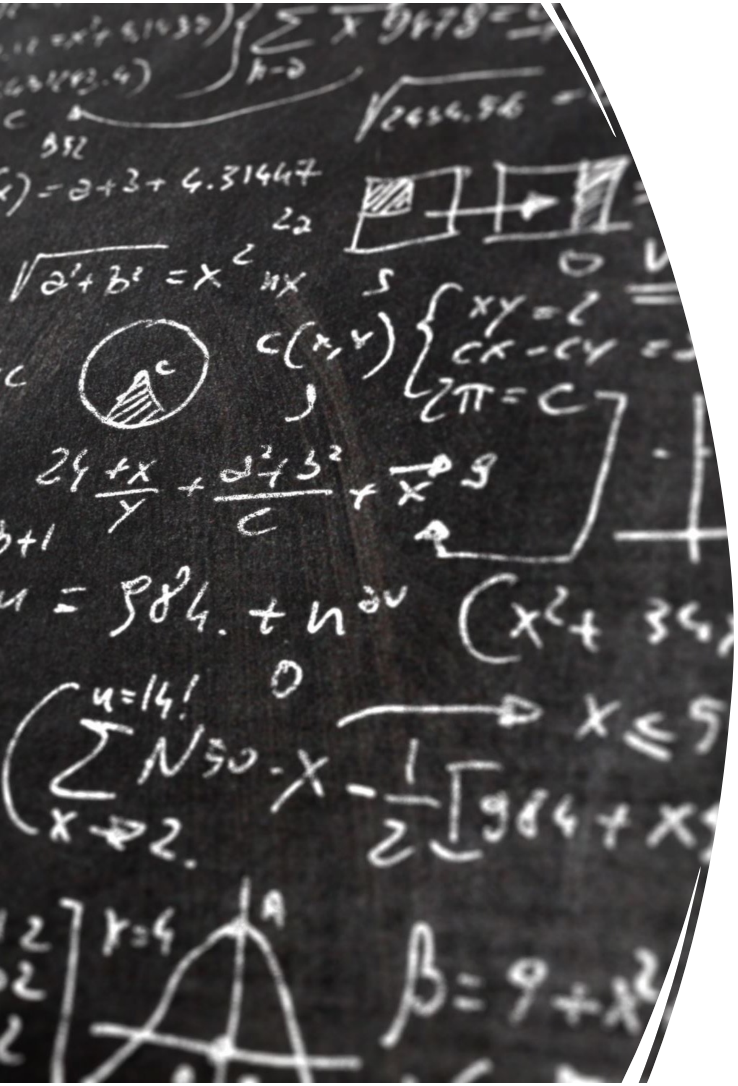


CS 383 – Computational Text Analysis

Lecture 10
Feed Forward Networks

Adam Poliak

02/20/2023

Slides adapted Dan Jurafsky, Jordan Boyd-Graber, Daniel Khashabi

Announcements

- HW04
  - Due next Wednesday 03/01
  - Will be released later today
- Reading 05
  - Due Monday 02/27 - CTA/TADA/CSS papers using Word Embeddings
- Office hours:
  - Need to change time this week

Final Project

Deliverables:

- Ideation
  250 write up – what idea do you have
  Due sometime next week

- Proposal
  Due write after spring break

- Presentation
  Maybe last day of class

- Writeup, code, data
  End of finals?

Recap – last week

- Regression vs classification

- Linear Regression vs Logistic Regression

- Learning weights
  - SGD!

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- Feed forward
- Non-linear Activation Functions
- Computation Graph
- Back-propagation
- Neural LM

SGD Example

Working through an example

- One step of gradient descent
- A mini-sentiment example, where the true y=1 (positive)
- Two features:

$x_1 = 3$ (count of positive lexicon words)

$x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$w_1 = w_2 = b = 0$

$\eta = 0.1$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0$;
$x_1 = 3$; $x_2 = 2$

- Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} =$$

Example of gradient descent

$w_1 = w_2 = b = 0$;
$x_1 = 3$; $x_2 = 2$

- Update step for update θ is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} \;=\; \theta_t - \eta \nabla L(f(x;\theta), y)$$

$\eta = 0.1$;

$$\theta^1 =$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} \;=\; \theta_t - \eta \nabla L(f(x;\theta), y)$$

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w\cdot x+b)-y)x_1 \\ (\sigma(w\cdot x+b)-y)x_2 \\ \sigma(w\cdot x+b)-y \end{bmatrix} = \begin{bmatrix} (\sigma(0)-1)x_1 \\ (\sigma(0)-1)x_2 \\ \sigma(0)-1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

$\eta = 0.1;$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks
- Feed forward
- Computation Graph
- Back-propagation

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- Feed forward
- Computation Graph
- Back-propagation

Multinomial Logistic Regression
aka softmax regression, multinomial logit

Softmax: a generalization of the sigmoid

- Takes a vector of *k* values
  - think scores for each class
- Outputs a probability distribution
  - each value in the range [0,1]
  - all the values summing to 1

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \qquad 1 \le i \le k$$

$$softmax(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{j=1}^{k} \exp(z_j)}, \frac{\exp(z_2)}{\sum_{j=1}^{k} \exp(z_j)}, \ldots, \frac{\exp(z_3)}{\sum_{j=1}^{k} \exp(z_j)}\right]$$

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- Feed forward
- Non-linear Activation Functions
- Computation Graph
- Back-propagation
- Neural LM

Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- That can result in choppy movements
- More common to compute gradient over batches of training instances.
- **Batch training**: entire dataset
- **Mini-batch training**: *m* examples (512, or 1024)

Overfitting

- A model that perfectly match the training data has a problem.
- It will also **overfit** to the data, modeling noise
  - A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.
- A good model should be able to **generalize**

Overfitting

+

- This movie drew me in, and it'll do the same to you.

–

I can't tell you how much I hated this movie. It sucked.

Useful or harmless features

X1 = "this"
X2 = "movie
X3 = "hated"

X4 = "drew me in"

4gram features that just "memorize" training set and might cause problems

X5 = "the same to you"
X7 = "tell you how much"

Overfitting

- 4-gram model on tiny data will just memorize the data
  - 100% accuracy on the training set
- But it will be surprised by the novel 4-grams in the test data
  - Low accuracy on test set
- Models that are too powerful can **overfit** the data
  - Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
    - How to avoid overfitting?
      - Regularization in logistic regression
      - Dropout in neural networks

Regularization

- A solution for overfitting
- Add a regularization term *R*(θ) to the loss function
  (for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

- Idea: choose an *R*(θ) that penalizes large weights
  - fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 Regularization (= ridge regression)

- The sum of the squares of the weights
- The name is because this is the (square of the) **L2 norm** $||\theta||_2$, **= Euclidean distance** of θ to the origin.

$$R(\theta) \;=\; ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

- L2 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 Regularization (= lasso regression)

- The sum of the (absolute value of the) weights
- Named after the **L1 norm** $\|W\|_1$, = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

- L1 regularized objective function:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- Feed forward
- Non-linear Activation Functions
- Computation Graph
- Back-propagation
- Neural LM

Prediction: NLP/ML vs CTA/TADA

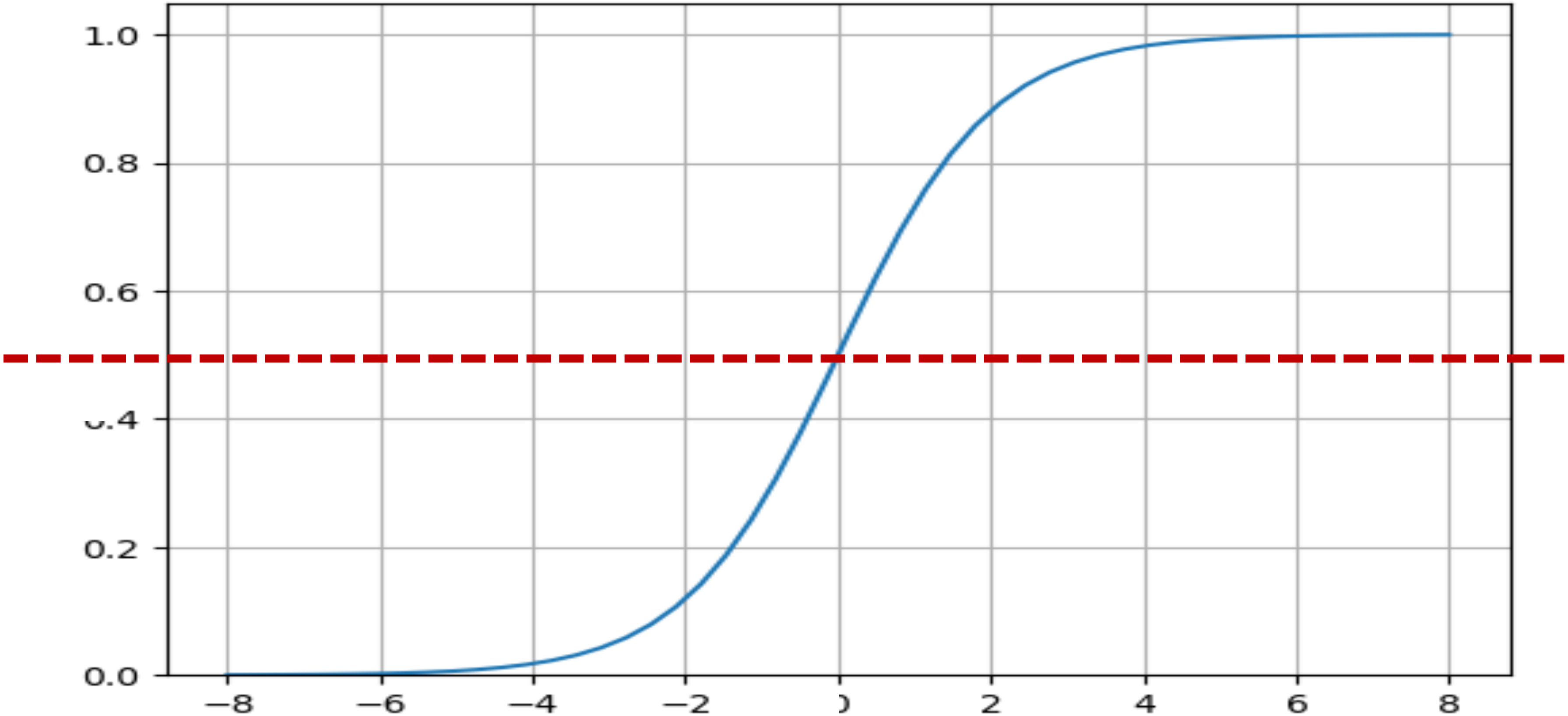
NLP/ML:

- Make prediction about unseen data
  - Predict if a stock will go up or down based on social media posts

CTA/TADA/CSS:

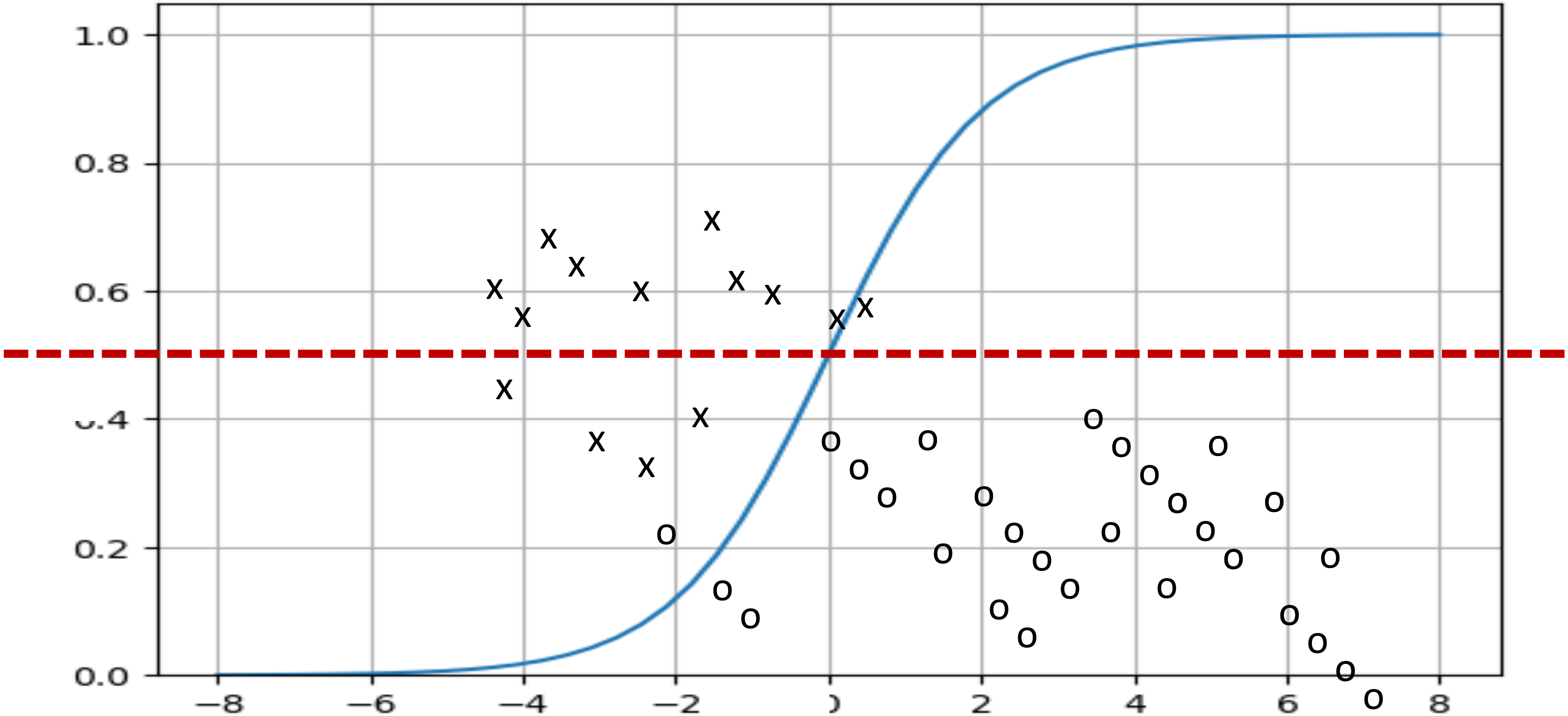
- Apply labels to examples
  - Use previous methods to find differences
- Learn something about different categories
  - Are there different terms/concepts used to describe male vs female professors on course reviews

