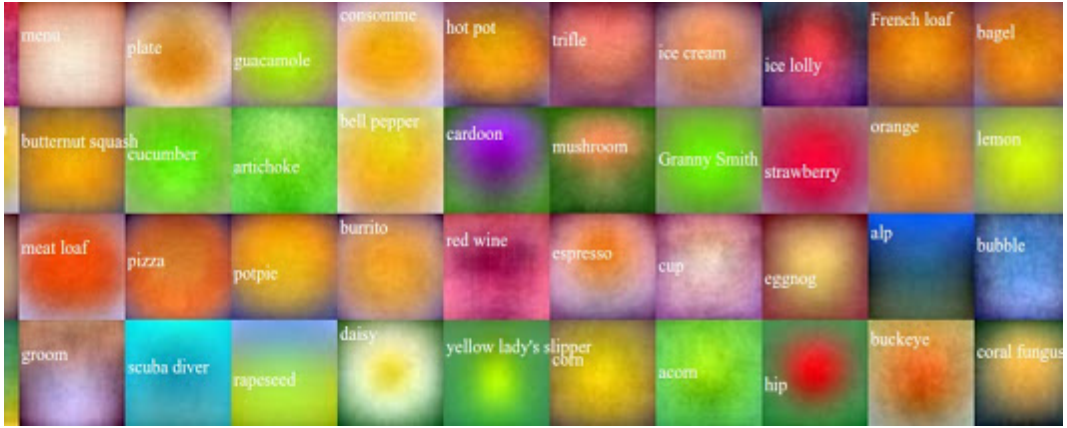
Introduction to Neural Networks

Beyond linear

- Linear score function $f = Wx$

On CIFAR-10

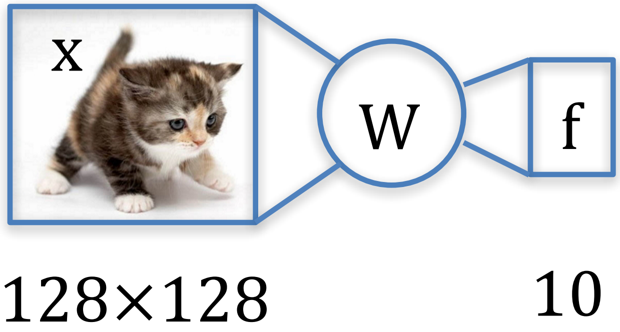


On ImageNet

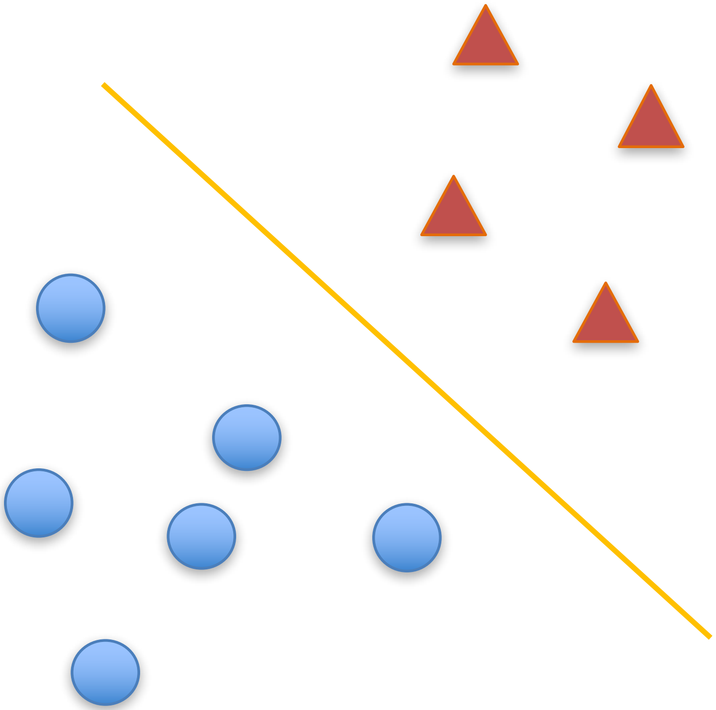
Credit Li/Karpathy/Johnson

Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$

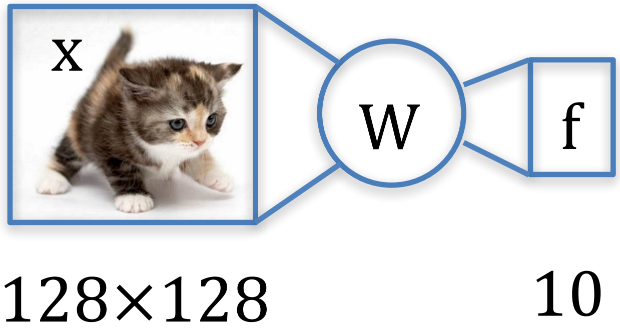


LINEAR TRANSFORMATION

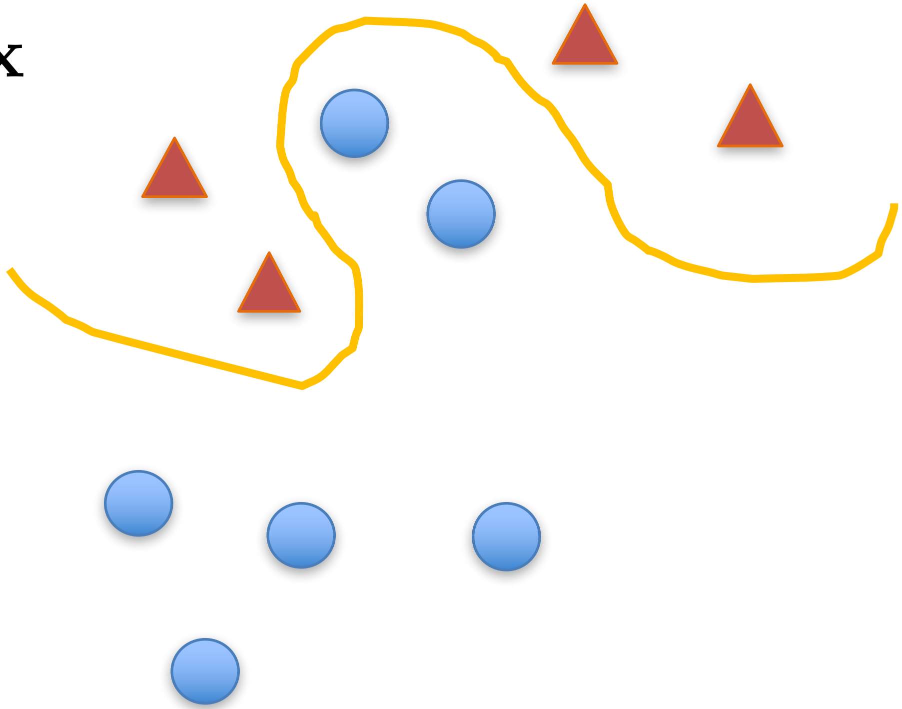


Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$



LINEAR
TRANSFORMATION



Kernel trick

1-layer network: $f = \mathbf{W}\mathbf{x}$

$f = \mathbf{W}\phi(\mathbf{x})$

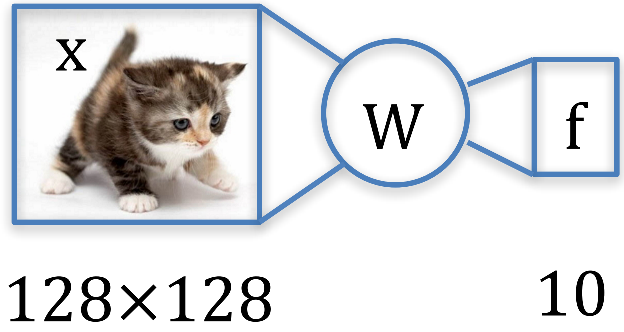
kernel

128×128

10

Neural networks

1-layer network: $f = \mathbf{W}\mathbf{x}$



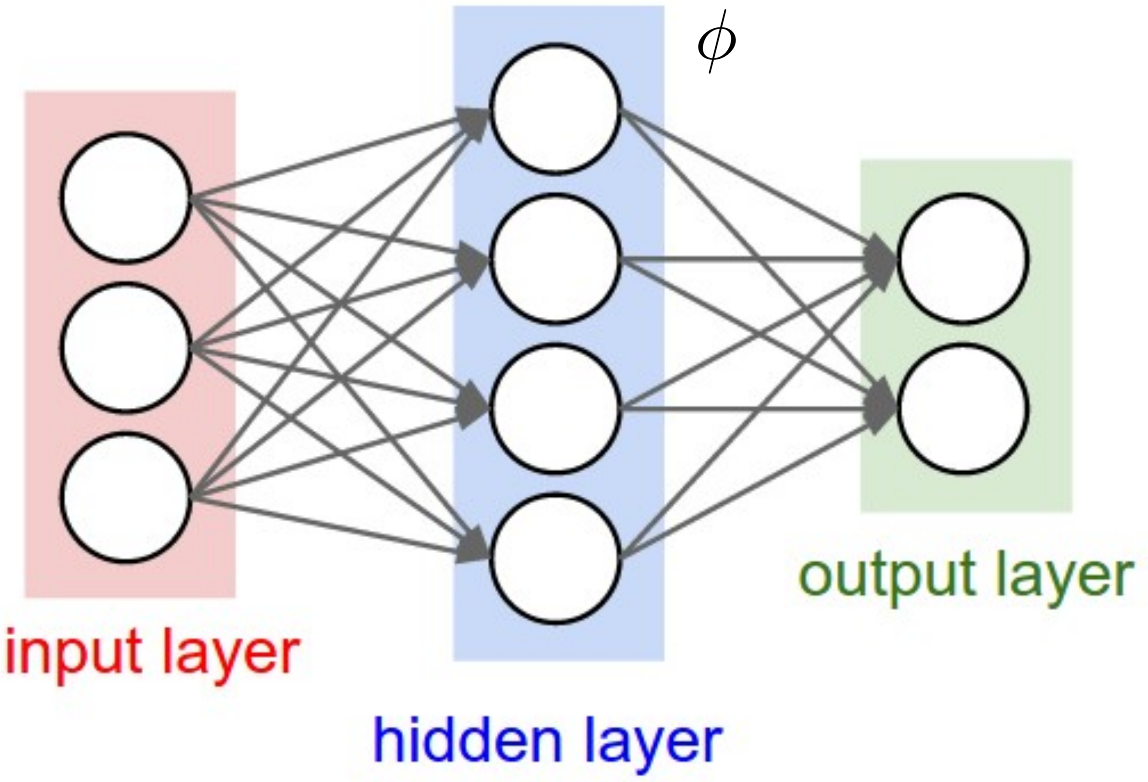
$$f = \mathbf{W}\phi(\mathbf{x}; \boldsymbol{\theta})$$

kernel

parameters

From the broad family of functions ϕ we learn the best representation by learning the parameters $\boldsymbol{\theta}$

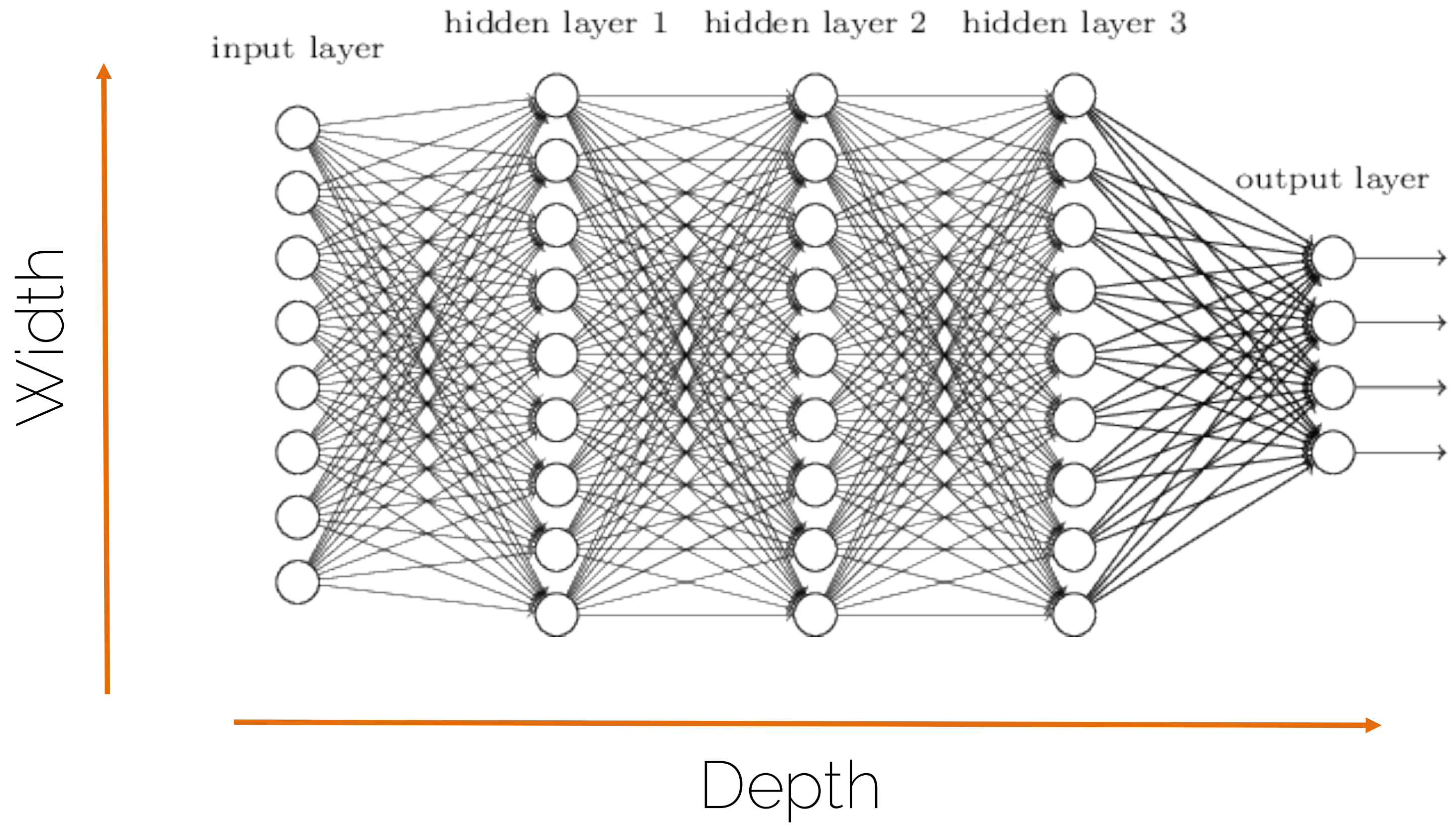
Neural Network



Also SVM is in this category

Credit: Li/Karpathy/Johnson

Neural Network



Neural Network

- Linear score function $f = Wx$

- Neural network is a nesting of 'functions'
  - 2-layers: $f = W_2 \max(0, W_1 x)$
  - 3-layers: $f = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 4-layers: $f = W_4 \tanh\ (\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x)))$
  - 5-layers: $f = W_5 \sigma(W_4 \tanh(\mathrm{W}_3, \max(0, W_2 \max(0, W_1 x))))$
  - … up to hundreds of layers

Playing around with networks

- http://cs.stanford.edu/people/karpathy/convnetjs/index.html

Neural Network

- Problems of going deeper…

- The impact of small decisions (architecture, activation functions…)

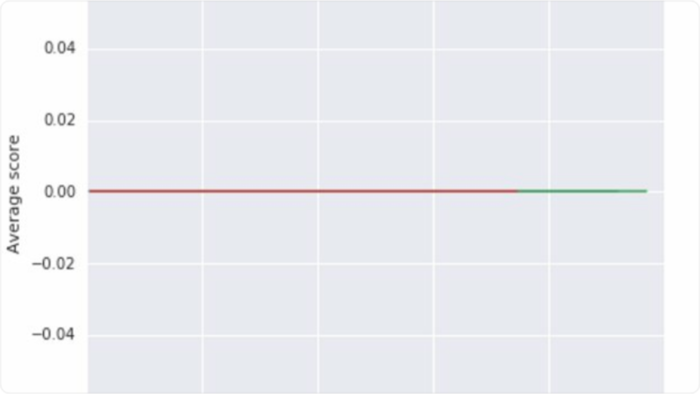
- Is my network training correctly?

