Math Primer

jenhsin0@gmail.com

Roadmap

- Structure in data; features and their signals
  - Entropy
  - Variable dependency
  - Dimensionality reduction

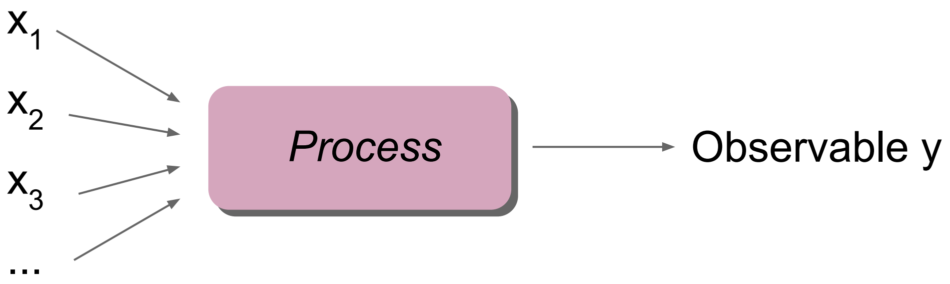
- Making predictions with models
  - Graph representation of models
  - Activation functions in NNet
  - Loss functions
  - Gradient descent

Structure in data

- Some interpretations to “structure in data”
  - Given some data, one can predict other data points with some confidence
  - One can compress the data, i.e., store the same amount of information, with less space

$$A = (1, 2, 6, 2, 4, 7)$$

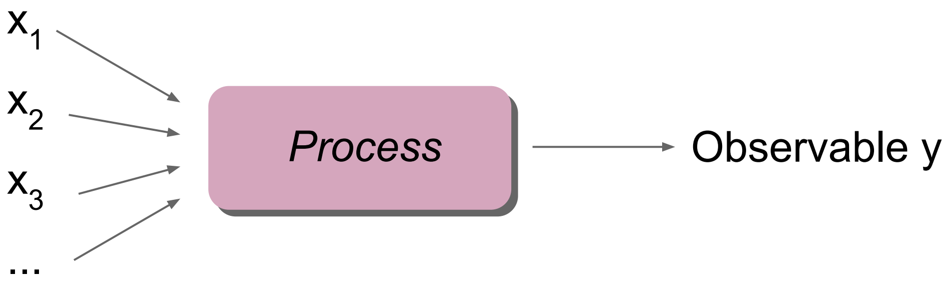
$$B = (1, 2, 1, 2, 1, 2)$$



Structure in data

- Quantify as *Entropy* of a process

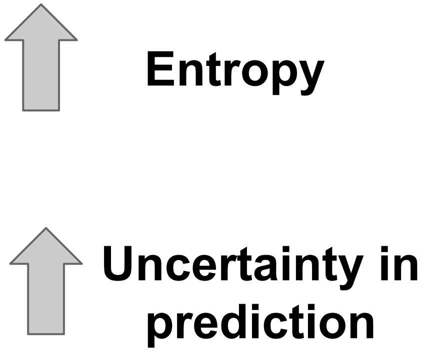
$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$

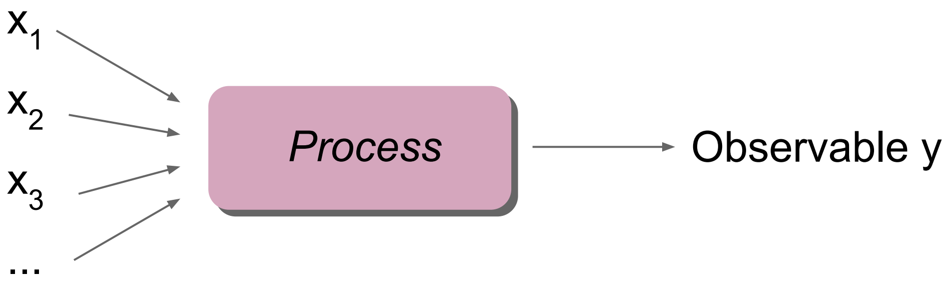


Structure in data

- Quantify as *Entropy* of a process

$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$





Entropy of a process

$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$

p(x) = {1/2, 1/2}

p(x) = {1/6, 1/6, 1/6, 1/6, 1/6, 1/6}

Entropy of a process

$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$

Entropy

Harder to predict

p(x) = {1/2, 1/2}

p(x) = {1/6, 1/6, 1/6, 1/6, 1/6, 1/6}

Entropy of a process

$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$

p(x) = {1/2, 1/2}

p(x) = {1/5, 4/5}

Entropy of a process

$$H(X) = -\sum_{i=1}^{N} p(x_i)\, log_2\, p(x_i)$$

Entropy

Harder to predict

p(x) = {1/2, 1/2}

p(x) = {1/5, 4/5}

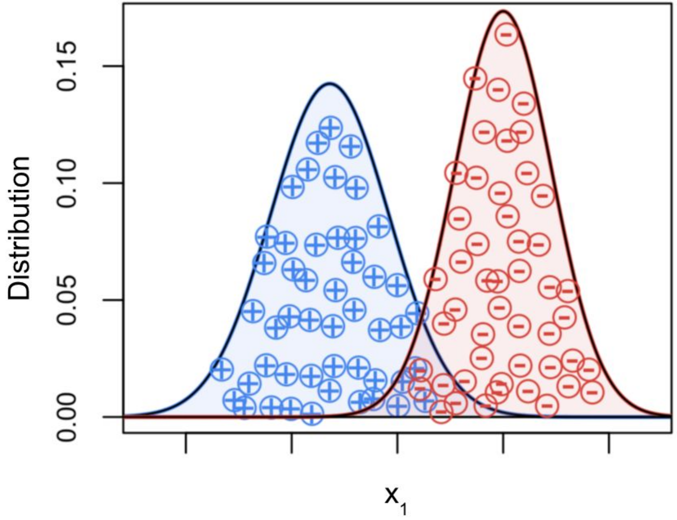
Information in features

- Prediction is only possible if there are information-rich signals

- Poses an upper bound on model performance and confidence in prediction

Information in features

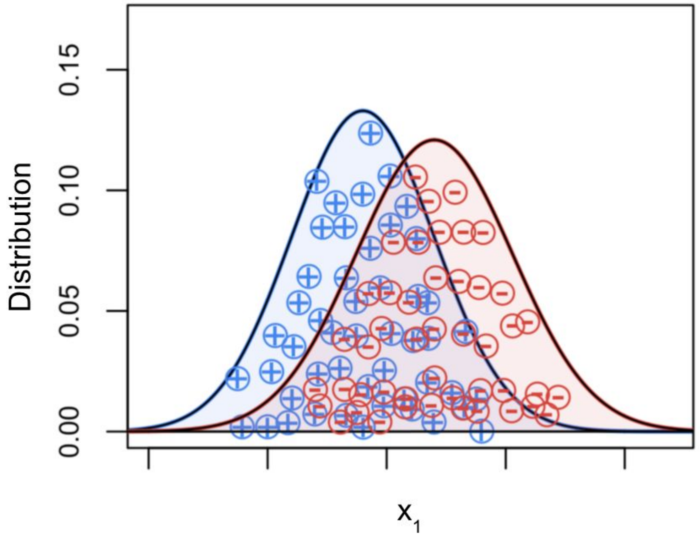
- Prediction is only possible if there are information-rich signals
- Poses an upper bound on model performance and confidence in prediction



