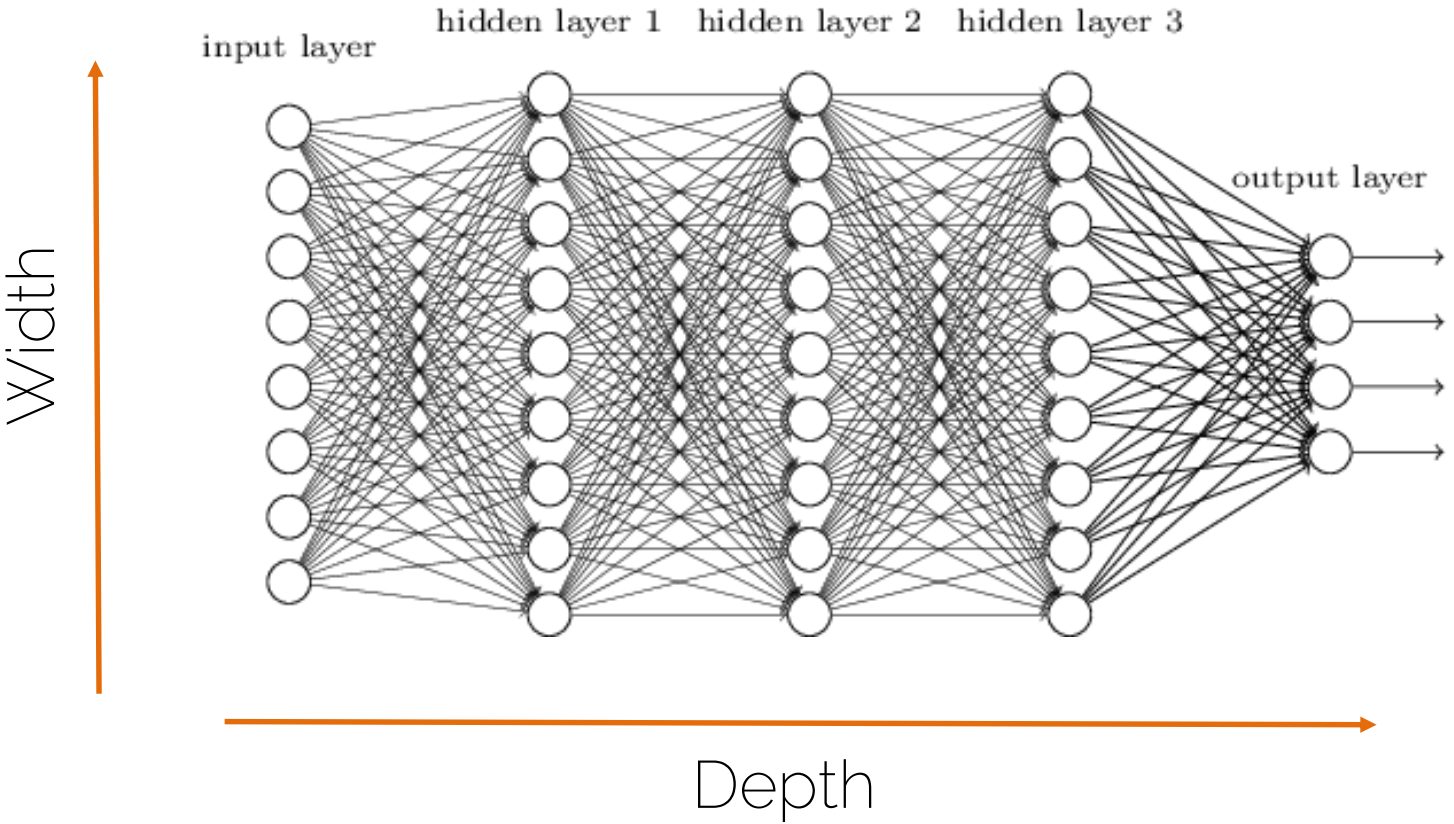
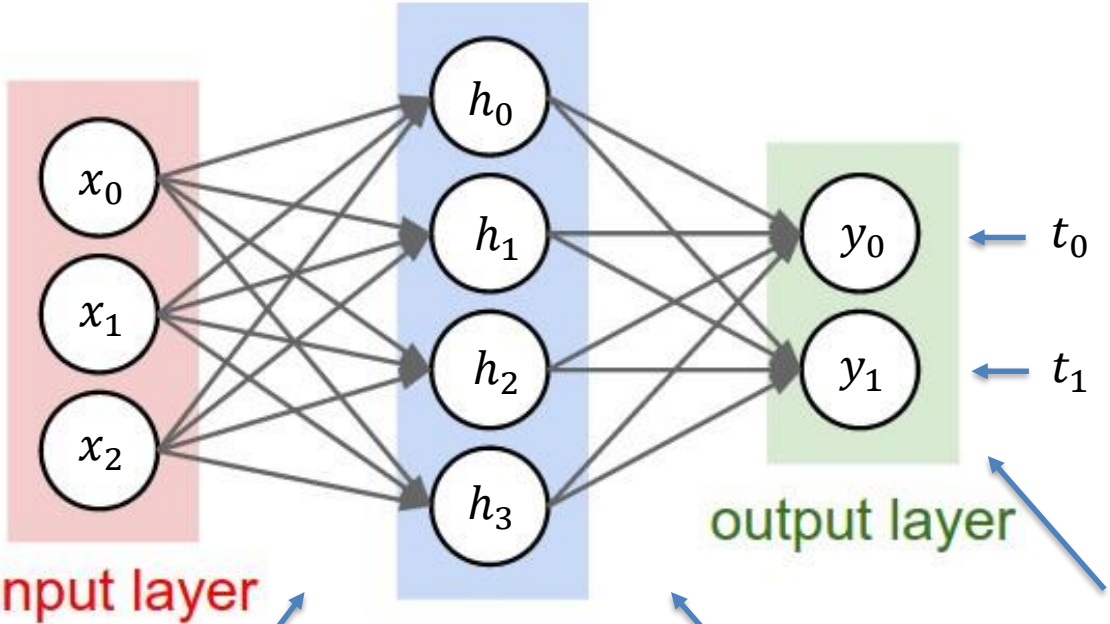
Lecture 6 recap

Neural Network



Gradient Descent for Neural Networks



$$\nabla_{w,b} f_{\{x,t\}}(w) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \dots \\ \dots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \dots \\ \dots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

$$L_i = (y_i - t_i)^2$$

$$h_j = A\left(b_{0,j} + \sum_k x_k w_{0,j,k}\right)$$

$$y_i = A\left(b_{1,i} + \sum_j h_j w_{1,i,j}\right)$$

Just simple: $A(x) = \max(0, x)$

Stochastic Gradient Descent (SGD)

$$\theta^{k+1} = \theta^{k} - \alpha \nabla_\theta L(\theta^{k}, x_{\{1..m\}}, y_{\{1..m\}})$$