Logistic Regression



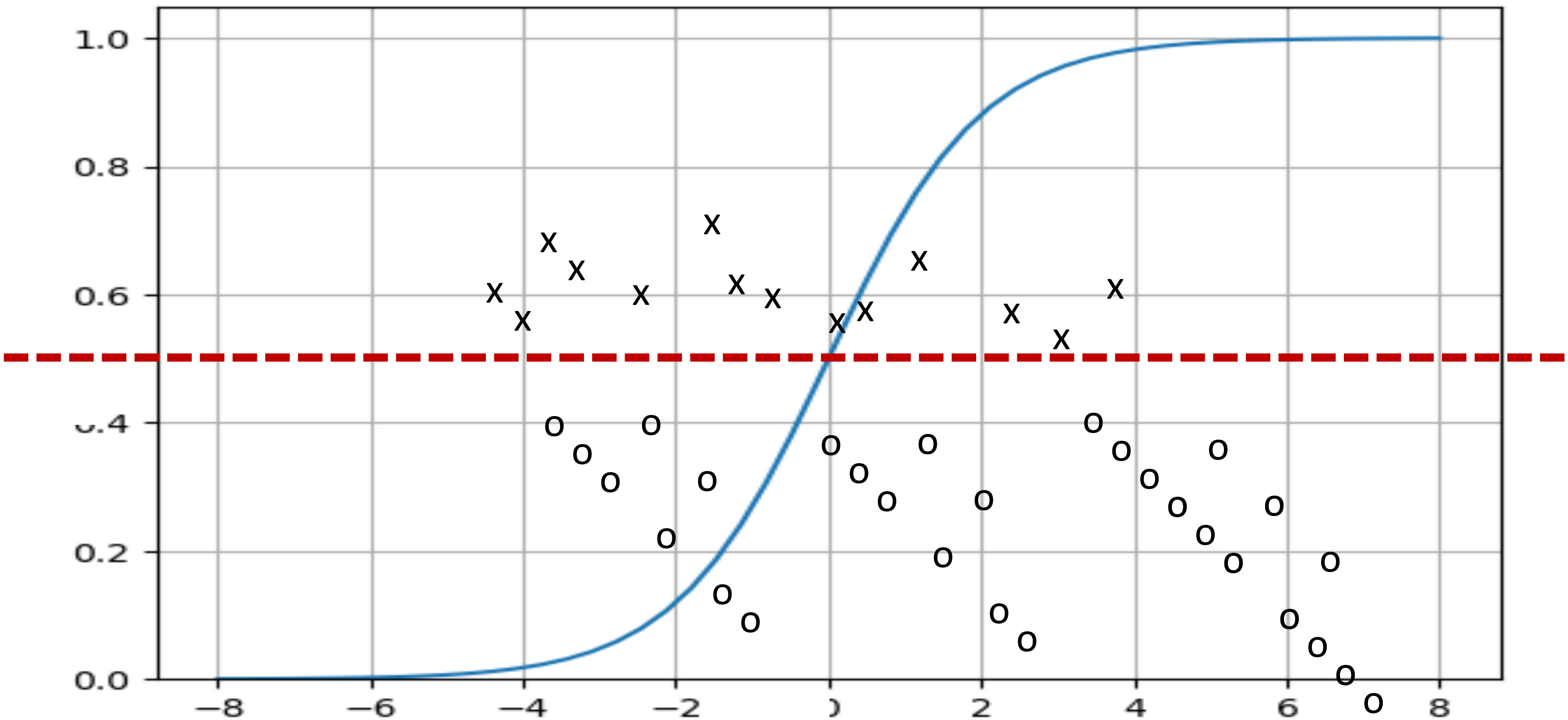
wx + b

Logistic Regression



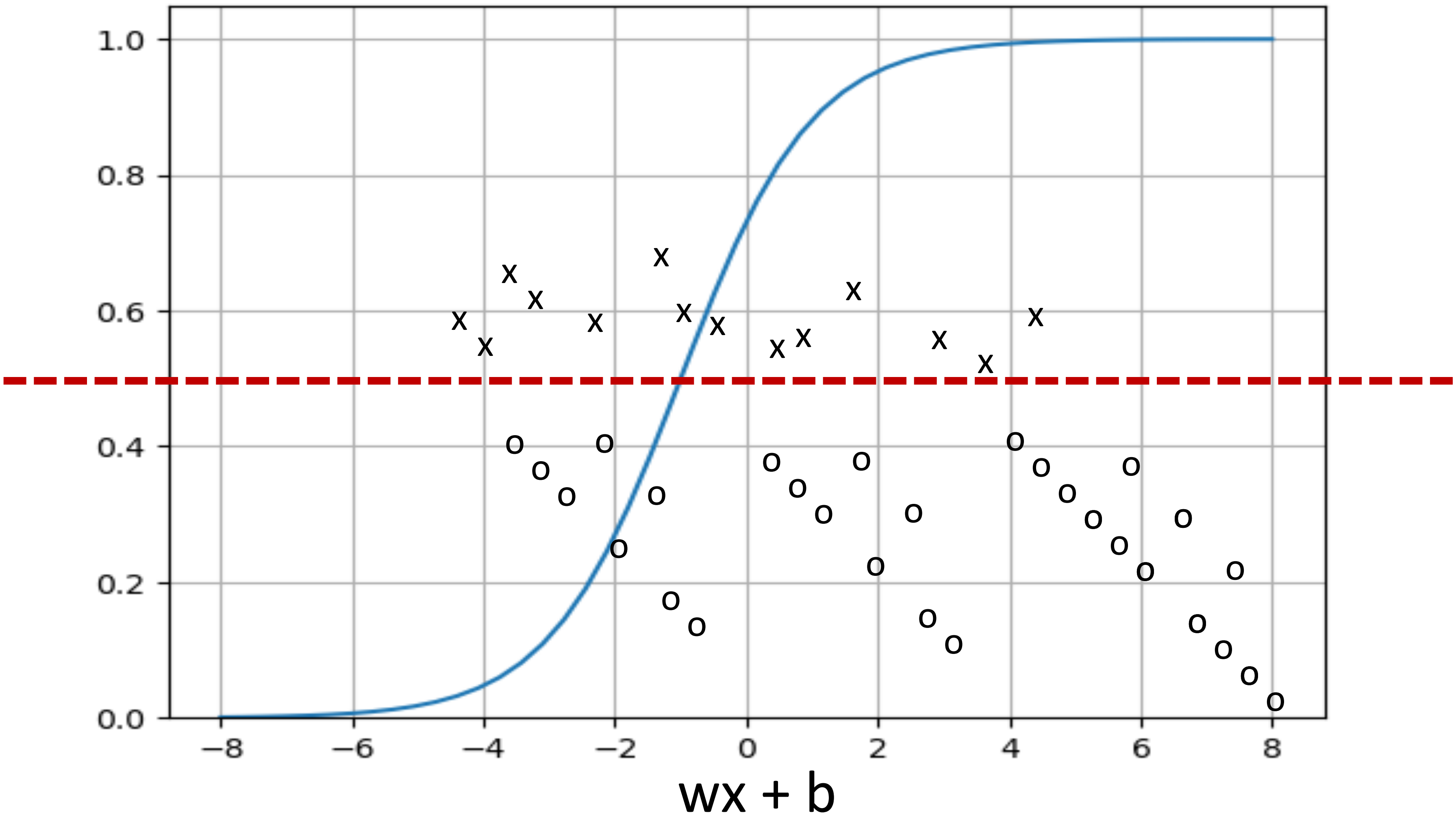
wx + b

Logistic Regression

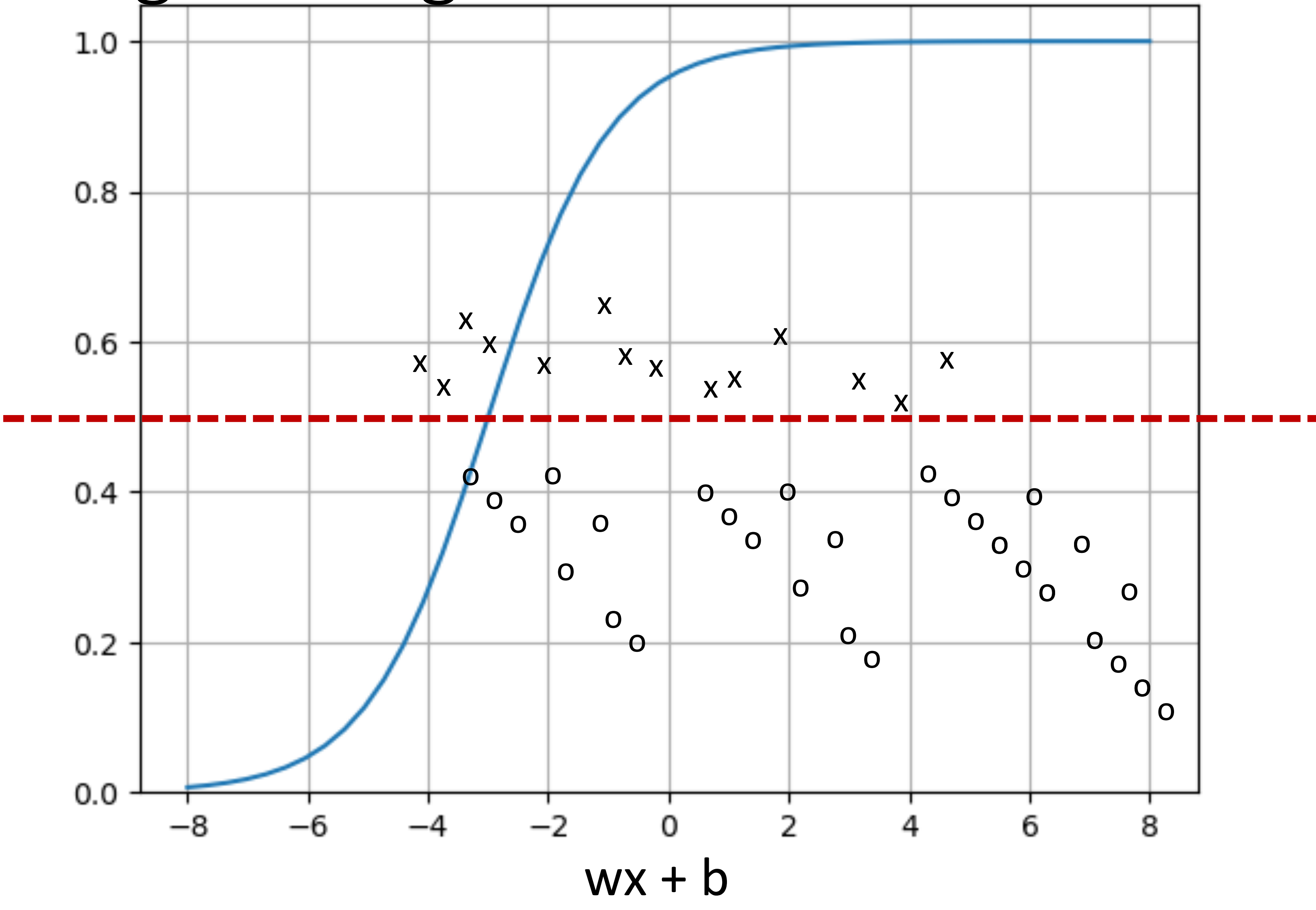


wx + b

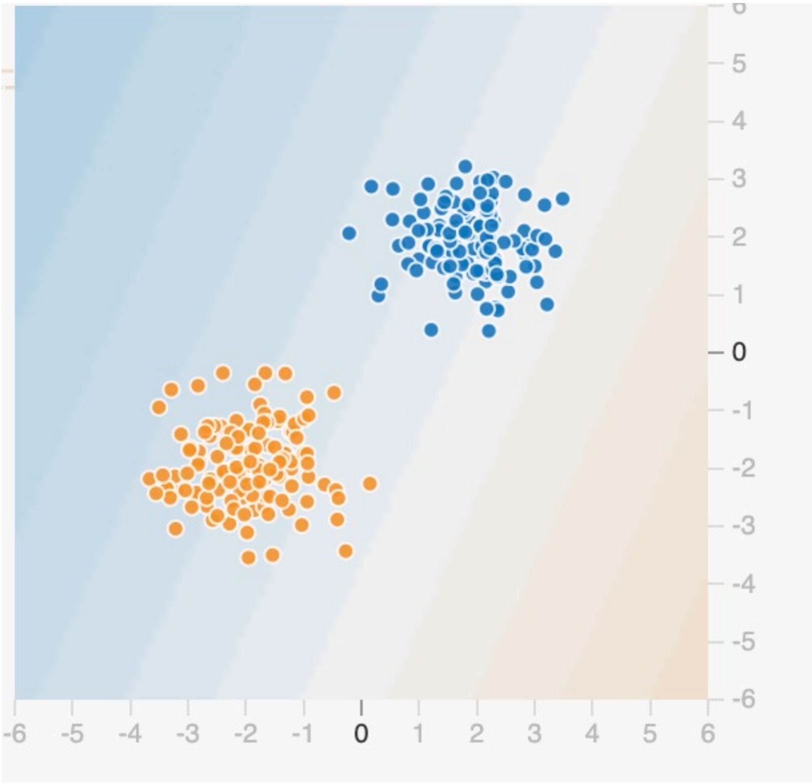
Logistic Regression

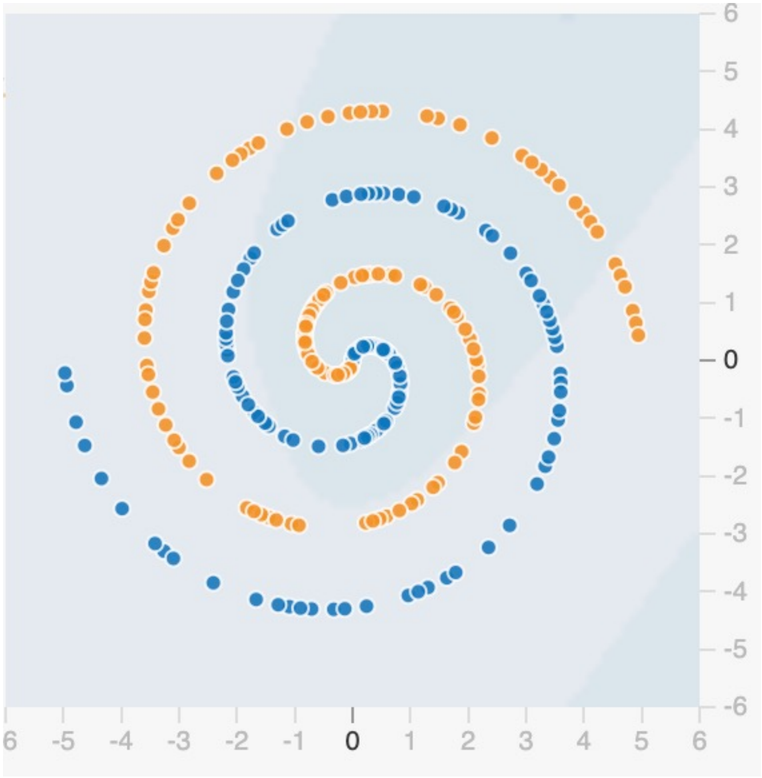


Logistic Regression

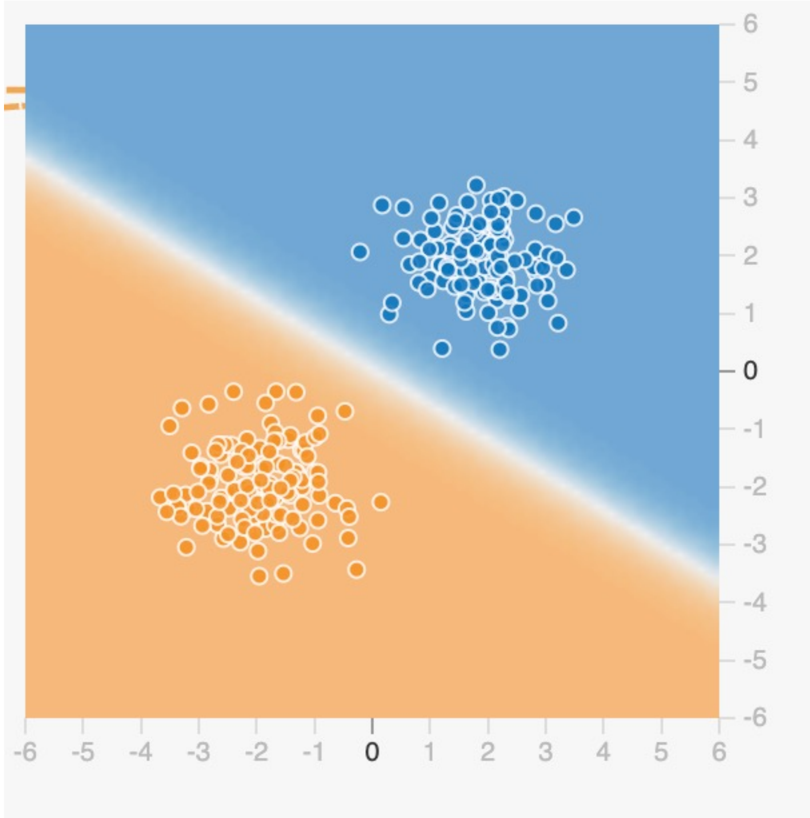


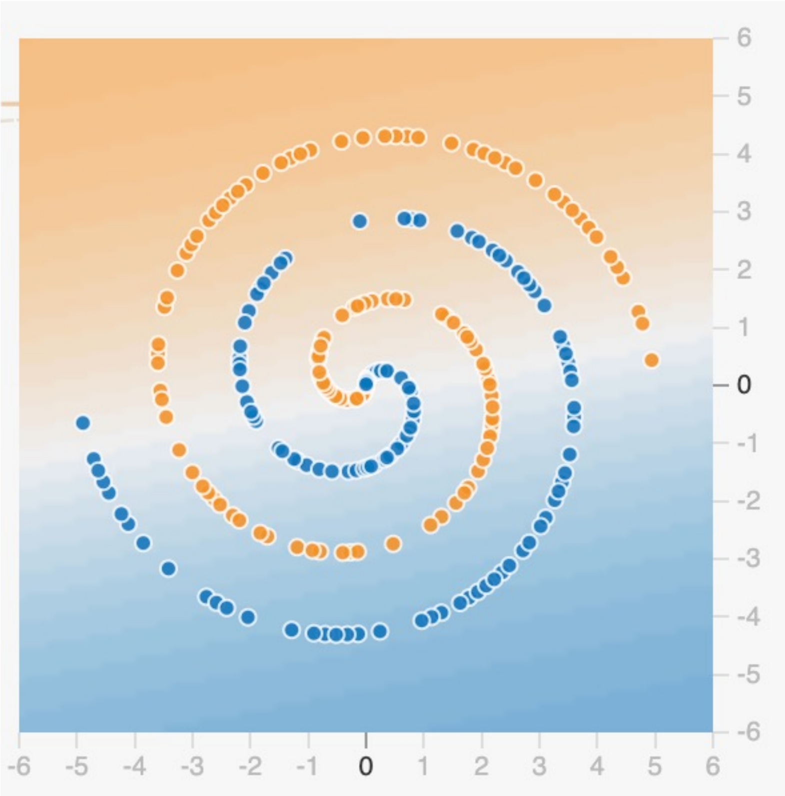
Could we train Logistic Regression on these two training sets?





Training Logistic Regression on these two training sets





Predicting with Logistic Regression

Given $\boldsymbol{x} = [x_1, \quad x_2, \quad x_{3,} \ \dots, x_j]$

Learn weights $\boldsymbol{\beta} = [\beta_1, \quad \beta_2, \quad \beta_{3,} \ \dots, \beta_j]$

Compute a dot product $\boldsymbol{x} * \boldsymbol{\beta}$

Dot product is a linear combination

We need to add some non-linearity

Predicting with Logistic Regression

Is sigmoid enough? Does adding sigmoid allows us to model non-linearly separable data?

https://playground.tensorflow.org/

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

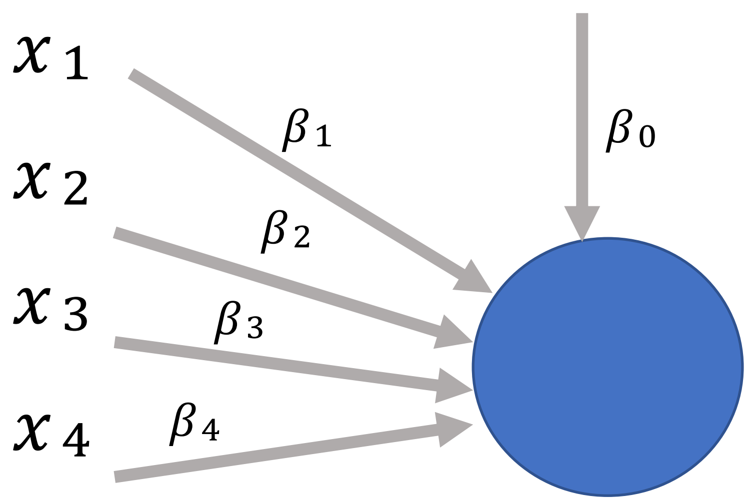
Neural Networks

- Feed forward
- Non-linear Activation Functions
- Computation Graph
- Back-propagation
- Neural LM

Logistic Regression

We make a prediction by taking the doc product of the features (covariates) and weights (coefficients)

