

CPSC 340: Machine Learning and Data Mining

More Deep Learning
Summer 2021

Admin

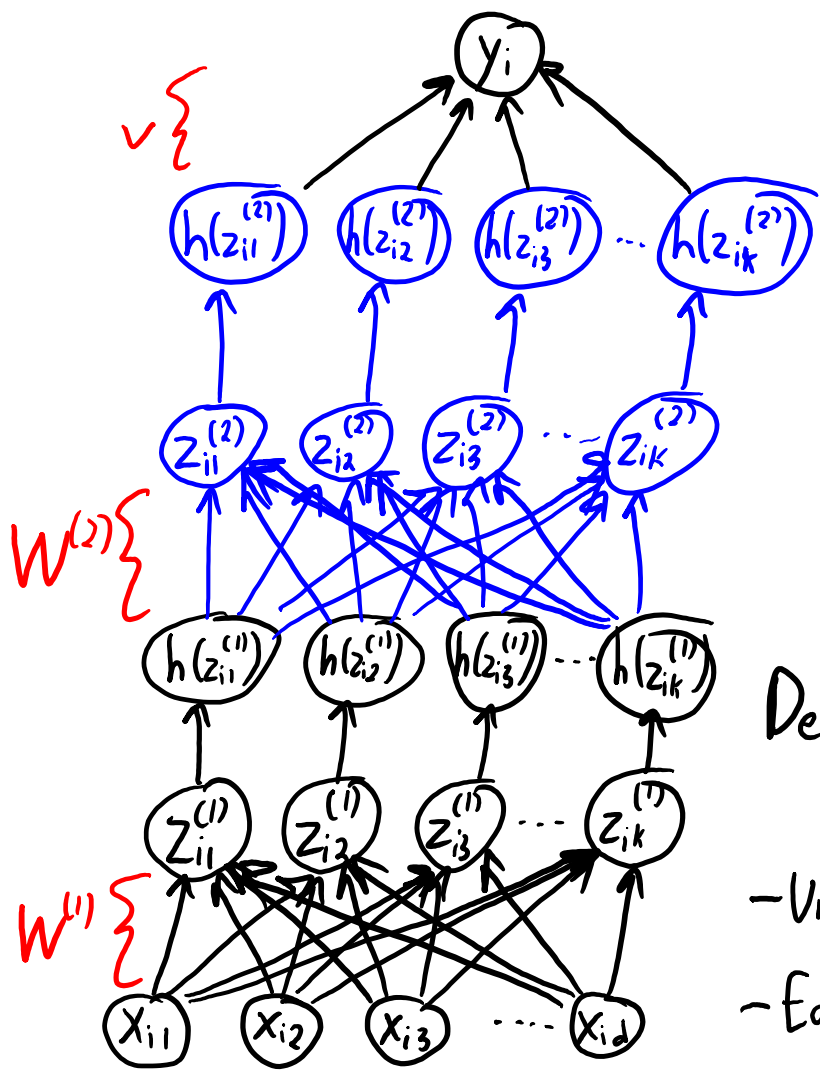
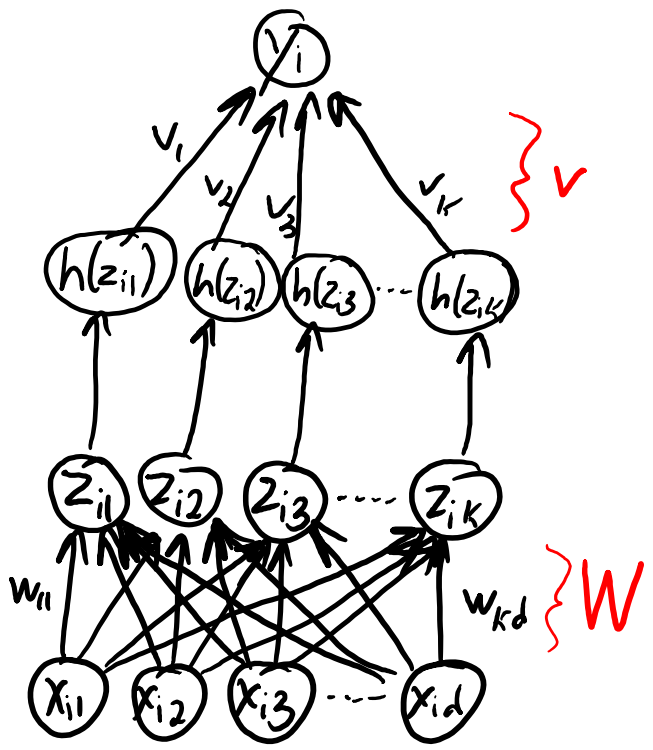
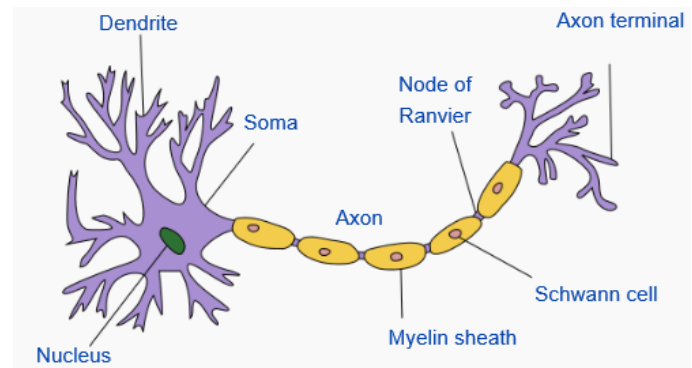
- Last day of classes!
 - Please fill out course evaluation
- Friday:
 - A6 due 11:55pm
 - A7 released (hopefully)
- Final Prep Megathread on Piazza
- Should we do an official review on Monday?
 - Do the Piazza poll

In This Lecture

1. ImageNet Challenge
2. Backpropagation
3. Training Neural Nets

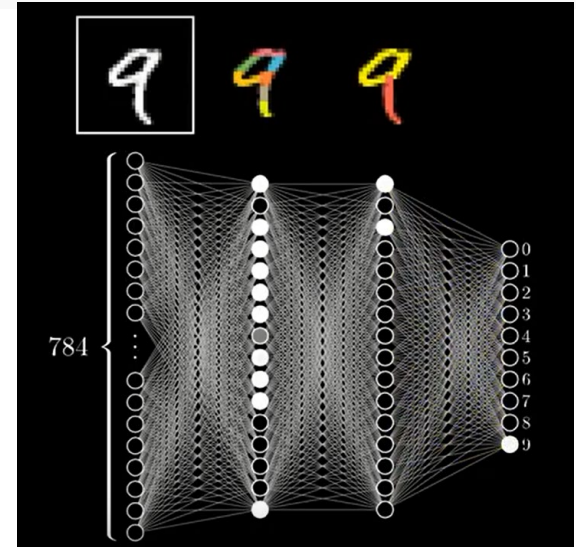
Neural network:

Last Time: Deep Learning



$y_i = v^T h(Wx_i)$
 Learn 'W' and 'v' together.
 - learn features for supervised learning.
 - Non-linear 'h' makes it a universal approximator for large 'K'

Deep neural networks:
 $y_i = v^T h(W^{(2)} h(W^{(1)} x_i))$
 - Unprecedented performance on difficult problems.
 - Each layer combines "parts" from previous layer.

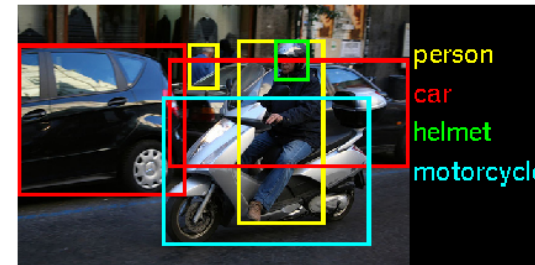
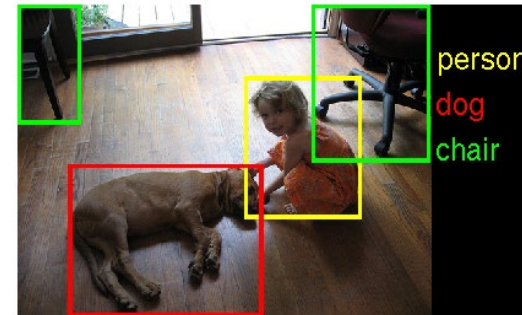
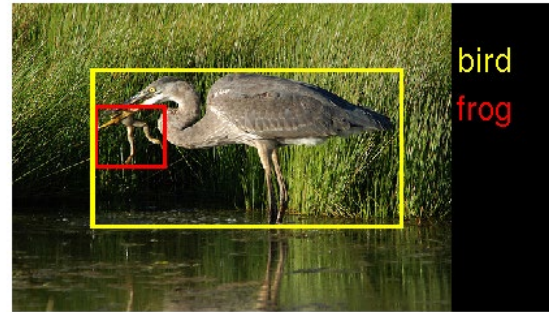


Coming Up Next

IMAGENET CHALLENGE

ImageNet Challenge

- Millions of labeled images, 1000 object classes.



Easy for humans but
hard for computers.

ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.

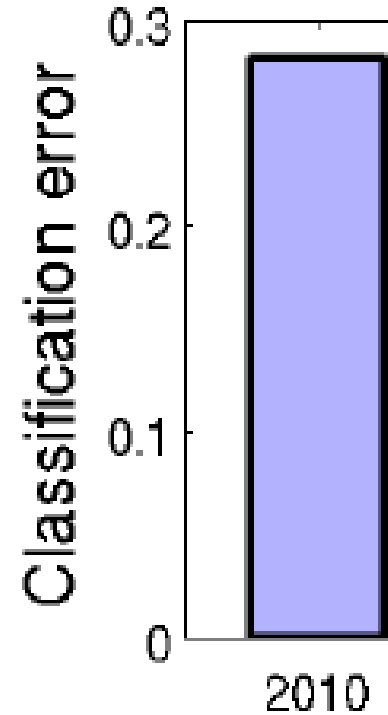


(a) Siberian husky



(b) Eskimo dog

Image classification



ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.

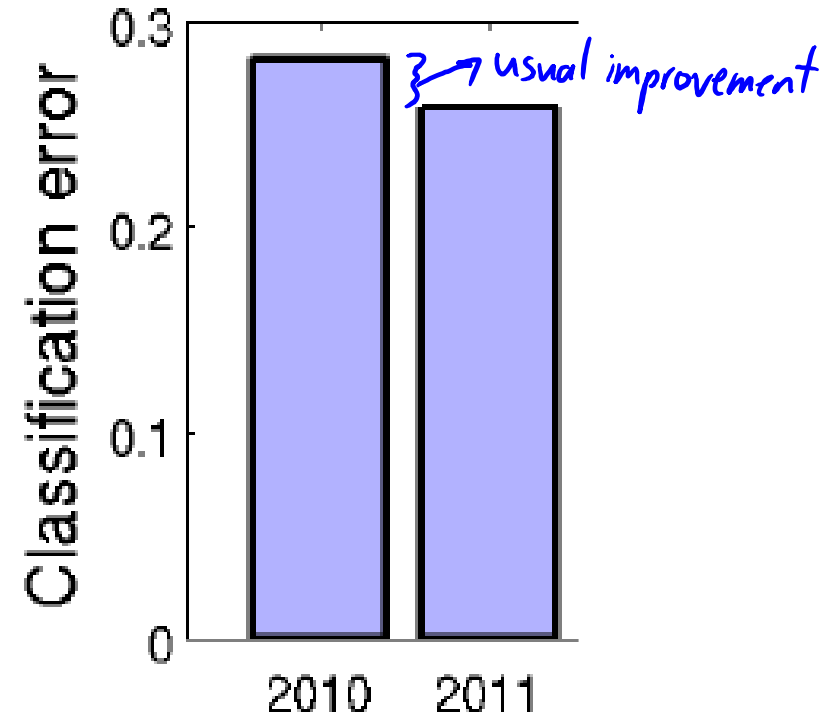


(a) Siberian husky



(b) Eskimo dog

Image classification



ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.

