

CPSC 340: Machine Learning and Data Mining

Principal Component Analysis
Summer 2021

Admin

- **Assignment 5 due Friday at 11:55pm**
 - Extended by 12+ hours
 - We will cover all relevant details today
- **This lecture will be 75 minutes long**
- **Office hours and tutorials had low attendance**
 - I promise they save you TONS of time in the long run
 - We have 10+ office hours per week
 - Going to a few office hours will probably get most of the work out of the way

In This Lecture

1. MAP Estimation
2. Wrapping Up Part 3
3. Principal Component Analysis

Last Time: MLE of Gaussian Likelihood

- Let's assume that $y_i = w^T x_i + \varepsilon_i$, with ε_i following **standard normal**:

$$p(\varepsilon_i) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\varepsilon_i^2}{2}\right)$$

also known
as "Gaussian"
distribution

- This leads to a **Gaussian likelihood** for example 'i' of the form:

$$p(y_i | x_i, w) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)$$

- Finding **MLE** (minimizing **NLL**) is **least squares**:

$$[1] \quad f(w) = -\sum_{i=1}^n \log(p(y_i | w, x_i))$$

$$[2] = -\sum_{i=1}^n \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right)$$

constant
in 'w'

$$[3] = -\sum_{i=1}^n \left[\log\left(\frac{1}{\sqrt{2\pi}}\right) + \log\left(\exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right) \right]$$

$$[4] = -\sum_{i=1}^n \left[(\text{constant}) - \frac{1}{2} (w^T x_i - y_i)^2 \right]$$

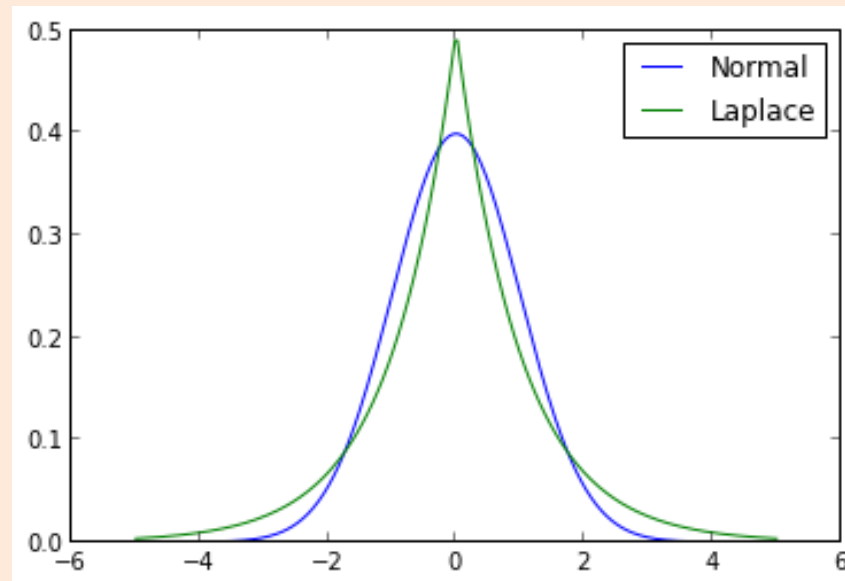
$$[5] = (\text{constant}) + \frac{1}{2} \sum_{i=1}^n (w^T x_i - y_i)^2$$

$$[6] = (\text{constant}) + \frac{1}{2} \|Xw - y\|^2$$

operations cancel

“Heavy” Tails vs. “Light” Tails

- We know that L1-norm is more robust than L2-norm.
 - What does this mean in terms of probabilities?



Here “tail” means
“mass of the
distribution away
from the mean.”

- Gaussian has “light tails”: assumes everything is close to mean.
- Laplace has “heavy tails”: assumes some data is far from mean.
- Student ‘t’ is even more heavy-tailed/robust, but NLL is non-convex.

Last Time: MLE of Sigmoid Likelihood

- For IID regression problems the conditional NLL can be written:

$$[1] \quad \underbrace{-\log(p(y|X, w))}_{NLL} = -\log\left(\underbrace{\prod_{i=1}^n p(y_i|x_i, w)}_{\text{IID assumption}}\right) = -\sum_{i=1}^n \log(p(y_i|x_i, w))$$

log turns product into sum

- Logistic regression assumes $\text{sigmoid}(w^T x_i)$ conditional likelihood:

$$[2] \quad p(y_i|x_i, w) = h(y_i w^T x_i) \quad \text{where} \quad h(z_i) = \frac{1}{1 + \exp(-z_i)}$$

- Plugging in the sigmoid likelihood, the **NLL is the logistic loss**:

$$[3] \quad NLL(w) = -\sum_{i=1}^n \log\left(\frac{1}{1 + \exp(-y_i w^T x_i)}\right) = \sum_{i=1}^n \log(1 + \exp(-y_i w^T x_i))$$

(since $\log(1) = 0$)

MLE Interpretation of Logistic Regression

- Instead of “smooth convex approximation of 0-1 loss”, we now have that **logistic regression is doing MLE in a probabilistic model.**
 - “Maximize +1-ness of +1 examples and -1-ness of -1 examples”
 - The **training and prediction would be the same** as before.
 - We still minimize the logistic loss in terms of ‘w’.
 - But MLE **justifies using sigmoid with learned w to get +1-ness:**

$$p(y_i | x_i, w) = \frac{1}{1 + \exp(-y_i w^T x_i)}$$

- Softmax function and softmax loss are also connected via NLL
 - See Piazza for derivations

Coming Up Next

MAXIMUM A POSTERIORI ESTIMATION

Maximum Likelihood Estimation and Overfitting

- In our abstract setting with data D the **MLE** is:

$$\hat{w} \in \operatorname{argmax}_w \{p(D|w)\}$$

- But conceptually MLE is a bit weird:

$$P(D|\cdot): \mathbb{R}^d \rightarrow [0, 1]$$

- “Find the ‘ w ’ that makes ‘ D ’ have the highest probability given ‘ w ’.”

- And MLE often leads to **overfitting**:

- Data could be very likely for some **very unlikely ‘ w ’**.

- For example, a complex model that overfits by memorizing the data.

- What we really want: $P(\cdot | D): \mathbb{R}^d \rightarrow [0, 1]$

- “Find the ‘ w ’ that has the highest probability given the data D .”

Maximum a Posteriori (MAP) Estimation

- Maximum a posteriori (MAP) estimate maximizes the reverse probability:

$$\hat{w} \in \operatorname{argmax}_w \{p(w|D)\} \quad P(\cdot|D) \neq P(D|\cdot)$$

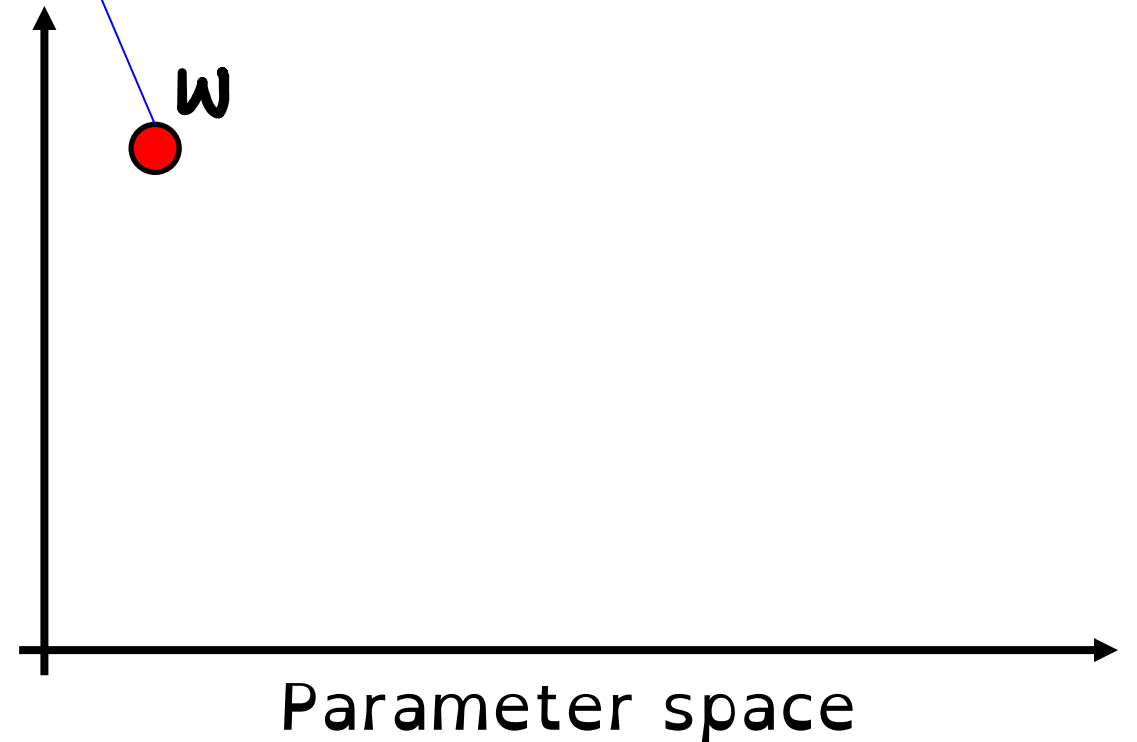
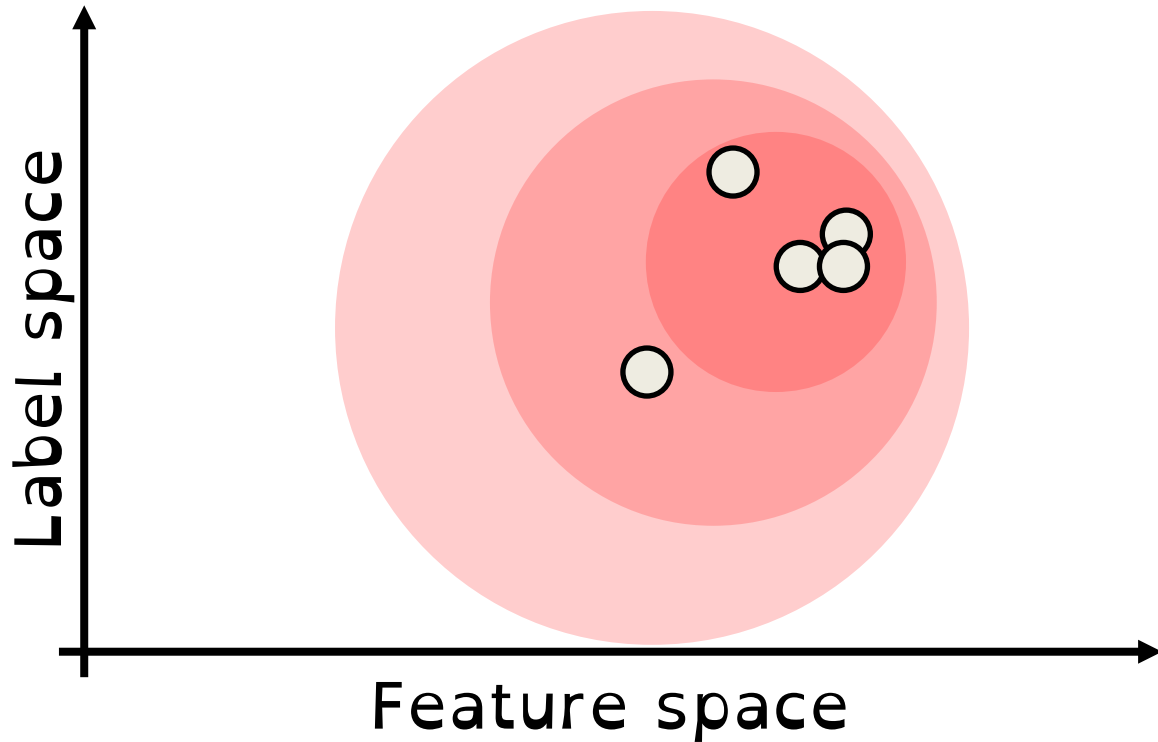
- This is **what we want**: the probability of 'w' given our data.
- MLE and MAP are connected by **Bayes rule**:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \propto \underbrace{p(D|w)}_{\text{likelihood}} \underbrace{p(w)}_{\text{prior}} \quad \begin{array}{l} \text{same } w \\ \hline \underline{p(D|\cdot)} \underline{p(\cdot)} \\ \neq P(D|\cdot) \end{array}$$

- So MAP maximizes the **likelihood** $p(D|w)$ times the **prior** $p(w)$:
 - Prior is our “belief” that 'w' is correct before seeing data.
 - Prior can reflect that **complex models are likely to overfit**.

