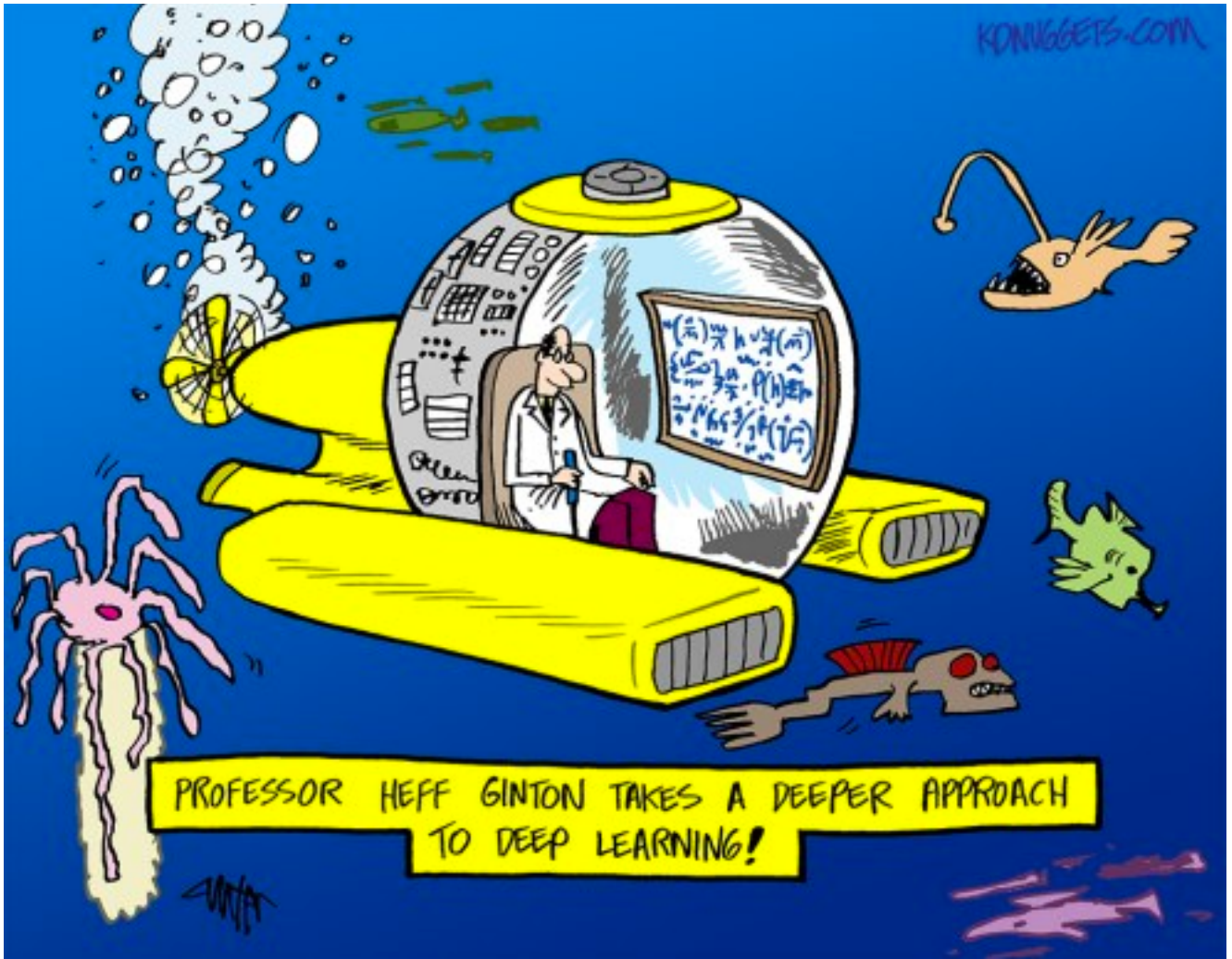
DEEP LEARNING AND NEURAL NETWORKS: BACKGROUND AND HISTORY



On-line Resources

- http://neuralnetworksanddeeplearning.com/index.html Online book by Michael Nielsen
- http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo - of convolutions
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html - demo of CNN
- http://scs.ryerson.ca/~aharley/vis/conv/ - 3D visualization
- http://cs231n.github.io/ Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition.

- http://www.deeplearningbook.org/ MIT Press book from Bengio et al, free online version

A history of neural networks

- 1940s-60's:
  - McCulloch & Pitts; Hebb: modeling real neurons
  - Rosenblatt, Widrow-Hoff: : perceptrons
  - 1969: Minskey & Papert, *Perceptrons* book showed formal limitations of one-layer linear network
- 1970's-mid-1980's: …
- mid-1980's – mid-1990's:
  - backprop and multi-layer networks
  - Rumelhart and McClelland *PDP* book set
  - Sejnowski's NETTalk, BP-based text-to-speech
  - Neural Info Processing Systems (NIPS) conference starts
- Mid 1990's-early 2000's: …
- Mid-2000's to current:
  - More and more interest and experimental success

TECH INSIDER INNOVATION

'Mach
the ir

Drake B
Apr. 1,

FACEBO

The Economist | World politics | Business & finance | Economics | Science & technology | Culture

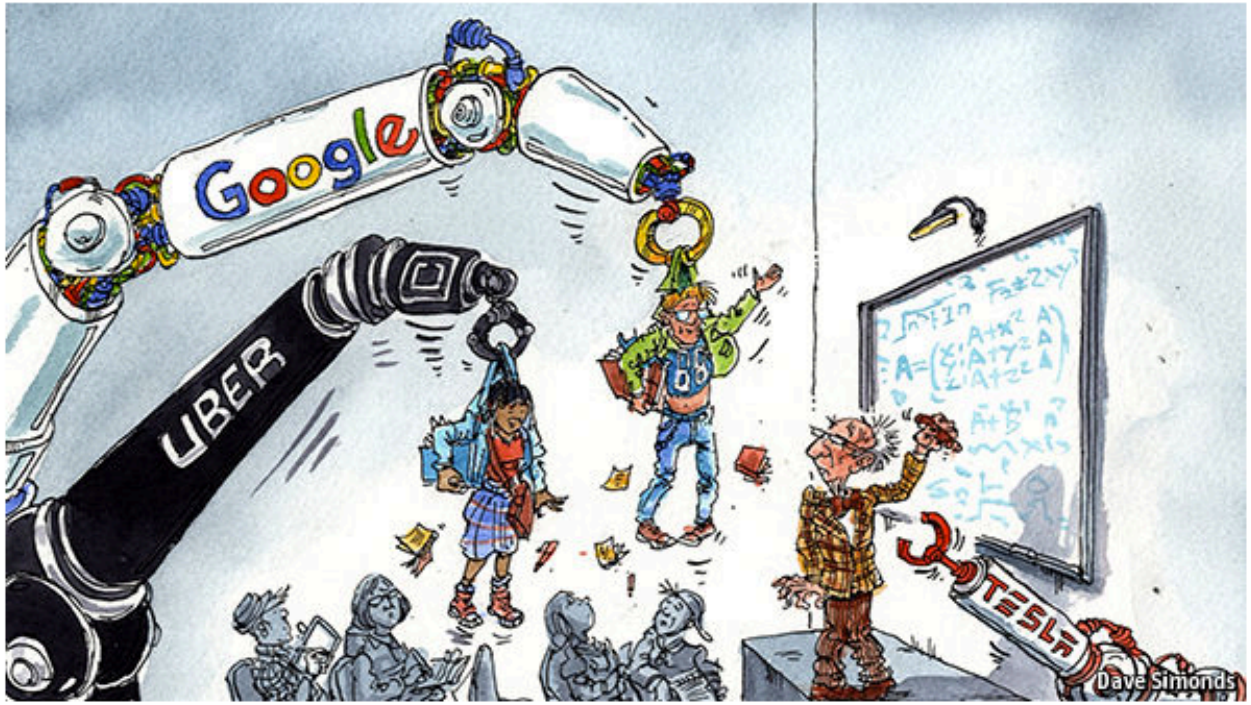
Artificial intelligence

Million-dollar babies

As Silicon Valley fights for talent, universities struggle to hold on to their stars

Apr 2nd 2016 | SAN FRANCISCO | From the print edition

Timekeeper | Like 6.8K | Tweet



CIENCE LIFE T-LOUNGE TE

odfather Of
aid Of AI

SUBSCRIBE

program AlphaGo
world Go champion Lee

ence experts, who
er program would need at
ough to be able to beat a

uter program is that
said that AlphaGo could
h with Sedol was able to

st skilled humans in
d Garry Kasparov two

lerts Talks

Don't worry,

We're in the

It used to b

SECTIONS

Facebook
and Intrus
Paying Off

TECHNOLOGY

Silicon Valley Looks to Artificial Intelligence for the Next Big Thing

Elon Musk: A
outcome' wi
Friday, 20 Jun 2014 | 1:0
Tesla Motors and Spac
artificial intelligence c
fledgling industry, and

Science AAAS

Home | News | Journals | Topics | Careers

Latest News | ScienceInsider | ScienceShots | Sifter | From the Magazine | About News | Quizzes

SHARE

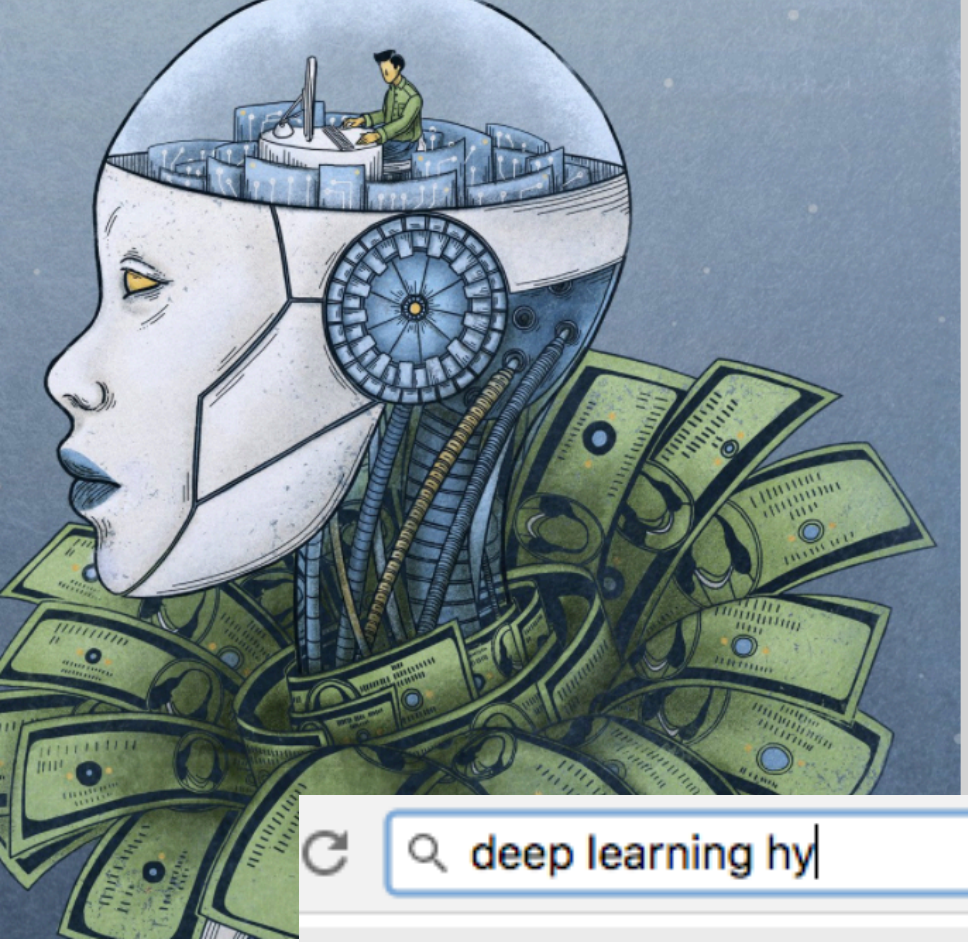
© FilippoBacci/iStockphoto

Artificial intelligence steals money from banking customers

By Adrian Cho | Apr. 1, 2016 , 3:00 AM

TECH • BRAINSTORM TECH

Yes, Your Company Needs a Chief AI Officer. Here's Why.



Home TECHNOLOGY

Tech Giants Are Paying Huge Salaries for Scarce A.I. Talent

Nearly all big tech companies have an artificial intelligence project, and they are willing to pay experts of dollars to help get it done.

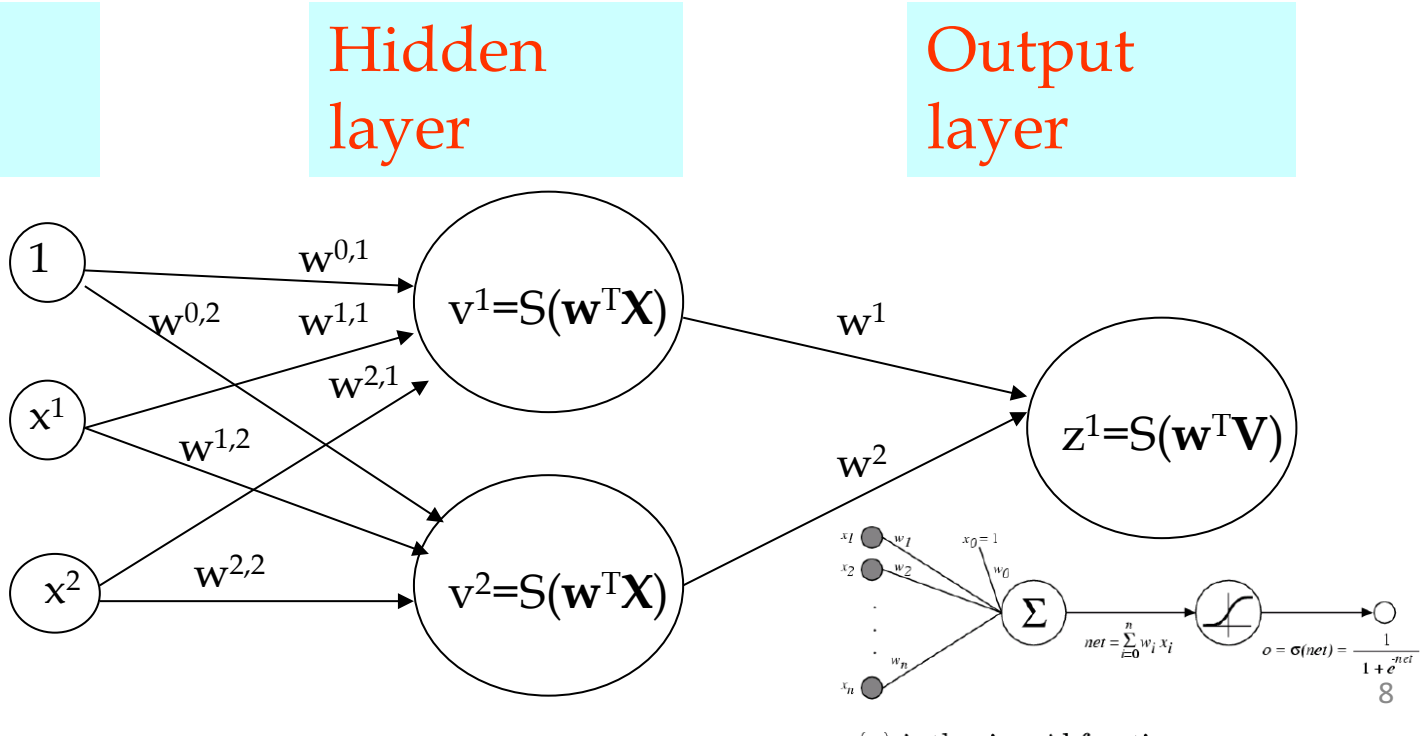
By CADE METZ OCT. 22, 2017

deep learning hy

- deep learning hy - Google Search
- deep learning hype
- deep learning hyperparameters
- deep learning hyperspectral
- deep learning hyperspectral image
- deep learning hyperparameter tuning

Multilayer networks

- Simplest case: classifier is a multilayer *network* of *logistic units*
- Each *unit* takes some inputs and produces one output using a logistic classifier
- Output of one unit can be the input of another