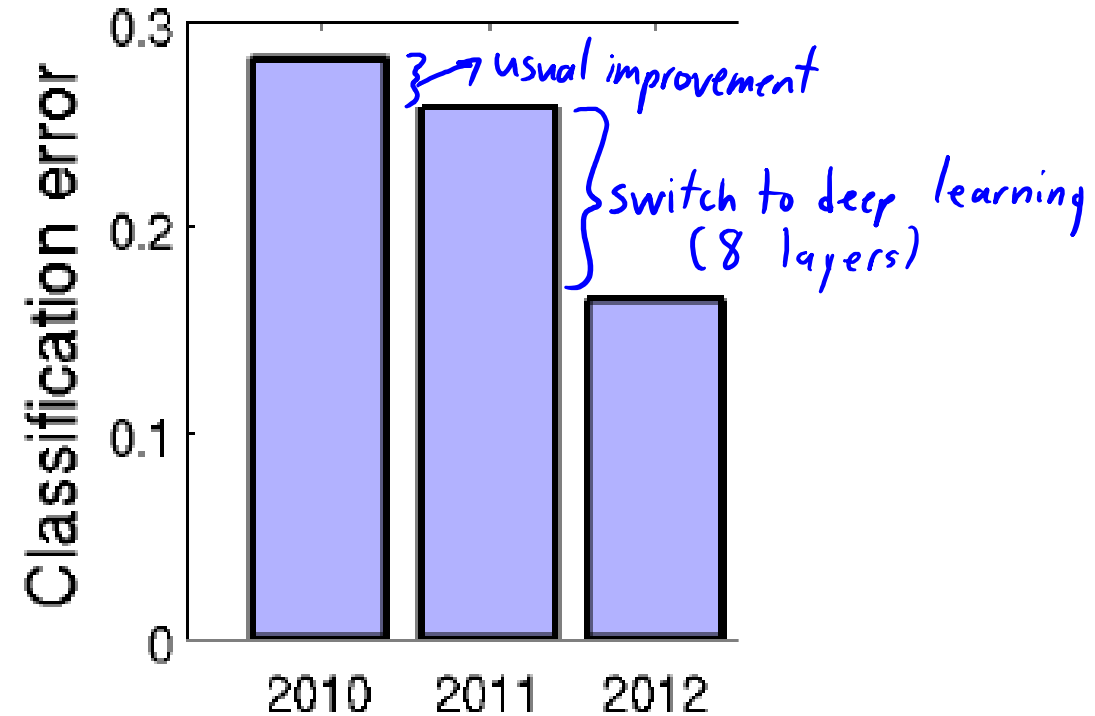


(a) Siberian husky



(b) Eskimo dog

Image classification



ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.

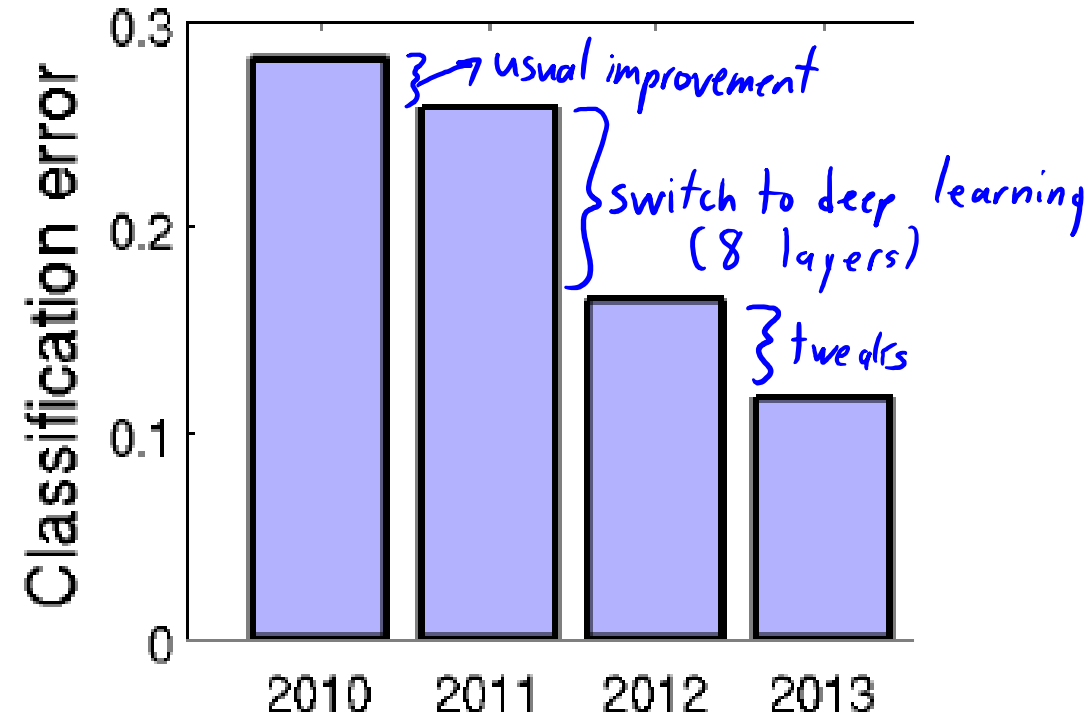


(a) Siberian husky



(b) Eskimo dog

Image classification



ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.

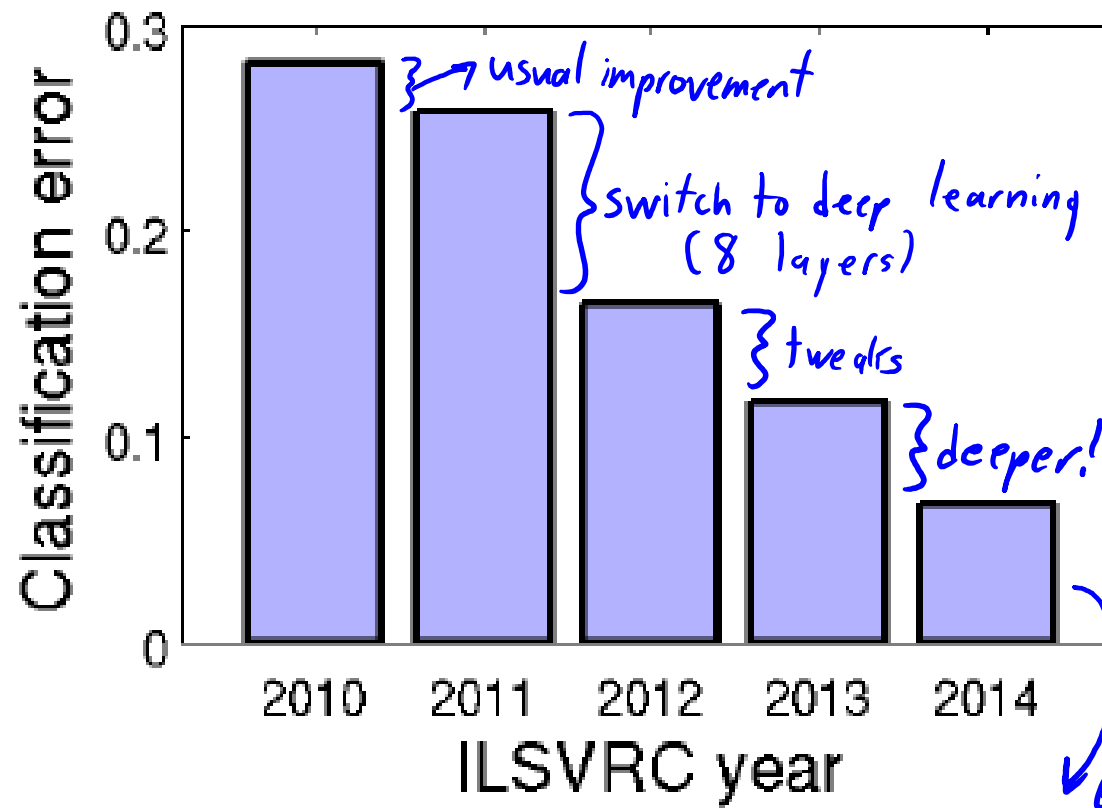


(a) Siberian husky



(b) Eskimo dog

Image classification

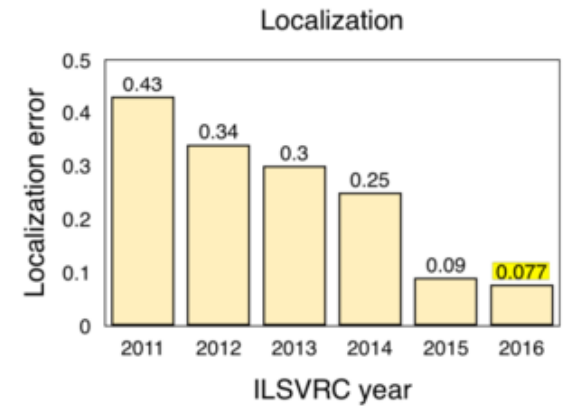
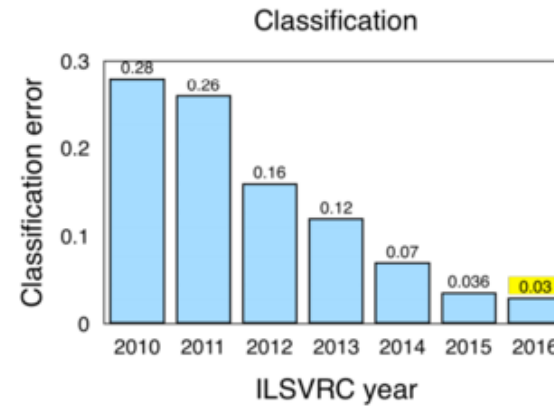


GoogleNet:
6.7% error
22 layers



ImageNet Challenge

- Object detection task:
 - Single label per image.
 - Humans: ~5% error.
- **2015: Won by Microsoft Asia**
 - 3.6% error.
 - 152 layers, introduced “ResNets”.
 - Also won “localization” (finding location of objects in images).
- **2016: Chinese University of Hong Kong:**
 - Ensembles of previous winners and other existing methods.
- **2017: fewer entries, organizers decided this would be last year.**



Deep Learning Practicalities

- This lecture focus on deep learning practical issues:
 - **Backpropagation** to compute gradients.
 - **Stochastic gradient** training.
 - **Regularization** to avoid overfitting.
- Next lecture:
 - Special 'W' restrictions to further avoid overfitting.

Coming Up Next

ADDING BIAS VARIABLES

Adding Bias Variables

- Recall fitting line regression with a **bias**:

$$\hat{y}_i = \sum_{j=1}^d w_j x_{ij} + \beta$$

- We did this by **adding** _____ to **X**.

**Q: How do we do this
with neural nets?**

Adding Bias Variables

- In neural networks we often want a **bias on the output**:

$$\hat{y}_i = \sum_{j=1}^d w_j x_{ij} + \beta$$

$$\hat{y}_i = \sum_{c=1}^k v_c h(w_c^T x_i) + \beta$$

- But we also often also include **biases on each z_{ic}** :

Q: How do we include bias to matrix multiplication?

Adding Bias Variables

- In neural networks we often want a **bias on the output**:

$$\hat{y}_i = \sum_{j=1}^d w_j x_{ij} + \beta$$

$$\hat{y}_i = \sum_{c=1}^K v_c h(w_c^T x_i) + \beta$$

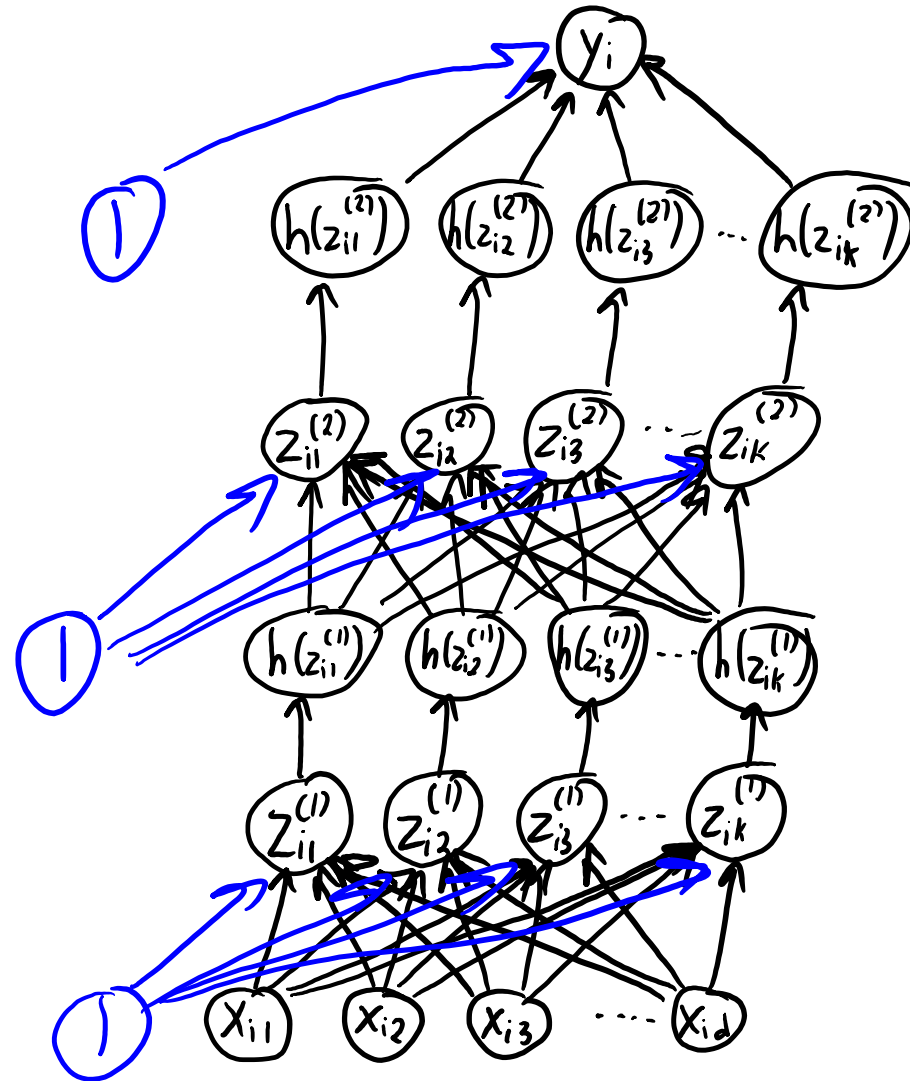
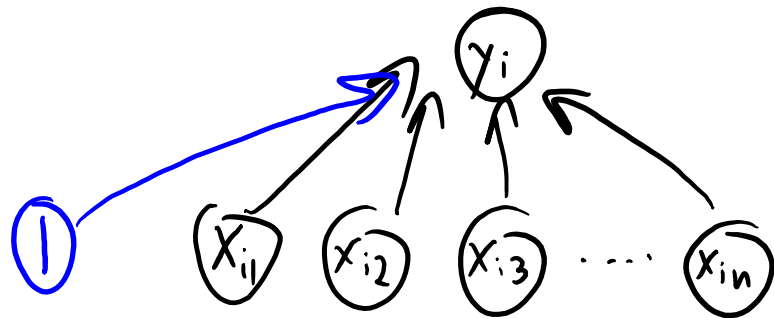
- But we also often also include **biases on each z_{ic}** :

$$\hat{y}_i = \sum_{c=1}^K v_c h(w_c^T x_i + \beta_c) + \beta$$

Q: What does it mean when β_c is large in positive/negative directions?