Continuous variable x_1 → *Process* → Observable y = {0 or 1}

Information in features

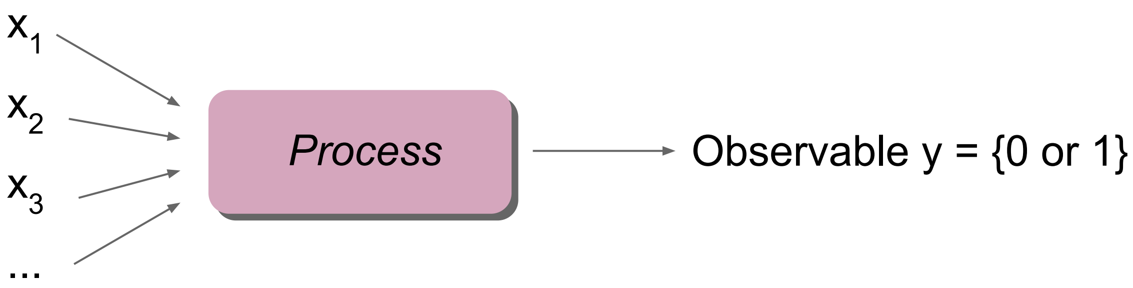
- Prediction is only possible if there are information-rich signals
- Poses an upper bound on model performance and confidence in prediction



Continuous variable x_1 → *Process* → Observable y = {0 or 1}

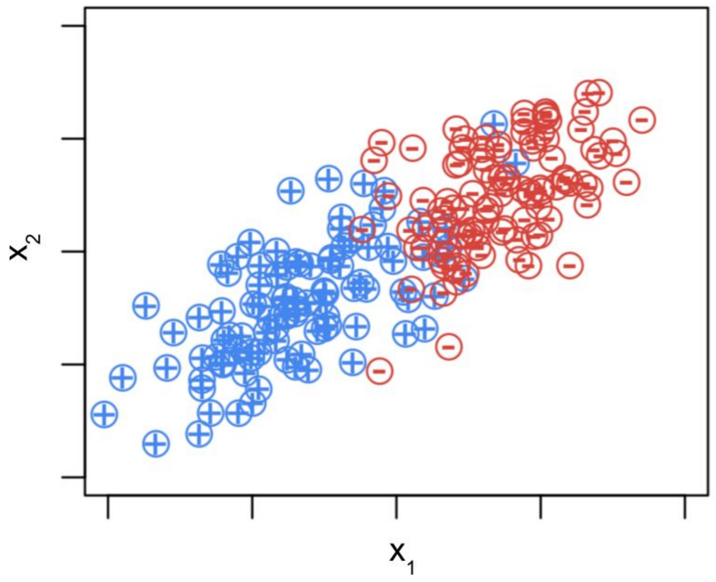
Information in features

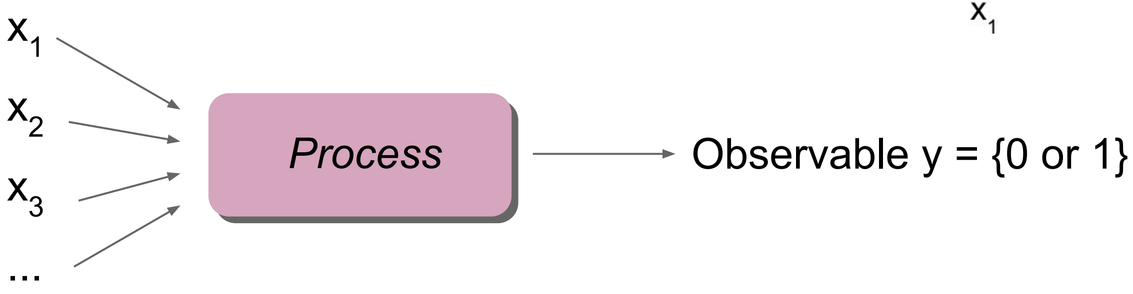
- More features, more information?
- Features can carry redundant information



Information in features

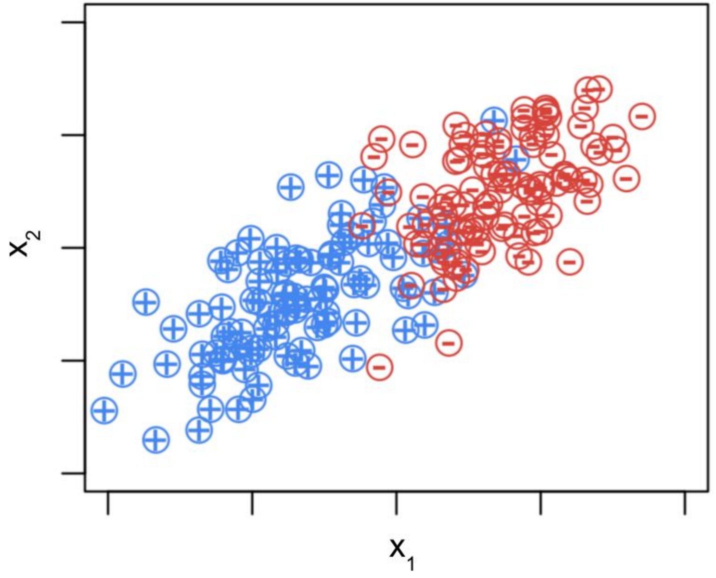
- More features, more information?
- Features can carry redundant information