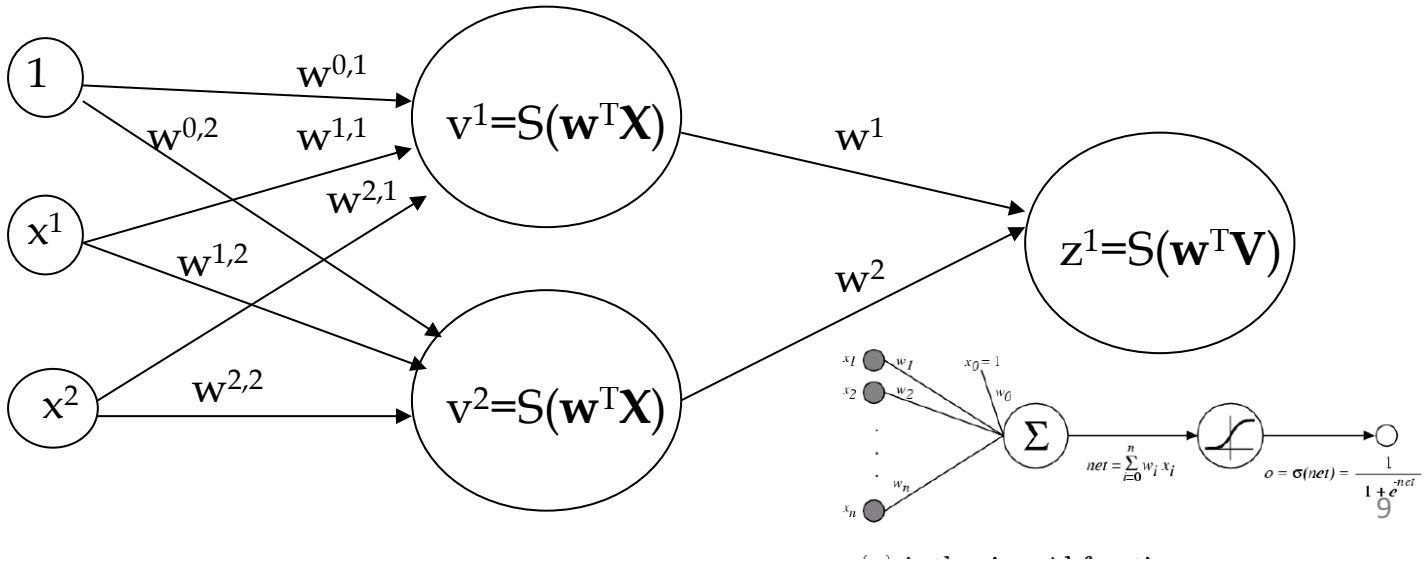


Learning a multilayer network

- Define a loss (simplest case: squared error)
  - But over a network of "units"
- Minimize loss with gradient descent

$$J_{\mathbf{x},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2$$

  - You can do this over complex networks if you can take the *gradient* of each unit: every computation is *differentiable*



ANNs in the 90's

- Mostly 2-layer networks or else carefully constructed "deep" networks (eg CNNs)
- Worked well but training was slow and finicky

Nov 1998 – Yann LeCunn, Bottou, Bengio, Haffner

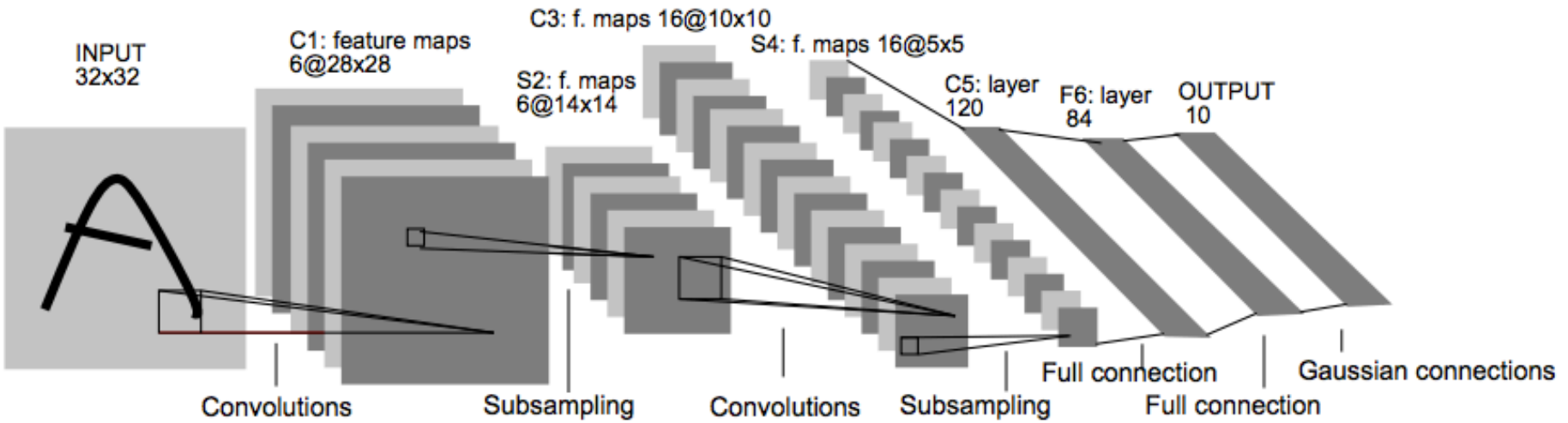
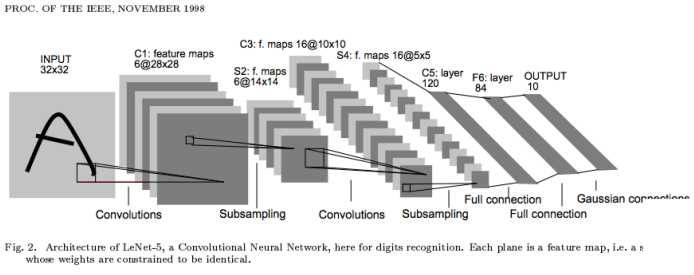


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

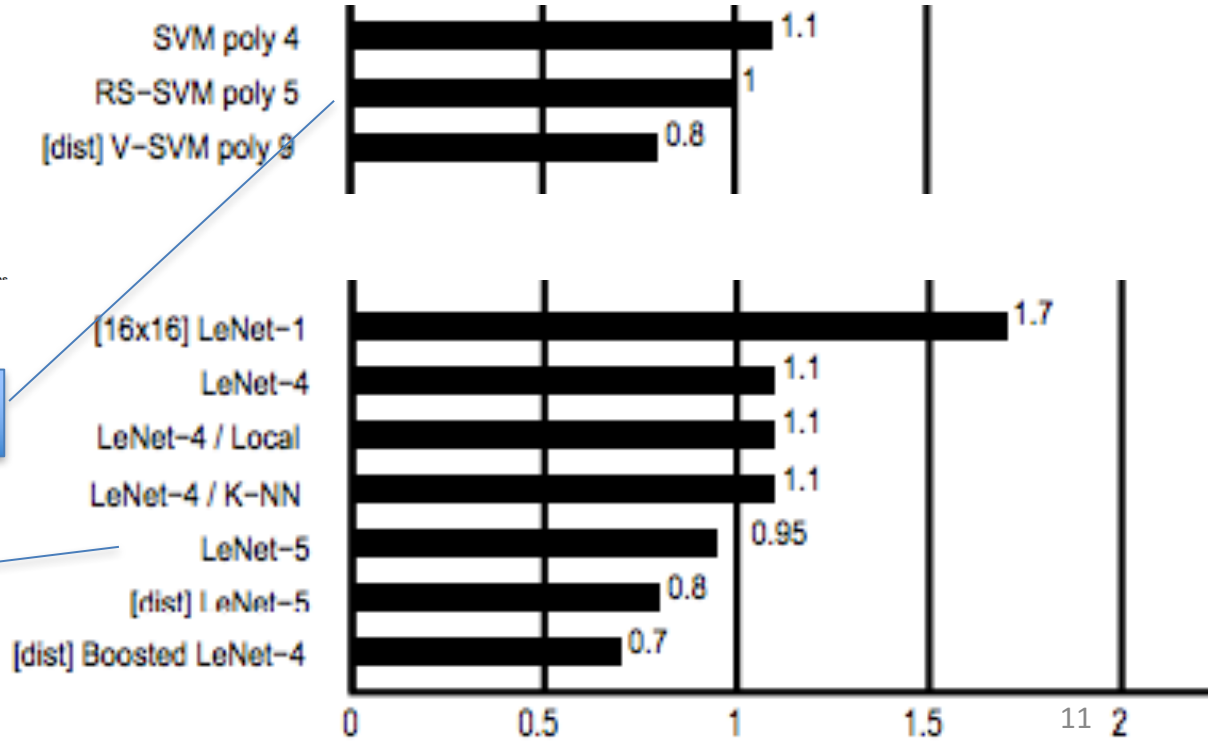
ANNs in the 90's

- Mostly 2-layer networks or else carefully constructed "deep" networks
- Worked well but training typically took weeks when guided by an expert



SVM: 98.9-99.2% accurate

CNNs: 98.3-99.3% accurate

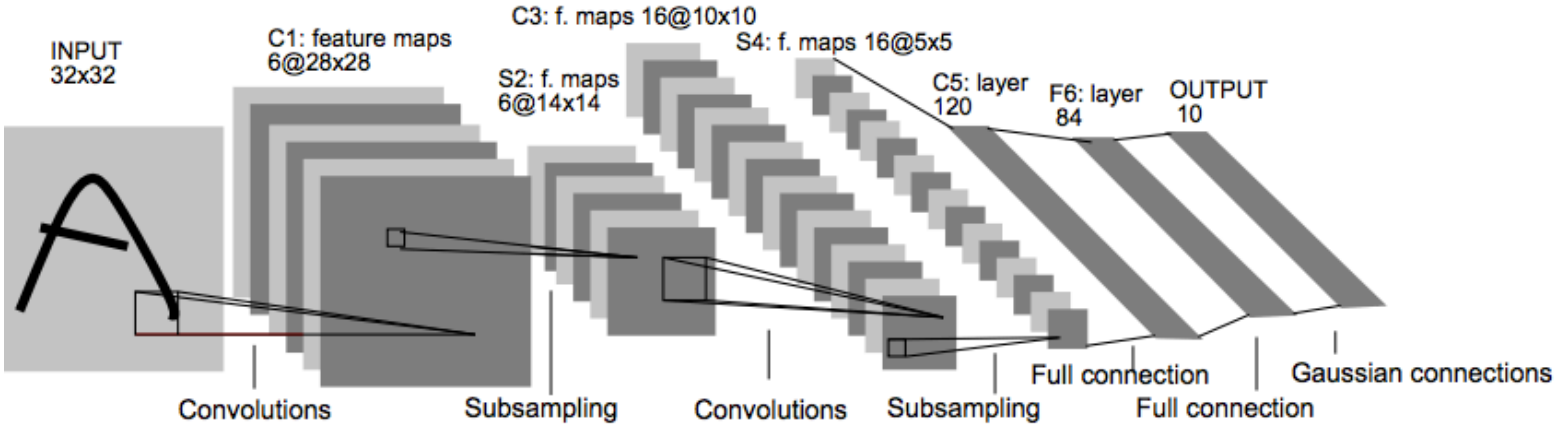


Learning a multilayer network

- Define a loss (simplest case: squared error)
  - But over a network of "units"
- Minimize loss with gradient descent

$$J_{\mathbf{x},\mathbf{y}}(\mathbf{w}) = \sum_i \left(y^i - \hat{y}^i\right)^2$$

  - You can do this over complex networks if you can take the *gradient* of each unit: every computation is *differentiable*



Example: weight updates for multilayer ANN with square loss and logistic units

For nodes k in output layer:

$$\delta_k \equiv \left(t_k - a_k\right)\ a_k\left(1 - a_k\right)$$

For nodes j in hidden layer:

$$\delta_j \equiv \sum_k \left(\delta_k w_{kj}\right)\ \ a_j\left(1 - a_j\right)$$

For all weights:

$$w_{kj} = w_{kj} - \varepsilon\ \ \delta_k a_j$$

$$w_{ji} = w_{ji} - \varepsilon\ \ \delta_j a_i$$

"Propagate errors backward" BACKPROP

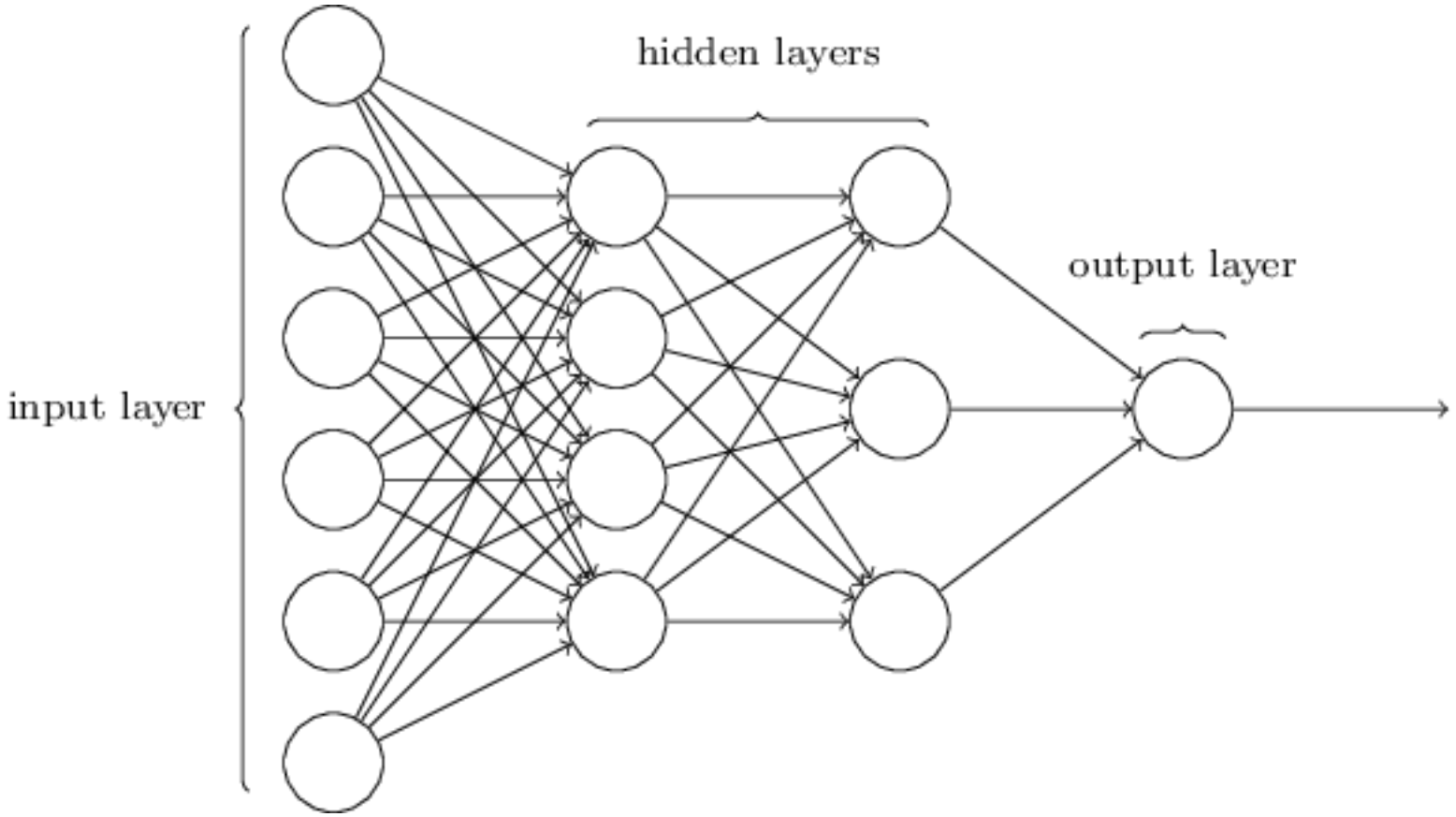
Can carry this recursion out further if you have multiple hidden layers