Adding Bias Variables

Linear model with bias:



Coming Up Next

MULTI-VARIATE CHAIN RULE AND BACKPROPAGATION

Training Neural Networks

- With squared loss and 1 hidden layer, our objective function is:

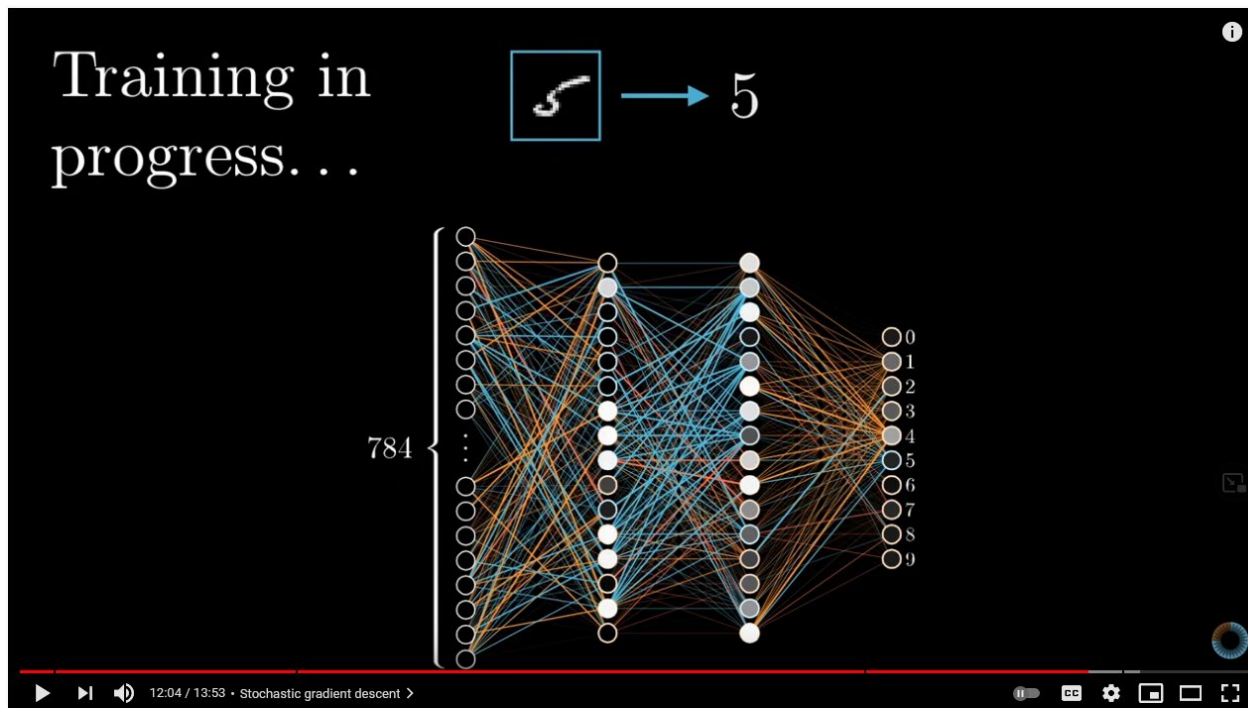
$$f(v, W) = \frac{1}{2} \sum_{i=1}^n (v^T h(Wx_i) - y_i)^2$$

- Usual training procedure: **stochastic gradient**.
 - Compute gradient of random example 'i', update both 'v' and 'W'.
 - **f is highly non-convex and optimizer can be difficult to tune.**

Q: How do we compute the gradients?

What is Backpropagation?

- Computing the gradient is known as “backpropagation”.
 - Video giving motivation [here](#).



3Blue1Brown series S3 · E3

What is backpropagation really doing? | Chapter 3, Deep learning

2,457,509 views · Nov 3, 2017

45K 375 SHARE SAVE ...



3Blue1Brown ✓

3.74M subscribers

What's actually happening to a neural network as it learns?

Next chapter: <https://youtu.be/tleHLnjs5U8>

Help fund future projects: <https://www.patreon.com/3blue1brown>

SHOW MORE

Andrej Karpathy's Backpropagation Lecture

The diagram on the slide illustrates a function f that takes two inputs, x and y , and produces an output z . The inputs x and y are labeled as "activations". The function f is represented by a circle containing the letter f . The partial derivatives of z with respect to x and y are shown as $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ respectively, and are collectively labeled as the "local gradient".

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 4 - 23 13 Jan 2016

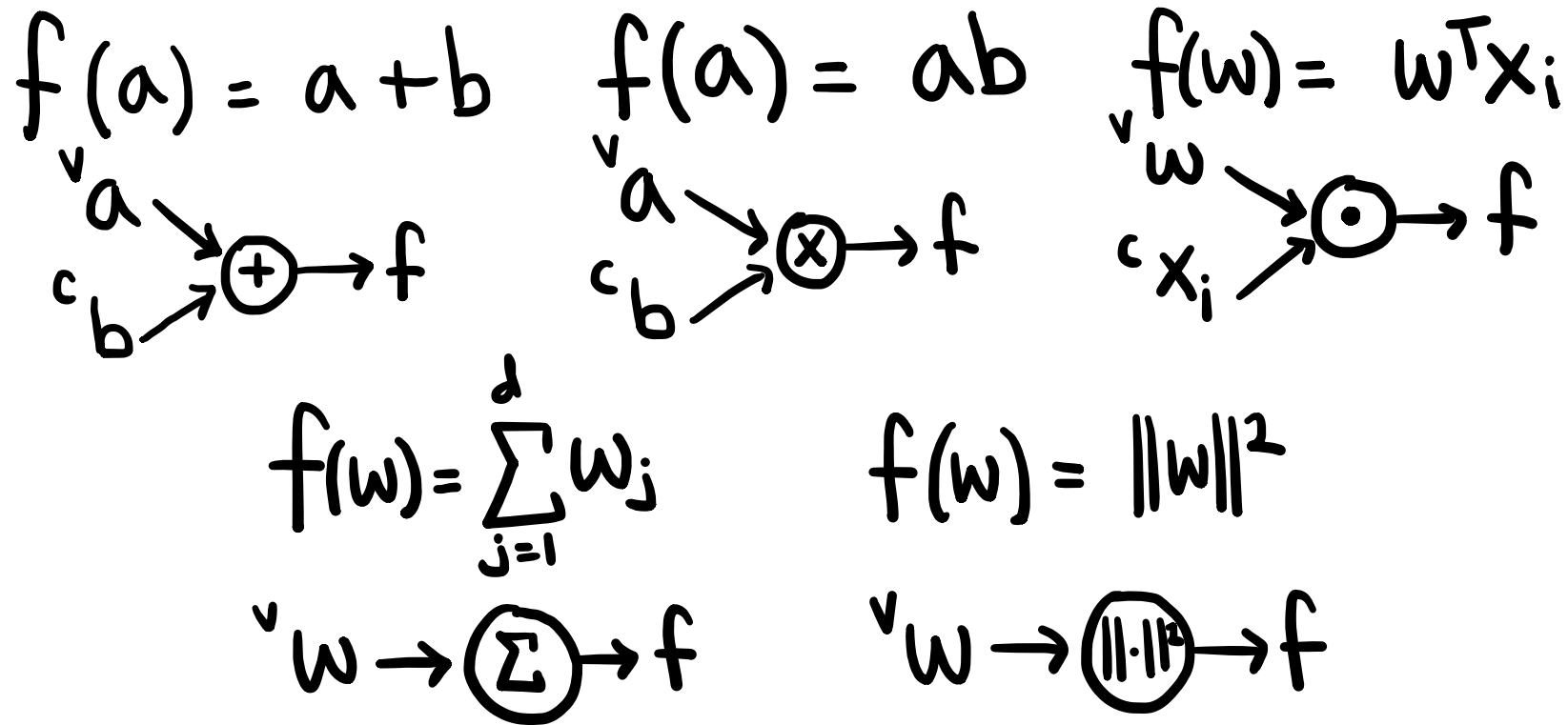
CS231n Winter 2016: Lecture 4: Backpropagation, Neural Networks 1
213,949 views · Jan 13, 2016

2.2K 19 SHARE SAVE ...

<https://www.youtube.com/watch?v=i94OvYb6noo>

Computational Graph

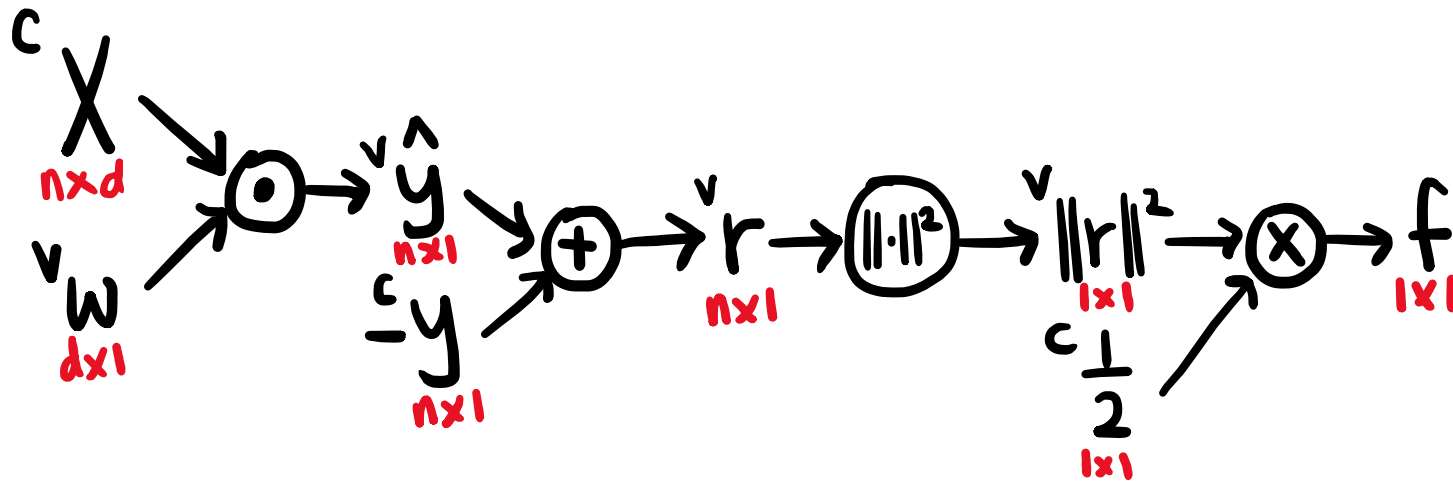
- “**Computational graph**”: directed graph showing operations between variables and constants



Computational Graph for Least Squares

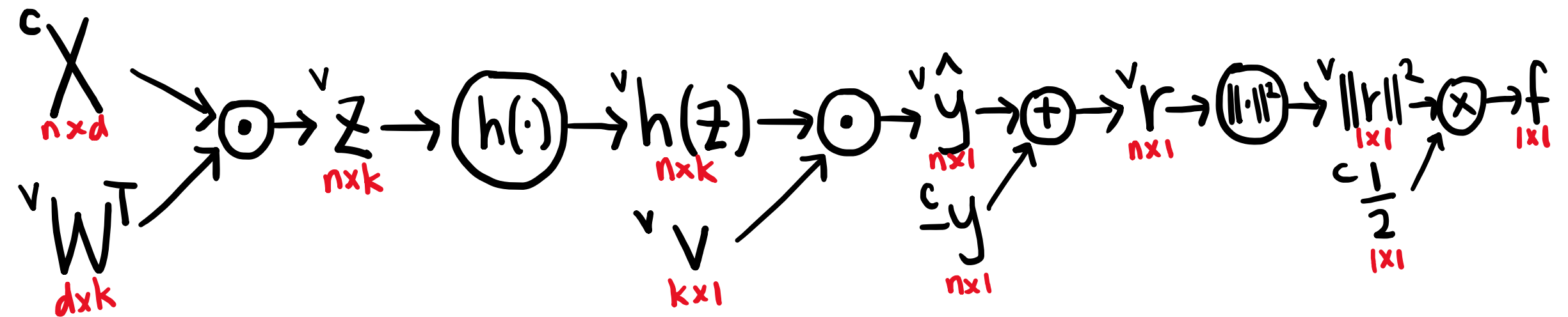
- Nodes are **variables/constants/results of operation**

$$f(w) = \frac{1}{2} \|Xw - y\|^2 = \frac{1}{2} \|r\|^2 = \frac{1}{2} \|\hat{y} - y\|^2$$