$$\nabla_\theta L = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L_i$$

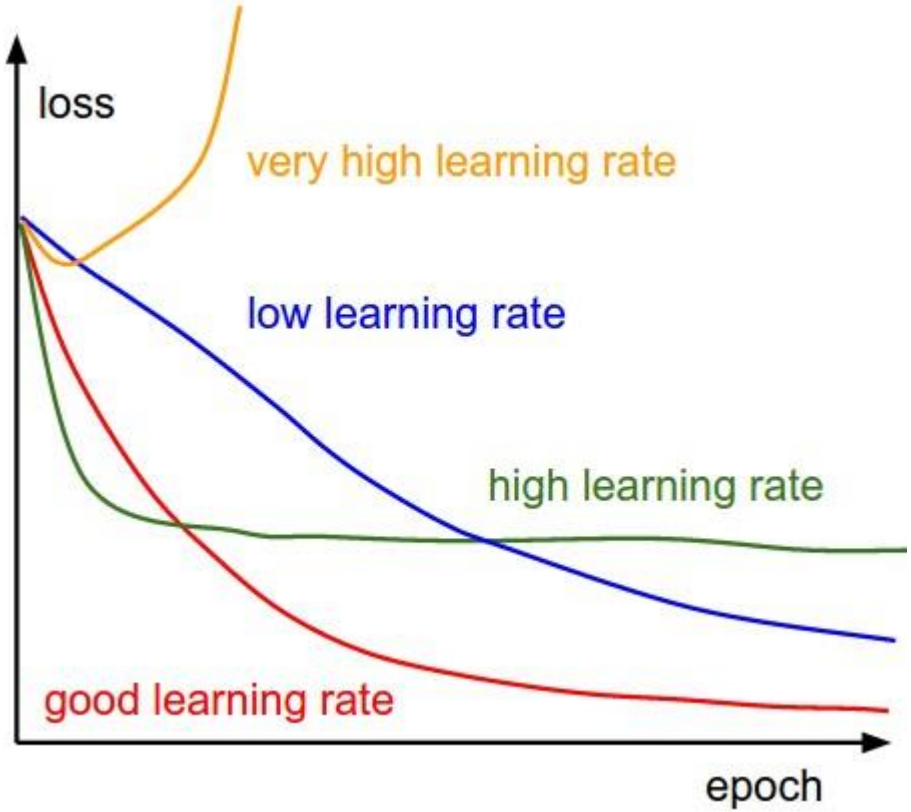
k now refers to k-th iteration

m training samples in the current batch

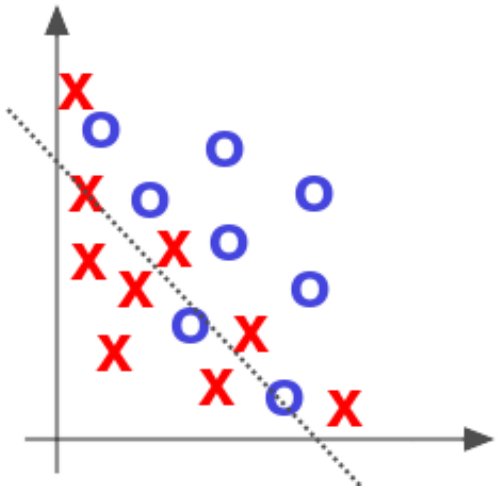
Gradient for the k-th batch

+ all variations of SGD: momentum, RMSProp, Adam, …

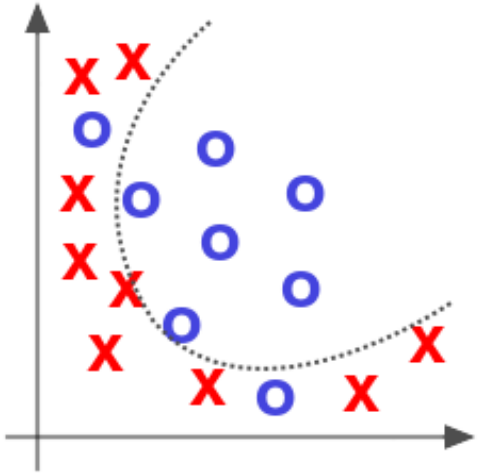
Importance of Learning Rate



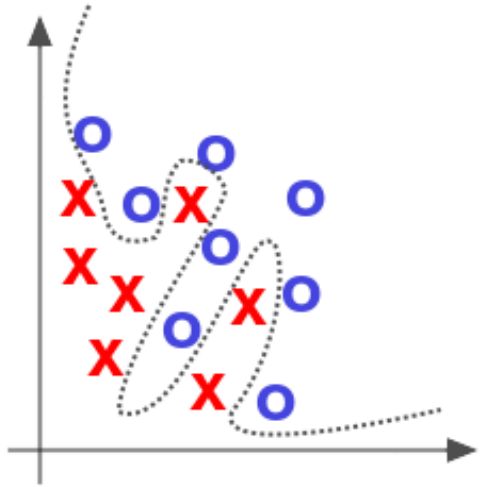
Over- and Underfitting



Underfitted



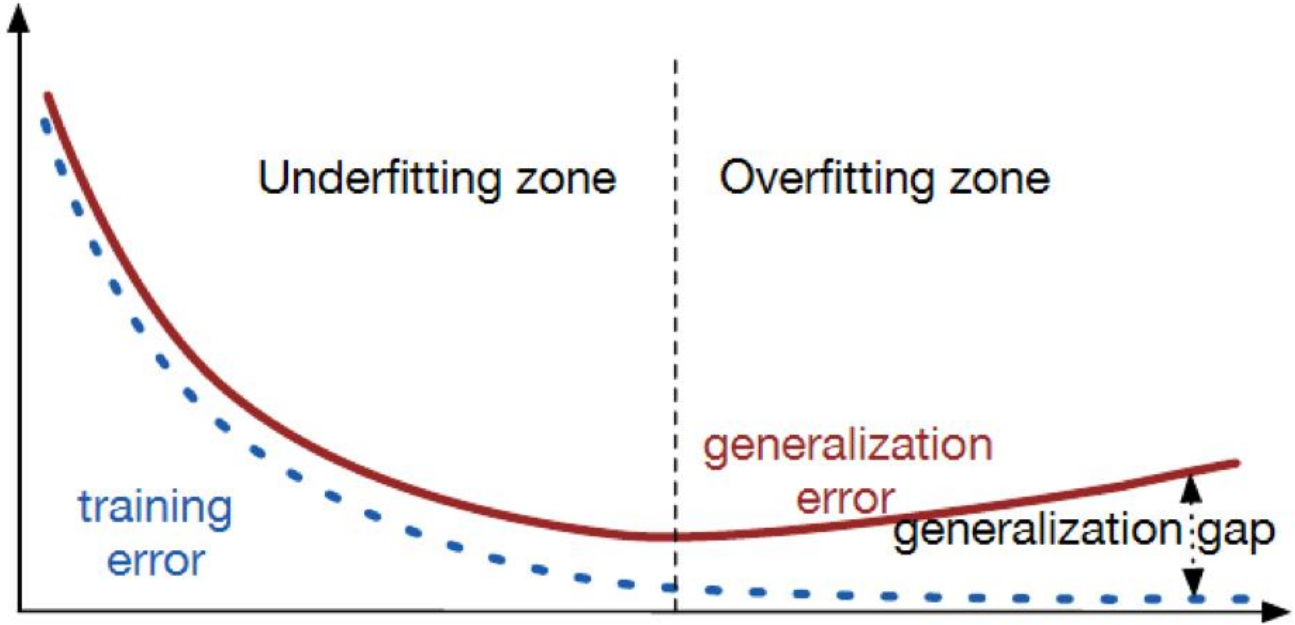
Appropriate



Overfitted

Figure extracted from Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

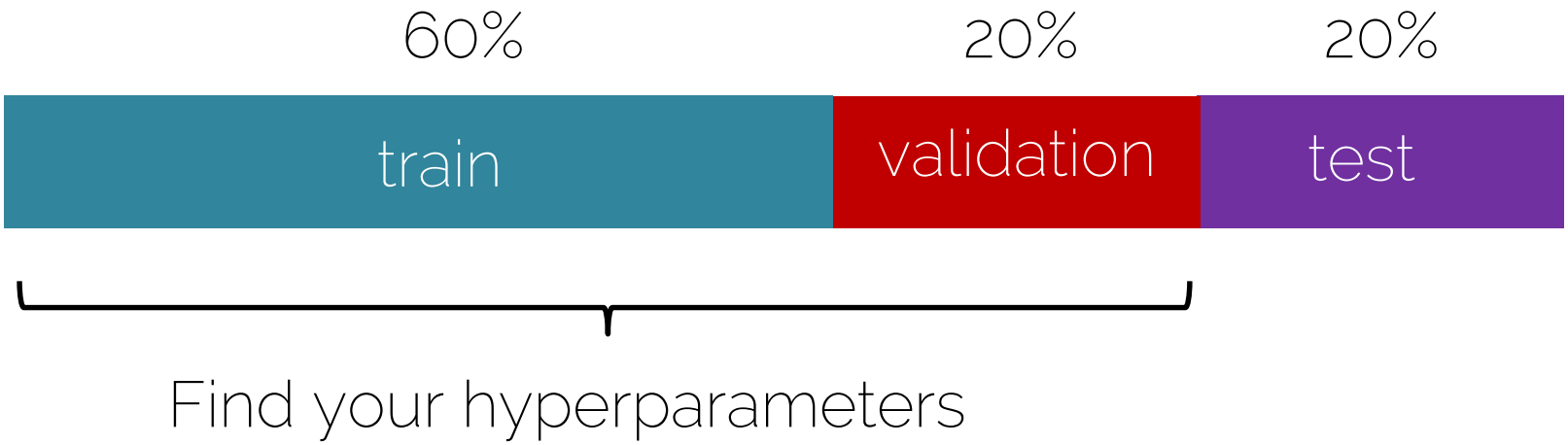
Over- and Underfitting



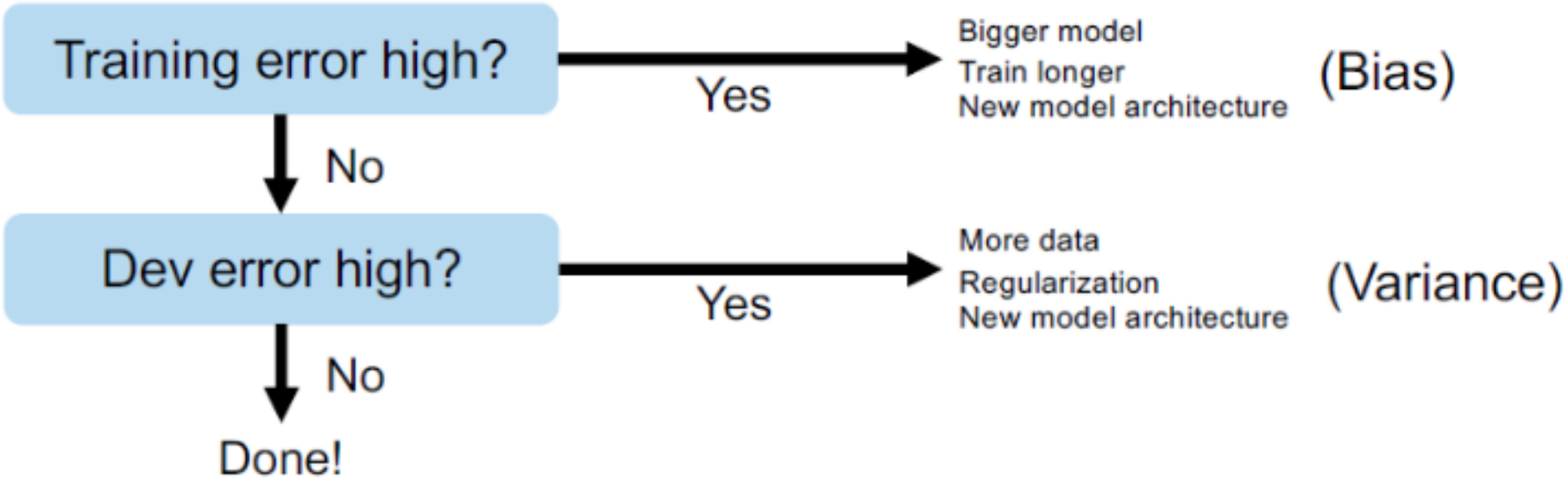
Source: http://srdas.github.io/DLBook/ImprovingModelGeneralization.html

Basic recipe for machine learning

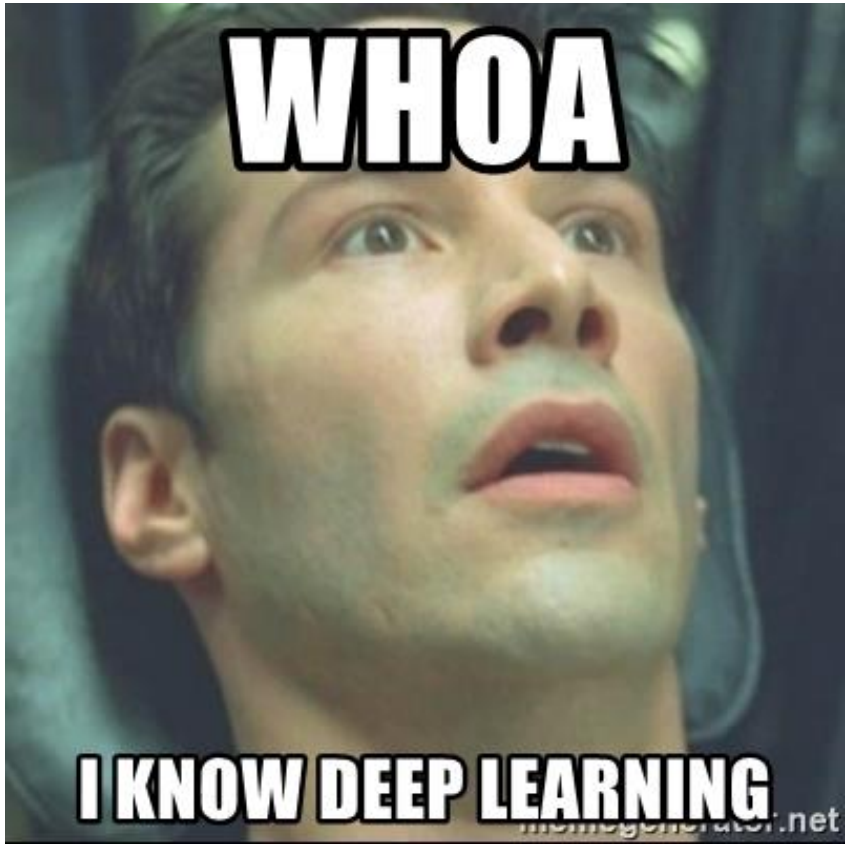
- Split your data



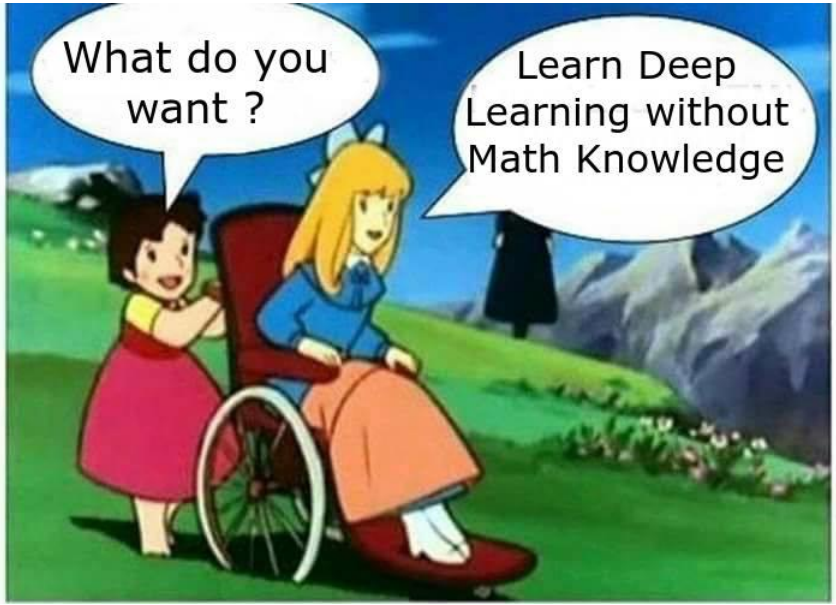
Basic recipe for machine learning



Basically…

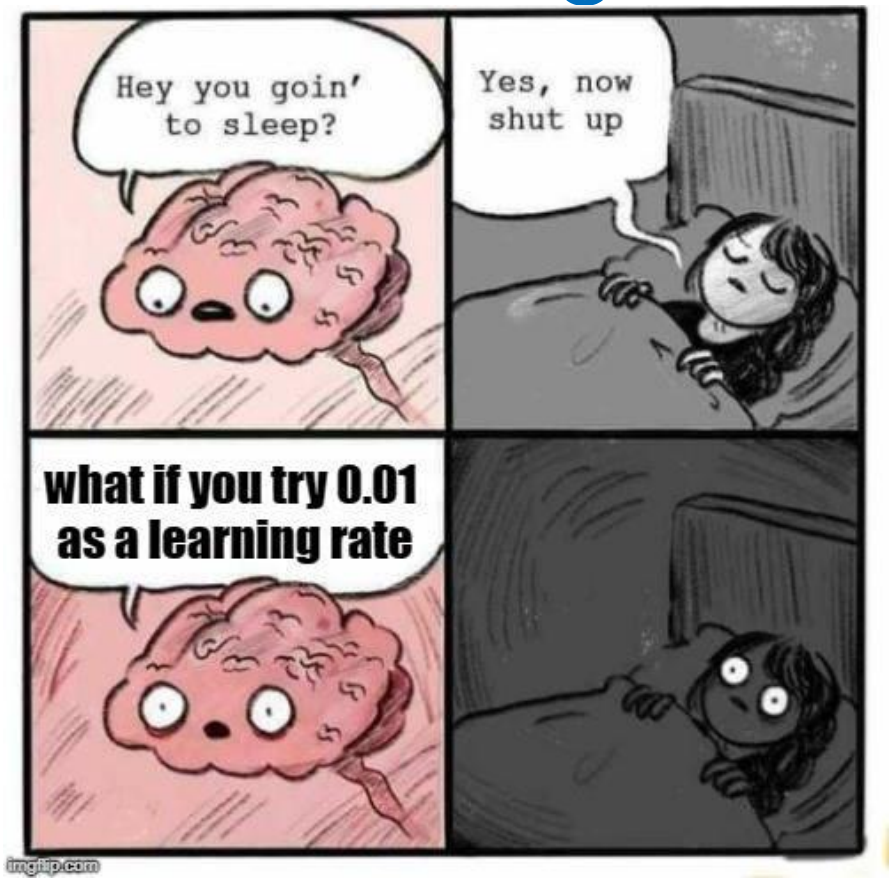


Fun things…


What do you want ?
Learn Deep Learning without Math Knowledge

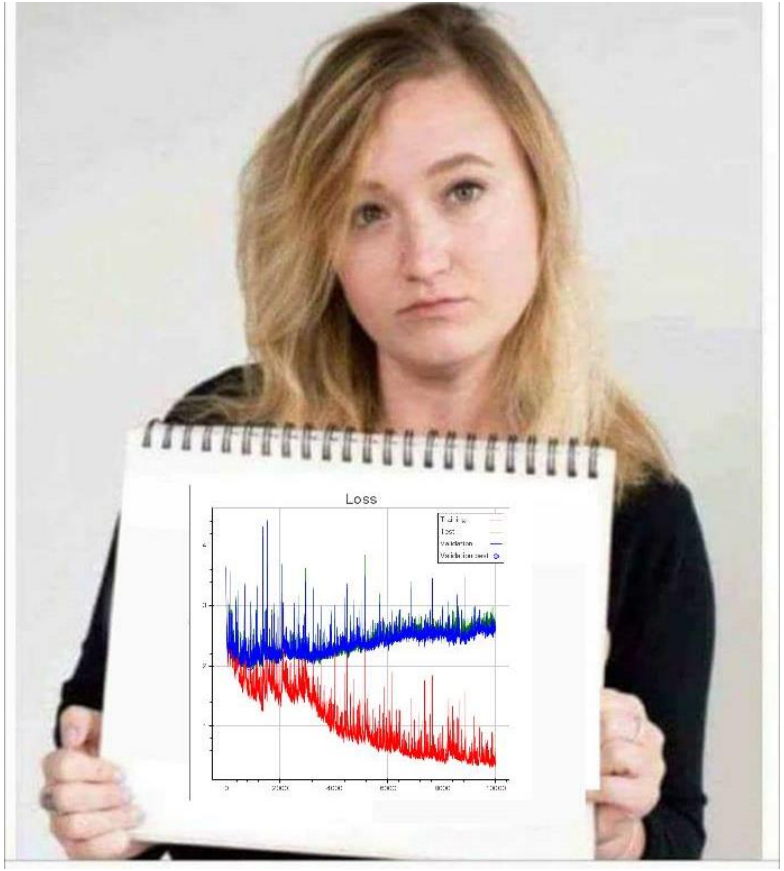

Sorry but you can't
ProgrammersCreateLife

Fun things…


Hey you goin' to sleep?
Yes, now shut up
what if you try 0.01 as a learning rate
imgflip.com