BACKPROP FOR MLPS

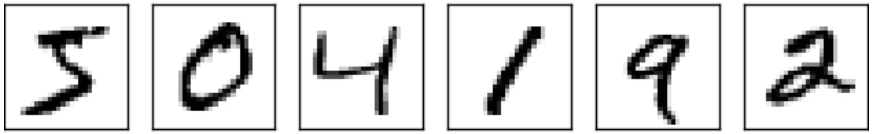
BackProp in Matrix-Vector Notation

Michael Nielson: http://neuralnetworksanddeeplearning.com/

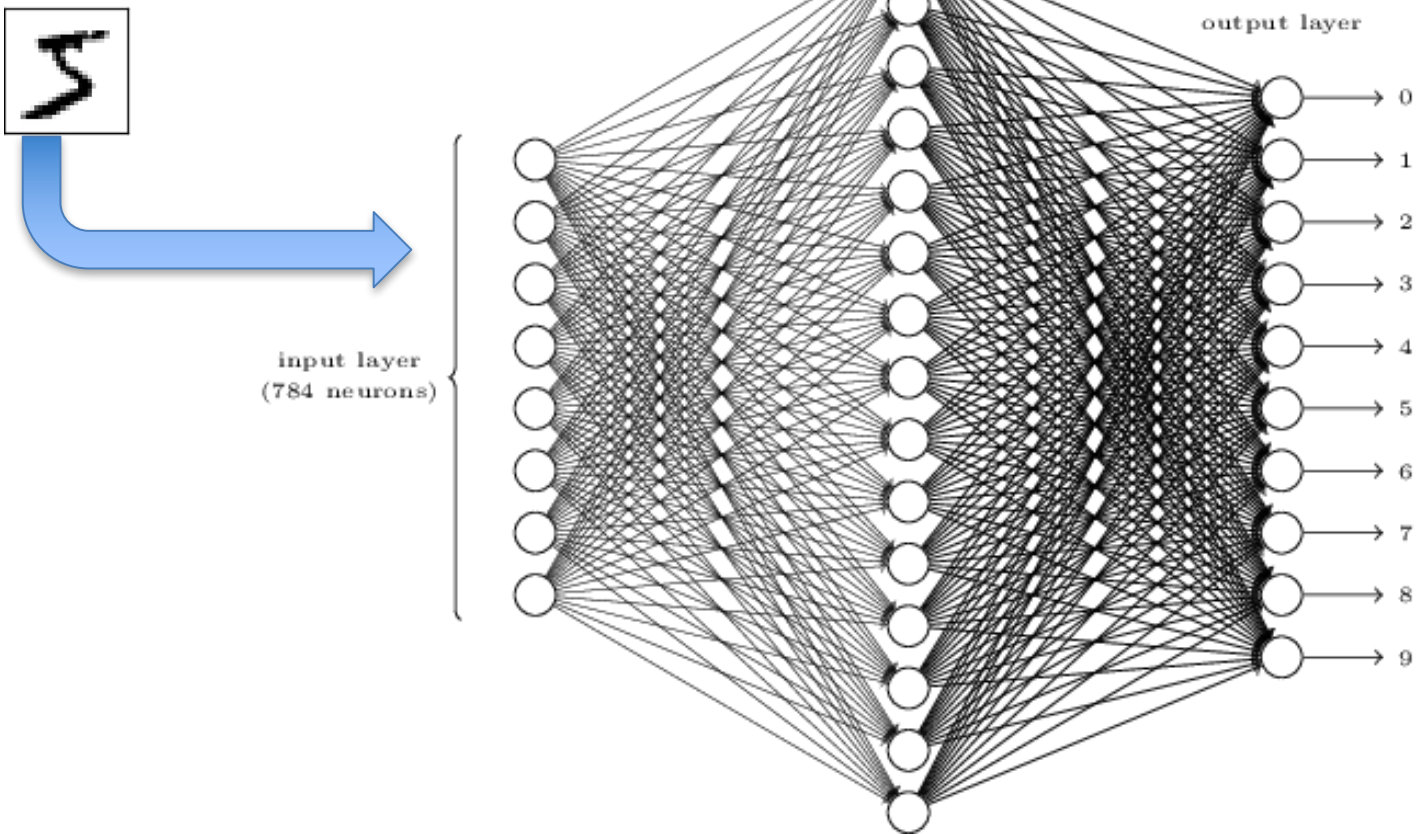
Notation



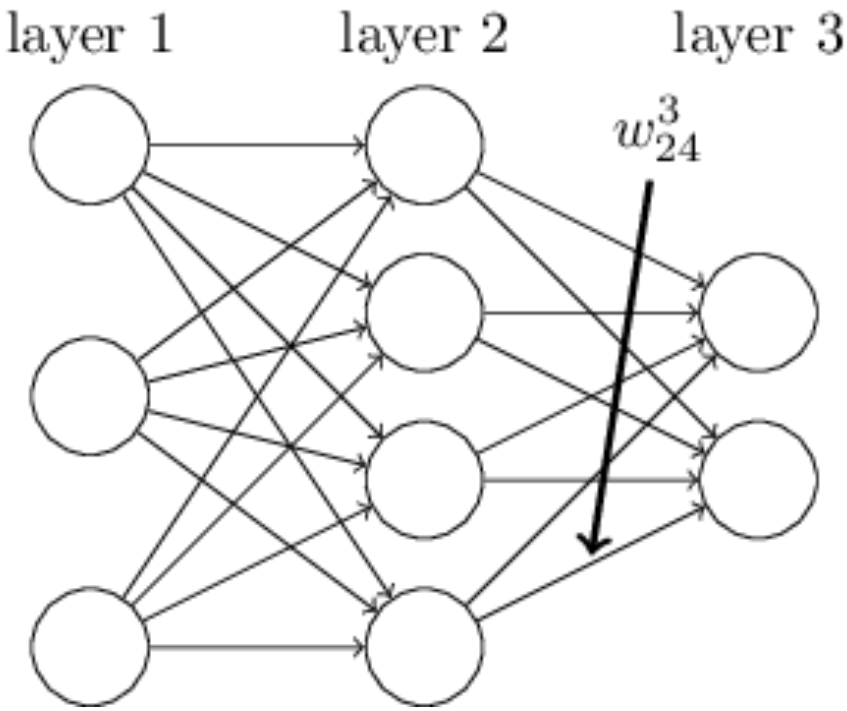
Notation



Each digit is 28x28 pixels = 784 inputs



Notation



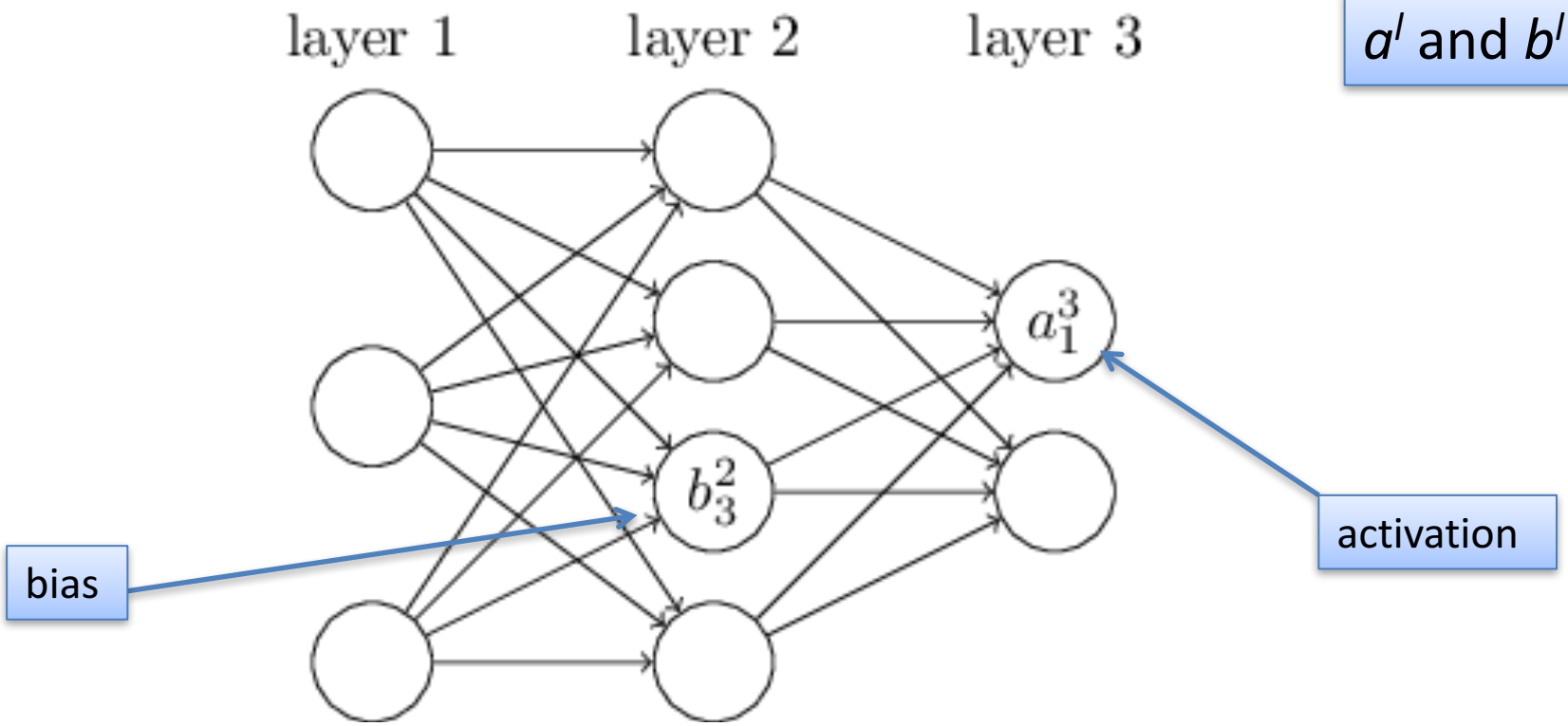
w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

w^l is weight matrix for layer l

Notation

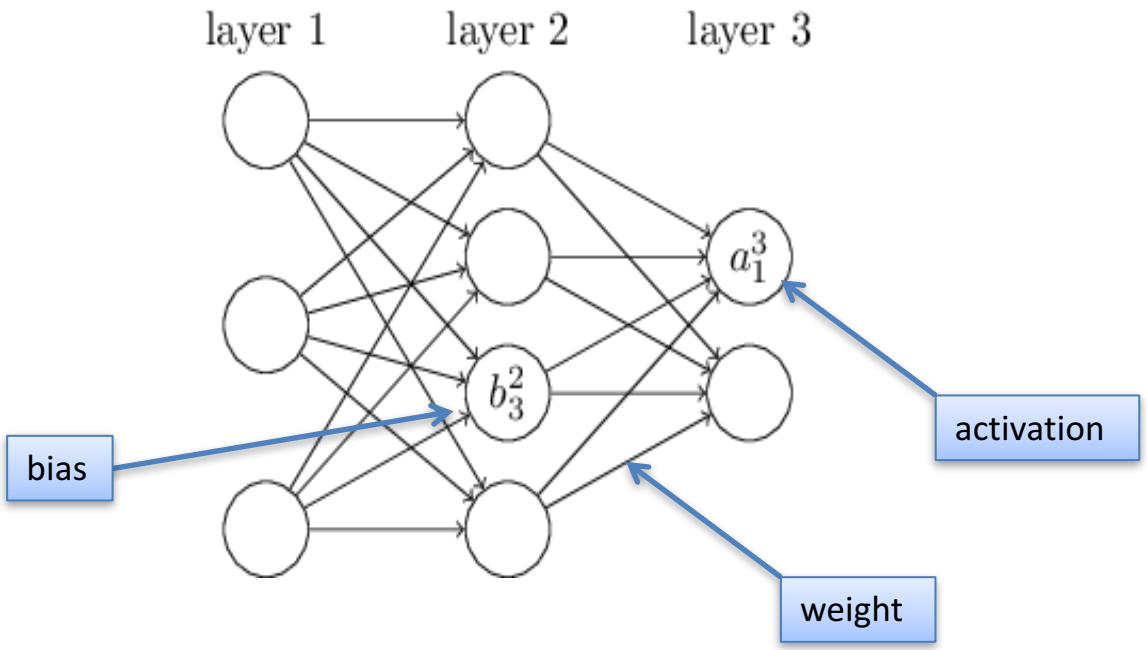
w^l

a^l and b^l



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

Notation



Matrix: w^l
Vector: a^l
Vector: b^l

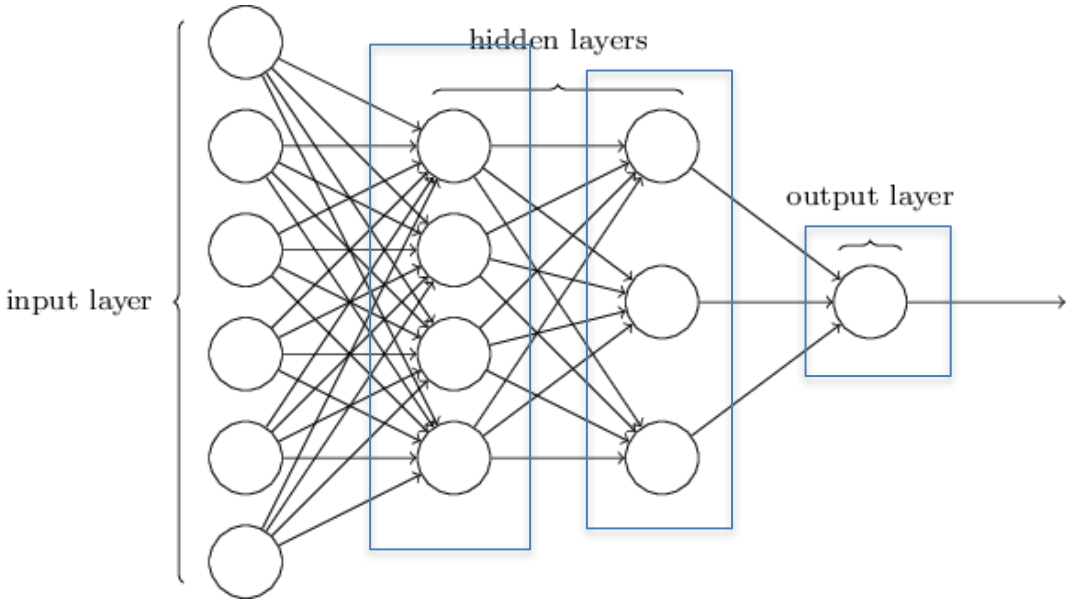
Vector: z^l

vector→vector function: componentwise logistic

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \quad \Rightarrow \quad a^l = \sigma(w^l a^{l-1} + b^l).$$

$$z^l \equiv w^l a^{l-1} + b^l$$

Computation is “feedforward”



for *l=1, 2, … L:*

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

Notation

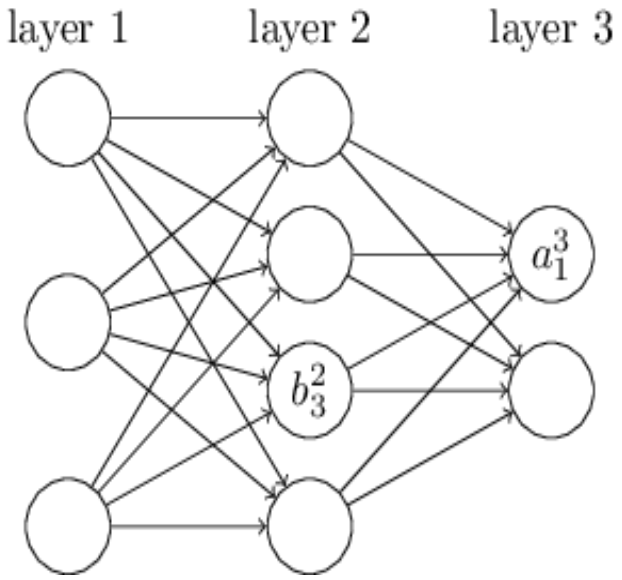
Cost function to optimize:
sum over examples *x*

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2,$$

$$= \frac{1}{n} \sum_x C_x$$

where

$$C_x = \frac{1}{2} \|y - a^L\|^2$$

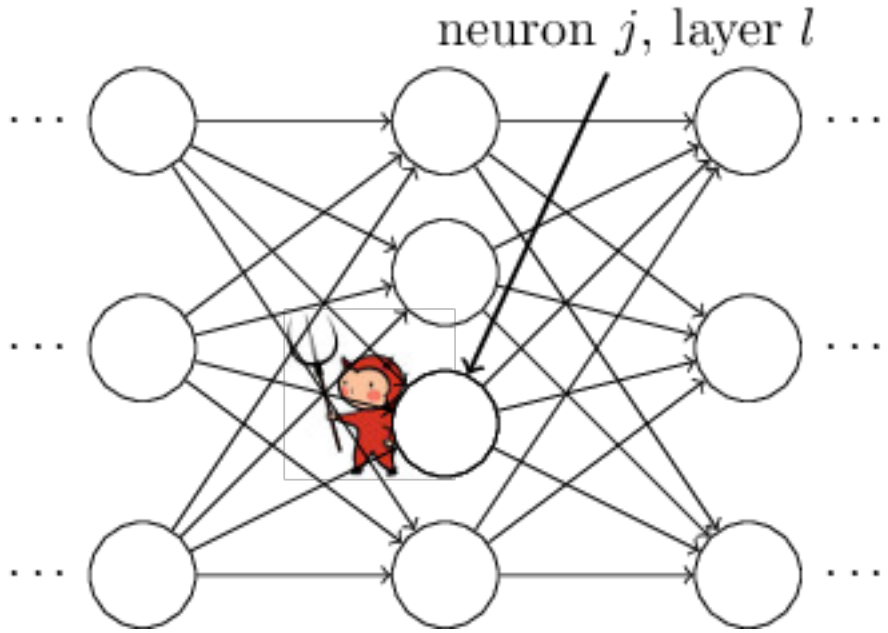


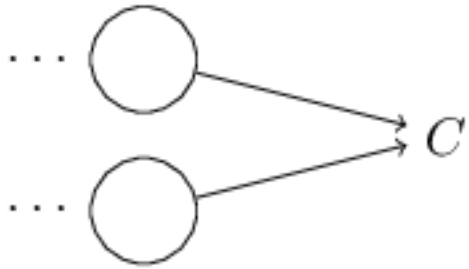
Matrix: w^l
Vector: a^l
Vector: b^l

Vector: z^l

Vector: y

Notation

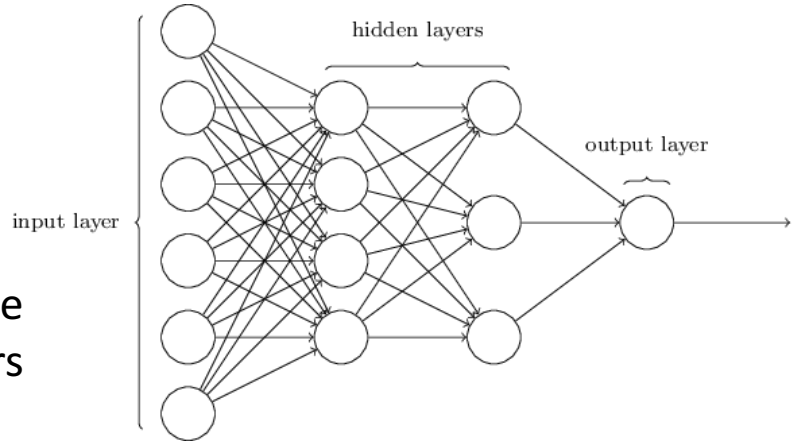




$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

BackProp: last layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$



componentwise product of vectors

Matrix form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L).$$

components are $\frac{\partial C}{\partial a_j^L}$

components are $\sigma'(z_j^L)$

Level *l* for *l*=1,…,*L*

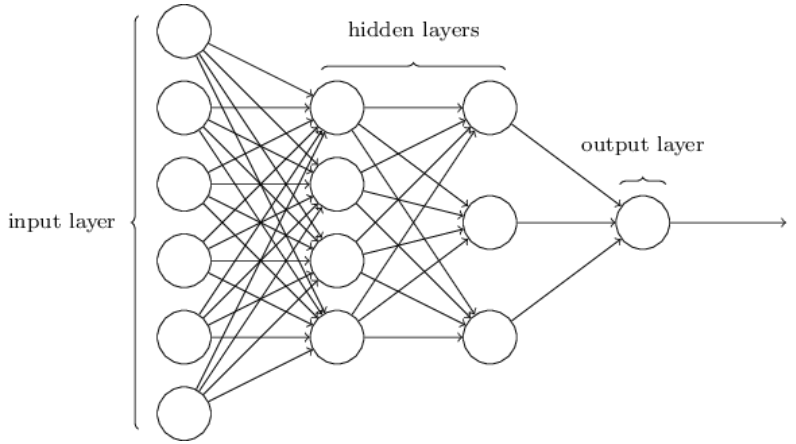
Matrix: w^l

Vectors:

- bias b^l
- activation a^l
- pre-sigmoid activ: z^l
- target output *y*
- "local error" δ^l

BackProp: last layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$



Matrix form for square loss:

$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$

Level *l* for *l=1,…,L*

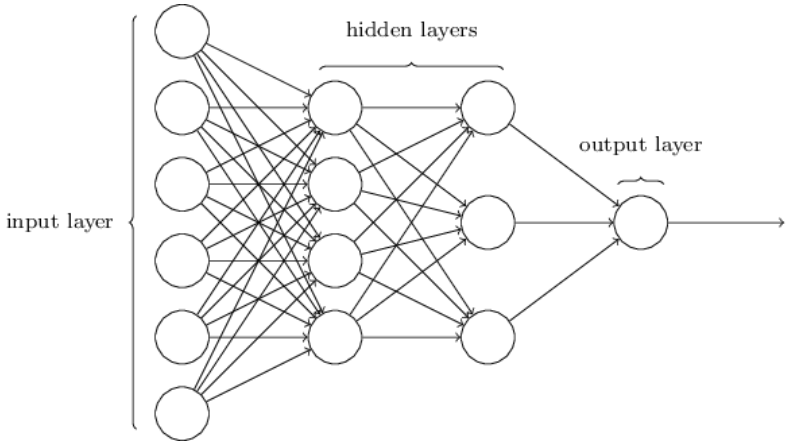
Matrix: w^l

Vectors:

- bias b^l
- activation a^l
- pre-sigmoid activ: z^l
- target output *y*
- "local error" δ^l

BackProp: error at level *l* in terms of error at level *l+1*

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$



which we can use to compute

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \Rightarrow \frac{\partial C}{\partial b} = \delta$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \Rightarrow \frac{\partial C}{\partial w} = a_{\text{in}} \delta_{\text{out}}$$

Level *l* for *l=1,…,L*

Matrix: w^l

Vectors:

- bias b^l
- activation a^l
- pre-sigmoid activ: z^l
- target output *y*
- "local error" δ^l

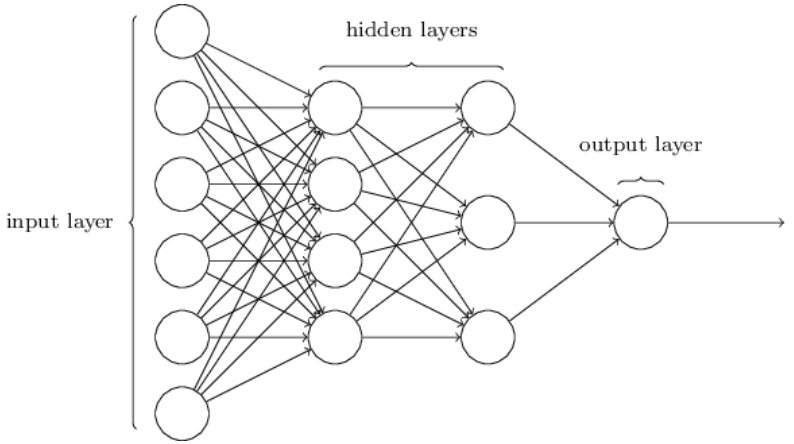
BackProp: summary

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j$$



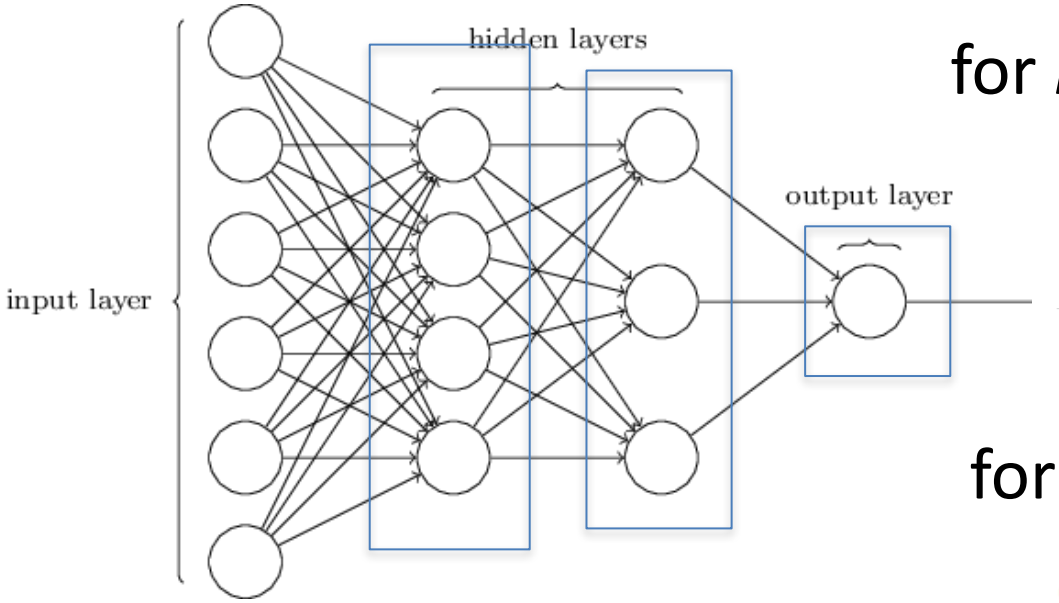
Level *l* for *l=1,…,L*

Matrix: w^l

Vectors:

- bias b^l
- activation a^l
- pre-sigmoid activ: z^l
- target output *y*
- "local error" δ^l

Computation propagates errors backward



for *l=1, 2, … L:*

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

for *l=L,L-1,…1:*

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

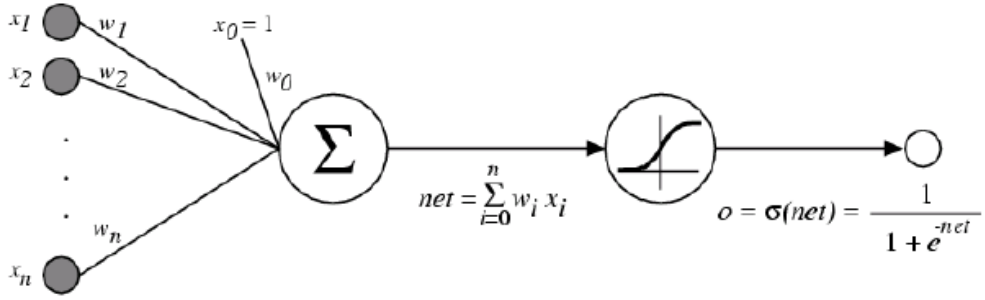
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

EXPRESSIVENESS OF DEEP NETWORKS

Deep ANNs are expressive

- One logistic unit can implement and AND or an OR of a subset of inputs
  - e.g., (x_3 AND x_5 AND … AND x_{19})
- Every boolean function can be expressed as an OR of ANDs
  - e.g., (x_3 AND x_5) OR (x_7 AND x_{19}) OR …
- So one hidden layer can express any BF

(But it might need lots and lots of hidden units)



Deep ANNs are expressive

- One logistic unit can implement and AND or an OR of a subset of inputs
  - e.g., (x_3 AND x_5 AND … AND x_{19})
- Every boolean function can be expressed as an OR of ANDs
  - e.g., (x_3 AND x_5) OR (x_7 AND x_{19}) OR …
- So one hidden layer can express any BF

- Example: parity(x1,…,xN)=1 iff off number of xi's are set to one

Parity(a,b,c,d) =

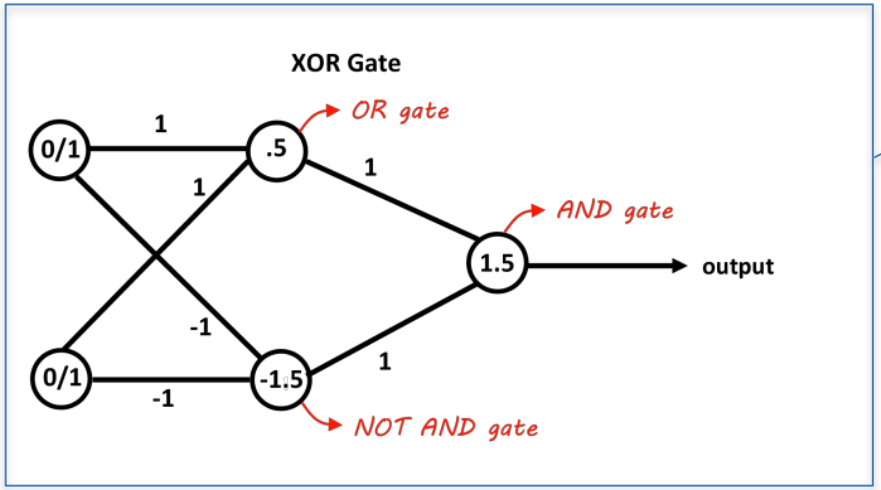
(a & -b & -c & -d) OR (-a & b & -c & -d) OR … #list all the "1s"

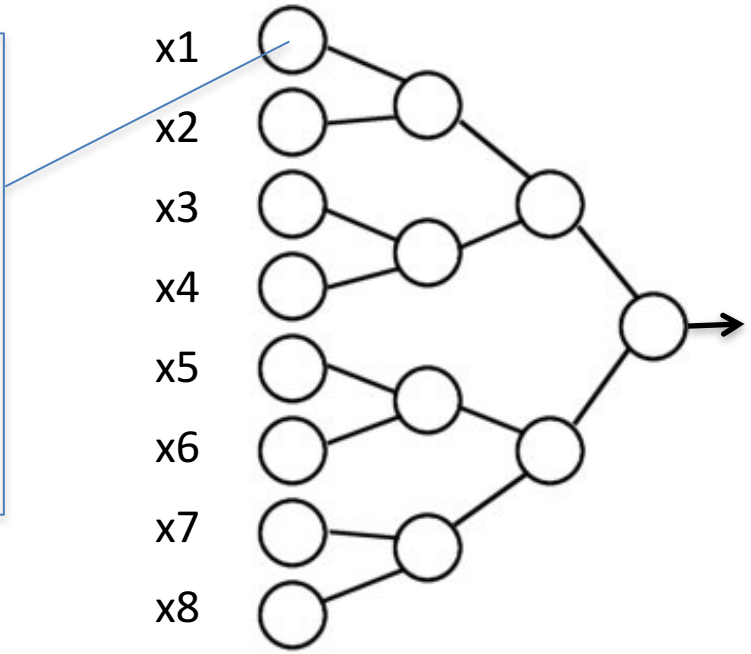
(a & b & c & -d) OR (a & b & -c & d) OR … #list all the "3s"

Size in general is $O(2^N)$

Deeper ANNs are more expressive

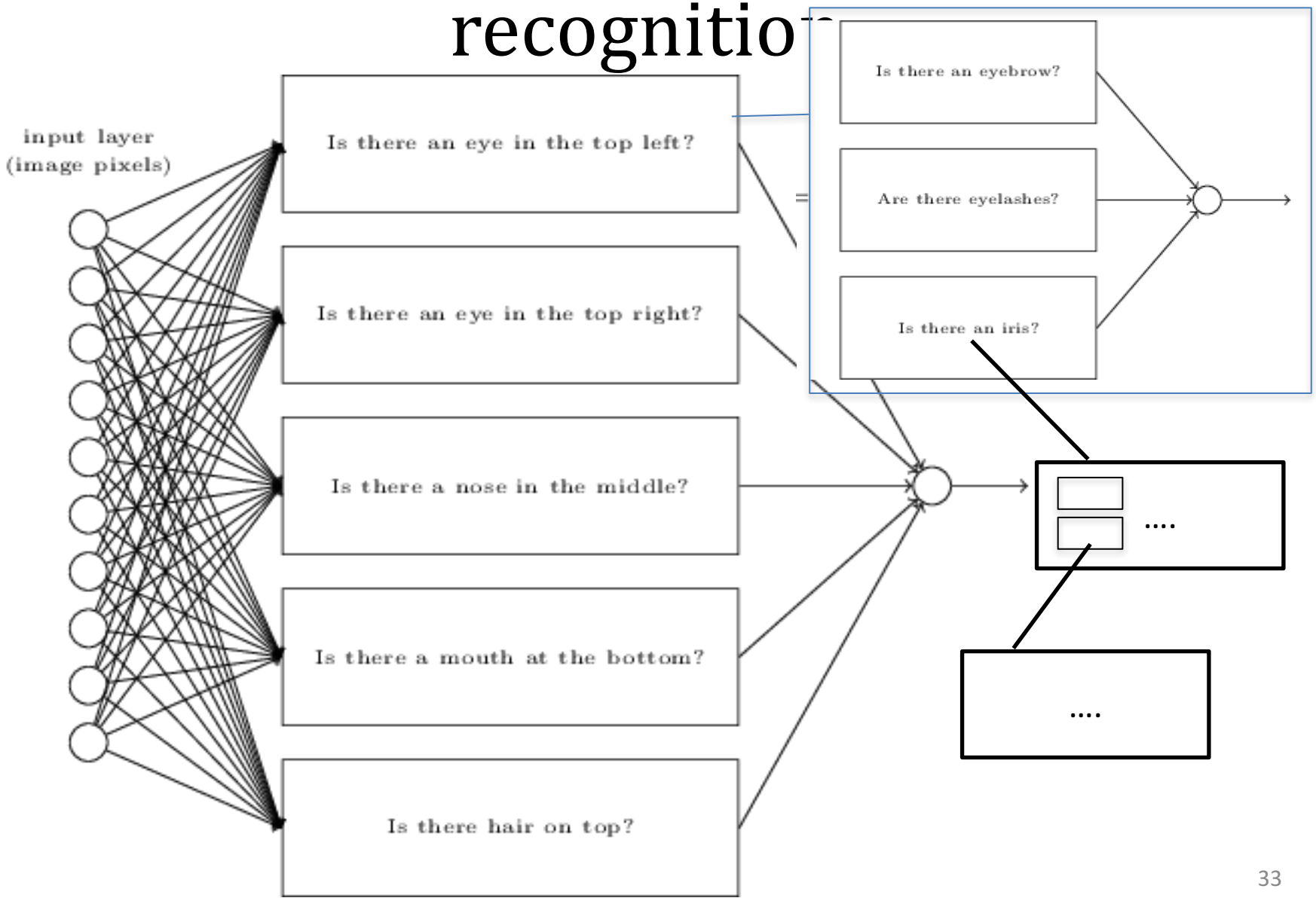
- A two-layer network needs $O(2^N)$ units
- A two-layer network can express binary XOR
- A 2*logN layer network can express the parity of N inputs (even/odd number of 1's)
  - With O(logN) units in a binary tree
- Deep network + parameter tying ~= subroutines





http://neuralnetworksanddeeplearning.com/chap1.html

Hypothetical code for face recognitio



PARALLEL TRAINING FOR ANNS

How are ANNs trained?

- Typically, with some variant of streaming SGD
  - Keep the data on disk, in a preprocessed form
  - Loop over it multiple times
  - Keep the model in memory

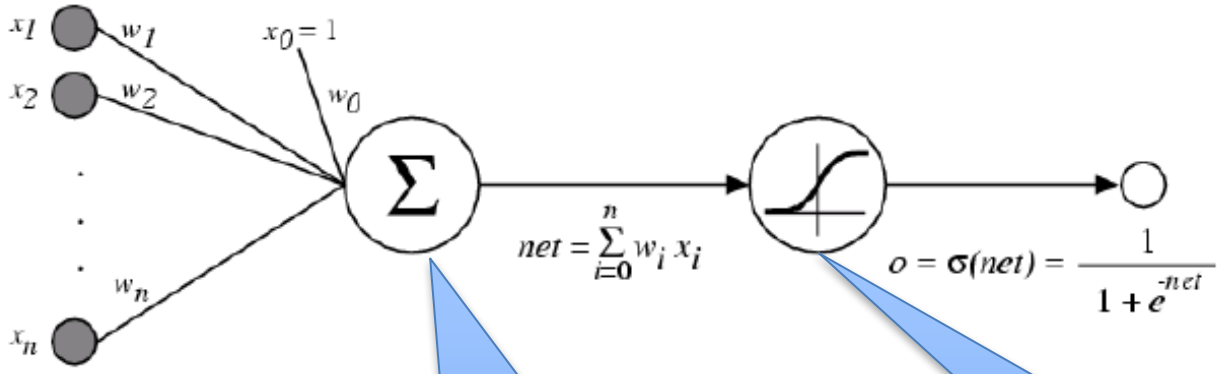
- Solution to big data: but long training times!