Computational Graph for Neural Nets

$$f(v, W) = \frac{1}{2} \|\hat{y} - y\|^2 \quad \text{where} \quad \hat{y} = v^T h(XW^T)$$



Jacobians

- Generalizes gradients for multi-output functions

$$f: \mathbb{R}^d \rightarrow \mathbb{R}$$

$$\nabla f(w) \quad \leftarrow \text{gradient}$$

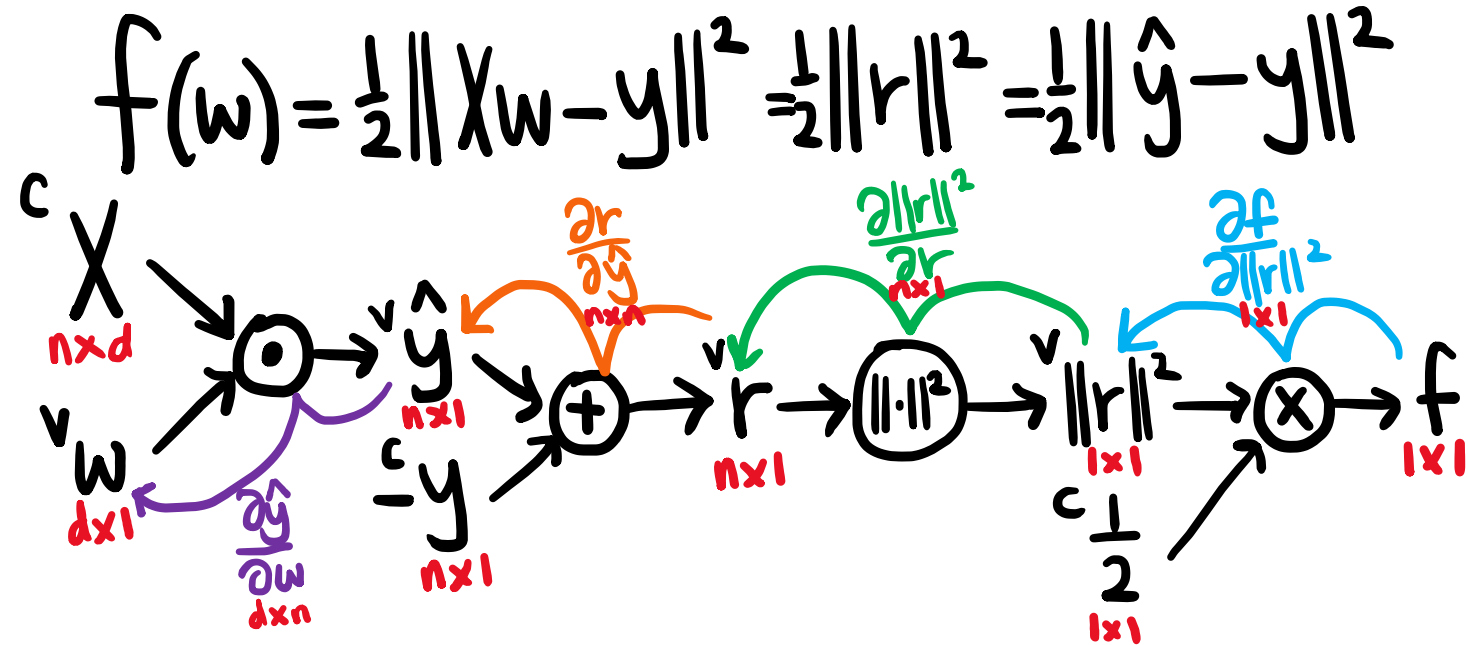
$$= \frac{\partial f}{\partial w} \quad \leftarrow \text{Jacobian}$$

$$\hat{y}: \mathbb{R}^d \rightarrow \mathbb{R}^n = \begin{bmatrix} \hat{y}_1(w) \\ \hat{y}_2(w) \\ \vdots \\ \hat{y}_n(w) \end{bmatrix}$$

$$\frac{\partial \hat{y}}{\partial w_j} =$$

$$= \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial w_j} & \frac{\partial \hat{y}_2}{\partial w_j} & \dots & \frac{\partial \hat{y}_n}{\partial w_j} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_1}{\partial w_m} & \frac{\partial \hat{y}_2}{\partial w_m} & \dots & \frac{\partial \hat{y}_n}{\partial w_m} \end{bmatrix}$$

Multi-Variate Chain Rule



$$\frac{\partial f}{\partial w} = \frac{\partial \hat{y}}{\partial w} \frac{\partial r}{\partial \hat{y}} \frac{\partial \|r\|^2}{\partial r} \frac{\partial f}{\partial \|r\|^2}$$

- Gradient is a **product of Jacobians!**
- **Chain rule** := recursive computation of Jacobians

Backpropagation (A7)

- Overview of how we compute neural network gradient:

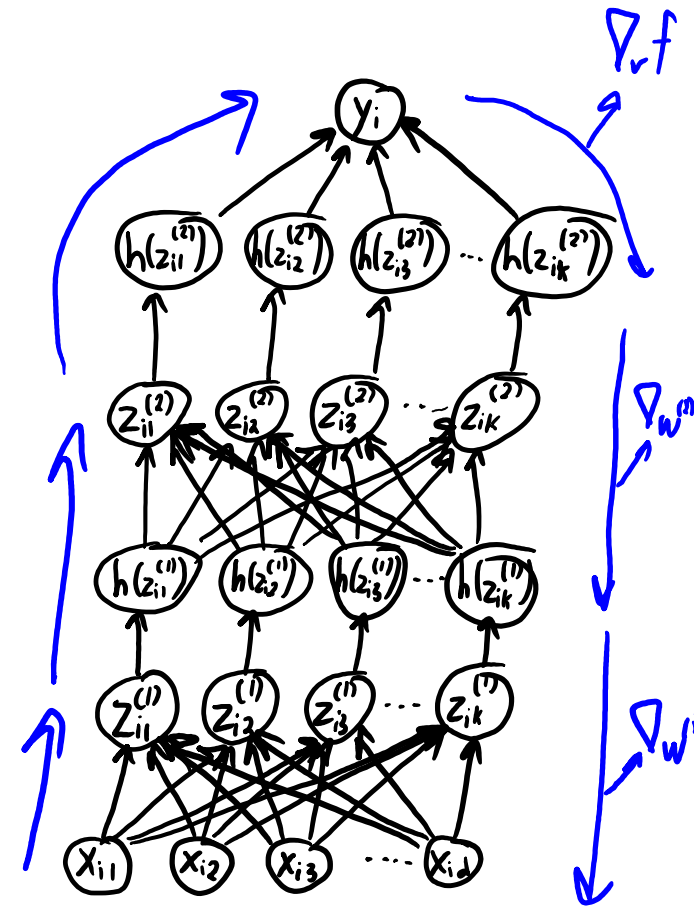
- **Forward propagation:**

- Compute $z_i^{(1)}$ from x_i .
- Compute $z_i^{(2)}$ from $z_i^{(1)}$.
- ...
- Compute \hat{y}_i from $z_i^{(m)}$, and use this to compute error.

- **Backpropagation:**

- Compute gradient with respect to regression weights 'v'.
- Compute gradient with respect to $z_i^{(m)}$ weights $W^{(m)}$.
- Compute gradient with respect to $z_i^{(m-1)}$ weights $W^{(m-1)}$.
- ...
- Compute gradient with respect to $z_i^{(1)}$ weights $W^{(1)}$.

- “Backpropagation” is _____ plus some _____.



Backpropagation

- Do you need to know how to do this?

Yes you should understand backprop



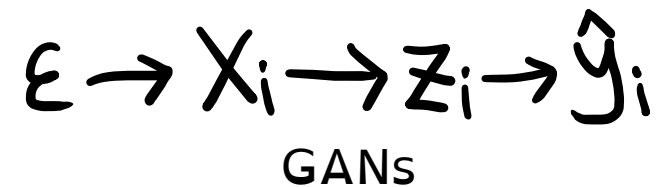
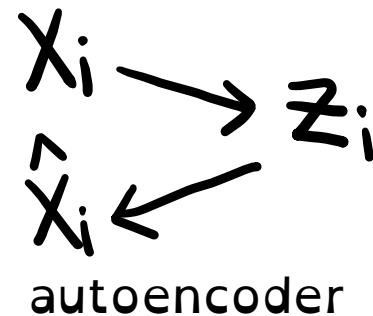
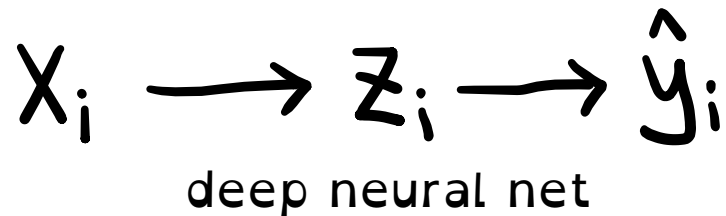
Andrej Karpathy Dec 19, 2016 · 7 min read



- Implementing backpropagation is usually part of **graduate courses involving deep learning** (e.g. CPSC 532/533 variants)

“Differentiable Programming”

- “Automatic differentiation” (A7) is standard practice for optimization
 - Get gradient with backward(), not by hand-crafted definition of ‘g’
- “Differentiable programming”: lets us focus on writing high-level code and reason about variable interactions without worrying about gradients
 - TensorFlow, Torch, MXNet, Theanos, Zygote, AutoGrad, etc.
- Coding on whiteboard with symbols is easy once things are differentiable



Backpropagation

- You should know cost of backpropagation:
 - Forward pass dominated by matrix multiplications by $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, and 'v'.
 - Backward pass has same cost as forward pass.

Multi-Output Models

- We've focused on single-output models so far:

$$\hat{y} = Xw \quad f(w) = \frac{1}{2} \|Xw - y\|^2$$

$n \times 1$ $n \times d$ $d \times 1$

- **Multi-output models** are straightforward (for 'q'-dimensional output):

$$\hat{Y} = XW^T \quad f(W) = \frac{1}{2} \|XW^T - Y\|_F^2$$

$n \times q$ $n \times d$ $d \times q$

- For neural networks, we replace the predictor's weights 'v' by a matrix
 - Many of our losses have "multi-output" equivalents
 - Softmax loss is multi-output logistic, "**cross entropy**" in neural network papers.

Deep Learning Vocabulary

- “**Deep learning**”: Models with many hidden layers.
 - Usually neural networks.
- “**Neuron**”: node in the neural network graph.
 - “**Visible unit**”: feature.
 - “**Hidden unit**”: latent factor z_{ic} or $h(z_{ic})$.
- “**Activation function**”: non-linear transform.
- “**Activation**”: $h(z_i)$.
- “**Backpropagation**”: compute gradient of neural network.
 - Sometimes “backpropagation” means “**training with SGD**”.
- “**Weight decay**”: L2-regularization.
- “**Cross entropy**”: softmax loss.
- “**Learning rate**”: SGD step-size.
- “**Learning rate decay**”: using decreasing step-sizes.
- “**Vanishing gradient**”: underflow/overflow during gradient calculation.

Coming Up Next

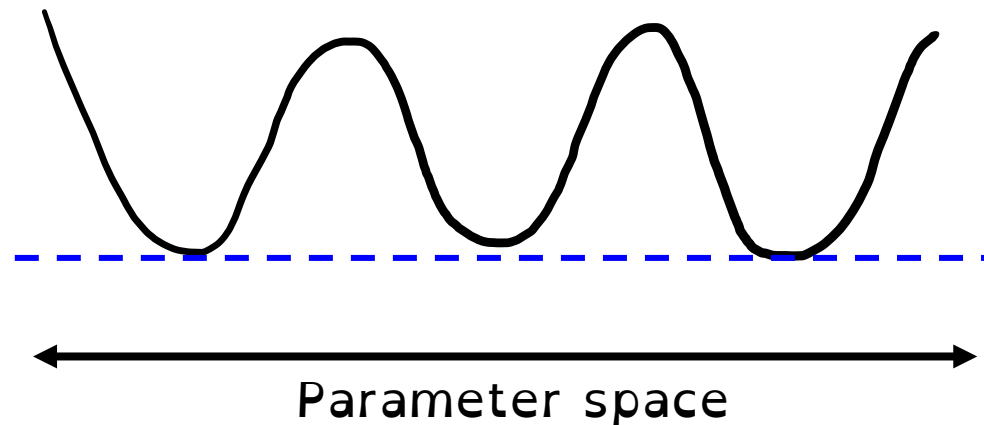
OPTIMIZATION OF NEURAL NETS

ImageNet Challenge and Optimization

- ImageNet organizer visited UBC summer 2015.
- “Besides huge dataset/model/cluster, what is the most important?”
 1. Data augmentation (translation, rotation, scaling, lighting, etc.).
 2. Optimization.
- Why would optimization be so important?
 - Neural network objectives are **highly non-convex** (and worse with depth).
 - Optimization has huge influence on quality of model.

