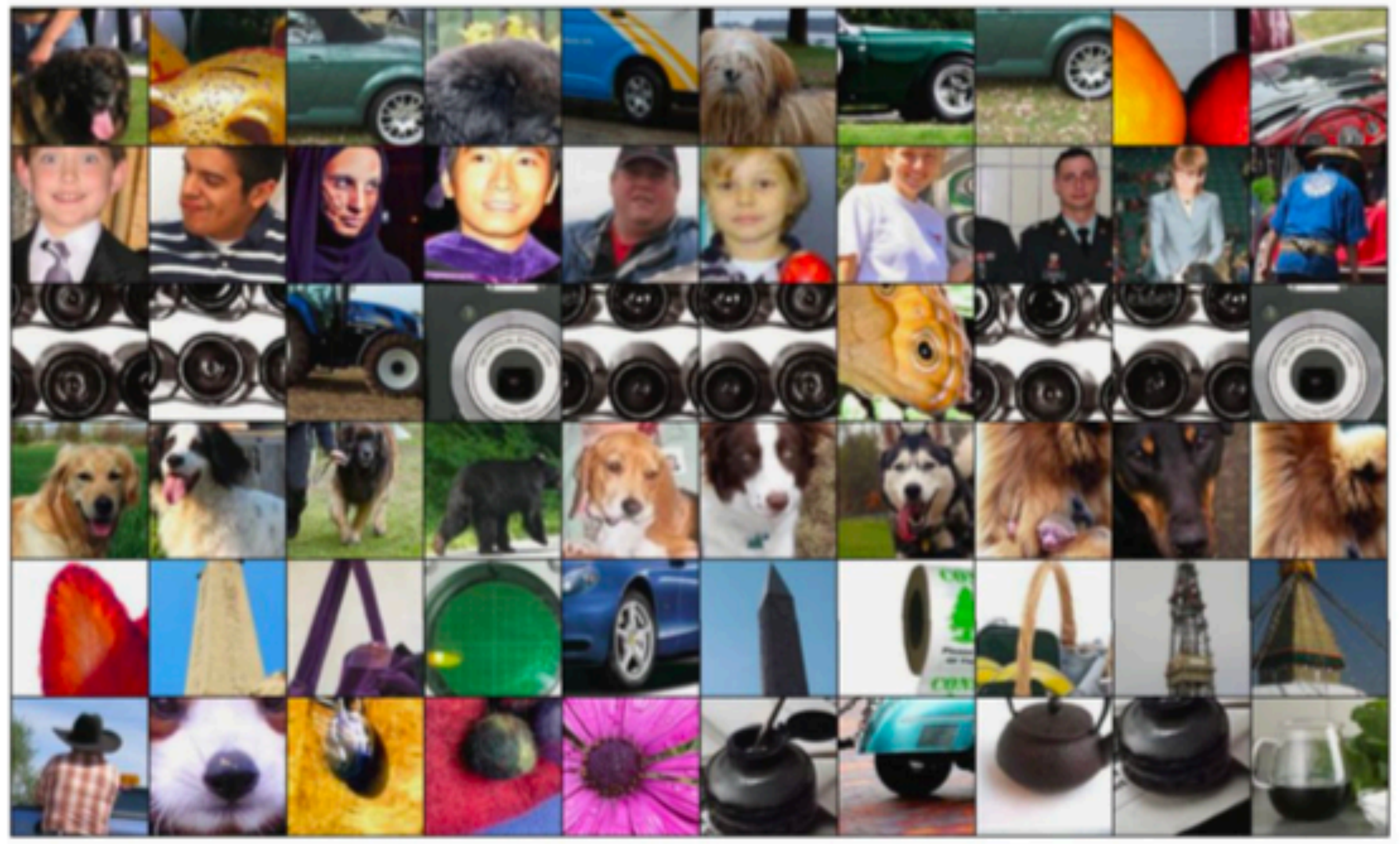
… surprisingly similar across variety of networks

… and nearly any dataset

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Maximally **Activating Patches**

- Pick a layer and a channel; e.g., cons5 of AlexNet is 128x13x13
- Run many images through the network
- Visualize image patches that correspond to maximal activation of the neuron



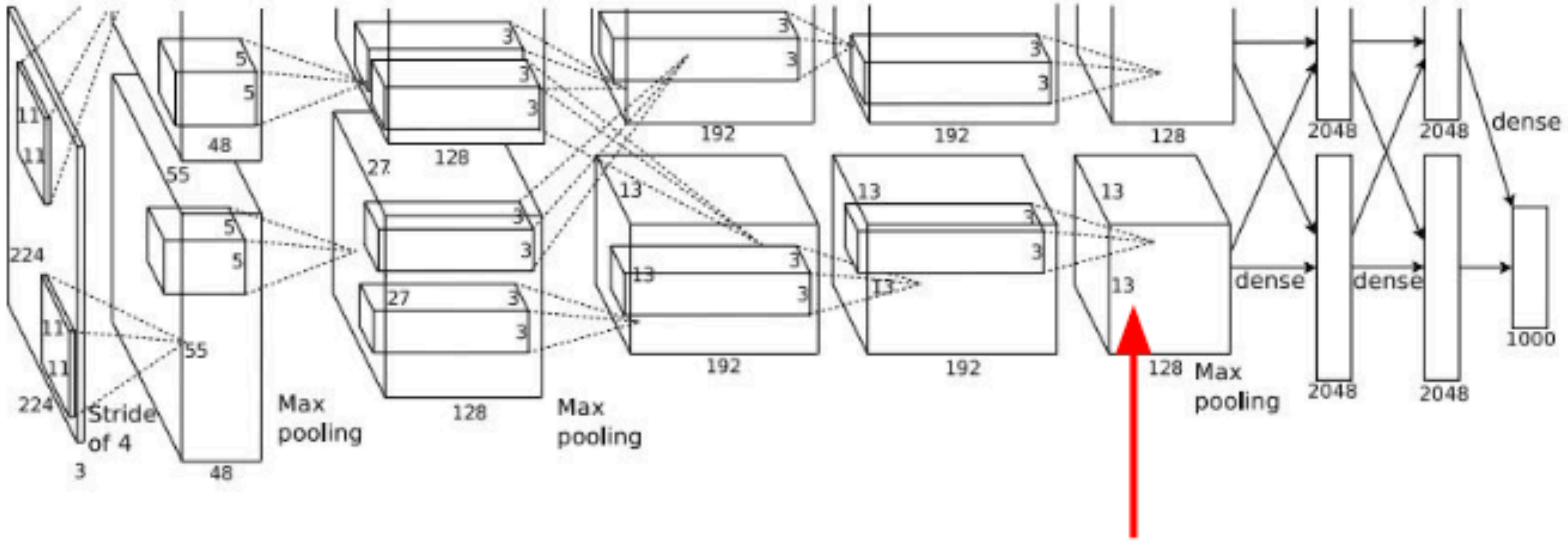
[Springenberg et al., 2015]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Intermediate Features through (**Guided**) **BackProp**

— Pick a single intermediate neuron somewhere in the network, e.g., neuron in 128x13x13 conv5 feature map

— Compute **gradient of neuron value with respect to image pixels**

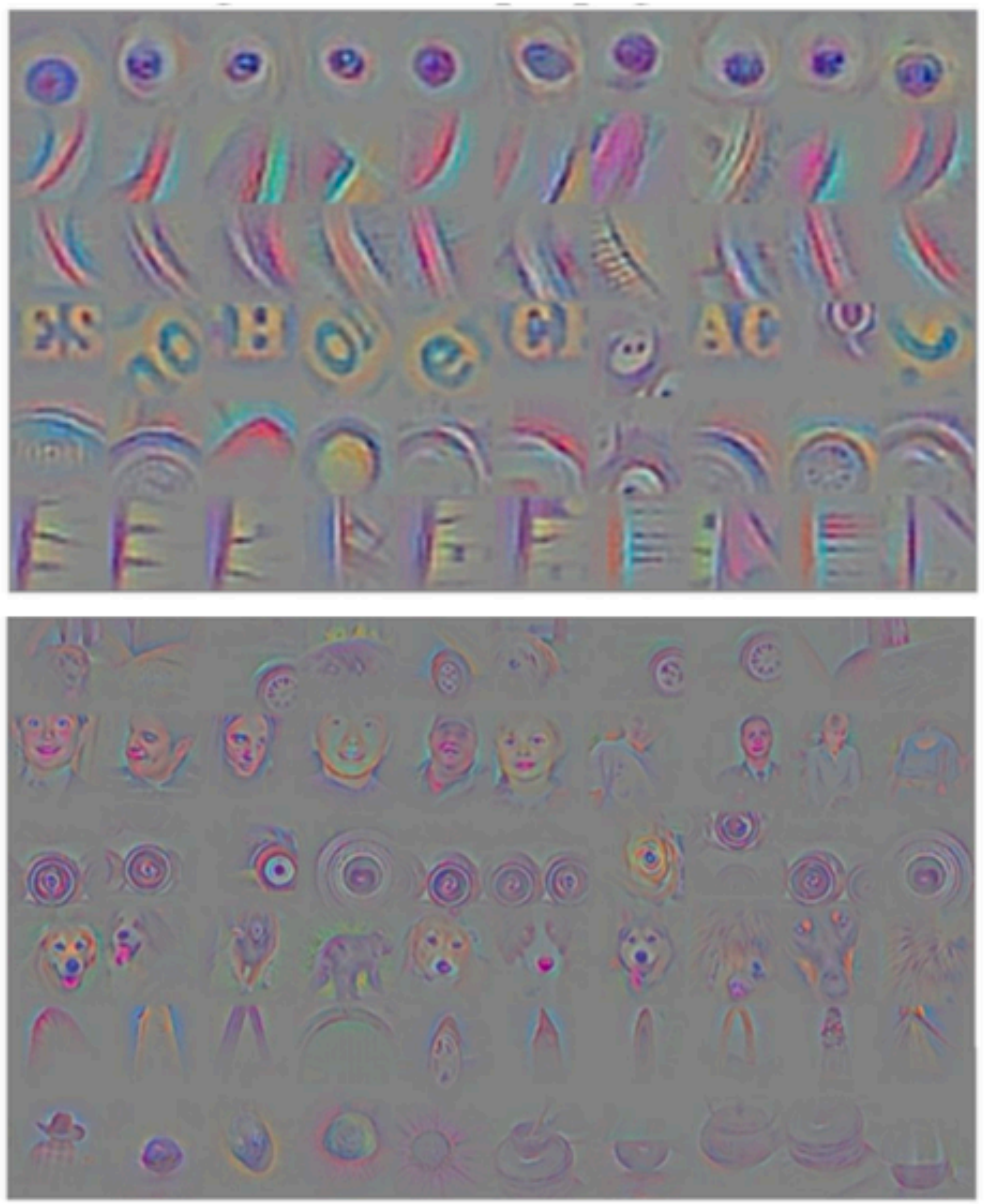


[Zeiler and Fergus, 2014]