Fun things…

We Asked Her To Draw What Depression Feels Like. This Is What She Drew... #DepressionIsADisease



Going Deep into Neural Networks

Simple Starting Points for Debugging

- Start simple!
  - First, overfit to a single training sample
  - Second, overfit to several training samples

- Always try simple architecture first
  - It will verify that you are learning something

- Estimate timings (how long for each epoch?)

Neural Network

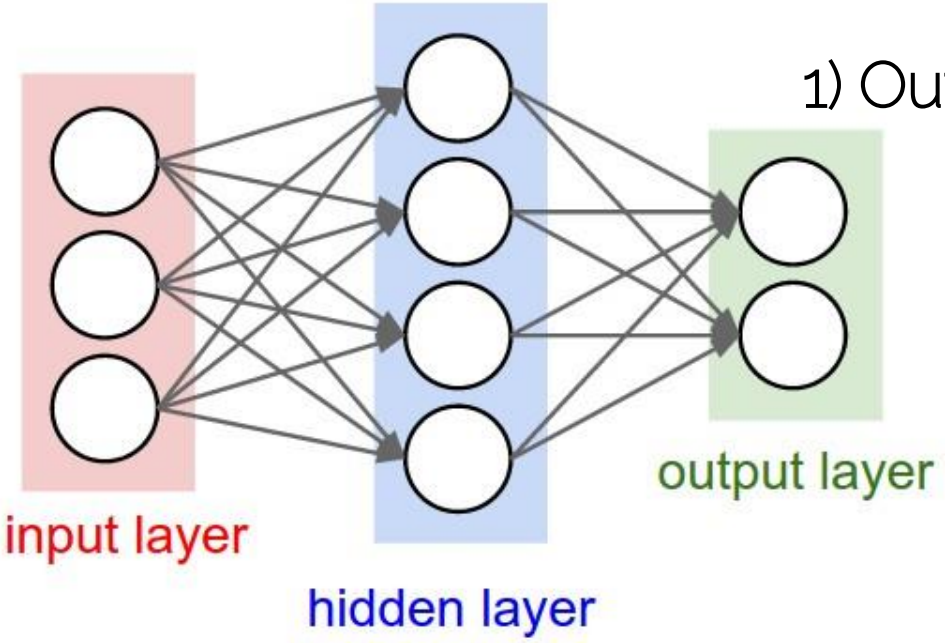
- Problems of going deeper…
  - Vanishing gradients (multiplication of chain rule)
- The impact of small decisions (architecture, activation functions…)
- Is my network training correctly?

