Machine Learning 10-601/301

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

March 8, 2021

This section:
- Artificial neural networks
- Backpropagation
- Representation learning

Reading:
- Goodfellow: Chapter 6
- optional: Mitchell: Chapter 4

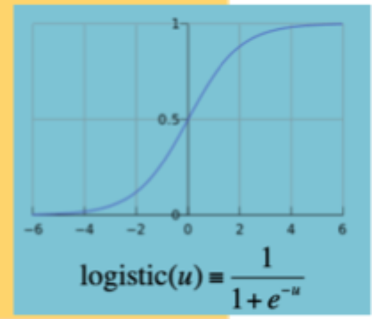
Logistic Regression

Data: Inputs are continuous vectors of length M. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^M \text{ and } y \in \{0,1\}$$

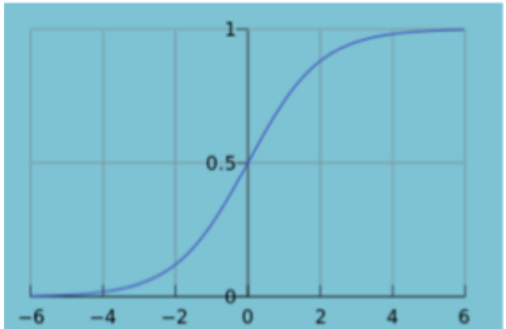
Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Logistic Regression



$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

We like logistic regression, but

$$P(Y=1|X=<X_1,...X_n>)=\frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

- how would it perform when trying to learn P(image contains Hillary Clinton | pixel values X_1, X_2 ... $X_{10,000}$)

We like logistic regression, but

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

what X_i image features to use? edges? color blotches? generic face? subwindows? lighting invariant properties? position independent? SIFT features?

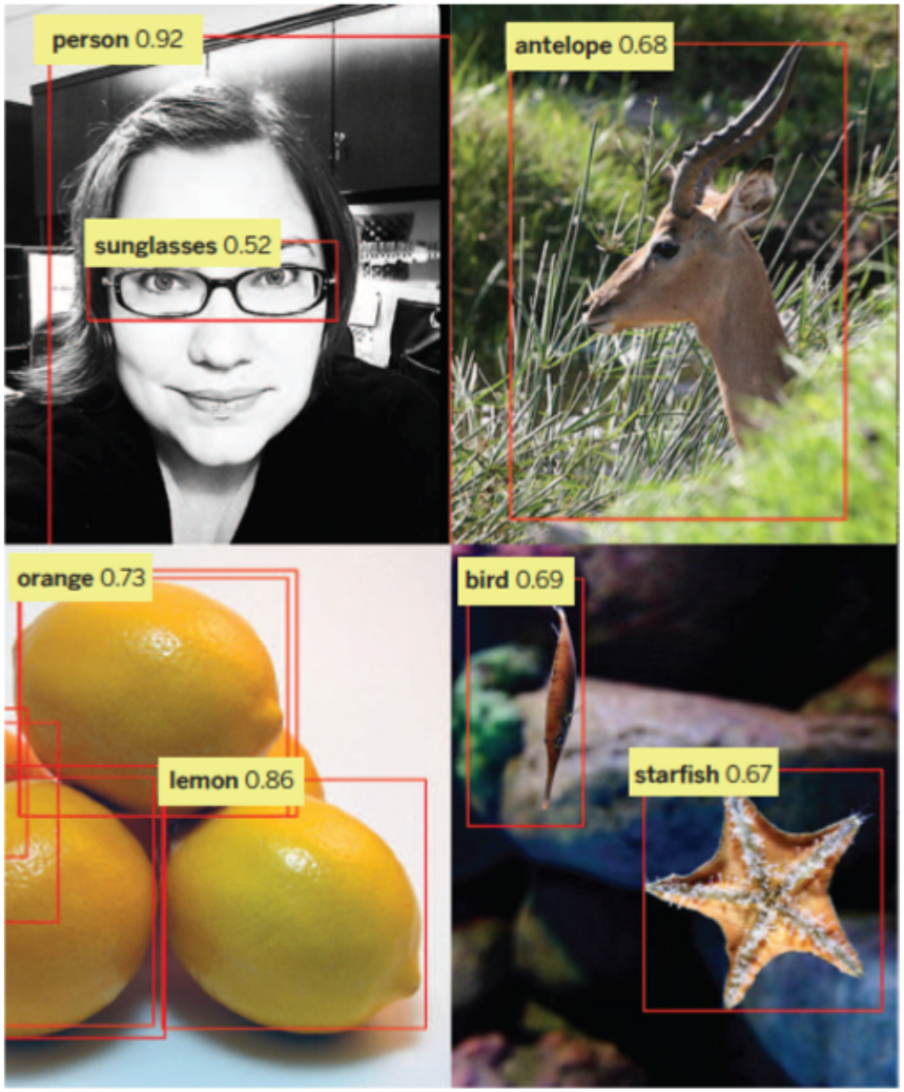
We like logistic regression, but

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

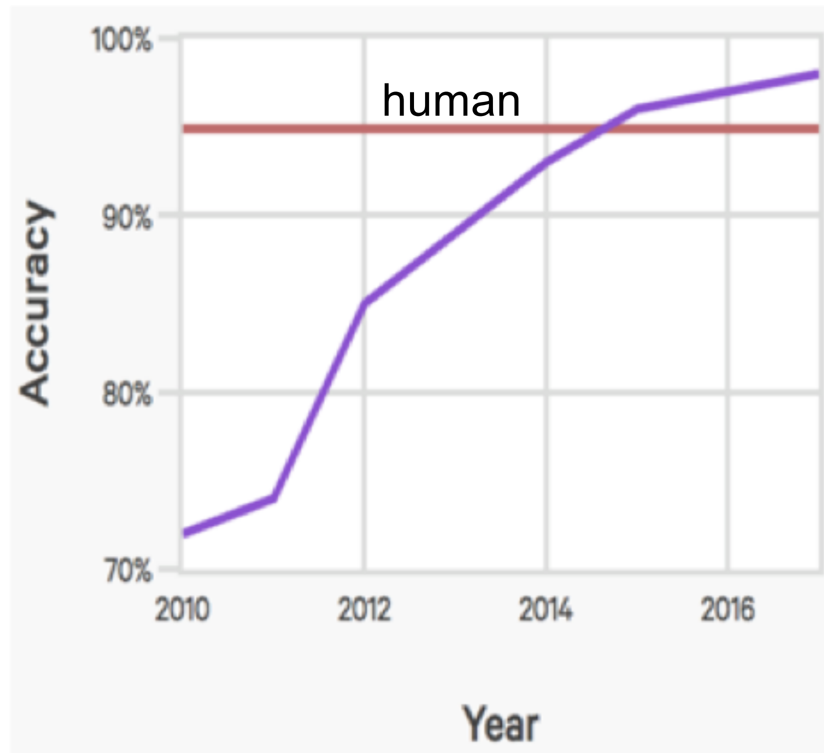
what X_i image features to use? edges? color blotches? generic face? subwindows? lighting invariant properties? position independent?

Deep nets : learn the features automatically !

Computer Vision



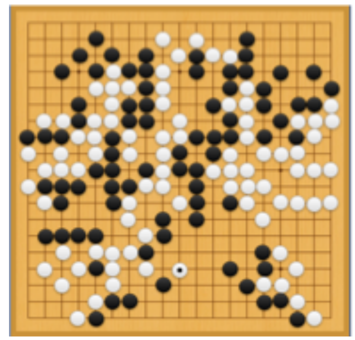
Imagenet Visual Recognition Challenge



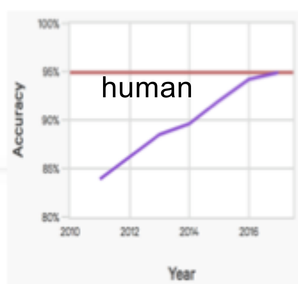
Machine Translation

Vision

Strategic reasoning



Speech to Text



Robots

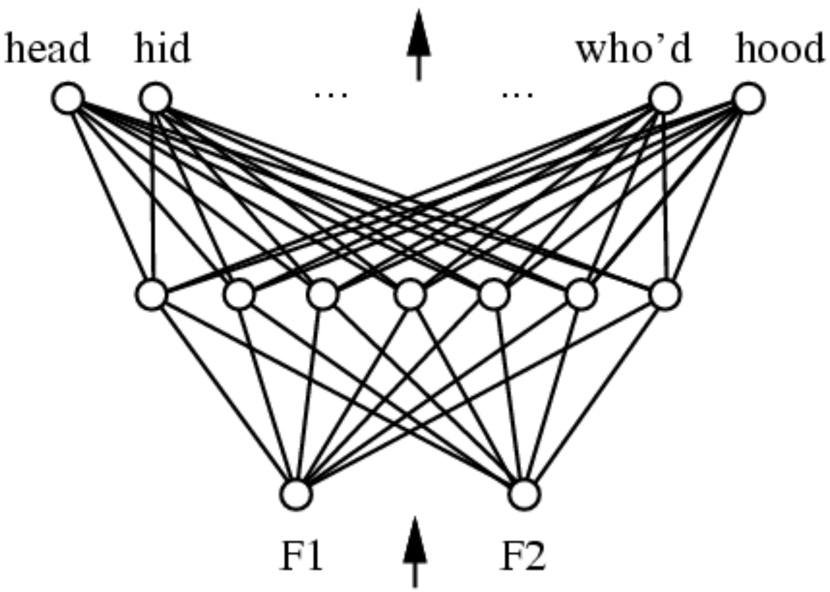
Artificial Neural Networks learn f: X → Y

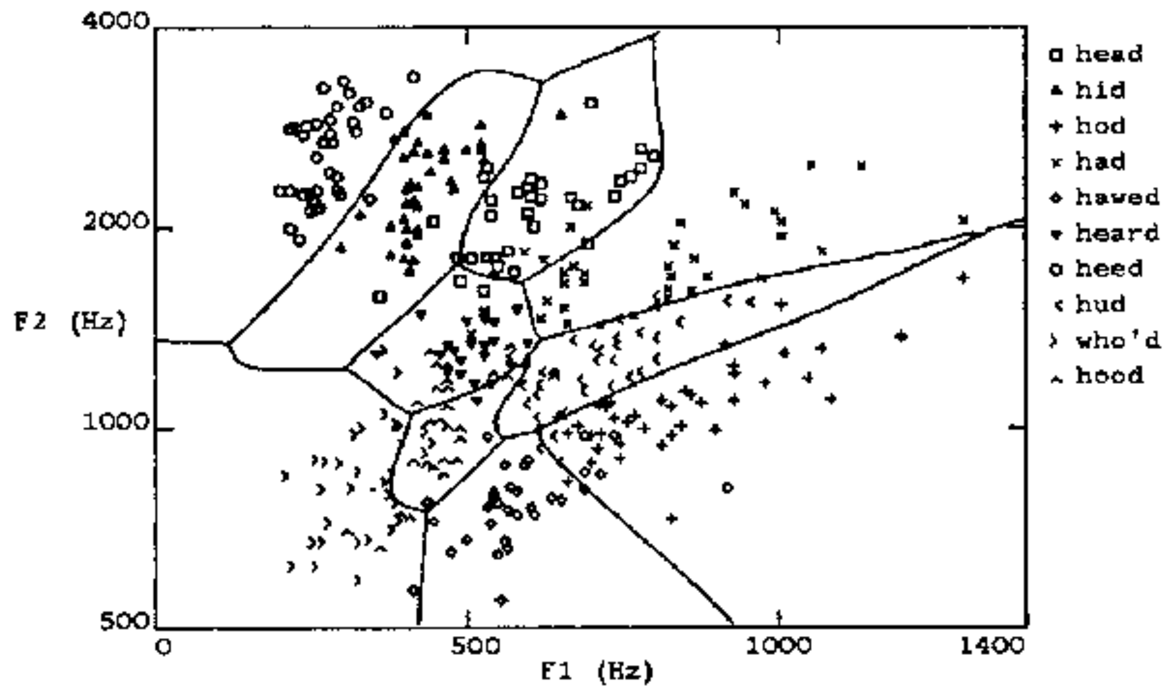
- f might be complex, non-linear, real or discrete-valued
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars
  - X = image, Y = new image if you drive forward 1 meter
  - X = microphone signal, Y = which word was spoken