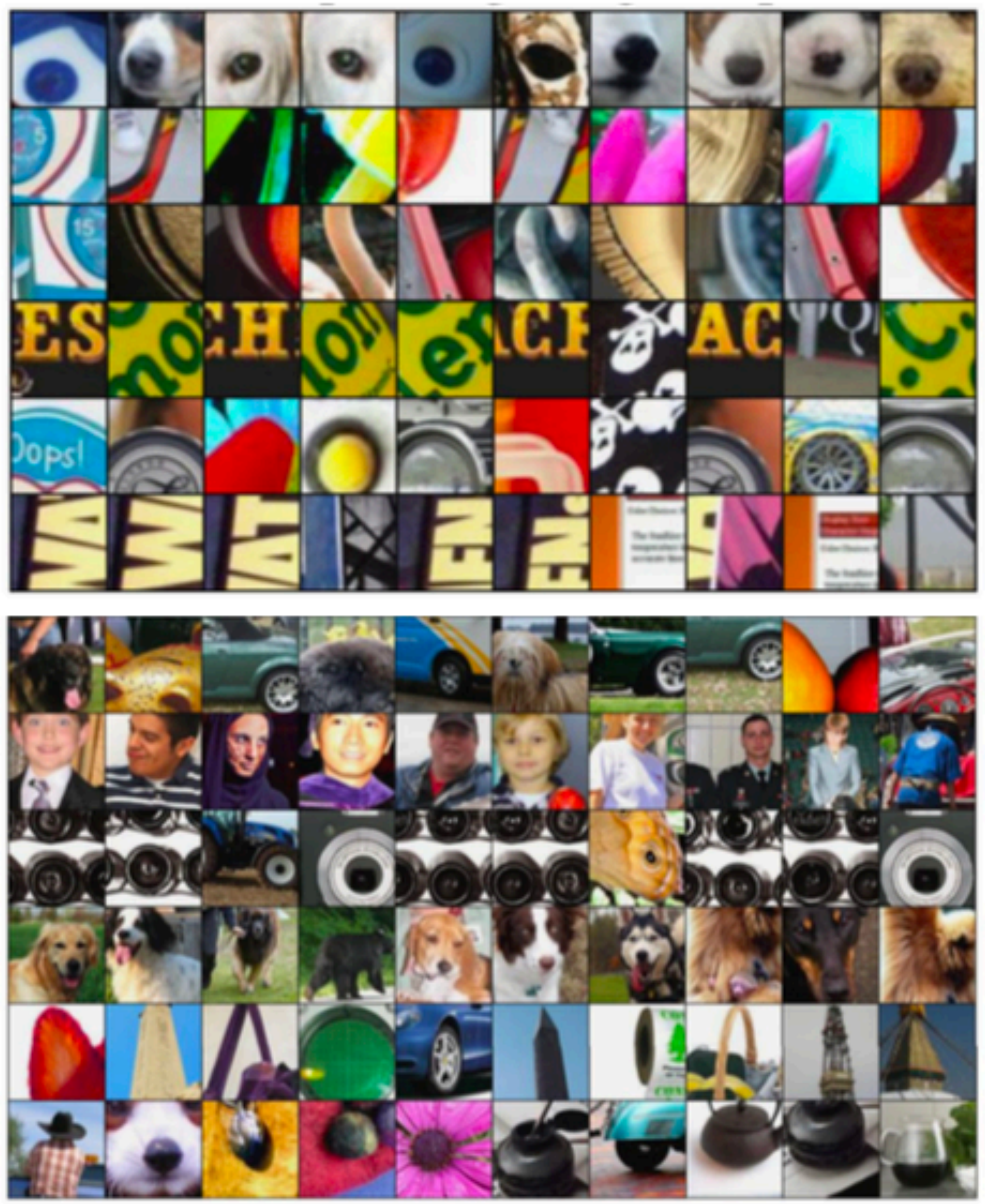
[Springenberg et al., 2015]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Intermediate Features through (**Guided**) **BackProp**



[Springenberg et al., 2015]



[Zeiler and Fergus, 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Gradient **Ascent**

(Guided) **BackProp**: find the part of an image that a neuron responds to

Gradient ascent: generate a synthetic image that maximally activates a neuron

Gradient **Ascent**

(Guided) **BackProp**: find the part of an image that a neuron responds to

Gradient ascent: generate a synthetic image that maximally activates a neuron

$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Gradient **Ascent**

(Guided) **BackProp**: find the part of an image that a neuron responds to

Gradient ascent: generate a synthetic image that maximally activates a neuron

$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Neuron Value

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Gradient **Ascent**

(Guided) **BackProp**: find the part of an image that a neuron responds to

Gradient ascent: generate a synthetic image that maximally activates a neuron

Natural **Image Regularizer**

$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Neuron Value

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Gradient **Ascent**

1. Initialize image with all zeros (can also start with an existing image)
2. Forward image to compute the current scores
3. BackProp to get gradient of the neuron with respect to image pixels
4. Make a small update to an image

Natural **Image Regularizer**

$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Neuron Value

Gradient **Ascent**

1. Initialize image with all zeros (can also start with an existing image)
2. Forward image to compute the current scores
3. BackProp to get gradient of the neuron with respect to image pixels
4. Make a small update to an image

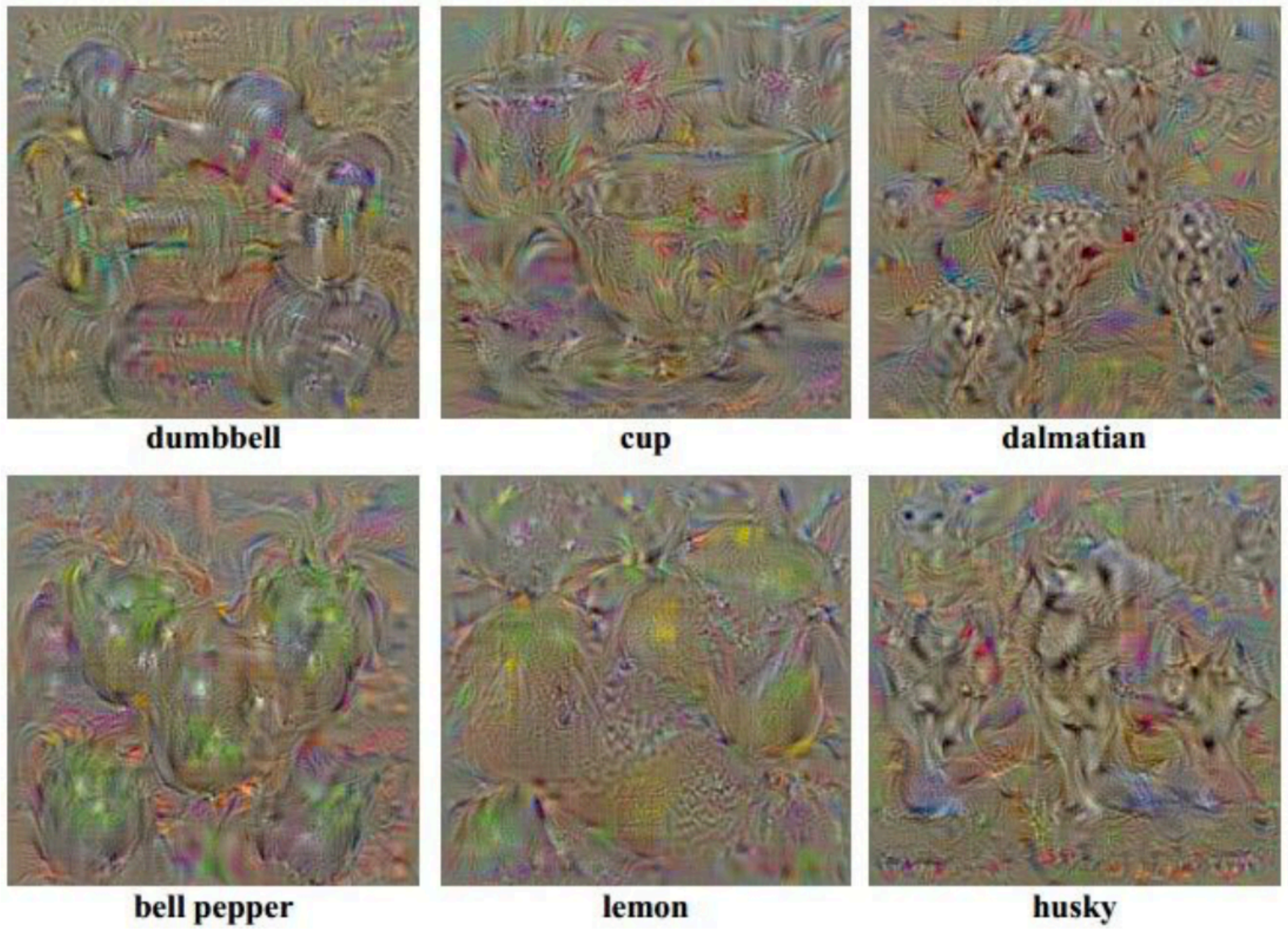
Natural **Image Regularizer** $R(\mathbf{I}) = -\lambda ||\mathbf{I}||_2^2$

$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Score for class C before softmax

[Simonyan et al., 2014]

Gradient **Ascent**



$$\mathbf{I}^* = \arg\max_{\mathbf{I}} f(\mathbf{I}) + R(\mathbf{I})$$

Natural **Image Regularizer** $R(\mathbf{I}) = -\lambda||\mathbf{I}||_2^2$

Score for class C before softmax

[Simonyan et al., 2014]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Gradient **Ascent**