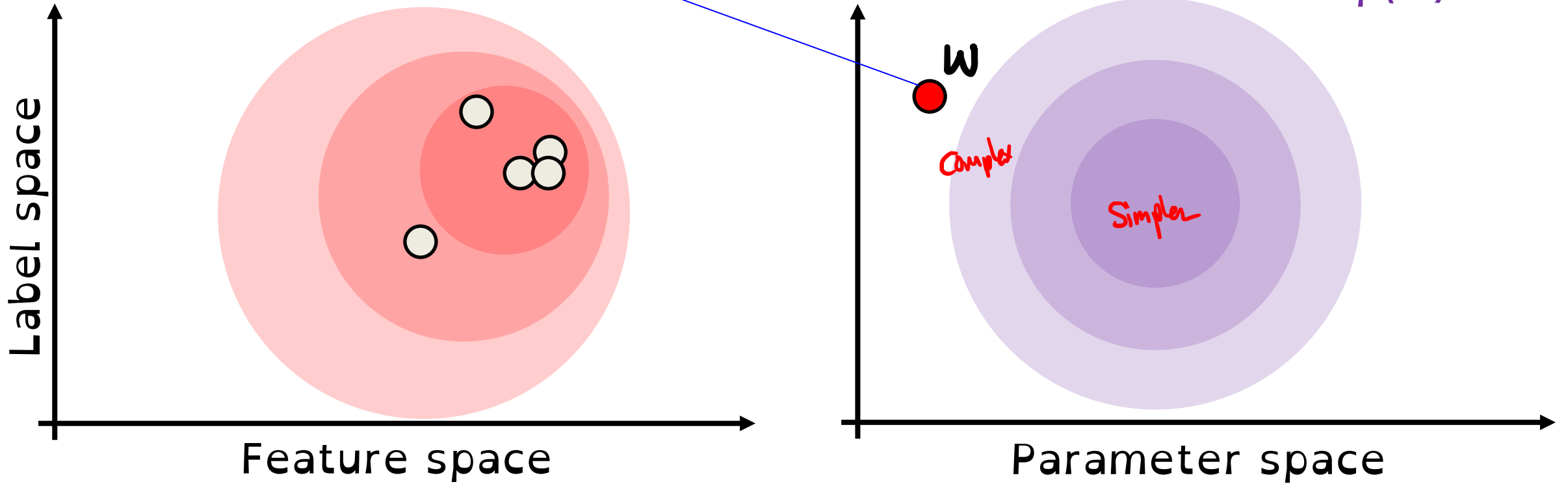
“Prior”

Q: What if this is overfitting?
How do we discourage w from going here?



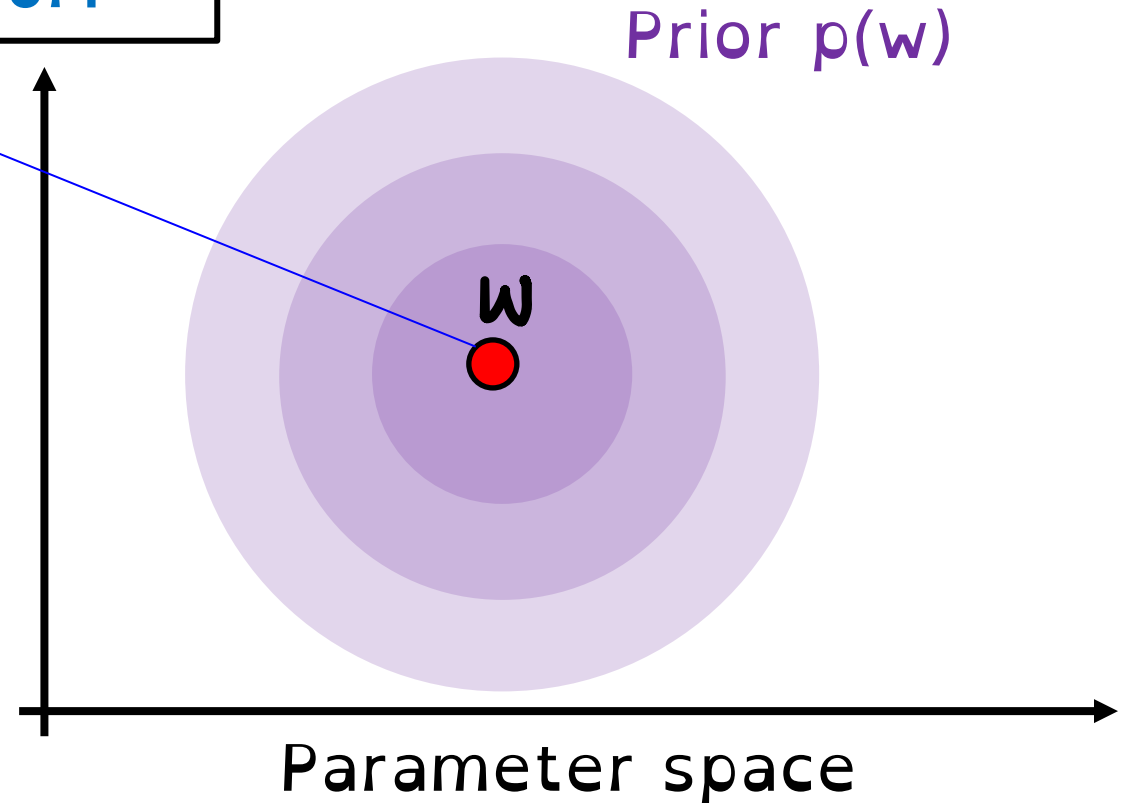
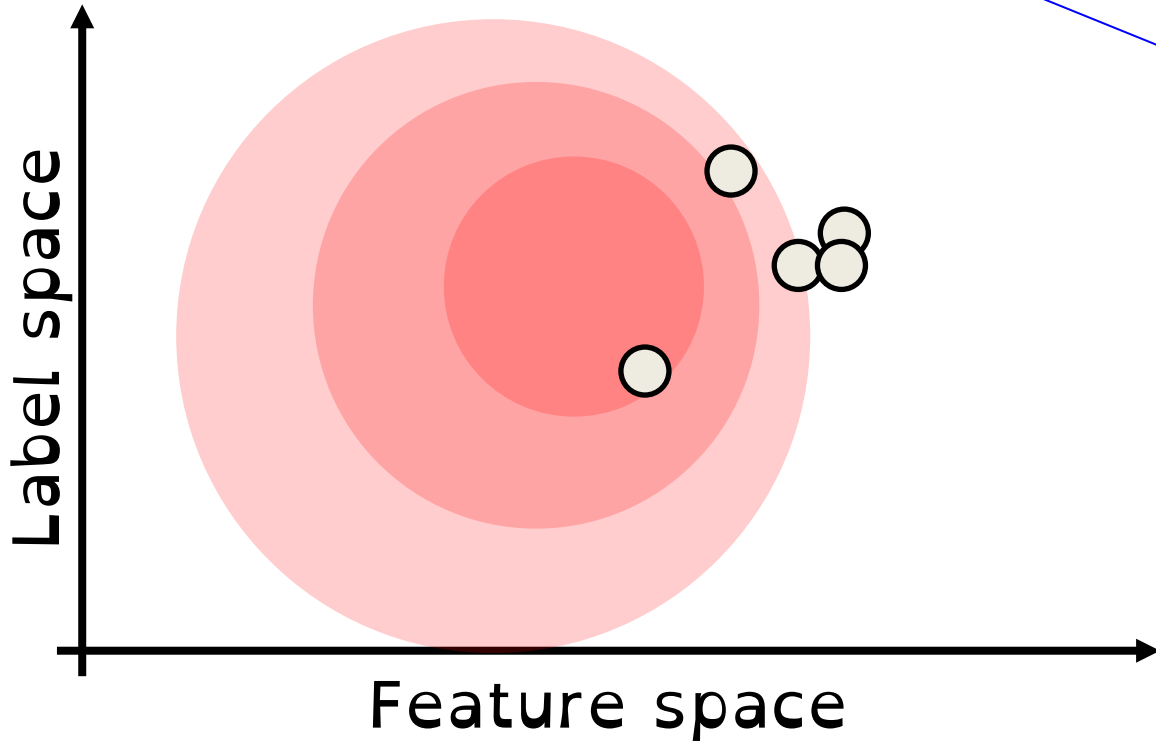
“Prior”

Q: Is this a good value according to the prior?



"Prior"

Q: Is this a good value according to the prior?



Q: Haven't we seen a similar concept before?

regularization

MAP Estimation and Regularization

- From Bayes rule, the MAP estimate with IID examples D_i is:

$$\hat{w} \in \operatorname{argmax}_w \{ p(w | D) \} \equiv \operatorname{argmax}_w \left\{ \prod_{i=1}^n [p(D_i | w)] p(w) \right\}$$

- By again taking the negative of the logarithm as before we get:

$$\hat{w} \in \operatorname{argmin}_w \left\{ \underbrace{-\sum_{i=1}^n [\log (p(D_i | w))]}_{\text{loss}} - \underbrace{\log (p(w))}_{\text{regularizer}} \right\}$$

- So we can view the negative log-prior as a regularizer:
 - Many regularizers are equivalent to negative log-priors.

L2-Regularization and MAP Estimation

- We obtain L2-regularization under an independent Gaussian assumption:

[1] Assume each w_j comes from a Gaussian with mean 0 and variance $1/\lambda$

- This implies that:

$$[2] \quad p(w) = \prod_{j=1}^d p(w_j) \stackrel{\text{independence}}{\propto} \prod_{j=1}^d \exp\left(-\frac{\lambda}{2} w_j^2\right) \stackrel{\text{Gaussian assumption}}{=} \exp\left(-\frac{\lambda}{2} \sum_{j=1}^d w_j^2\right)$$

$e^\alpha e^\beta = e^{\alpha+\beta}$

- So we have that:

$$[3] \quad -\log(p(w)) = -\log\left(\exp\left(-\frac{\lambda}{2} \|w\|^2\right)\right) + (\text{constant}) = \frac{\lambda}{2} \|w\|^2 + (\text{constant})$$

- With this prior, the MAP estimate with IID training examples would be

$$[4] \quad \hat{w} \in \operatorname{argmin}_w \left\{ -\log(p(y|X,w)) - \log(p(w)) \right\} \equiv \operatorname{argmin}_w \left\{ -\sum_{i=1}^n \left[\log(p(y_i|x_i,w)) \right] + \frac{\lambda}{2} \|w\|^2 \right\}$$

MAP Estimation and Regularization

- MAP estimation gives **link between probabilities and loss functions**.
 - Gaussian likelihood ($\sigma = 1$) + Gaussian prior gives L2-regularized least squares.

$$\text{If } p(y_i | x_i, w) \propto \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right) \quad p(w_j) \propto \exp\left(-\frac{\lambda}{2} w_j^2\right)$$

Then MAP estimation is equivalent to minimizing $f(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2$

- Laplace likelihood ($\sigma = 1$) + Gaussian prior give L2-regularized robust regression:

$$\text{If } p(y_i | x_i, w) \propto \exp(-|w^T x_i - y_i|) \quad p(w) \propto \exp\left(-\frac{\lambda}{2} w_j^2\right)$$

Then MAP estimation is equivalent to minimizing $f(w) = \|Xw - y\|_1 + \frac{\lambda}{2} \|w\|^2$

- As 'n' goes to infinity, effect of prior/regularizer goes to zero.
- Unlike with MLE, the **choice of σ changes the MAP solution** for these models.

Summary of MAP Estimation

- Many of our **loss functions and regularizers have probabilistic interpretations.**
 - Laplace likelihood leads to absolute error.
 - Laplace prior leads to L1-regularization.
- The choice of **likelihood** corresponds to the choice of **loss.**
 - Our assumptions about how the y_i -values can come from the x_i and 'w'.
- The choice of **prior** corresponds to the choice of **regularizer.**
 - Our assumptions about which 'w' values are plausible.

Regularizing Other Models

- We can view **priors in other models as regularizers.**
- Remember the problem with MLE for naïve Bayes:
 - The MLE of $p(\text{'lactase'} = 1 | \text{'spam'})$ is: $\text{count}(\text{spam}, \text{lactase}) / \text{count}(\text{spam})$.
 - But this **caused problems if $\text{count}(\text{spam}, \text{lactase}) = 0$.**
- Our solution was **Laplace smoothing**:
 - Add “+1” to our estimates: $(\text{count}(\text{spam}, \text{lactase}) + 1) / (\text{count}(\text{spam}) + 2)$.
 - This corresponds to a “Beta” prior. **Laplace smoothing is a regularizer.**

Why do we care about MLE and MAP?

- Unified way of thinking about many of our tricks
 - Probabilistic interpretation of logistic loss.
 - Laplace smoothing and L2-regularization are doing the same thing.
- Remember our two ways to reduce overfitting in complicated models:
 - Model averaging (ensemble methods).
 - Regularization (linear models).
- “Fully”-Bayesian methods (CPSC 440) combine both of these.
 - Average over all models, weighted by posterior (including regularizer).
 - Can use extremely-complicated models without overfitting.

Losses for Other Discrete Labels

- MLE/MAP gives loss for classification with basic labels:
 - Least squares and absolute loss for regression.
 - Logistic regression for binary labels {"spam", "not spam"}.
 - Softmax regression for multi-class {"spam", "not spam", "important"}.
- But MLE/MAP lead to losses with other discrete labels (bonus):
 - Ordinal: {1 star, 2 stars, 3 stars, 4 stars, 5 stars}.
 - Counts: 602 'likes'.
 - Survival rate: 60% of patients were still alive after 3 years.
 - Unbalanced classes: 99.9% of examples are classified as +1.
- Define likelihood of labels, and use NLL as the loss function.
- We can also use ratios of probabilities to define more losses (bonus):
 - Binary SVMs, multi-class SVMs, and "pairwise preferences" (ranking) models.