- we can also train the network to learn P(Y|X)
  - Logistic regression is just a special case of a neural network

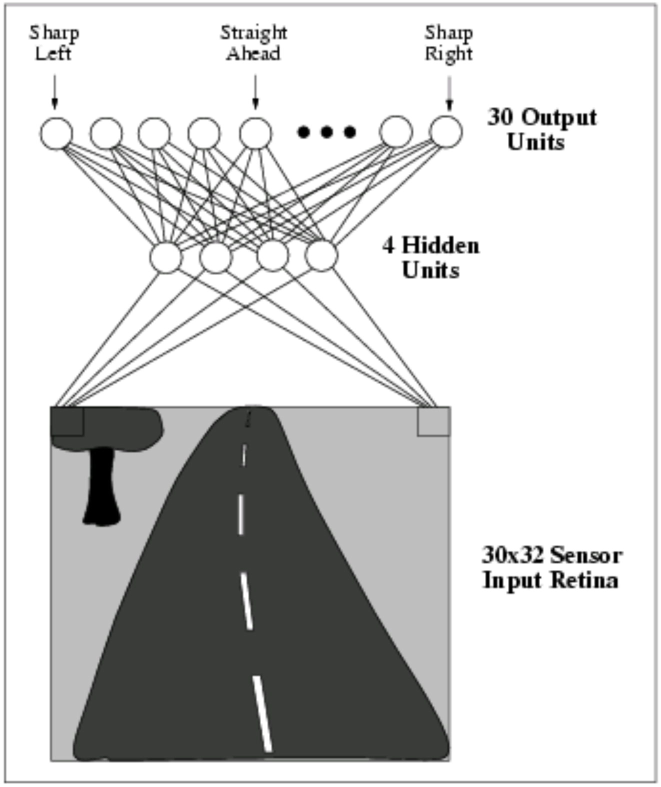
- Represent f by *network* of computational units which may contain billions of trained parameters

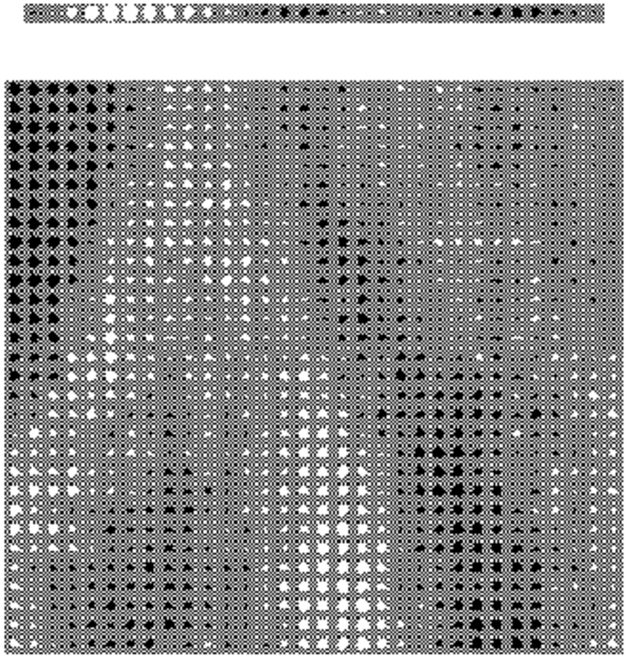
Multilayer Networks of Sigmoid Units





ALVINN
[Pomerleau 1993]





Caption Generation

LZ

a car is parked in
the middle of nowhere .

a wooden table and chairs
arranged in a room .

there is a cat sitting on a shelf .

a ferry boat on a marina
with a group of people .

a little boy with a bunch
of friends on the street .

[from Russ Salakhutdinov]

Caption Generation

the two birds are trying to be seen in the water . (can't count)

a giraffe is standing next to a fence in a field . (hallucination)

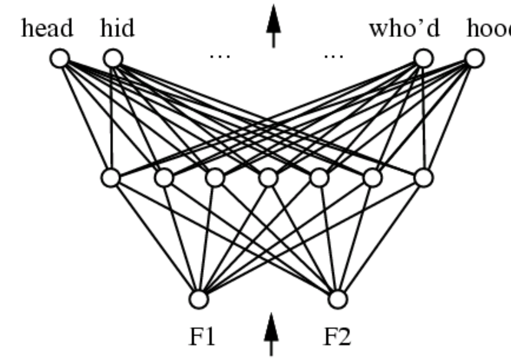
a parked car while driving down the road . (contradiction)

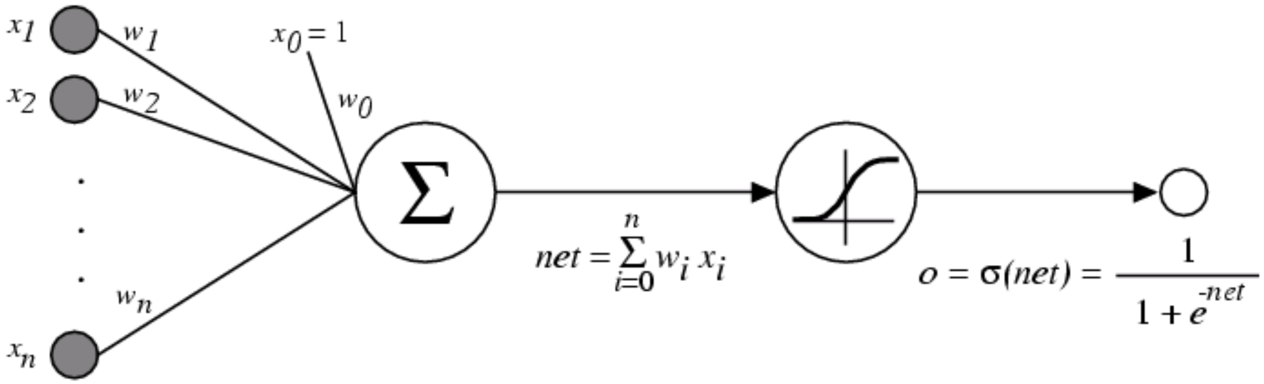
the handlebars are trying to ride a bike rack . (nonsensical)

a woman and a bottle of wine in a garden . (gender)

[from Russ Salakhutdinov]

Sigmoid Unit





$\sigma(x)$ is the sigmoid function

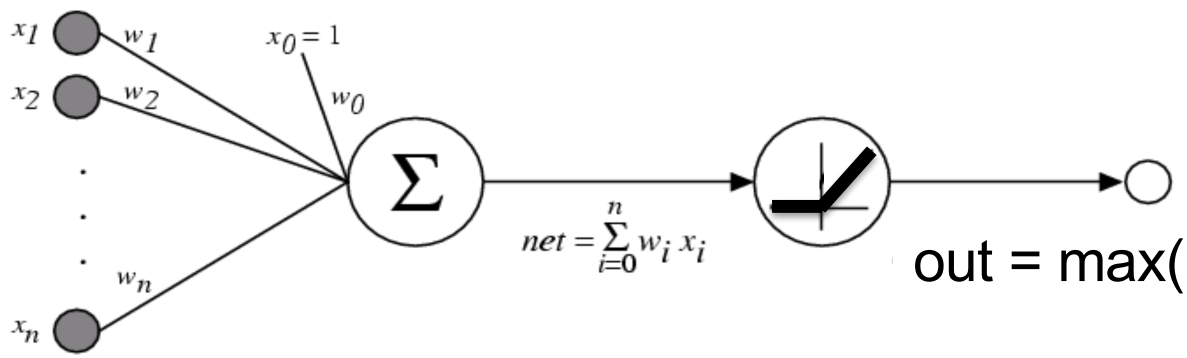
$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

Sigmoid units are exactly the form we use for logistic regression!

Many kinds of units

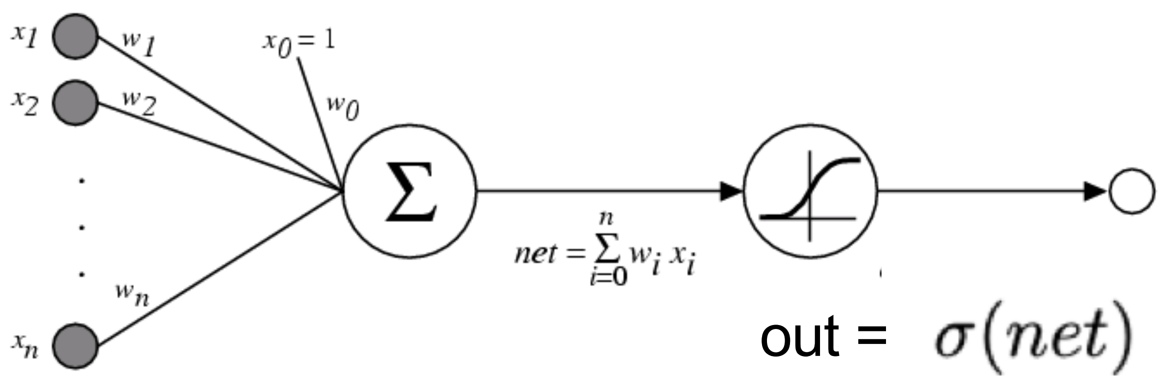
- Rectified linear unit : linear with thresholded output



out = max(net,0)

= max($\mathbf{W}^{\mathrm{T}}\mathbf{x}$, 0)

- Sigmoid unit:

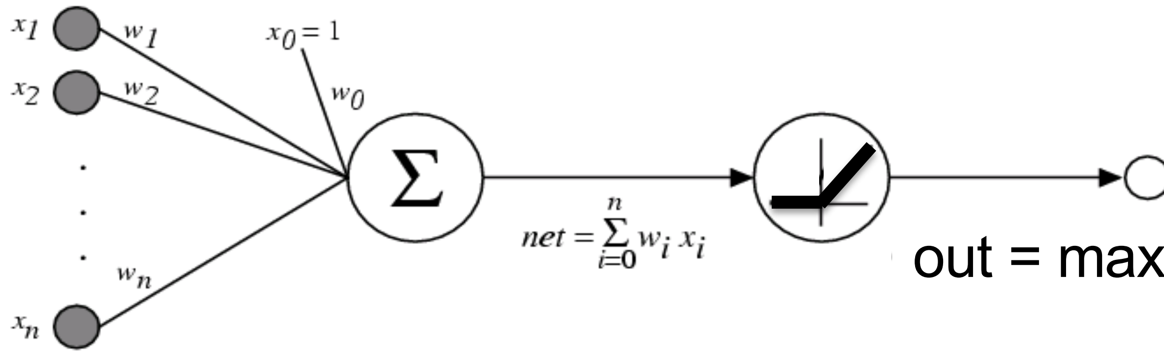


out = $\sigma(net)$

$$= \frac{1}{1 + \exp(-\mathbf{W}^{\mathrm{T}}\mathbf{x})}$$

Units often constructed so that output is of the form f(g(x)).