A typical Deep Learner day

A Andrej Karpathy i Ian Goodfellow els agrada

Oriol Vinyals @OriolVinyalsML · 9h

A typical training curve in Montezuma's Revenge (note: there are several random seeds which overlap) 😫 #nips #rl #exploration

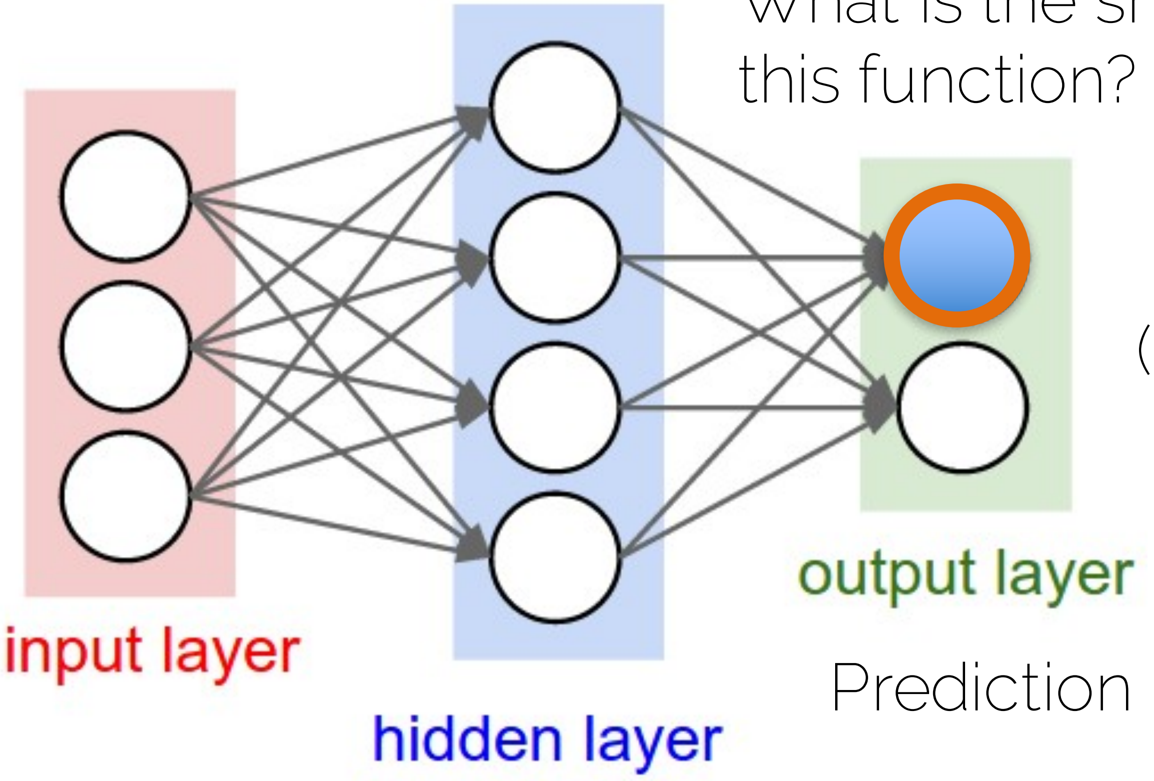


2 9 63

Output functions

Neural networks

What is the shape of this function?

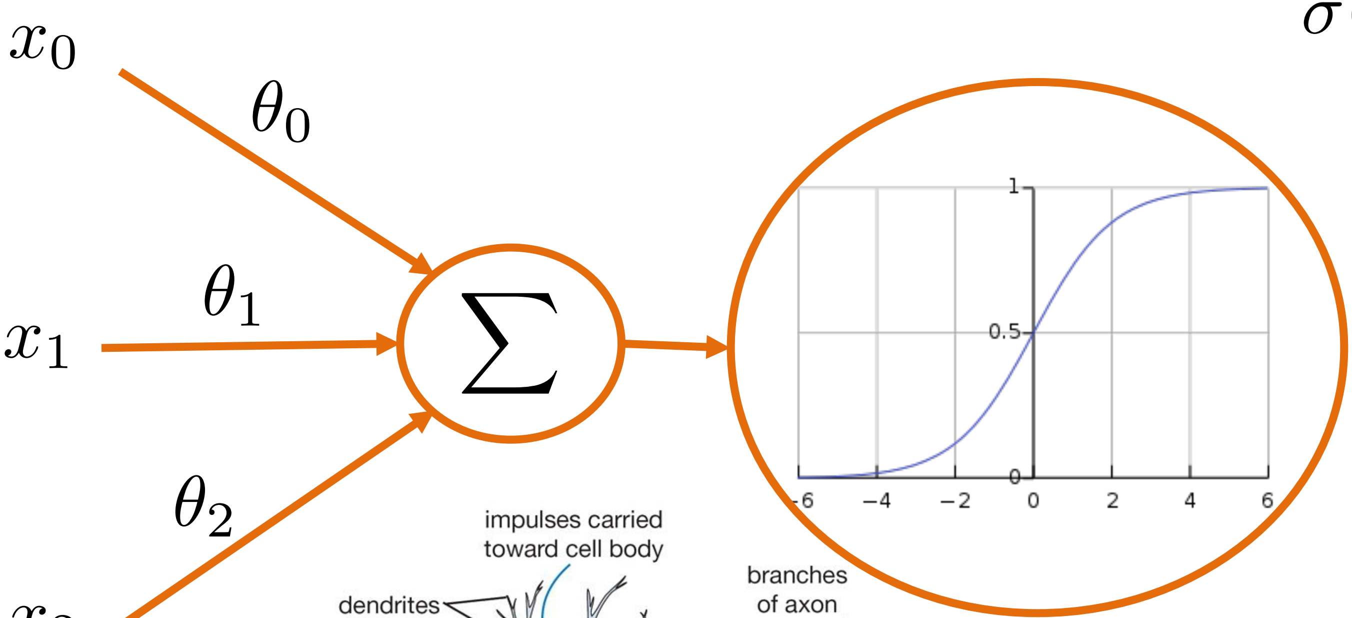


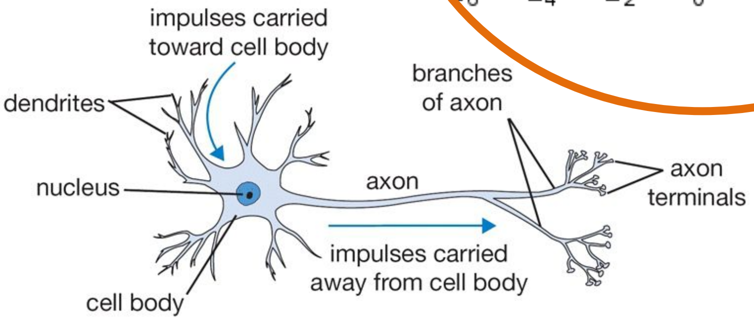
Loss (Softmax, Hinge)

Prediction

Sigmoid for binary predictions

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





1

Can be interpreted as a probability

0

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

Logistic regression

- Probability of a binary output

$$p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \prod_{i=1}^{n} \mathrm{Ber}(y_i|\mathrm{sigm}(\mathbf{x}_i,\boldsymbol{\theta}))$$

Model for coins

$$p(x|\phi) = \phi^x(1-\phi)^{1-x} = \begin{cases} \phi & \text{if} \quad x = 1 \\ 1-\phi & \text{if} \quad x = 0 \end{cases}$$

Logistic regression

- Probability of a binary output

$$p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = \prod_{i=1}^{n} \mathrm{Ber}(y_i|\mathrm{sigm}(\mathbf{x}_i,\boldsymbol{\theta}))$$

$$= \prod_{i=1}^{n} \underbrace{\left[\frac{1}{1+e^{-\mathbf{x}_i\boldsymbol{\theta}}}\right]^{y_i}}_{\Pi_i} \left[1 - \underbrace{\frac{1}{1+e^{-\mathbf{x}_i\boldsymbol{\theta}}}}_{\Pi_i}\right]^{1-y_i}$$

$$p(x|\phi) = \phi^x(1-\phi)^{1-x}$$

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

Logistic regression

- Probability of a binary output

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{n} [\Pi_i]^{y_i} \, [1 - \Pi_i]^{1-y_i}$$

- Maximum Likelihood

$$\boldsymbol{\theta}_{ML} = \arg\max_{\theta} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$$

Logistic regression

- Probability of a binary output

$$\Pi_i = \frac{1}{1 + e^{-\mathbf{x}_i \boldsymbol{\theta}}}$$

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{n} [\Pi_i]^{y_i} [1 - \Pi_i]^{1-y_i}$$

$$\begin{aligned} C(\boldsymbol{\theta}) &= -\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \\ &= -\sum_{i=1}^{n} y_i \log(\Pi_i) + (1 - y_i) \log(1 - \Pi_i) \end{aligned}$$

Referred to as cross-entropy

Logistic regression

- Optimize using gradient descent

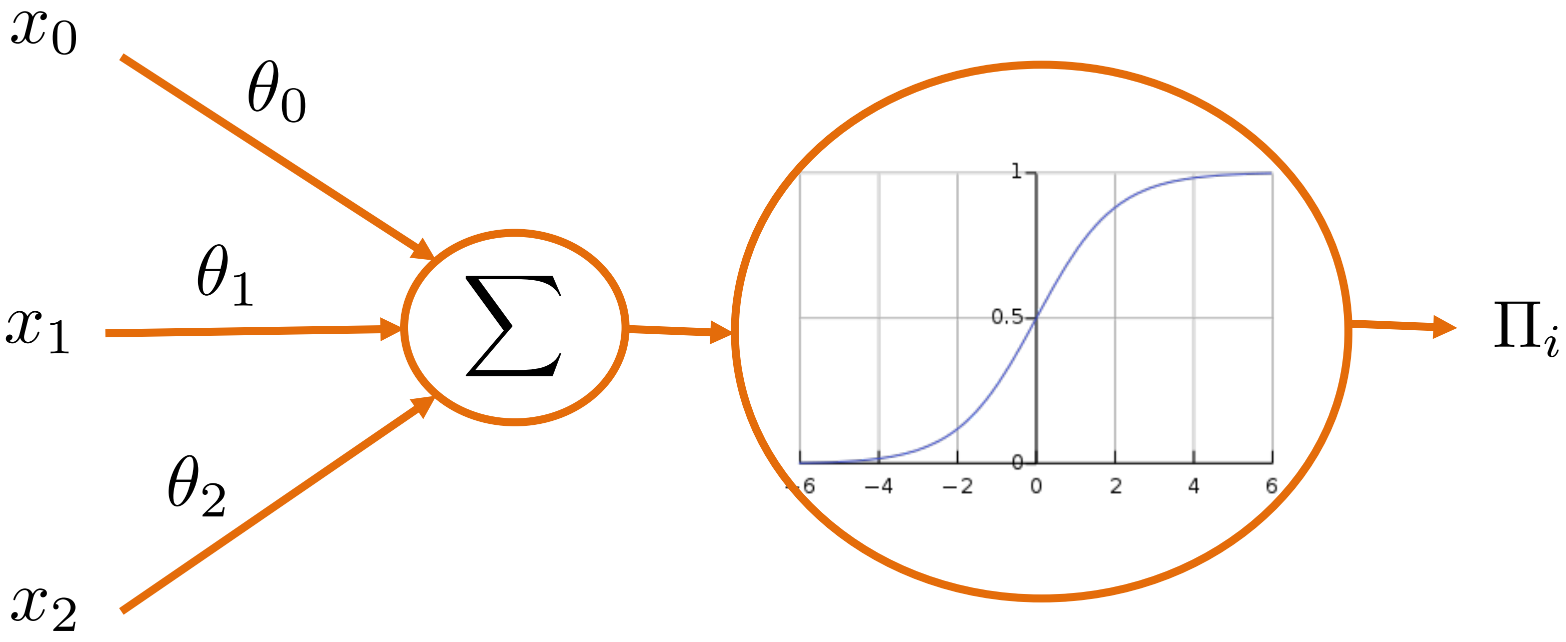
- Saturation occurs only when the model already has the right answer

$$C(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \log(\Pi_i) + (1 - y_i)\log(1 - \Pi_i)$$

Referred to as cross-entropy

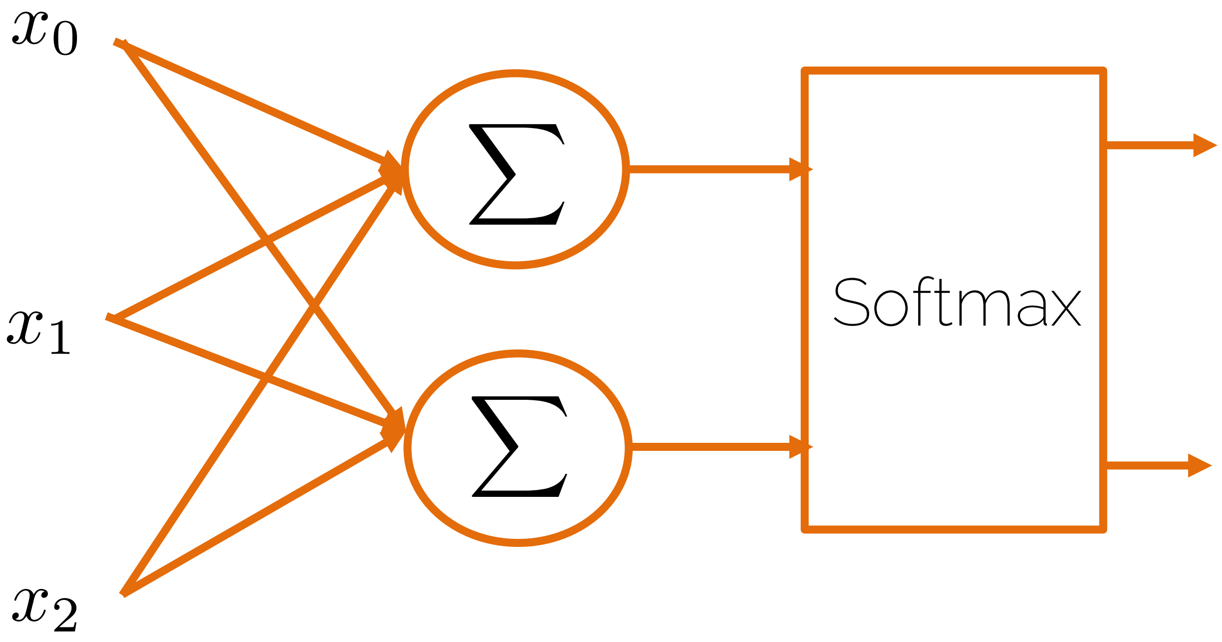
Softmax formulation

- What if we have multiple classes?