End of Part 3: Linear Models

End of Part 3: Key Concepts

- **Linear models** predict based on linear combination(s) of features:

$$W^T x_i = w_1 x_{i1} + w_2 x_{i2} + \dots + w_d x_{id}$$

- We model non-linear effects using a **change of basis**:
 - Replace **d-dimensional** x_i with **k-dimensional** z_i and use $v^T z_i$.
 - Examples include **polynomial basis** and (non-parametric) **RBFs**.
- **Regression** is supervised learning with continuous labels.
 - Logical error measure for regression is **squared error**:
 - Can be solved as a **system of linear equations**.

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

End of Part 3: Key Concepts

- **Gradient descent** finds local minimum of smooth objectives.
 - Converges to a global optimum for **convex functions**.
 - Can use smooth approximations (**Huber**, **log-sum-exp**)
- **Stochastic gradient** methods allow huge/infinite 'n'.
 - Though very **sensitive to the step-size**.
- **Kernels** let us use similarity between examples, instead of features.
 - Lets us use some **exponential- or infinite-dimensional features**.
- **Feature selection** is a messy topic.
 - Classic method is **forward selection** based on **L0-norm**.
 - **L1-regularization** simultaneously regularizes and selects features.

End of Part 3: Key Concepts

- Reduce over-fitting by using **regularization**:

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

- Squared error is **not always right** measure:
 - **Absolute error** is less sensitive to outliers.
 - **Logistic loss** and **hinge loss** are better for binary y_i .
 - **Softmax loss** is better for multi-class y_i .
- **MLE/MAP** perspective:
 - View **loss as log-likelihood** and **regularizer as log-prior**.
 - Allows us to define **losses based on probabilities**.

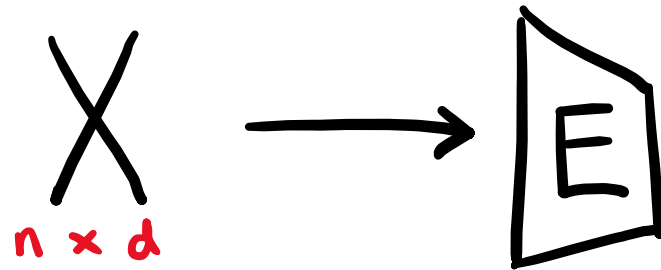
The Story So Far...

- Part 1: Supervised Learning.
 - Methods based on counting and distances.
- Part 2: Unsupervised Learning.
 - Methods based on counting and distances.
- Part 3: Supervised Learning (just finished).
 - Methods based on linear models and gradient descent.
- Part 4: Unsupervised Learning (starting now).
 - Methods based on linear models and gradient descent.

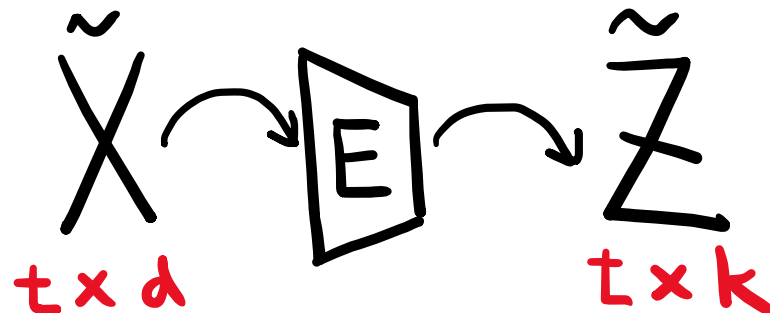
Part 4: Latent Factor Models

The “Encoder Learning Problem”

- The Encoder Learning Problem
 - Input: Feature matrix ‘X’
 - Output: An “encoder” model that can transform examples



- The Encoding Problem
 - Input: A test example \tilde{x}_i and a learned encoder model
 - Output: The “encoded” example \tilde{z}_i



Motivation: Human vs. Machine Perception

- **Huge difference** between what we see and what computer sees:

What we see:

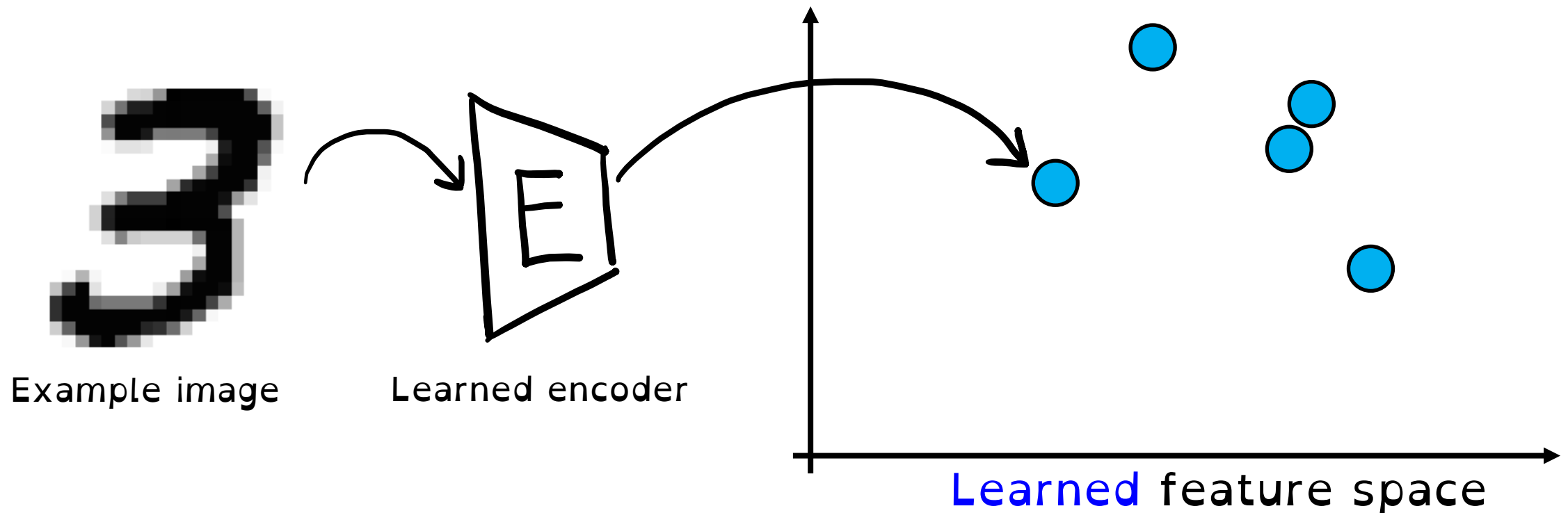


What the computer "sees":



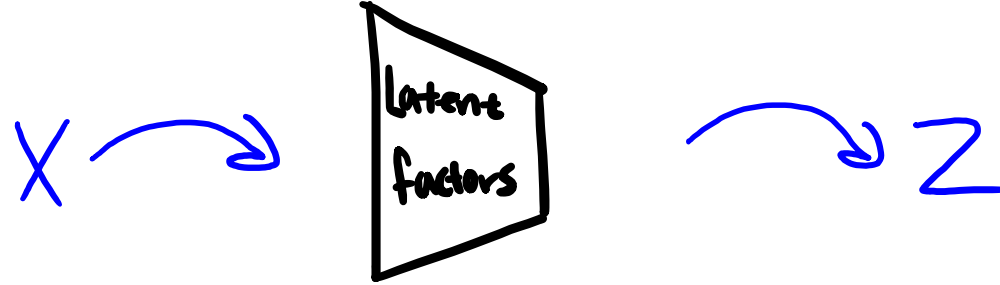
- But maybe **images shouldn't be written as combinations of pixels.**
 - Can we learn a better representation?
 - In other words, can we **learn good features**?

Encoding Images



- This is like feature engineering!
 - Find a better feature space for analyzing the data
- But now, we're **learning the feature space!**

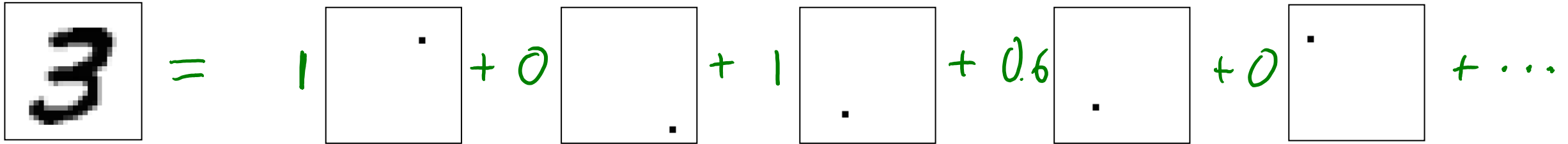
Part 4: Latent-Factor Models



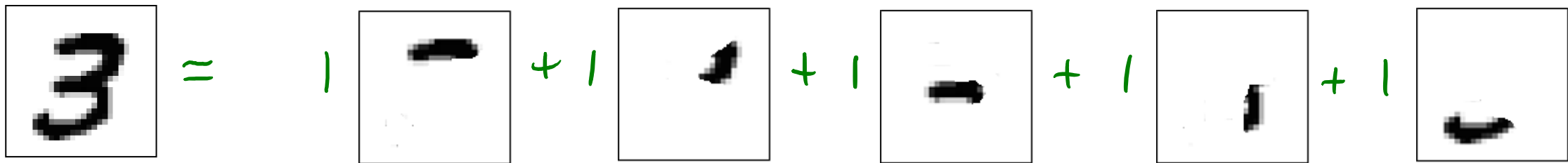
- Part 4 is about **learning the encoder from data**.
- Our encoders are linear models that use “latent factors”

Motivation: Pixels vs. Parts

- Can view 28x28 image as **weighted sum** of “single pixel on” images:


$$3 = 1 \cdot \text{[single pixel on]} + 0 \cdot \text{[single pixel on]} + 1 \cdot \text{[single pixel on]} + 0.6 \cdot \text{[single pixel on]} + 0 \cdot \text{[single pixel on]} + \dots$$

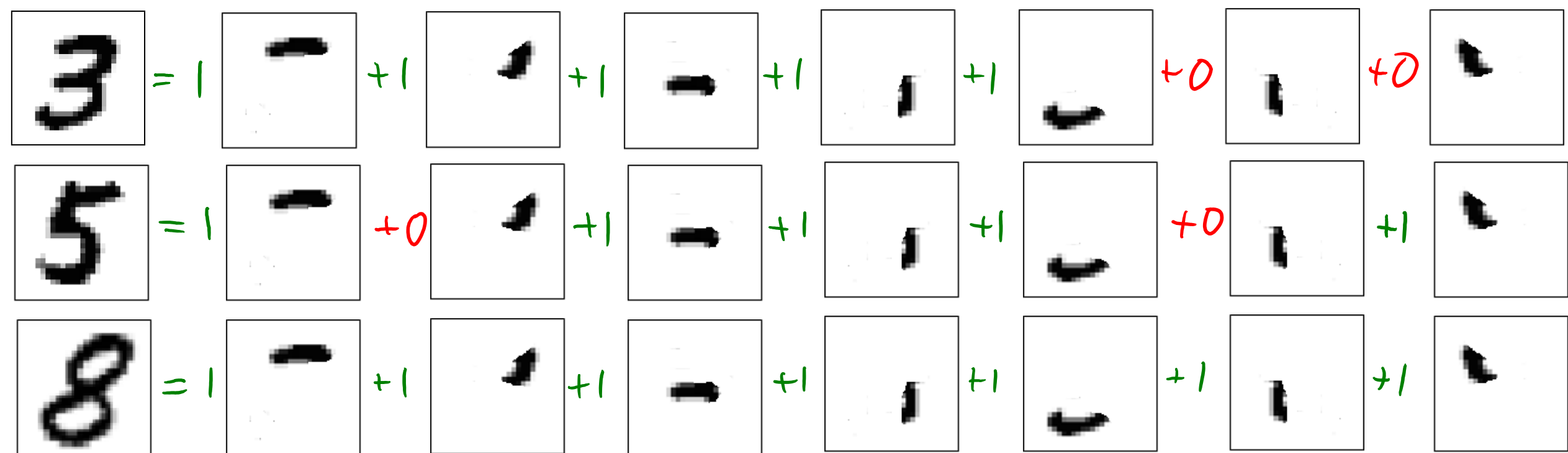
- We have one image/feature for each pixel.
- The **weights** specify “how much of this pixel is in the image”.
 - A weight of zero means that pixel is white, a weight of 1 means it’s black.
- This is **non-intuitive**, isn’t a “3” made of **small number of “parts”**?


$$3 = 1 \cdot \text{[part]} + 1 \cdot \text{[part]} + 1 \cdot \text{[part]} + 1 \cdot \text{[part]} + 1 \cdot \text{[part]}$$

- Now the weights are “**how much of this part is in the image**”.