Stochastic Gradient Training

- **Challenging to make SG work:**
 - Often doesn't work as a "black box" learning algorithm.
 - But people have developed a lot of tricks/modifications to make it work.
- **f is highly non-convex**, so are local minima the problem?
 - Some empirical/theoretical evidence that **local minima are not the problem**.
 - If the network is "deep" and "wide" enough, we think all local minima are good.
 - But it can be hard to get SG to close to a local minimum in reasonable time.



Parameter Initialization

- **Parameter initialization** is crucial:
 - Can't initialize weights in same layer to same value, or they will stay same.
 - Can't initialize weights too large, it will take too long to learn.
- A traditional **random initialization**:
 - Initialize bias variables to 0.
 - **Sample** from standard normal, divided by 10^5 (`0.00001*randn`).
 - `w = .00001*randn(k,1)`
 - Performing multiple initializations does not seem to be important.

```
272 self.W = scale * np.random.randn(output_dim, input_dim)
273 self.b = np.zeros(output_dim)
```

encoders.py

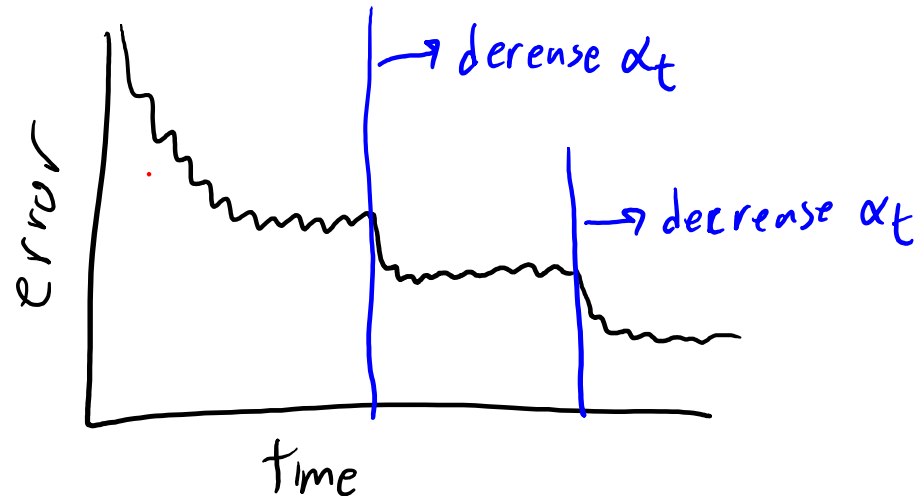
Standardization is Important

- Also common to **transform data** in various ways:
 - **standardize X**, “whiten”, standardize y.
- More recent initializations try to **standardize initial z_i** :
 - Use **different initialization in each layer**.
 - Try to **make variance of z_i the same across layers**.
 - Popular approach is to sample from standard normal, divide by $\sqrt{2 \cdot n_{\text{inputs}}}$.
 - Use samples from uniform distribution on $[-b, b]$, where

$$b = \frac{\sqrt{6}}{\sqrt{k^{(m)} + k^{(m-1)}}}$$

Setting the Step Size

- Stochastic gradient is **very sensitive to the step size** in deep models.
- Common approach: **manual “babysitting”** of the step-size.
 - Run SG for a while with a fixed step-size.
 - Occasionally measure error and plot progress:



- If error is not decreasing, decrease step-size.

Setting the Step-Size

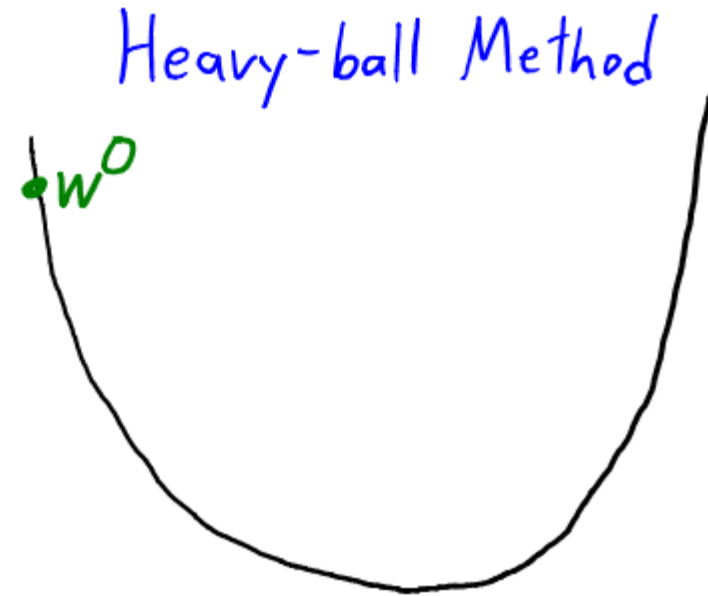
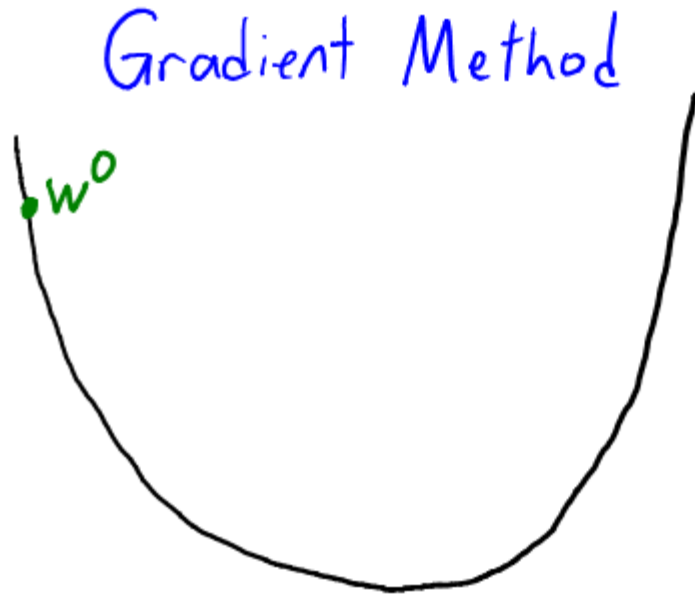
- Stochastic gradient is **very sensitive to the step size** in deep models.
- **Bias step-size multiplier**: use bigger step-size for the bias variables.
- **Momentum** (stochastic version of “heavy-ball” algorithm):
 - Add term that moves in previous direction:

$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) + \beta^t (w^t - w^{t-1})$$

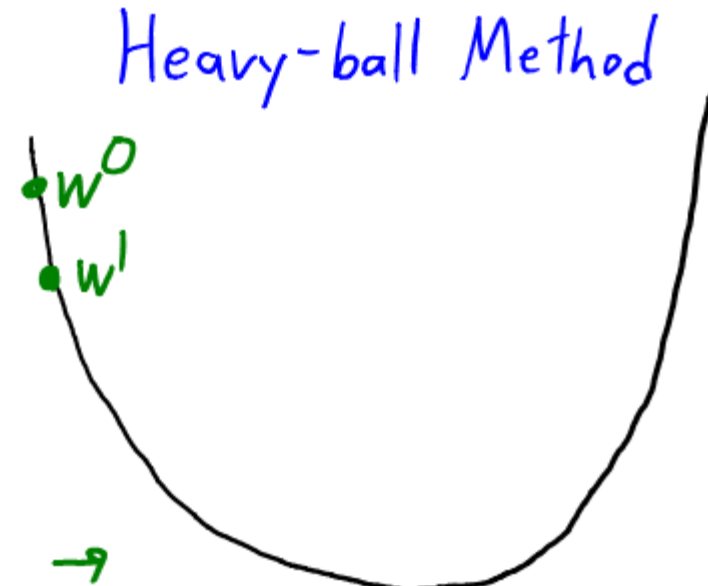
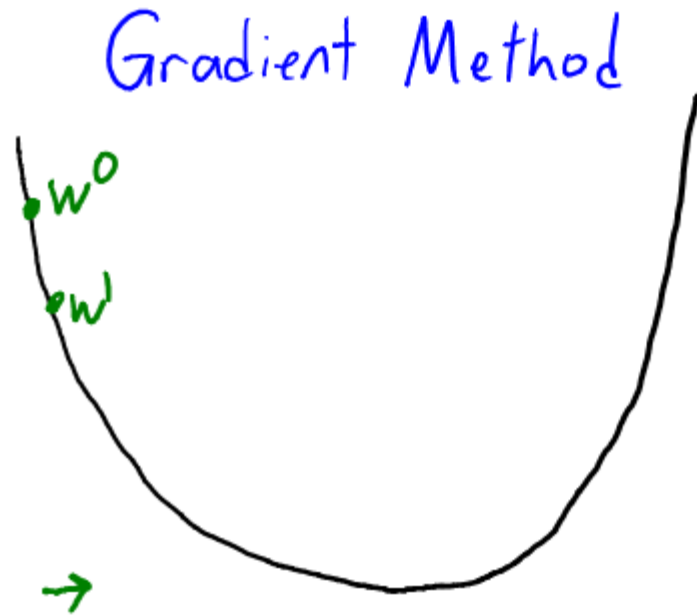
→ Keep going in the old direction

- Usually $\beta^t = 0.9$.

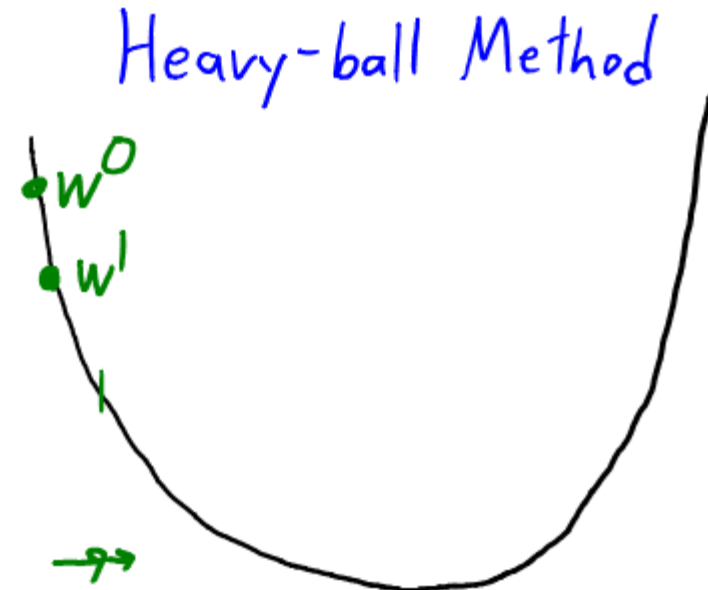
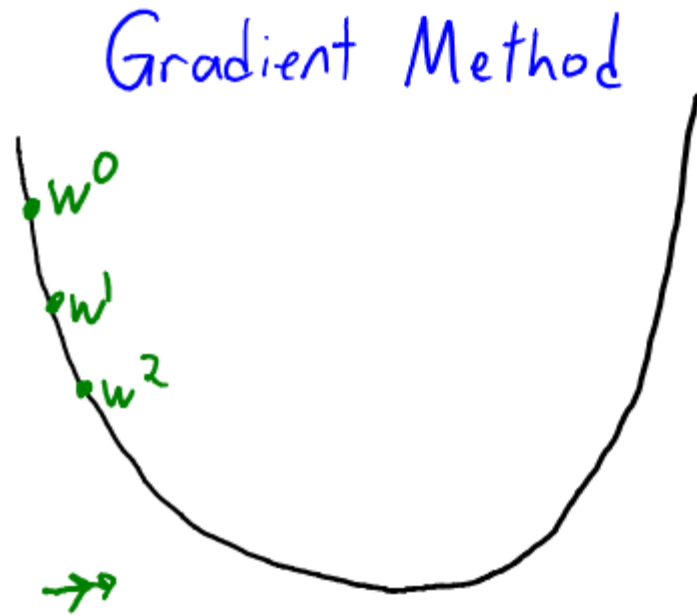
Gradient Descent vs. Heavy-Ball Method