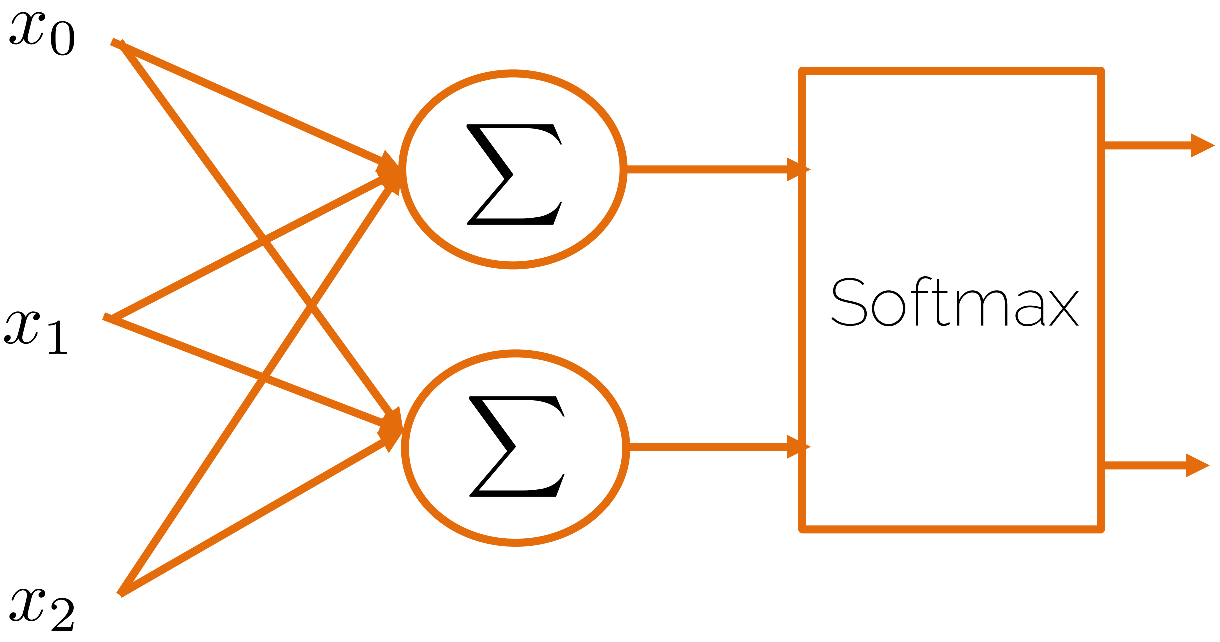
Softmax formulation

- What if we have multiple classes?



Softmax formulation

- What if we have multiple classes?



$$\Pi_1 = \frac{e^{\mathbf{x}_i \boldsymbol{\theta}_1}}{e^{\mathbf{x}_i \boldsymbol{\theta}_1} + e^{\mathbf{x}_i \boldsymbol{\theta}_2}}$$

$$\Pi_2 = \frac{e^{\mathbf{x}_i \boldsymbol{\theta}_2}}{e^{\mathbf{x}_i \boldsymbol{\theta}_1} + e^{\mathbf{x}_i \boldsymbol{\theta}_2}}$$

Softmax formulation

- Softmax

exp

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

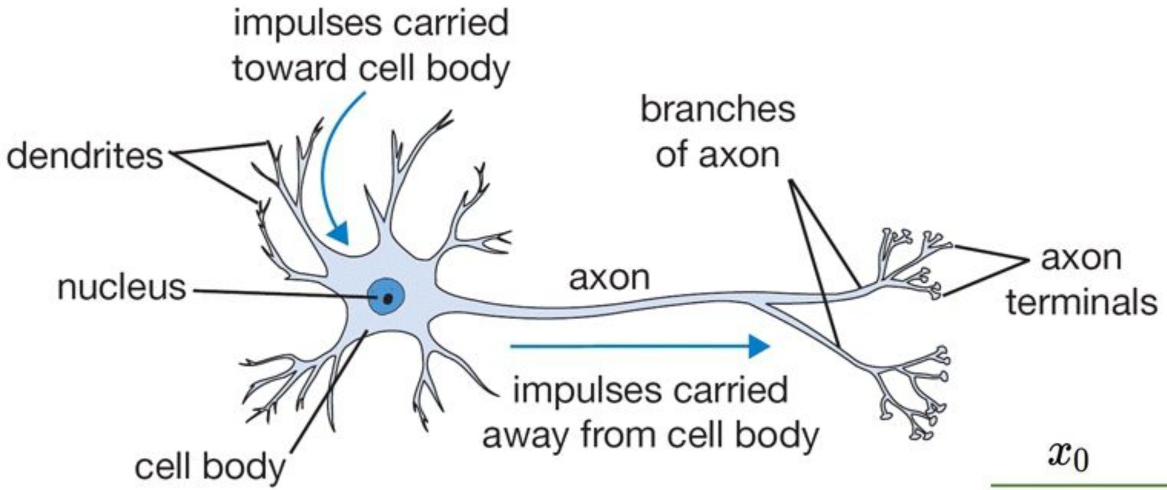
normalize

- Softmax loss (ML)

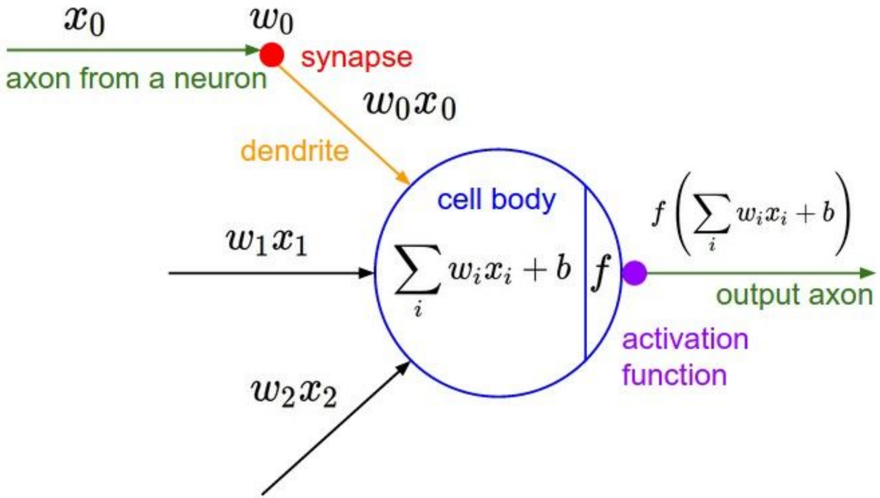
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Activation functions

Neurons

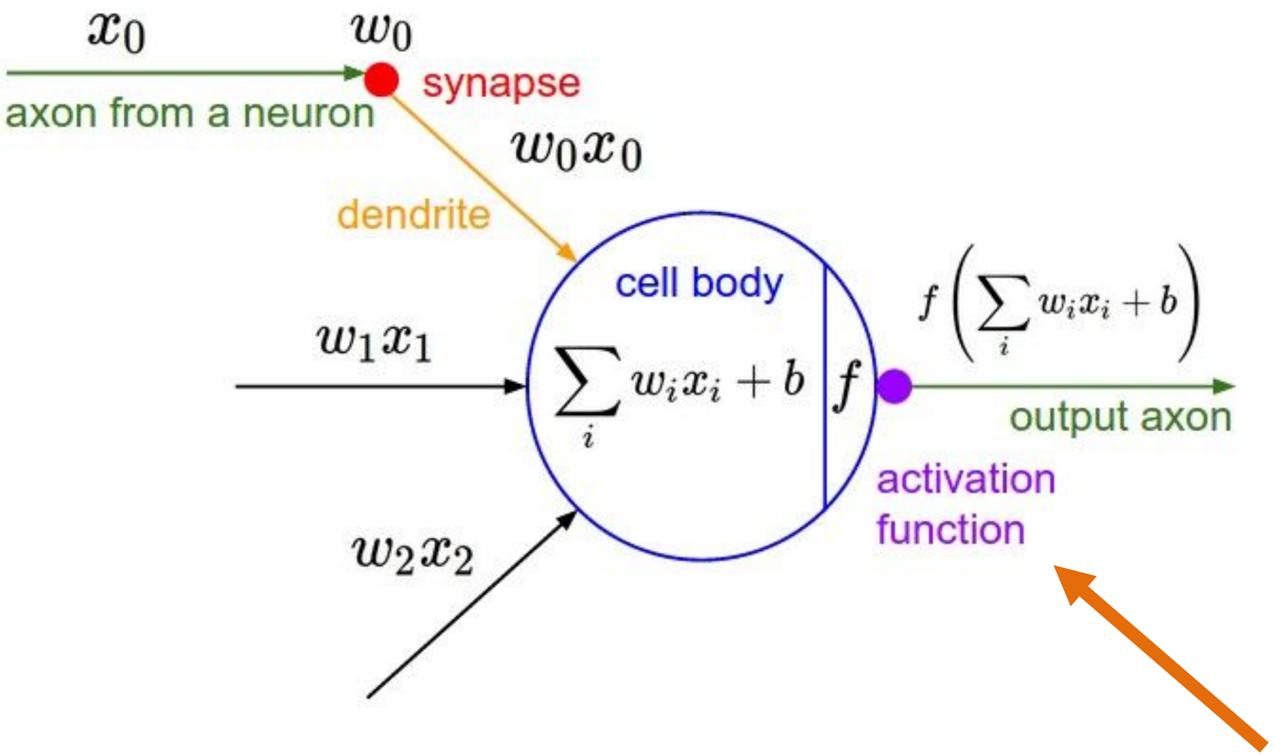


impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body



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

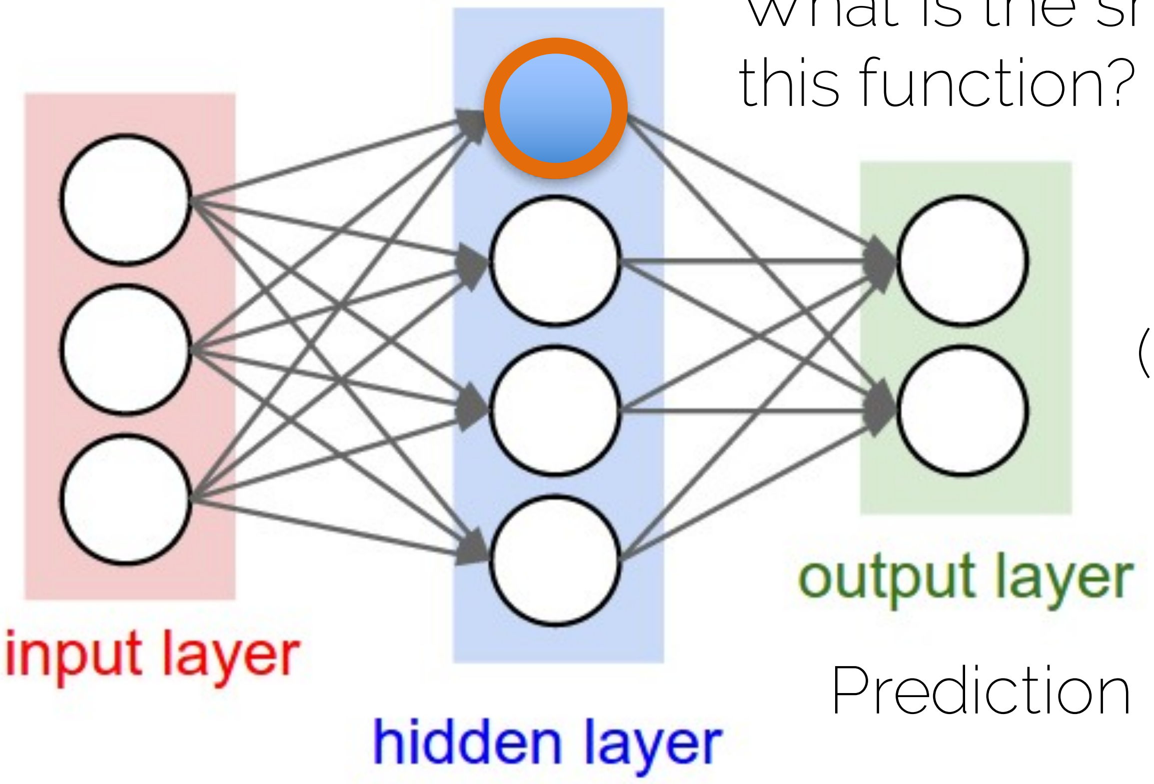
Neurons



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

Neural networks

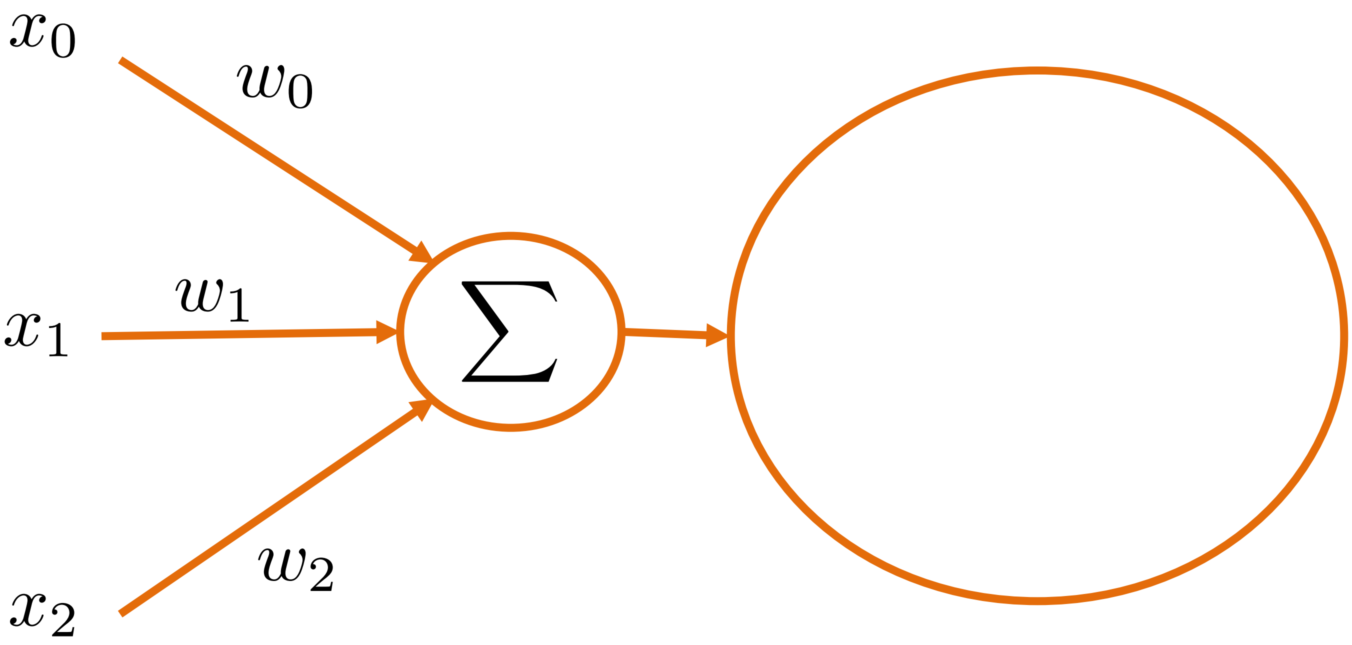
What is the shape of this function?