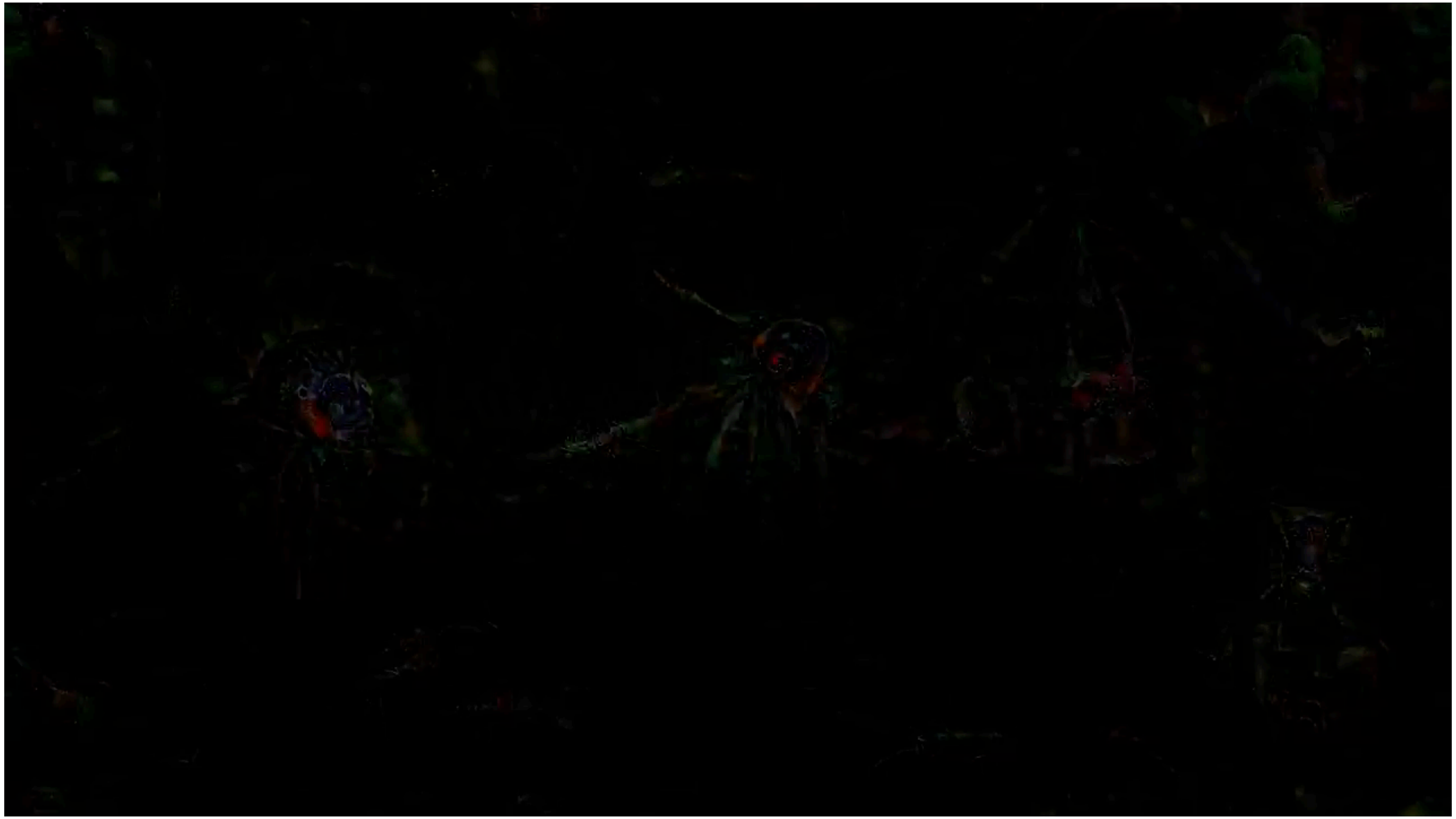
… with a few additional tweaks

[Nguyen et al., 2015]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Deep **Dream**

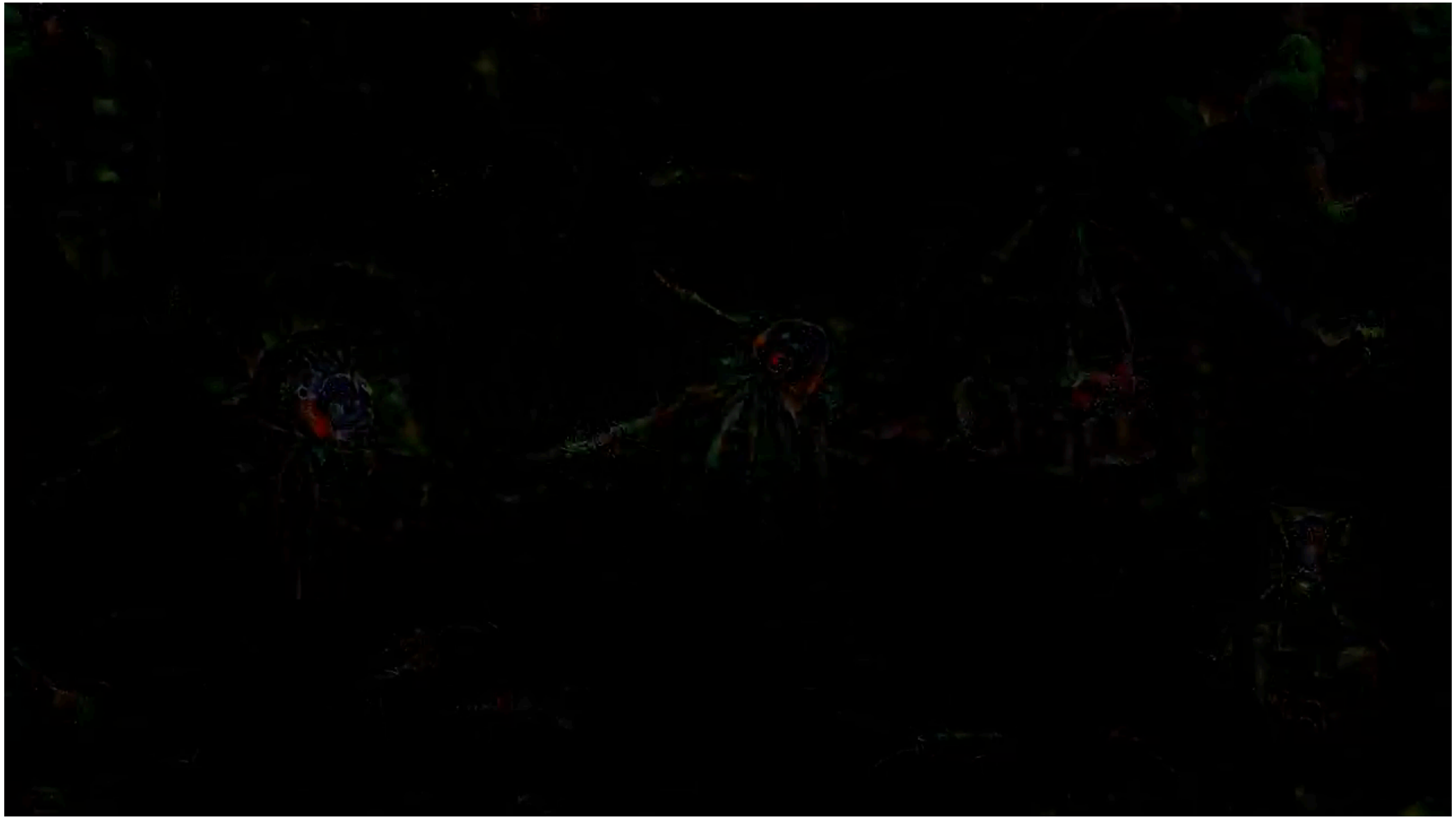
[Mordvinsev, Olah, Tyka]



https://www.youtube.com/watch?v=DgPaCWJL7XI&t=11s

Deep **Dream**

[Mordvinsev, Olah, Tyka]

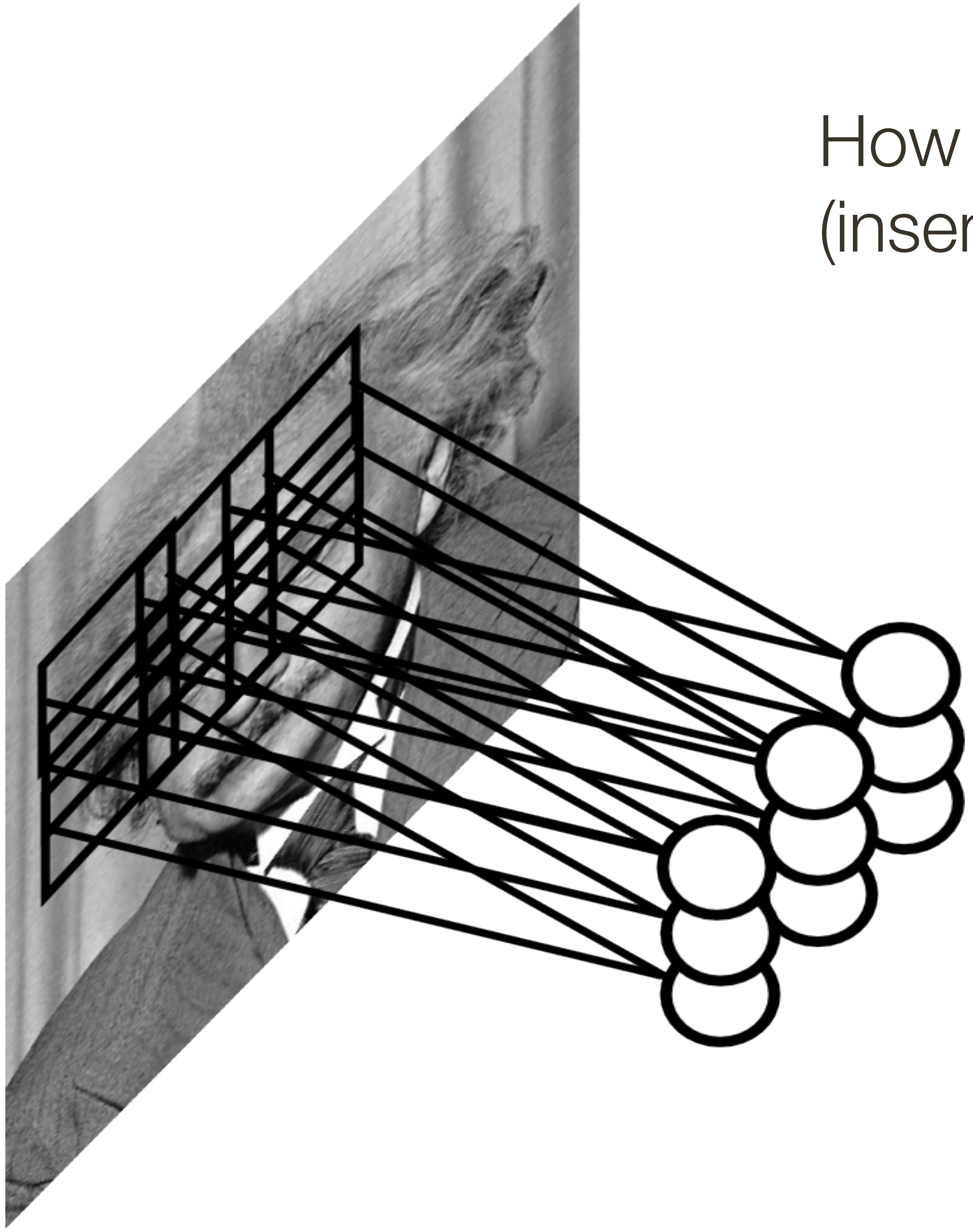


https://www.youtube.com/watch?v=DgPaCWJL7XI&t=11s

Pooling Layer

Let us assume the filter is an “eye” detector

How can we make detection spatially invariant
(insensitive to position of the eye in the image)