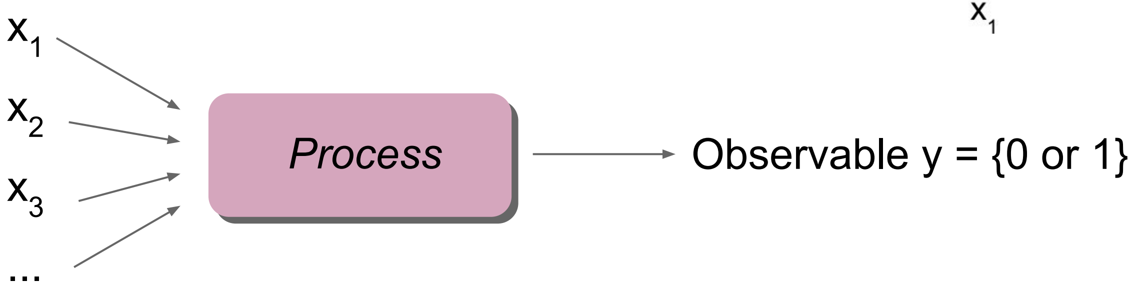




Information in features

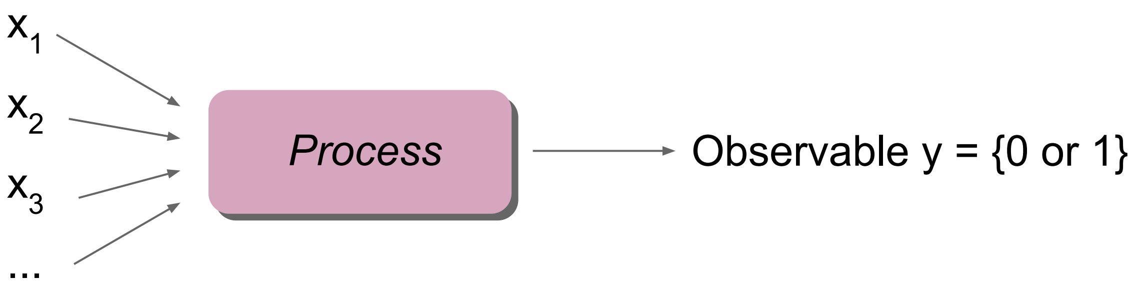
- More features, more information?
- Features can carry redundant information
- For continuous variables that are linearly related, can characterize with the familiar Pearson correlation coefficient





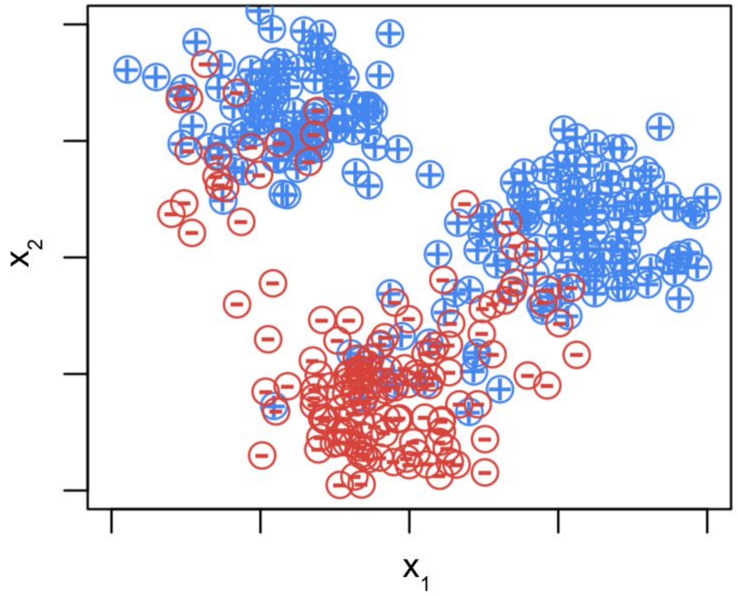
Information in features

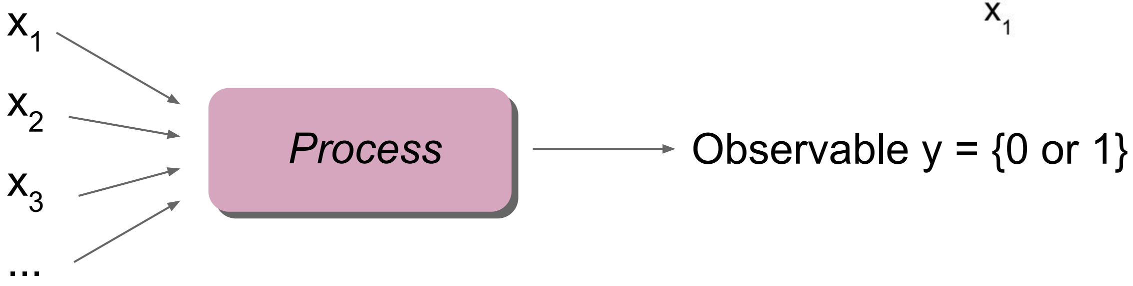
- Variables can be related but not linearly
- Variables can be categorical
- Correlation coefficient wouldn't work



Information in features

- Variables can be related but not linearly
- Variables can be categorical
- Correlation coefficient wouldn't work