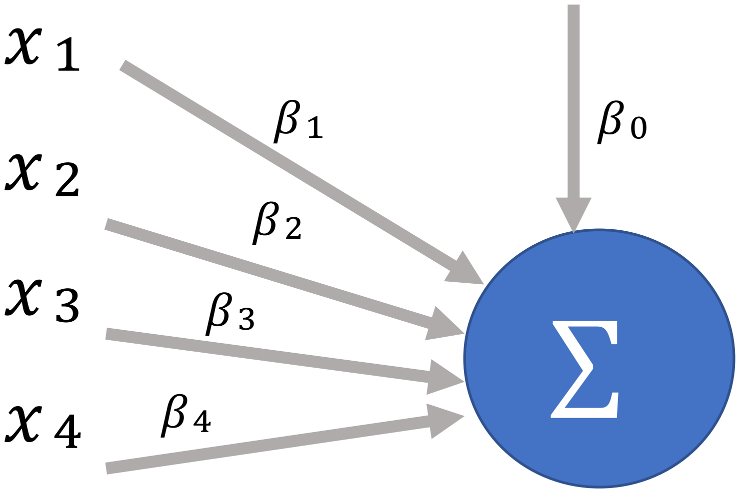
$$\sigma(\boldsymbol{\beta} \cdot \boldsymbol{x}) = \sum_i^j \sigma(\beta_i \cdot x_i)$$



Logistic Regression

We make a prediction by taking the doc product of the features (covariates) and weights (coefficients)

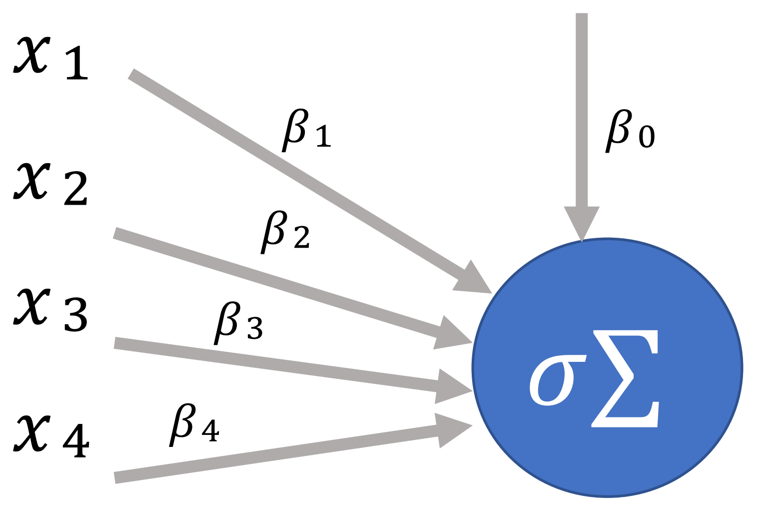
$$\sigma(\boldsymbol{\beta} \cdot \boldsymbol{x}) = \sum_i^j \sigma(\beta_i \cdot x_i)$$



Logistic Regression

We make a prediction by taking the doc product of the features (covariates) and weights (coefficients)

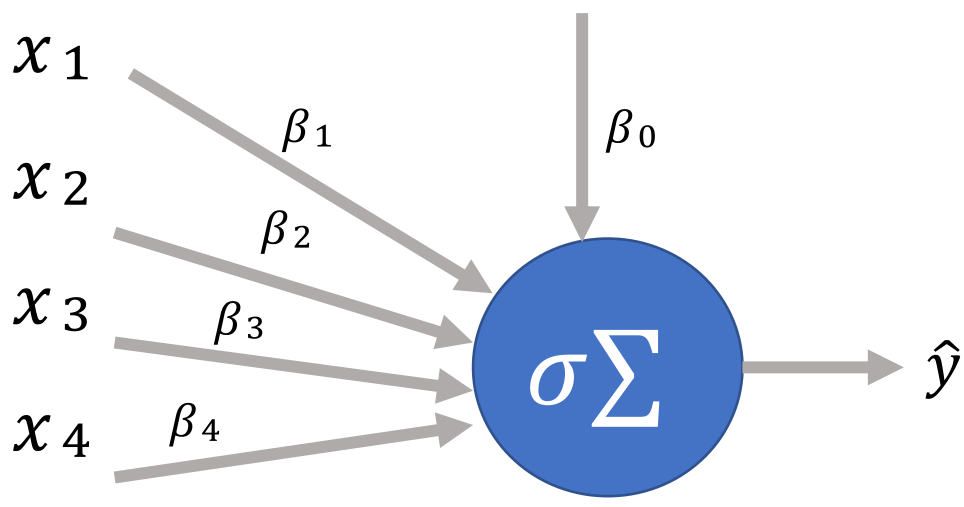
$$\sigma(\boldsymbol{\beta} \cdot \boldsymbol{x}) = \sum_i^j \sigma(\beta_i \cdot x_i)$$



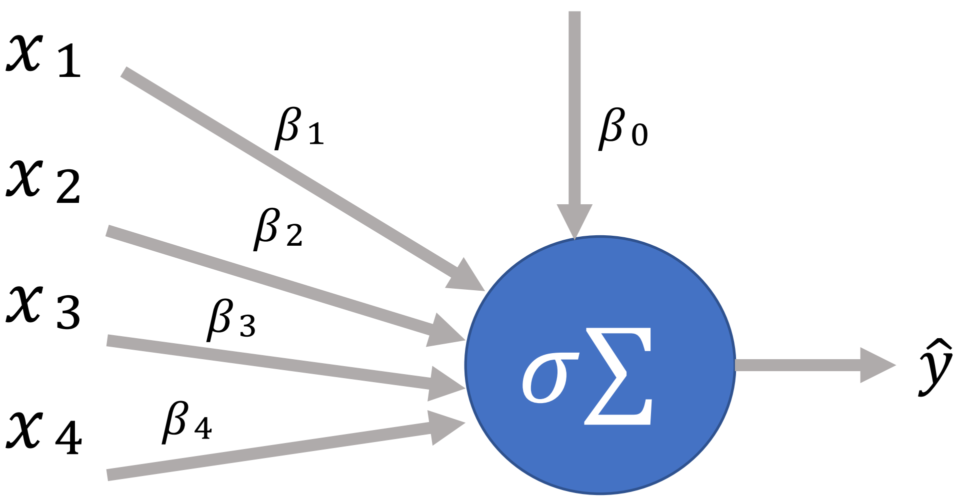
Logistic Regression

We make a prediction by taking the doc product of the features (covariates) and weights (coefficients)

$$\sigma(\boldsymbol{\beta} \cdot \boldsymbol{x}) = \sum_i^j \sigma(\beta_i \cdot x_i)$$



A neuron



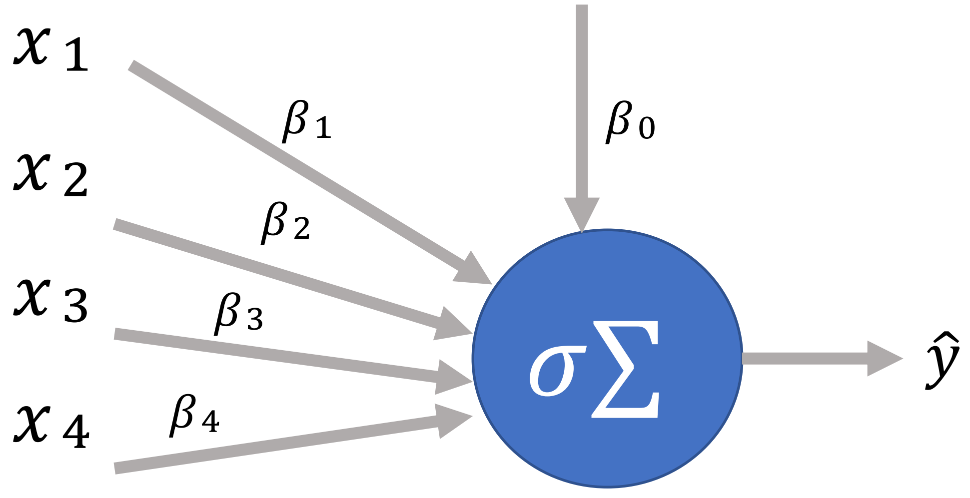
Logistic Regression as NN

A single layer neural network

Input layer: features

Output layer: prediction

We can pass the output of the neuron to another neuron



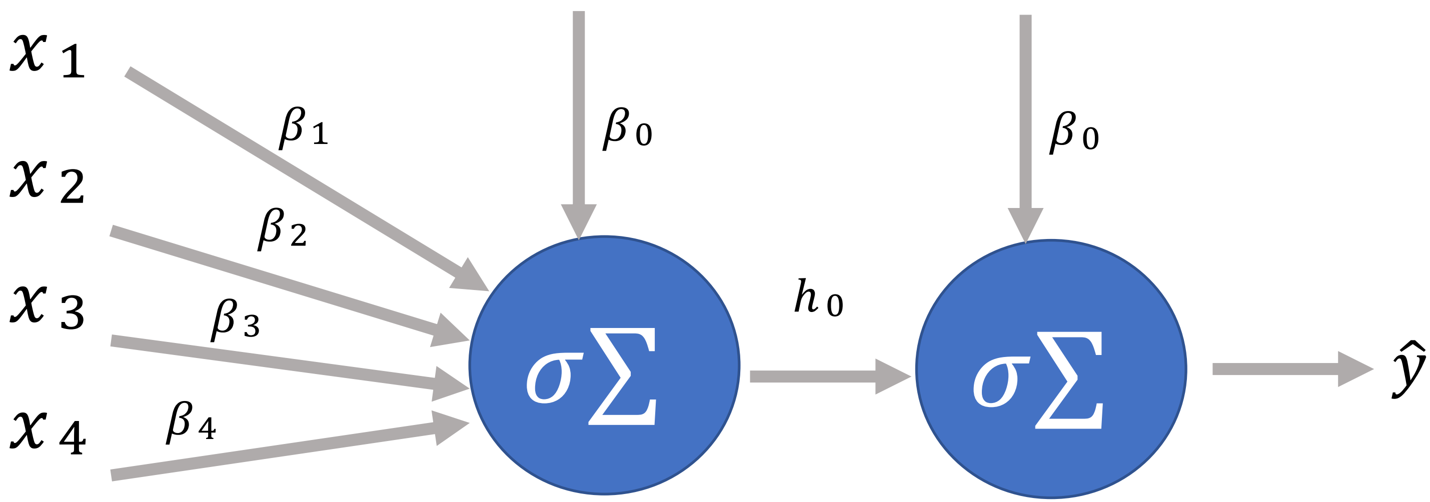
A two layered network

Input layer: features

Output layer: prediction

Hidden layer: h_0

We can add more hidden layers and more neurons at each layer https://playground.tensorflow.org/



Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- **Feed forward**
- Non-linear Activation Functions
- Computation Graph
- Back-propagation

Feed forward NN

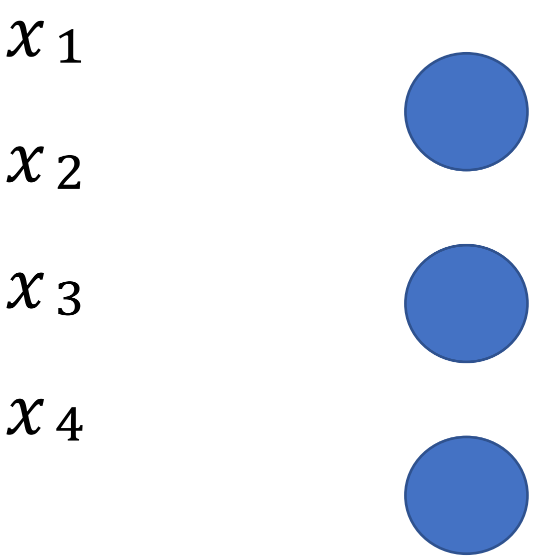
x_1

x_2

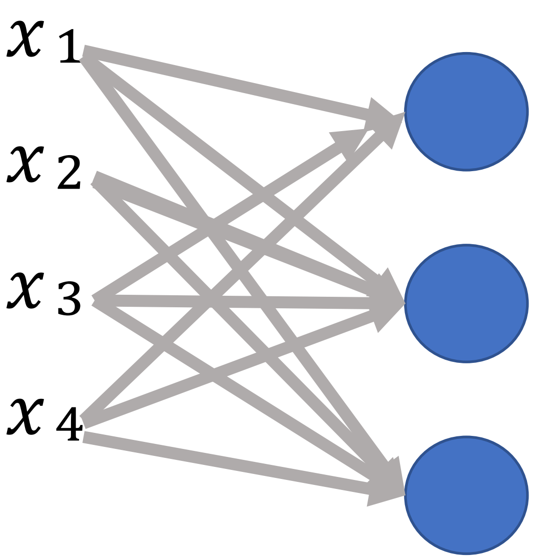
x_3

x_4

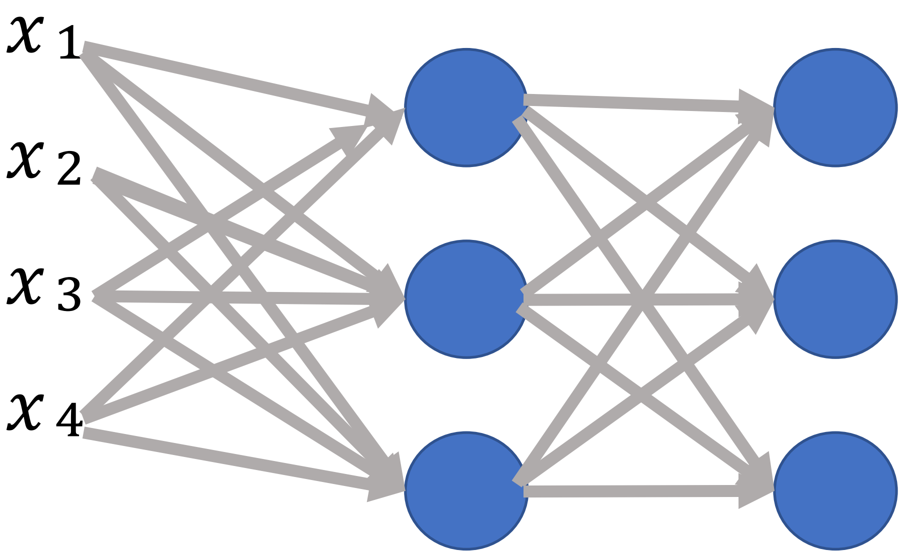
Feed forward NN



Feed forward NN



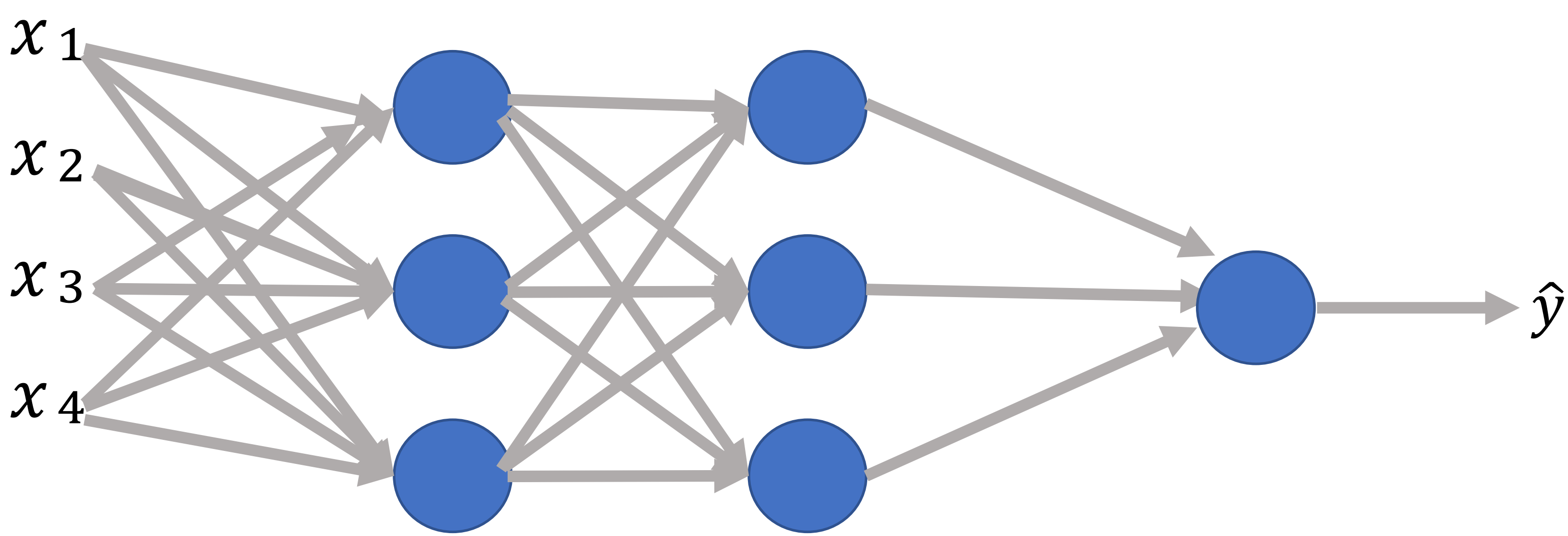
Feed forward NN



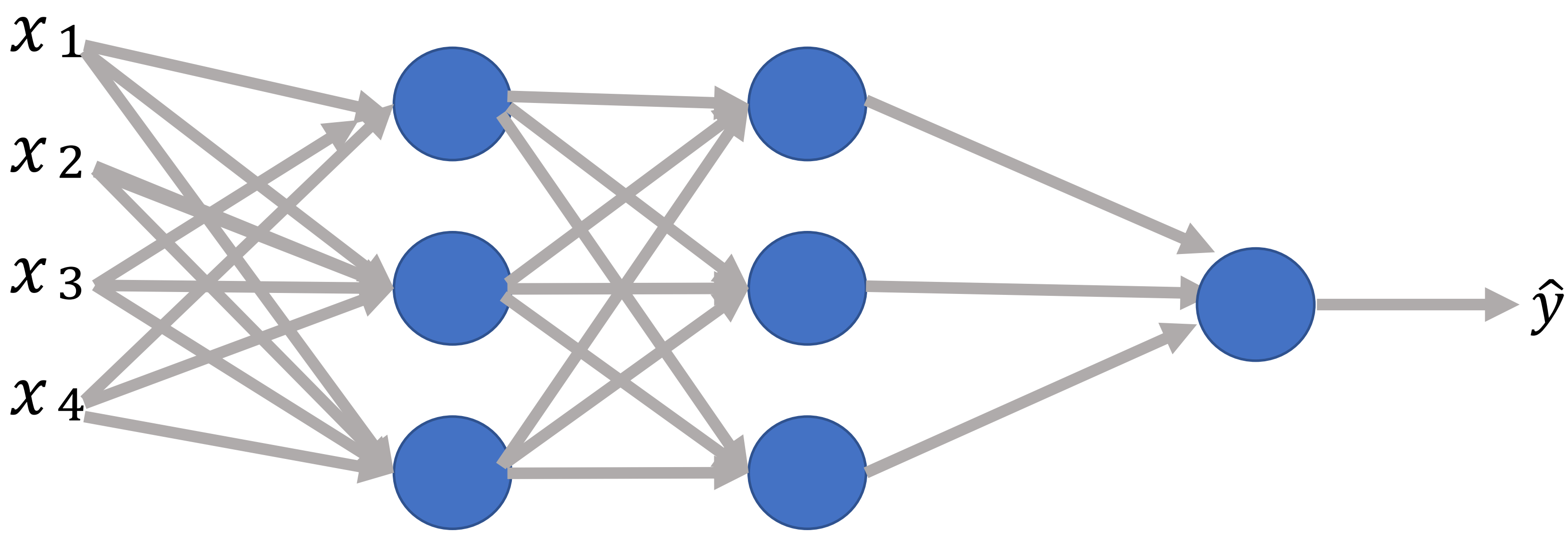
Feed forward NN

All nodes in between each layer is are connected

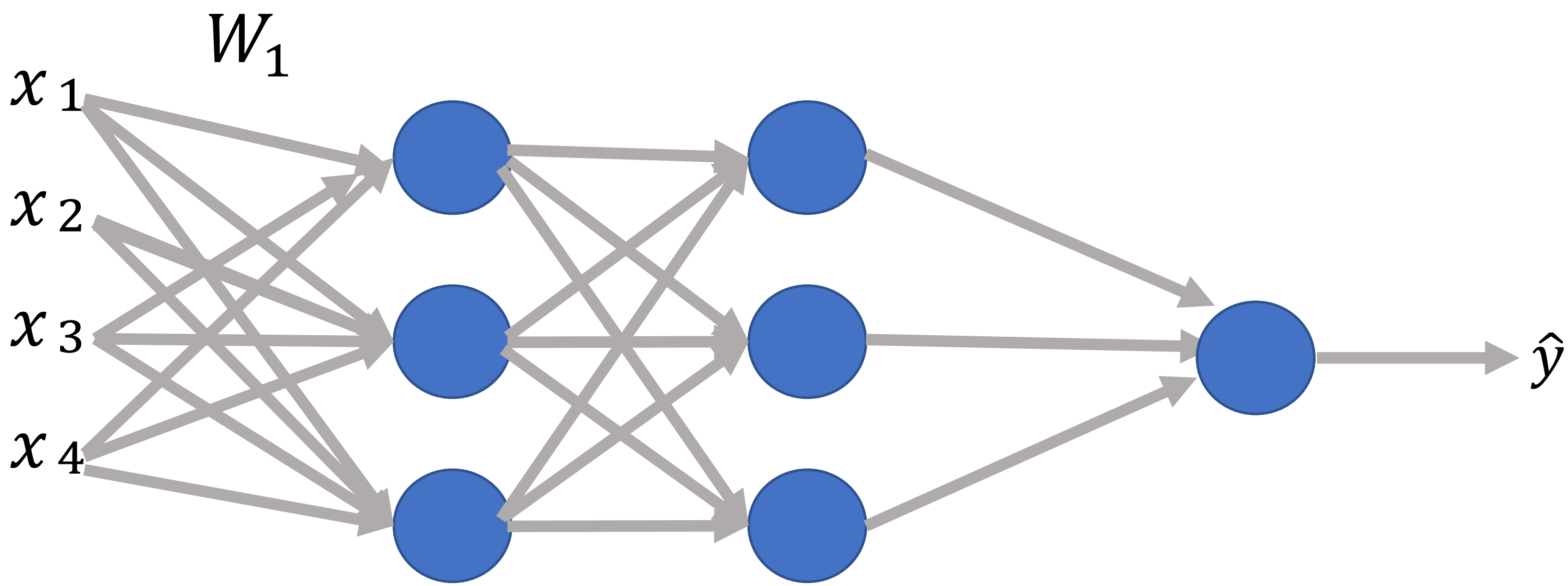
Input layer: features, Output layer: prediction, 2 Hidden layers