Neural Networks

2) Functions in Neurons

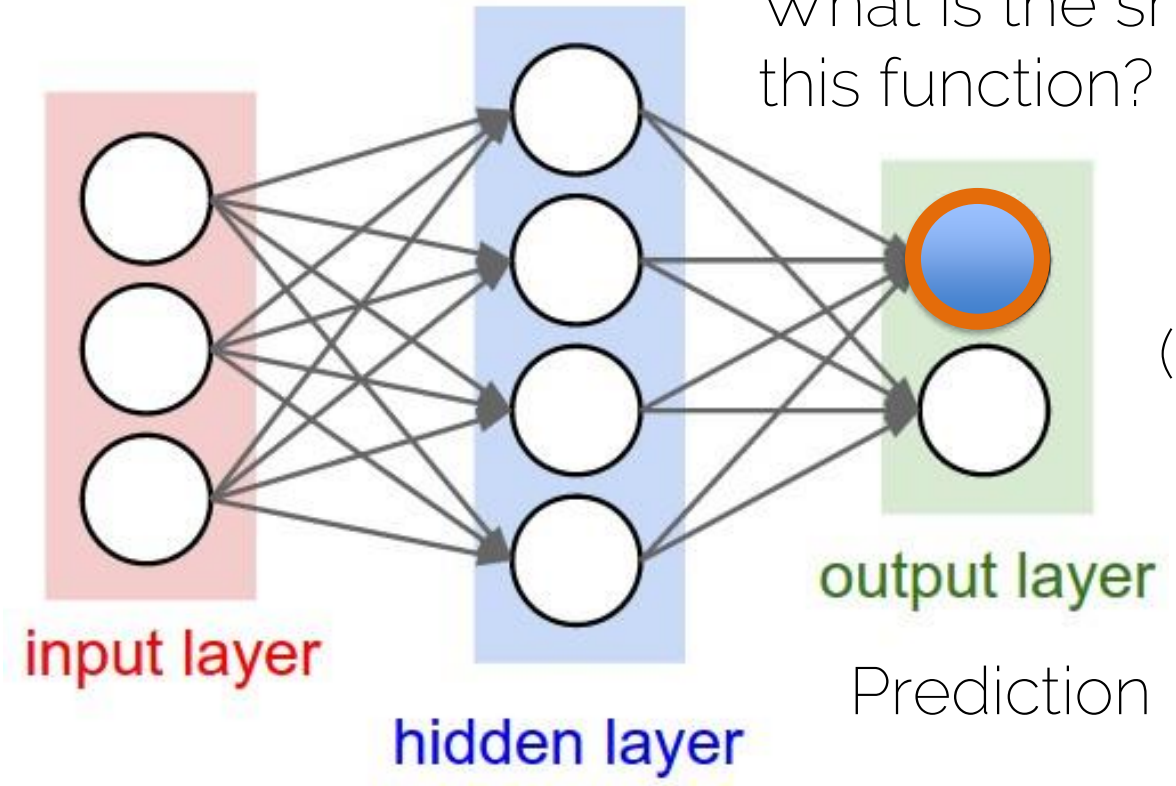
3) Input of data

1) Output functions



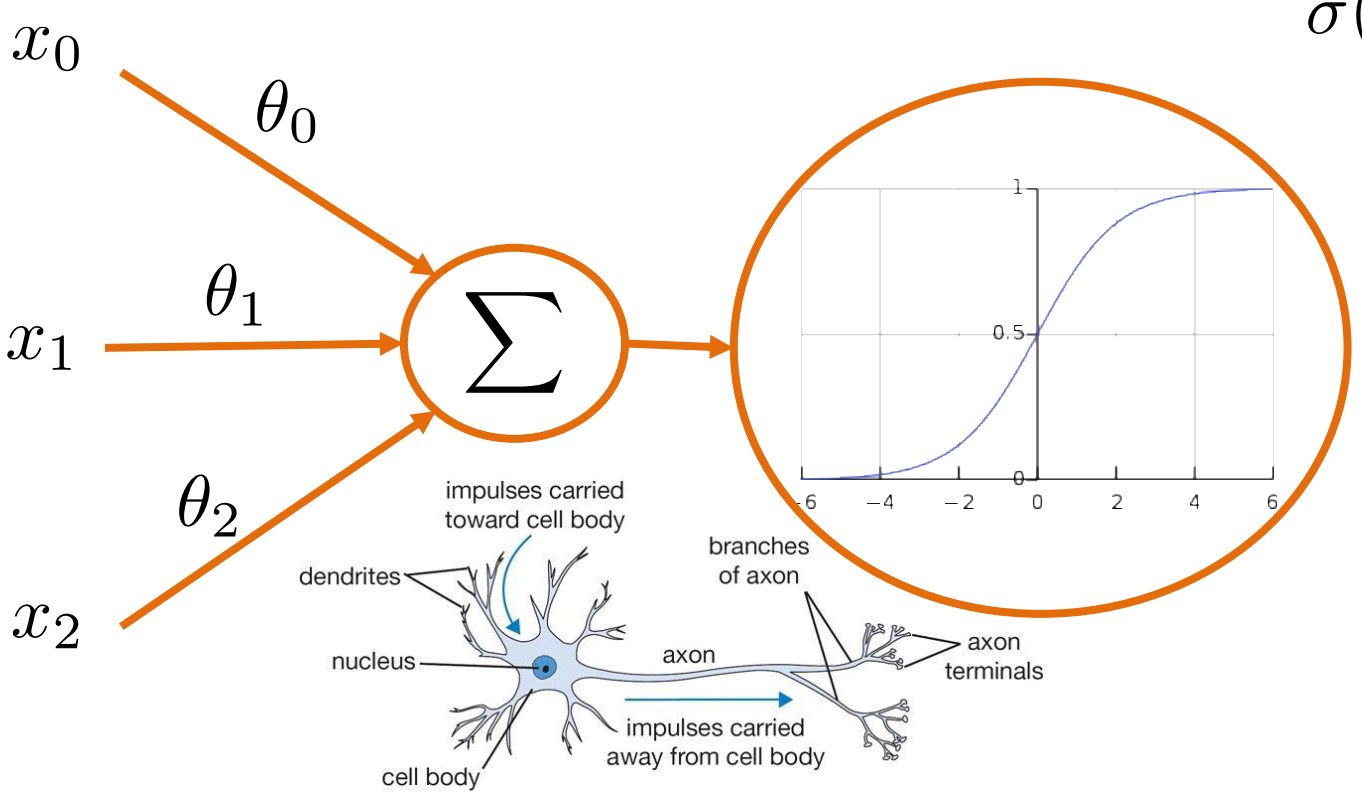
Output Functions

Neural Networks



Sigmoid for Binary Predictions

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



1

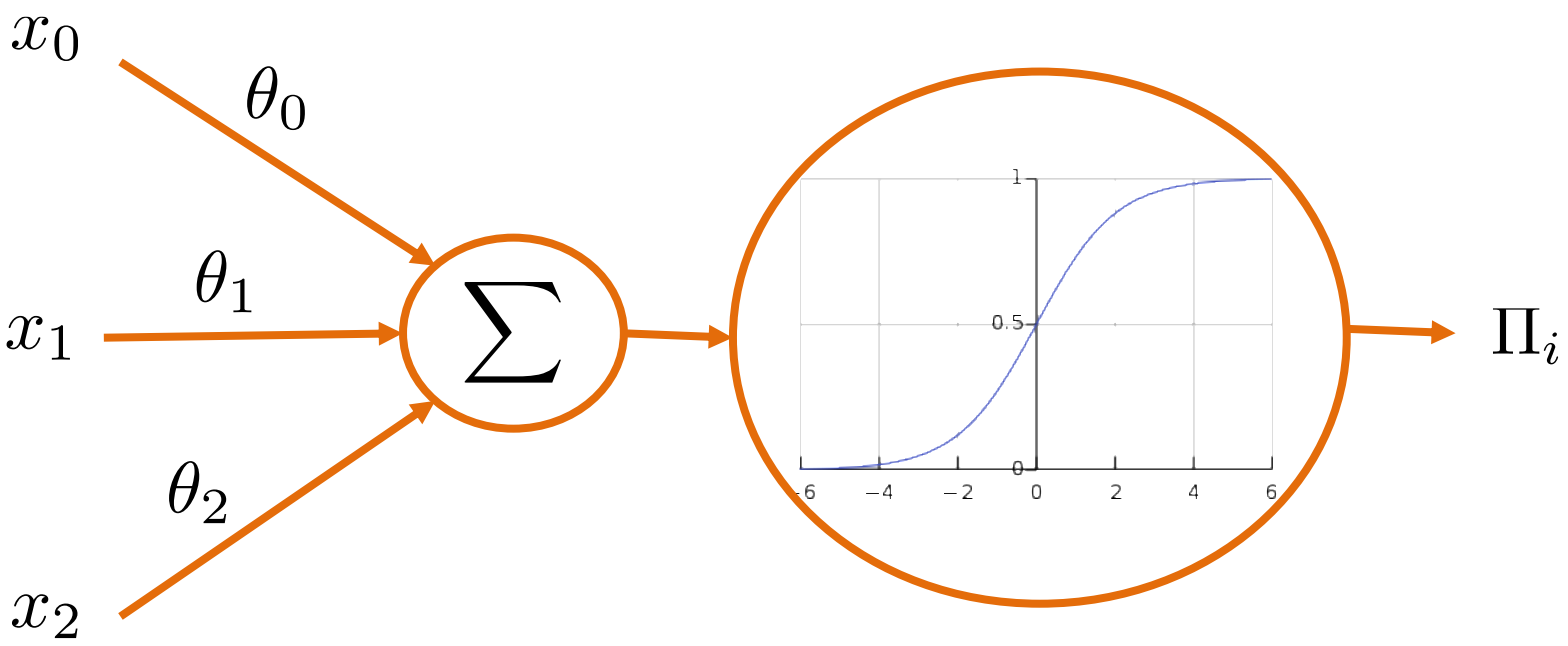
Can be interpreted as a probability

0

$$p(y_i = 1 | \mathbf{x}_i, \boldsymbol{\theta})$$

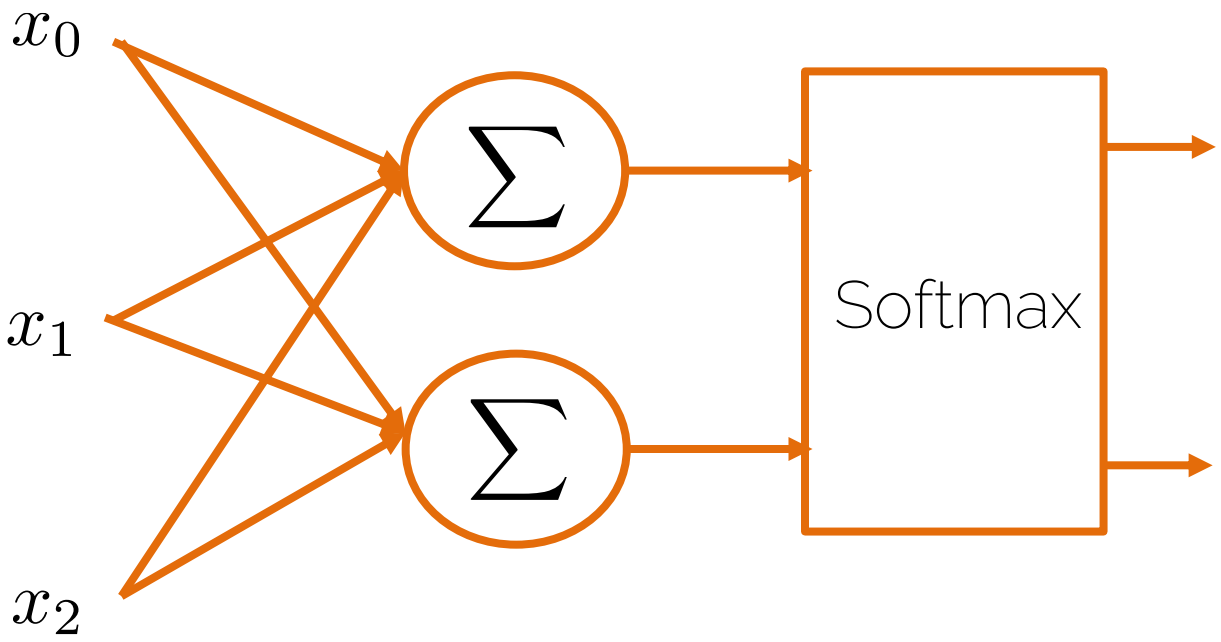
Softmax formulation

- What if we have multiple classes?



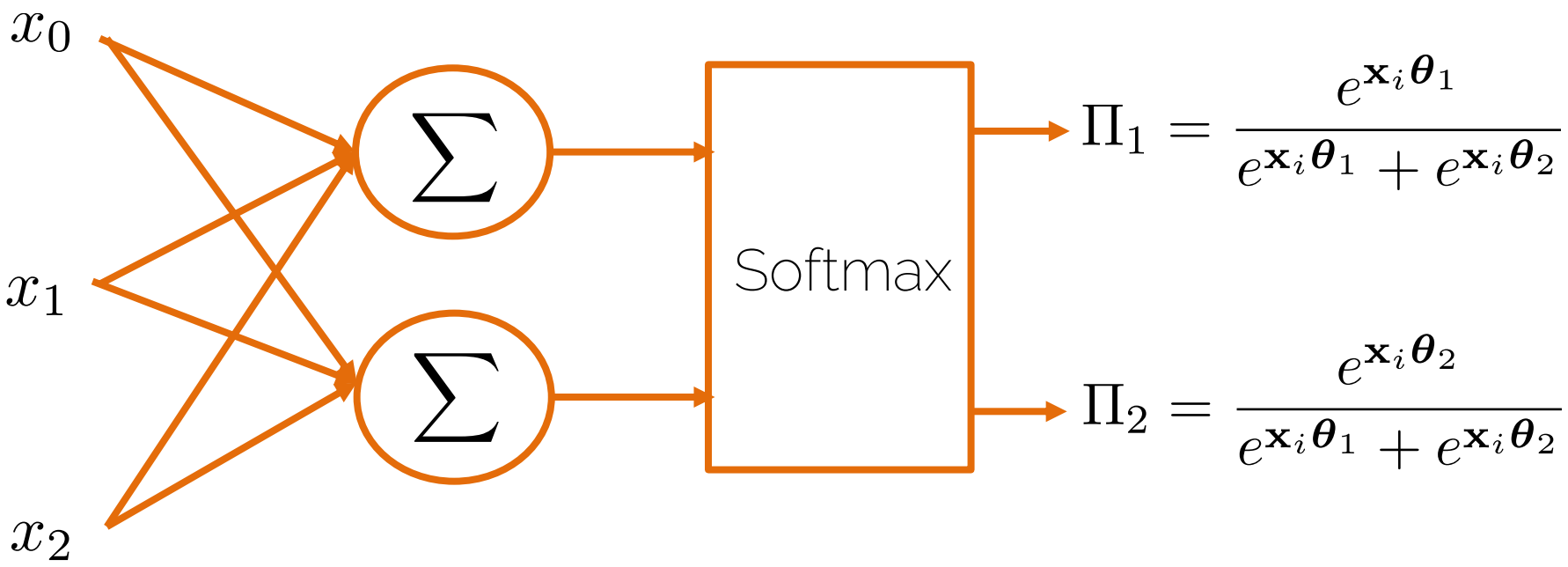
Softmax formulation

- What if we have multiple classes?



Softmax formulation

- What if we have multiple classes?



Softmax formulation

- Softmax

$$p(y_i|\mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\mathbf{x}\boldsymbol{\theta}_i}}{\sum_{k=1}^{n} e^{\mathbf{x}\boldsymbol{\theta}_k}}$$

exp

normalize

- Softmax loss (Maximum Likelihood Estimate)

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Loss Functions

Naïve Losses

L2 Loss: $L^2 = \sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$

- Sum of squared differences (SSD)
- Prune to outliers
- Compute-efficient (optimization)
- Optimum is the mean

L1 Loss: $L^1 = \sum_{i=1}^{n} |y_i - f(x_i)|$

- Sum of absolute differences
- Robust
- Costly to compute
- Optimum is the median

12	24	42	23
34	32	5	2
12	31	12	31
31	64	5	13

x_i

15	20	40	25
34	32	5	2
12	31	12	31
31	64	5	13

y_i

$$L^2(x, y) = 9 + 16 + 4 + 4 + 0 + \cdots + 0 = 66$$

$$L^1(x, y) = 3 + 4 + 2 + 2 + 0 + \cdots + 0 = 15$$

Cross-Entropy (Softmax)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1

Loss

Cross-Entropy (Softmax)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1

3.2
5.1
-1.7

Loss

Cross-Entropy (Softmax)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Score function $s = f(x_i, W)$

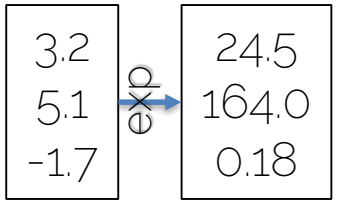
e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1



Loss

Cross-Entropy (Softmax)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Score function $s = f(x_i, W)$

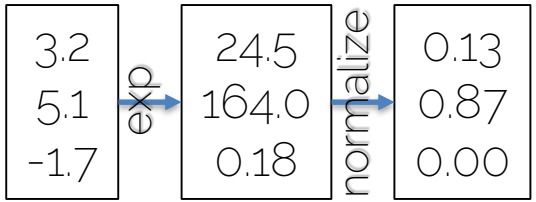
e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1



Loss

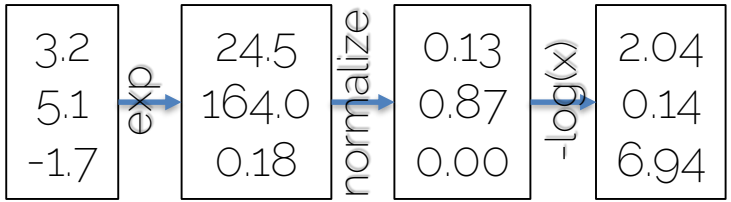
Cross-Entropy (Softmax)

Softmax $$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}}\right)$$

Score function $$s = f(x_i, W)$$

e.g., $$f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)



Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
Loss		2.04	0.14	6.94

Cross-Entropy (Softmax)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Score function $s = f(x_i, W)$

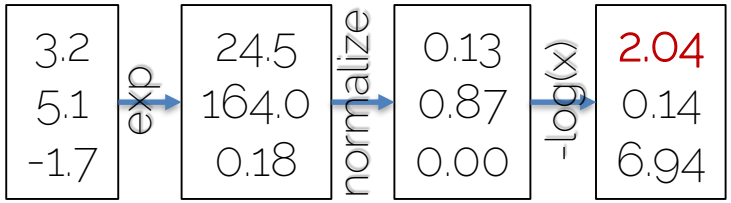
e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
Loss		2.04	0.079	6.156



$$L = \frac{1}{N}\sum_{i=1}^{N} L_i =$$

$$= \frac{L_1 + L_2 + L_3}{3} =$$

$$= \frac{2.04 + 0.079 + 6.156}{3} =$$

$$= \mathbf{2.76}$$

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
Loss		2.9		

$L_1 =$
$\max(0, 5.1 - 3.2 + 1) +$
$\max(0, -1.7 - 3.2 + 1) =$
$= \max(0, 2.9) + \max(0, -3.9) =$
$= 2.9 + 0 =$
$= \mathbf{2.9}$

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
	Loss	2.9	0	

$L_2 =$
$\max(0, 1.3 - 4.9 + 1) +$
$\max(0, 2.0 - 4.9 + 1) =$
$= \max(0, -2.6) + \max(0, -1.9) =$
$= 0 + 0 =$
$= \mathbf{0}$

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
Loss		2.9	0	12.9

$L_3 =$
$\max(0, 2.2 - (-3.1) + 1) +$
$\max(0, 2.5 - (-3.1) + 1) =$
$= \max(0, 6.3) + \max(0, 6.6) =$
$= 6.3 + 6.6 =$
$= \mathbf{12.9}$

Hinge Loss (SVM Loss)

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Suppose: 3 training examples and 3 classes



scores				
	cat	3.2	1.3	2.2
	chair	5.1	4.9	2.5
	"car"	-1.7	2.0	-3.1
	Loss	2.9	0	12.9

Full Loss (over all pairs):

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i =$$

$$= \frac{L_1 + L_2 + L_3}{3} =$$

$$= \frac{2.9 + 0 + 10.9}{3} =$$

$$= \mathbf{4.6}$$

Hinge Loss vs Softmax

Hinge loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Hinge Loss vs Softmax

Hinge loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Given the following scores:

$s = [5, -3, 2]$

$s = [5, 10, 10]$

$s = [5, -20, -20]$

$y_i = 0$

Hinge Loss vs Softmax

Hinge loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Given the following scores:

	Hinge loss:
$s = [5, -3, 2]$	$\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$
$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$
$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$

$y_i = 0$

Hinge Loss vs Softmax

Hinge loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Given the following scores:	Hinge loss:	Softmax loss:
$s = [5, -3, 2]$	$\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$	Google… -ln(e^(5)/(e^(5)+e^(-3)+e^(2)))=0.05
$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$	Google… -ln(e^(5)/(e^(5)+e^(10)+e^(10)))=5.70
$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$	Google… -ln(e^(5)/(e^(5)+e^(-20)+e^(-20)))=2.e-11

$y_i = 0$

Hinge Loss vs Softmax

Hinge loss: $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

Softmax: $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

Given the following scores:	Hinge loss:	Softmax loss:
$s = [5, -3, 2]$	$\max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0$	Google… -ln(e^(5)/(e^(5)+e^(-3)+e^(2)))=0.05
$s = [5, 10, 10]$	$\max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12$	Google… -ln(e^(5)/(e^(5)+e^(10)+e^(10)))=5.70
$s = [5, -20, -20]$	$\max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0$	Google… -ln(e^(5)/(e^(5)+e^(-20)+e^(-20)))=2.e-11