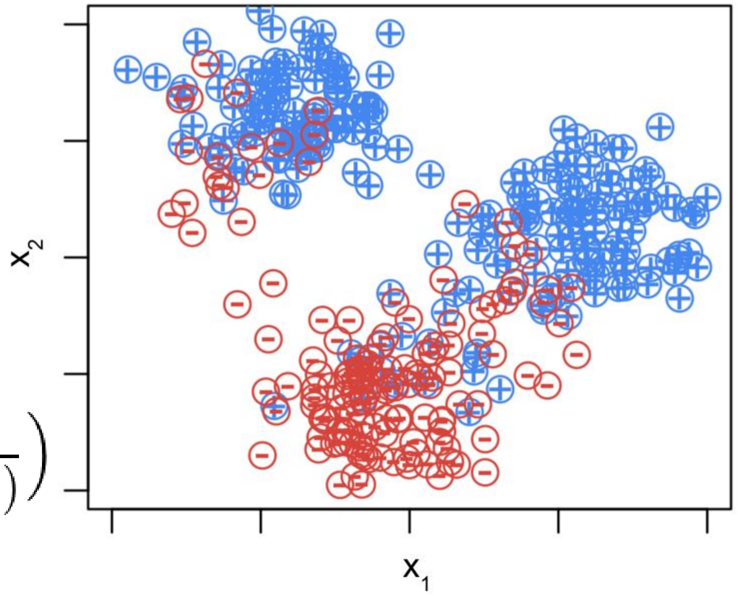


Information in features

- ***Mutual information*** also applies to non-linear dependency

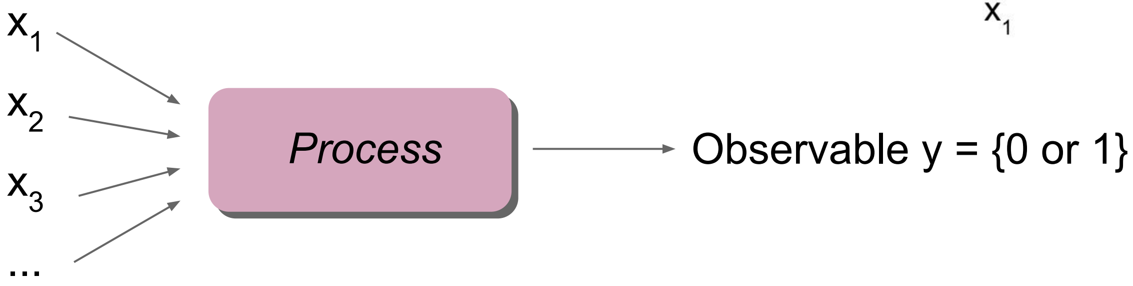
- Based on the joint probability distribution

- I = 0 if two variables are independent; I = entropy of the variable if they're the same



$$I(X_1; X_2) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2) \, log\left(\frac{p(x_1, x_2)}{p(x_1)\, p(x_2)}\right)$$

(discrete version)

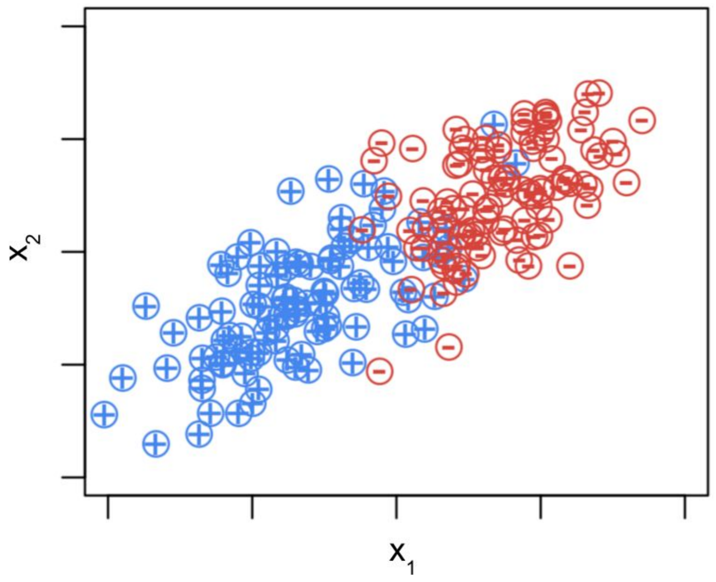


Dimensionality Reduction

- If different features carry redundant information, then how to interpret what signal is most informative?

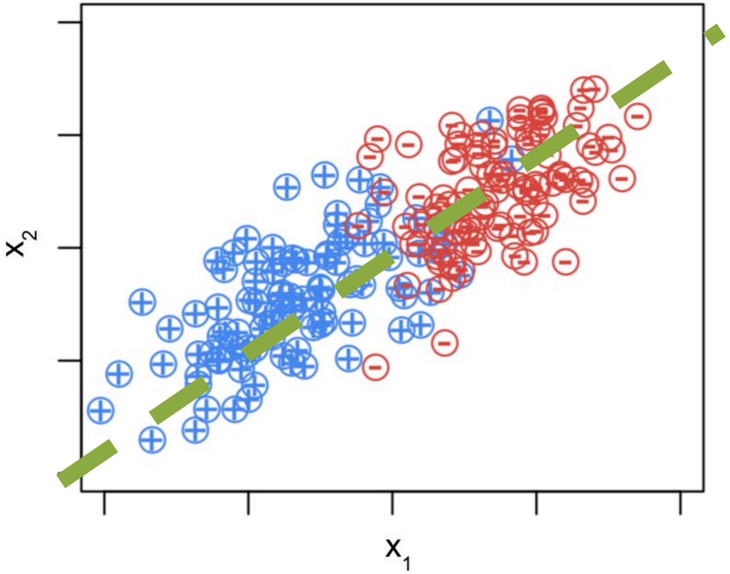
Dimensionality Reduction

- Neither x_1 nor x_2 is the most informative signal



Dimensionality Reduction

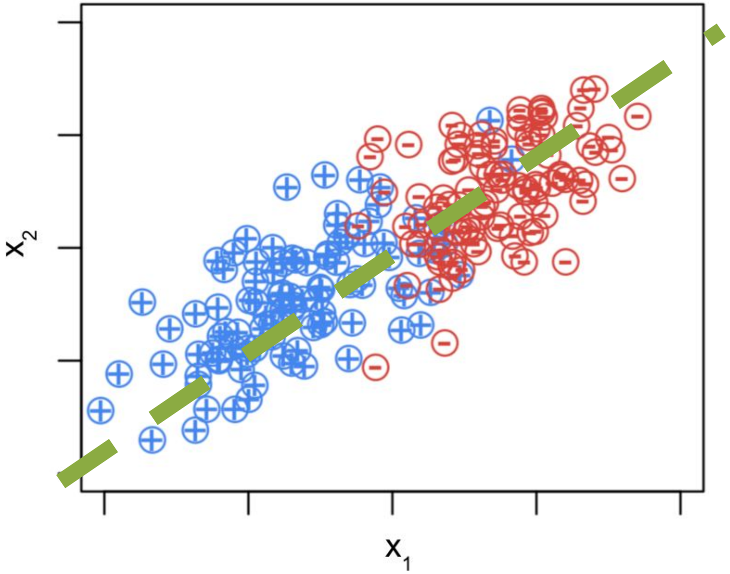
- Neither x_1 nor x_2 is the most informative signal
- It seems to be this new axis
- Why? Most variance fall along this axis, so its signal-to-noise ratio is most promising



PCA

- How to extract this new axis?
- Represent features as a matrix
- m-many features, and n-many samples

$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^n \\ x_2^1 & x_1^2 & \dots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ x_m^1 & x_m^2 & \dots & x_m^n \end{bmatrix}$$