- **Rectified linear unit** : linear with thresholded output



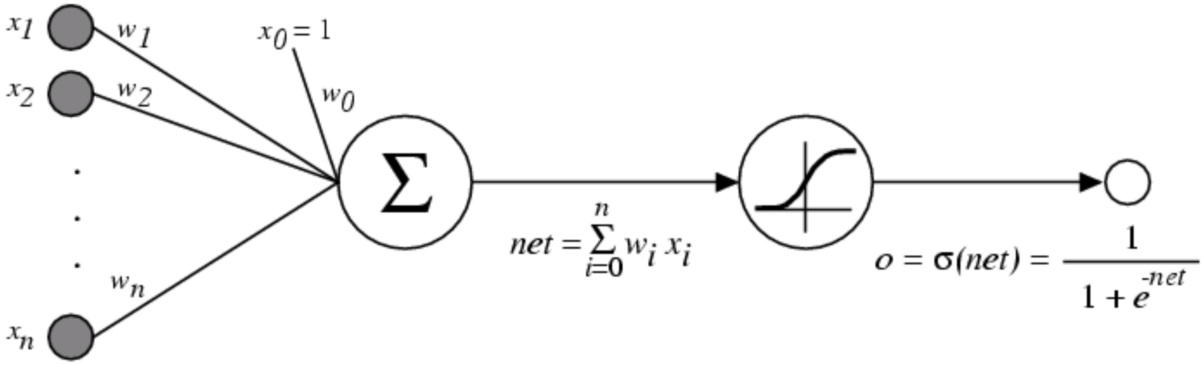
out = max(net,0)

= max($\mathbf{W^T x}$, 0)

- g(x) = net = $\mathbf{W^T x}$
- f(g(x)) = max($\mathbf{W^T x}$, 0)

Units often constructed so that output is of the form f(g(x)).

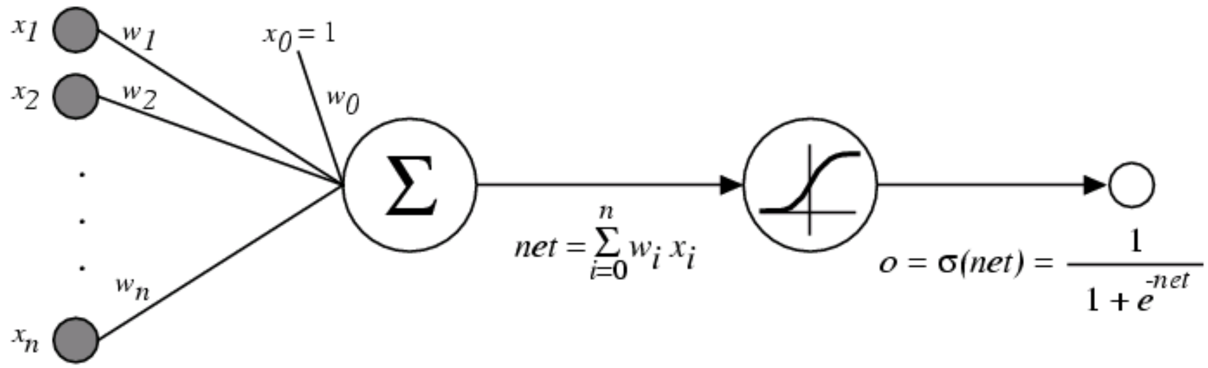
- **Sigmoid unit** : linear with thresholded output



- g(x) = net = $\mathbf{W}^{\mathbf{T}}\mathbf{x}$
- f(g(x)) = $\dfrac{1}{1+\exp(-\mathbf{W}^{\mathbf{T}}\mathbf{x})}$

Units often constructed so that output is of the form f(g(x)).

- **Sigmoid unit** : linear with thresholded output



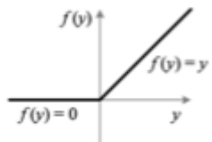
- g(x) = net = $\mathbf{W^T x}$
- f(g(x)) = $\dfrac{1}{1 + \exp(-\mathbf{W^T x})}$
- Use chain rule to compute gradients!

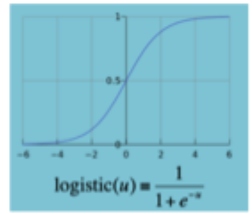
$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

$$\frac{\partial f(g(x))}{\partial w_i} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial w_i}$$

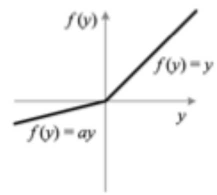
Many types of parameterized units

- Sigmoid units
- ReLU





- Leaky ReLU (fixed non-zero slope for input<0)
- Parametric ReLU (trainable slope)



- Max Pool
- Inner Product
- GRU's
- LSTM's
- Matrix multiply
- …. no end in sight

Any unit h(X; W) whose output is differentiable w.r.t. X and W

Training Deep Nets

1. Choose loss function $J(\theta)$ to optimize
   - sum of squared errors for y continuous: $\Sigma\,(y - h(x;\theta))^2$
   - maximize conditional log likelihood: $\Sigma \log P(y|x;\theta)$
   - MAP estimate: $\Sigma \log P(y|x;\theta)\, P(\theta)$
   - ~~0/1 loss. Sum of classification errors: $\Sigma\, \delta(y = h(x;\theta)$~~
   - ...
2. Design network architecture
   - Network of layers (ReLU's, sigmoid, convolutions, ...)
   - Widths of layers
   - Fully or partly interconnected
   - ...
3. Training algorithm
   - Derive gradient components $\frac{\partial J(\theta)}{\partial \theta_i}$
   - Choose gradient descent method, including stopping condition
   - (optional: Experiment with alternative network architectures)

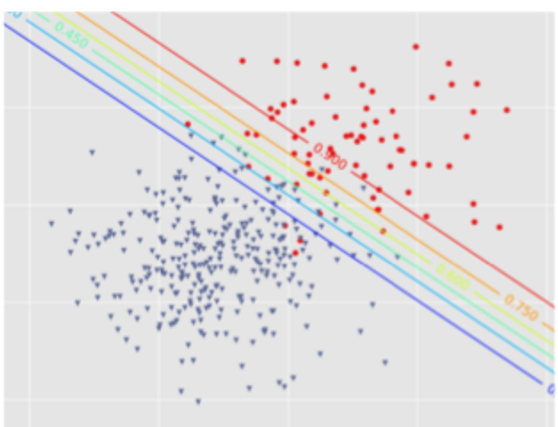
Example

Example: Learn probabilistic XOR

- Given boolean Y, X_1, X_2 learn $P(Y|X_1,X_2)$, where

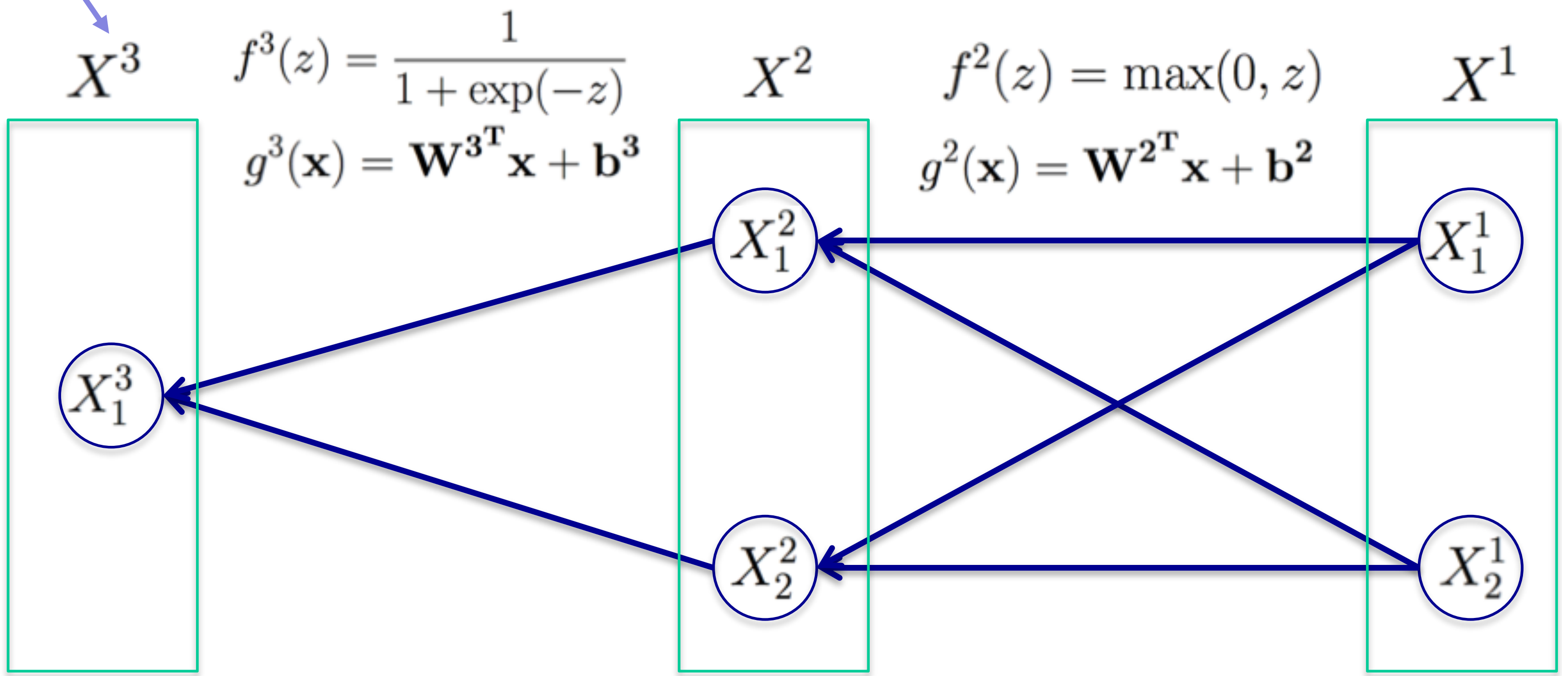
X_1	X_2	$P(Y=1|X_1,X_2)$	$P(Y=0|X_1,X_2)$
0	0	0.1	0.9
1	0	0.9	0.1
0	1	0.9	0.1
1	1	0.1	0.9

- Can we learn this with logistic regression?



Superscript denotes layer

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$



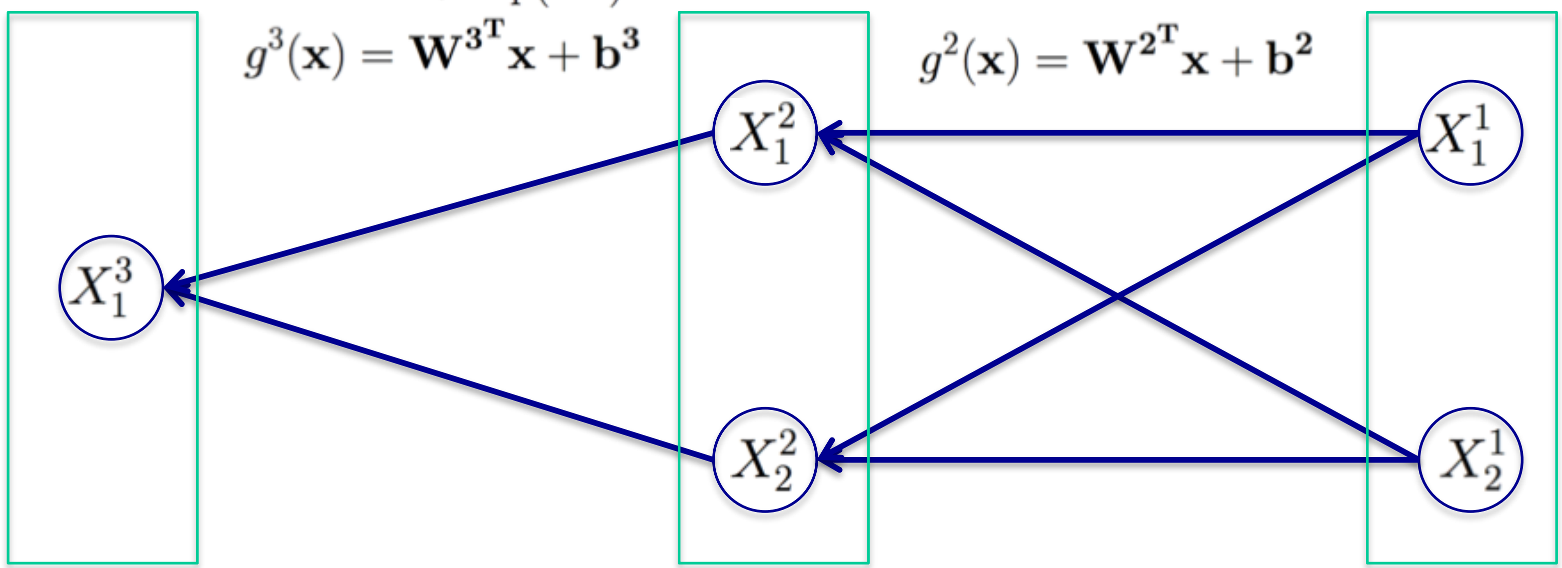
where $X_1^3 = P(Y = 1 | X = \mathbf{X^1})$