$y_i = 0$

Softmax *always* wants to improve!
Hinge Loss saturates

Loss in Compute Graph

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

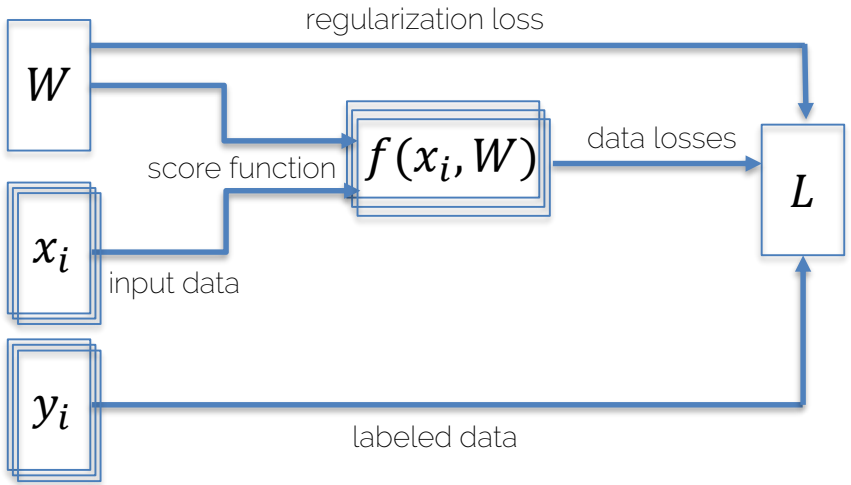
Given a function with weights W,
Training pairs $[x_i; y_i]$ (input and labels)

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

e.g., L^2-reg: $R^2(W) = \sum_{i=1}^{N} w_i^2$

Full Loss $L = \frac{1}{N} \sum_{i=1}^{N} L_i + R^2(W)$



Compute Graphs

Score function $s = f(x_i, W)$

e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

e.g., L^2-reg: $R^2(W) = \sum_{i=1}^{N} w_i^2$

Full Loss $L = \frac{1}{N} \sum_{i=1}^{N} L_i + R^2(W)$

Compute Graphs

Score function $s = f(x_i, W)$

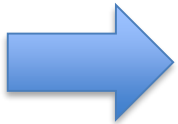
e.g., $f(x_i, W) = W \cdot [x_0, x_1, \ldots, x_N]^T$

Softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}})$

SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

e.g., L^2-reg: $R^2(W) = \sum_{i=1}^{N} w_i^2$

Full Loss $L = \frac{1}{N} \sum_{i=1}^{N} L_i + R^2(W)$



Want to find optimal W. I.e., weights are unknowns of optimization problem

Compute gradient w.r.t. W.

Gradient $\nabla_W L$ is computed via backpropagation

Weight Regularization & SVM Loss

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$

Full loss $L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max\left(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1\right) + \lambda R(W)$

L^1-reg: $R^1(W) = \sum_{i=1}^{N} \sum_{j \neq y_i} |w_i|$

L^2-reg: $R^2(W) = \sum_{i=1}^{N} \sum_{j \neq y_i} w_i^2$

Weight Regularization & SVM Loss

Multiclass SVM loss $L_i = \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$

Full loss $L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max\left(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1\right) + \lambda R(W)$

L^1-reg: $R^1(W) = \sum_{i=1}^{N} \sum_{j \neq y_i} |w_i|$

L^2-reg: $R^2(W) = \sum_{i=1}^{N} \sum_{j \neq y_i} w_i^2$

Example:

$x = [1,1,1,1]$

$w_1 = [1, 0, 0, 0]$

$w_2 = [0.25, 0.25, 0.25, 0.25]$

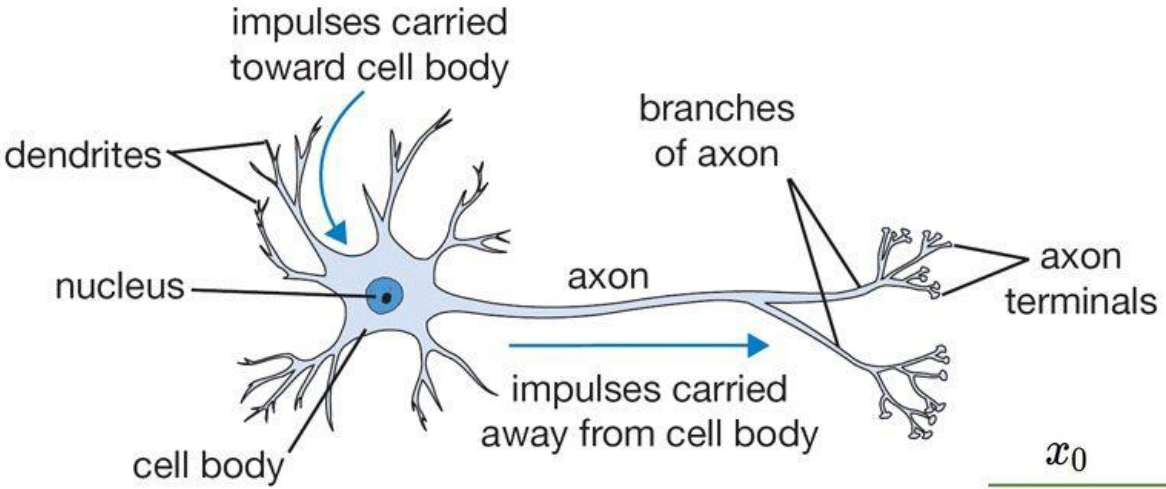
$w_1^T x = w_2^T x = 1$

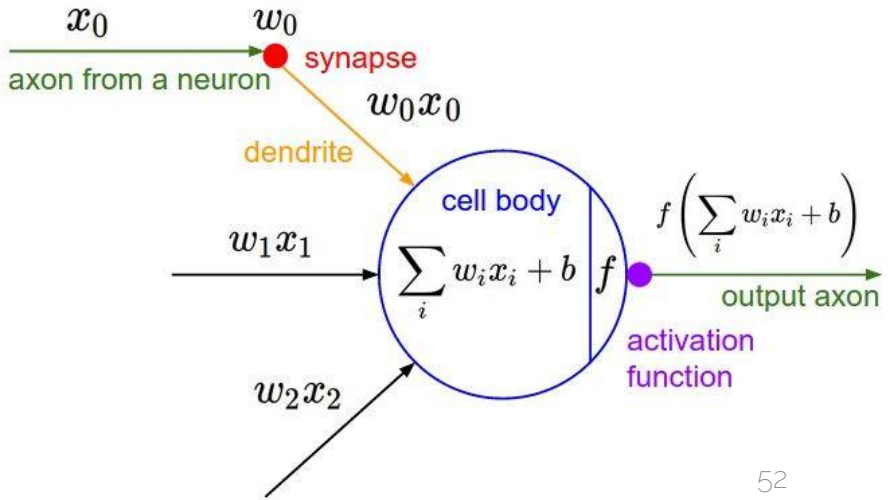
$R^2(w_1) = 1$

$R^2(w_2) = 0.25^2 + 0.25^2 + 0.25^2 + 0.25^2 = 0.25$

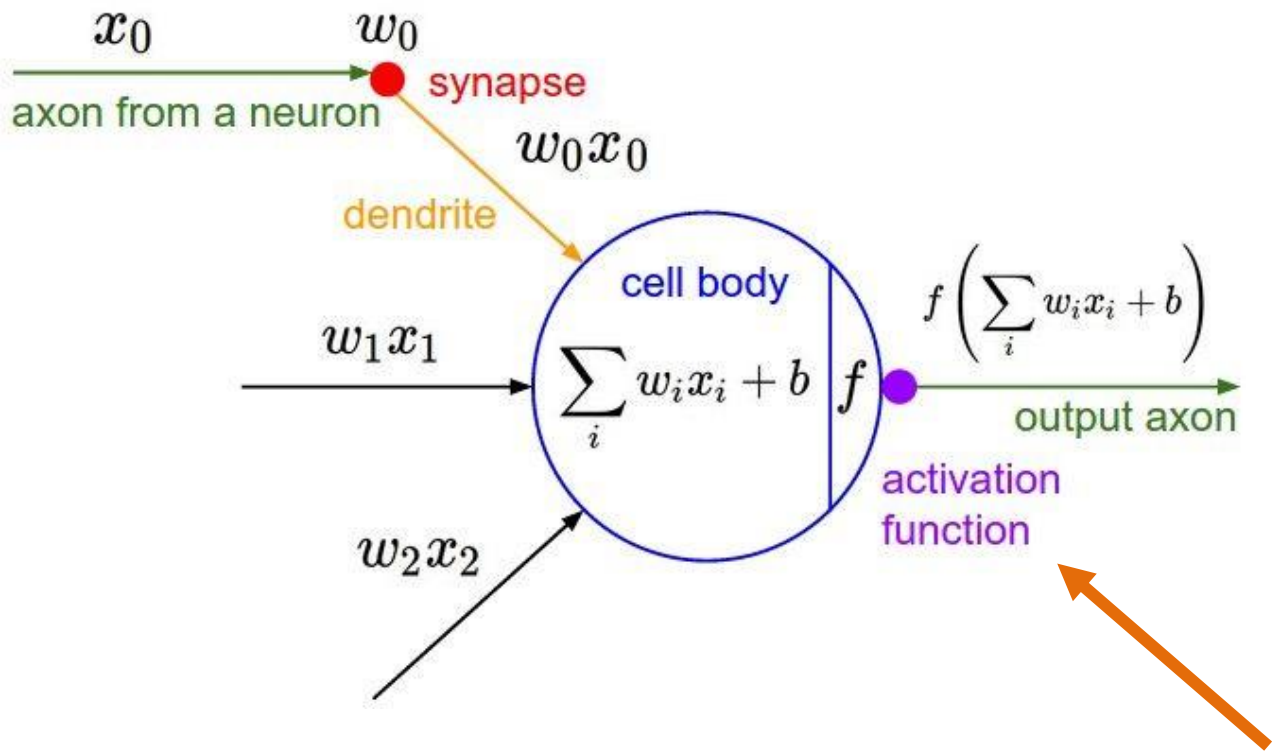
Activation Functions

Neurons

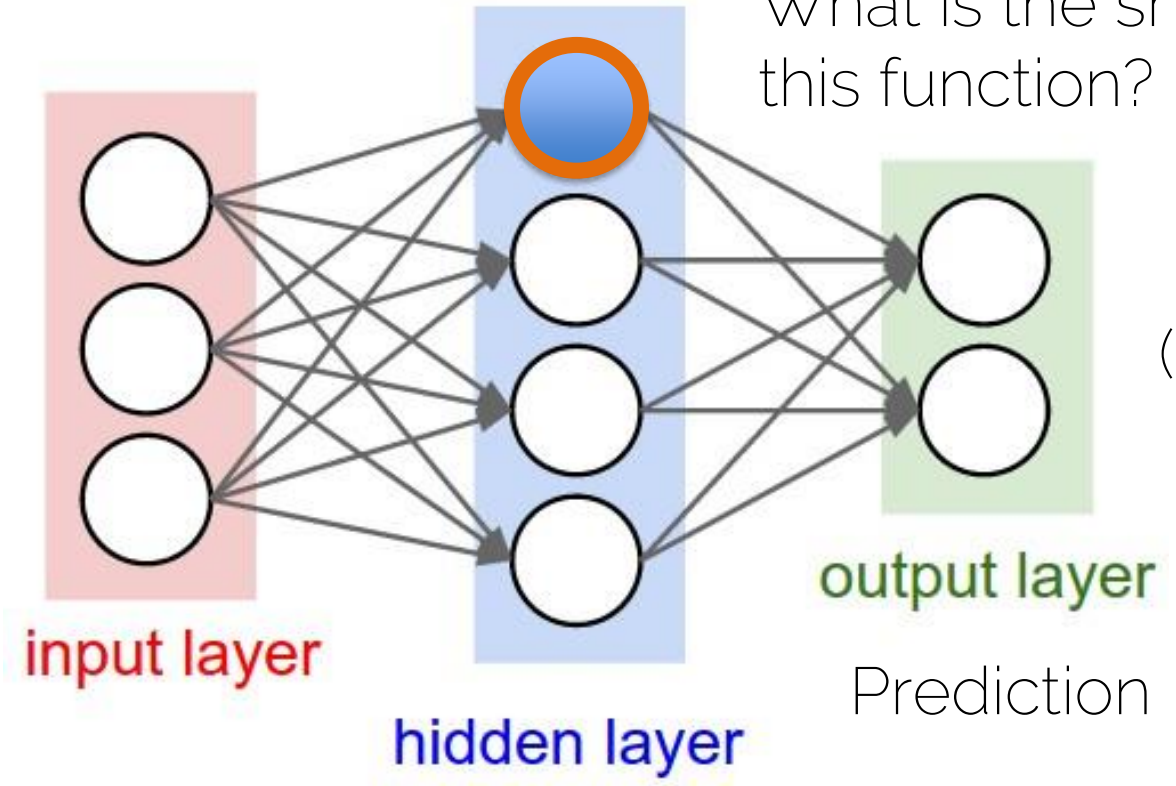

impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$

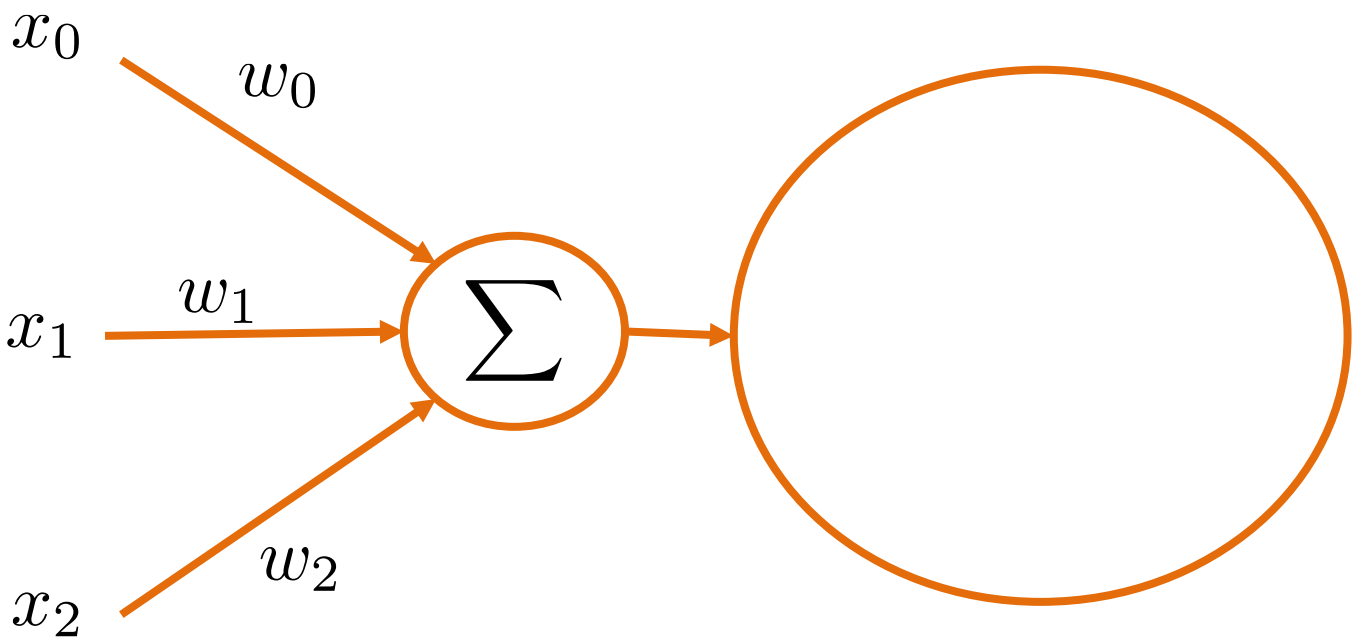
Neurons


x_0
w_0
axon from a neuron
synapse
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation function
$w_2 x_2$

Neural Networks

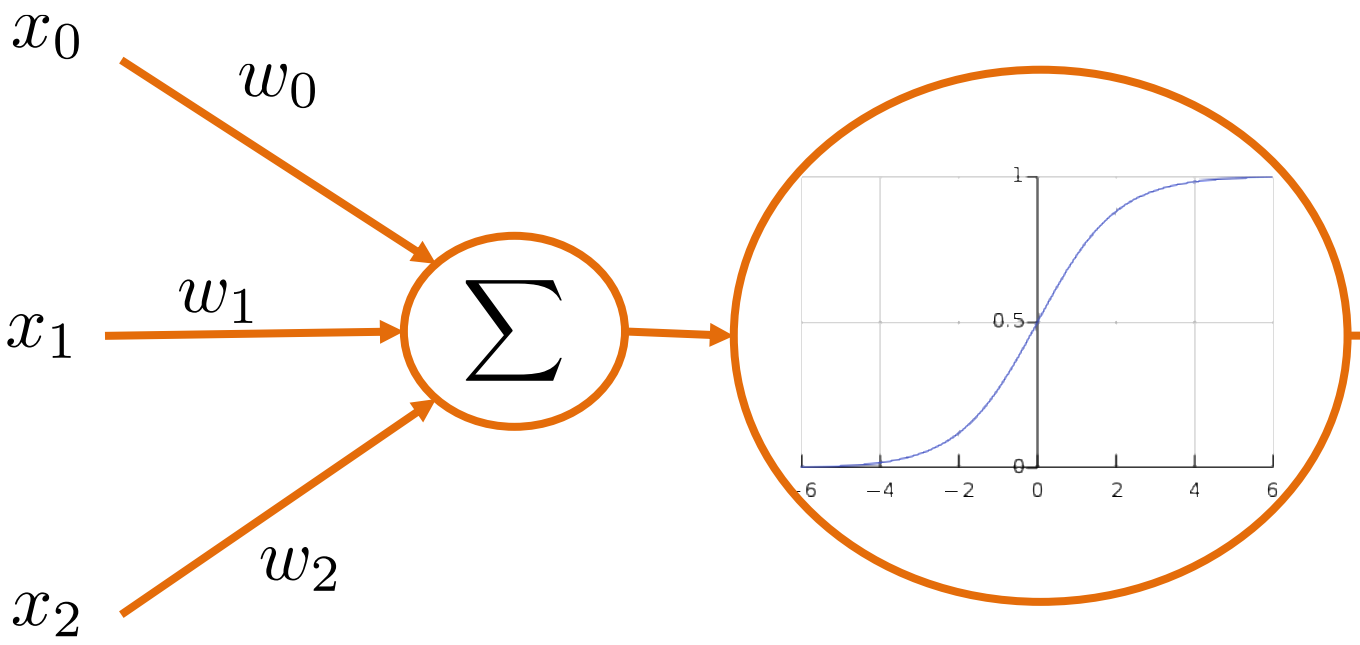


Activation Functions or Hidden Units



Sigmoid

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



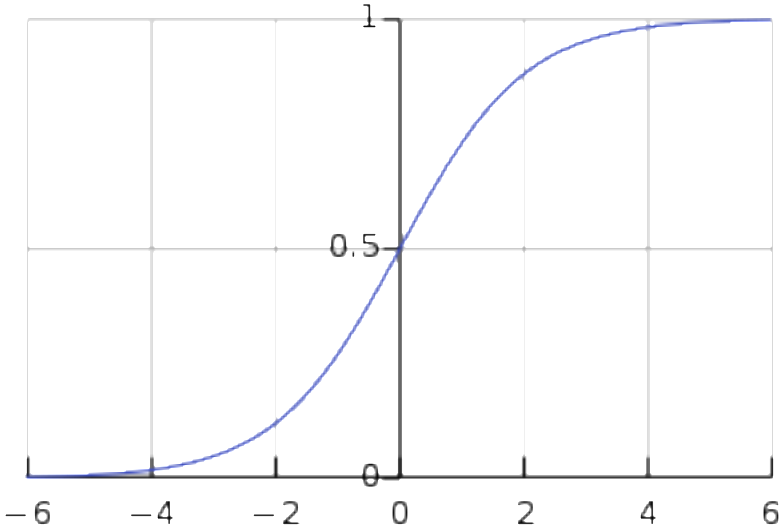
$y_i \in \{0, 1\}$

Can be interpreted as a probability

Sigmoid

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

Forward



$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma}$$

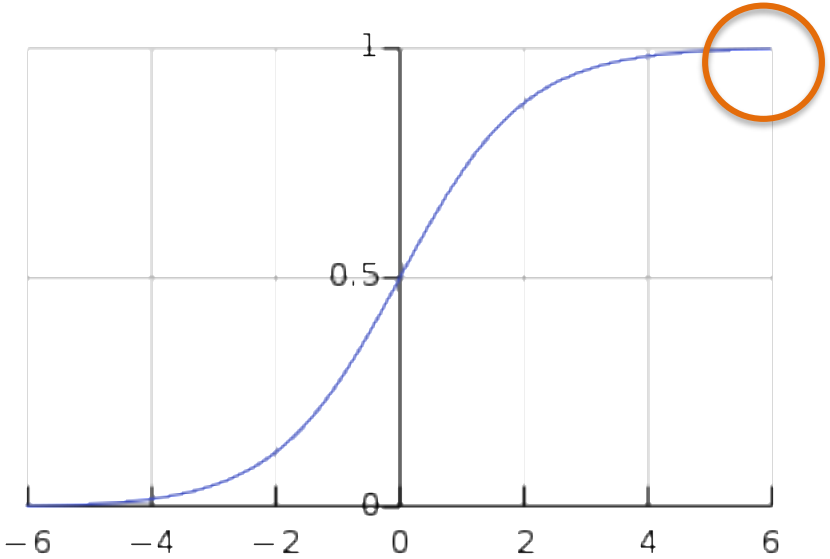
$$\frac{\partial \sigma}{\partial x}$$

$$\frac{\partial L}{\partial \sigma}$$

Sigmoid

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

Forward



$$x = 6$$

✘ Saturated neurons kill the gradient flow

$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x} \frac{\partial L}{\partial \sigma}$$

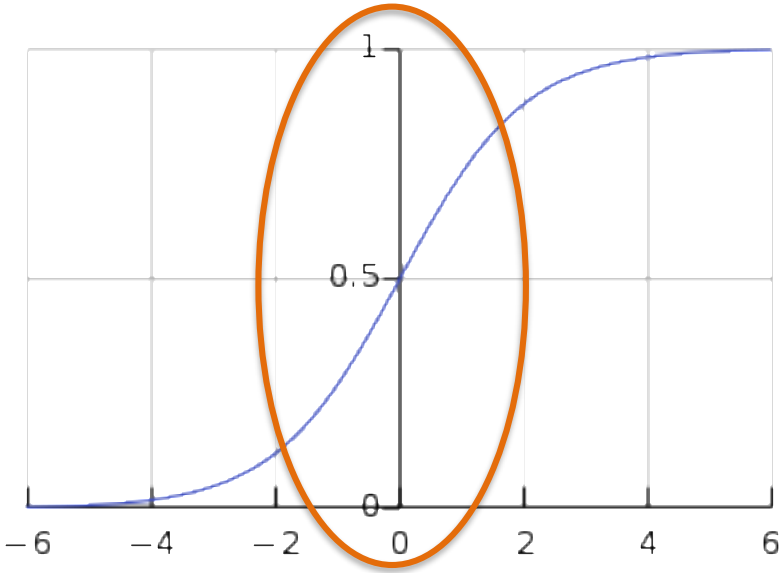
$$\frac{\partial \sigma}{\partial x}$$

$$\frac{\partial L}{\partial \sigma}$$

Sigmoid

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

Forward

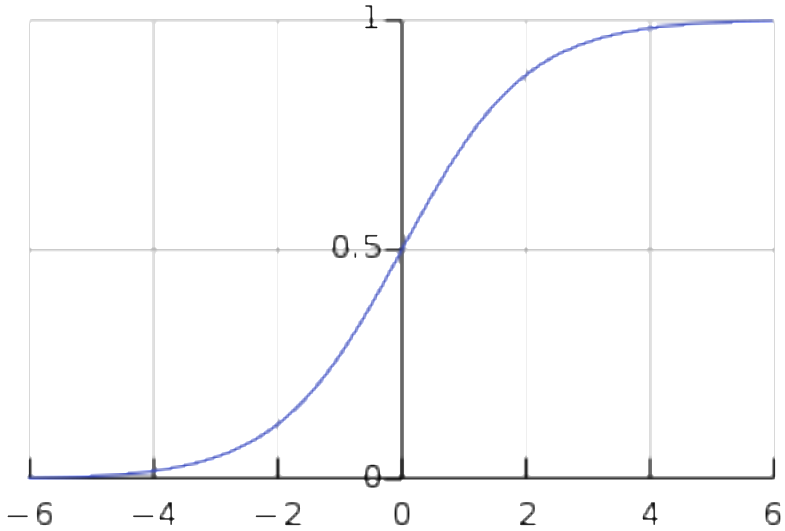


Active region for gradient descent

$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma} \quad \leftarrow \quad \frac{\partial \sigma}{\partial x} \quad \leftarrow \quad \frac{\partial L}{\partial \sigma}$$