PCA

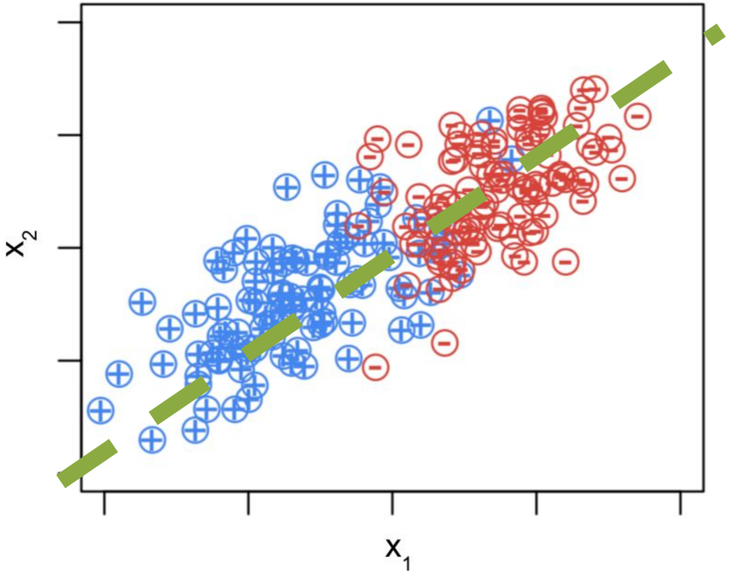
- We want new axes to have the most variances, so start with measuring variances between variables

- Start with a simple example of $\mathbf{x_1}$ and $\mathbf{x_2}$

- n-many samples:

$$\mathbf{x_1} = \{x_1^1, x_1^2, ..., x_1^n\}$$

$$\mathbf{x_2} = \{x_2^1, x_2^2, ..., x_2^n\}$$



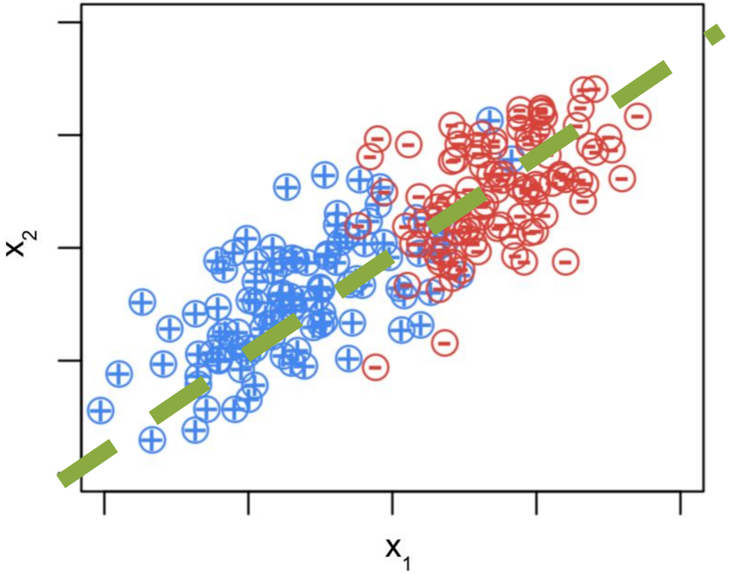
PCA

- We want new axes to have the most variances, so start with measuring variances between variables
- Start with a simple example of $\mathbf{x_1}$ and $\mathbf{x_2}$
- n-many samples:

$$\mathbf{x_1} = \{x_1^1, x_1^2, ..., x_1^n\}$$

$$\mathbf{x_2} = \{x_2^1, x_2^2, ..., x_2^n\}$$

$$cov(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} x_1^i \, x_2^i$$ (if $\mathbf{x_1}$ and $\mathbf{x_2}$ are standardized to mean 0)



PCA

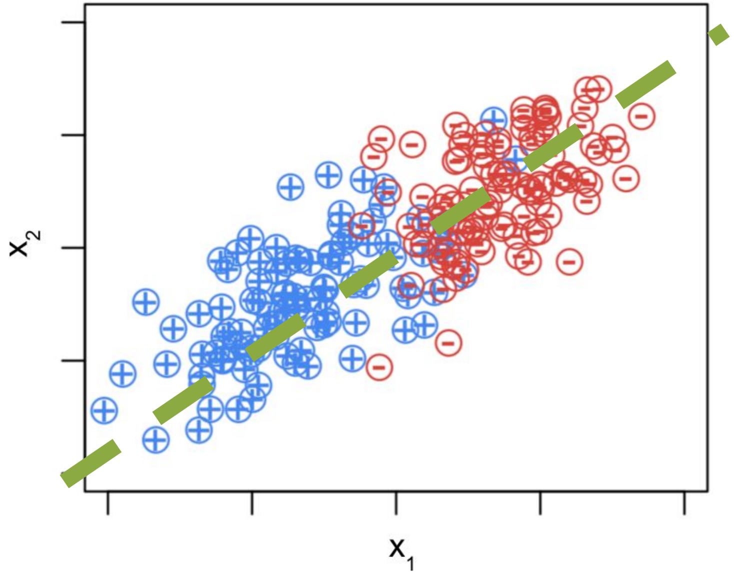
- Covariance can also be written as matrix multiplication

$$cov(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} x_1^i \, x_2^i = \frac{1}{n}\mathbf{x_1}\mathbf{x_2}^T$$

(works if standardize x_1 and x_2 to mean 0)

- We can generalize this to the entire feature matrix **X**

$$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T$$



PCA

- What does the covariance matrix of **X** look like?

$$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T = \begin{bmatrix} d_{11} & & \text{Off-diagonal} & \\ & d_{22} & \text{elements} & \\ & & \ddots & \\ \text{Off-diagonal elements} & & & d_{mm} \end{bmatrix}$$

PCA

- What does the covariance matrix of **X** look like?

$$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T = \begin{bmatrix} d_{11} & & \text{Off-diagonal elements} \\ & d_{22} & \\ & & \ddots & \\ \text{Off-diagonal elements} & & & d_{mm} \end{bmatrix}$$

Variance of each feature x_i. Large variance in that feature means that's a dimensionality we want to keep.

PCA

- What does the covariance matrix of **X** look like?