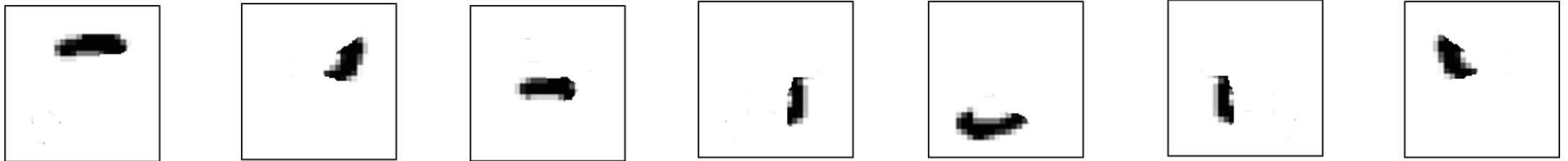
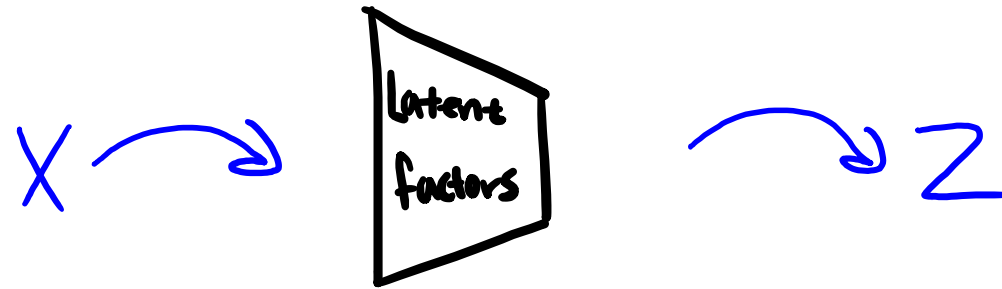
Motivation: Pixels vs. Parts

- We could represent other digits as different combinations of “parts”:



- Consider replacing images x_i by the weights z_i of the different parts:
 - The 784-dimensional x_i for the “5” image is replaced by 7 numbers: $z_i = [1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1]$.
 - Features like this could make learning much easier.

Why Latent Factors?



- These “parts” are called “**factors**”
- **Factors** are learned from data

Latent Factor Models are Useful

- **Supervised learning:**
 - we could use learned features as input features.
- **Outlier detection:**
 - example might be an outlier if isn't a combination of usual parts.
- **Dimension reduction:**
 - compress data into limited number of “part weights”.
- **Visualization:**
 - if we have only 2 “part weights”, we can view data as a scatterplot.
- **Interpretation:**
 - we can try and figure out what the “parts” represent.

Coming Up Next

PRINCIPAL COMPONENT ANALYSIS

INTRO

Principal Component Analysis (PCA) Applications

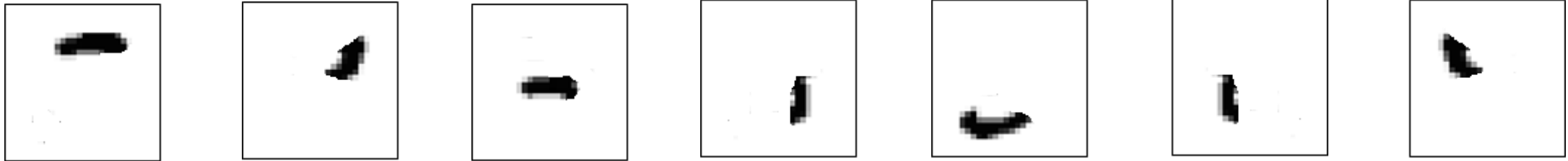
- Principal component analysis (PCA) has been invented many times:

PCA was invented in 1901 by [Karl Pearson](#),^[1] as an analogue of the [principal axis theorem](#) in mechanics; it was later independently developed (and named) by [Harold Hotelling](#) in the 1930s.^[2] Depending on the field of application, it is also named the discrete [Kosambi–Karhunen–Loève](#) transform (KLT) in signal processing, the [Hotelling](#) transform in multivariate quality control, proper orthogonal decomposition (POD) in mechanical engineering, [singular value decomposition](#) (SVD) of \mathbf{X} (Golub and Van Loan, 1983), [eigenvalue decomposition](#) (EVD) of $\mathbf{X}^T\mathbf{X}$ in linear algebra, [factor analysis](#) (for a discussion of the differences between PCA and factor analysis see Ch. 7 of ^[3]), [Eckart–Young theorem](#) (Harman, 1960), or [Schmidt–Mirsky theorem](#) in psychometrics, [empirical orthogonal functions](#) (EOF) in meteorological science, [empirical eigenfunction decomposition](#) (Sirovich, 1987), [empirical component analysis](#) (Lorenz, 1956), [quasiharmonic modes](#) (Brooks et al., 1988), [spectral decomposition](#) in noise and vibration, and [empirical modal analysis](#) in structural dynamics.

standard deviation of 3 in roughly the (0.878, 0.478) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the [covariance matrix](#) scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

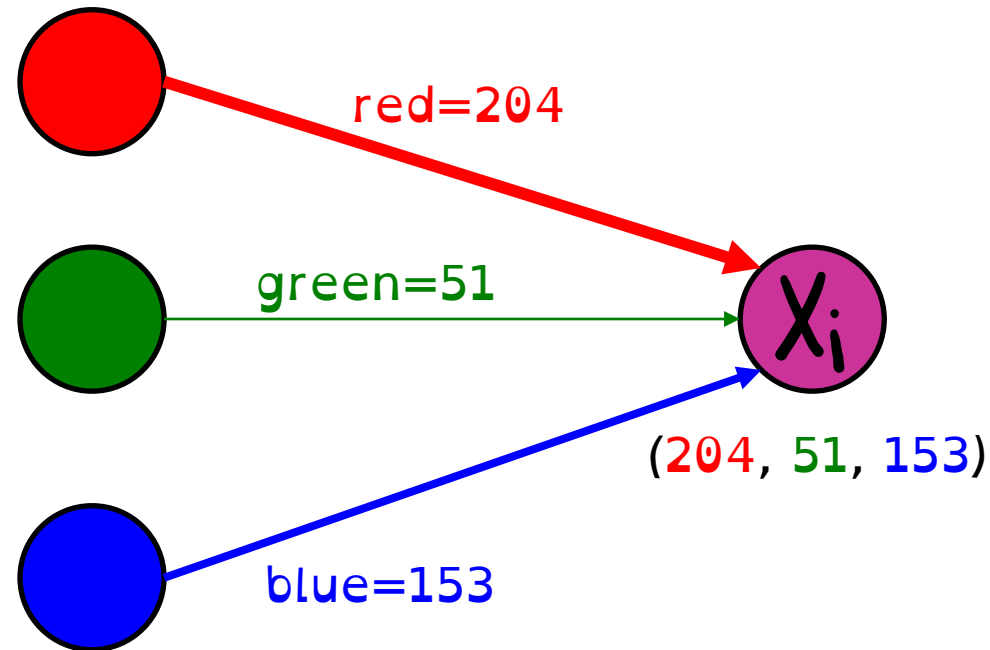
What is PCA?

- “PCA” := Principal Component Analysis
- An instance of latent factors model
 - Learn the factors from data → principal components (PCs)
 - Use factors to transform data



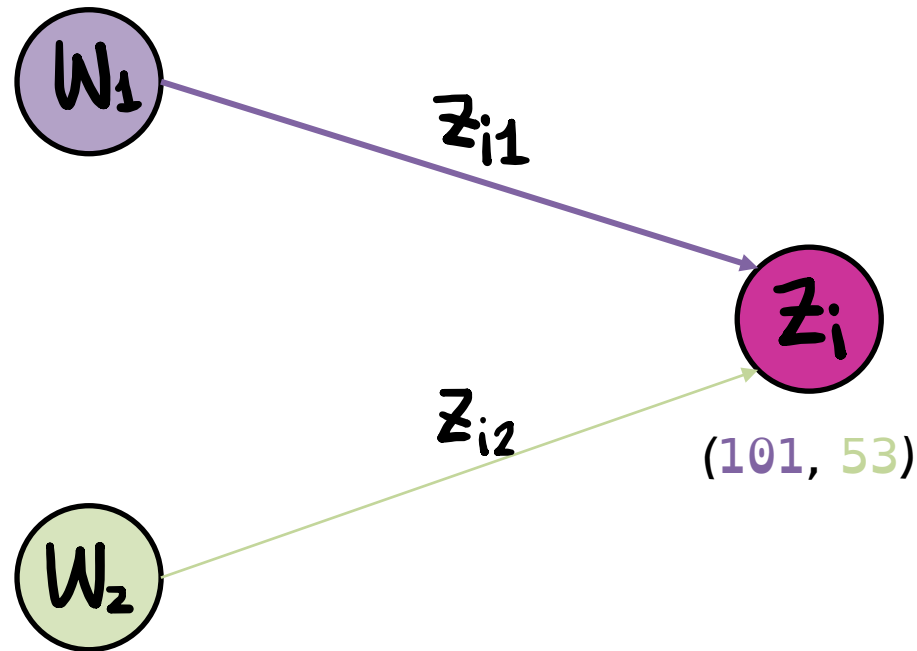
- **Assumption:** an example in feature space is a linear combination of factors

Visualizing Factors



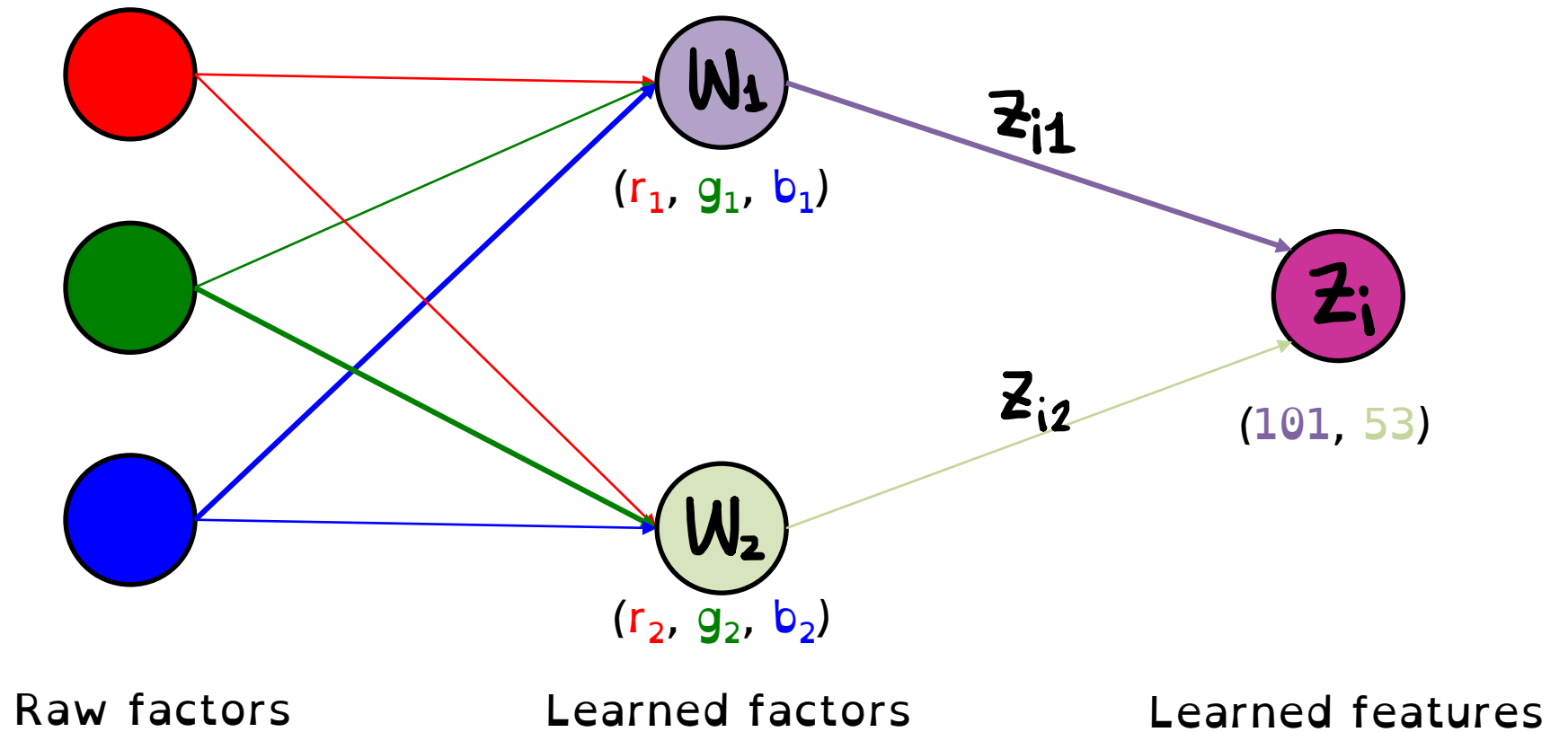
- (204, 51, 153) is the RGB value of this colour
- Given these RGB factors, the features are “factor-ness” scores