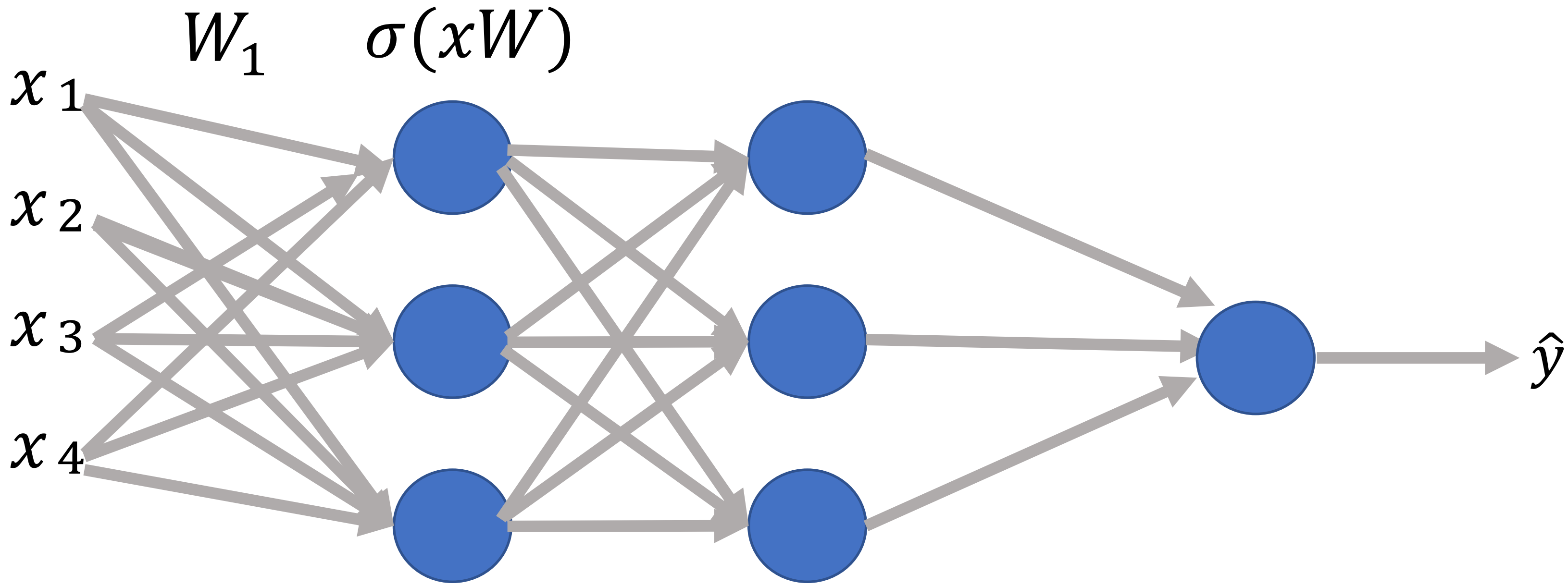
Feed forward NN
Making predictions



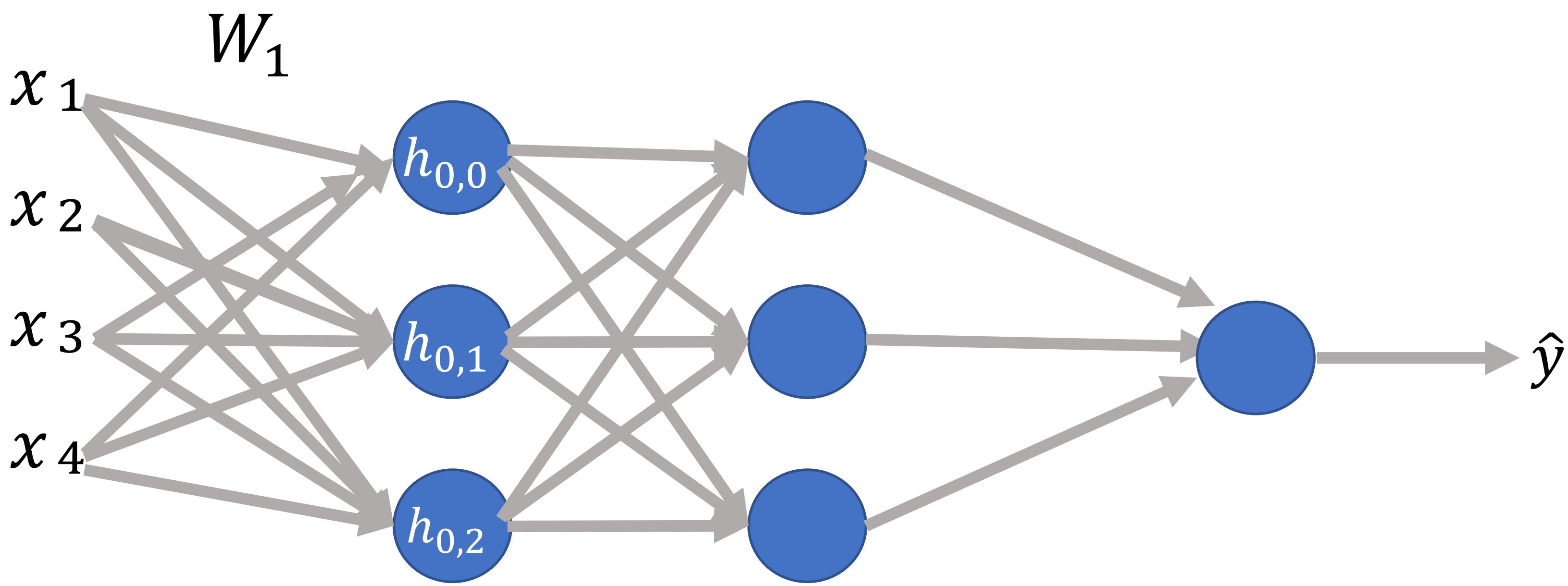
Feed forward NN
Making predictions



Feed forward NN
Making predictions

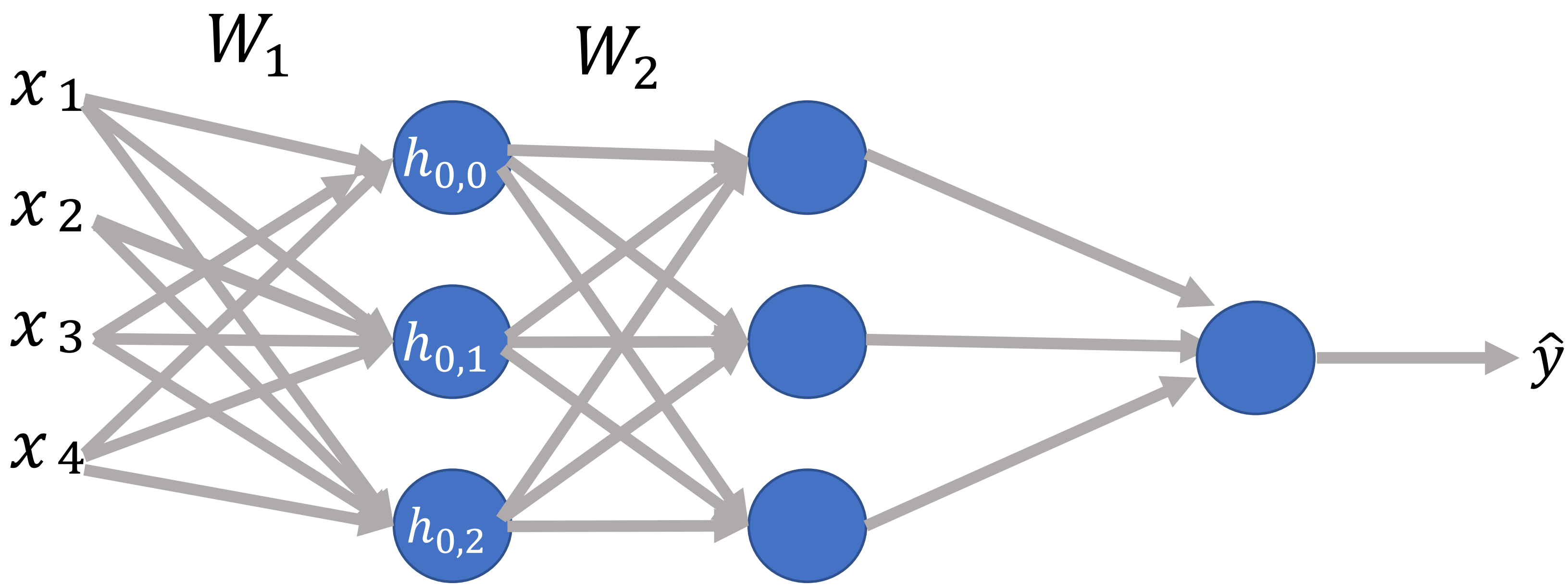


Feed forward NN
Making predictions



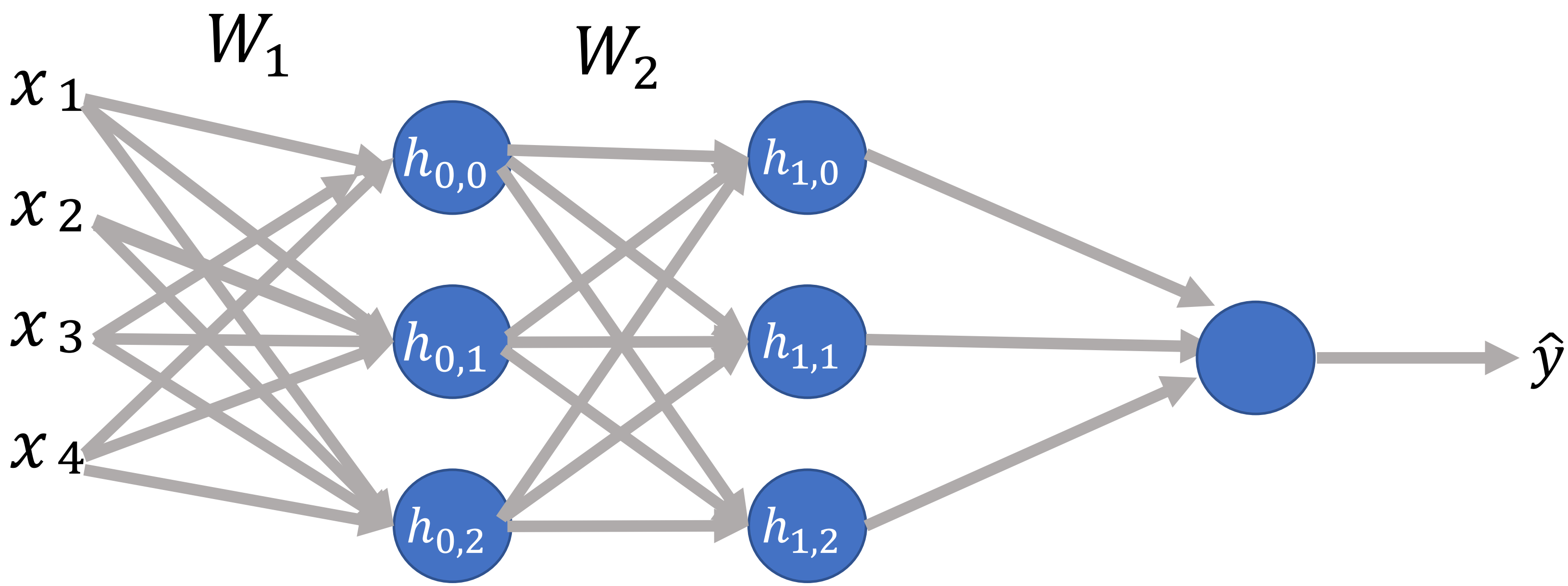
$$\boldsymbol{h}_0 = \sigma(xW_1)$$

Feed forward NN
Making predictions



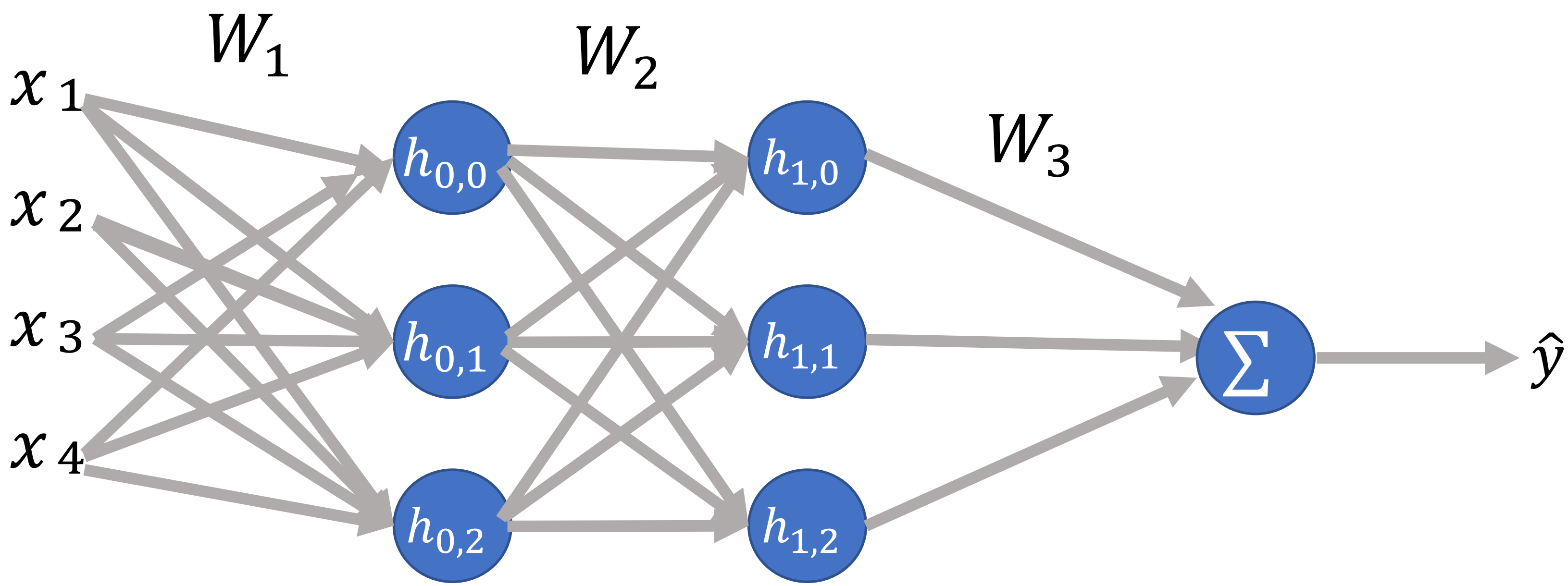
$$\boldsymbol{h}_0 = \sigma(xW_1)$$

Feed forward NN
Making predictions



$$\boldsymbol{h}_0 = \sigma(xW_1)$$

Feed forward NN
Making predictions



$$\boldsymbol{h}_0 = \sigma(xW_1)$$

Outline

https://playground.tensorflow.org/

SGD example

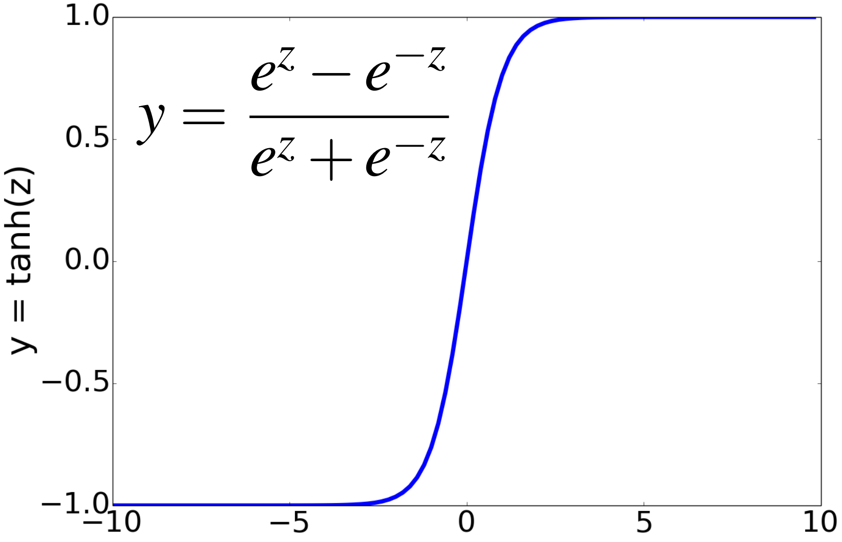
Beyond Binary Classification

Implementation Tricks

Neural Networks

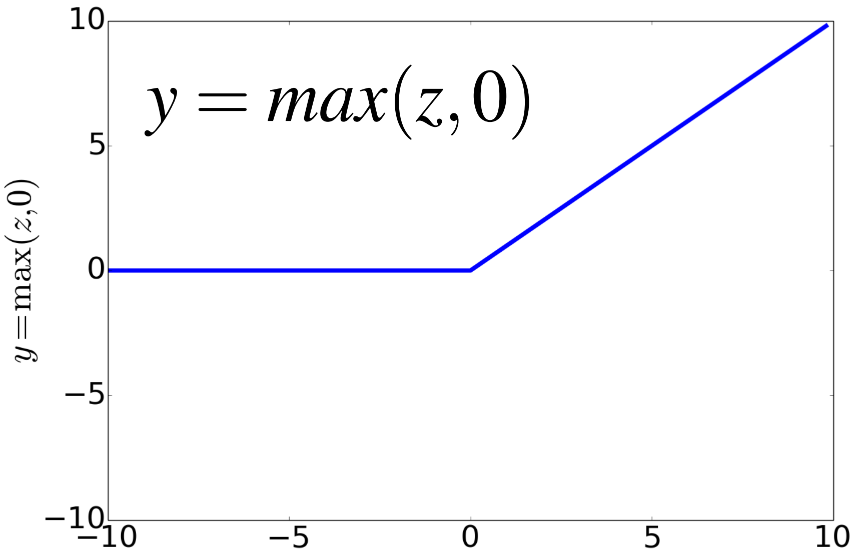
- Feed forward
- **Non-linear Activation Functions**
- Computation Graph
- Back-propagation