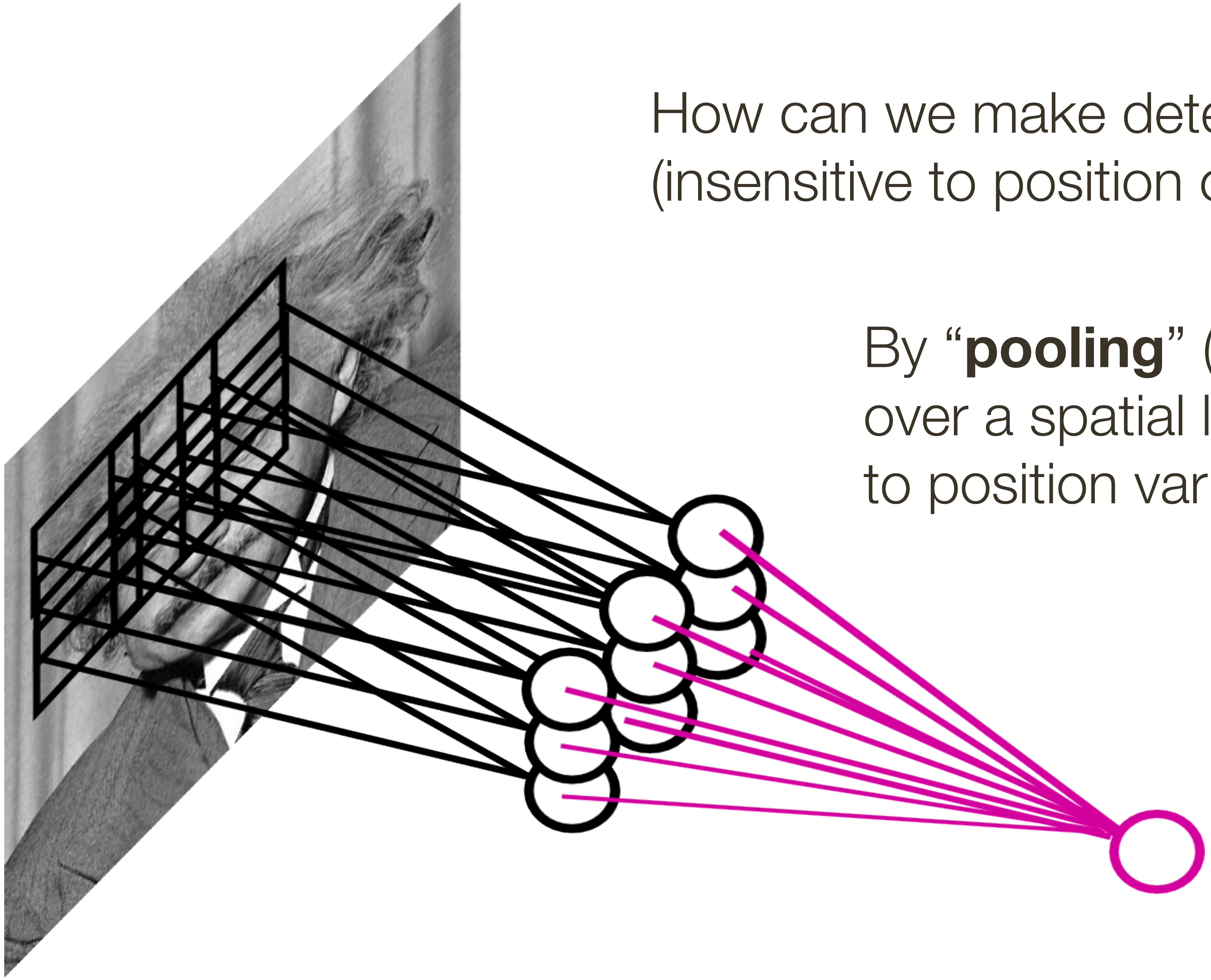


* slide from Marc’Aurelio Renzato

Pooling Layer

Let us assume the filter is an “eye” detector

How can we make detection spatially invariant (insensitive to position of the eye in the image)

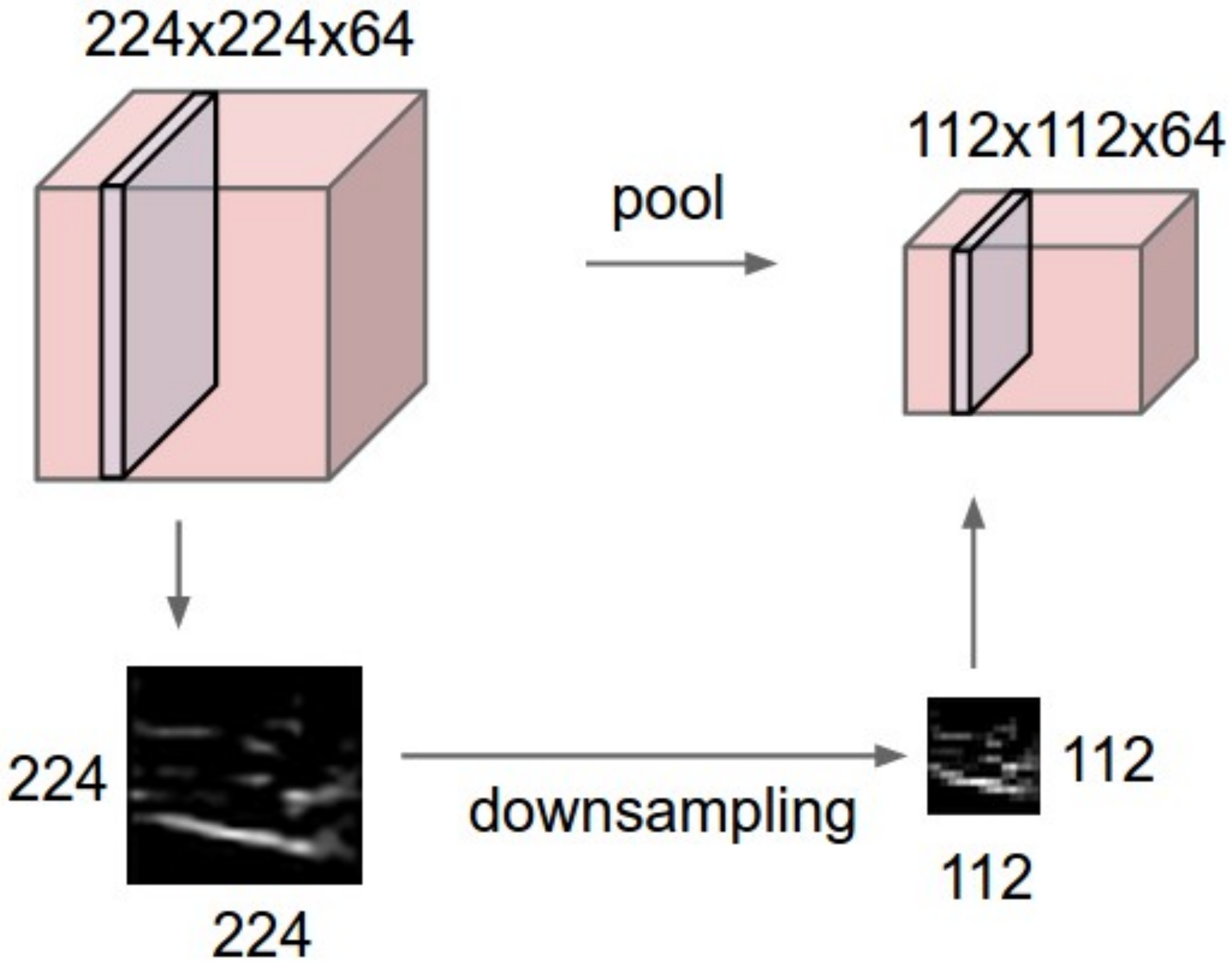


By “**pooling**” (e.g., taking a max) response over a spatial locations we gain robustness to position variations

* slide from Marc’Aurelio Renzato

Pooling Layer

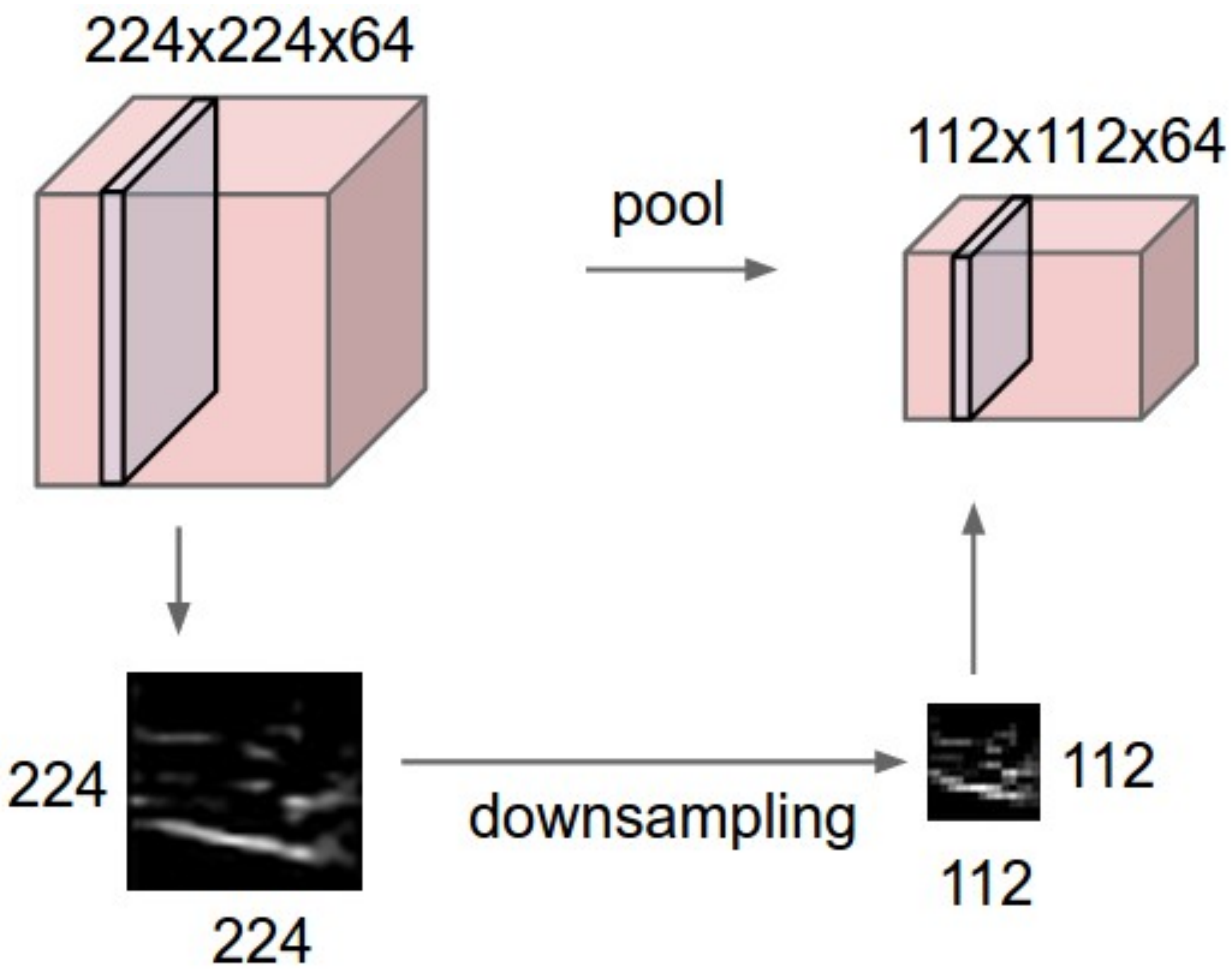
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