Superscript denotes layer

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$

X^3 $\quad f^3(z) = \dfrac{1}{1+\exp(-z)}$ $\quad X^2$ $\quad f^2(z) = \max(0, z)$

$g^3(\mathbf{x}) = \mathbf{W^3}^{\mathbf{T}}\mathbf{x} + \mathbf{b^3}$ $\quad g^2(\mathbf{x}) = \mathbf{W^2}^{\mathbf{T}}\mathbf{x} + \mathbf{b^2}$

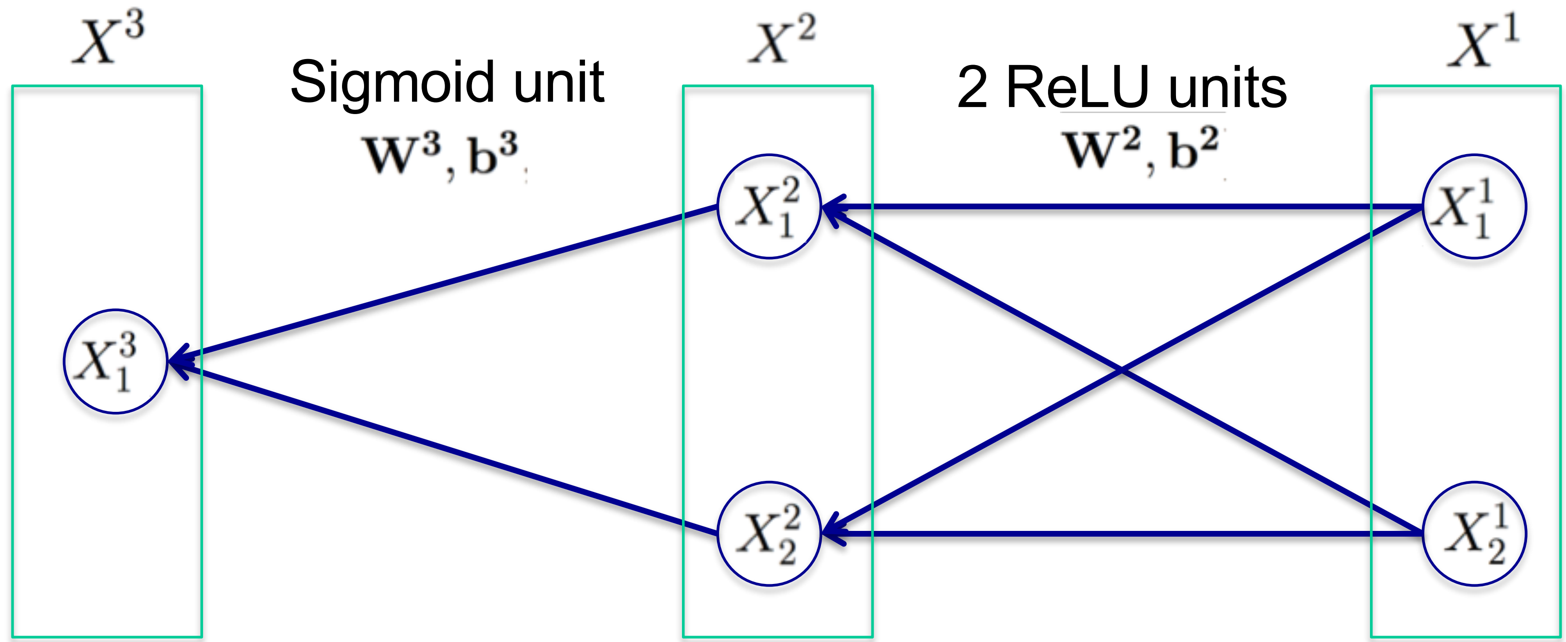


Loss function $J(\theta)$ to be minimized: negative log likelihood

$$J(\theta) = \sum_{\langle \mathbf{x}, y\rangle \in D} -\log P(Y = y | X = \mathbf{x})$$

Same as logistic regression

where $X_1^3 = P(Y = 1 | X = \mathbf{X^1})$, $\theta = \{\mathbf{W^3}, \mathbf{b^3}, \mathbf{W^2}, \mathbf{b^2}\}$



Loss function $J(\theta)$ to be minimized: negative log likelihood of training data D

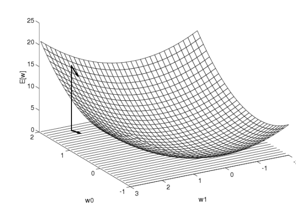
$$J(\theta) = \sum_{\langle x,y\rangle \in D} -\log P_\theta(Y = y|X = x;\ \theta)$$

where we define

$$X_1^3 = P(Y = 1|X^1; \theta), \quad \theta = \{\mathbf{W^3}, \mathbf{b^3}, \mathbf{W^2}, \mathbf{b^2}\}$$

gradient

$$\nabla_\theta J(\theta) = \langle \frac{\partial J(\theta)}{\partial W_1^3}, \frac{\partial J(\theta)}{\partial W_2^3}, \frac{\partial J(\theta)}{\partial b^3}, \ldots, \frac{\partial J(\theta)}{\partial b_1^2} \rangle$$



note by chain rule

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{\partial J(\theta)}{\partial X_1^3} \cdot \frac{\partial X_1^3}{\partial \theta_i}$$

Feed Forward

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$

X^3

$$f^3(z) = \frac{1}{1+\exp(-z)}$$

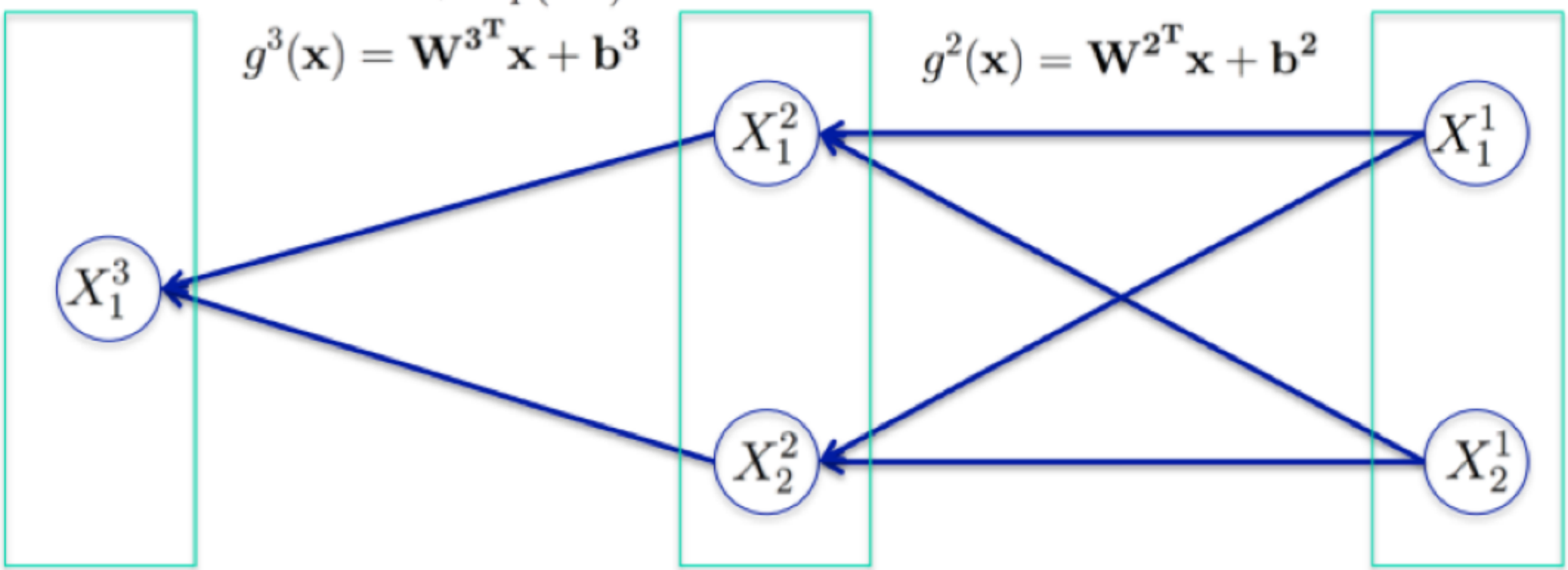
$$g^3(\mathbf{x}) = \mathbf{W^{3^T}}\mathbf{x} + \mathbf{b}^3$$

X^2

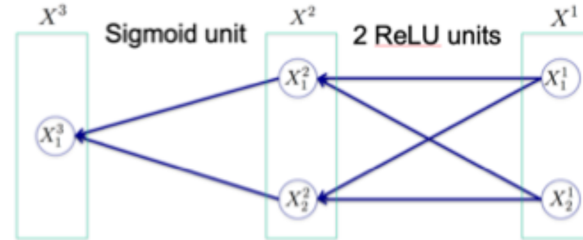
$$f^2(z) = \max(0, z)$$

$$g^2(\mathbf{x}) = \mathbf{W^{2^T}}\mathbf{x} + \mathbf{b}^2$$

X^1



X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*



Derive the gradient using chain rule

$$J(\theta) = -\sum_k \log P(Y = y_k | X = \mathbf{x}_k)$$
$$= -\sum_k y_k \log P(Y = 1 | X = x_k) + (1 - y_k) \log(1 - P(Y = 1 | X = x_k))$$

simplify notation by considering just one training example

$$\frac{\partial J(\theta)}{\partial X_1^3} = \frac{\partial\left(-Y \log P(Y=1|X) - (1-Y)\log(1-P(Y=1|X))\right)}{\partial X_1^3}$$
$$= \frac{\partial\left(-Y \log X_1^3 - (1-Y)\log(1-X_1^3)\right)}{\partial X_1^3}$$
$$= \frac{-Y}{X_1^3} - (1-Y)\frac{1}{1-X_1^3}(-1)$$
$$= \frac{-Y}{X_1^3} + \frac{(1-Y)}{1-X_1^3}$$

$$\boxed{\text{recall } \frac{\partial \ln z}{\partial z} = \frac{1}{z}}$$

Derive the gradient we need

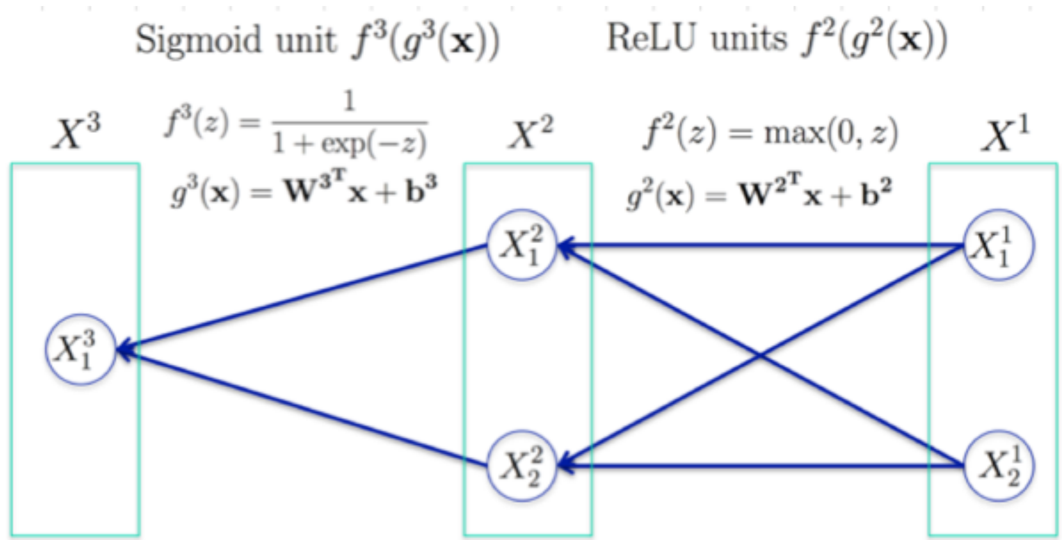
$$
\begin{aligned}
J(\theta) &= -\sum_k \log P(Y = y_k | X = \mathbf{x}_k) \\
&= -\sum_k y_k \log P(Y = 1 | X = x_k) + (1 - y_k) \log(1 - P(Y = 1 | X = x_k))
\end{aligned}
$$

simplify notation by considering just one training example

$$
\begin{aligned}
\frac{\partial J(\theta)}{\partial X_1^3} &= \frac{\partial\left(-Y \log P(Y = 1|X) - (1 - Y) \log(1 - P(Y = 1|X))\right)}{\partial X_1^3} \\
&= \frac{\partial\left(-Y \log X_1^3 - (1 - Y) \log(1 - X_1^3)\right)}{\partial X_1^3} \\
&= \frac{-Y}{X_1^3} - (1 - Y) \frac{1}{1 - X_1^3}(-1) \\
&= \frac{-Y}{X_1^3} + \frac{(1 - Y)}{1 - X_1^3}
\end{aligned}
$$

$$\boxed{\text{recall } \frac{\partial \ln z}{\partial z} = \frac{1}{z}}$$

	X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
$Y_{true}=1$	0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
						0.00	-0.15	-0.20	0.10	0.05	0
						1					*1*