- However, *some* parallelism is often used….

Recap: logistic regression with SGD

$$P(Y=1 \mid X=\mathbf{x}) = p = \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}}$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y-p)\mathbf{x}$$



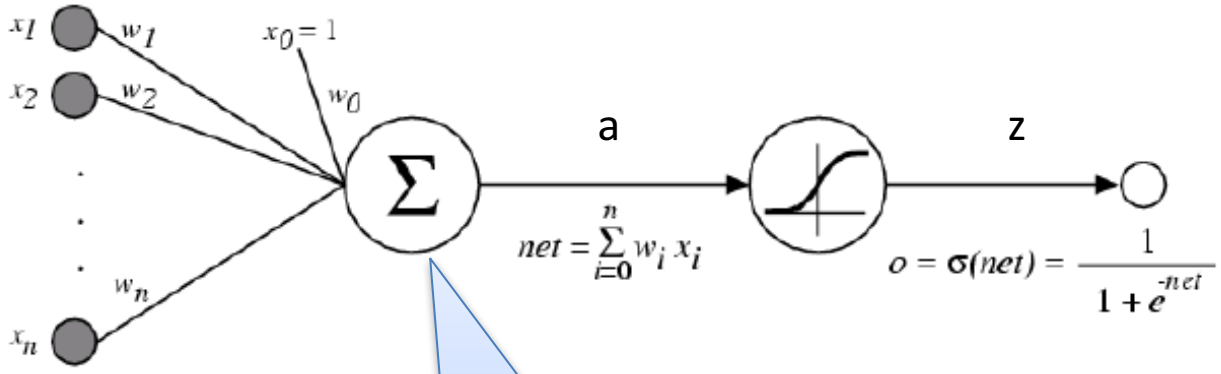
This part computes inner product **<x,w>**

This part logistic of **<x,w>**

Recap: logistic regression with SGD

$$P(Y=1 \mid X=\mathbf{x}) = p = \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}}$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y-p)\mathbf{x}$$

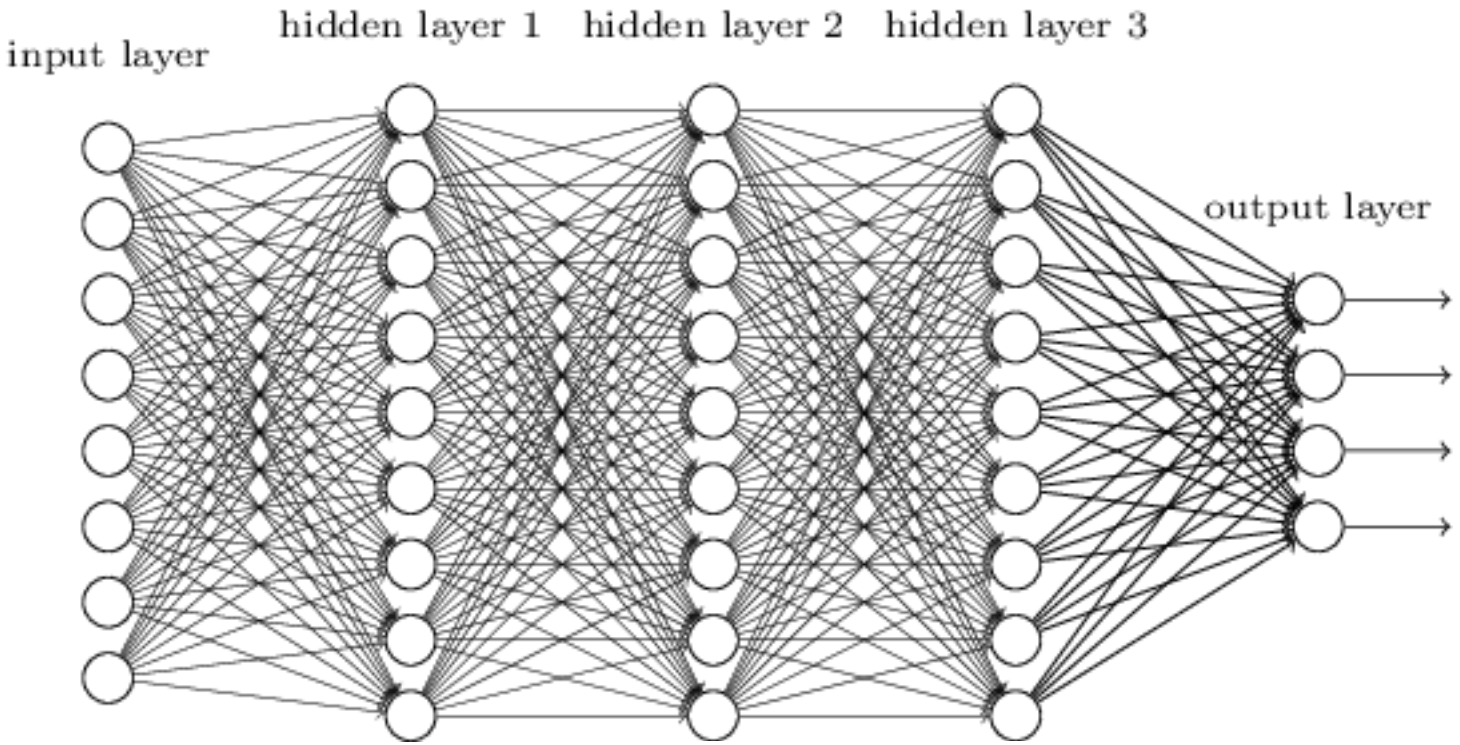


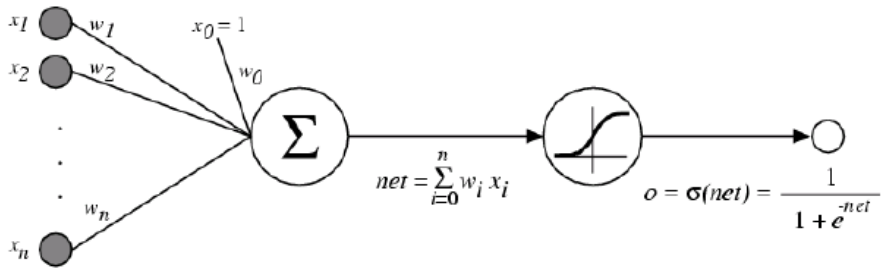
On one example: computes inner product **<x,w>**

$$\sum_j x^j w^j$$

There's some chance to compute this in parallel…can we do more?

In ANNs we have many many logistic regression nodes



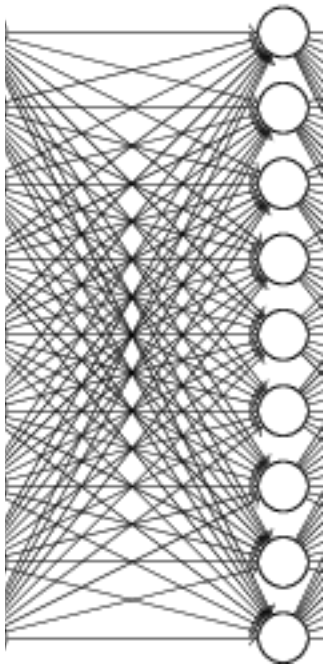


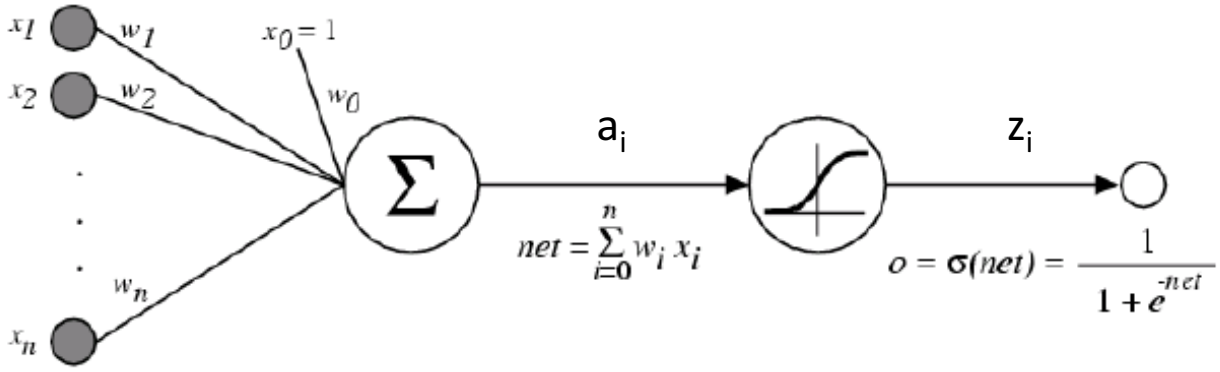
Recap: logistic regression with SGD

Let **x** be an example

Let $\mathbf{w}_i$ be the input weights for the i-th hidden unit

Then output $\mathbf{a}_i = \mathbf{x} \cdot \mathbf{w}_i$





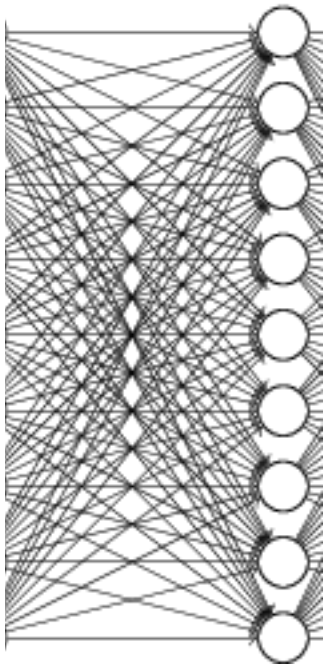
Recap: logistic regression with SGD

Let **x** be an example

Let $\mathbf{w}_i$ be the input weights for the i-th hidden unit

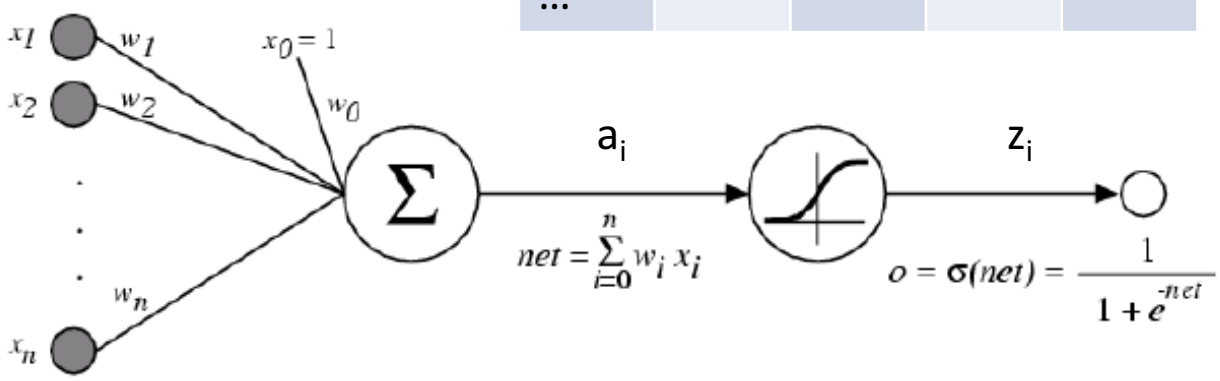
Then **a** = **x** W

is output for all *m* units



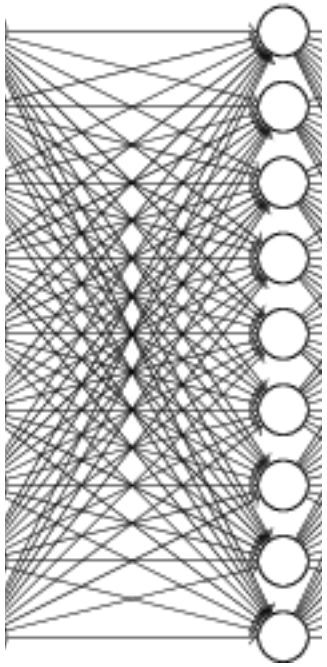
W =

w_1	w_2	w_3	…	w_m
0.1	-0.3	…		
-1.7	…			
..				
…				



Recap: logistic regression with SGD

Let X be a matrix with *k* examples
Let $\mathbf{w}_i$ be the input weights for the i-th hidden unit
Then A = X W is output for all *m* units
for all *k* examples



x_1	1	0	1	1
x_2	…			
…				
x_k				

w_1	w_2	w_3	…	w_m
0.1	-0.3	…		
-1.7	…			
0.3	…			
1.2				

There's a lot of chances to do this in parallel

XW =

$x_1.w_1$	$x_1.w_2$	…	$x_1.w_m$
$x_k.w_1$	…	…	$x_k.w_m$

ANNs and multicore CPUs

- Modern libraries (Matlab, numpy, …) do matrix operations fast, in parallel
- Many ANN implementations exploit this parallelism automatically
- Key implementation issue is working with matrices comfortably

ANNs and GPUs

- GPUs do matrix operations very fast, in parallel
  - For dense matrixes, not sparse ones!
- Training ANNs on GPUs is common
  - SGD and minibatch sizes of 128
- Modern ANN implementations can exploit this
- GPUs are not super-expensive
  - $500 for high-end one
  - large models with $O(10^7)$ parameters can fit in a large-memory GPU (12Gb)
- Speedups of 20x-50x have been reported

ANNs and multi-GPU systems

- There are ways to set up ANN computations so that they are spread across multiple GPUs
  - Sometimes involves some sort of IPM
  - Sometimes involves partitioning the model across multiple GPUs
  - Often needed for very large networks
  - Not especially easy to implement and do with most current tools

WHY ARE DEEP NETWORKS HARD TO TRAIN?

Recap: weight updates for multilayer ANN

For nodes k in output layer L:

$$\delta_k^L \equiv (t_k - a_k)\ a_k(1 - a_k)$$

For nodes j in hidden layer h:

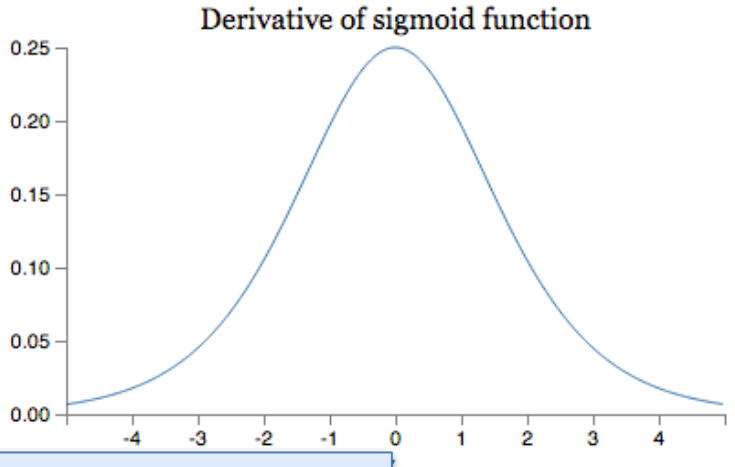
$$\delta_j^h \equiv \sum_k \left(\delta_j^{h+1}\ w_{kj}\right)\ a_j(1 - a_j)$$

What happens as the layers get further and further from the output layer? E.g., what's gradient for the bias term with several layers after it?

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

Gradients are unstable

Max at 1/4



If weights are usually < 1 then we are multiplying by many numbers < 1 so the gradients get very small.

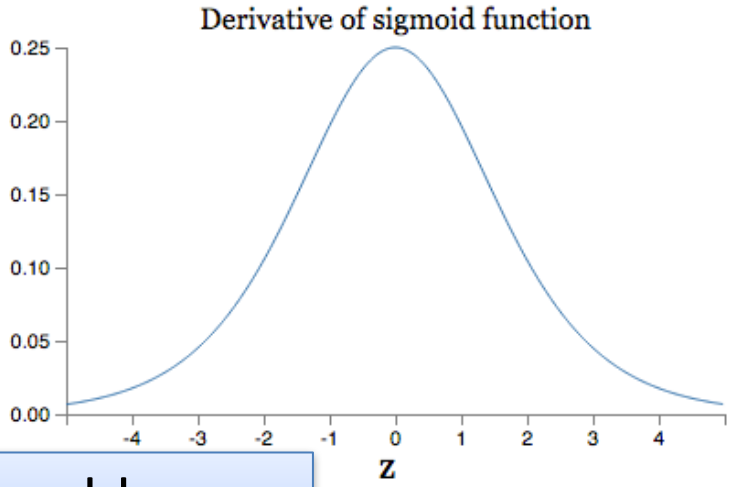
The **vanishing gradient** problem

What happens as the layers get further and further from the output layer? E.g., what's gradient for the bias term with several layers after it in a trivial net?