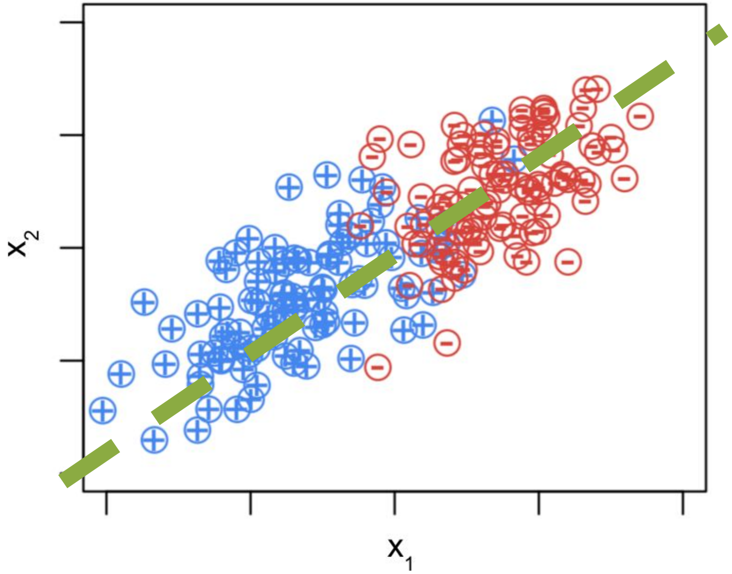
$$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T = \begin{bmatrix} d_{11} & & \text{Off-diagonal elements} \\ & d_{22} & \\ \text{Off-diagonal elements} & \ddots & \\ & & d_{mm} \end{bmatrix}$$

Covariance between two features.
Large covariance means there is redundancy.

PCA

- Goal: transform covariance matrix so that off-diagonal elements are 0, and diagonal elements are rank ordered

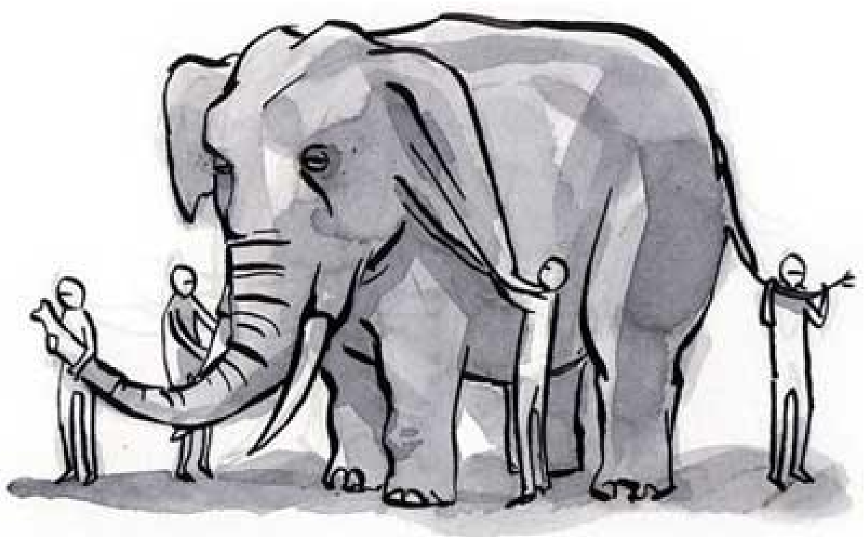
$$\mathbf{C_X} = \frac{1}{n}\mathbf{X}\mathbf{X}^T = \begin{bmatrix} d_{11} & & \text{Off-diagonal elements} \\ & d_{22} & \\ \text{Off-diagonal elements} & \ddots & \\ & & d_{mm} \end{bmatrix}$$



- This transformed matrix tells us about the order of most informative new axes
- Eigenvalue decomposition is a way to do this

Making predictions

- 15-minute check

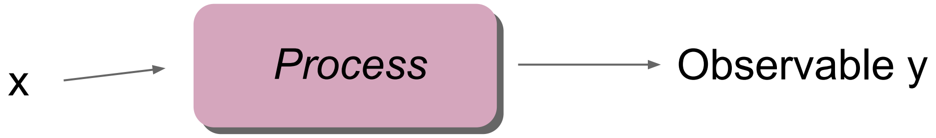


Making predictions

- Given samples of a variable, we want to make predictions on the outcome

- A friendly linear model

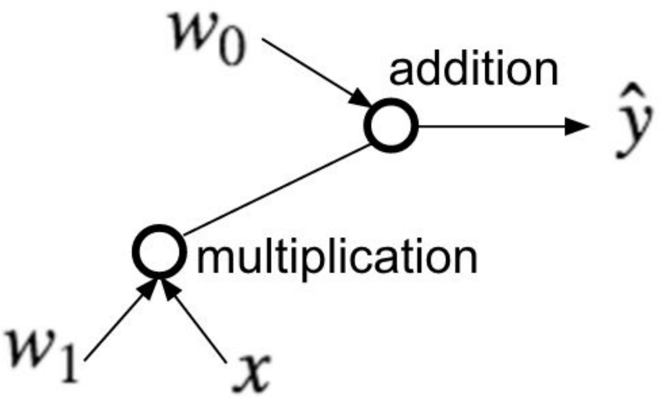
$$w_0 + w_1 x \;=\; \hat{y}$$



Making predictions

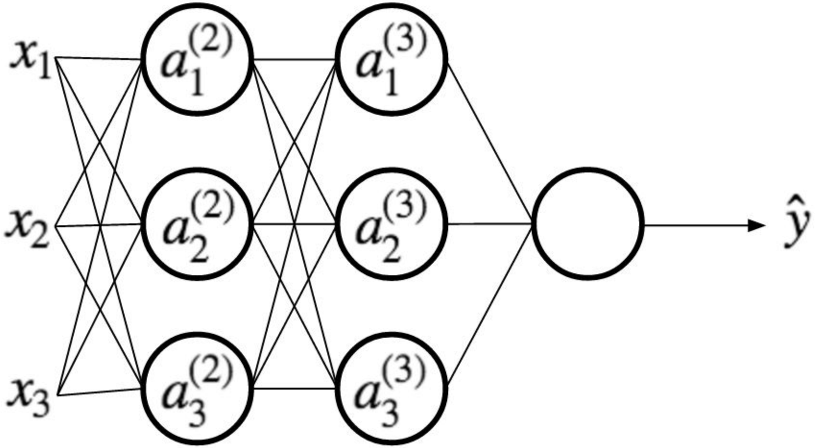
- Given samples of a variable, we want to make predictions on the outcome
- A friendly linear model