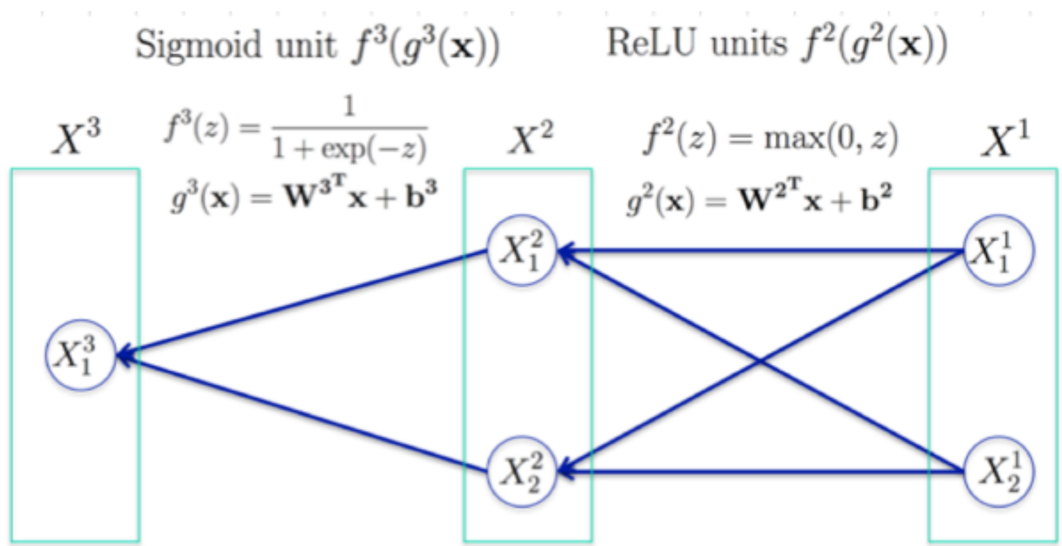


already know this

$$\frac{\partial J(\theta)}{\partial g_1^3} = \boxed{\frac{\partial J(\theta)}{\partial X_1^3}} \frac{\partial X_1^3}{\partial g^3} =$$

recall $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$
where $\sigma(z) = \frac{1}{1+\exp(-z)}$

	X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
$Y_{true}=1$	0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
						0.00	-0.15	-0.20	0.10	0.05	0
						1					*1*

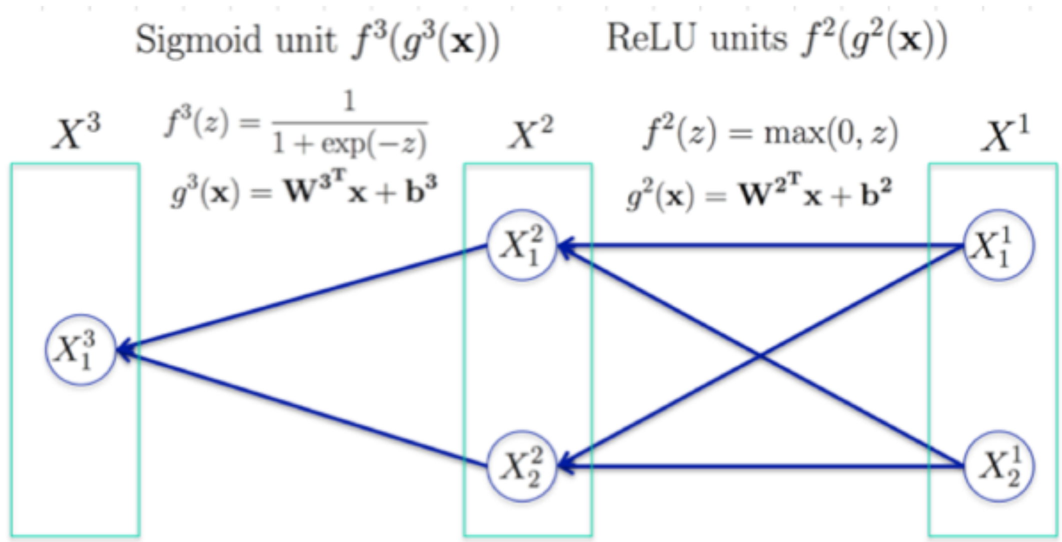


already know this

$$\frac{\partial J(\theta)}{\partial g_1^3} = \boxed{\frac{\partial J(\theta)}{\partial X_1^3}} \frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1 - X_1^3) =$$

recall $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$
where $\sigma(z) = \frac{1}{1+\exp(-z)}$

	X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
$Y_{true}=1$	0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
						0.00	-0.15	-0.20	0.10	0.05	0
						1					*1*

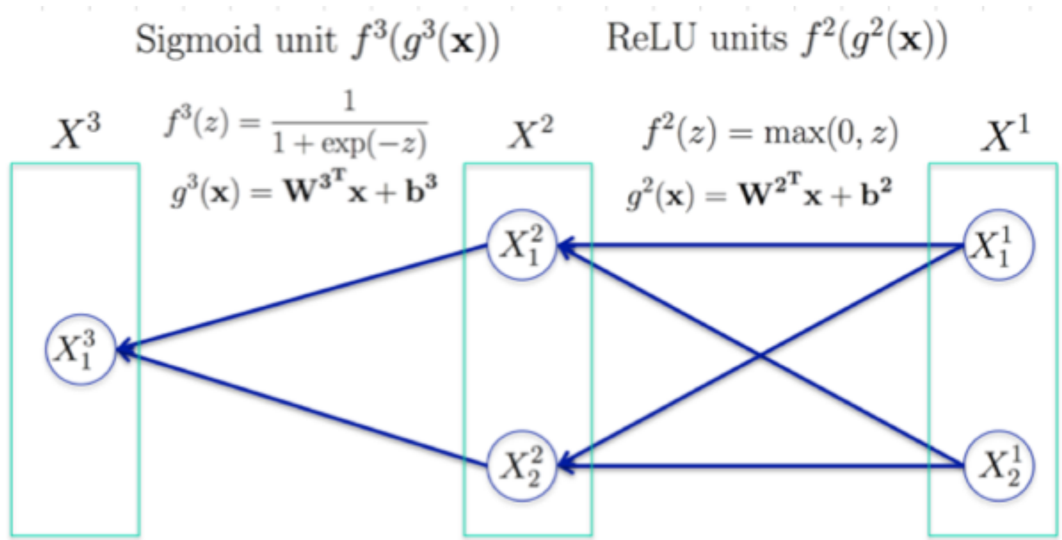


$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1-X_1^3) = -0.47$$

already know this

$$\frac{\partial J(\theta)}{\partial w_i^3} = \boxed{\frac{\partial J(\theta)}{\partial g^3}}\frac{\partial g^3}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3} X_i^2$$

	X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
$Y_{true}=1$	0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
						0.00	-0.15	-0.20	0.10	0.05	0
						1					***1***



$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1 - X_1^3)$$

$$\frac{\partial J(\theta)}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial w_i^3} = \frac{\partial J(\theta)}{\partial g^3} X_i^2$$

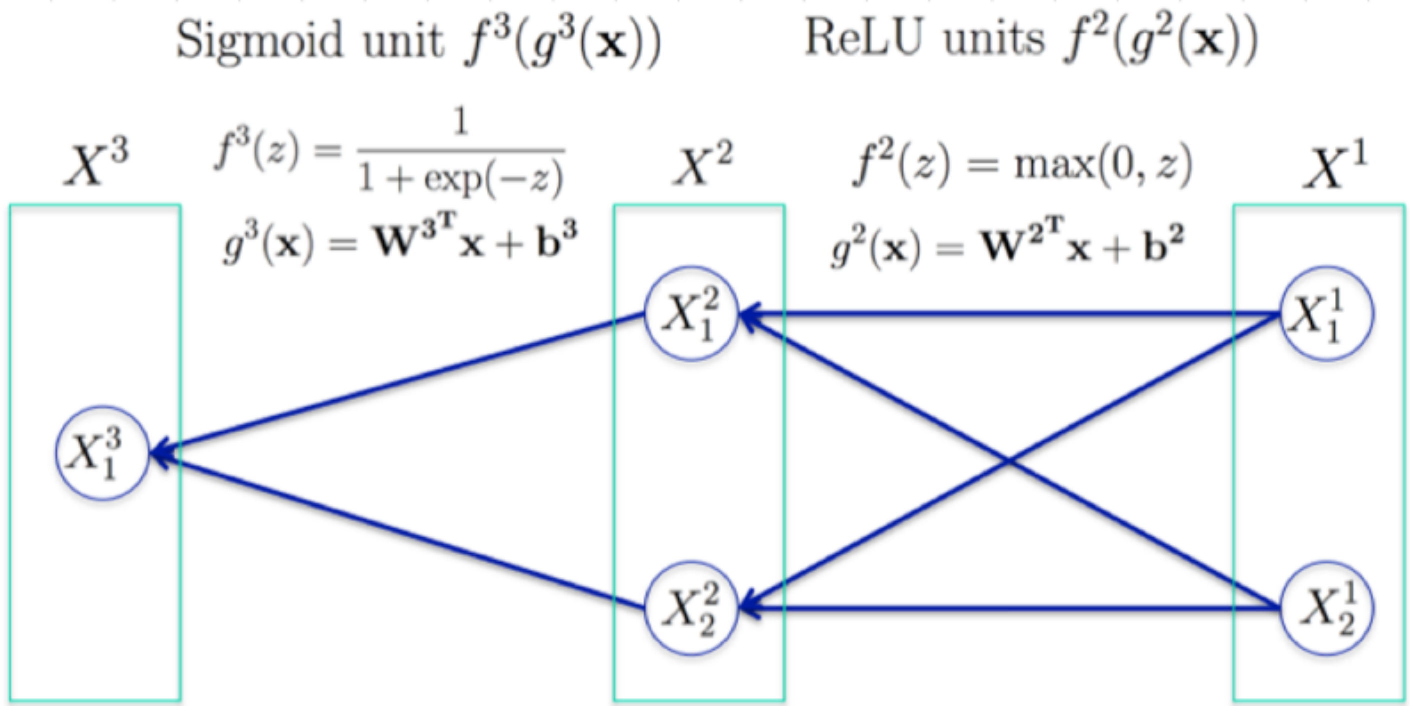
$$\frac{\partial J(\theta)}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3} w_i^3$$

$$\frac{\partial J(\theta)}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2}\frac{\partial X_i^2}{\partial g_i^2} = \frac{\partial J(\theta)}{\partial X_i^2} \times [\text{if } g_i^2 > 0 \text{ then } 1 \text{ else } 0]$$

$$\frac{\partial J(\theta)}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2}\frac{\partial g_i^2}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2} X_k^1$$