Non-Linear Activation Functions besides sigmoid



tanh

Most Common:



ReLU
Rectified Linear Unit

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

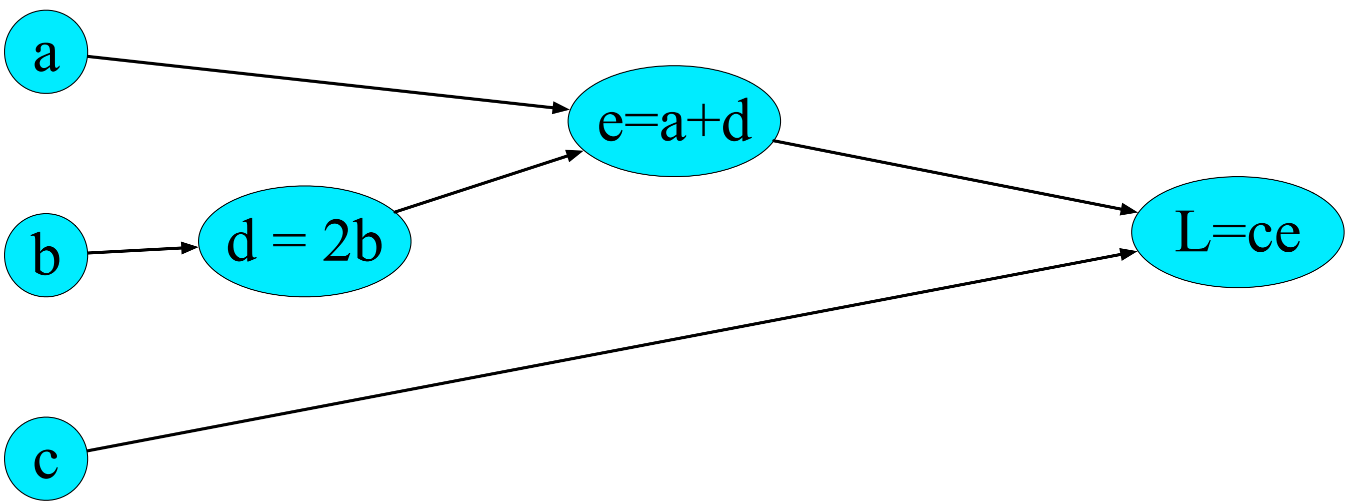
Neural Networks
- Feed forward
- Non-linear Activation Functions
- **Computation Graph**
- Back-propagation

Computation graph

$$L(a,b,c) = c(a+2b)$$

Computations:

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

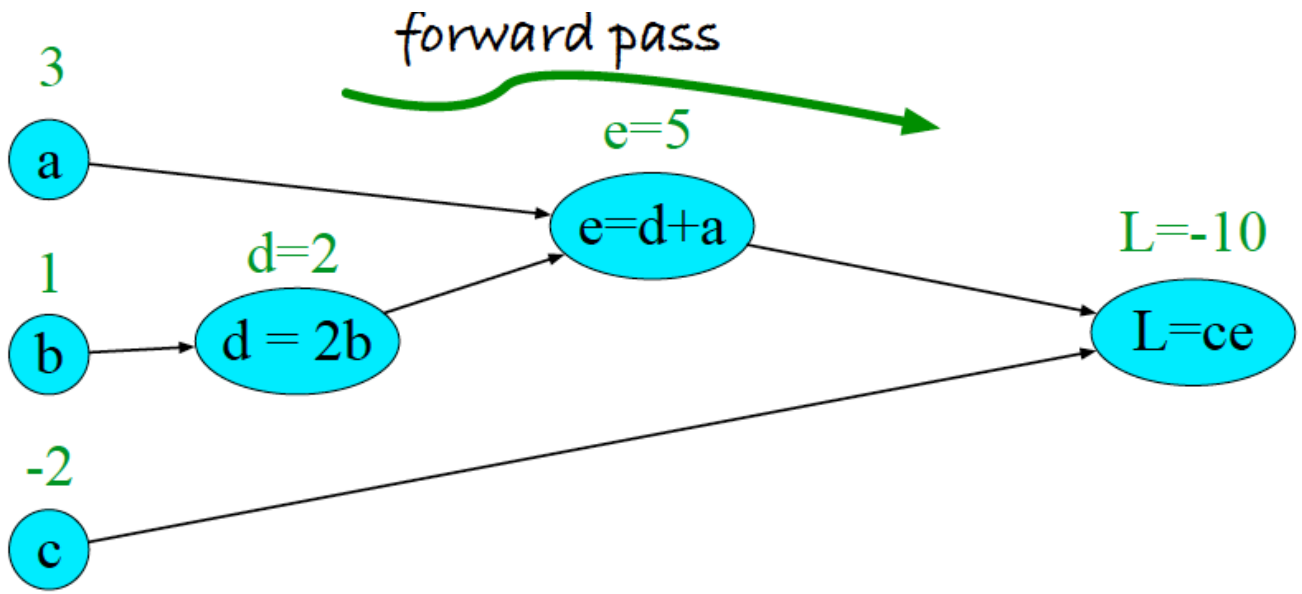


Example:

$$L(a,b,c) = c(a+2b)$$

Computations:

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks

- Feed forward
- Non-linear Activation Functions
- Computation Graph
- **Back-propagation**
- Neural LM

Backwards differentiation in computation graphs

- The importance of the computation graph comes from the backward pass
- This is used to compute the derivatives that we'll need for the weight update.

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

We want: $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$

The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in *a* affects *L*.

The chain rule

- Computing the derivative of a composite function:

- $f(x) = u(v(x))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

- $f(x) = u(v(w(x)))$ $$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial c} &= e \\ \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

$$L = ce \ :$$

$$e = a+d \ :$$

$$d = 2b \ :$$

Example

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$L(a,b,c) = c(a+2b)$$

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$

$L = ce$: $\frac{\partial L}{\partial e} =$ $\frac{\partial L}{\partial c} =$

$e = a+d$:

$d = 2b$:

Example

$$
\begin{aligned}
\frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}
\end{aligned}
$$

$$L(a,b,c) = c(a+2b)$$

$$
\begin{aligned}
d &= 2*b \\
e &= a+d \\
L &= c*e
\end{aligned}
$$

$$L = ce \;:\quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a+d \;:$$

$$d = 2b \;:$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

$$L = ce \ : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a+d \ : \quad \frac{\partial e}{\partial a} = \quad \frac{\partial e}{\partial d} =$$

$$d = 2b \ :$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

$$\begin{aligned} L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\ e = a+d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = \\ d = 2b &: \quad \frac{\partial d}{\partial b} = \end{aligned}$$

Example

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

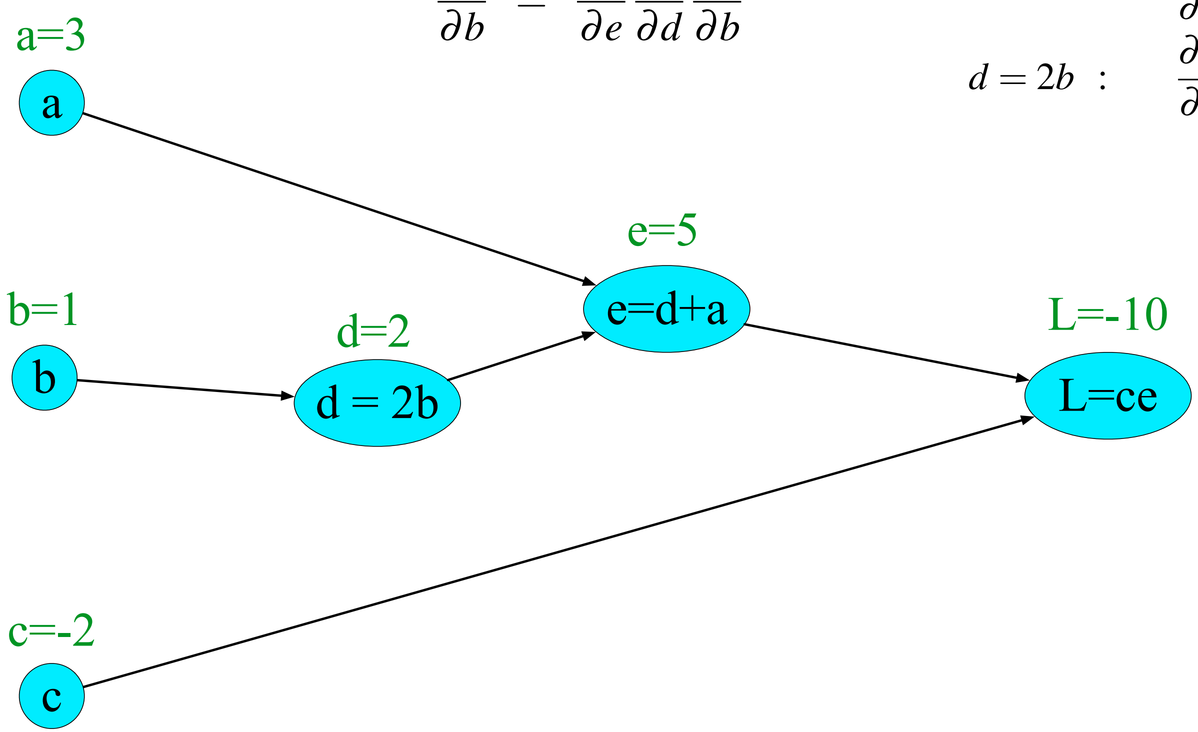
$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} \end{aligned}$$

$$\begin{aligned} L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\ e = a+d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = \\ d = 2b &: \quad \frac{\partial d}{\partial b} = 2 \end{aligned}$$

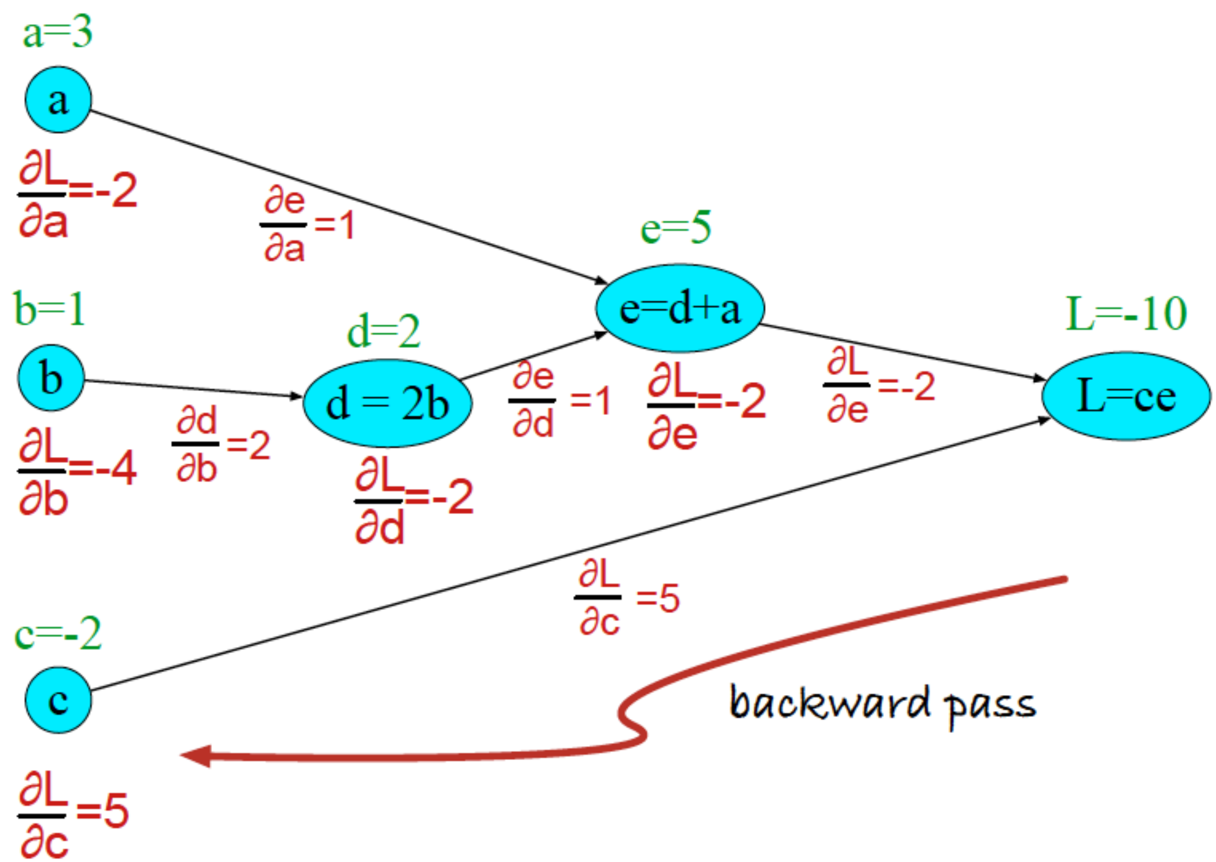
Example

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$L = ce \;:\; \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$
$$e = a + d \;:\; \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$
$$d = 2b \;:\; \frac{\partial d}{\partial b} = 2$$



Example



a=3
a
$\frac{\partial L}{\partial a}$=-2
$\frac{\partial e}{\partial a}$=1
e=5
e=d+a
b=1
b
d=2
d = 2b
$\frac{\partial e}{\partial d}$ =1
$\frac{\partial L}{\partial e}$=-2
$\frac{\partial L}{\partial e}$ =-2
L=-10
L=ce
$\frac{\partial L}{\partial b}$=-4
$\frac{\partial d}{\partial b}$=2
$\frac{\partial L}{\partial d}$=-2
$\frac{\partial L}{\partial c}$ =5
c=-2
c
backward pass
$\frac{\partial L}{\partial c}$=5