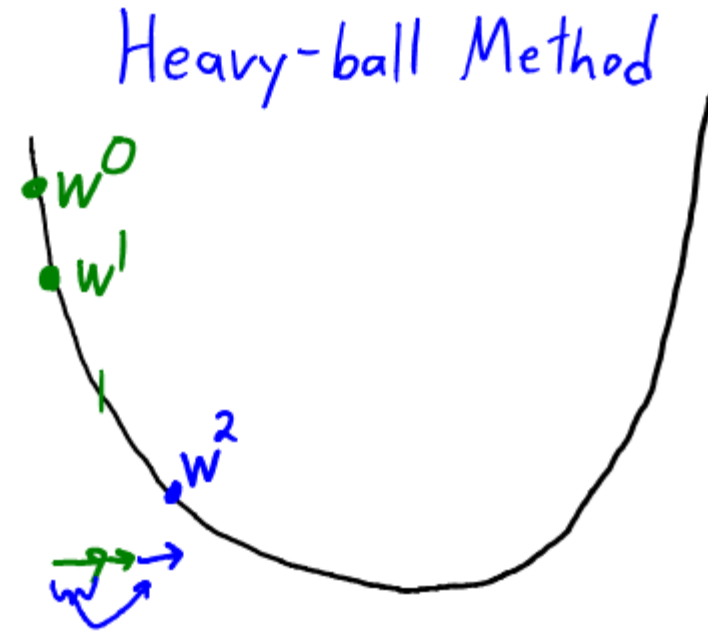
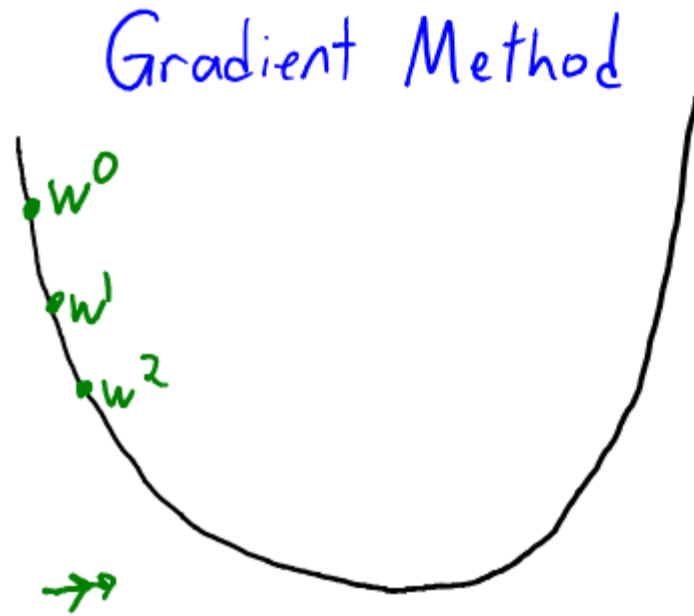
Gradient Descent vs. Heavy-Ball Method



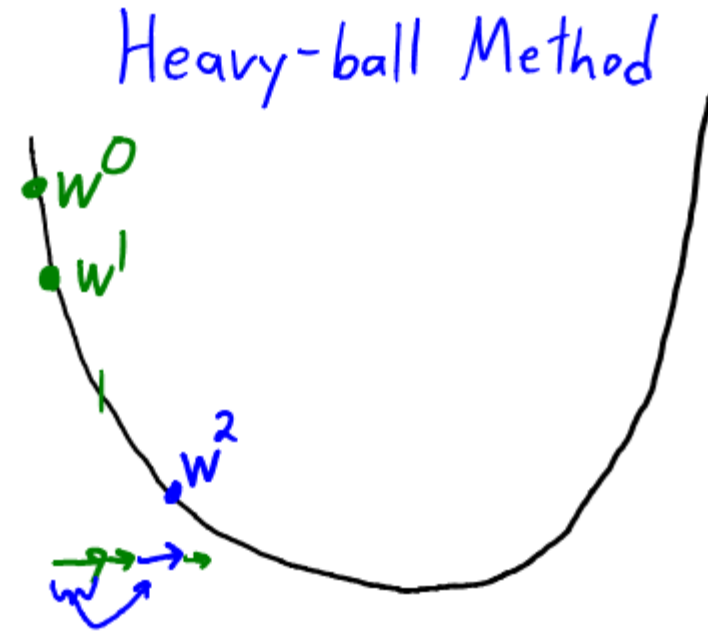
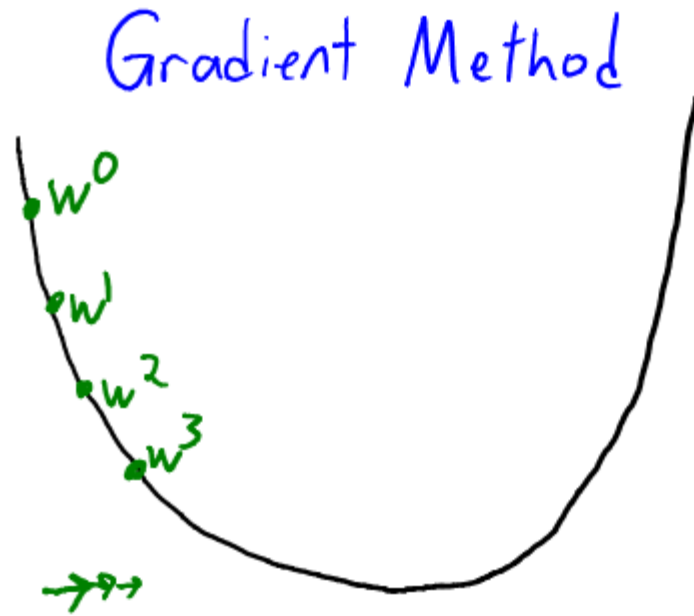
Gradient Descent vs. Heavy-Ball Method



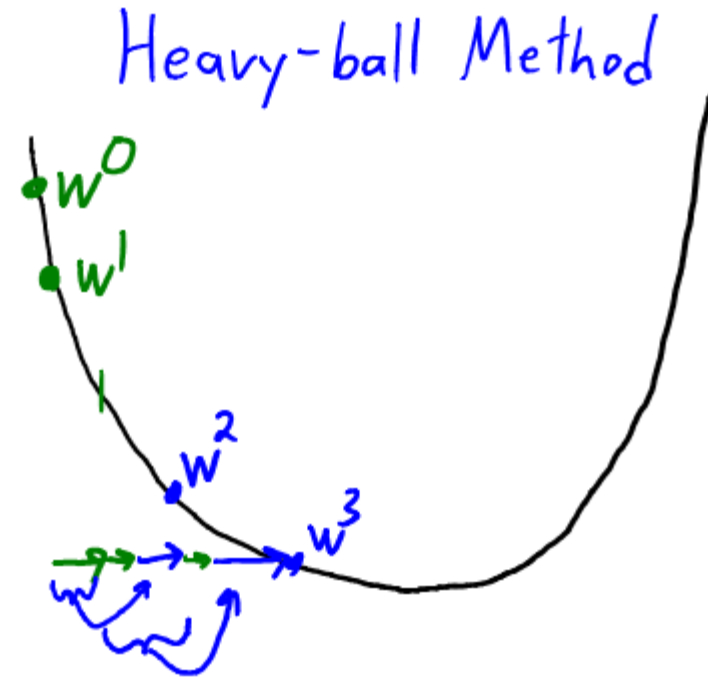
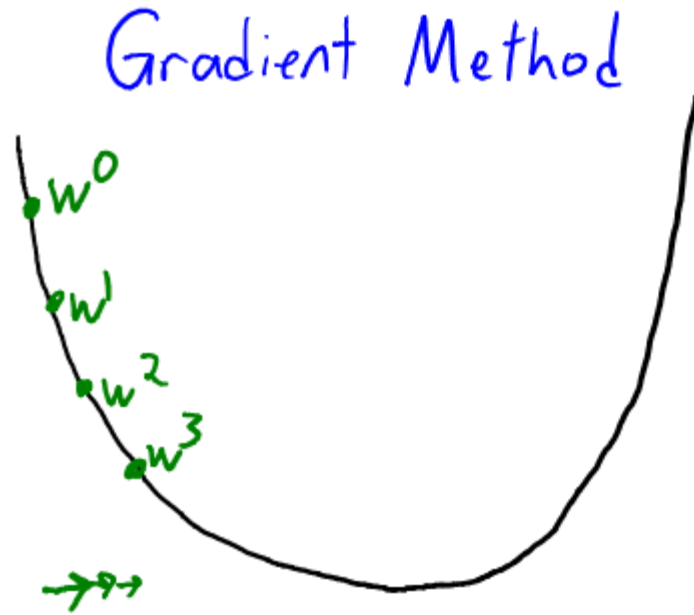
Gradient Descent vs. Heavy-Ball Method



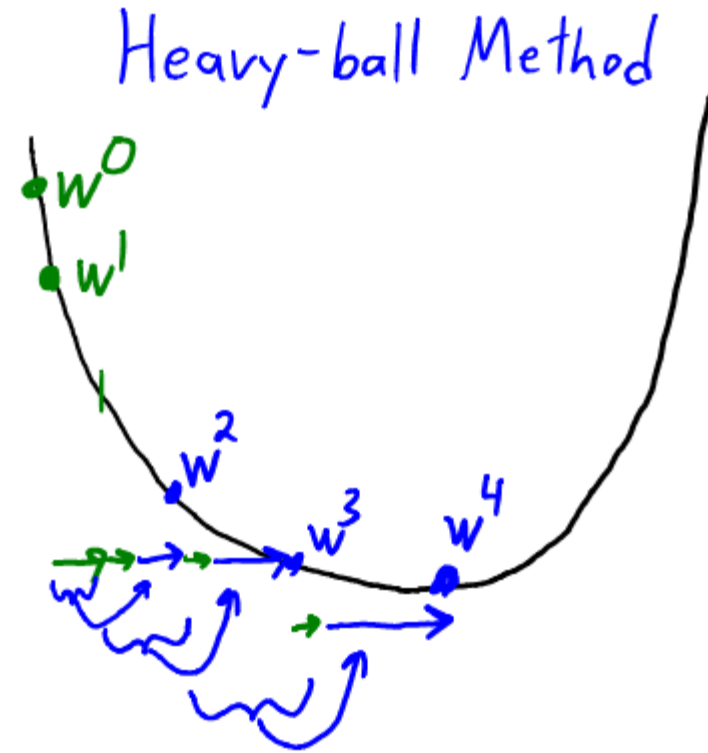
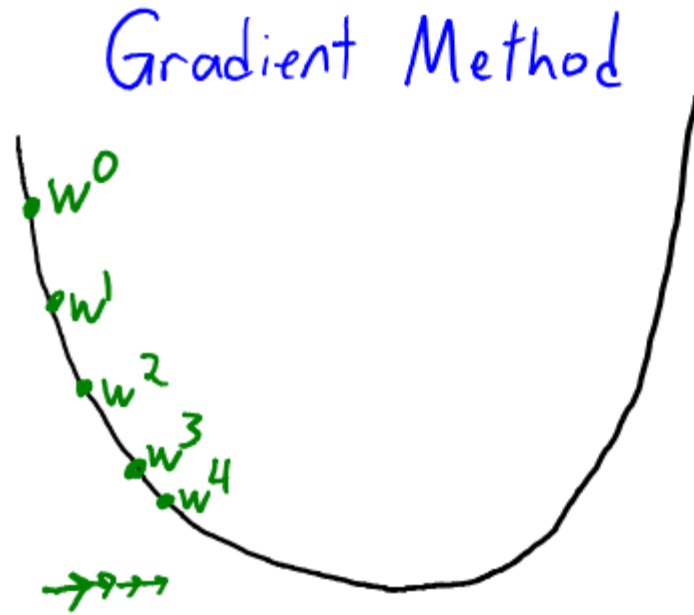
Gradient Descent vs. Heavy-Ball Method