Back propagation

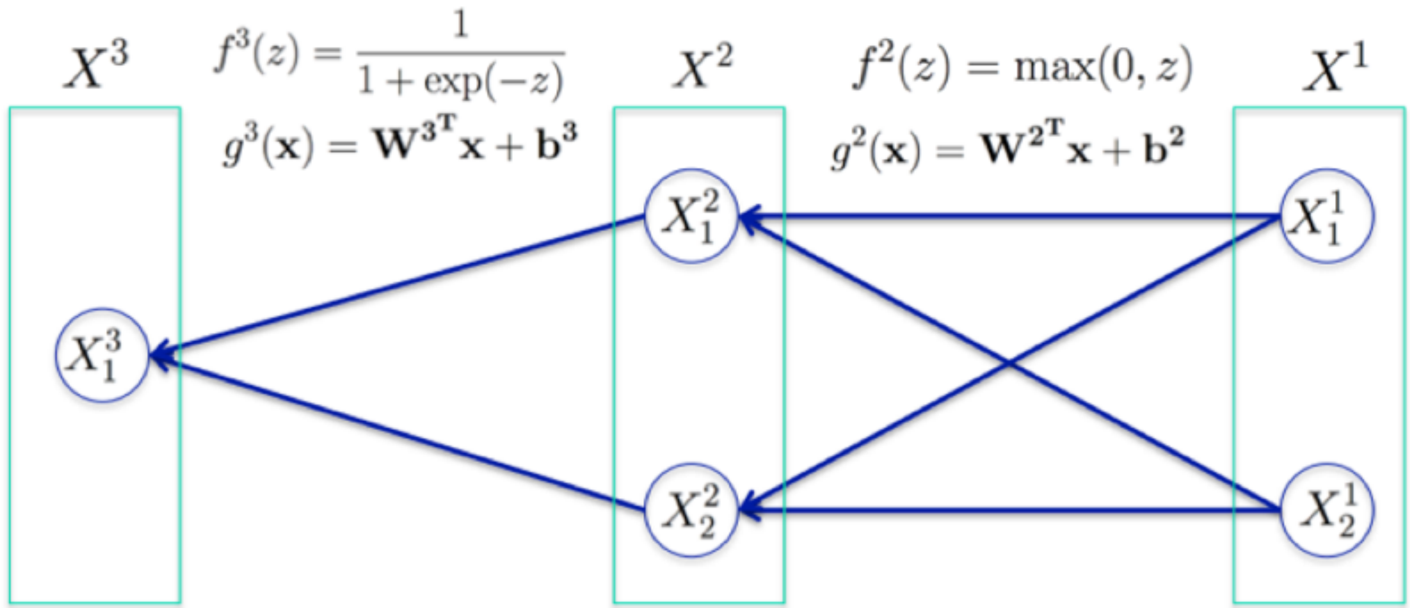
Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00



X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

$$\text{update each parameter according to } \theta_i \leftarrow \quad \theta_i - \eta\frac{\partial J(\theta)}{\partial \theta_i}$$

Sigmoid unit $f^3(g^3(\mathbf{x}))$ ReLU units $f^2(g^2(\mathbf{x}))$



Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00

X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.53	0.12	0.10	-0.09	0.10	0.20	0.20	0.10	-0.10	0.10	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

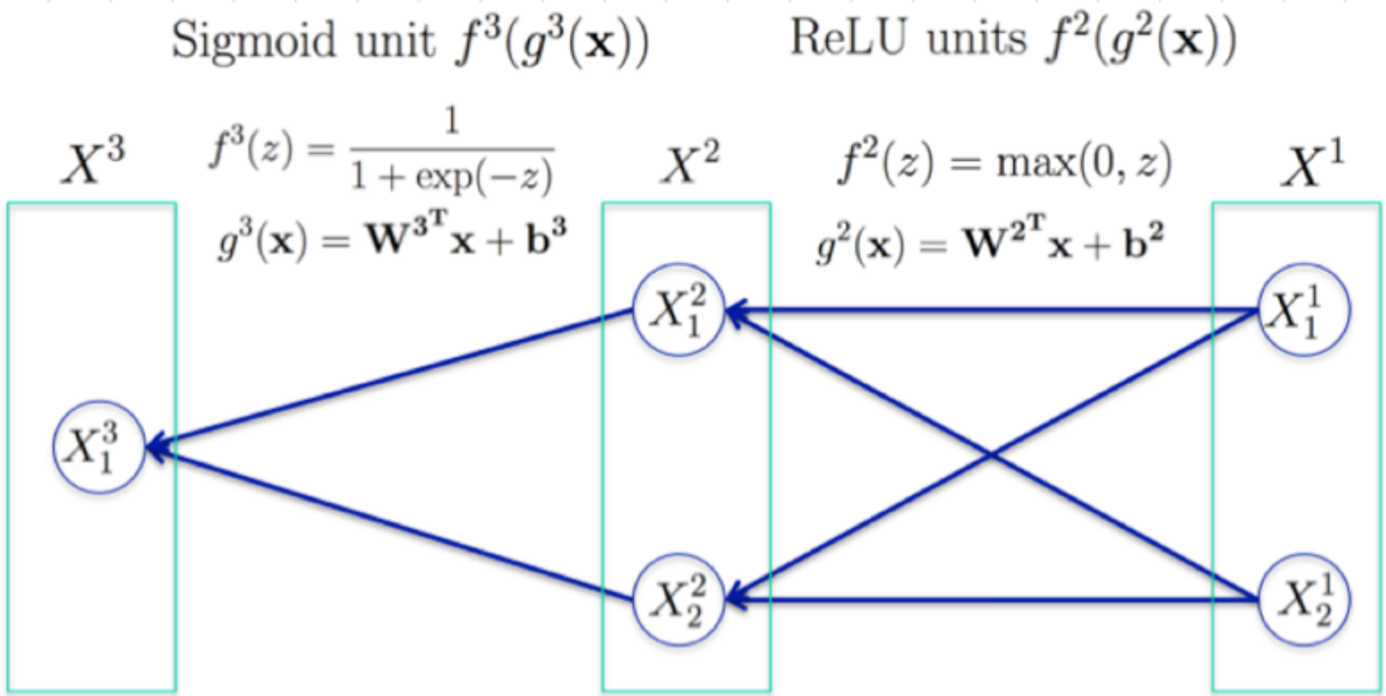
update each parameter according to $\theta_i \leftarrow \theta_i - \eta \frac{\partial J(\theta)}{\partial \theta_i}$

using $\eta = 1$:

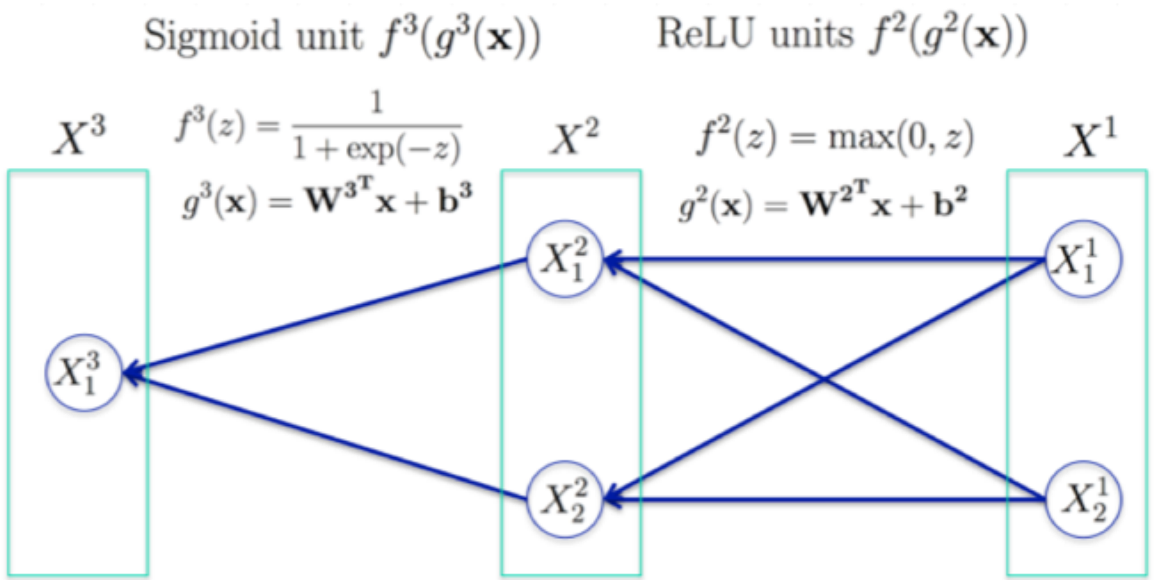
X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.65	0.63	0.19	-0.09	0.57	0.30	0.30	0.15	-0.10	0.15	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*

Next training example.. next stochastic gradient step…

Ytrue	1		
dJ/dX3	-1.53		
dJ/dg3	-0.35		
dJ/dW3T	-0.10	0.00	-0.35
dJ/dX2	-0.07		
	0.03		
dJ/dg2	-0.07		
	0.00		
dJ/dW2T	-0.07	0.00	-0.07
	0.00	0.00	0.00



X^3	$g^3(X^2)$	W^{3T}		b^3	X^2	$g^2(X^1)$	W^{2T}		b^2	X^1
0.65	0.63	0.19	-0.09	0.57	0.30	0.30	0.15	-0.10	0.15	1
					0.00	-0.15	-0.20	0.10	0.05	0
					1					*1*



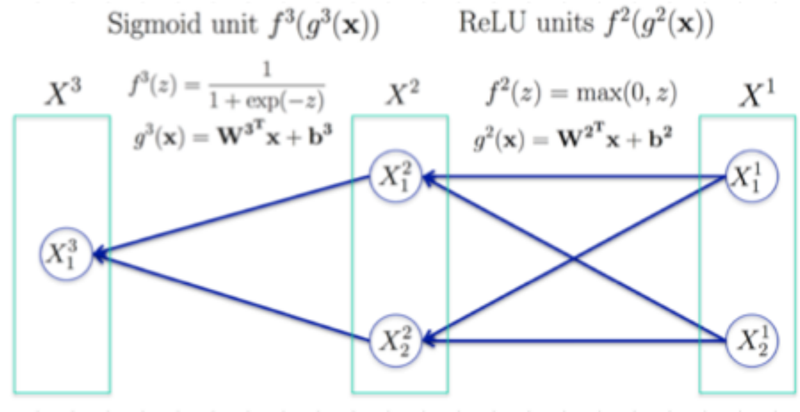
Given boolean Y, X_1, X_2 learn $P(Y|X_1,X_2)$, where

$$P(Y=0|X_1=X_2)=0.9$$
$$P(Y=1|X_1\neq X_2)=0.9$$

Training:

- stochastic gradient descent
- 20,000 iterations of gradient descent
- minibatch size 4
- no momentum, regularization, ...