Loss (Softmax, Hinge)

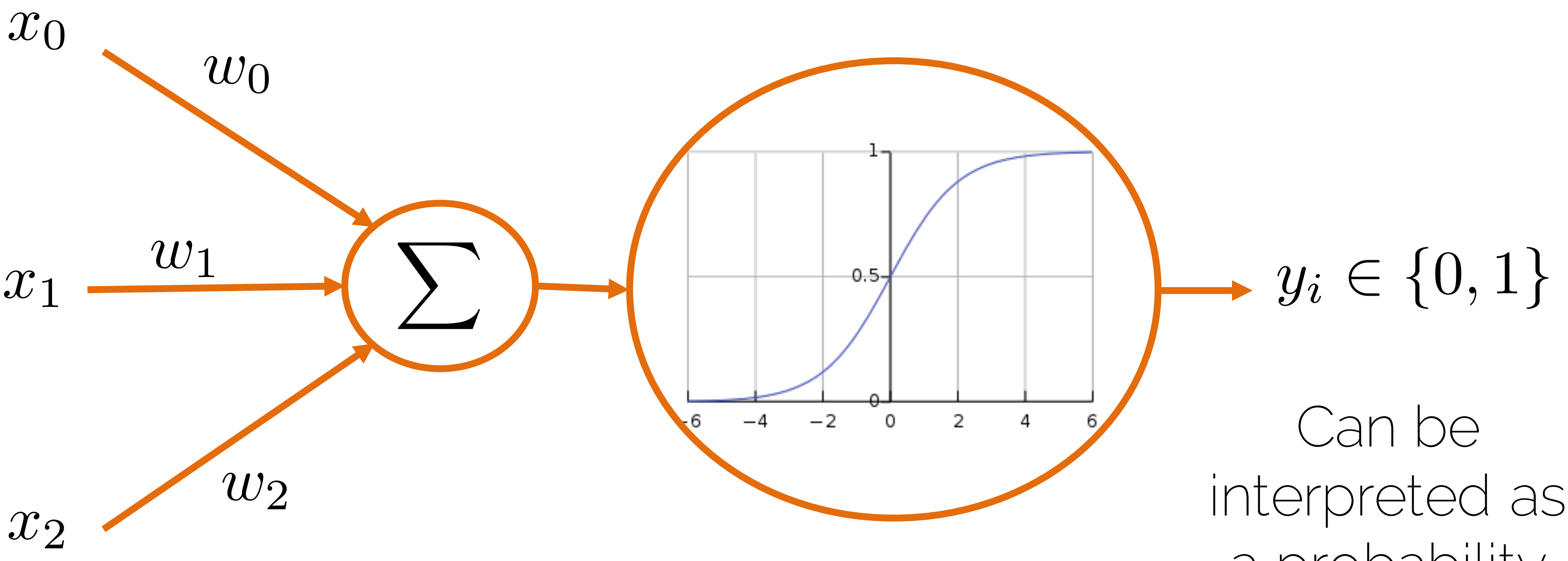
Prediction

Activation functions or hidden units



Sigmoid

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

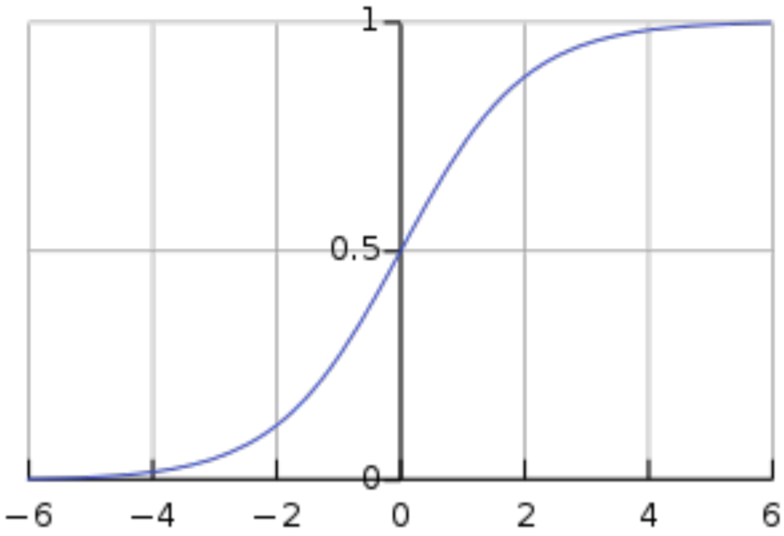


Can be interpreted as a probability

Sigmoid

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

Forward

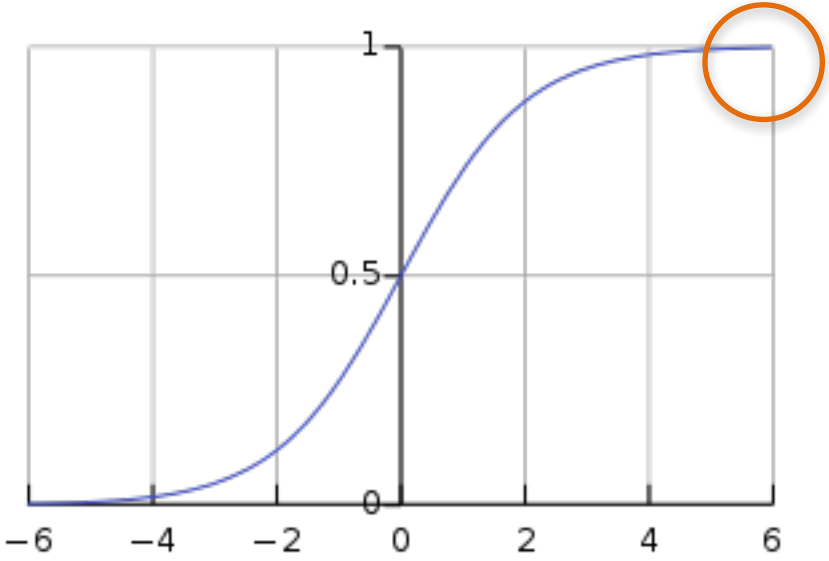


$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma} \qquad \frac{\partial \sigma}{\partial x} \qquad \frac{\partial L}{\partial \sigma}$$