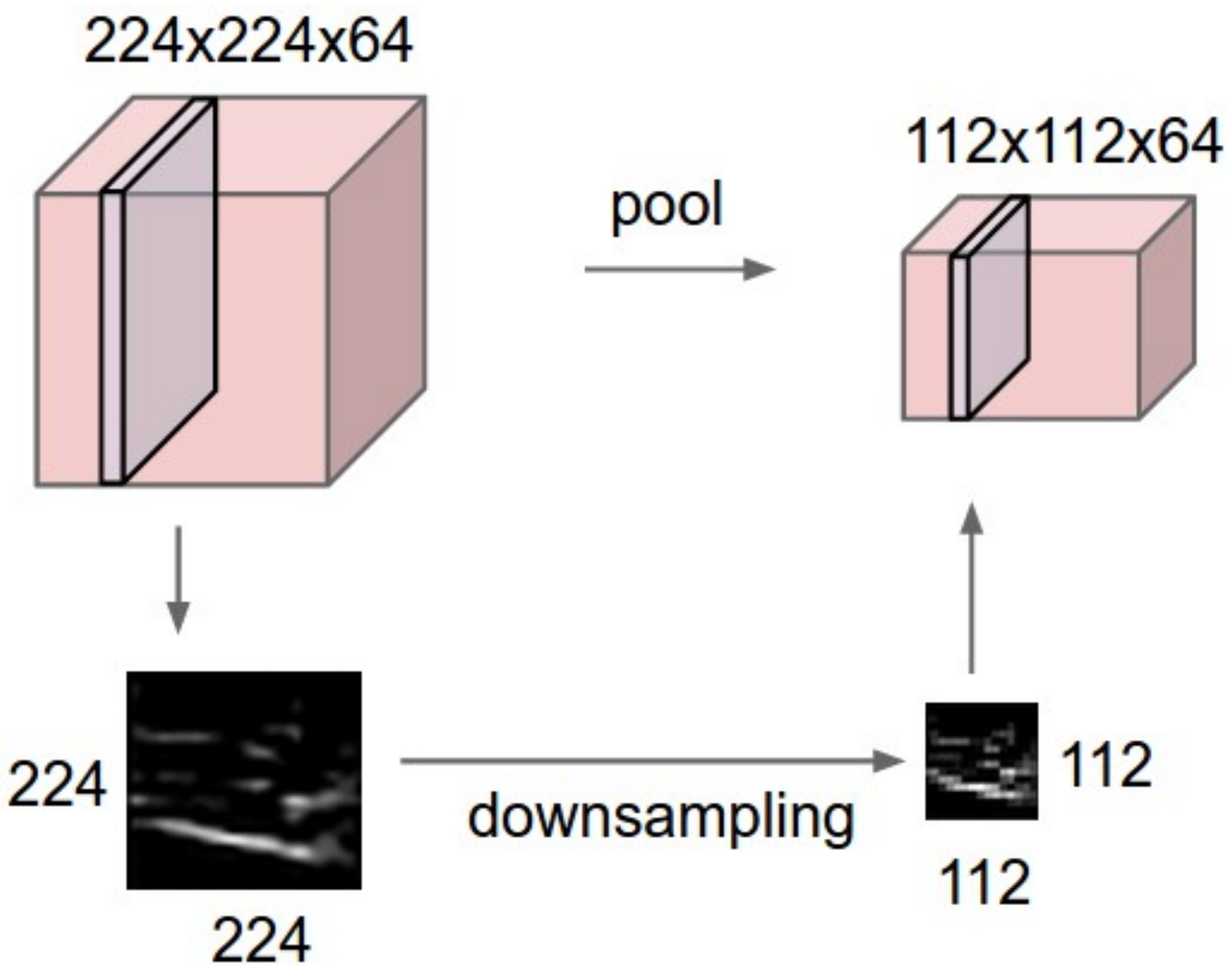


How many **parameters**?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How many **parameters**?

None!

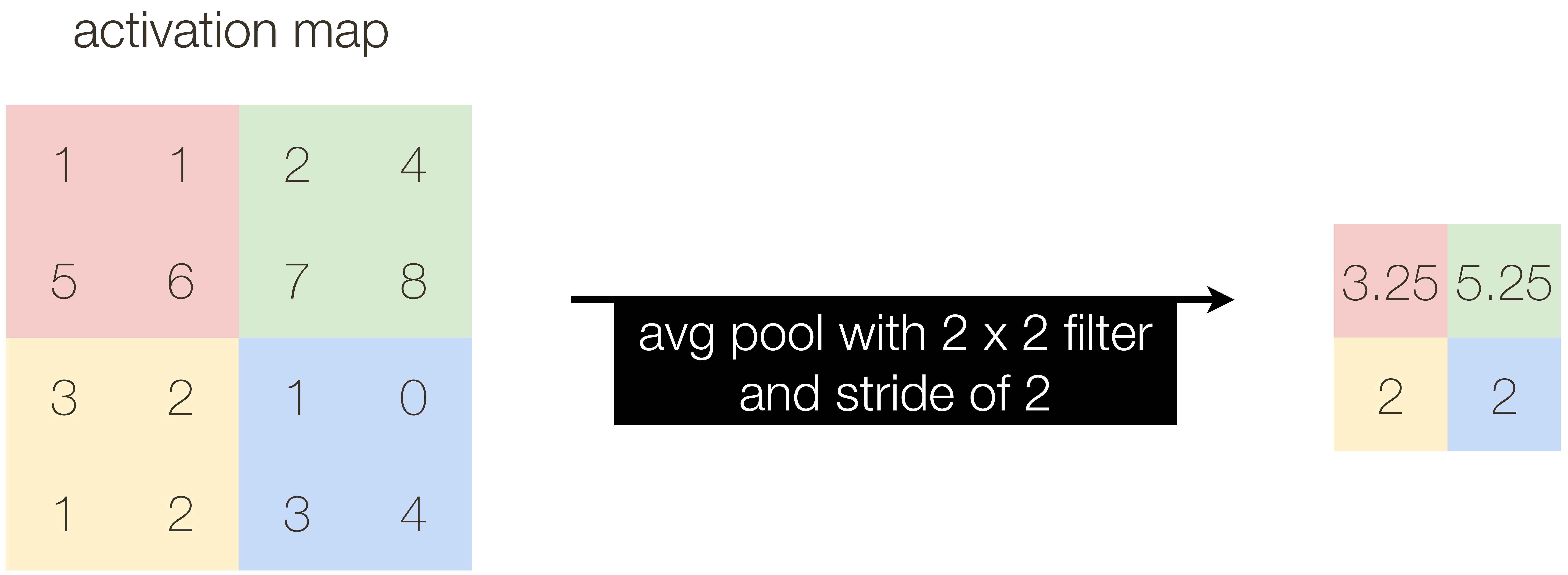
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Max **Pooling**

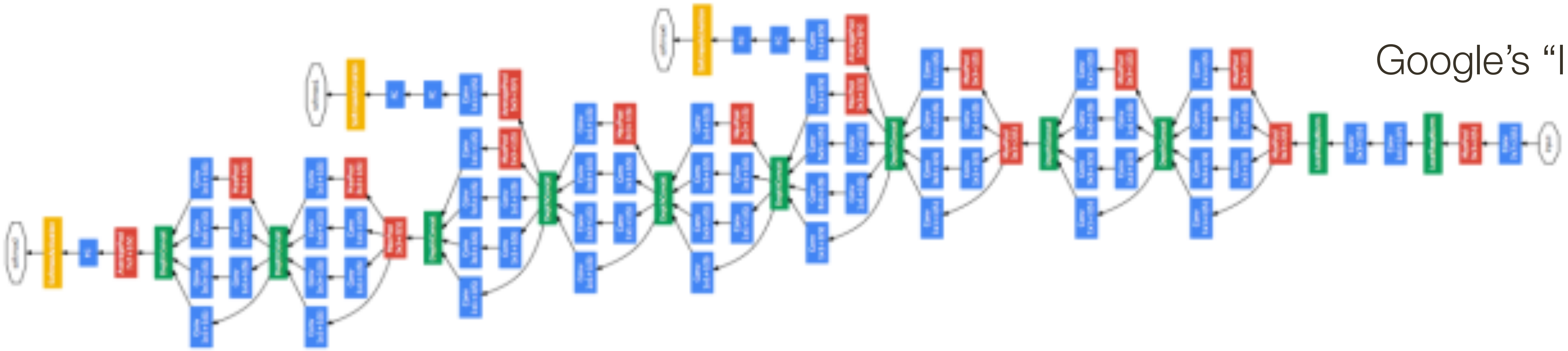
activation map

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Average **Pooling**



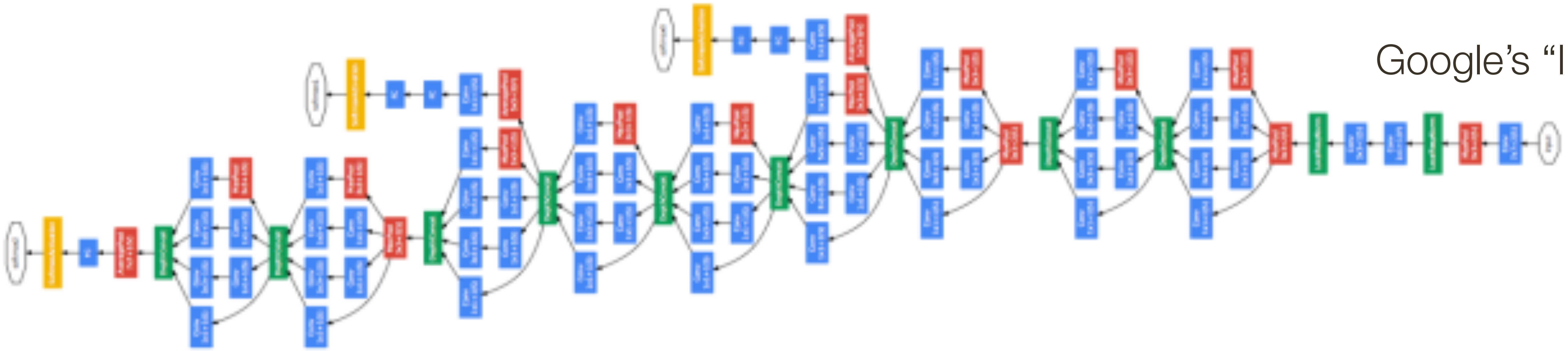
Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