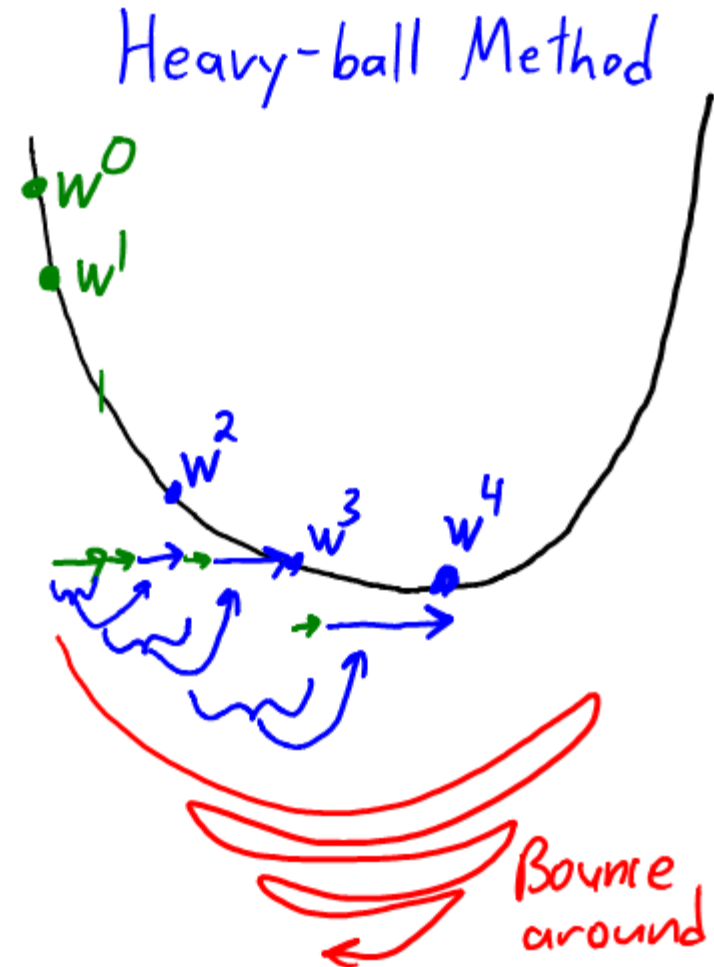
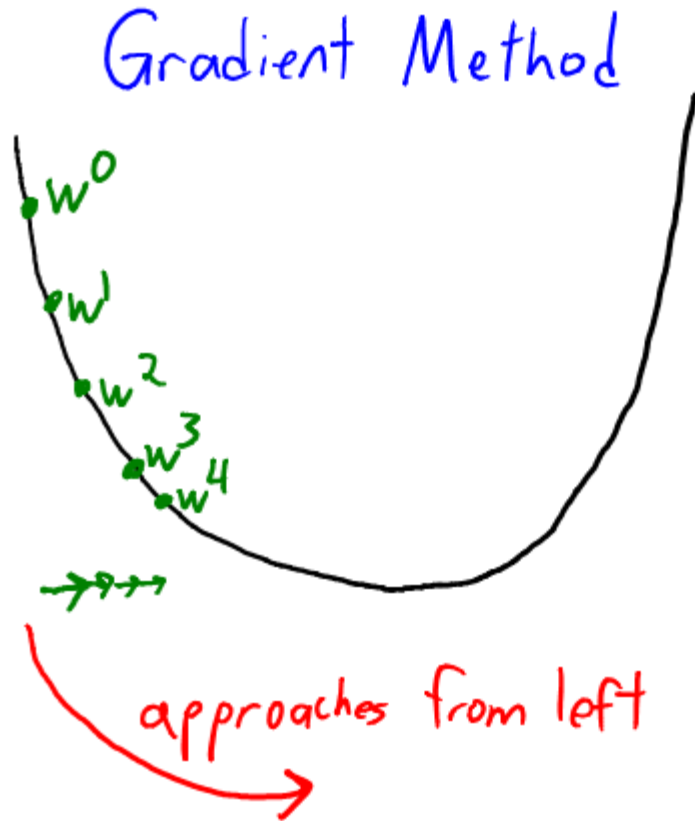
Gradient Descent vs. Heavy-Ball Method



Gradient Descent vs. Heavy-Ball Method



Gradient Descent vs. Heavy-Ball Method



Bottou Trick

- **Bottou Trick**: automated step size search
 1. Grab a small set of training examples (maybe 5% of total).
 2. Do a **binary search for a step size** that works well on them.
 3. Use this step size for a long time (or slowly decrease it from there).

SGD Variants and Batching

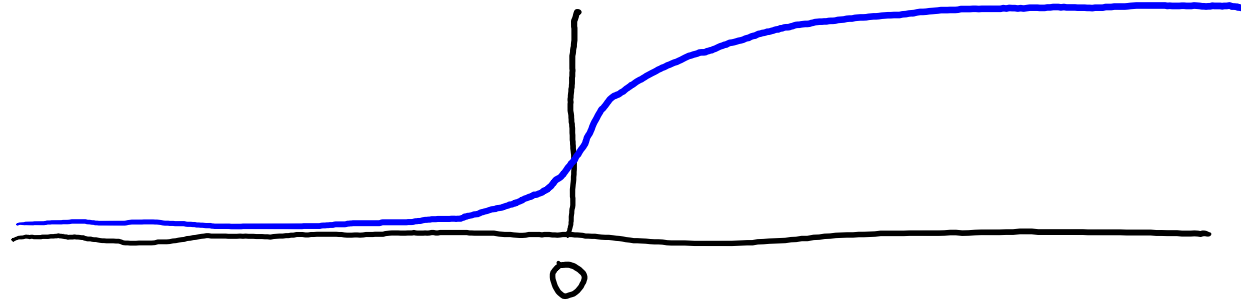
- Several recent SGD variants using a **step size for each variable**:
 - **AdaGrad, RMSprop, Adam** (often work better “out of the box”).
 - Seem to be losing popularity to vanilla SGD (often with momentum).
 - SGD often yields lower test (requires more tuning of step-size though)
- Batch size (number of random examples) also influences results.
 - Bigger batch sizes often give faster convergence but maybe to worse solutions?
- Another recent trick is **batch normalization**:
 - Try to “standardize” the hidden units within the random samples as we go.
 - Held as example of deep learning “**alchemy**” (blog post [here](#) about deep learning claims).
 - Sounds science-ey and often works but little theoretical justification/understanding.

Coming Up Next

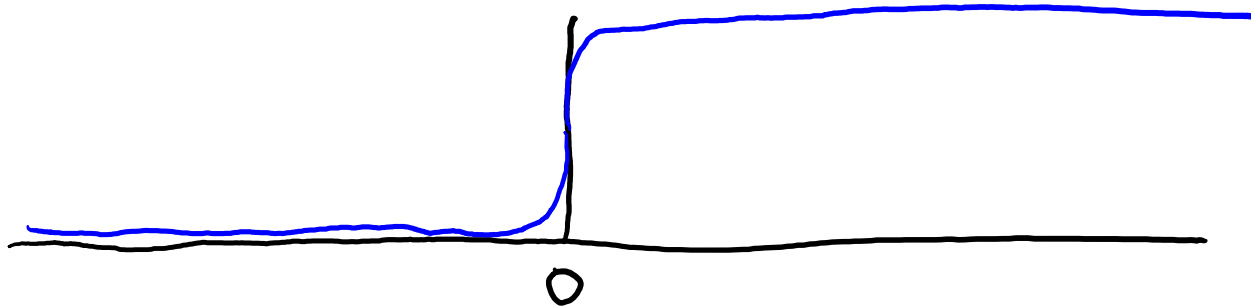
OTHER ACTIVATION FUNCTIONS

Vanishing Gradient Problem

- Consider the sigmoid function:



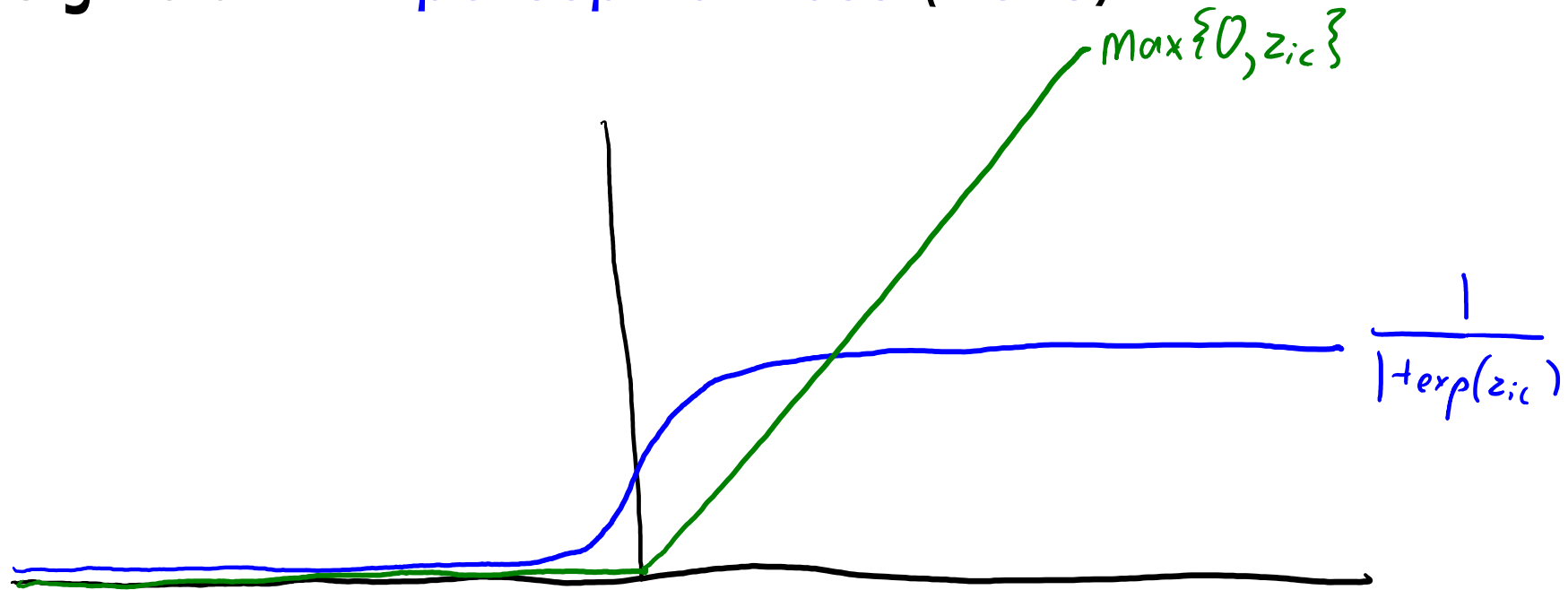
- Away from the origin, the **gradient is nearly zero**.
- The problem gets worse when you take the sigmoid of a sigmoid:



- In deep networks, many **gradients can be nearly zero everywhere**.

Rectified Linear Units (ReLU)

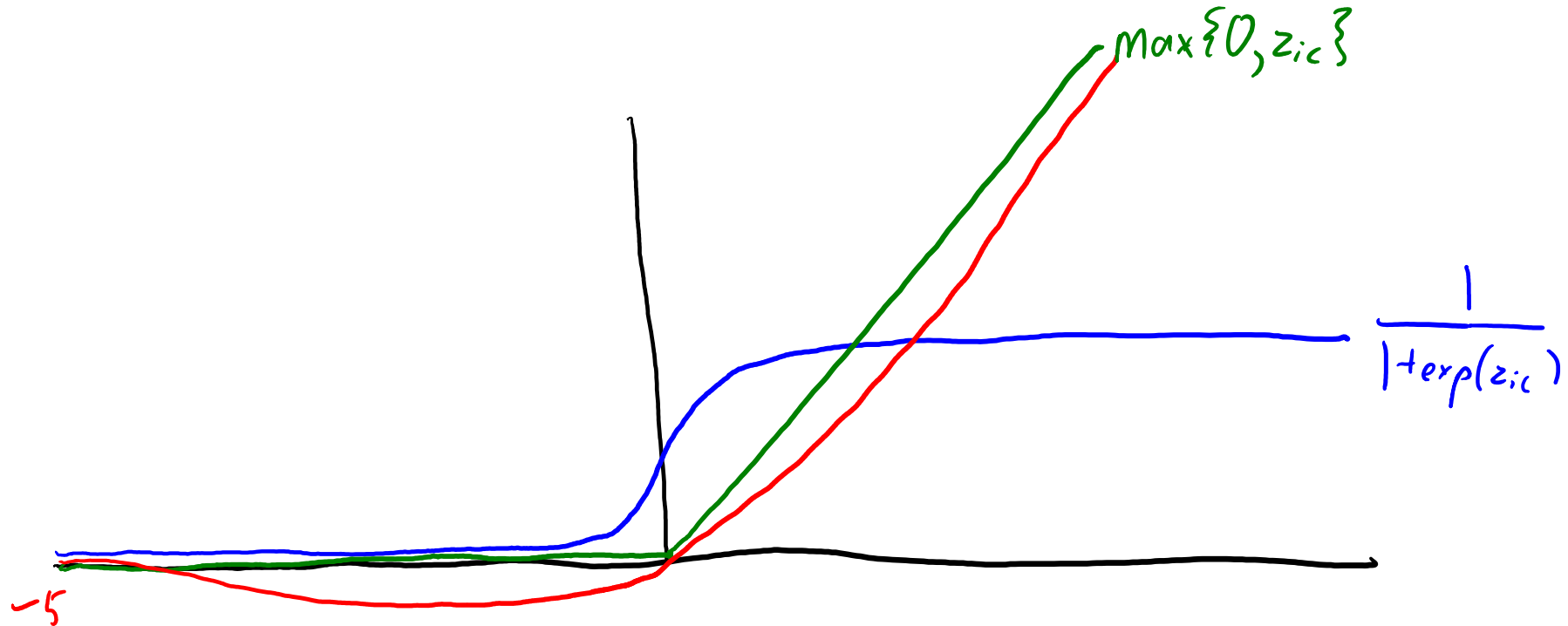
- Replace sigmoid with **perceptron loss (ReLU)**:



- **Just sets negative values z_{ic} to zero.**
 - Fixes vanishing gradient problem.
 - Gives sparser activations.
 - Not really simulating binary signal, but could be simulating “rate coding”.

“Swish” Activation

- Recent work searched for “best” activation:



- Found that $z_{ic}/(1+\exp(-z_{ic}))$ worked best (“swish” function).
 - A bit weird because it allows negative values and is non-monotonic.
 - But basically the same as ReLU when not close to 0.