Sigmoid

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

Forward



$$x = 6$$

Saturated neurons kill the gradient flow

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

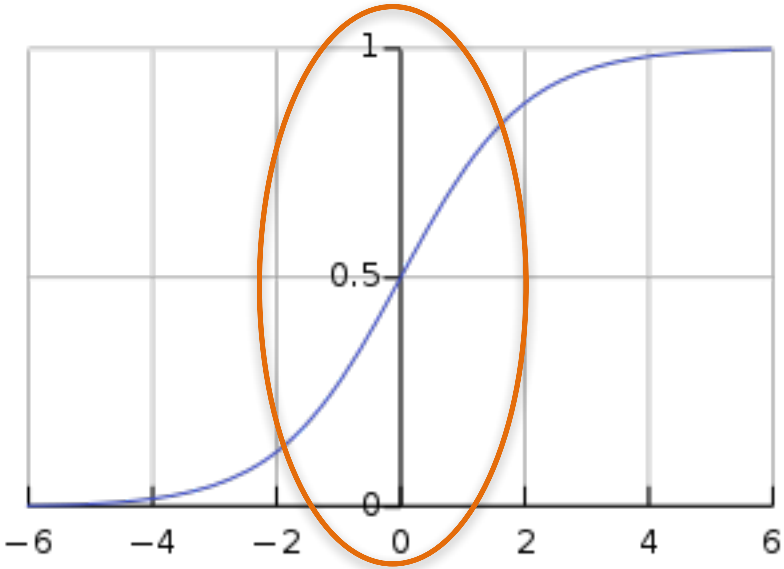
$$\frac{\partial \sigma}{\partial x}$$

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

Sigmoid

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

Forward

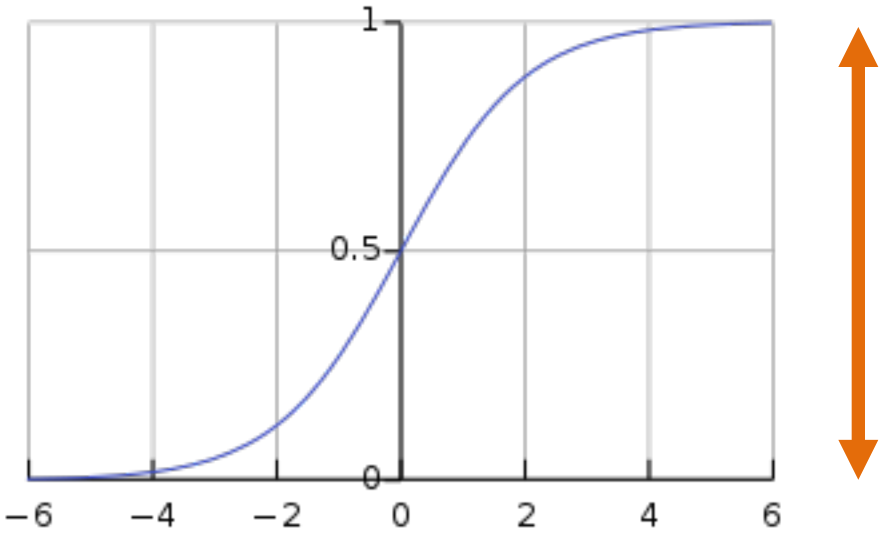


Active region for gradient descent

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

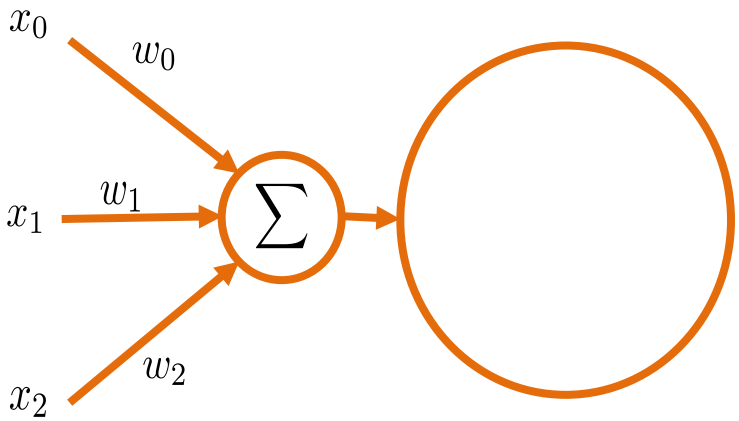
Sigmoid

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



Output is always positive

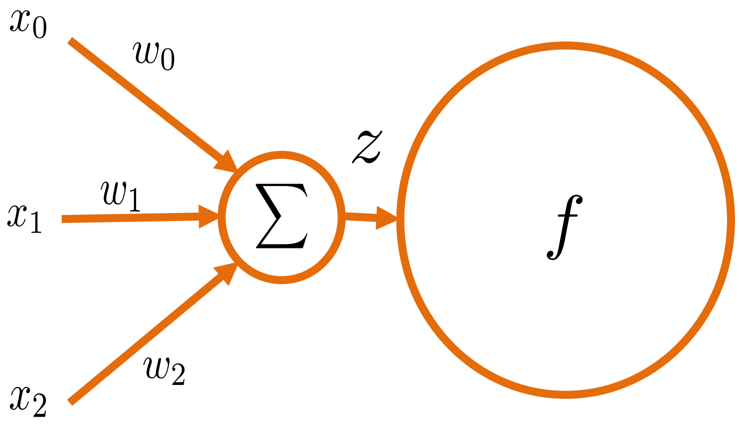
Problem of positive output



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

We want to compute the gradient wrt the weights

Problem of positive output

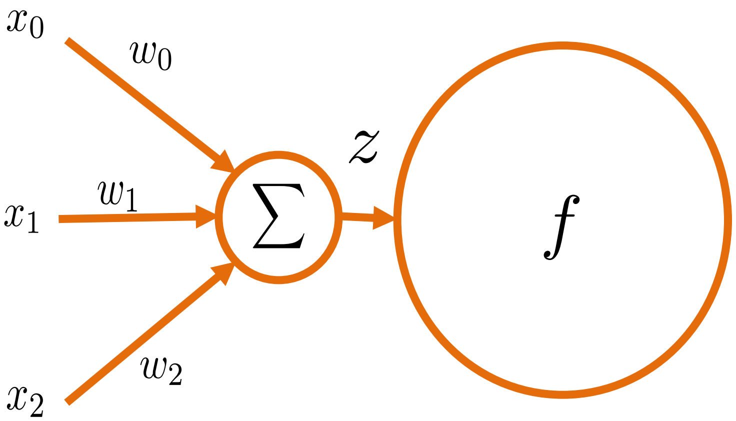


$$f\left(\underbrace{\sum_i w_i x_i + b}\right)$$

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

We want to compute the gradient wrt the weights

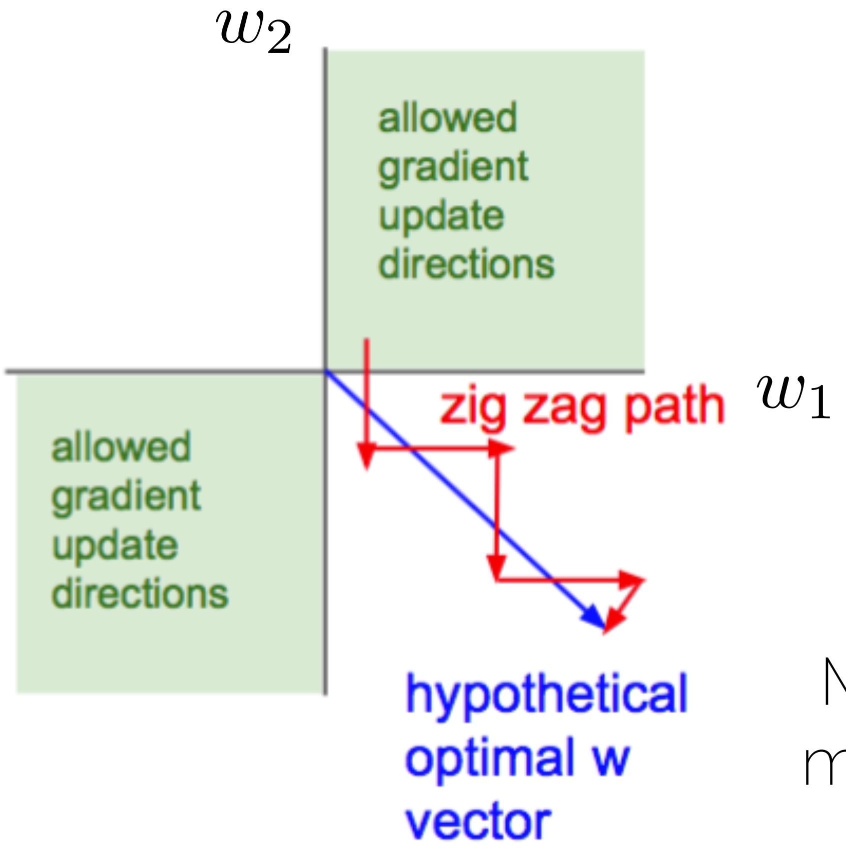
Problem of positive output



$$\frac{\partial f}{\partial z} \quad f\Bigg(\underbrace{\sum_i w_i x_i + b}_{\frac{\partial z}{\partial w} = x_i > 0}\Bigg)$$

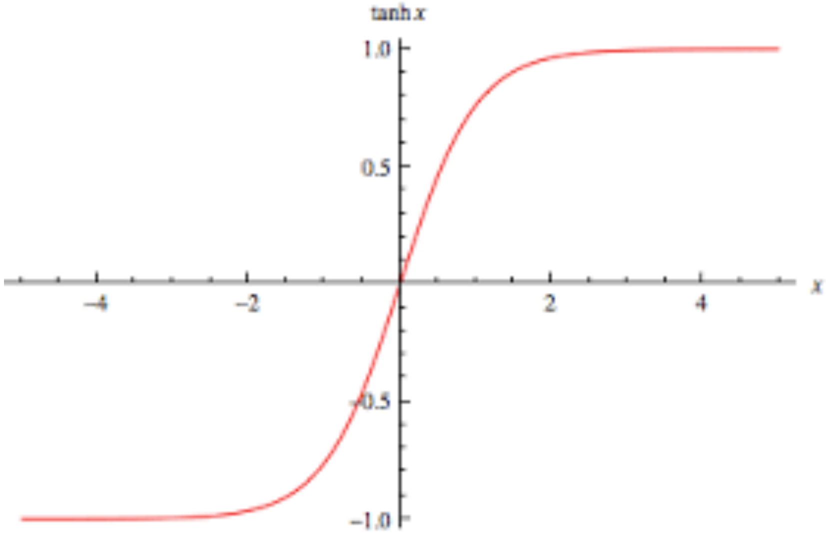
It is going to be either positive or negative for all weights

Problem of positive output



More on zero-mean data later

tanh



✗ Still saturates

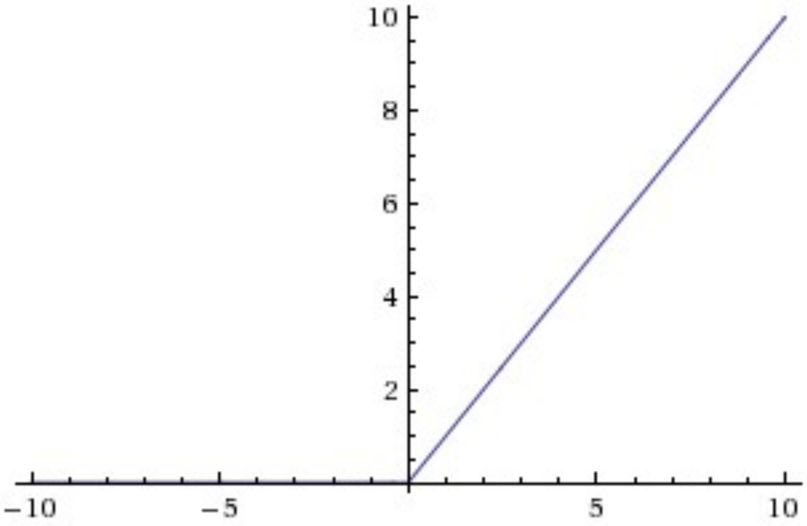
✓ Zero-centered

✗ Still saturates

LeCun 1991

Rectified Linear Units (ReLU)

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



Large and consistent gradients ✓

✓ Fast convergence

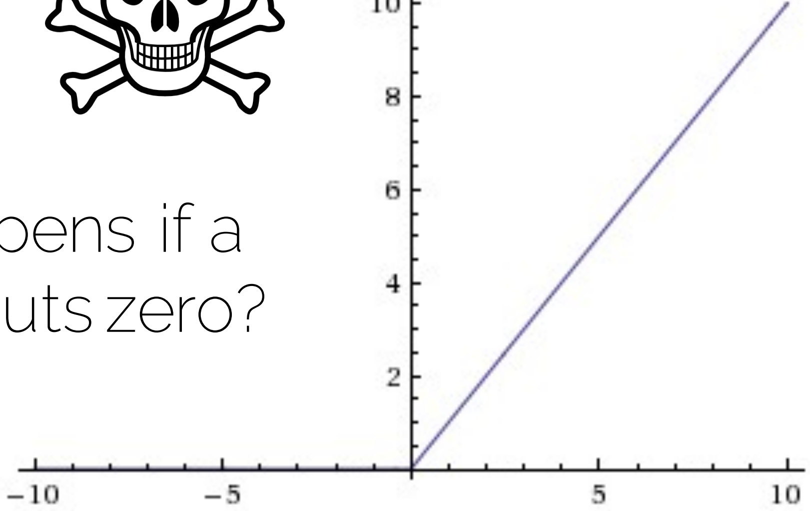
✓ Does not saturate

Krizhevsky 2012

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?



Large and consistent gradients ✓

✓ Fast convergence

✓ Does not saturate

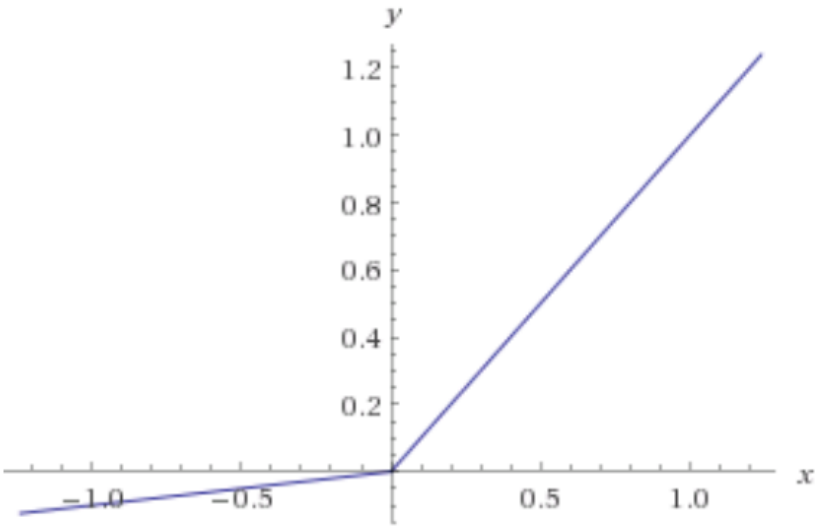
Rectified Linear Units (ReLU)