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

Gradients are unstable

Max at 1/4



If weights are usually > 1 then we are multiplying by many numbers > 1 so the gradients get very big.

The **exploding gradient** problem (less common but possible)

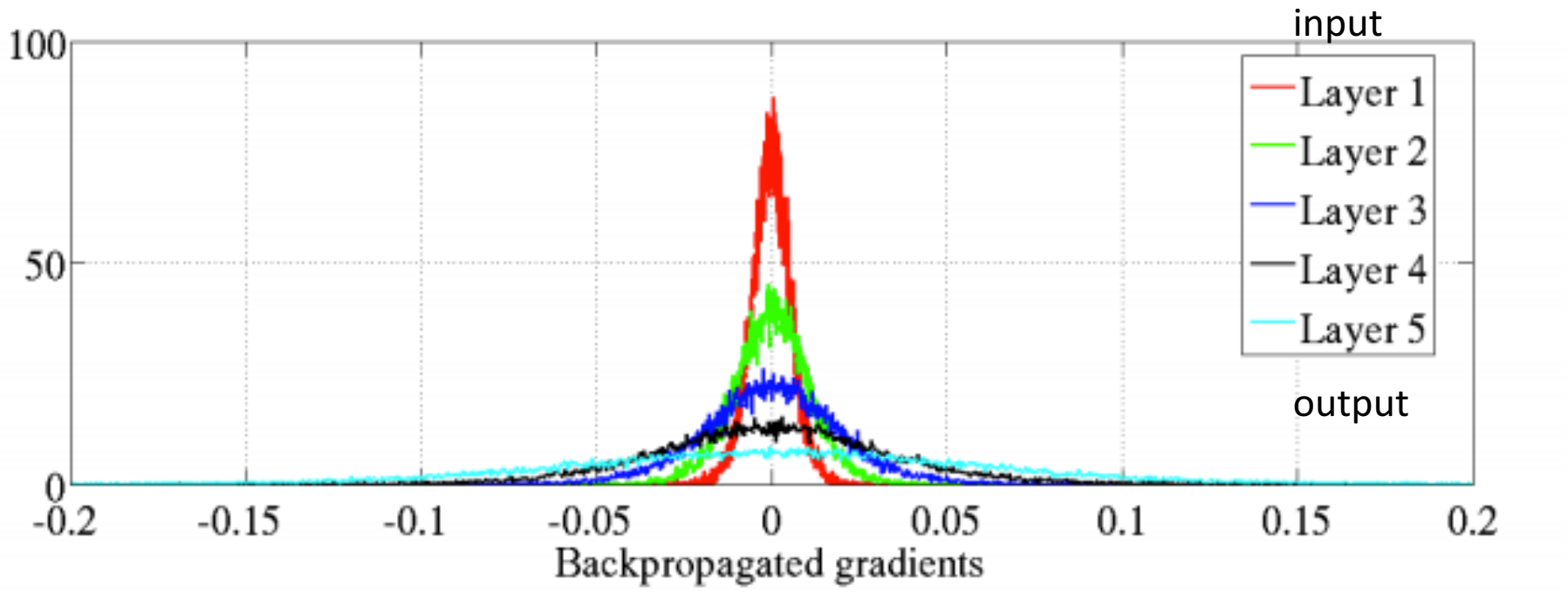
What h… further from the output layer? E.g., what's gradient for the bias term with several layers after it in a trivial net?

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

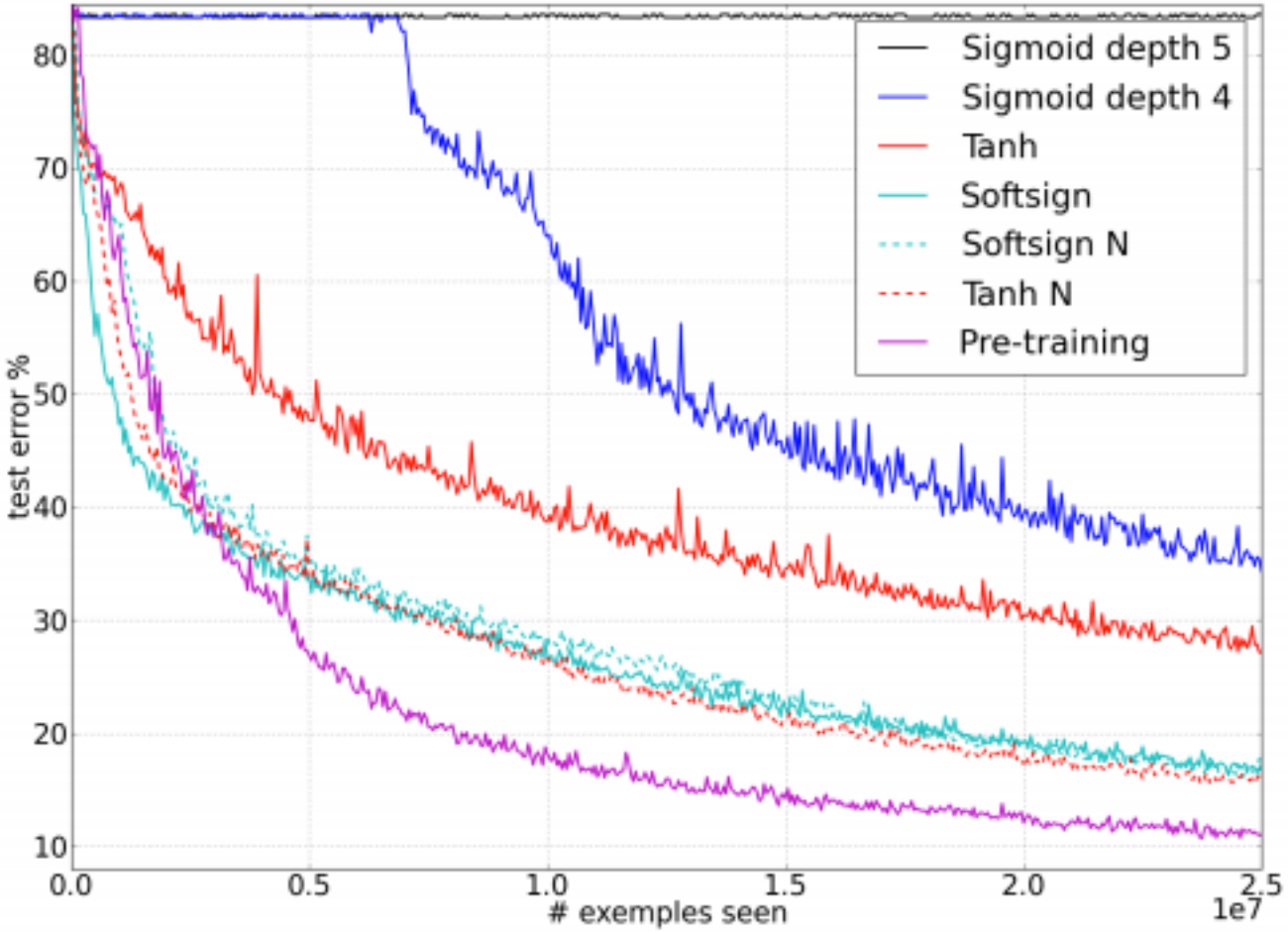
Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada



Histogram of gradients in a 5-layer network for an artificial image recognition task

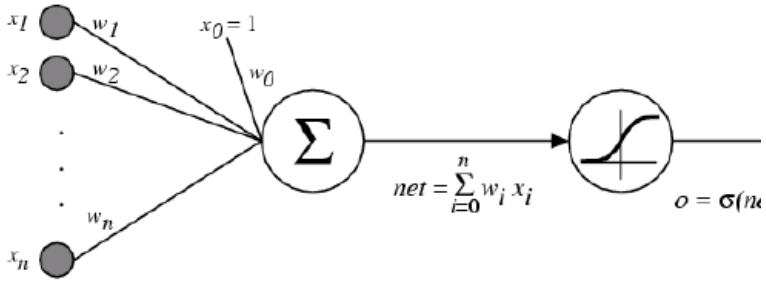
Understanding the difficulty of training deep feedforward neural networks

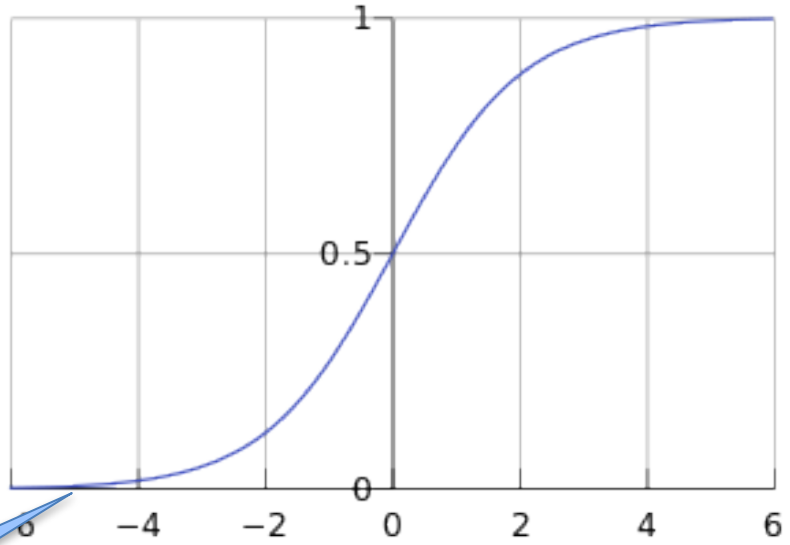
AI Stats 2010

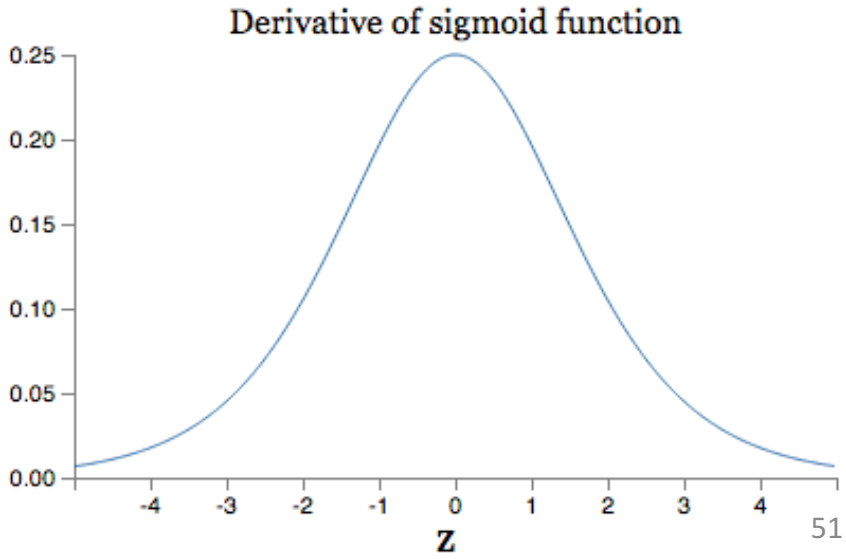


We will get to these tricks eventually….

It's easy for sigmoid units to saturate

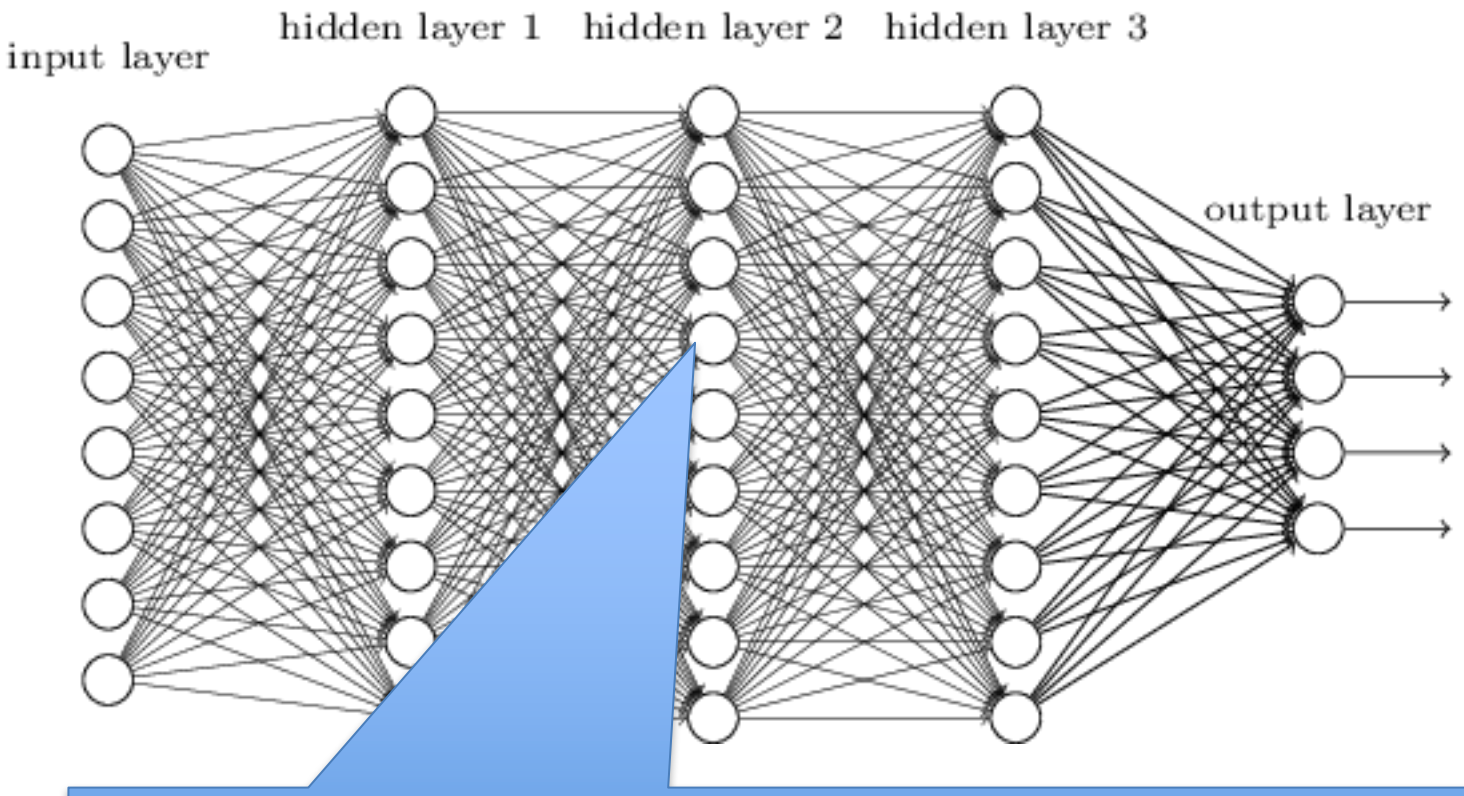






components are Learning rate approaches zero and unit is "stuck"

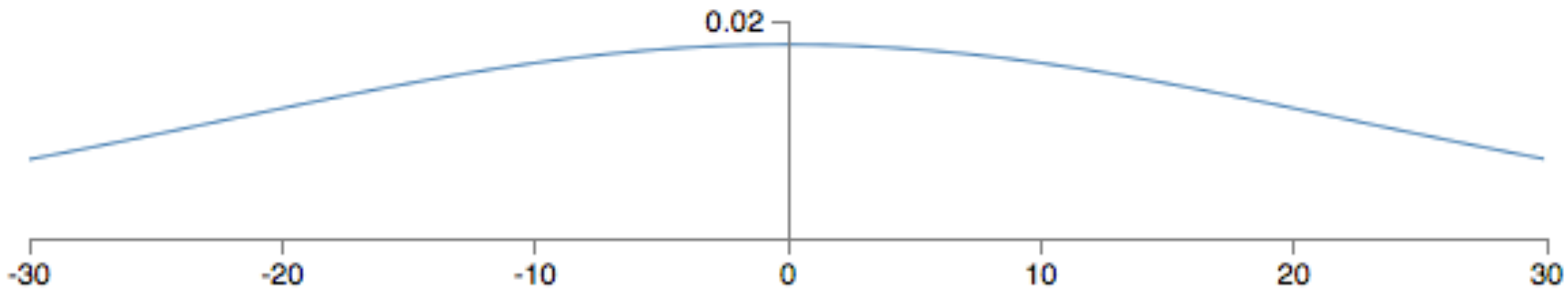
It's easy for sigmoid units to saturate



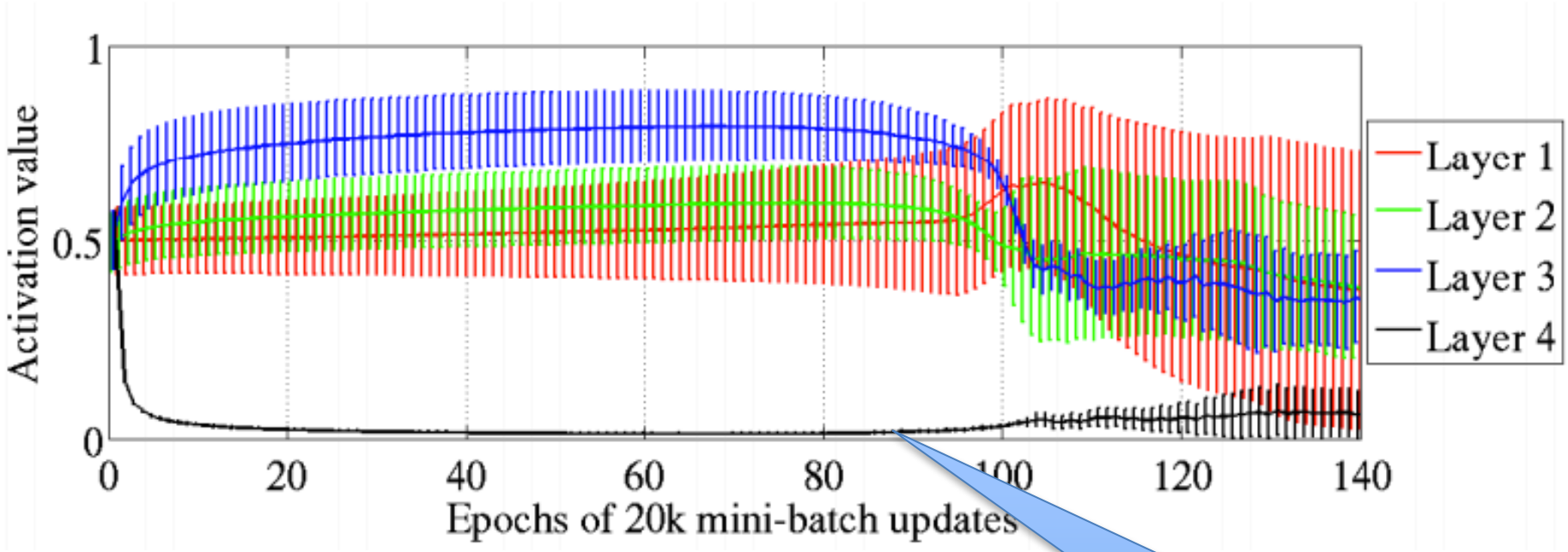
For a big network there are lots of weighted inputs to each neuron. If any of them are too large then the neuron will saturate. So neurons get stuck with a few large inputs OR many small ones.