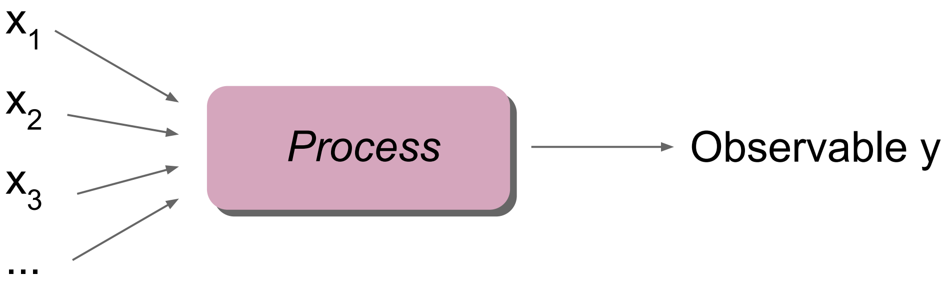
$$w_0 + w_1 x \;=\; \hat{y}$$



Making predictions

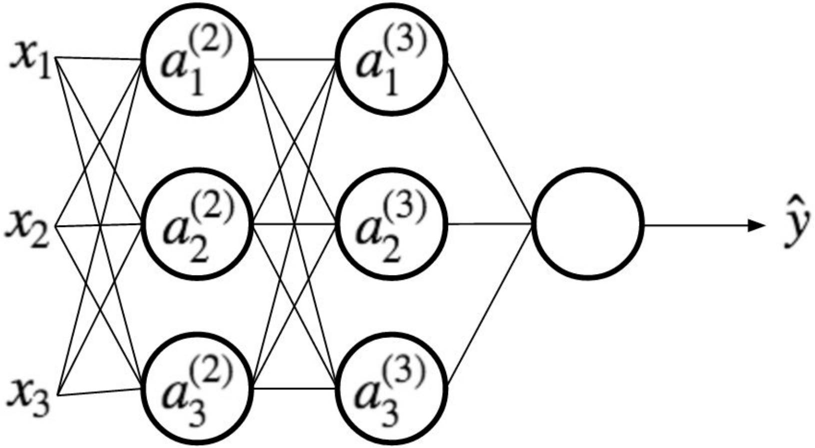
- We can represent different models as graphs of how data flow and are transformed by operations

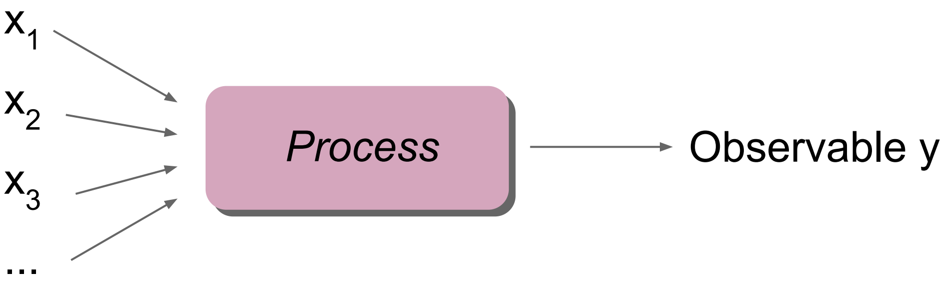




Making predictions

- Activation function: can be linear, or can be other functions, there are many choices:
  - Sigmoid
  - tanh(x)
  - Maxout
  - ReLU = max(0, x)
  - … …
- When in doubt, try ReLU





Model output

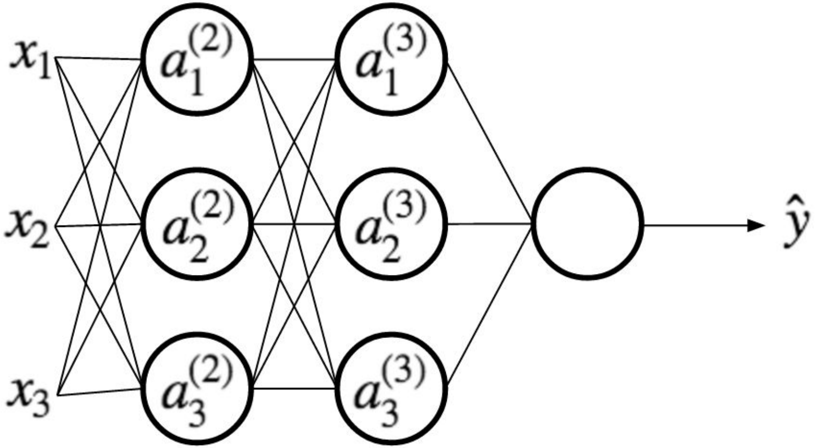
- Unless with choice of activation function, generally output can be any real number

$$\hat{y} \in [-\infty, +\infty]$$

- What if need to make binary prediction? Like logistic regression, can use sigmoid function

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

$$\hat{y} \in [0, 1]$$



Model output

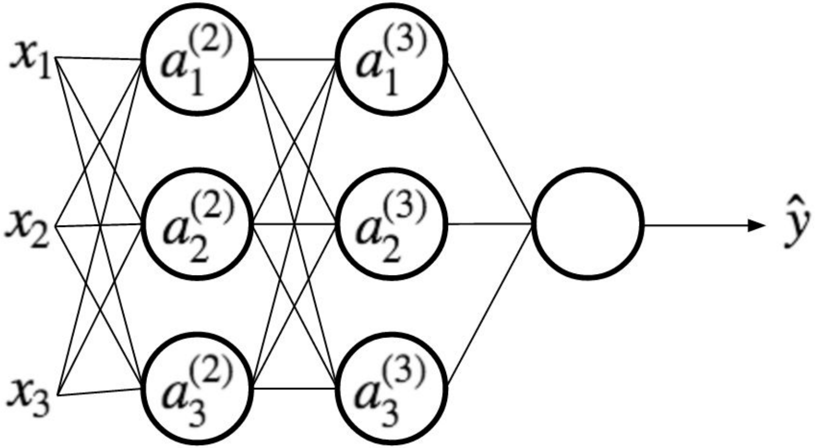
- Unless with choice of activation function, generally output can be any real number

$$\hat{y} \in [-\infty, +\infty]$$

- What if need to make categorical prediction? Then use ***softmax function***

$$S_j(\mathbf{u}) = \frac{e^{u_j}}{\sum_{k=1}^{K} e^{u_k}}, \qquad j \in [1, 2, ...K]$$

- Can see that normalization condition is held



Loss function

- Models have parameters/weights w_i; tune the model to make as little prediction error as possible

- Need to define loss function; many options:
  - L1 - sum the abs. differences b/t predictions and observations
  - L2 - sum the square difference b/t predictions and observations
  - Most applicable for continuous predictions

$$L(w_0, w_1, ...) = \sum \left(\hat{y} - y\right)^2$$

(L2 loss)

Loss function

- What about loss function for binary or categorical predictions?