Visualizing Factors

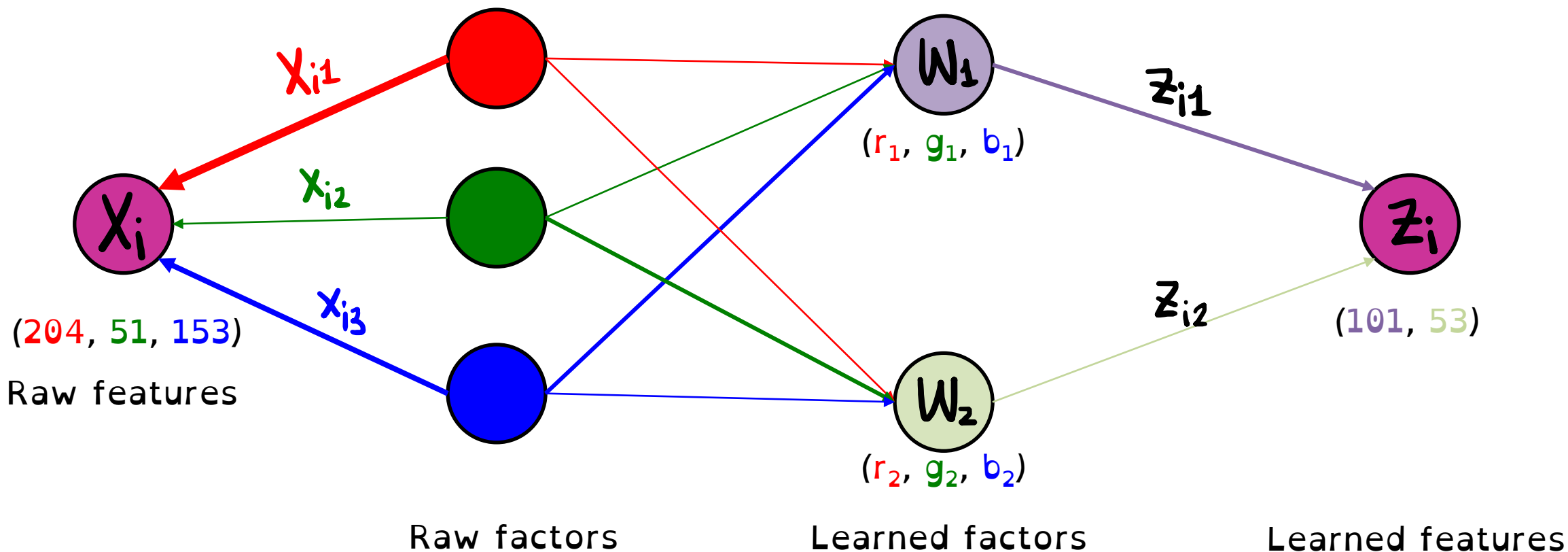


- $(101, 53)$ is NOT the RGB value of **this colour**
- But it represents the influence of these colours!
- **Idea:** use $(101, 53)$ as features
 - Given factors w_1, w_2 , the edge weights z_{i1}, z_{i2} are “factor-ness” scores

Visualizing Factors



Visualizing Factors



Facts:

① $X_i \neq Z_i$

②

$$X_{i1} \approx Z_{i1} r_1 + Z_{i2} r_2$$

$$X_{i2} \approx Z_{i1} g_1 + Z_{i2} g_2$$

$$X_{i3} \approx Z_{i1} b_1 + Z_{i2} b_2$$

$$\begin{bmatrix} X_{i1} \\ X_{i2} \\ X_{i3} \end{bmatrix} \approx \begin{bmatrix} r_1 & r_2 \\ g_1 & g_2 \\ b_1 & b_2 \end{bmatrix} \begin{bmatrix} Z_{i1} \\ Z_{i2} \end{bmatrix}$$

Visualizing Factors

W_1

(r_1, g_1, b_1)

W_2

(r_2, g_2, b_2)

Learned factors

$$\left\{ \begin{array}{ccc} r_1 & g_1 & b_1 \\ r_2 & g_2 & b_2 \end{array} \right\} = \begin{bmatrix} \text{---} w_1 \text{---} \\ \text{---} w_2 \text{---} \end{bmatrix} = W$$

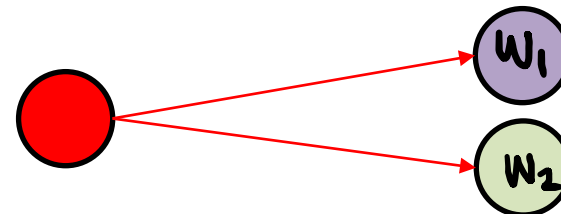
Q: What's the shape of this?

PCA Notation (MEMORIZE)

- PCA takes in a matrix 'X' and an input 'k', and outputs two matrices:

$$Z = \begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix} \left. \vphantom{\begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix}} \right\} n$$
$$W = \begin{bmatrix} - & w_1^T & - \\ - & w_2^T & - \\ & \vdots & \\ - & w_k^T & - \end{bmatrix} \left. \vphantom{\begin{bmatrix} - & w_1^T & - \\ - & w_2^T & - \\ & \vdots & \\ - & w_k^T & - \end{bmatrix}} \right\} k = \begin{bmatrix} | & | & \dots & | \\ w^1 & w^2 & \dots & w^d \\ | & | & & | \end{bmatrix} \left. \vphantom{\begin{bmatrix} | & | & \dots & | \\ w^1 & w^2 & \dots & w^d \\ | & | & & | \end{bmatrix}} \right\} k$$

- Row c of $W \rightarrow w_c$.
 - d -by- 1 vector, called a “factor” or “principal component”.
- Row i of $Z \rightarrow z_i$.
 - k -by- 1 vector, called “factor loadings” (or “scores”).
 - We have k factors, so this corresponds to k different “factor-ness” values
- Column j of $W \rightarrow w^j$.
 - k -by- 1 vector, index j of all the k “factors”
 - e.g. redness of w_1 and w_2



model = PCA(K=3)

model.fit(X)

PCA Notation (MEMORIZE)

- PCA takes in a matrix 'X' and an input 'k', and outputs two matrices:

$$Z = \begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix} \left. \vphantom{\begin{bmatrix} - & z_1^T & - \\ - & z_2^T & - \\ & \vdots & \\ - & z_n^T & - \end{bmatrix}} \right\} n$$

$$W = \begin{bmatrix} - & w_1^T & - \\ - & w_2^T & - \\ & \vdots & \\ - & w_k^T & - \end{bmatrix} \left. \vphantom{\begin{bmatrix} - & w_1^T & - \\ - & w_2^T & - \\ & \vdots & \\ - & w_k^T & - \end{bmatrix}} \right\} k$$

$$= \begin{bmatrix} | & | & \dots & | \\ w^1 & w^2 & \dots & w^d \\ | & | & \dots & | \end{bmatrix} \left. \vphantom{\begin{bmatrix} | & | & \dots & | \\ w^1 & w^2 & \dots & w^d \\ | & | & \dots & | \end{bmatrix}} \right\} k$$

- With this notation, we can write our approximation of one x_{ij} as:

$$\hat{x}_{ij} = z_{i1}w_{1j} + z_{i2}w_{2j} + \dots + z_{ik}w_{kj} = \sum_{c=1}^k z_{ic}w_{cj} = (w^j)^T z_i = \langle w^j, z_i \rangle$$

(NEW NOTATION)

- We can write approximation of the vector x_i as:

$$\hat{x}_i = \begin{bmatrix} \langle w^1, z_i \rangle \\ \langle w^2, z_i \rangle \\ \vdots \\ \langle w^d, z_i \rangle \end{bmatrix} = W^T z_i$$

$d \times 1$ $d \times k$ $k \times 1$

Different views (MEMORIZE)

- PCA approximates each x_{ij} by the inner product $\langle w^j, z_i \rangle$.
- PCA approximates each x_i by the matrix-vector product $W^T z_i$.
- PCA approximates matrix 'X' by the matrix-matrix product ZW .

$$\overset{n \times d}{X} \approx \overset{n \times k}{Z} \overset{k \times d}{W}$$

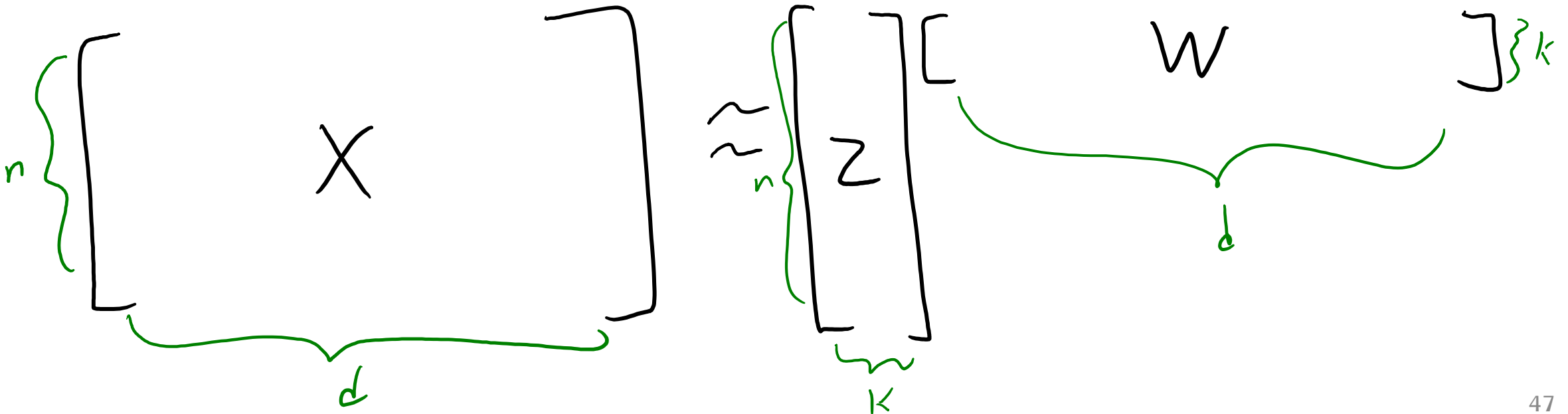
- PCA is also called a “**matrix factorization**” model.
 - Both ‘Z’ and ‘W’ are variables.
- This can be viewed as a “change of basis” from x_i to z_i values.
 - The “basis vectors” are the rows of W , the w_c .
 - z_i is example i in the new feature space

Coming Up Next

APPLICATIONS OF PCA

PCA Applications

- Applications of PCA:
 - **Dimensionality reduction**: replace 'X' with lower-dimensional 'Z'.
 - If $k \ll d$, then compresses data.
 - Often better approximation than vector quantization.



PCA Applications

- Applications of PCA:

- **Dimensionality reduction**: replace 'X' with lower-dimensional 'Z'.

- If $k \ll d$, then compresses data.
- Often better approximation than vector quantization.

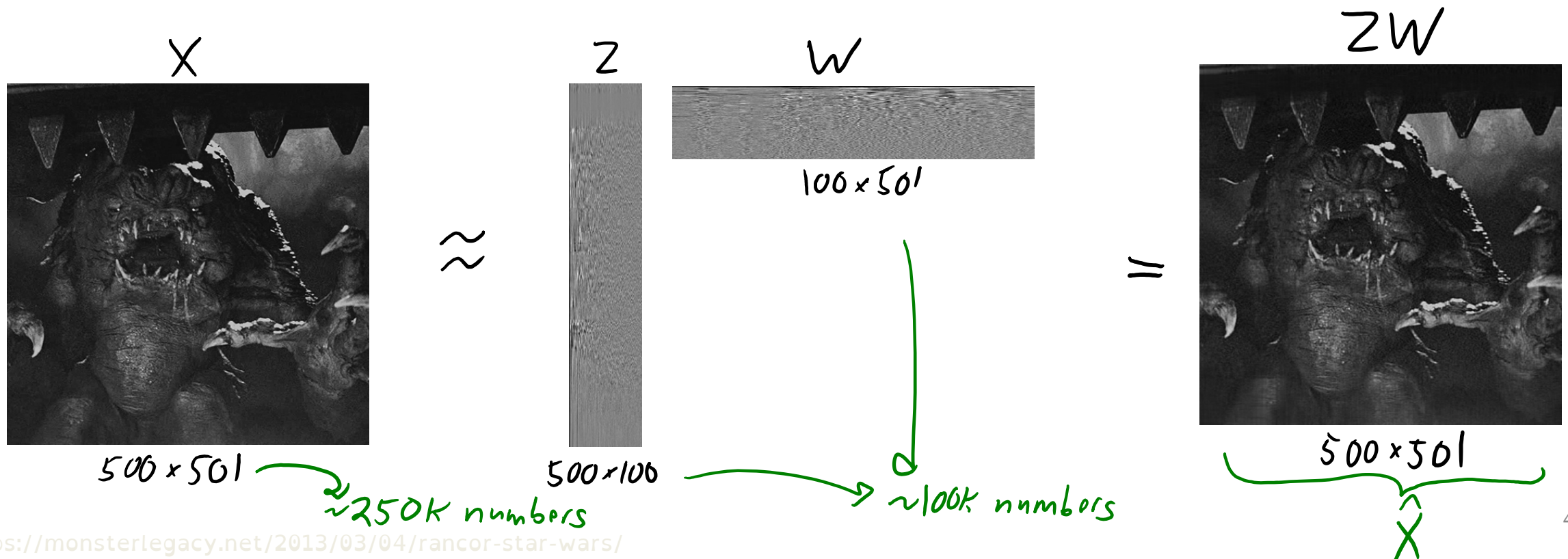
$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \\ 3 & 6 & 9 & 12 & 15 & 18 & 21 & 24 \\ 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 \\ 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 \\ 6 & 12 & 18 & 24 & 30 & 36 & 42 & 48 \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 & 56 \\ 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