Deep Learning **Terminology**

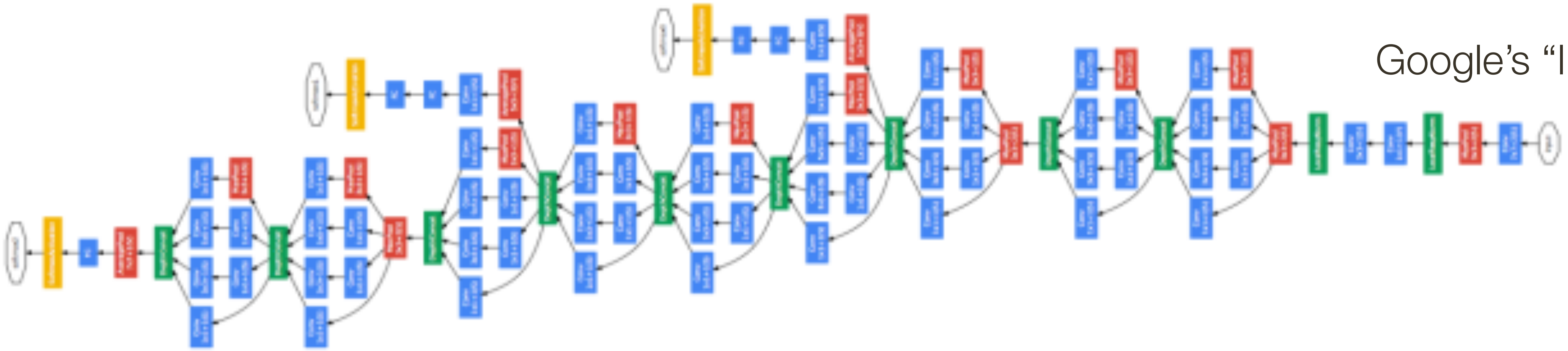


Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

Deep Learning **Terminology**



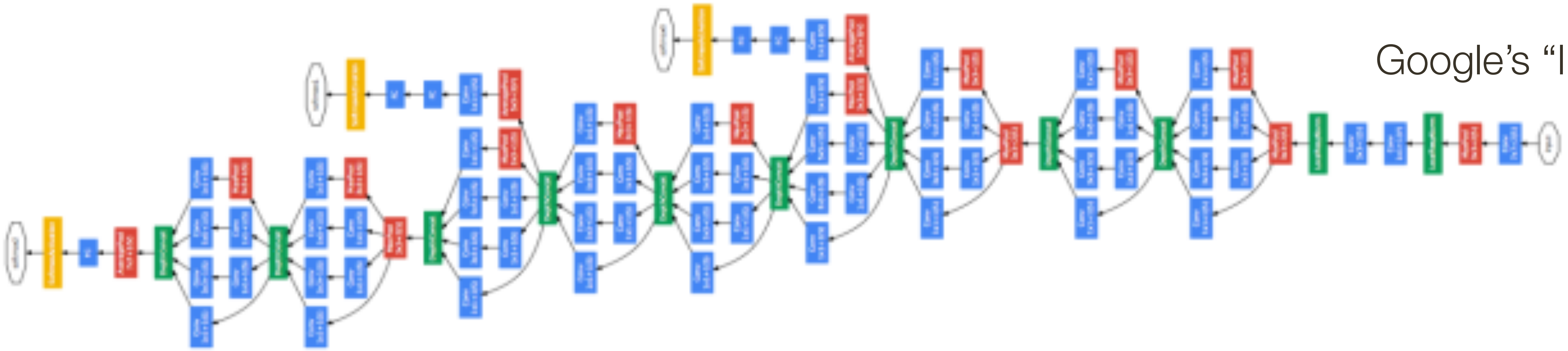
Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

Deep Learning **Terminology**



Google's "Inception" network

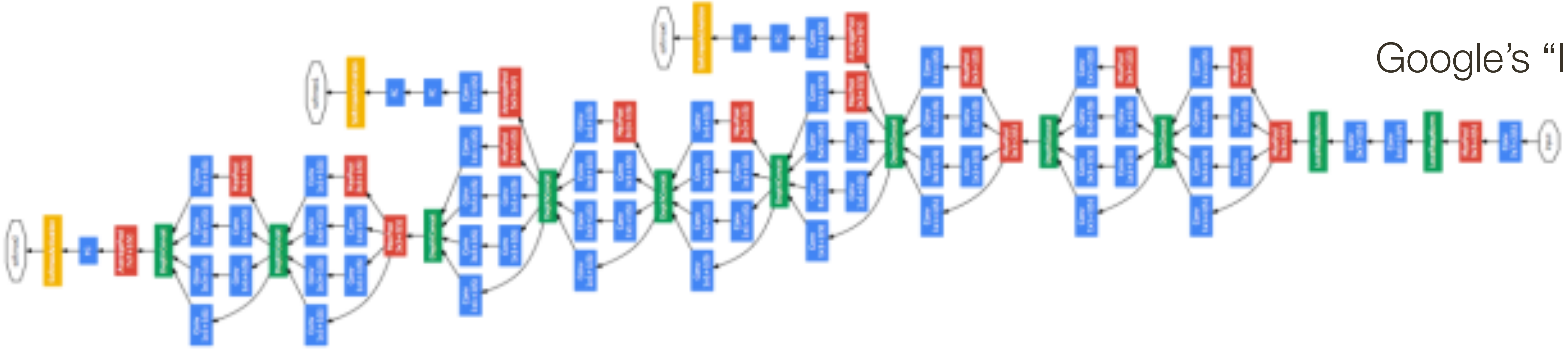
- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

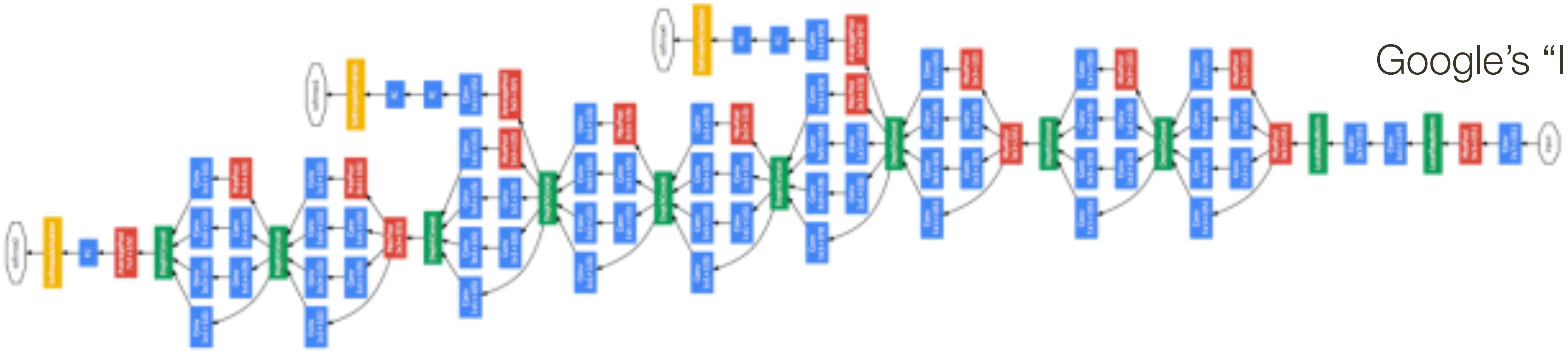
generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

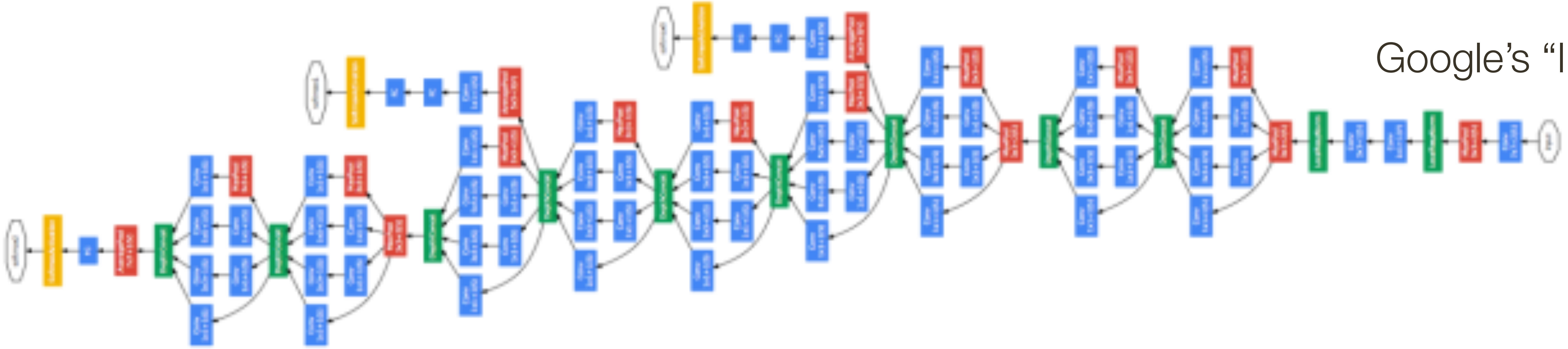
deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

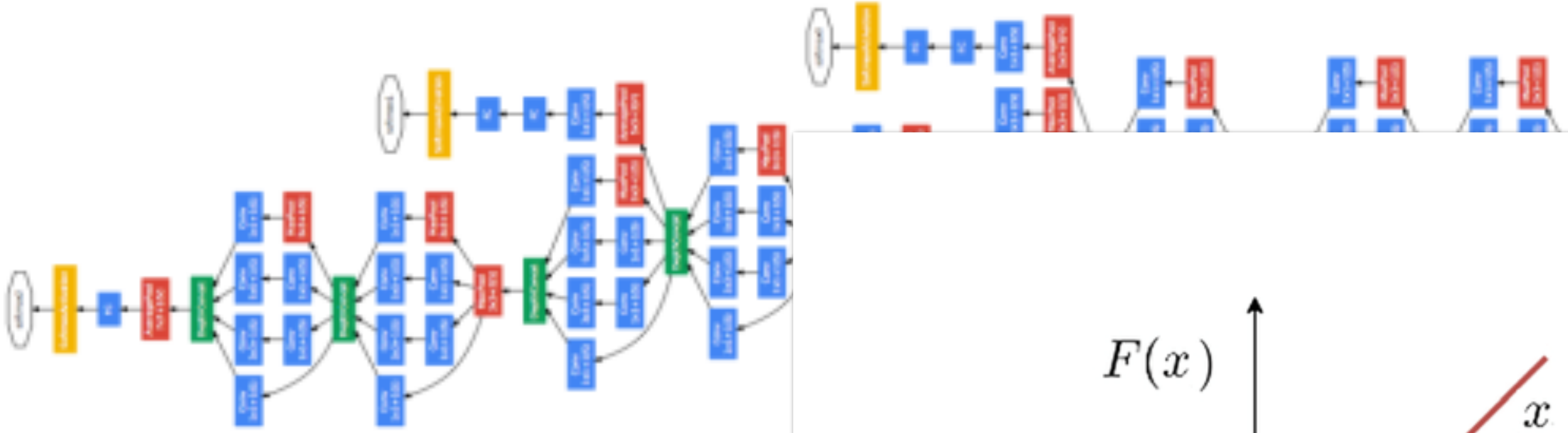
- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