Sigmoid



$$y = \frac{1}{1+e^{-x}}$$

Tanh



$$y = \tanh(x)$$

Step Function



$$y = \begin{cases} 0, & x < n \\ 1, & x \geq n \end{cases}$$

Softplus



$$y = \ln(1+e^x)$$

ReLU



$$y = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Softsign



$$y = \frac{x}{(1+|x|)}$$

ELU



$$y = \begin{cases} \alpha(e^x-1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

Log of Sigmoid



$$y = \ln\left(\frac{1}{1+e^{-x}}\right)$$

Swish



$$y = \frac{x}{1+e^{-x}}$$

Sinc



$$y = \frac{\sin(x)}{x}$$

Leaky ReLU



$$y = \max(0.1x, x)$$

Mish



$$y = x(\tanh(\text{softplus}(x)))$$

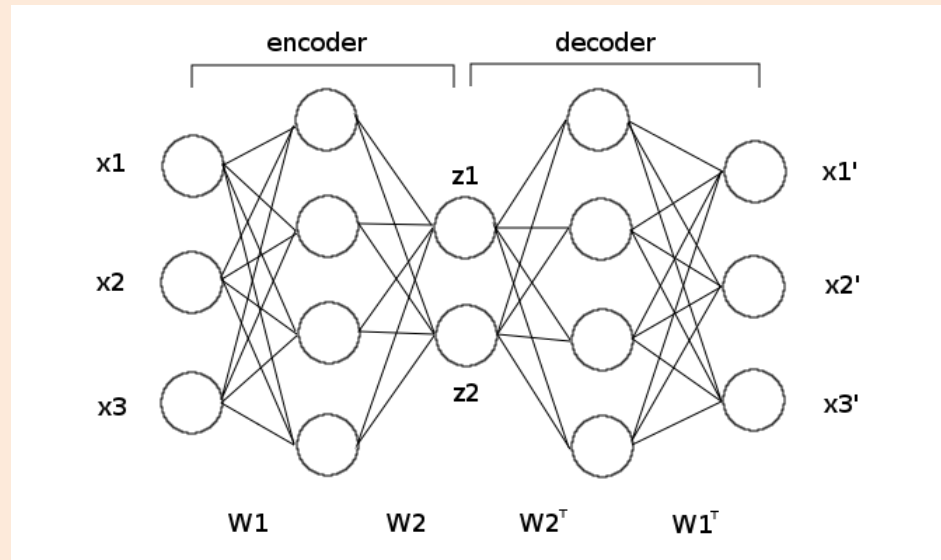
Summary

- **Unprecedented performance** on difficult pattern recognition tasks.
- **Backpropagation** computes neural network gradient via chain rule.
- **Parameter initialization** is crucial to neural net performance.
- **Optimization and step size** are crucial to neural net performance.
 - “Babysitting”, momentum.
- **ReLU** avoid “vanishing gradients”.
- Next time: The most important idea in computer vision?

Please do course evaluation!

Autoencoders

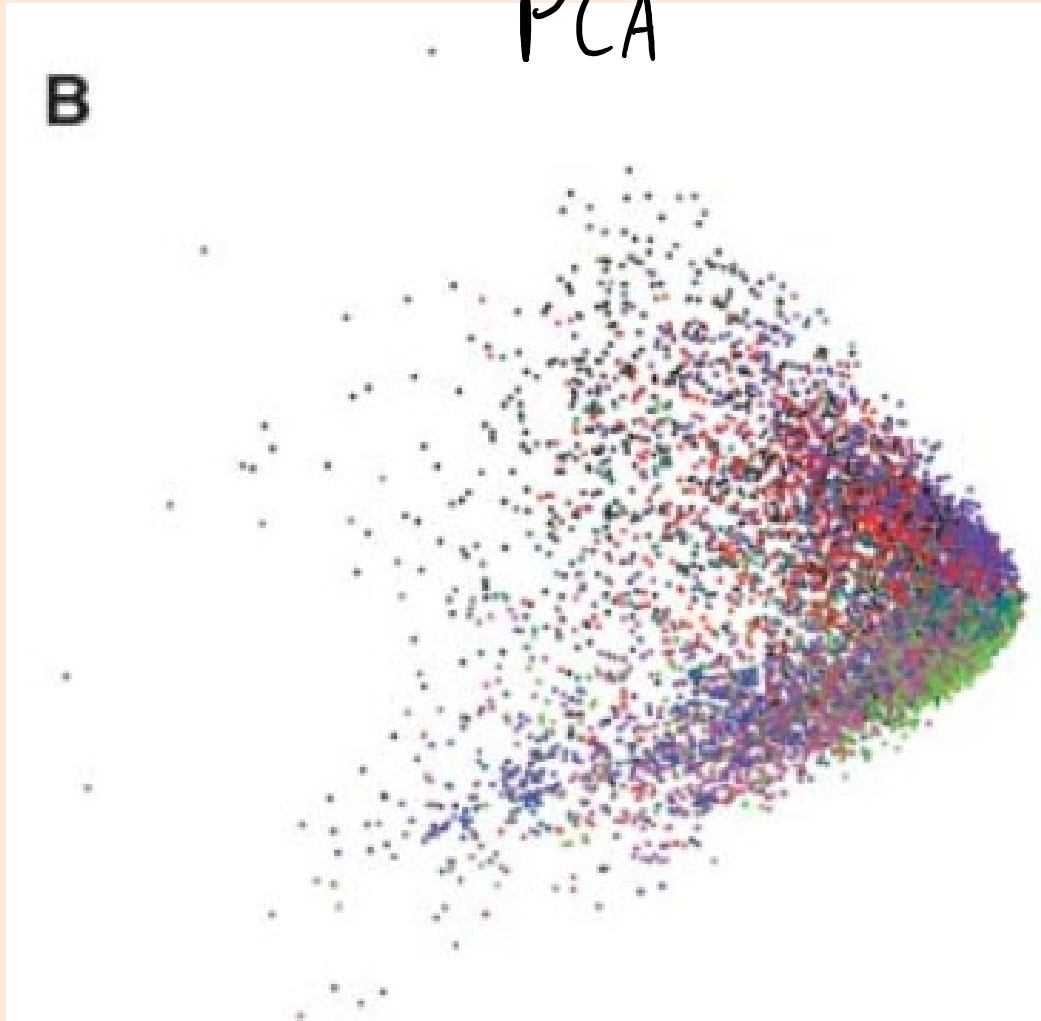
- **Autoencoders** are an **unsupervised deep learning** model:
 - Use the **inputs as the output** of the neural network.



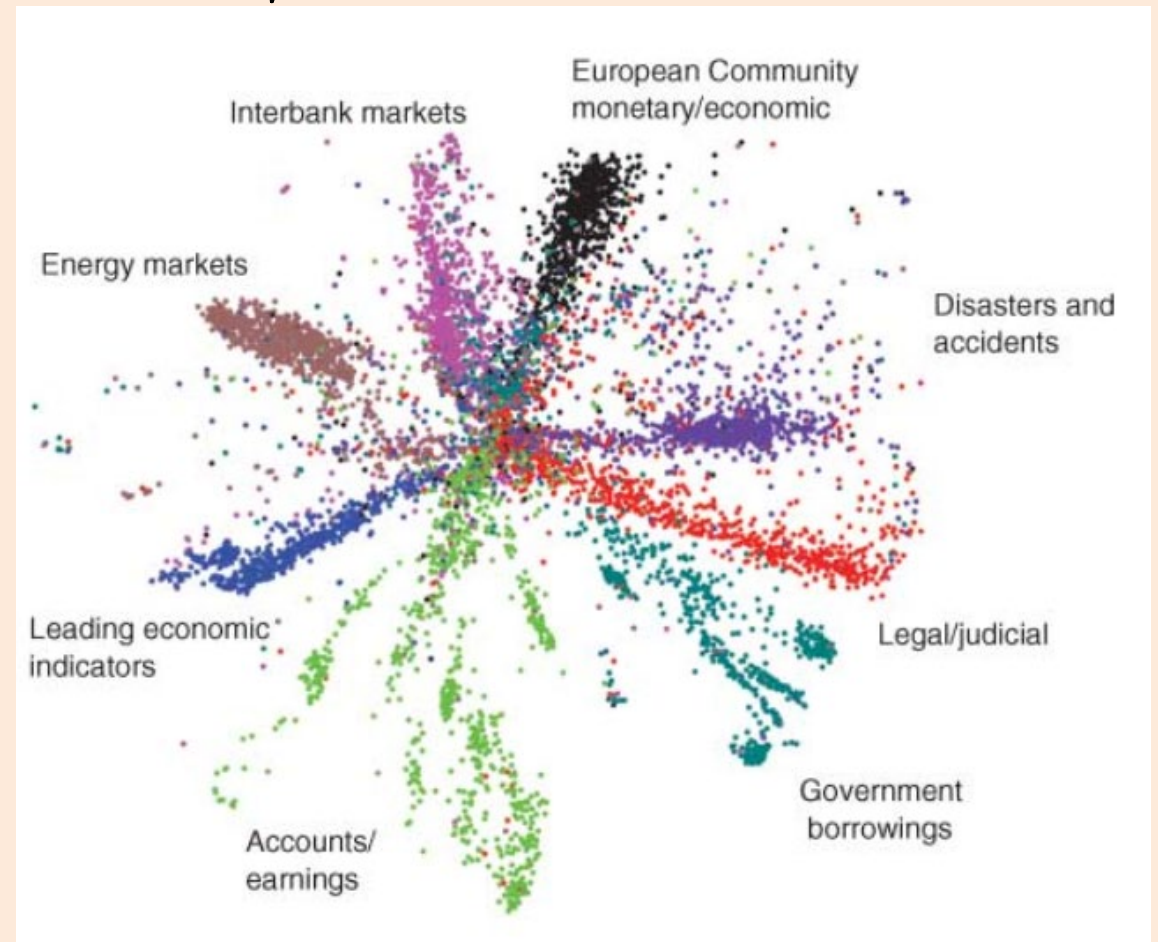
- Middle layer could be latent features in **non-linear latent-factor** model.
 - Can do outlier detection, data compression, visualization, etc.
- A non-linear generalization of PCA.
 - Equivalent to PCA if you don't have non-linearities.

Autoencoders

PCA

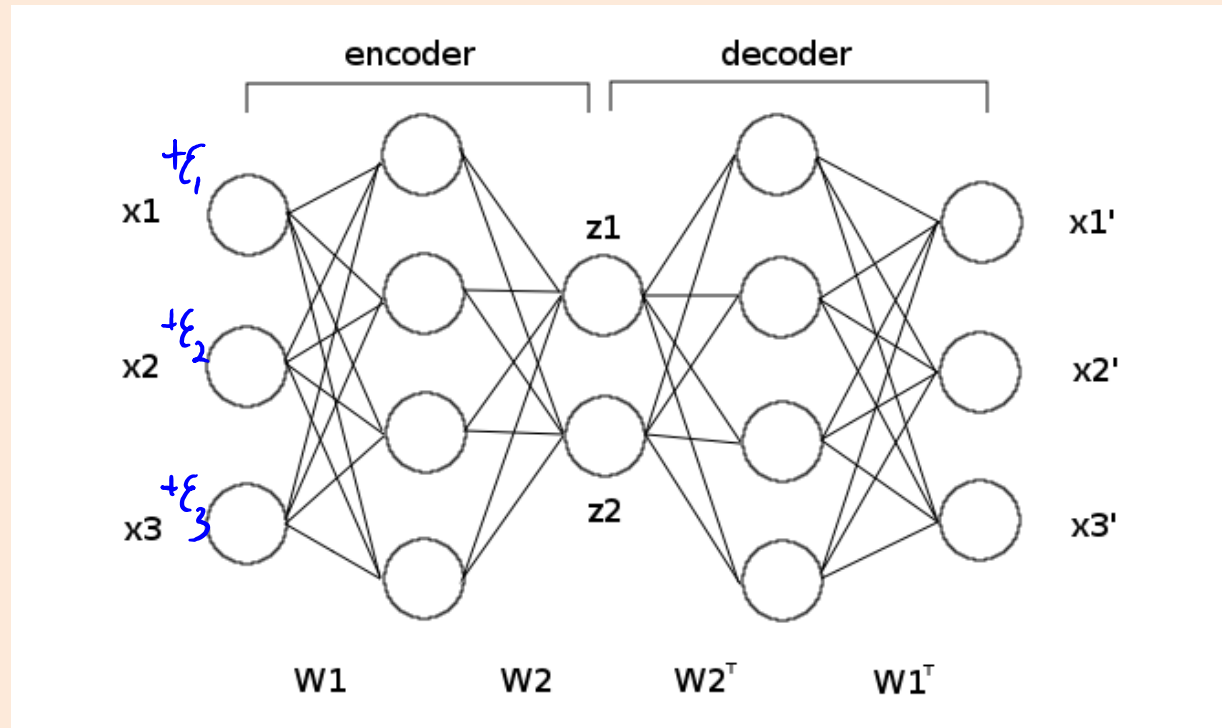


Autoencoder



Denoising Autoencoder

- **Denoising autoencoders** add noise to the input:



- Learns a model that can remove the noise.