Compresses 64 elements of 'X' down to 16 elements of 'Z' and 'W'

(can predict all x_i values from one z_i value)

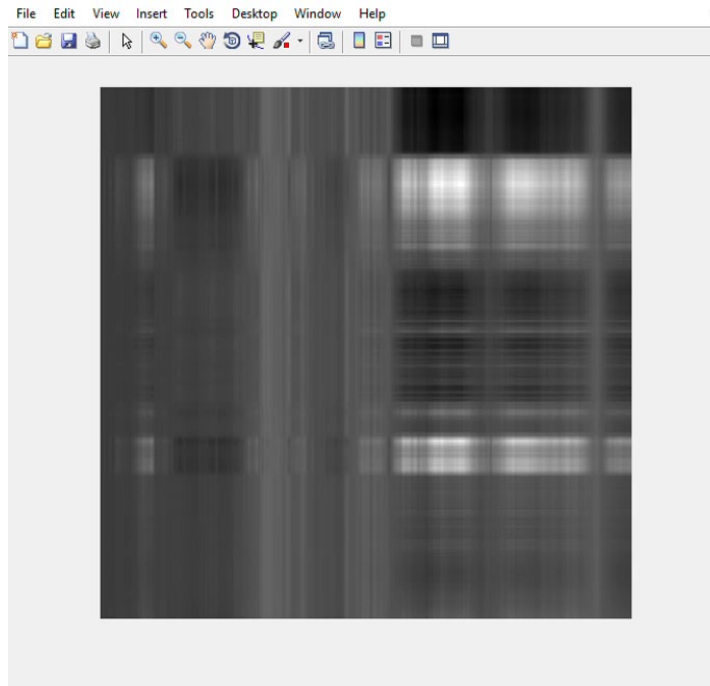
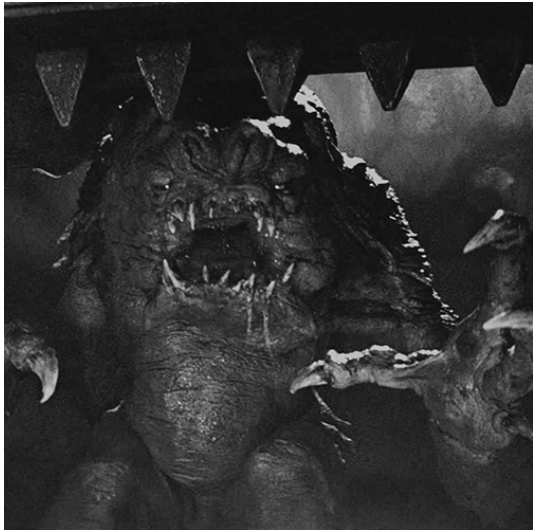
PCA Applications

- Applications of PCA:
 - **Dimensionality reduction**: replace 'X' with lower-dimensional 'Z'.
 - If $k \ll d$, then compresses data.
 - Often better approximation than vector quantization.



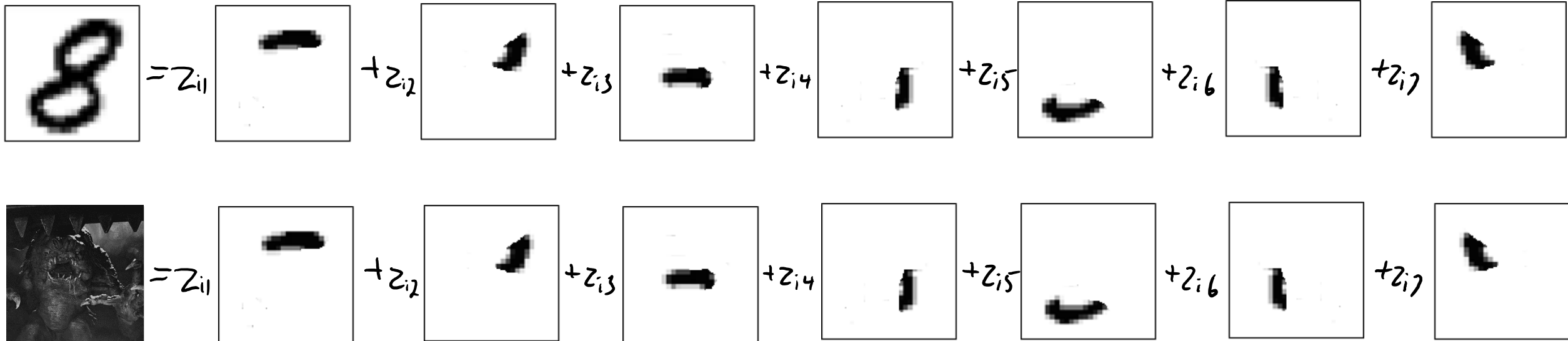
PCA Applications

- Applications of PCA:
 - **Dimensionality reduction**: replace 'X' with lower-dimensional 'Z'.
 - If $k \ll d$, then compresses data.
 - Often better approximation than vector quantization.



PCA Applications

- Applications of PCA:
 - **Outlier detection**: if PCA gives poor approximation of x_i , could be 'outlier'.
 - Though due to squared error **PCA is sensitive to outliers**.



PCA Applications

- Applications of PCA:
 - Partial least squares: uses PCA scores as basis for linear model.

Compute approximation $X \approx ZW$

Now use Z as features in a linear model:

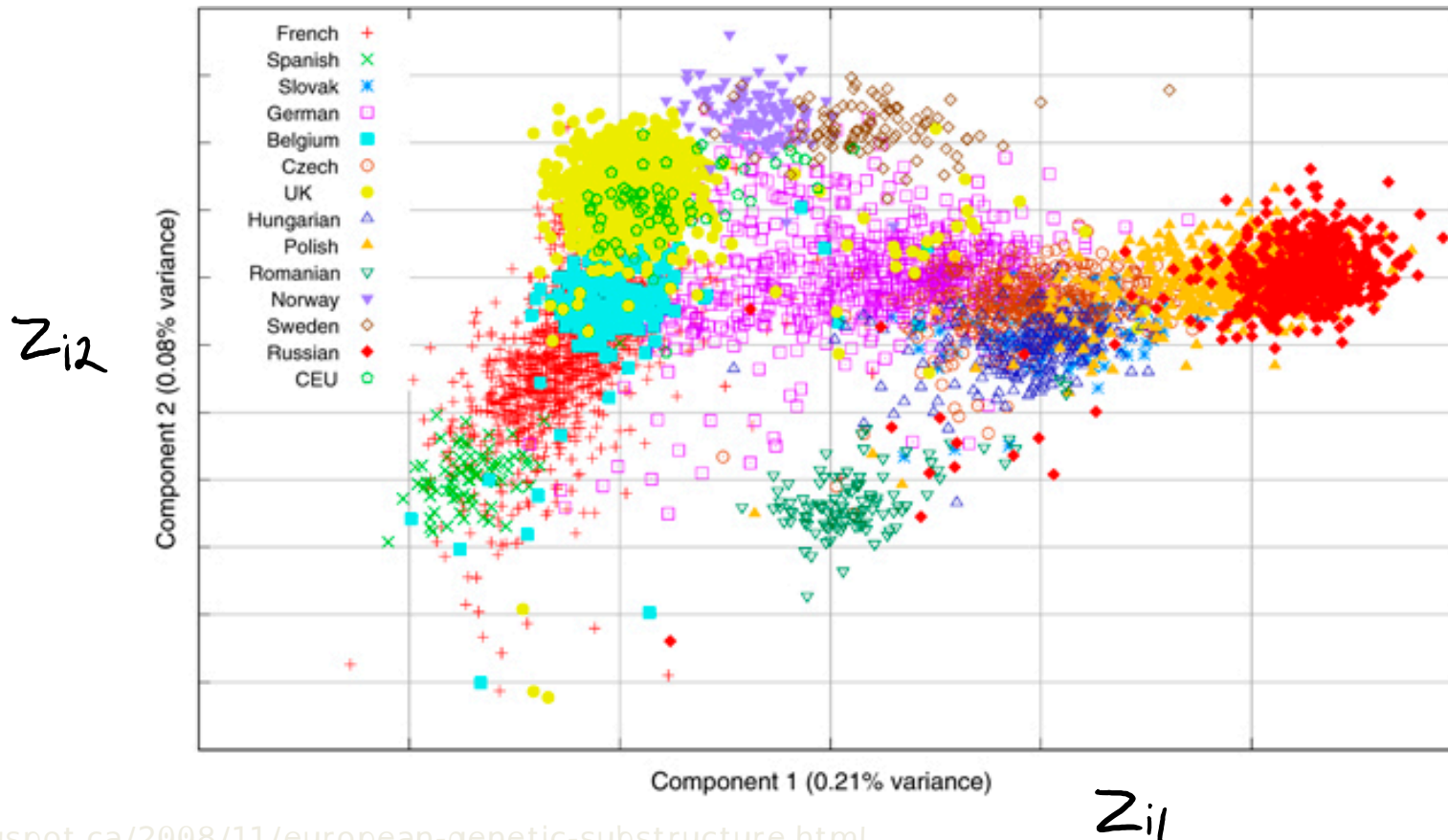
$$y_i = v^T z_i$$

linear regression
weights ' v ' trained
under this change
of basis.

lower-dimensional than original features so less overfitting

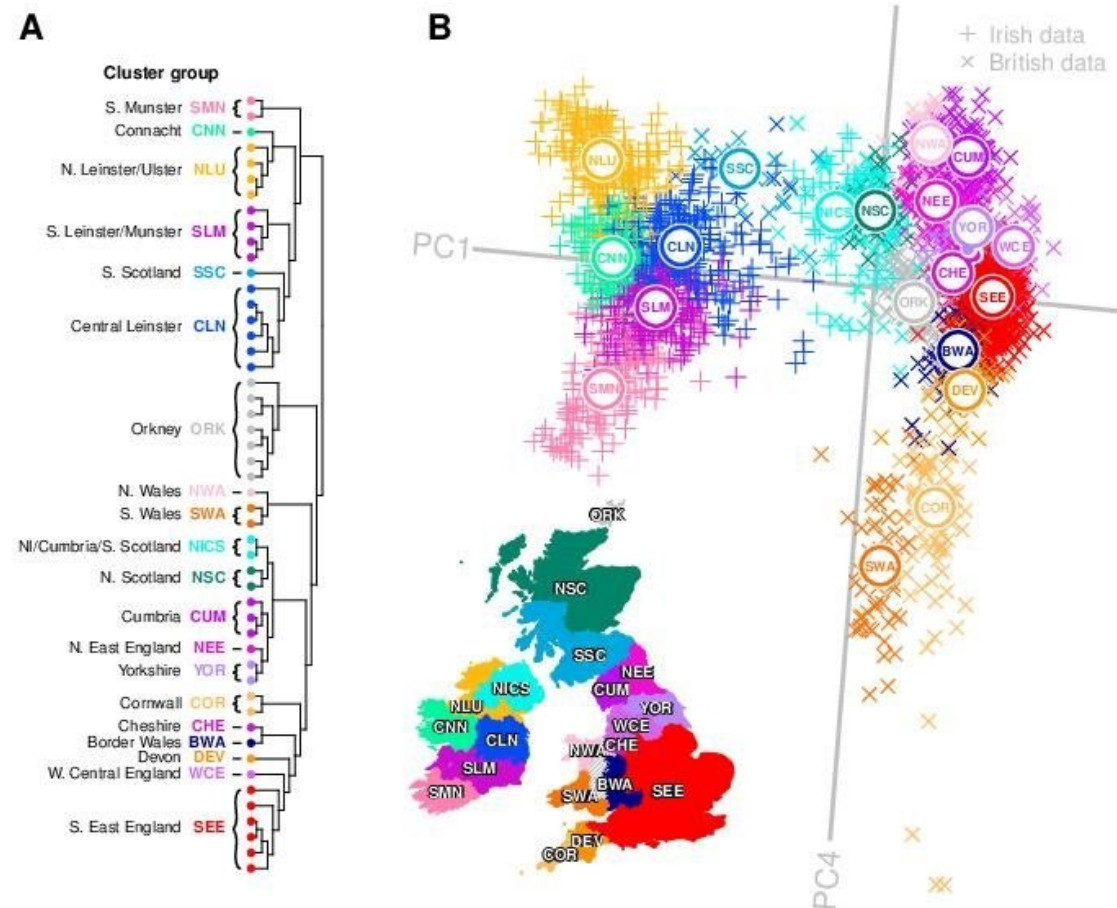
PCA Applications

- Applications of PCA:
 - **Data visualization**: plot z_i with $k = 2$ to visualize high-dimensional objects.