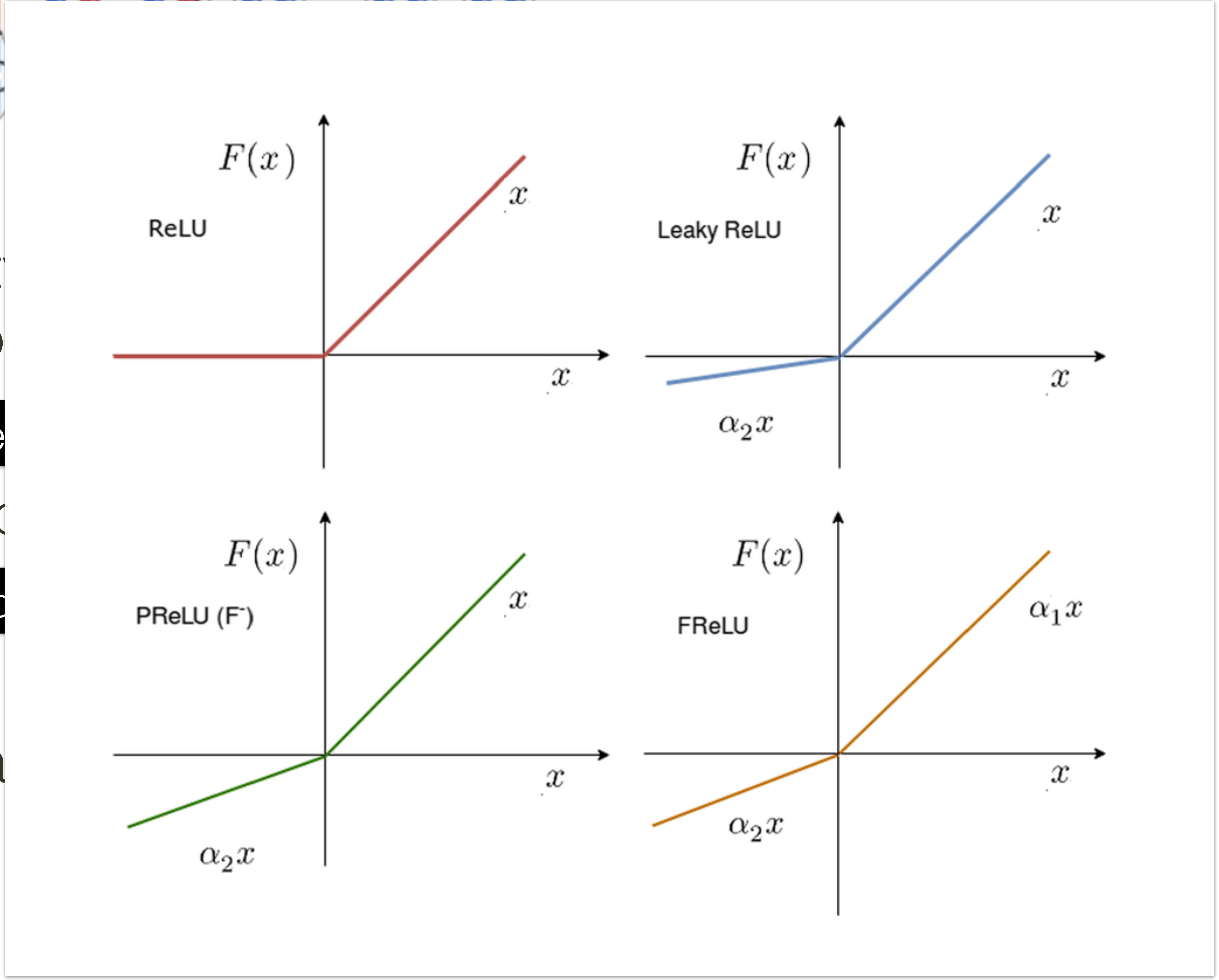
- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

Deep Learning **Terminology**



Google's "Inception" network



- **Network structure:** number and t dimensionality of each layer and co

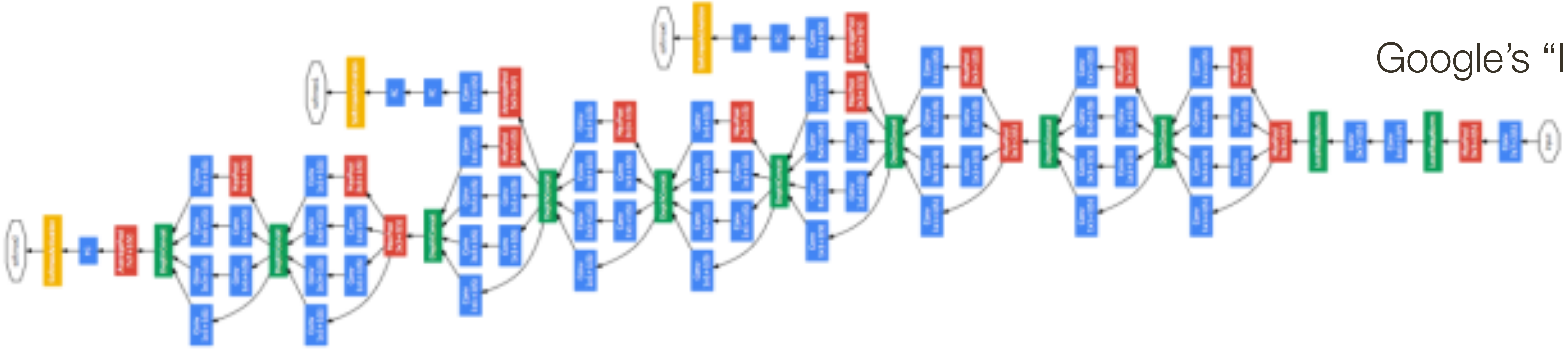
generally kept fixed, requires some knowledge

- **Loss function:** objective function b

requires knowled

- **Parameters:** trainable parameters linear/fc layers, parameters of the a

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

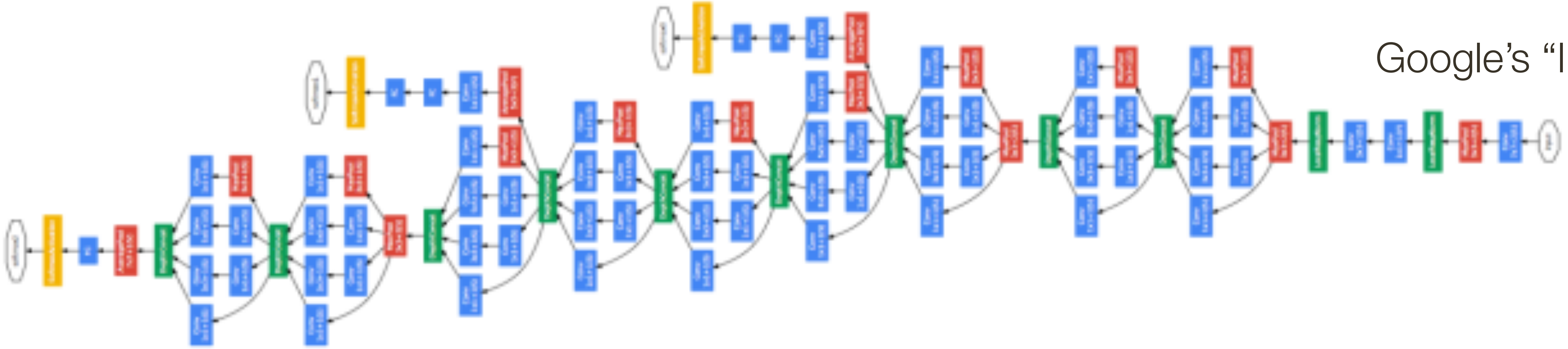
requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

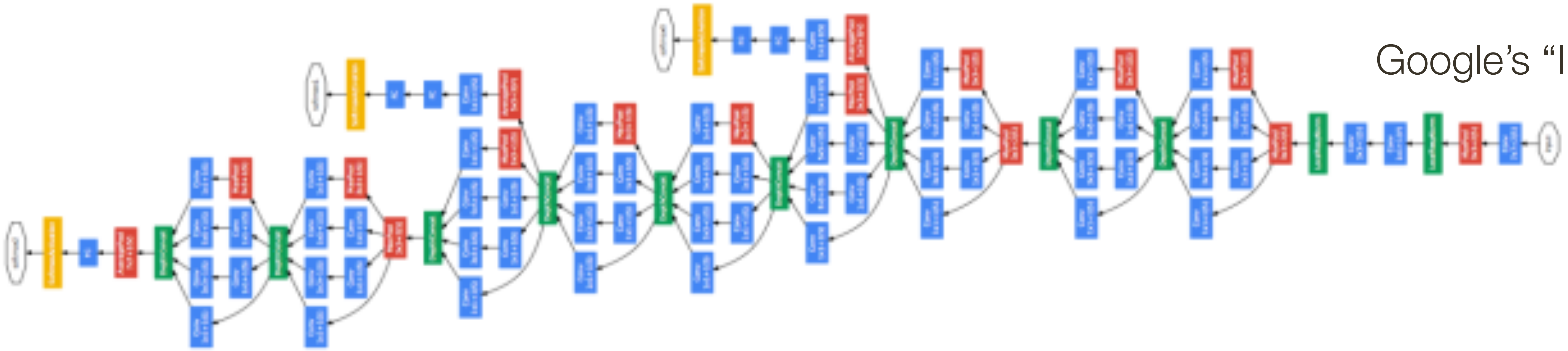
- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

grid search

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

Specification of neural architecture will define a **computational** graph.