- 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

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



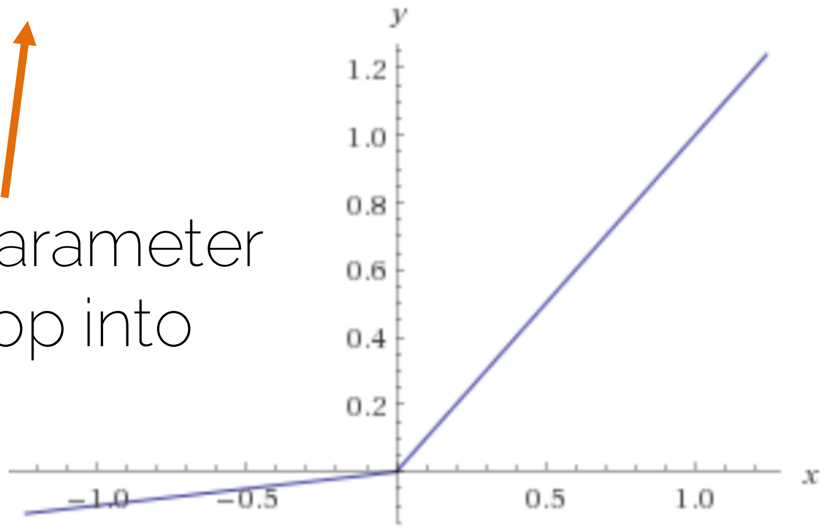
✓ Does not die

Mass 2013

Parametric ReLU

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

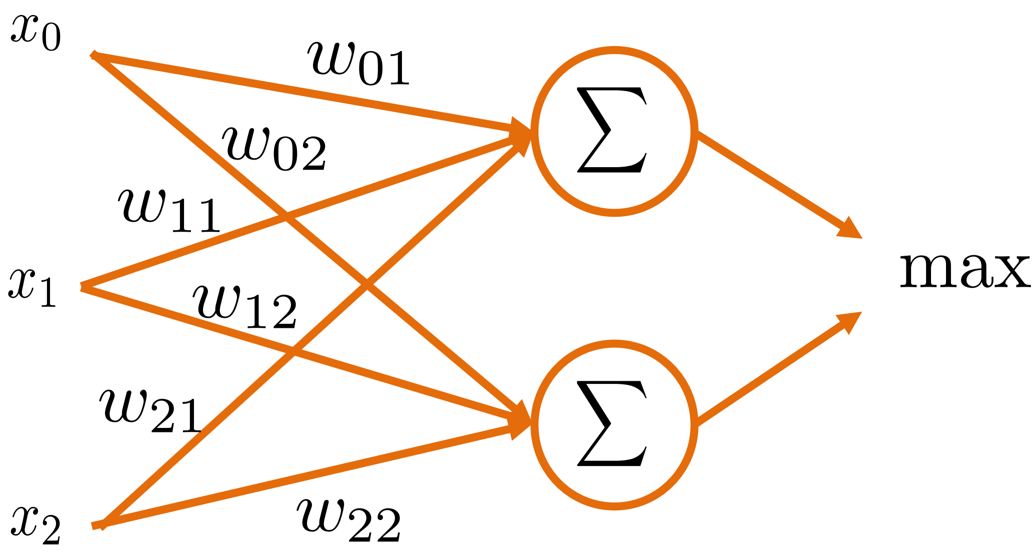
One more parameter
to backprop into



✓ Does not die

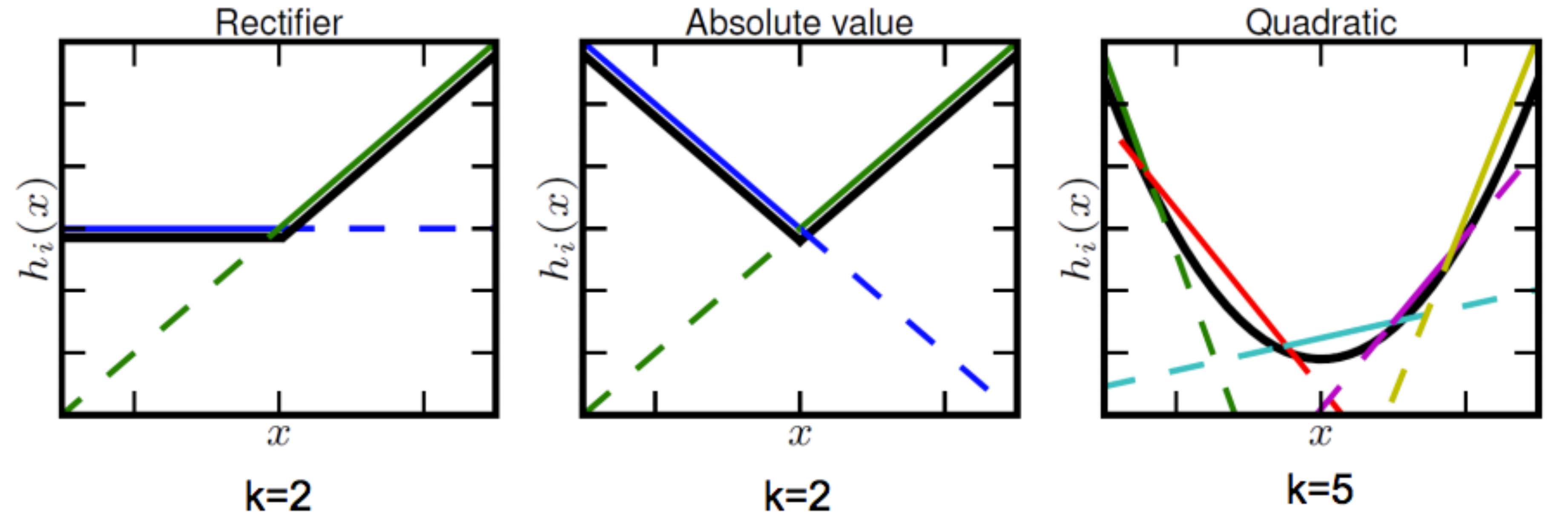
He 2015

Maxout units



Goodfellow 2013

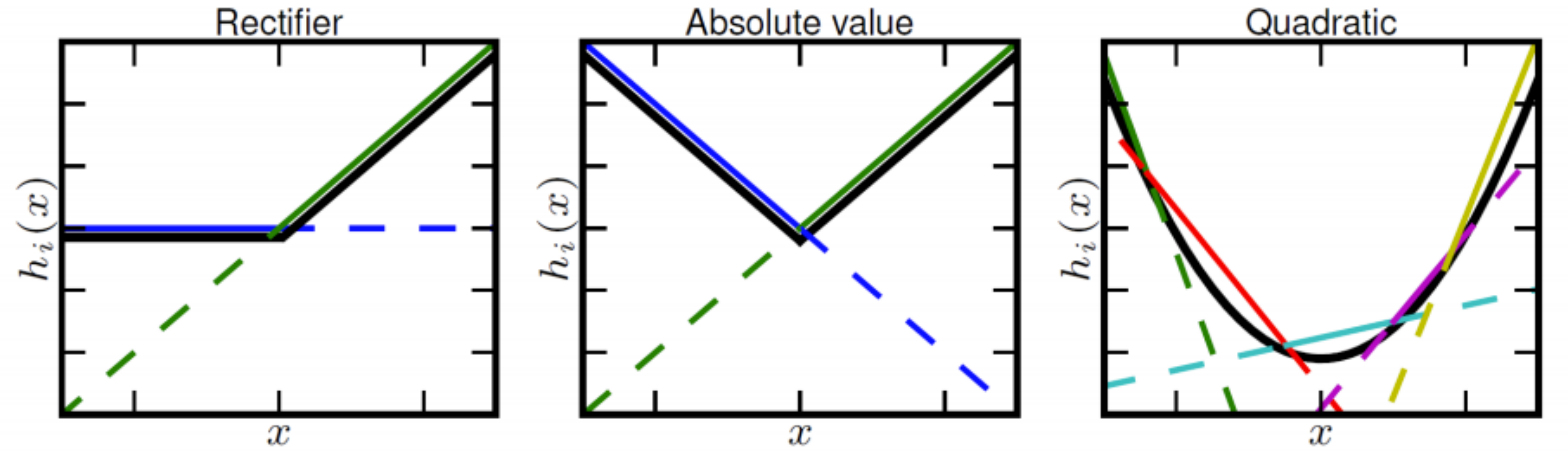
Maxout units



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

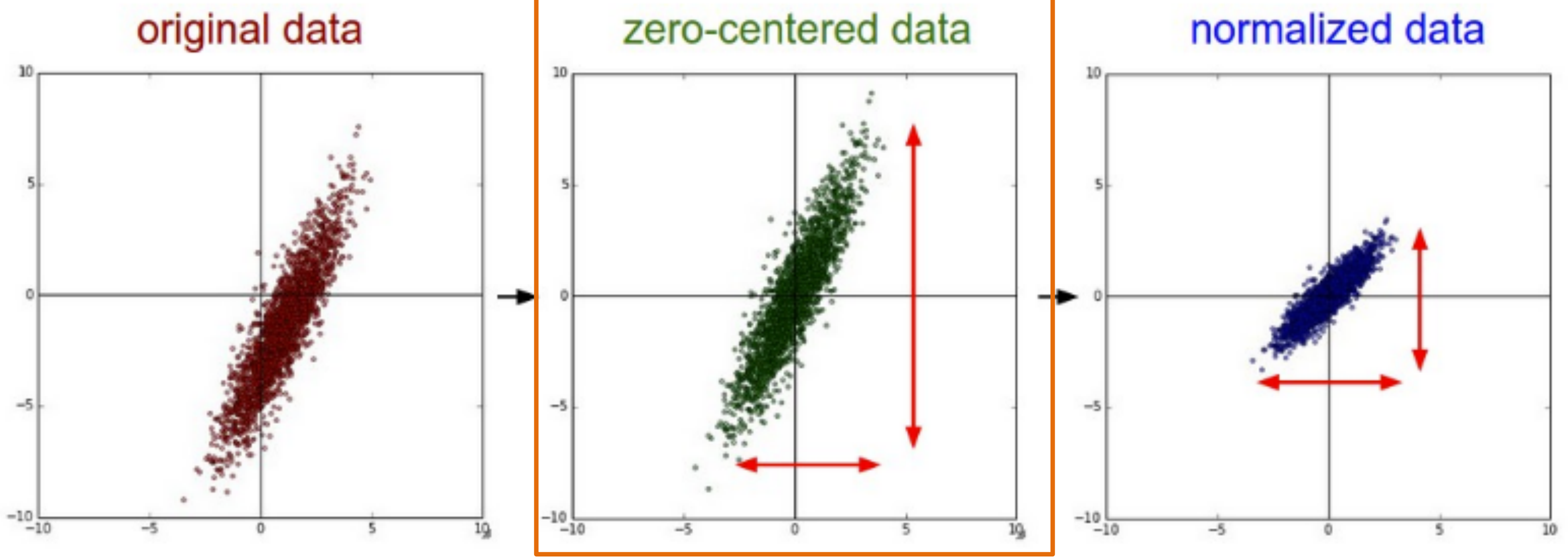
✗ Increase of the number of parameters

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

A quick word on data pre-processing

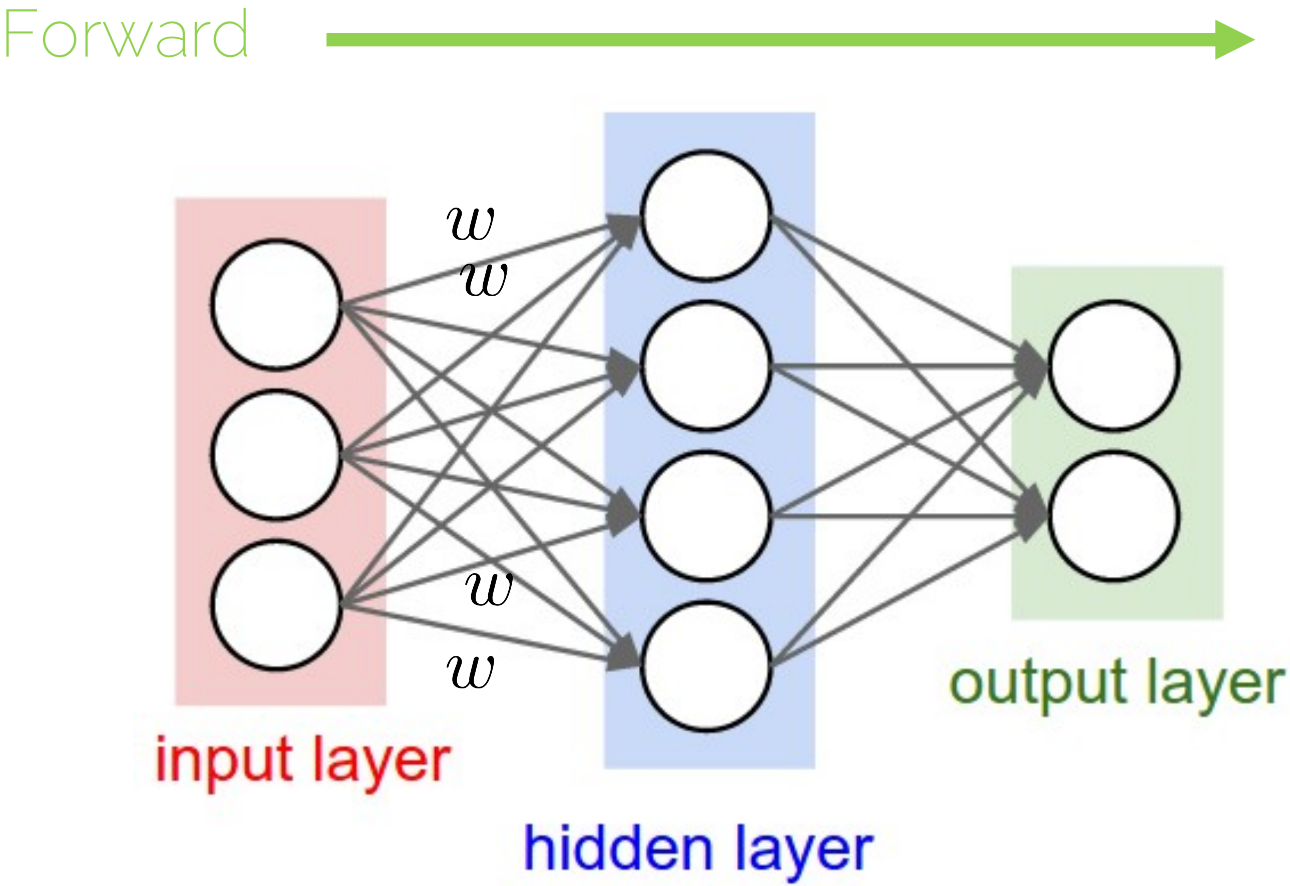
Data pre-processing



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

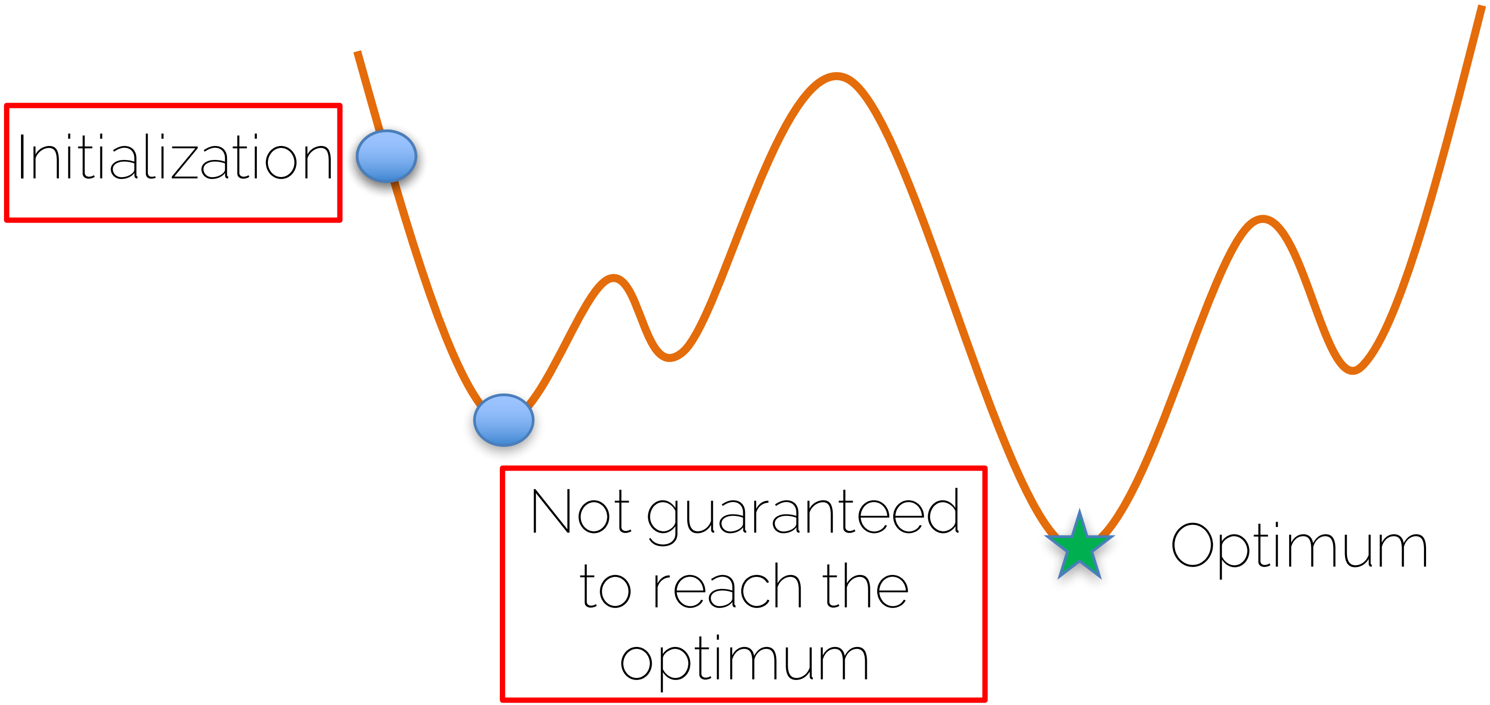
Weight initialization

How do I start?



Initialization is extremely important

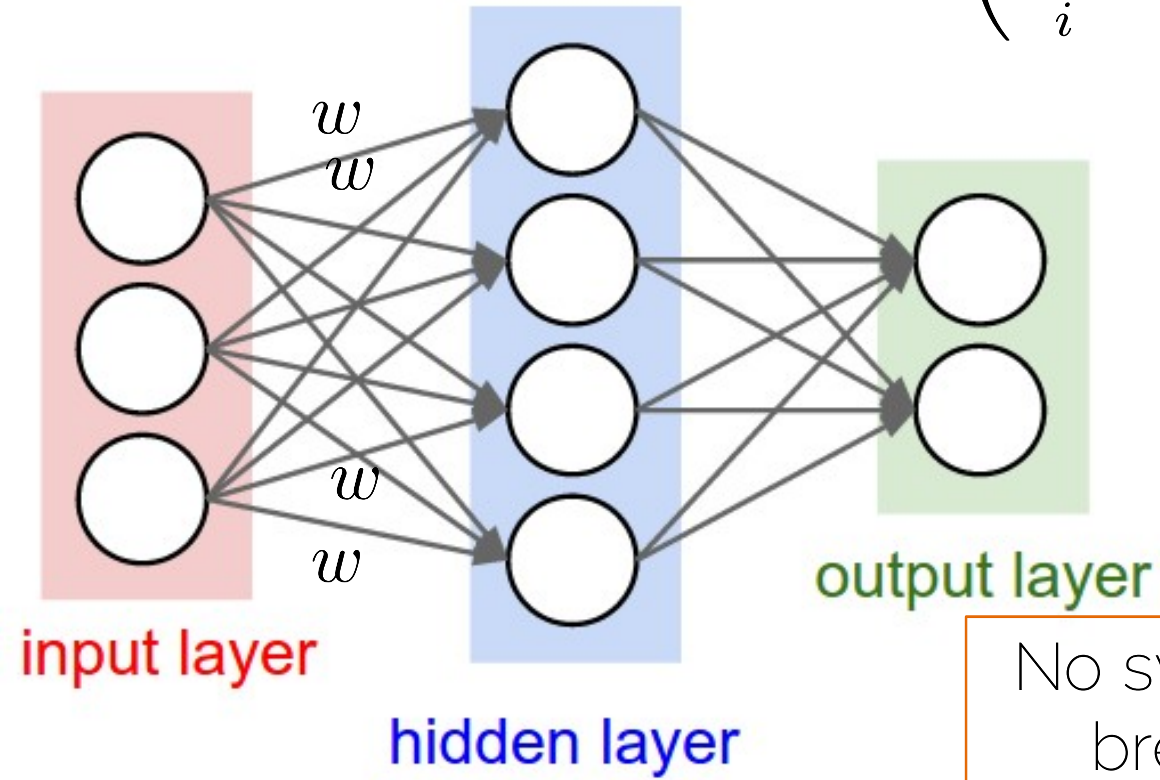
$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$



How do I start?