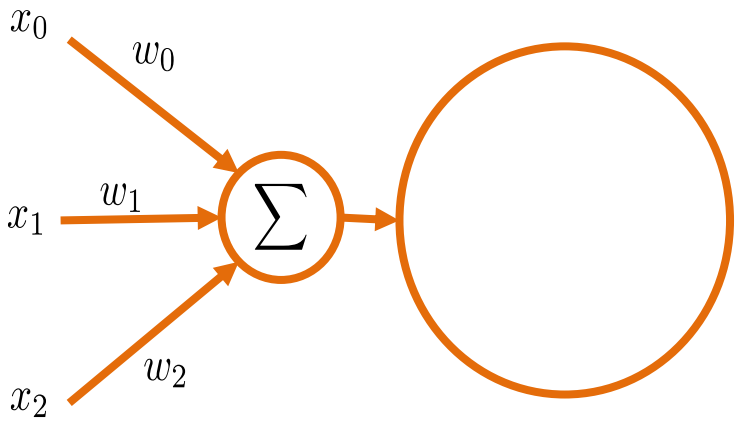
Sigmoid

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



Output is always positive

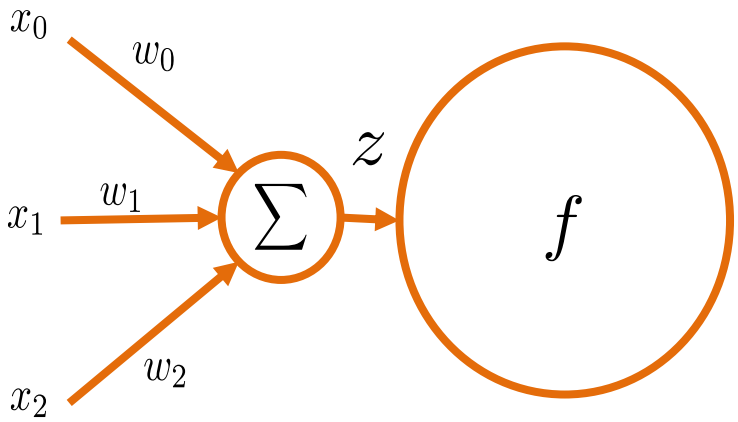
Problem of Positive Output



$$f\left(\sum_i w_i x_i + b\right)$$

We want to compute the gradient wrt the weights

Problem of Positive Output

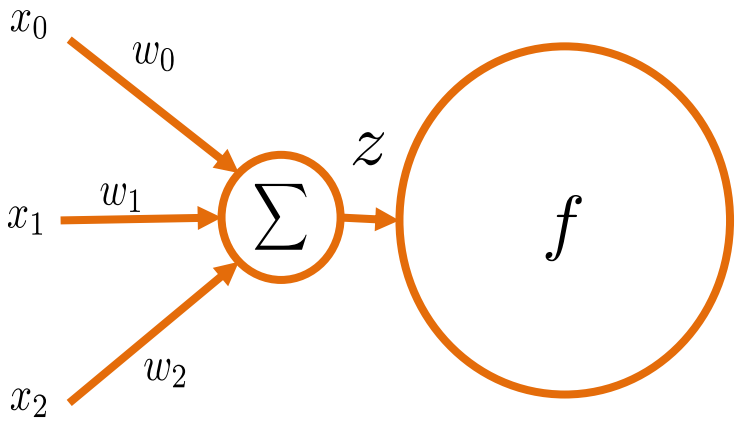


$$f\left(\underbrace{\sum_i w_i x_i + b}\right)$$

$$\frac{\partial z}{\partial w} = x_i > 0$$

We want to compute the gradient wrt the weights

Problem of Positive Output



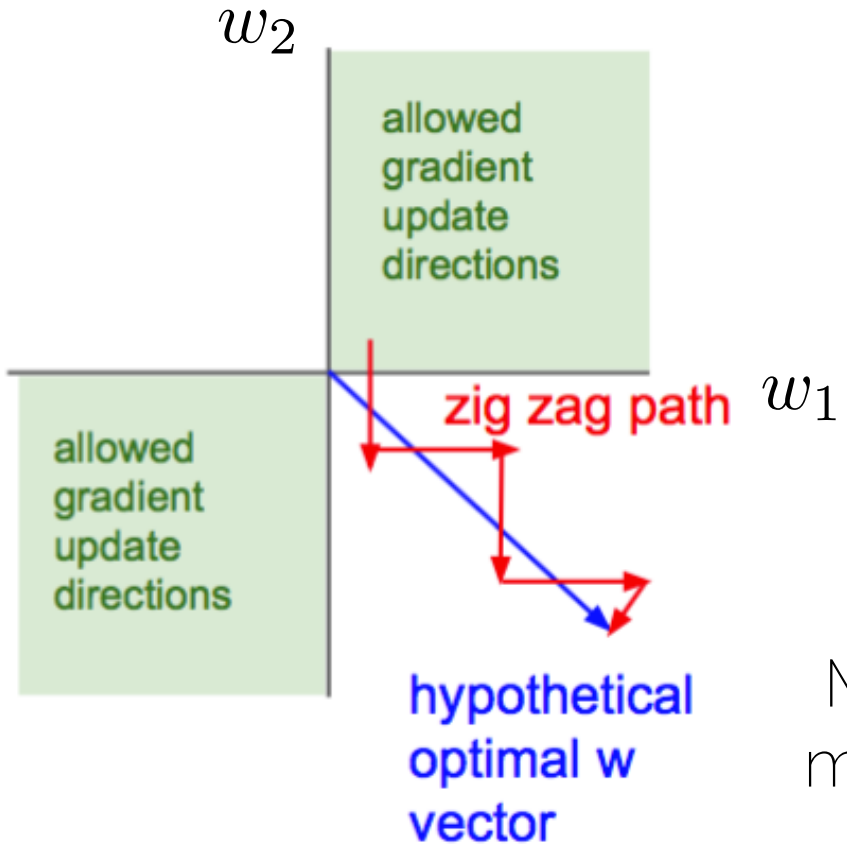
$$f\left(\sum_i w_i x_i + b\right)$$

$$\frac{\partial f}{\partial z}$$

$$\frac{\partial z}{\partial w} = x_i > 0$$

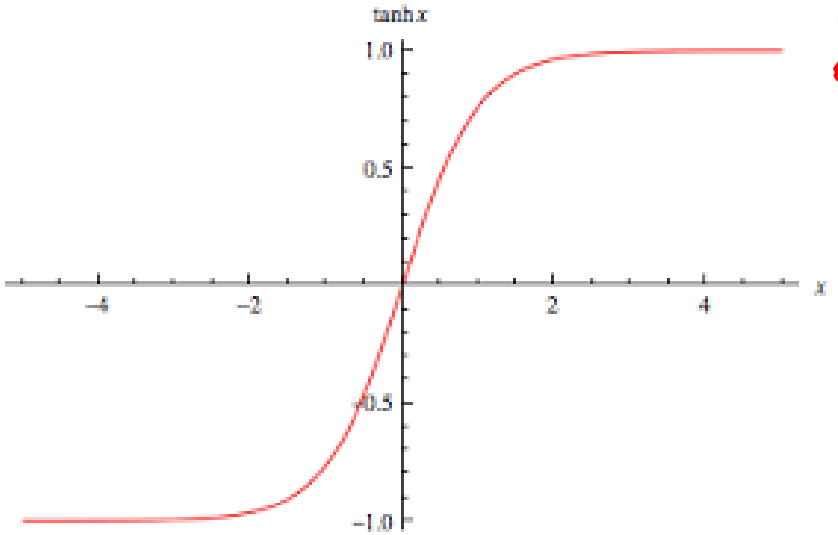
It is going to be either positive or negative for all weights

Problem of Positive Output



More on zero-mean data later

tanh



✘ Still saturates

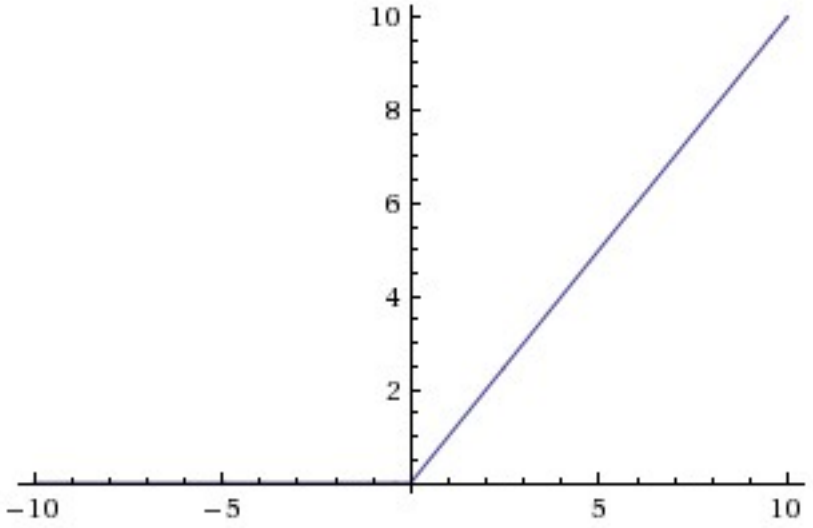
✘ Still saturates

✓ Zero-centered

LeCun 1991

Rectified Linear Units (ReLU)

$$\sigma(x) = \max(0, x)$$



Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

Krizhevsky 2012

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?

Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

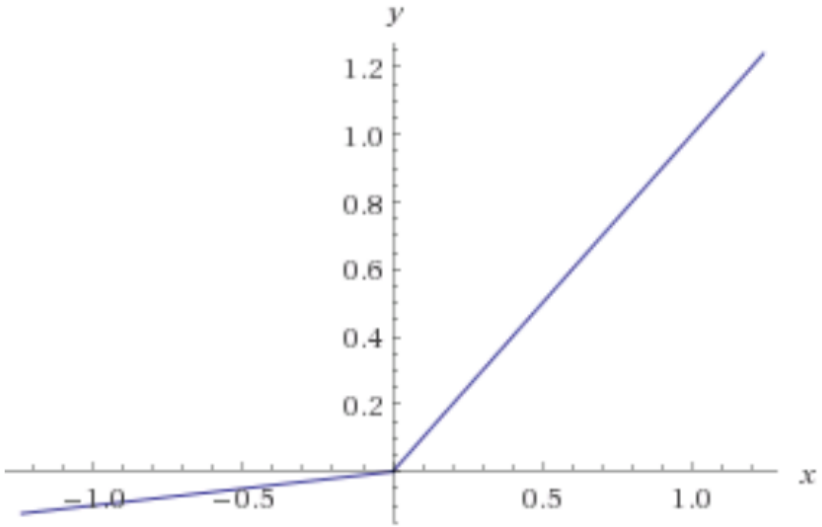
Rectified Linear Units (ReLU)

- Initializing ReLU neurons with slightly positive biases (0.1) makes it likely that they stay active for most inputs