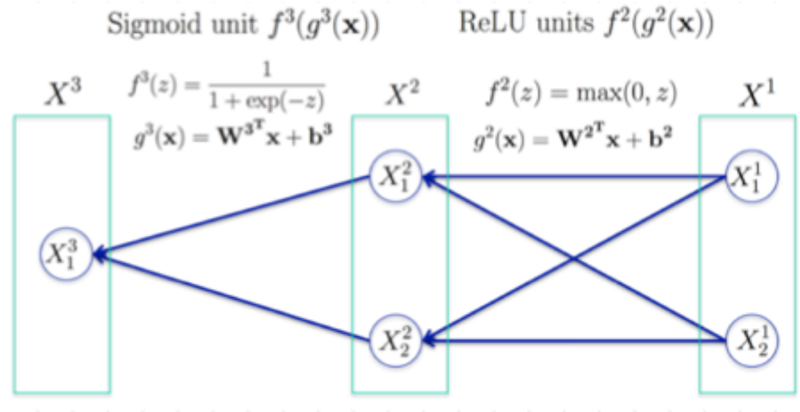
Learned $P(Y = 1 | X_1^1, X_2^1;\ \theta)$



Input: [0 1]	[1 1]	[1 0]	[0 0]
[0.5324]	[0.5343]	[0.5390]	[0.5348]
[0.5139]	[0.5145]	[0.5230]	[0.5149]
[0.5045]	[0.5048]	[0.5203]	[0.5046]
[0.5076]	[0.5073]	[0.5393]	[0.5075]
[0.4988]	[0.4976]	[0.5577]	[0.4987]
[0.4913]	[0.4883]	[0.5971]	[0.4895]
[0.4950]	[0.4877]	[0.6723]	[0.4898]
[0.4655]	[0.4541]	[0.7168]	[0.4553]
[0.4677]	[0.4527]	[0.7778]	[0.4422]
[0.4748]	[0.4093]	[0.8068]	[0.4093]
[0.5119]	[0.3754]	[0.8337]	[0.3752]
[0.5928]	[0.3440]	[0.8603]	[0.3450]
[0.6882]	[0.3108]	[0.8783]	[0.3083]
[0.7903]	[0.2789]	[0.8856]	[0.2789]
[0.8103]	[0.2315]	[0.8794]	[0.2315]
[0.8509]	[0.2062]	[0.8807]	[0.2062]
[0.8634]	[0.1860]	[0.8938]	[0.1860]
[0.8676]	[0.1626]	[0.8915]	[0.1626]
[0.8919]	[0.1511]	[0.8965]	[0.1511]
[0.8821]	[0.1392]	[0.8850]	[0.1392]
[0.8769]	[0.1294]	[0.9053]	[0.1292]

20,000 training iterations

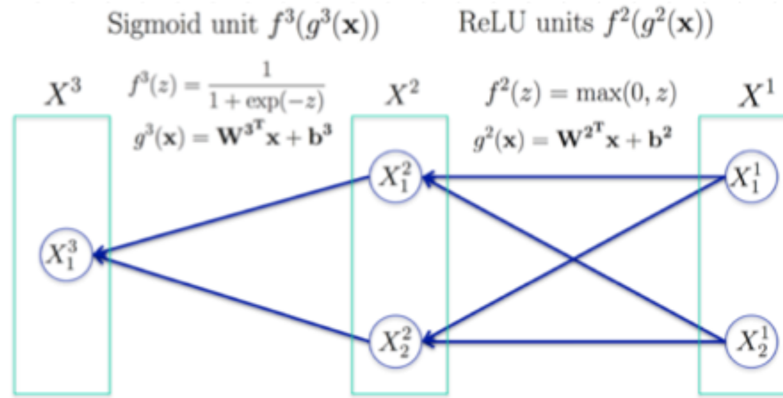
Learned representation for X^2

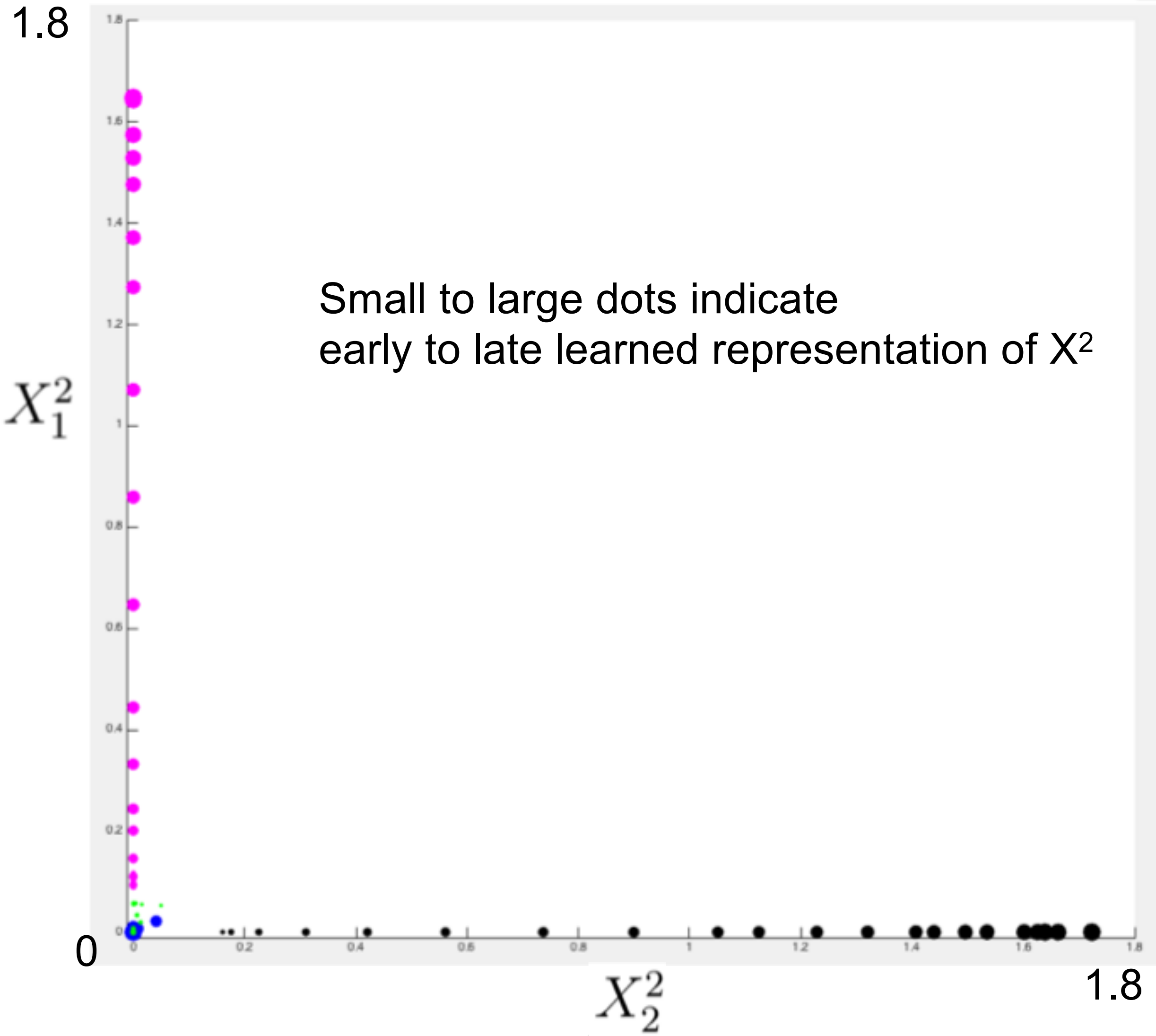


How does the hidden layer representation X^2 evolve over time during gradient descent training?

Learned representation for X^2

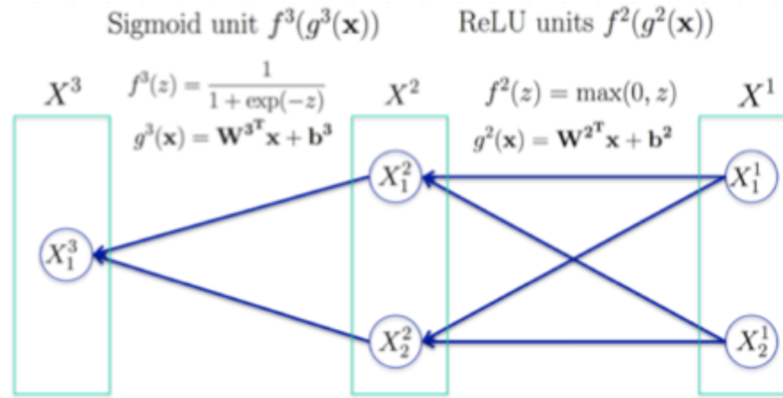
Input: [0 1] [1 1] [1 0] [0 0]

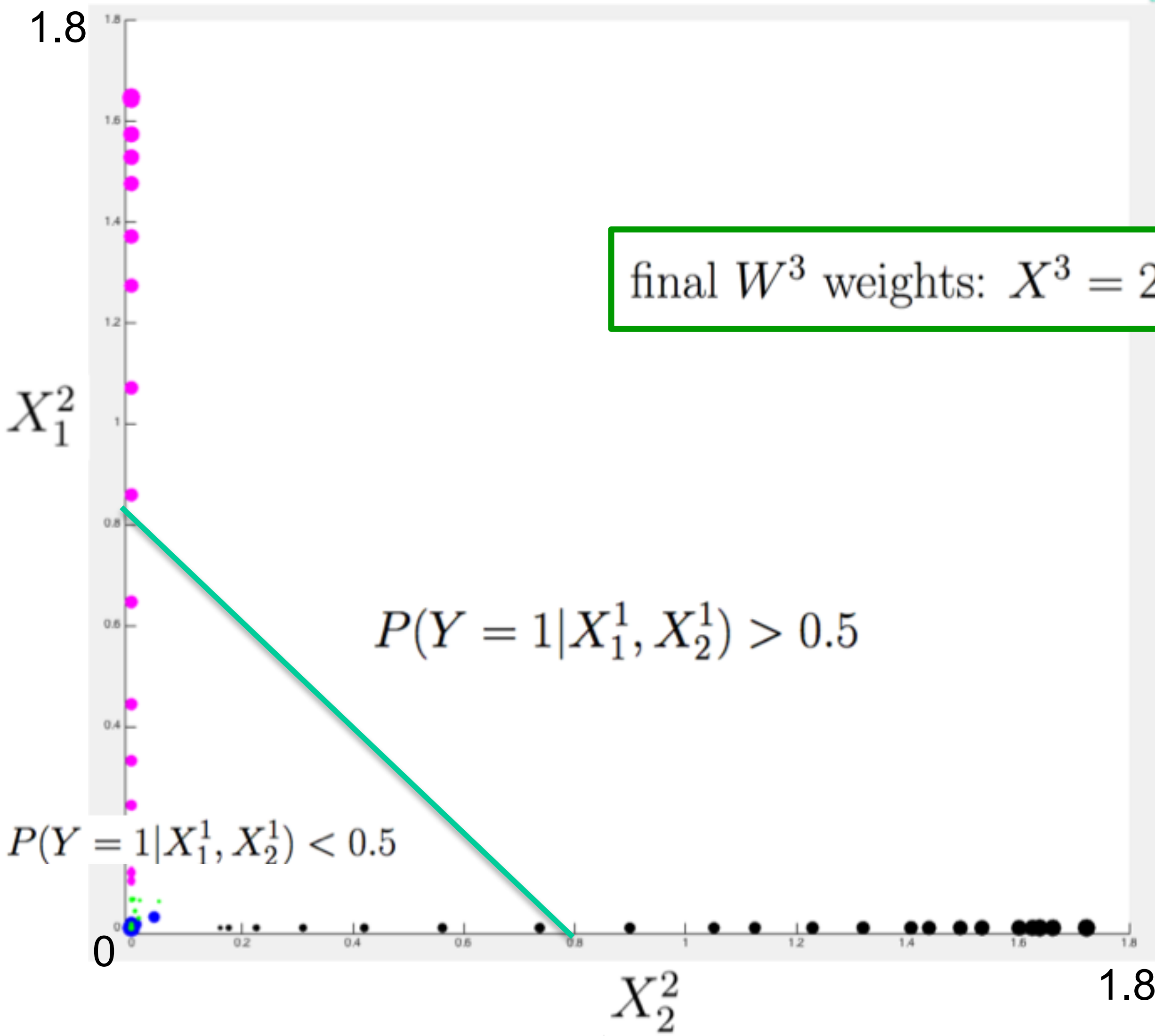




Final decision surface in terms of X^2

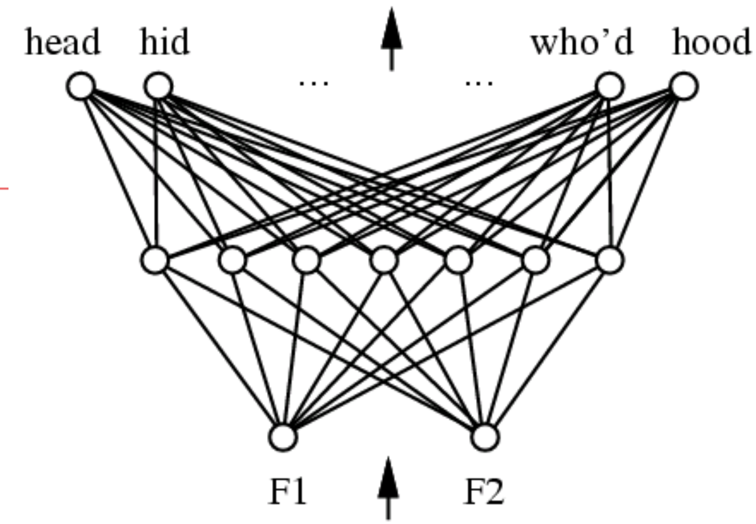
Input X^1: [0 1] [1 1] [1 0] [0 0]





$$\boxed{\text{final } W^3 \text{ weights: } X^3 = 2.417\ X_1^2 + 2.346\ X_2^2 - 1.908}$$