- In these cases we predict probability distributions

- We can use ***cross entropy*** to compare predicted probability against observed probability

$$H(p, \hat{p}) = \sum_i p_i \, log(\hat{p_i}) = -y \, log(\hat{y}) - (1 - y) \, log(1 - \hat{y})$$

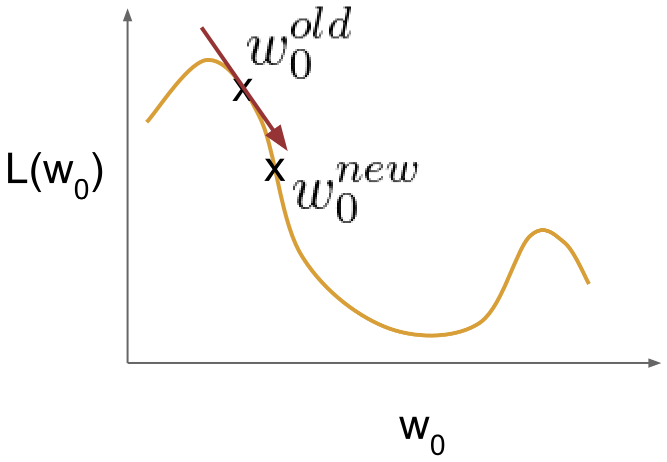
(Commonly known as log loss; shown here for logistic regression)

Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, ...)$$

learning rate

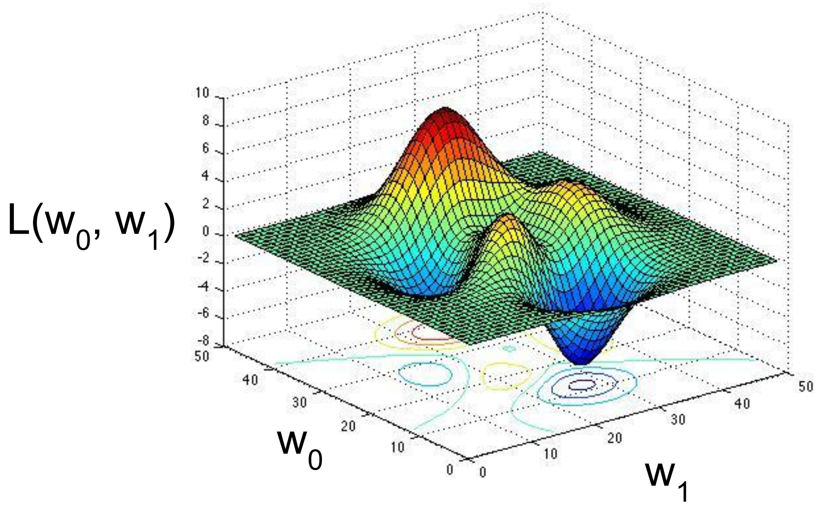


Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, ...)$$

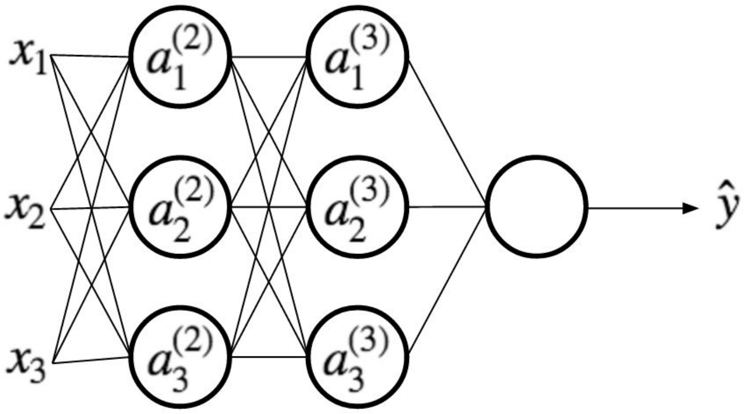
learning rate



Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, ...)$$



Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, \ldots)$$

$$x \longrightarrow f \longrightarrow g \longrightarrow \hat{y}$$

Want: $\dfrac{\partial L}{\partial w_0}$

$$g(f(x)) = \hat{y}$$

Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, ...)$$

$$x \longrightarrow f \longrightarrow g \longrightarrow \hat{y}$$

$$g(f(x)) = \hat{y}$$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{w_0}$$

Chain rule

Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, ...)$$

$$x \longrightarrow f \xrightarrow{\frac{\partial f}{\partial w_0}} g \xrightarrow{\frac{\partial g}{\partial f}} \hat{y}$$

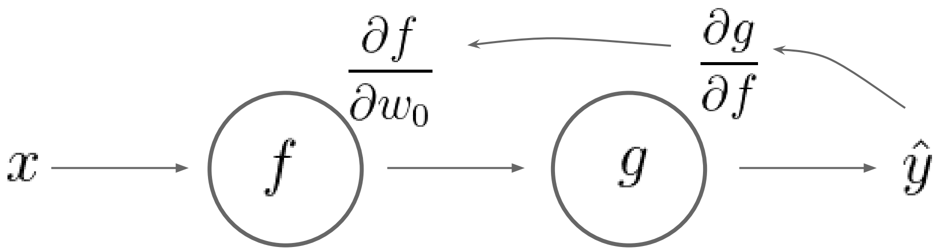
$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial w_0}$$

$$g(f(x)) = \hat{y}$$

Gradient descent

- To choose good weights, iteratively update each weight in a way that decreases L

$$w_0^{new} = w_0^{old} - \alpha \frac{\partial}{\partial w_0} L(w_0, w_1, \ldots)$$



$$g(f(x)) = \hat{y}$$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial g} \frac{\partial g}{\partial f} \frac{\partial f}{\partial w_0}$$

Appendix

- Not covered here: regularization (to prevent overfitting, which lead to ungeneralizable models), tree-based model, SVM, many more...

- Some resources:
  - A neural network playground
  - Udacity Deep Learning
  - CS231n - CovNet/Computer vision
  - CS224d - Deep learning for NLP
  - K. P. Murphy, Machine Learning
  - E. T. Jaynes, Probability Theory