Detailed View

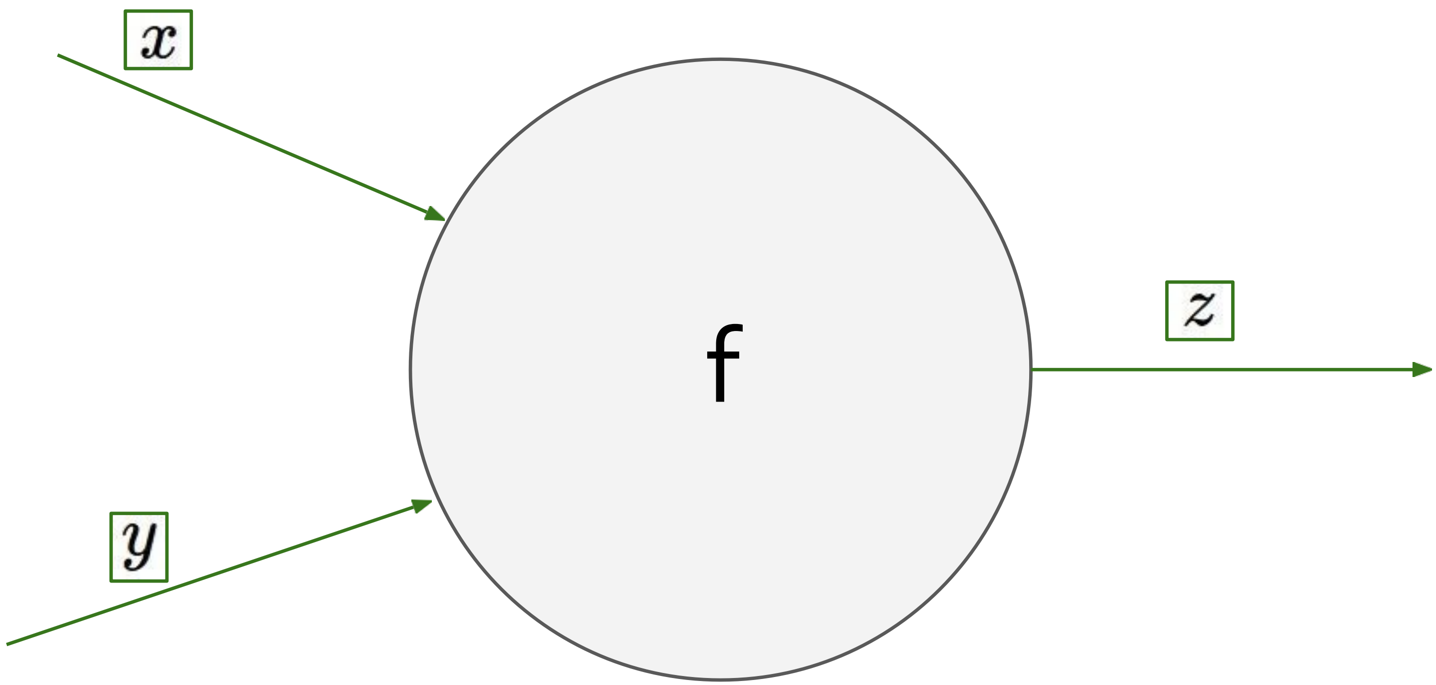


Figure from Andrej Karpathy

Detailed View

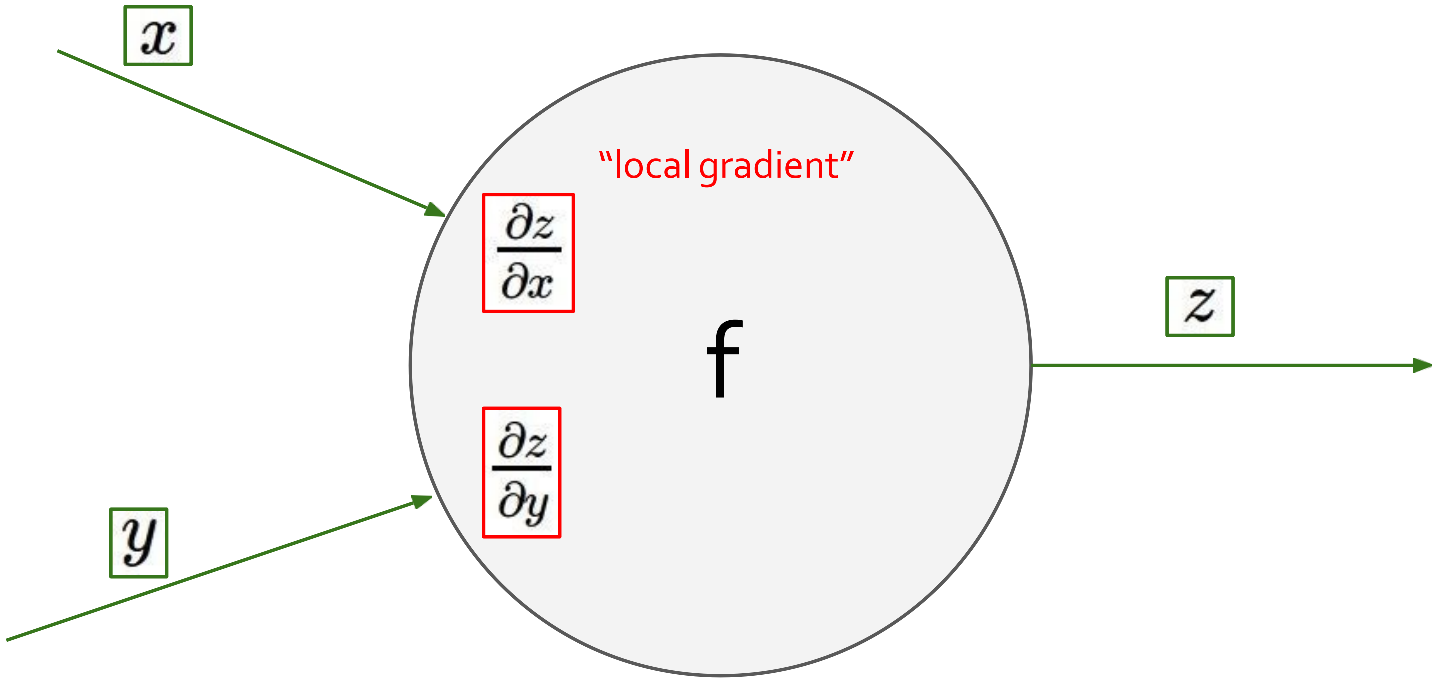


Figure from Andrej Karpathy

Detailed View

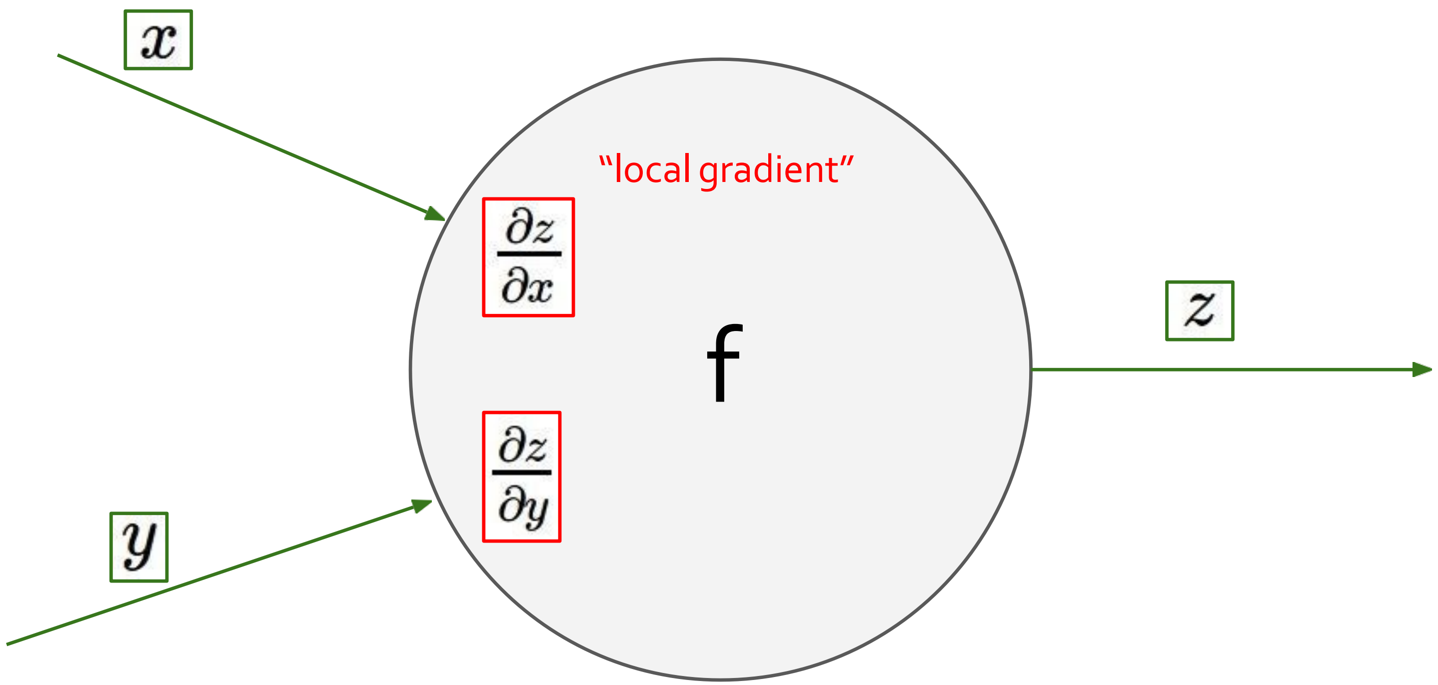


Figure from Andrej Karpathy

Detailed View

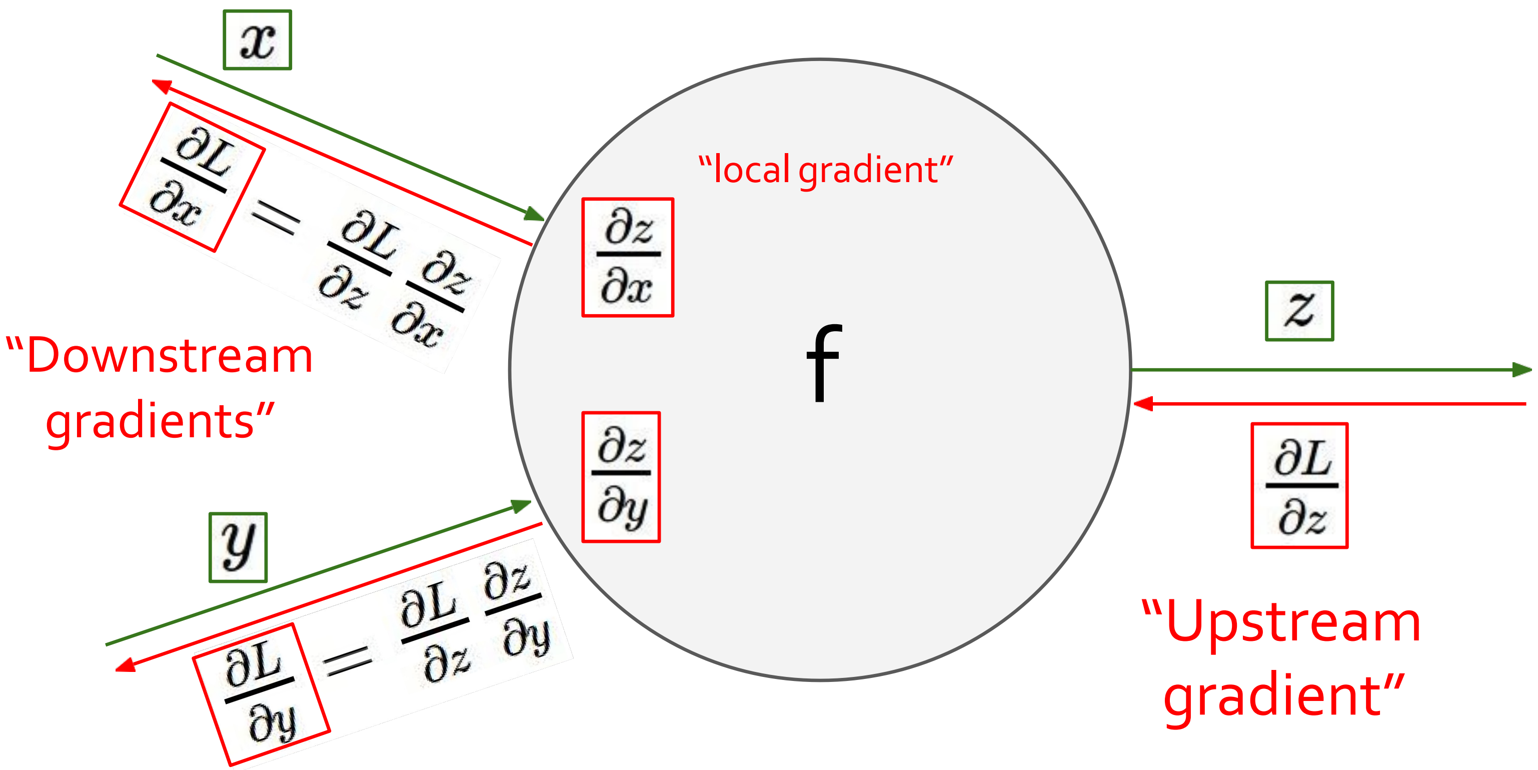


Figure from Andrej Karpathy

Detailed View

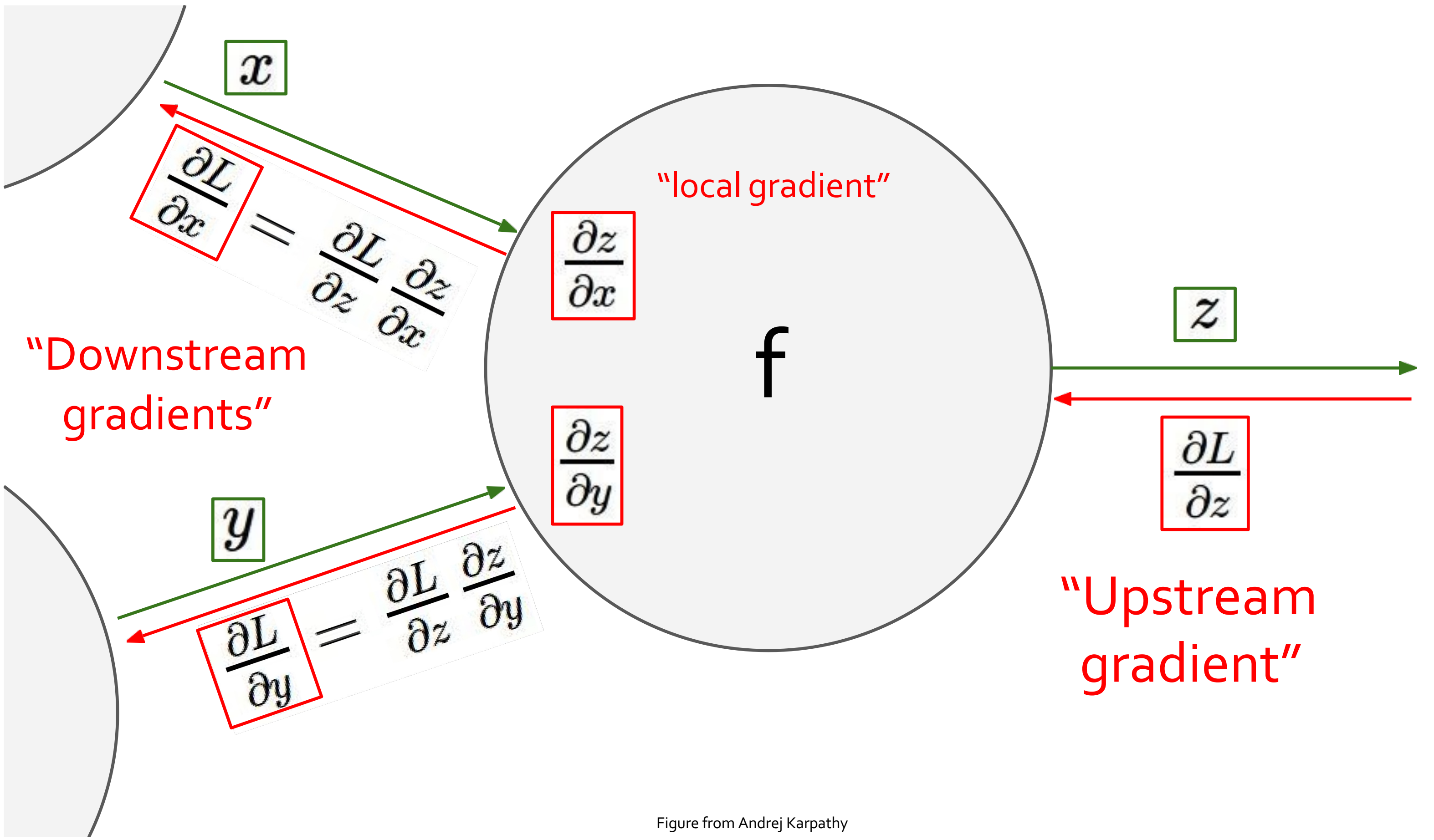
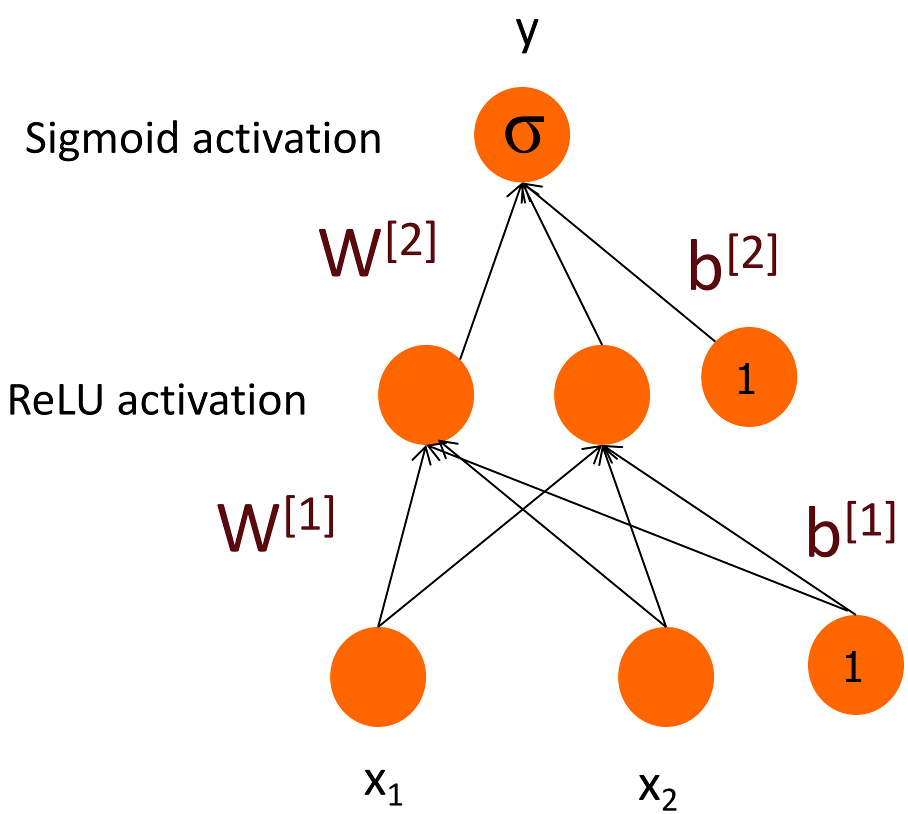


Figure from Andrej Karpathy

Backward differentiation on a two layer network



$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

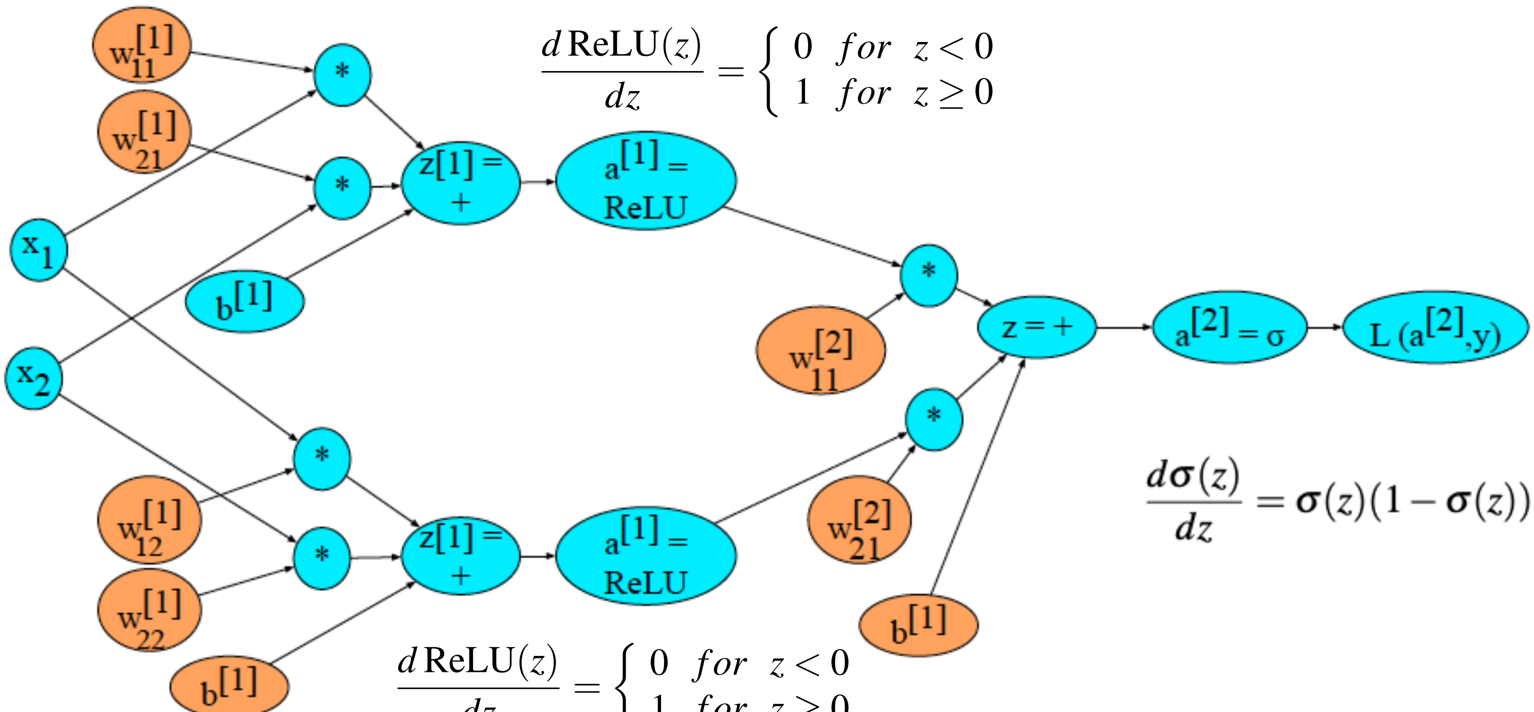
Backward differentiation on a two layer network

$$
\begin{aligned}
z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= \sigma(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

$$\frac{d\,\mathrm{ReLU}(z)}{dz} = \begin{cases} 0 & for \quad z < 0 \\ 1 & for \quad z \geq 0 \end{cases}$$

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

Backward differentiation on a 2-layer network



$$\frac{d\,\mathrm{ReLU}(z)}{dz} = \begin{cases} 0 & for \quad z < 0 \\ 1 & for \quad z \geq 0 \end{cases}$$

$$\frac{d\boldsymbol{\sigma}(z)}{dz} = \boldsymbol{\sigma}(z)(1 - \boldsymbol{\sigma}(z))$$

$$\frac{d\,\mathrm{ReLU}(z)}{dz} = \begin{cases} 0 & for \quad z < 0 \\ 1 & for \quad z \geq 0 \end{cases}$$

Starting off the backward pass: $\dfrac{\partial L}{\partial z}$

$$\begin{aligned} z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\ a^{[1]} &= \mathrm{ReLU}(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned}$$

(I'll write a for $a^{[2]}$ and z for $z^{[2]}$)

$$L(\hat{y}, y) = -(\mathrm{y}\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

$$L(a, y) = -(\mathrm{y}\log a + (1-y)\log(1-a))$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}$$

$$\begin{aligned} \frac{\partial L}{\partial a} &= -\left(\left(y\frac{\partial \log(a)}{\partial a}\right) + (1-\mathrm{y})\ \frac{\partial \log(1-a)}{\partial a}\right) \\ &= -\left(\left(y\frac{1}{a}\right) + (1-\mathrm{y})\ \frac{1}{1-a}(-1)\right) = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right) \end{aligned}$$

$$\frac{\partial a}{\partial z} = a(1-a) \qquad \frac{\partial L}{\partial z} = -\left(\frac{y}{a} + \frac{y-1}{1-a}\right)a(1-a) = \mathrm{a} - \mathrm{y}$$