Training

Initialize parameters of all layers

For a fixed number of iterations or until convergence

— Form **mini-batch** of examples (randomly chosen from a training dataset)

— Compute **forward** pass to make predictions for every example and compute the loss (this involves recursively calling forward() for each intermediate layer along computational graph)

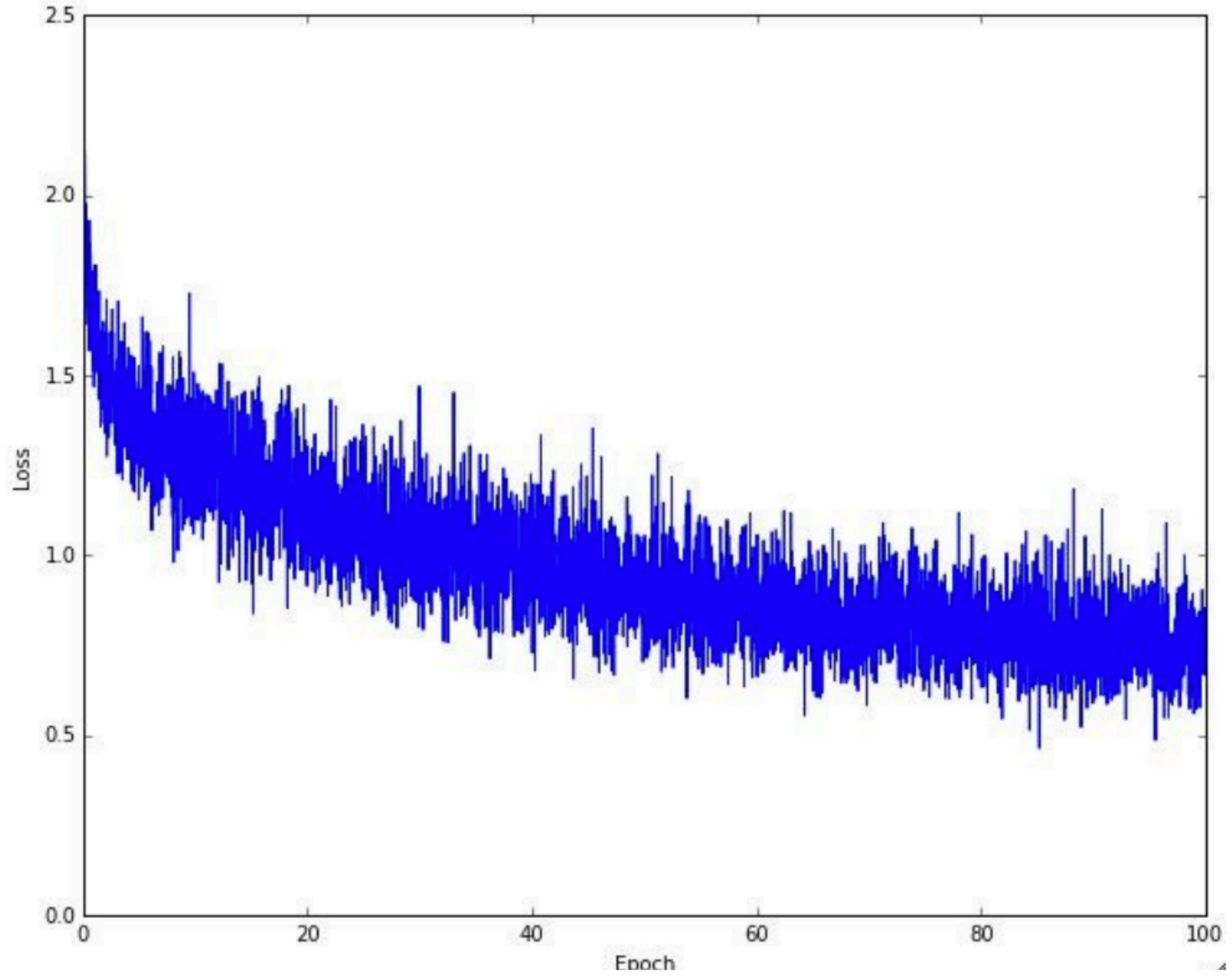
— Compute **backwards** pass to compute the gradient of the loss with respect to each parameter for each example (involves traversing computational graph in reverse order calling backward() on intermediate nodes and composing intermediate gradients — chain rule)

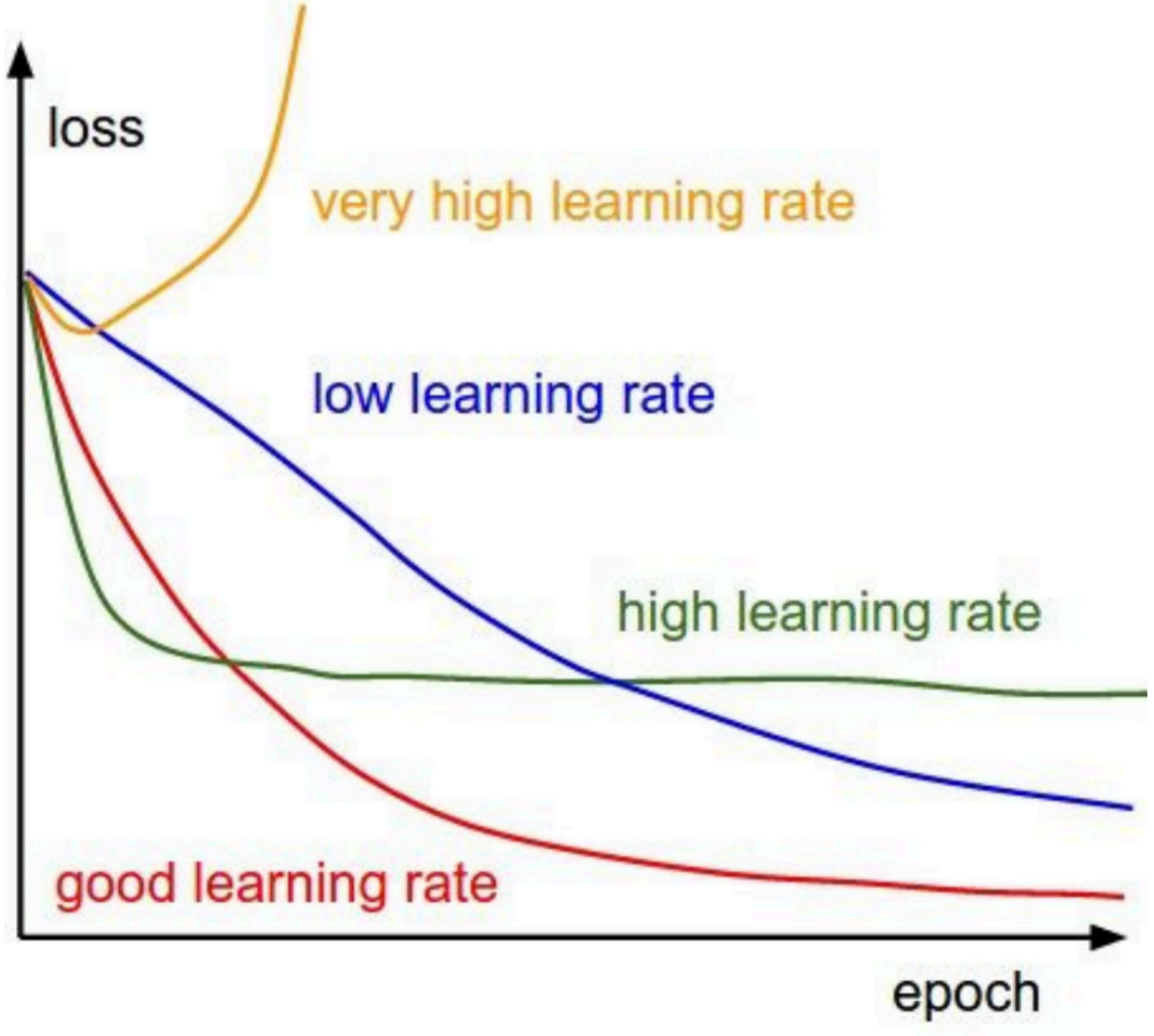
— **Update parameters** of all layers, by taking a step in the negative **average** gradient direction (computed over all examples in the mini-batch)

Inference / Prediction

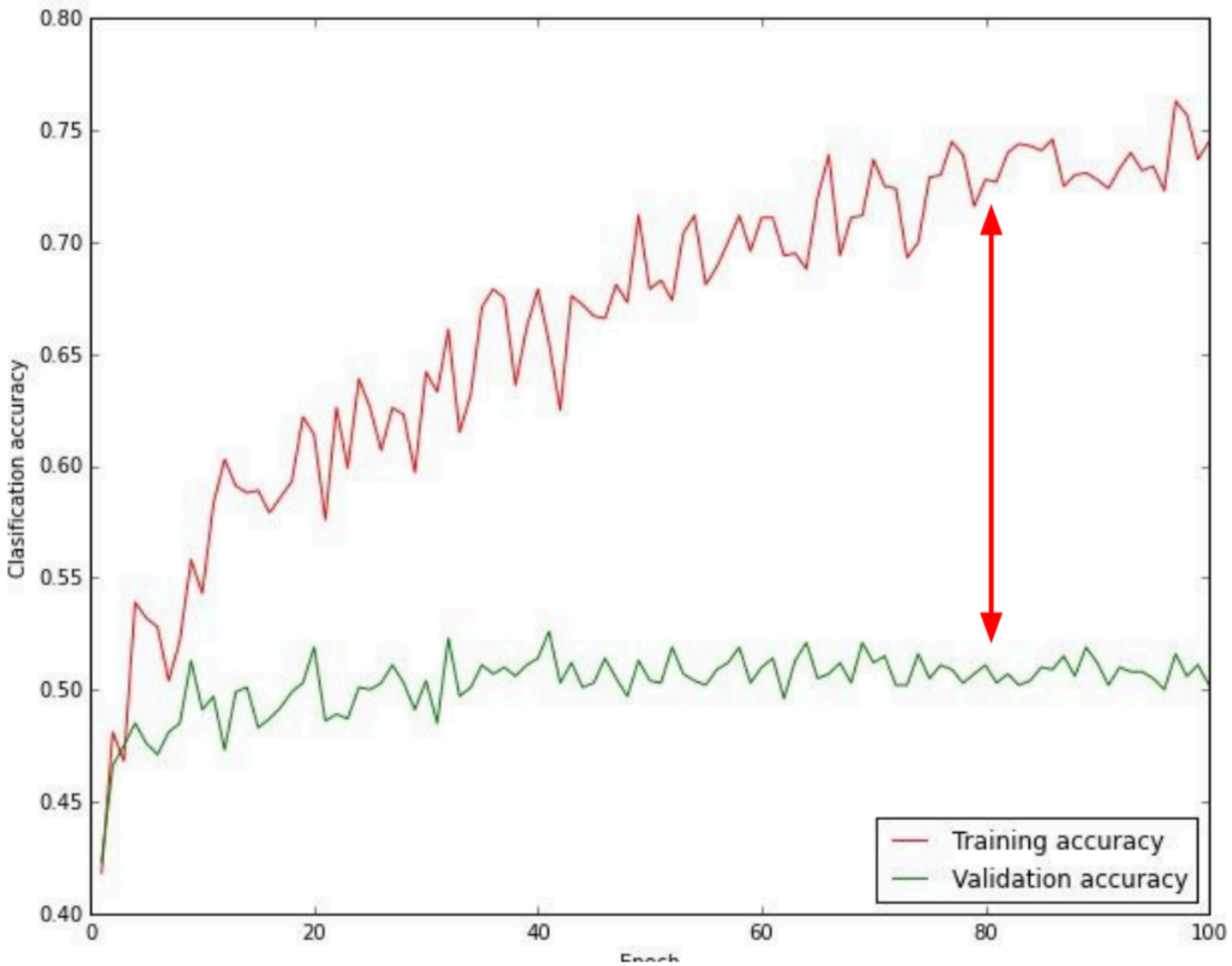
Compute **forward** pass with **optimized** parameters on test examples

Monitoring Learning: Visualizing the (training) loss





Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

Solution: increase regularization

No gap = undercutting

Solution: increase model capacity

Small gap = **ideal**

* slide from Li, Karpathy, Johnson's **CS231n at Stanford**