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

$w = 0$



What happens to the gradients?

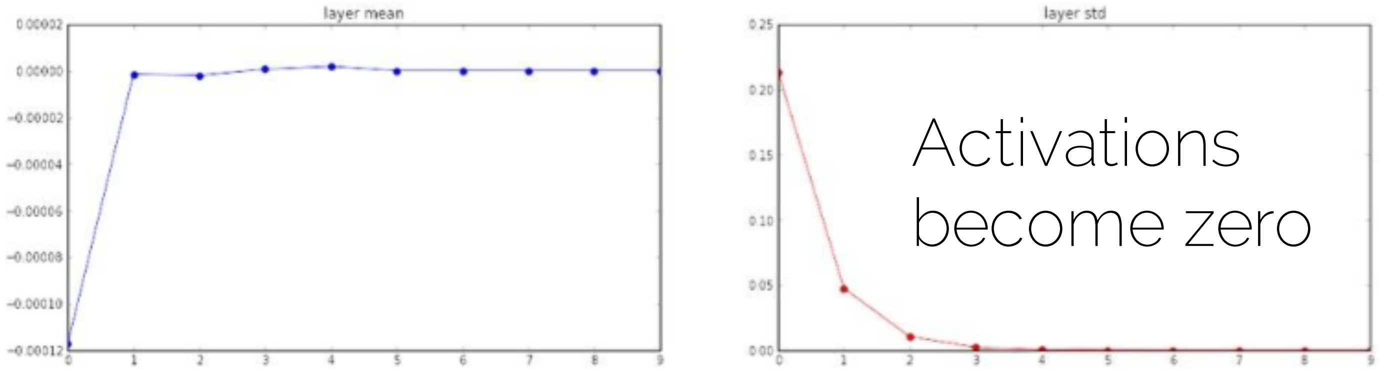
No symmetry breaking

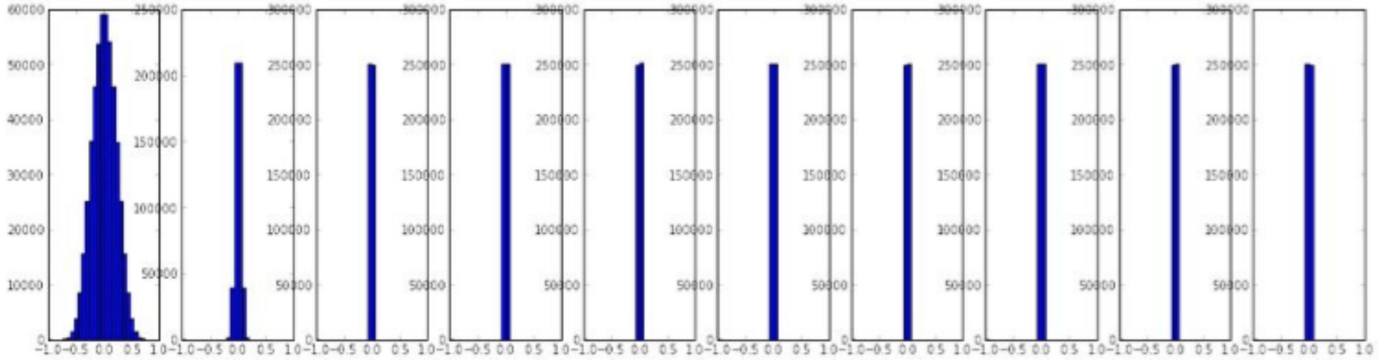
Small random numbers

- Gaussian with zero mean and standard deviation 0.01

- Let us see what happens:
  - Network with 10 layers with 500 neurons each
  - Tanh as activation functions
  - Input unit Gaussian data

Small random numbers



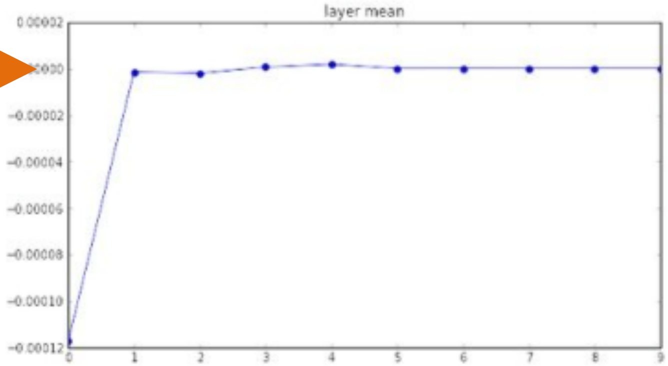


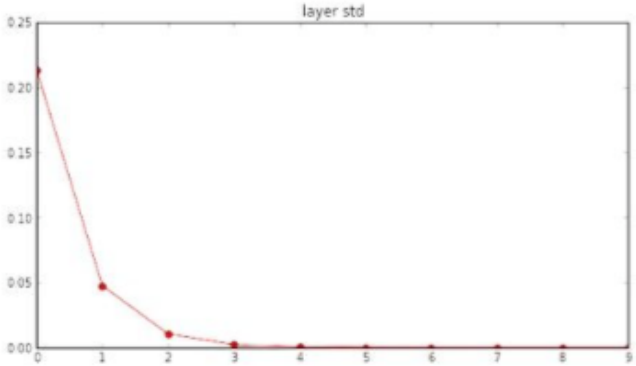
Input

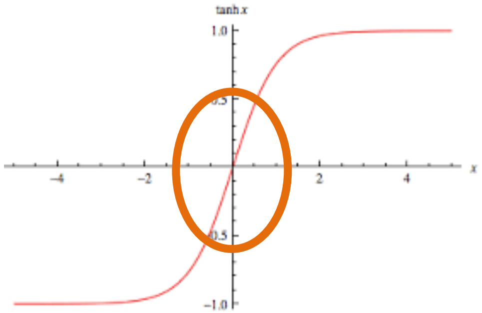
Last layer

Forward

Small random numbers





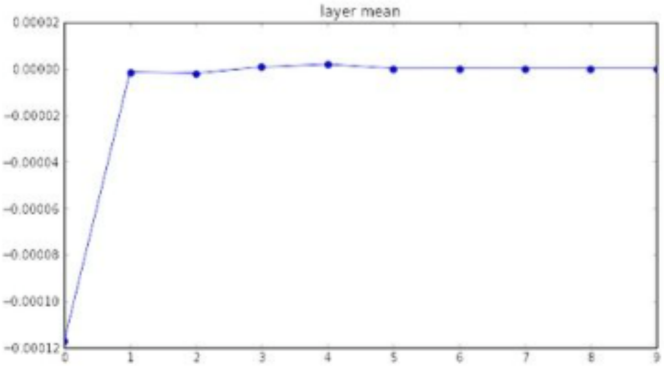


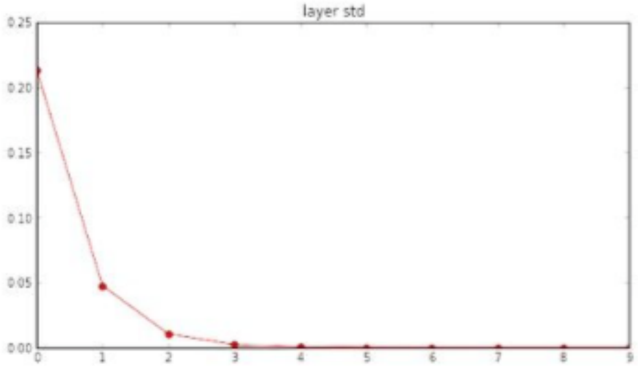
small

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

Forward

Small random numbers





Gradients vanish

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

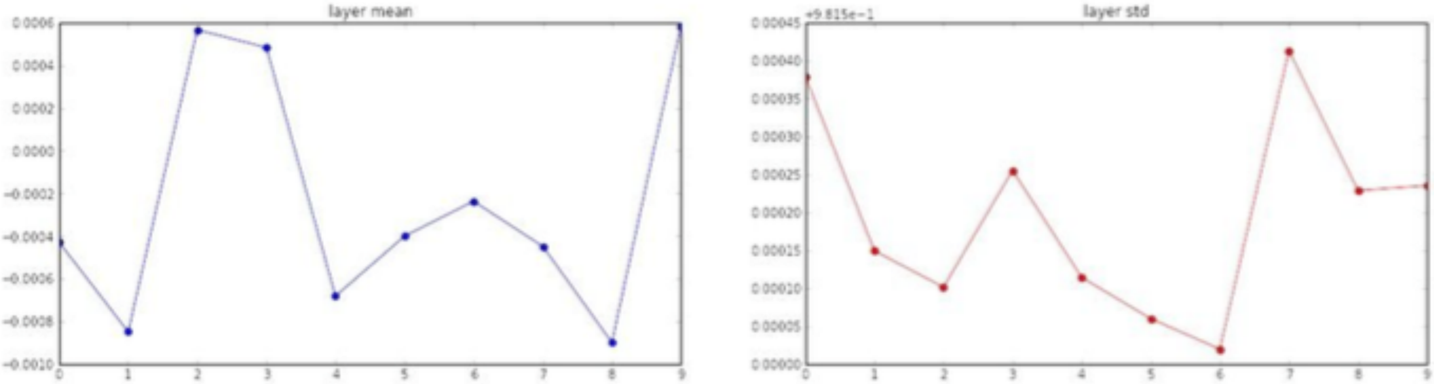
Backward

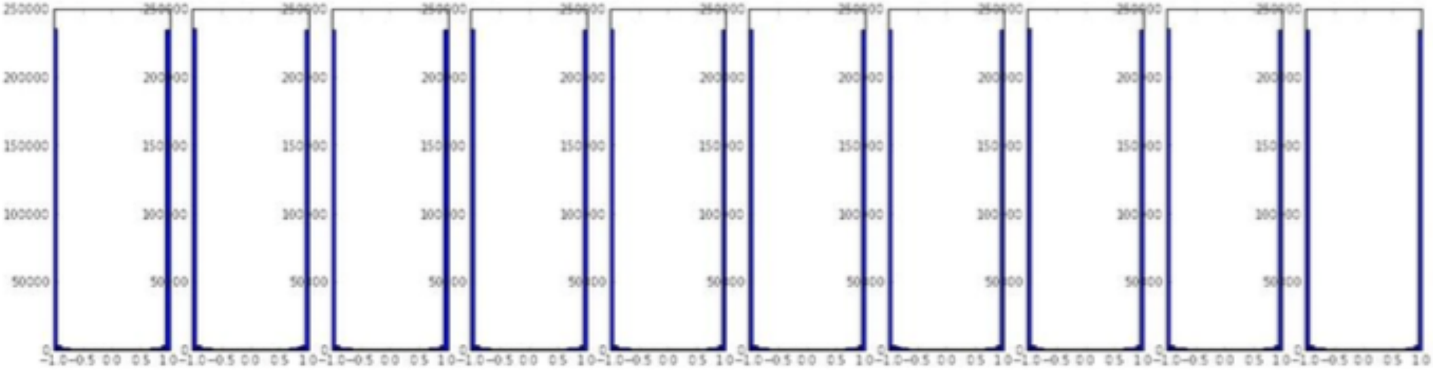
Big random numbers

- Gaussian with zero mean and standard deviation 1

- Let us see what happens:
  - Network with 10 layers with 500 neurons each
  - Tanh as activation functions
  - Input unit Gaussian data

Big random numbers





Everything is saturated

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_{i}^{n} w_i x_i) = \sum_{i}^{n} \mathrm{Var}(w_i x_i)$$

Glorot 2010

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right) = \sum_i^n \mathrm{Var}(w_i x_i)$$

Independent

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

Zero mean

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_{i}^{n} w_i x_i\right) = \sum_{i}^{n} \mathrm{Var}(w_i x_i)$$

$$= \sum_{i}^{n} [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_{i}^{n} \mathrm{Var}(x_i)\mathrm{Var}(w_i) = (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

Identically distributed

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_i^n w_i x_i) = \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) = (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

Variance gets multiplied by the number of inputs

Xavier initialization

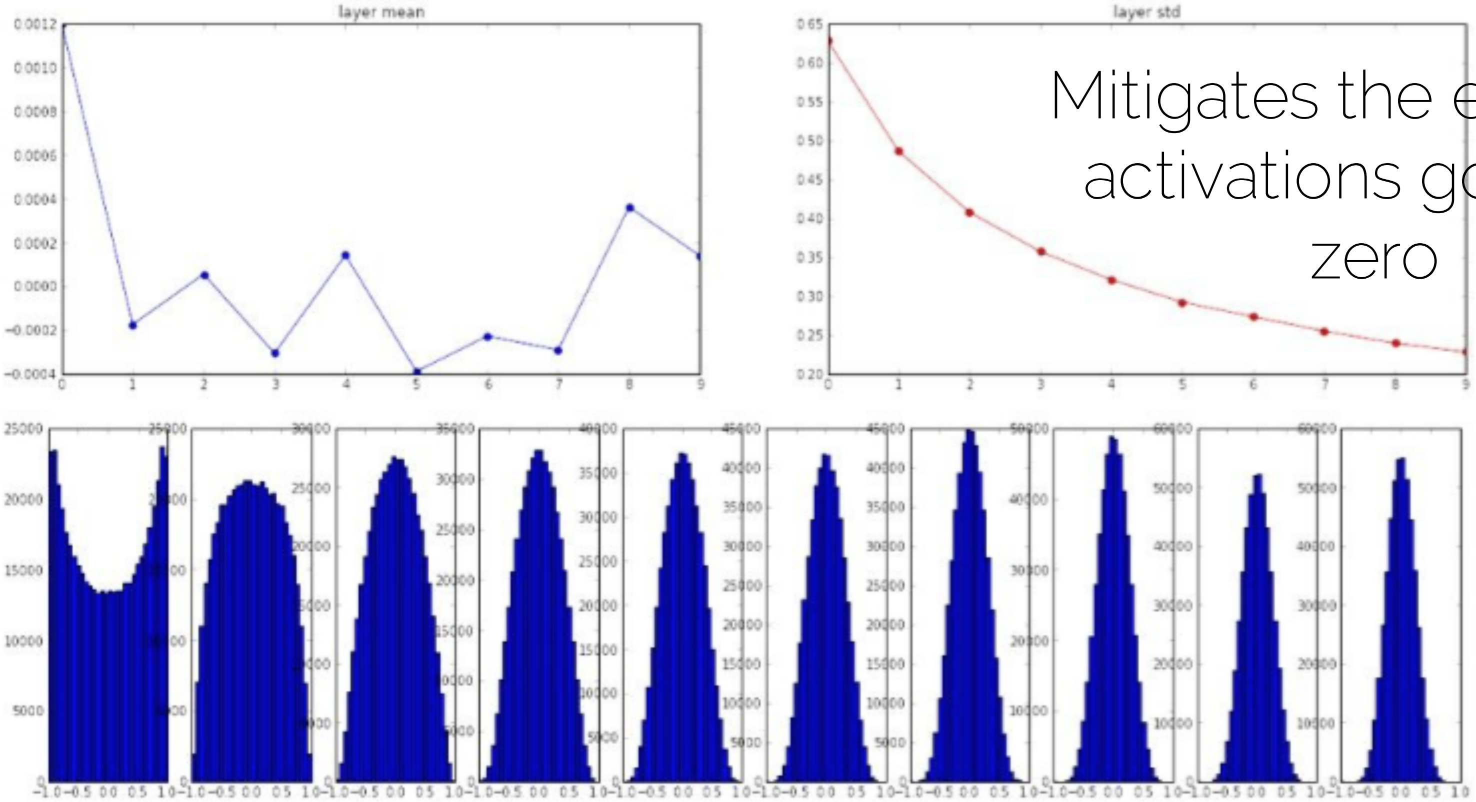
- How to ensure the variance of the output is the same as the input?