Summary

- For training, we need the derivative of the loss with respect to weights in early layers of the network
- But loss is computed only at the very end of the network!
- Solution: **backward differentiation**
- Given a computation graph and the derivatives of all the functions in it we can automatically compute the derivative of the loss with respect to these early weights.

Outline

SGD example

Beyond Binary Classification

Implementation Tricks

Neural Networks
- Feed forward
- Computation Graph
- Back-propagation
- **Neural LM**

Neural Language Models (LMs)

- **Language Modeling**: Calculating the probability of the next word in a sequence given some history.
- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models

•State-of-the-art neural LMs are based on more powerful neural network technology like Transformers

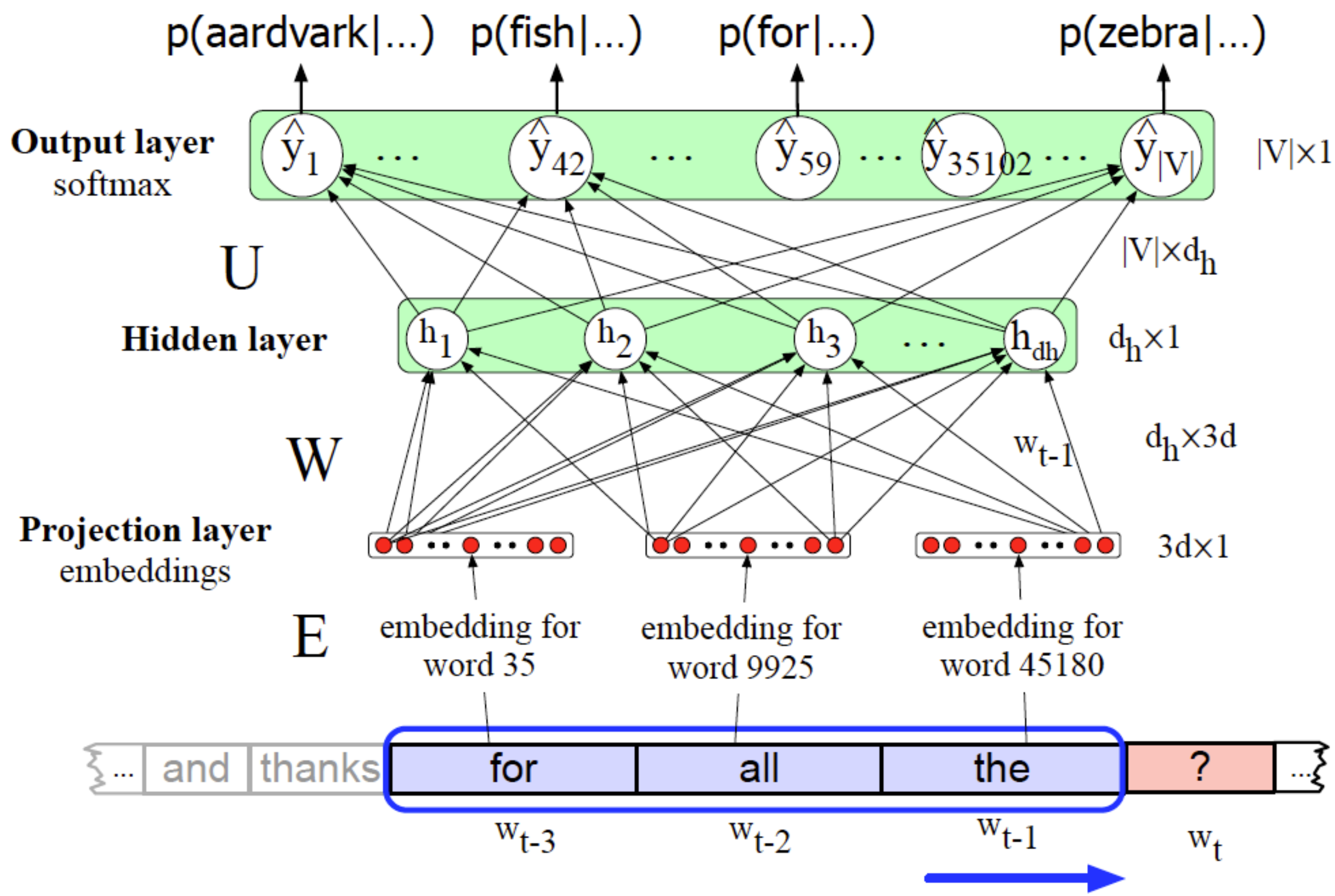
•But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

- **Task**: predict next word w_t
  given prior words w_{t-1}, w_{t-2}, w_{t-3}, …
- **Problem**: Now we're dealing with sequences of arbitrary length.
- **Solution**: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



Masked Language Model

- **Task**: predict w_t

  given prior surrounding words

  $w_{t-3}, w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}, w_{t+3}$

- **Problem**: Now we're dealing with sequences of arbitrary length.
- **Solution**: Sliding windows (of fixed length)

Why Neural LMs work better than N-gram LMs

- **Training data:**
- We've seen: I have to make sure that the cat gets fed.
- Never seen: dog gets fed
- **Test data:**
- I forgot to make sure that the dog gets ___
- N-gram LM can't predict "fed"!
- Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Summary – Feed Forward Networks

Every node in one layer is connected to every node in the next layer

Define the Network

- Number of hidden layers
- Size of each hidden layer
- Activation Function

Forward pass:

- Matrix multiplications

Backward pass:

- Compute the gradients and propagate them backwards – Backpropagation

Issues when training Neural Networks

Exploding/vanishing gradients

Overfitting

FFN’s issues

Fixed input size

Solutions:

1. Create a fixed length representation
2. Recurrent Neural Networks