PCA Applications

- Applications of PCA:
 - **Data visualization:** plot z_i with $k = 2$ to visualize high-dimensional objects.
 - Can augment other visualizations:



PCA Applications

- Applications of PCA:
 - **Data interpretation:** we can try to **assign meaning to latent factors w_c** .
 - Hidden “factors” that influence all the variables.

Trait	Description
O penness	Being curious, original, intellectual, creative, and open to new ideas.
C onscientiousness	Being organized, systematic, punctual, achievement-oriented, and dependable.
E xtraversion	Being outgoing, talkative, sociable, and enjoying social situations.
A greeableness	Being affable, tolerant, sensitive, trusting, kind, and warm.
N euroticism	Being anxious, irritable, temperamental, and moody.

<https://new.education.com/resources/big-5-personality-traits> "Most Personality Quizzes Are Junk Science. I Found One That Isn't." 55



Coming Up Next

GEOMETRIC INTUITION FOR PCA

Overhead Map and Latent-Factor Models

- Consider these crazy goats trying to get some salt:
 - Ignoring height gives poor approximation of goat location.

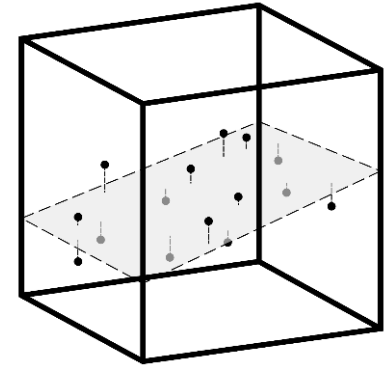


Top-down view of a dam

- But the “goat space” is basically a **two-dimensional plane**.
 - Better $k=2$ approximation: **define ‘W’ so that combinations give the plane.**

Overhead Map and Latent-Factor Models

- Consider these crazy goats trying to get some salt:
 - Ignoring height gives poor approximation of goat location.



- But the “goat space” is basically a **two-dimensional plane**.
 - Better $k=2$ approximation: **define ‘W’ so that combinations give the plane.**

Overhead Map and Latent-Factor Models

- A goat i 's location in the world can be described by 3 coordinates:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix} \begin{array}{l} \leftarrow \text{"x" coordinate} \\ \leftarrow \text{"y" coordinate} \\ \leftarrow \text{"z" coordinate} \end{array}$$

- The overhead view **approximates these 3 coordinates with only 2:**

$$z_i = \begin{bmatrix} z_{i1} \\ z_{i2} \end{bmatrix} \begin{array}{l} \leftarrow \text{"x" coordinate} \\ \leftarrow \text{"y" coordinate} \end{array}$$

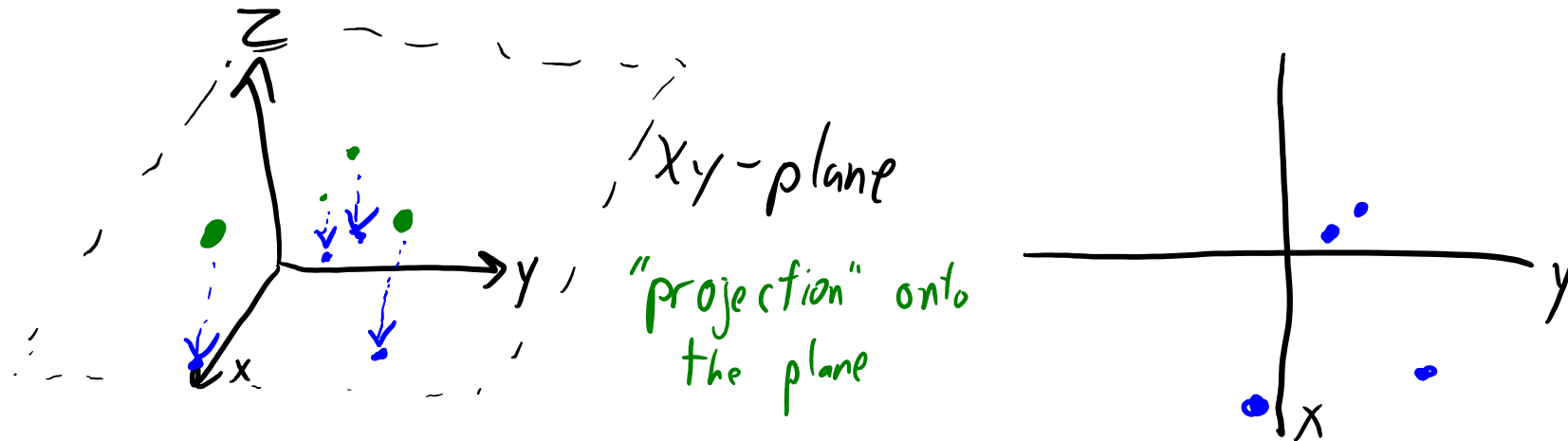
- Our $k=2$ **latent factors** are the following:

$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- So our approximation of x_i is: $\hat{x}_i = z_{i1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + z_{i2} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

Overhead Map and Latent-Factor Models

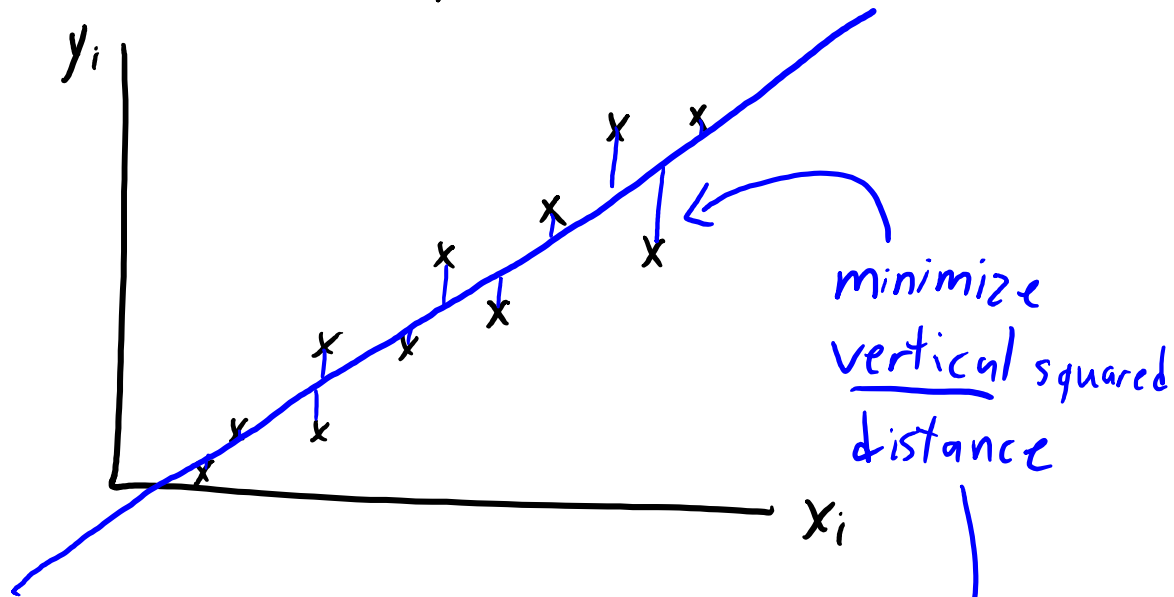
- The “overhead map” approximation just **ignores the “height”**.



- This is a **good approximation** if the world is flat.
- But it's a **poor approximation** if heights are different.

PCA with $d=2$ and $k=1$

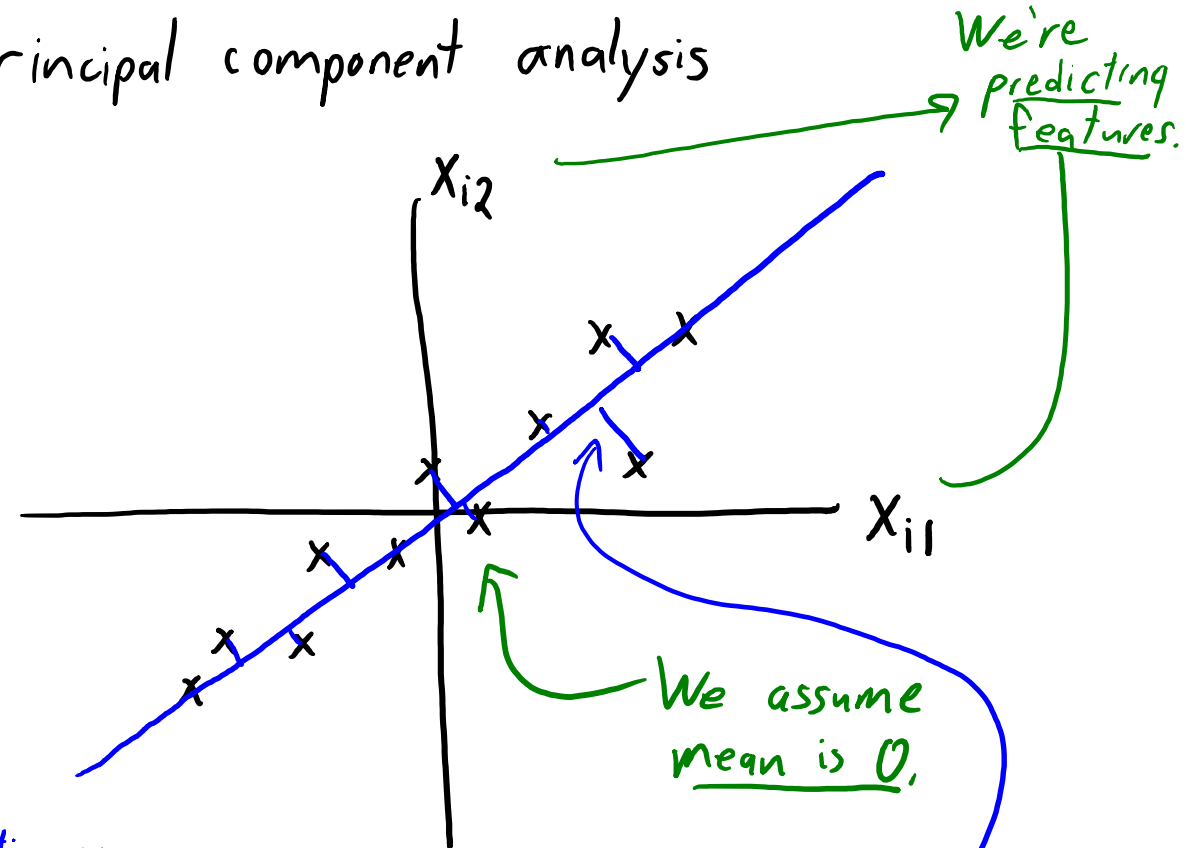
Least squares



minimize vertical squared distance

We only care about predicting y_i

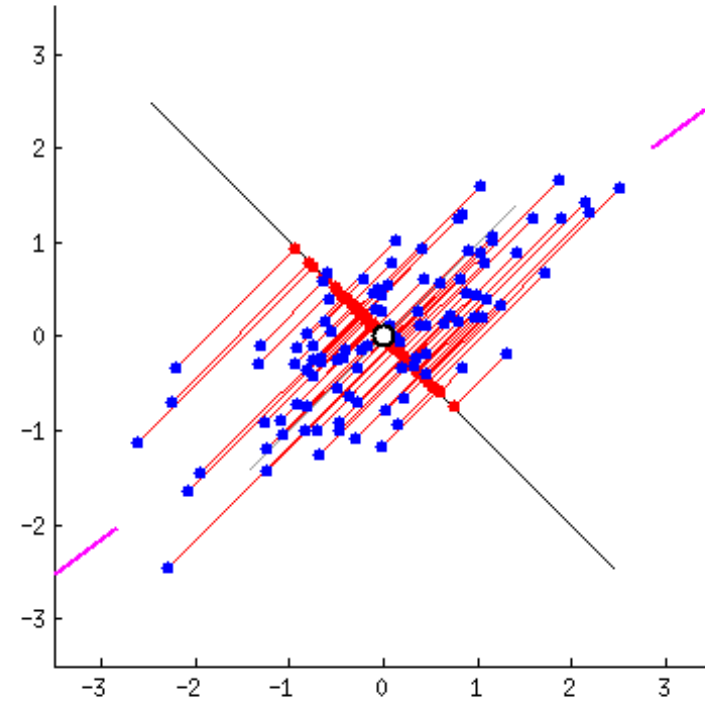
Principal component analysis



We're predicting features.