It's easy for sigmoid units to saturate

- If there are 500 non-zero inputs initialized with a Gaussian ~N(0,1) then the SD is $\sqrt{500} \approx 22.4$



It's easy for sigmoid units to saturate



- Saturation visualization from Glorot & Bengio 2010 -- using a smarter initialization scheme $W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$

Closest-to-output hidden layer still stuck for first 100 epochs

WHAT'S DIFFERENT ABOUT MODERN ANNS?

Some key differences

- Use of softmax and entropic loss instead of quadratic loss.
- Use of alternate non-linearities
  - reLU and hyperbolic tangent
- Better understanding of weight initialization
- Data augmentation
  - Especially for image data

Cross-entropy loss

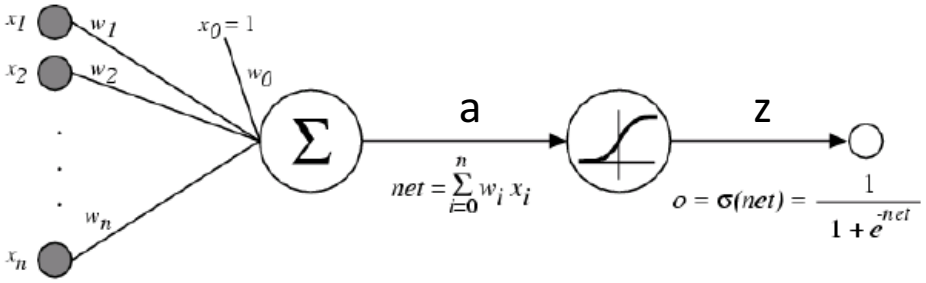
$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)],$$

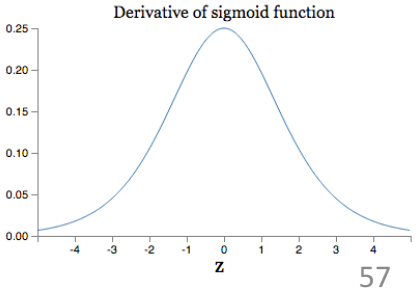
$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z) - y).$$

Compare to gradient for square loss when a~=1 y=0 and x=1

$$\frac{\partial C}{\partial w} = \sigma(z) - y$$

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$





Cross-entropy loss

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)],$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$

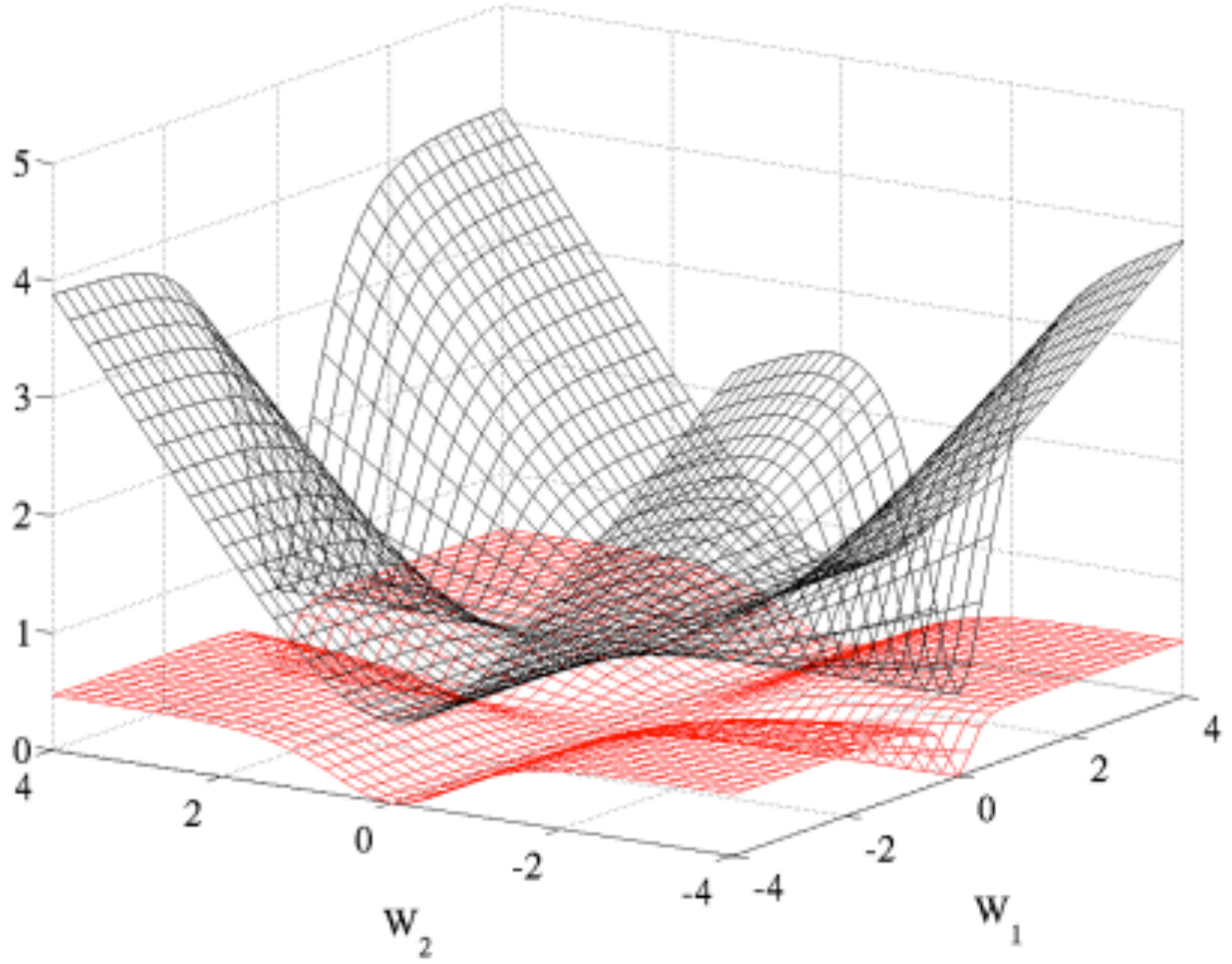
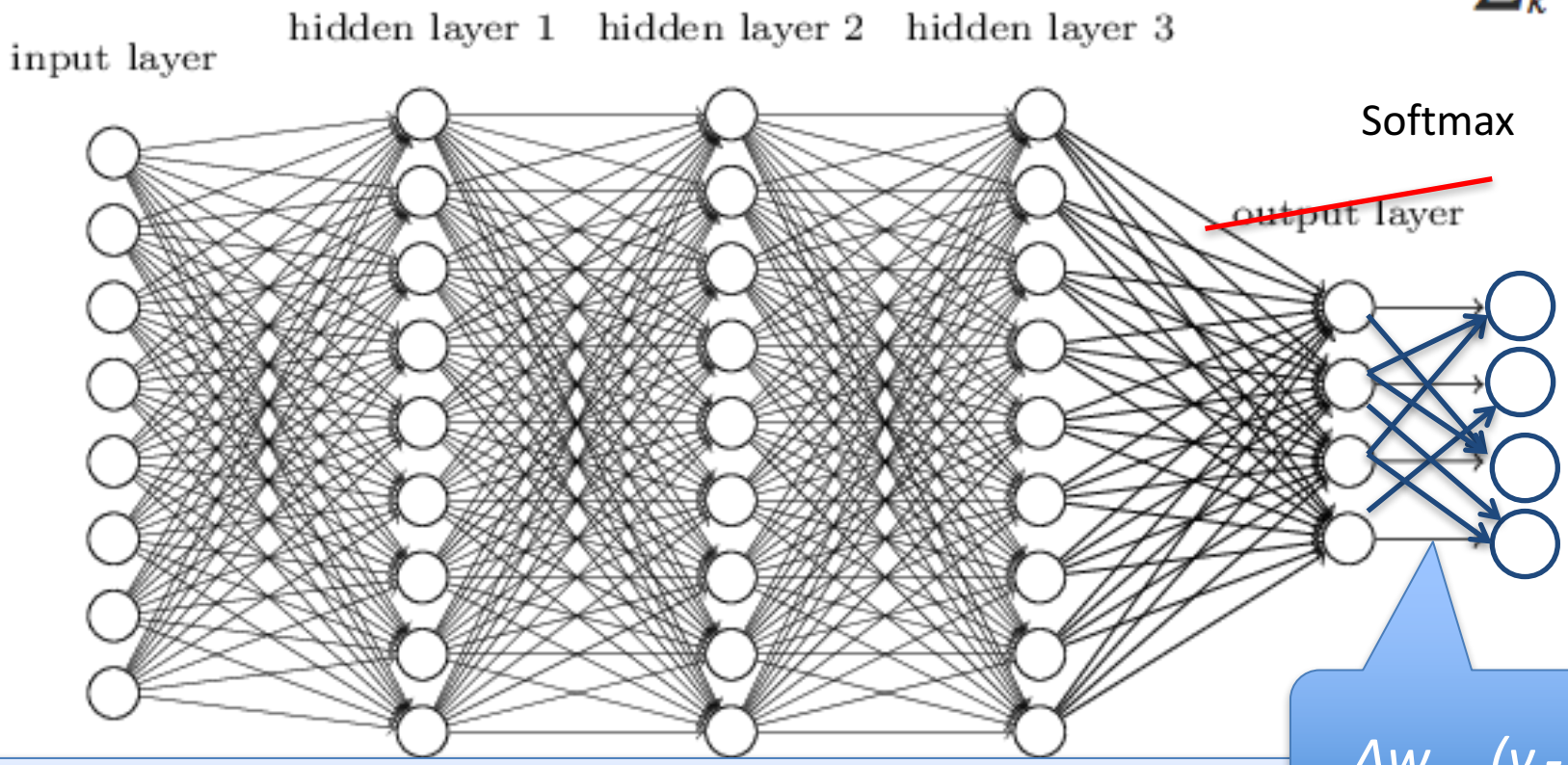


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Softmax output layer

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$



$\Delta w_{ij} = (y_i - z_i) y_j$

Network outputs a probability distribution!

Cross-entropy loss after a softmax layer gives a very simple, numerically stable gradient

Some key differences

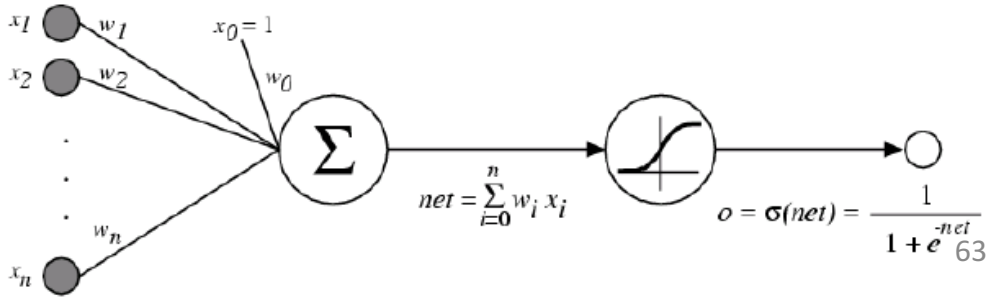
- Use of softmax and entropic loss instead of quadratic loss.
  - Often learning is faster and more stable as well as getting better accuracies in the limit
- Use of alternate non-linearities
- Better understanding of weight initialization
- Data augmentation
  - Especially for image data

Some key differences

- Use of softmax and entropic loss instead of quadratic loss.
  - Often learning is faster and more stable as well as getting better accuracies in the limit
- Use of alternate non-linearities
  - reLU and hyperbolic tangent
- Better understanding of weight initialization
- Data augmentation
  - Especially for image data

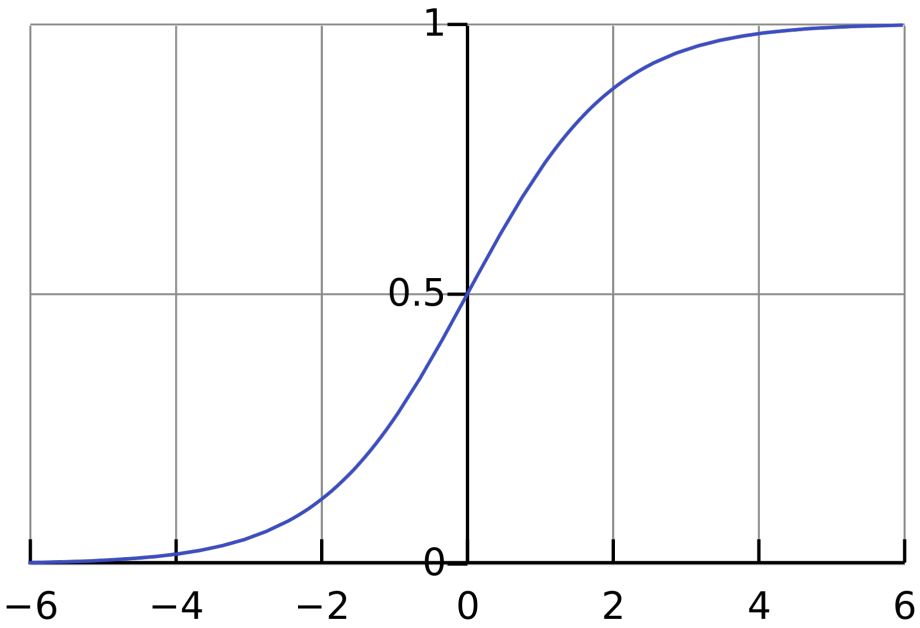
Alternative non-linearities

- Changes so far
  - Changed the **loss** from square error to cross-entropy
  - Proposed adding another output layer (softmax)
- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs



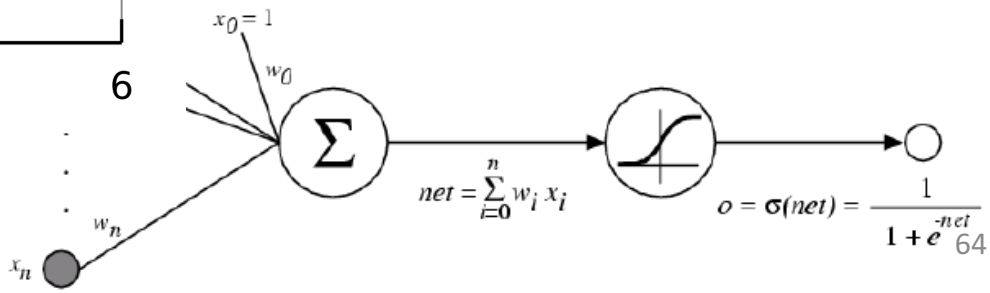
Alternative non-linearities

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs



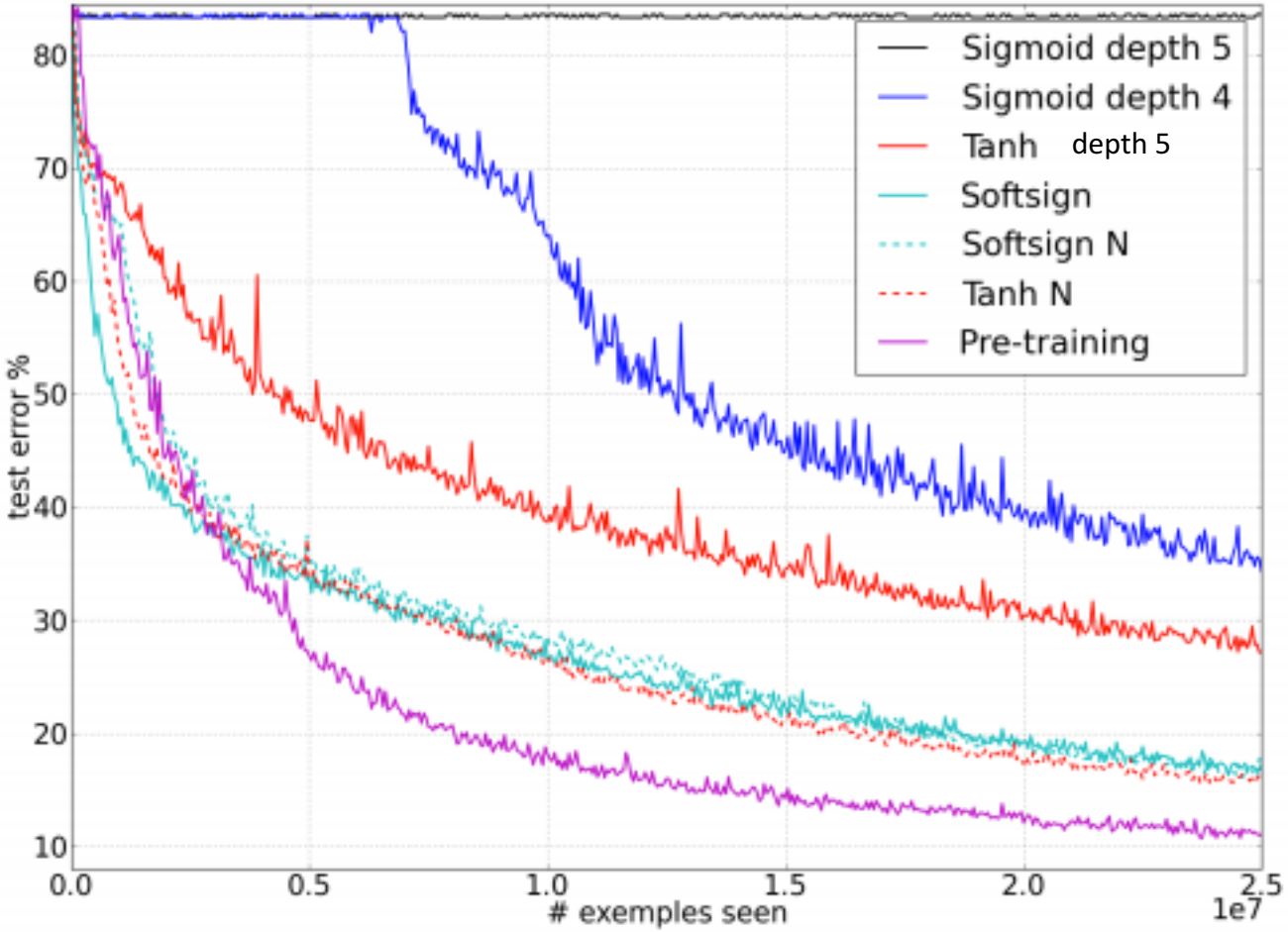
Alternate 1:
tanh

Like logistic function but shifted to range [-1, +1]



Understanding the difficulty of training deep feedforward neural networks

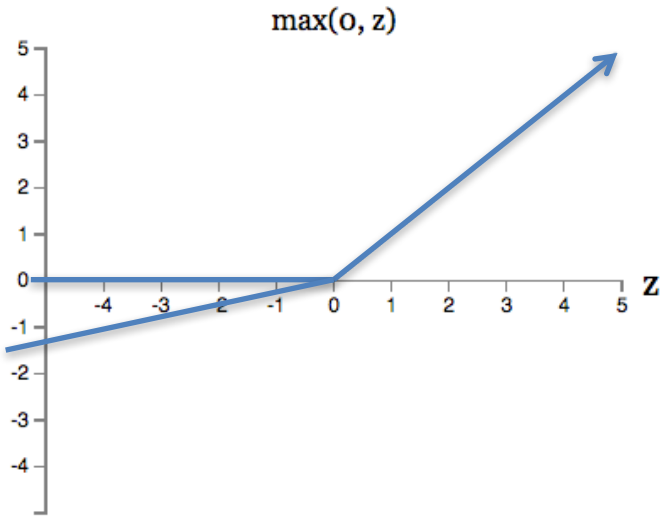
AI Stats 2010



We will get to these tricks eventually….

Alternative non-linearities

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

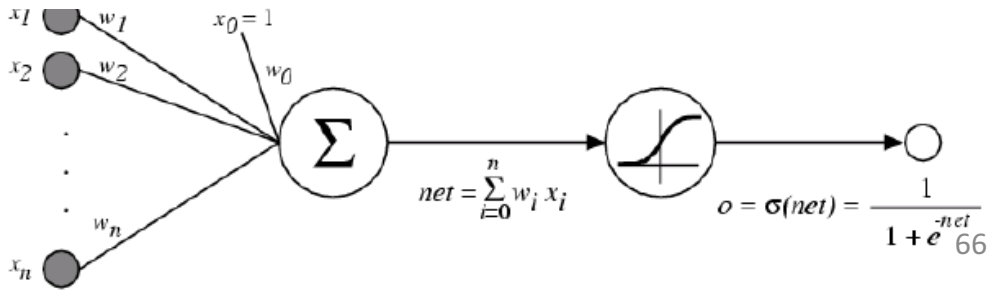


Alternate 2: rectified linear unit

Linear with a cutoff at zero

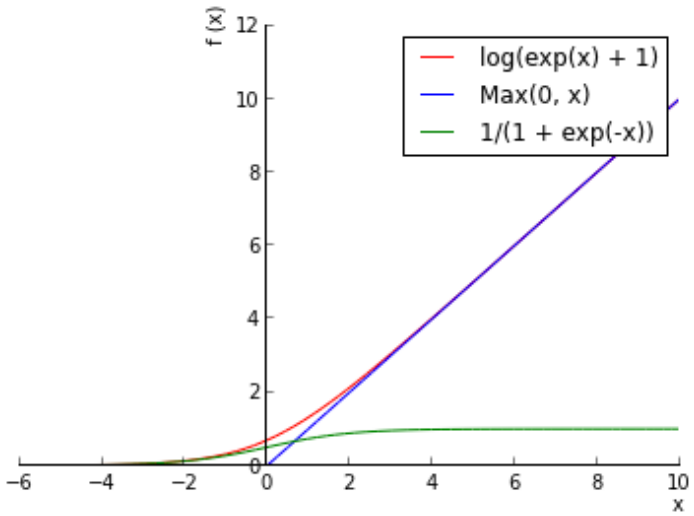
(Implement: clip the gradient when you pass zero)

$$\max(0, w \cdot x + b).$$



Alternative non-linearities

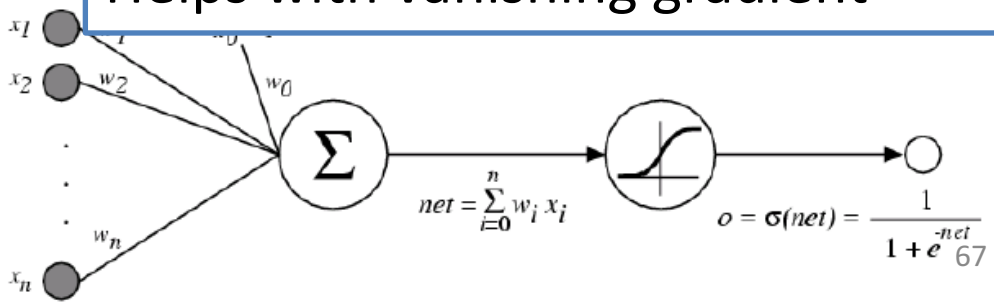
- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



Alternate 2: rectified linear unit

Soft version: log(exp(x)+1)

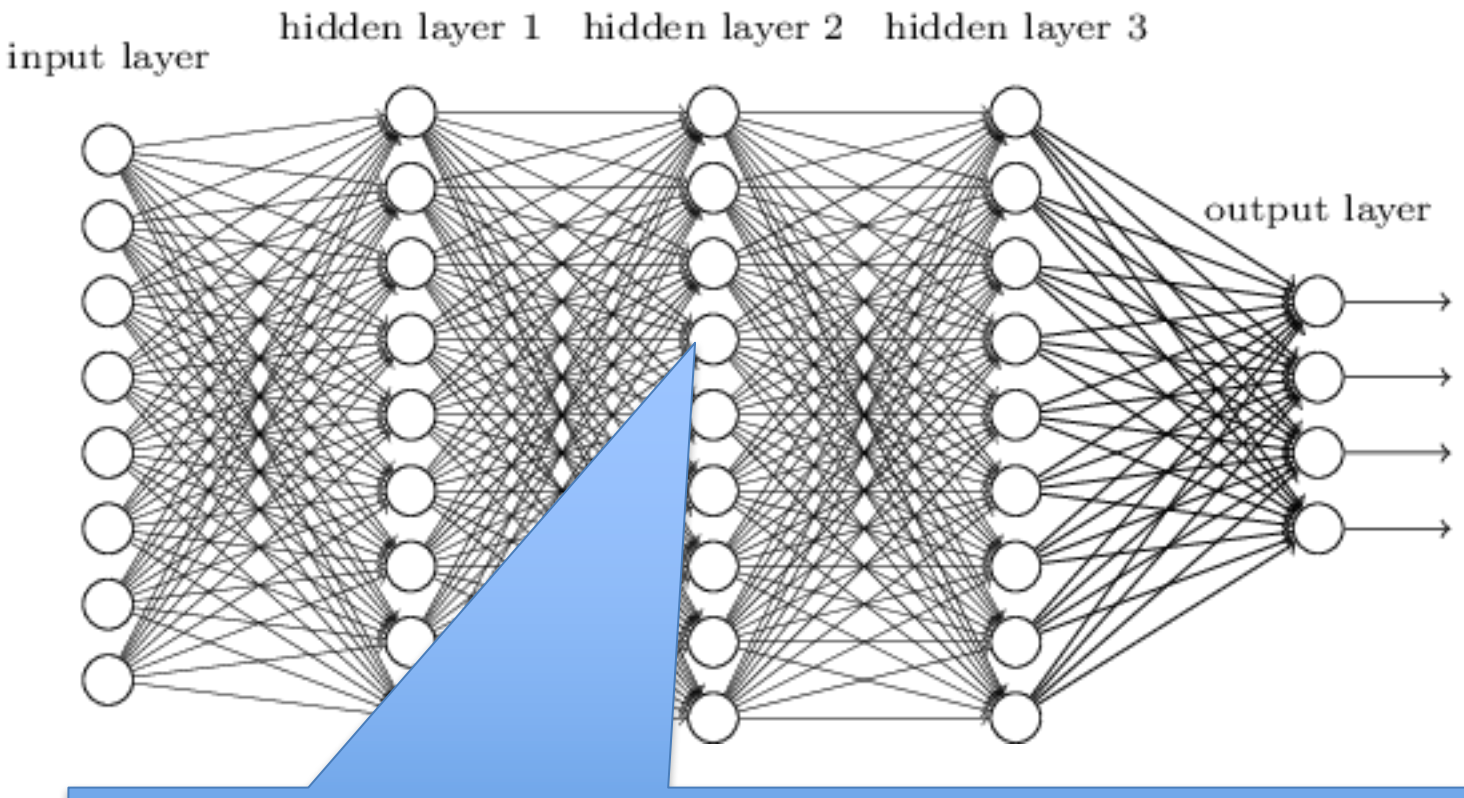
Doesn't saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient



Some key differences

- Use of softmax and entropic loss instead of quadratic loss.
  - Often learning is faster and more stable as well as getting better accuracies in the limit
- Use of alternate non-linearities
  - reLU and hyperbolic tangent
- Better understanding of weight initialization
- Data augmentation
  - Especially for image data

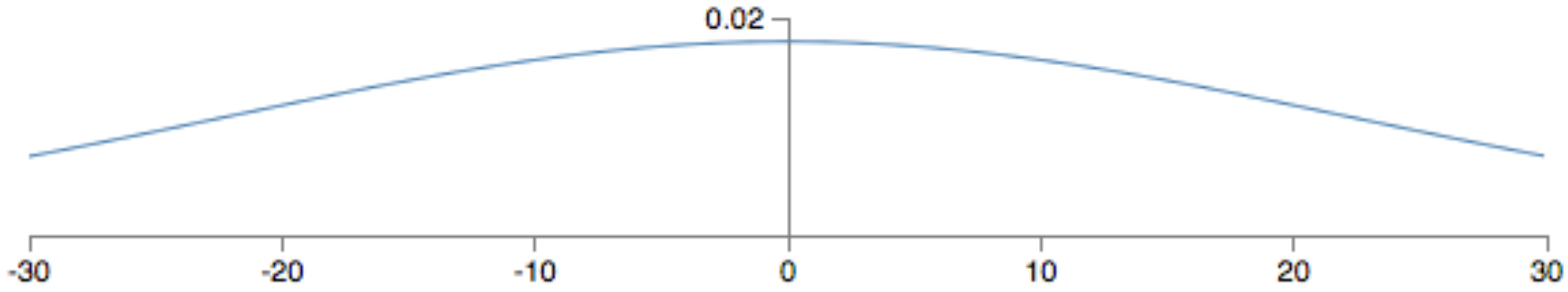
It's easy for sigmoid units to saturate



For a big network there are lots of weighted inputs to each neuron. If any of them are too large then the neuron will saturate. So neurons get stuck with a few large inputs OR many small ones.

It's easy for sigmoid units to saturate

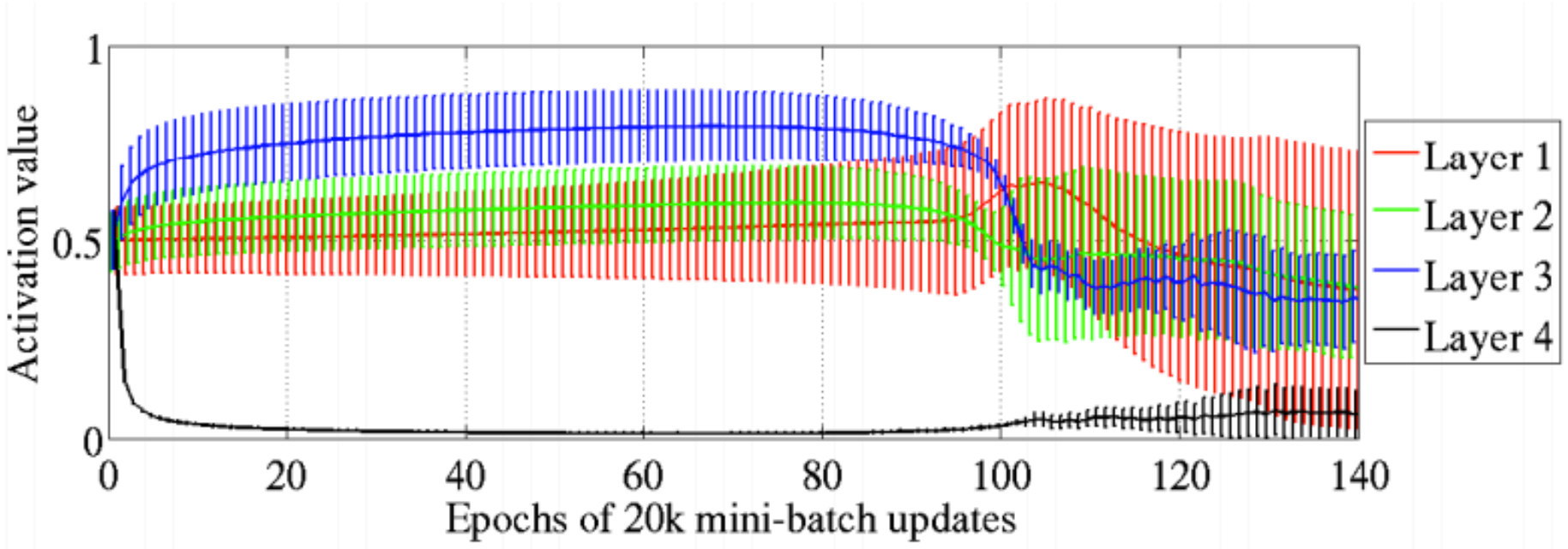
- If there are 500 non-zero inputs initialized with a Gaussian ~N(0,1) then the SD is $\sqrt{500} \approx 22.4$



- Common heuristics for initializing weights:

$$N\left(0, \frac{1}{\sqrt{\#\text{inputs}}}\right) \qquad U\left(\frac{-1}{\sqrt{\#\text{inputs}}}, \frac{-1}{\sqrt{\#\text{inputs}}}\right)$$

It's easy for sigmoid units to saturate



- Saturation visualization from Glorot & Bengio 2010 using

$$U\left(\frac{-1}{\sqrt{\#\text{inputs}}}, \frac{-1}{\sqrt{\#\text{inputs}}}\right)$$

Initializing to avoid saturation

- In C… if level *j* (w…

First breakthrough deep learning results were based on clever pre-training initialization schemes, where deep networks were seeded with weights learned from unsupervised strategies

TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

This is not always the solution – but good initialization is very important for deep nets!