$$(n\mathrm{Var}(w))\mathrm{Var}(x)$$

1

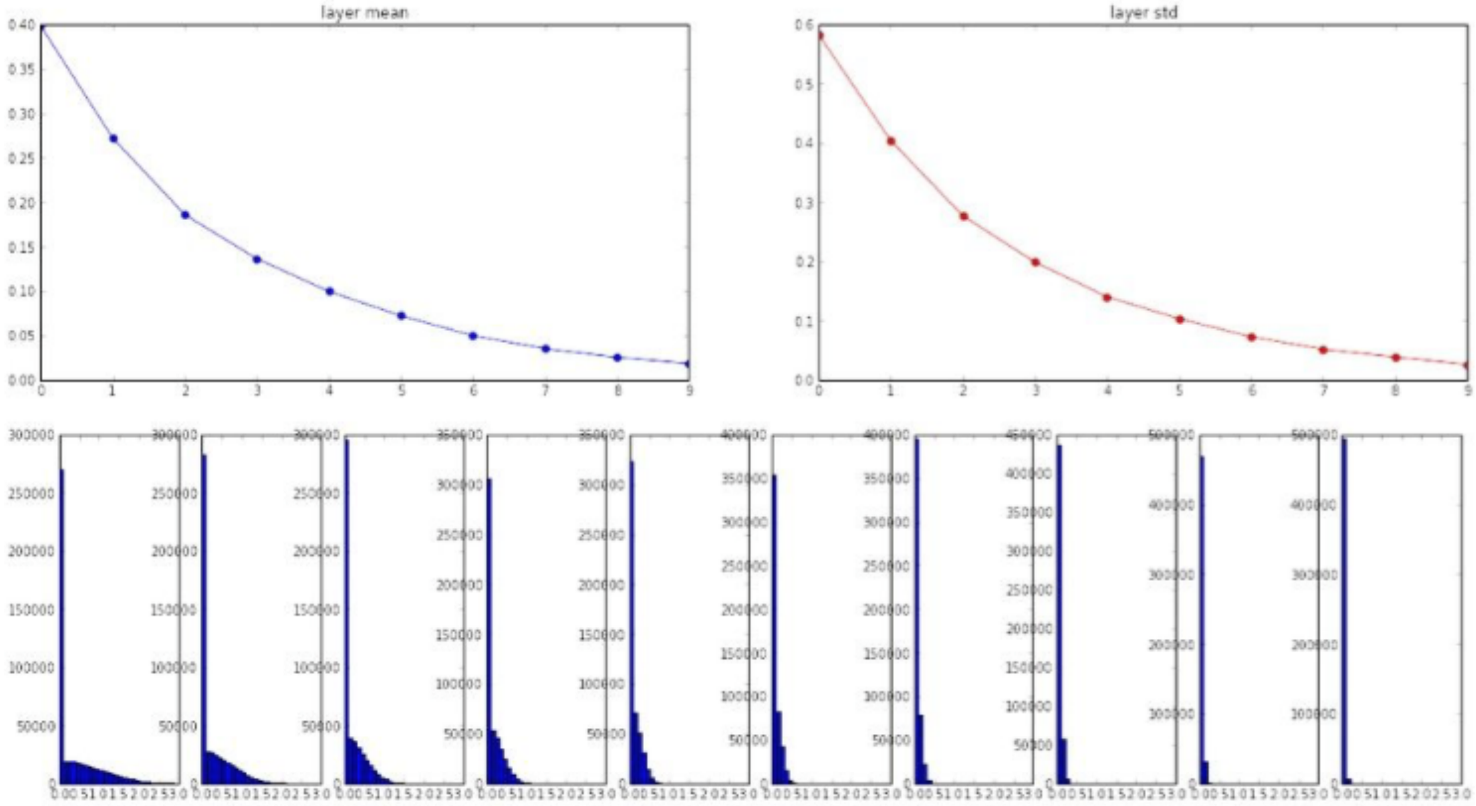
$$\boxed{Var(w) = \frac{1}{n}}$$

Xavier initialization



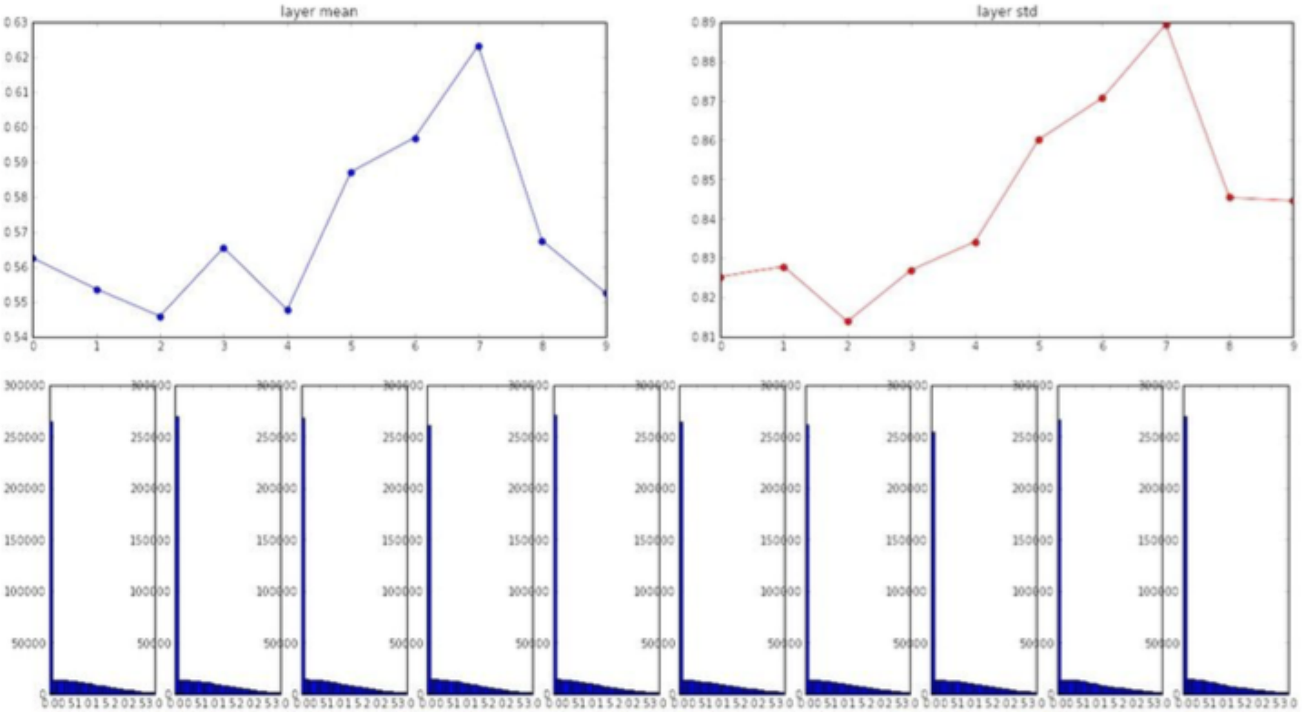
Mitigates the effect of activations going to zero

Xavier initialization with ReLU



ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

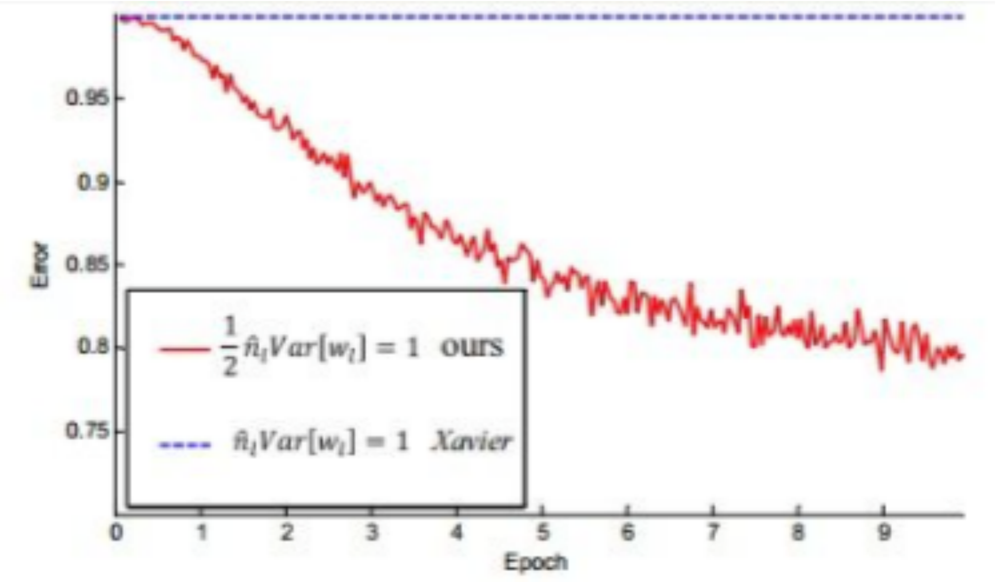


He 2015

ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

It makes a huge difference!



He 2015

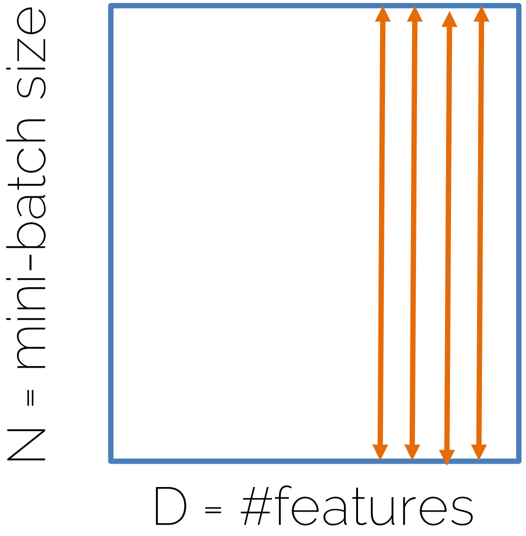
Tips and tricks

- Use ReLU and Xavier/2 initialization

Batch normalization

Batch normalization

- Wish: unit Gaussian activations
- Solution: let's do it



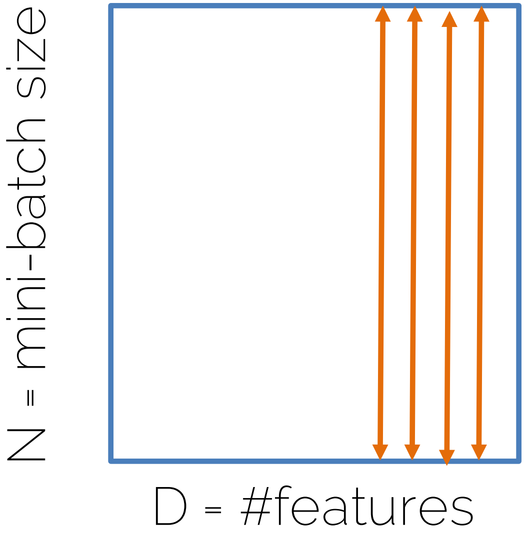
dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe and Szegedy 2015

Batch normalization

- In each dimension of the features, you have a unit gaussian



dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe and Szegedy 2015

Batch normalization

- In each dimension of the features, you have a unit Gaussian
- Is it ok to treat dimensions separately? Shown empirically that even if features are not decorrelated, convergence is still faster with this method

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

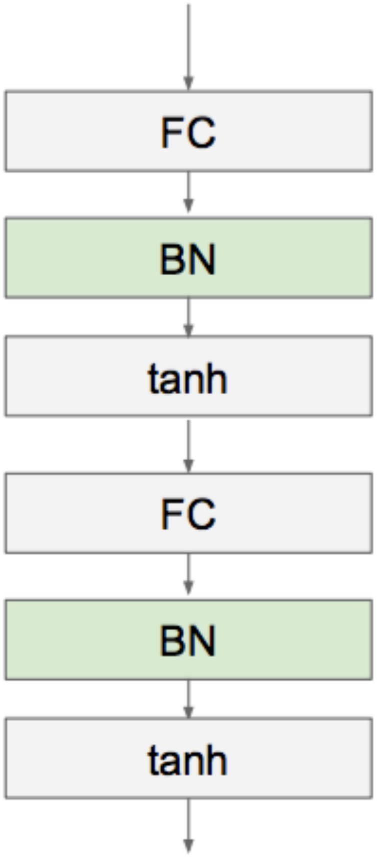
Differentiable function so we can backprop through it….

Ioffe and Szegedy 2015

Batch normalization

- A layer to be applied after Fully Connected (or Convolutional) layers and before non-linear activation functions

- Is it a good idea to have all unit Gaussians before tanh?



Ioffe and Szegedy 2015

Batch normalization

- Normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Allow the network to change the range

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

backprop

The network can learn to undo the normalization

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

Ioffe and Szegedy 2015

BN for Exercise 2

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
for $k = 1 \dots K$ **do**
  Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
  Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
end for
Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
for $k = 1 \dots K$ **do**
  // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
  Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

  In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
end for

Algorithm 2: Training a Batch-Normalized Network

Administrative Things

- Next Thursday May 25th: No Lecture!

- Thursday June 1st :
  - More on Neural Networks ☺

- Tomorrow: Solution 1st exercise, presentation 2nd