$$f\left(\sum_i w_i x_i + b\right)$$

Leaky ReLU

$$\sigma(x) = \max(0.01x, x)$$

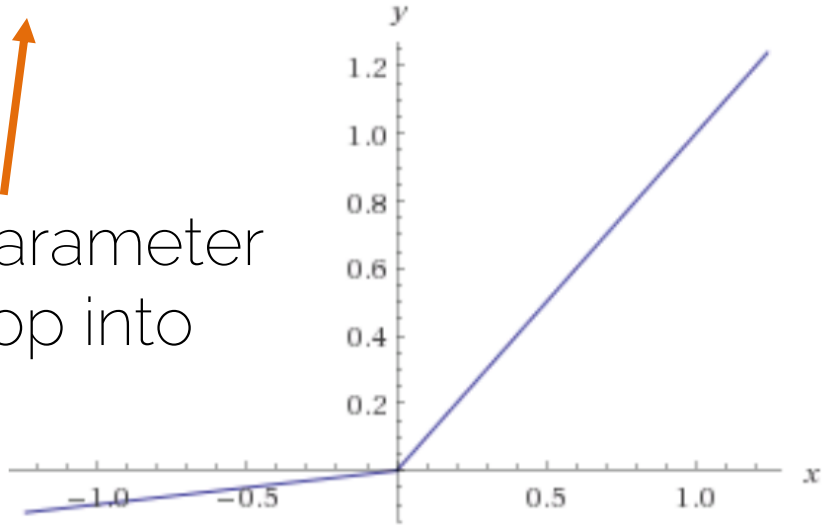


✓ Does not die

Parametric ReLU

$$\sigma(x) = \max(\alpha x, x)$$

One more parameter to backprop into

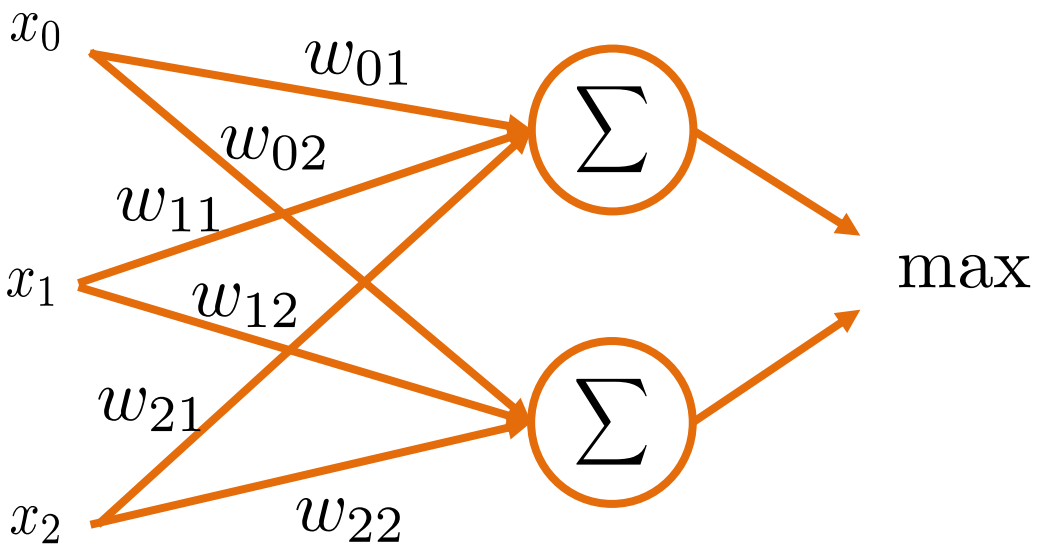


✓ Does not die

He 2015

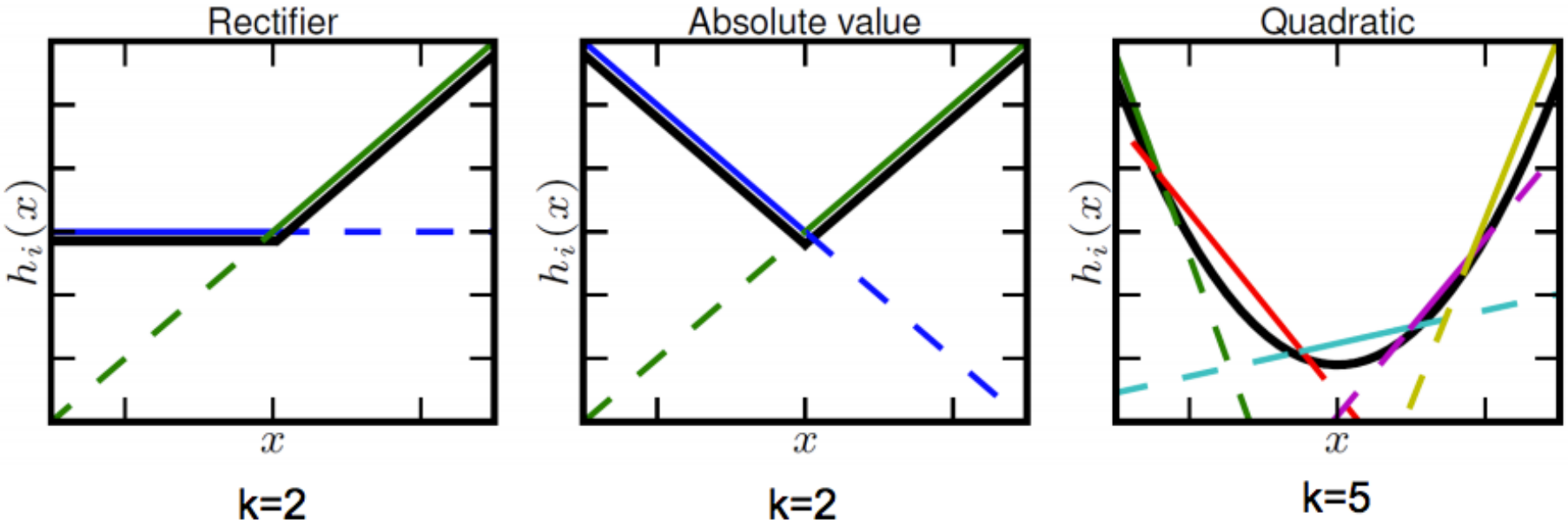
Maxout units

$$Maxout = \max(w_1^T x + b_1, w_2^T x + b_2)$$



Goodfellow 2013

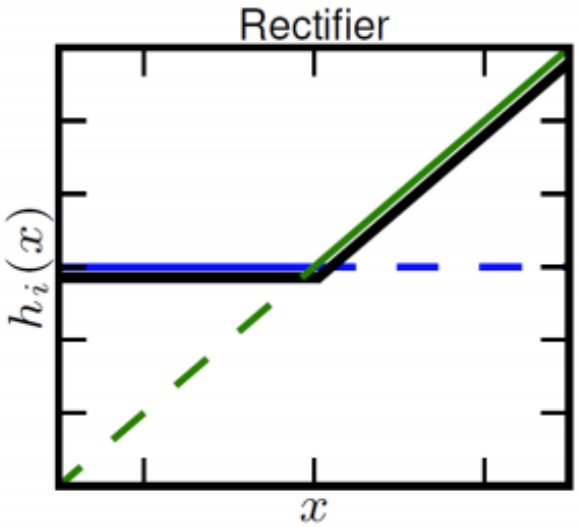
Maxout units



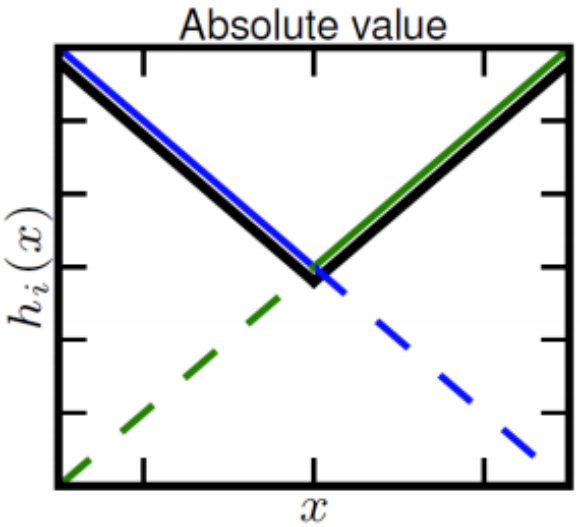
Piecewise linear approximation of a convex function with N pieces

Goodfellow 2013

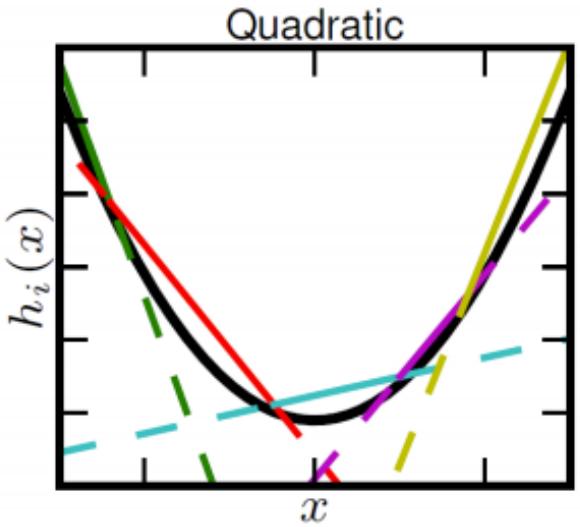
Maxout units



k=2



k=2



k=5

✓ Generalization of ReLUs

✓ Linear regimes

✓ Does not die

✓ Does not saturate

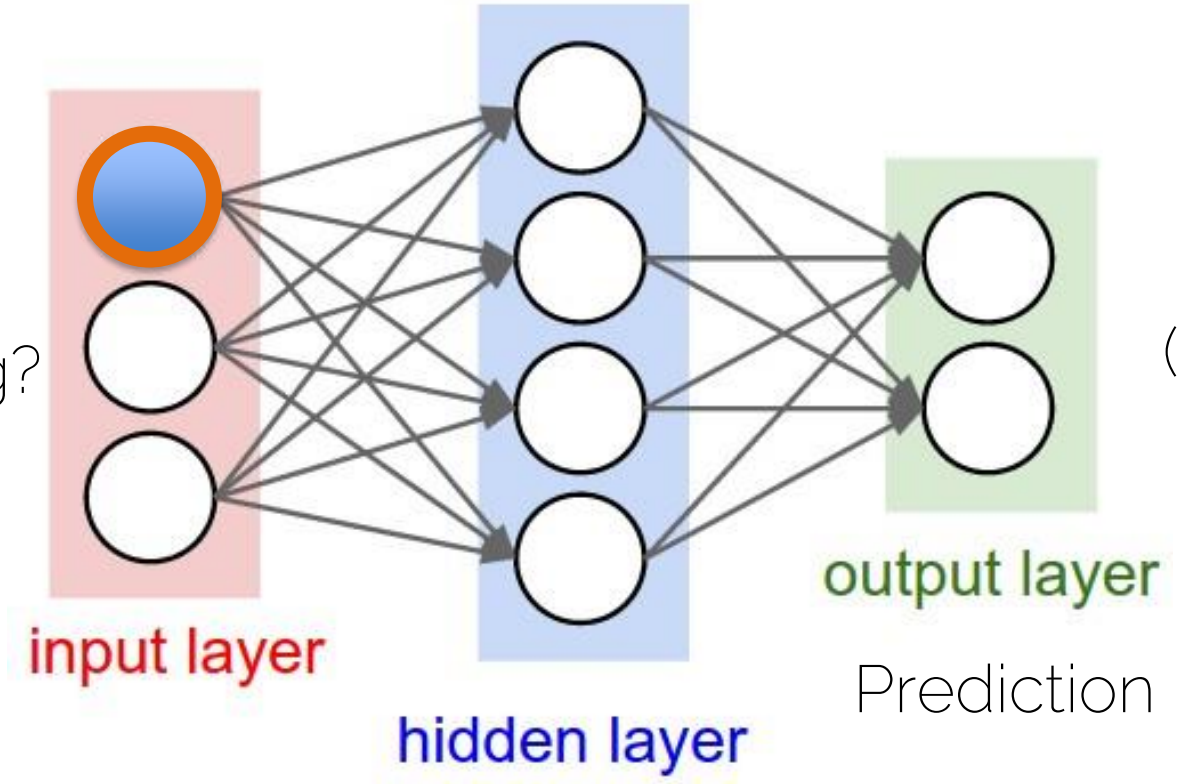
✗ Increase of the number of parameters

Quick Guide

- Sigmoid is not really used
- ReLU is the standard choice
- Second choice are the variants of ReLu or Maxout
- Recurrent nets will require tanh or similar

Neural Networks

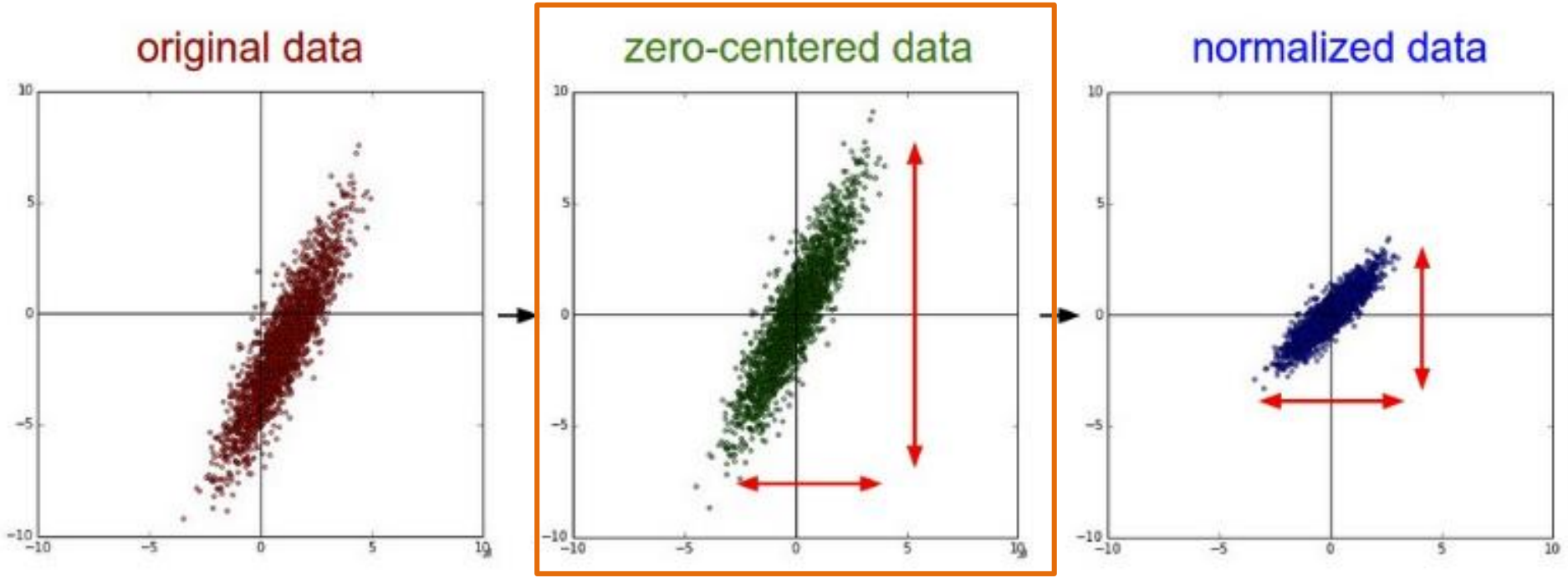
Data pre-processing?



Prediction

Loss (Softmax, Hinge)

Data pre-processing



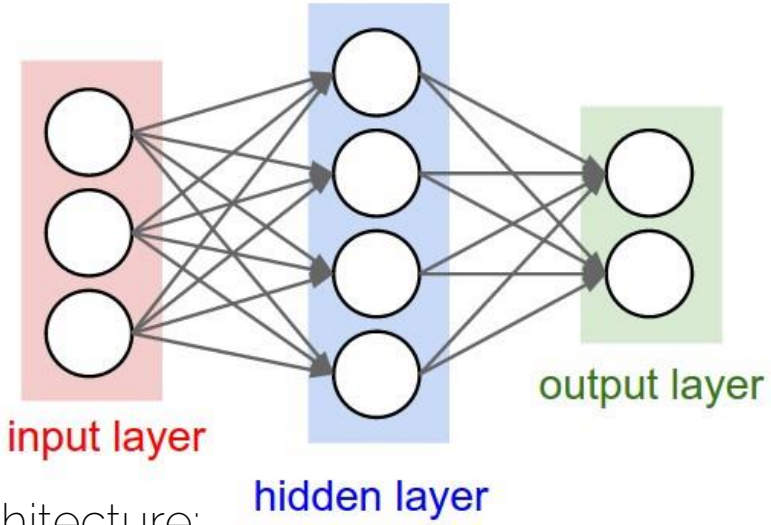
For images subtract the mean image (AlexNet) or per-channel mean (VGG-Net)

Outlook

Outlook

Regularization in the optimization:
- Dropout, weight decay, etc..

Init. of optimization
- How to set weights at beginning



Regularization in the architecture:
- Convolutions! (CNNs)

Handling limited training data
- Data augmentation

Next lecture

- Next lecture Dec 13th
  - More about training neural networks; regularization, BN, dropout, etc.
  - Followed by CNNs