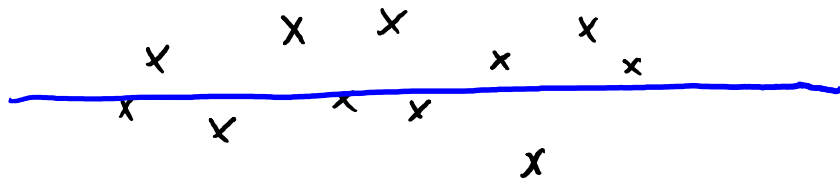
We assume mean is 0,

PCA finds line ' w ' minimizing squared distance in both dimensions.

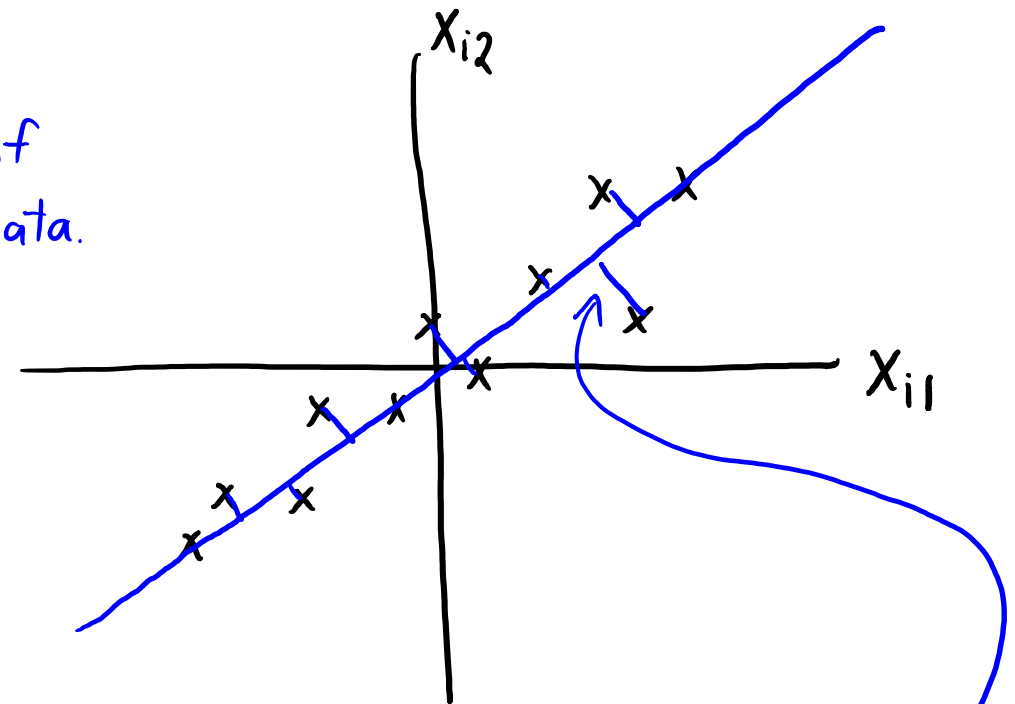


PCA with $d=2$ and $k=1$

Principal component analysis



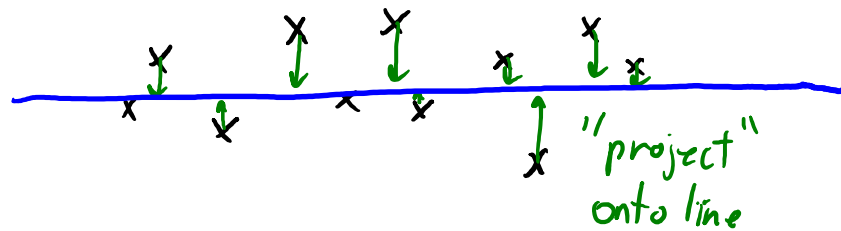
You can think of
'W' as rotating data.



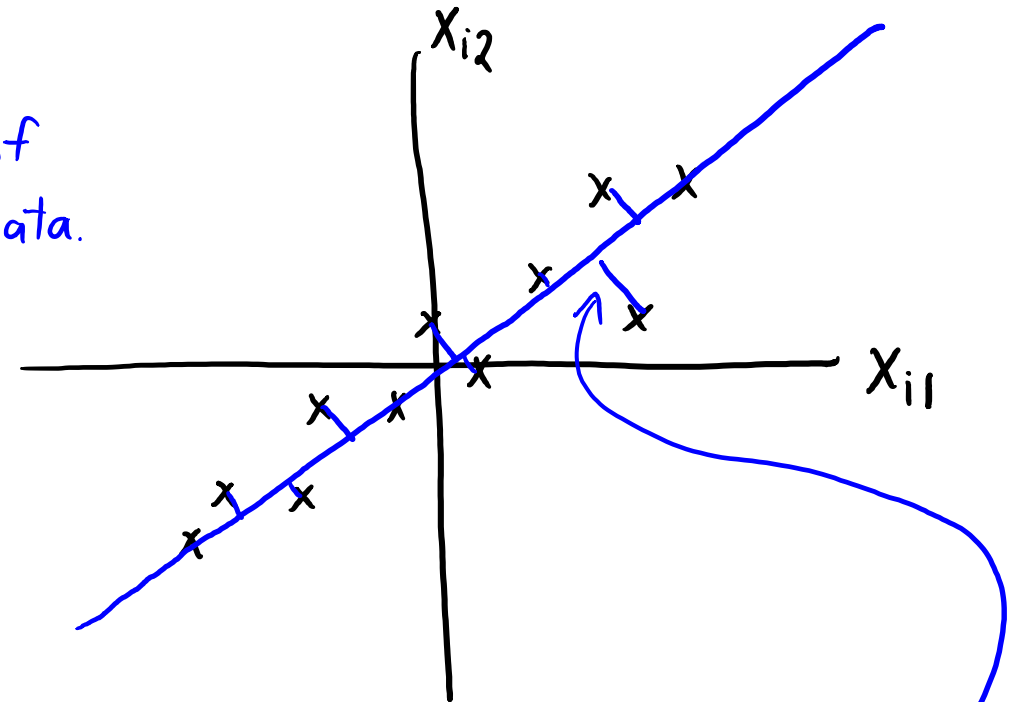
PCA finds line 'W'
minimizing squared distance
in both dimensions.

PCA with $d=2$ and $k=1$

Principal component analysis



You can think of 'W' as rotating data.



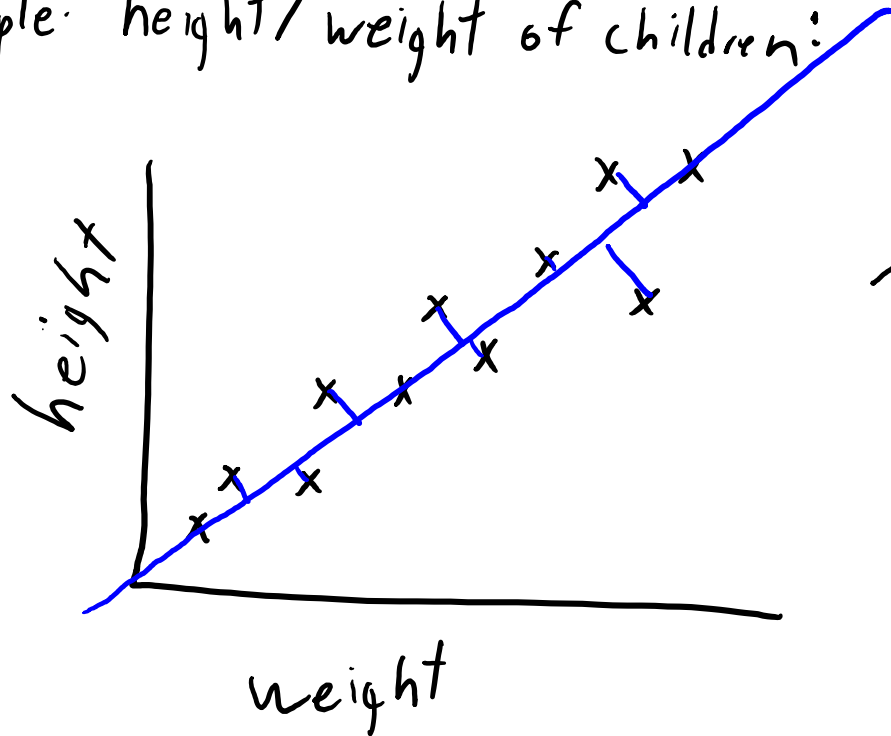
Z_i can be interpreted as position along the line.

(turned a 2d dataset into a 1d dataset)

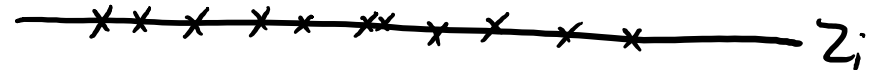

PCA finds line 'W' minimizing squared distance in both dimensions.

PCA with $d=2$ and $k=1$

Example: height/weight of children:



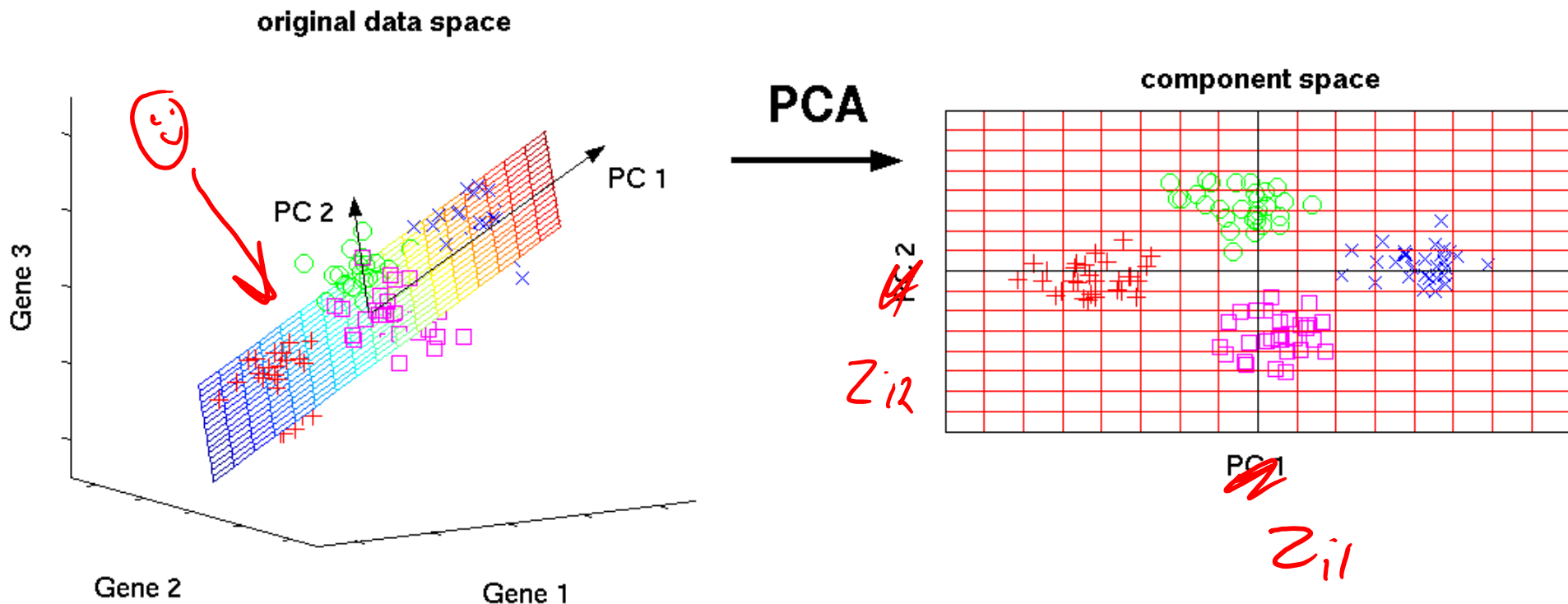
PCA with $k=1$



Latent factor could be viewed as measure of size.

PCA with $d=3$ and $k=2$.

- With $d=3$, PCA ($k=1$) finds **line minimizing squared distance** to x_i .
- With $d=3$, PCA ($k=2$) finds **plane minimizing squared distance** to x_i .



Summary

- **MAP estimation** directly models $p(w | X, y)$.
 - Gives probabilistic interpretation to regularization.
- **Losses for weird scenarios** are possible using MLE/MAP:
 - Ordinal labels, count labels, censored labels, unbalanced labels.
- **Latent-factor models:**
 - Try to learn basis Z from training examples X .
 - Usually, the z_i are “part weights” for “parts” w_c .
 - Useful for dimensionality reduction, visualization, factor discovery, etc.
- **Principal component analysis:**
 - Writes each training examples as linear combination of parts.
 - We learn both the “parts” ‘ W ’ and the “features” Z .
 - We can view ‘ W ’ as best lower-dimensional hyper-plane.
 - We can view ‘ Z ’ as the coordinates in the lower-dimensional hyper-plane.
- Next time: PCA in 4 lines of code.

Review Questions

- Q1: How does Bayes' rule connect likelihood and prior?
- Q2: How is regularization related to prior?
- Q3: What does it mean for PCA to compress the data?
- Q4: What is the difference between the residuals of linear regression and the residuals of PCA?