Backpropagation Algorithm
for sigmoid netwk, minimizing $\Sigma_d (t_d - o_d)^2$

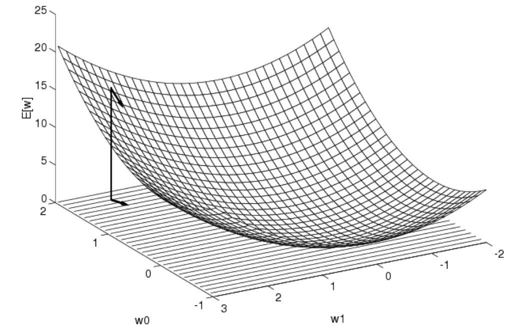


Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k
  $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
  3. For each hidden unit h
  $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$
  4. Update each network weight $w_{i,j}$
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
  where
  $$\Delta w_{i,j} = \eta \delta_j x_i$$

x_d = input
o_i = observed i^{th} unit output
t_i = target output
w_{ij} = wt from i to j
δ_k= error term backpropagated to unit k; that is: $dJ(\theta) / dg^k(X^{k-1})$

Incremental (Stochastic) Gradient Descent



Given set D of training examples

Batch mode gradient descent:
Do until satisfied:
1. Compute the gradient $\nabla_\theta J_D(\theta)$
2. $\theta \leftarrow \theta - \eta \, \nabla_\theta J_D(\theta)$

$$J_D(\theta) \equiv \sum_{d \in D} loss(d, \theta)$$

Incremental mode gradient descent
Do until satisfied:
- For each training example d in D
  1. Compute the gradient $\nabla_\theta J_d(\theta)$
  2. $\theta \leftarrow \theta - \eta \, \nabla_\theta J_d(\theta)$

$$J_d(\theta) \equiv loss(d, \theta)$$

Theory:
Incremental Gradient Descent (aka *stochastic gradient descent*) approximates *Batch Gradient Descent* arbitrarily closely as $\eta \to 0$.

Many modifications to gradient descent

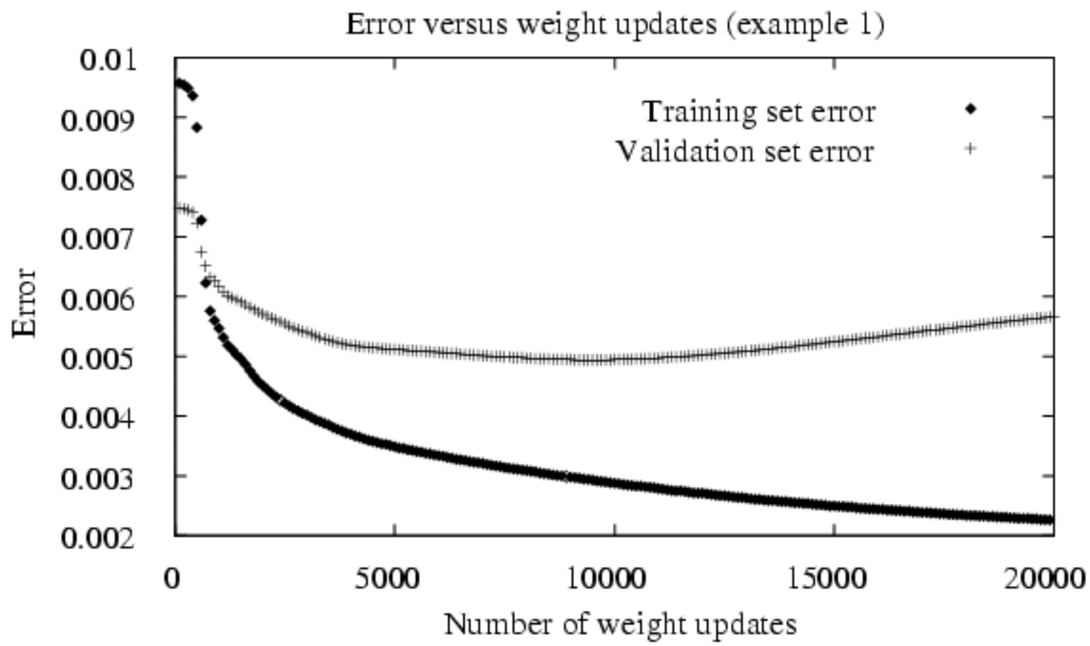
- Stochastic vs. Batch gradient descent (and mini-batches)
- Momentum
- Weight decay (MAP estimate with zero-mean prior)
- Gradient clipping
- Batch normalization
- Dropout
- Adagrad
- Adam
- ….   no end in sight

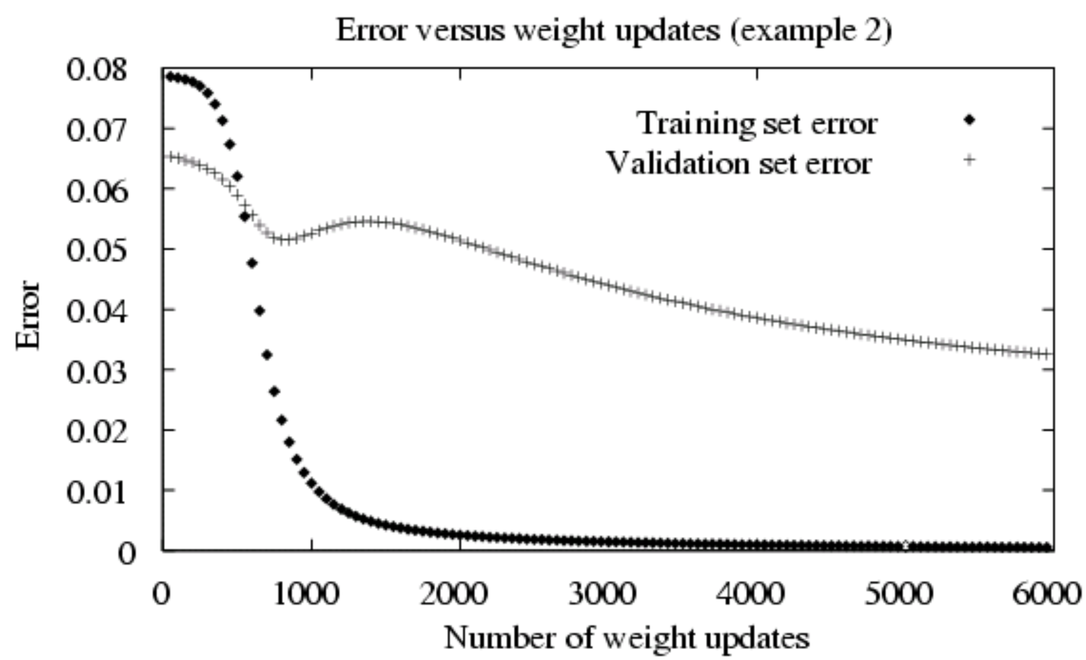
See ML Department course on Optimization Methods

Gradient Descent and Backpropagation

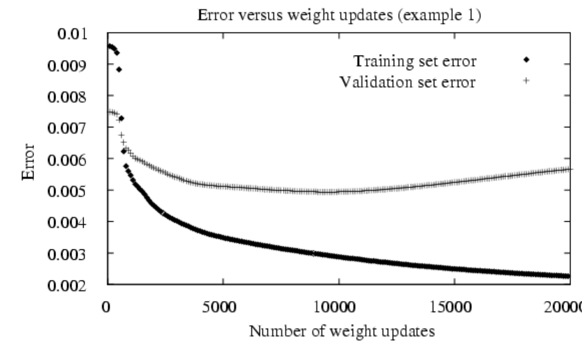
- Updates every network parameter simultaneously, each iteration
- Because we repeatedly use the chain rule to calculate gradients, easily generalized to arbitrary directed acyclic graphs
- Finds local minimum in $J(\theta)$, not necessarily global min
- Minimizes $J(\theta)$ over training examples, not necessarily future…
- Training can require hours, days, weeks, GPUs
- Applying network after training is relatively very fast

Overfitting in Neural Nets





Dealing with Overfitting



Deep net training involves a hyperparameter

n=number of gradient descent iterations

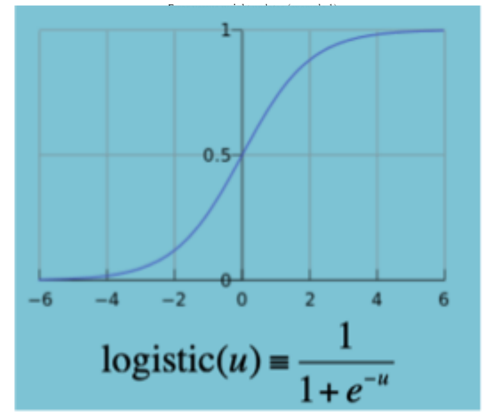
How do we choose n to optimize future accuracy?

- Separate available data into training and validation set
- Use training to perform gradient descent
- test validation error frequently, save network at each step
- n ← number of iterations that optimizes validation set error

→ *gives unbiased estimate of optimal n*

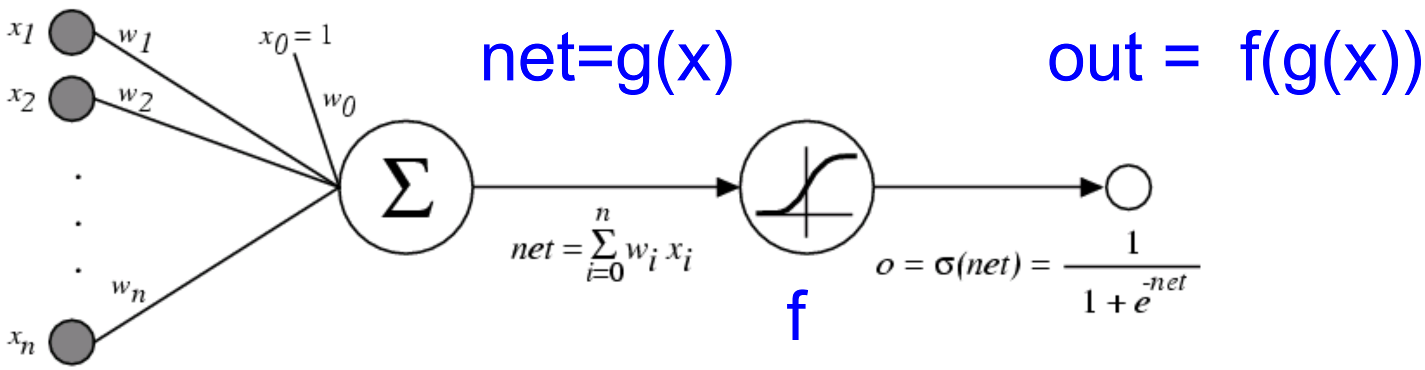
(but still an optimistically biased estimate of true error)

Initializing neural net weights



Usually initialize weights to random values close to zero

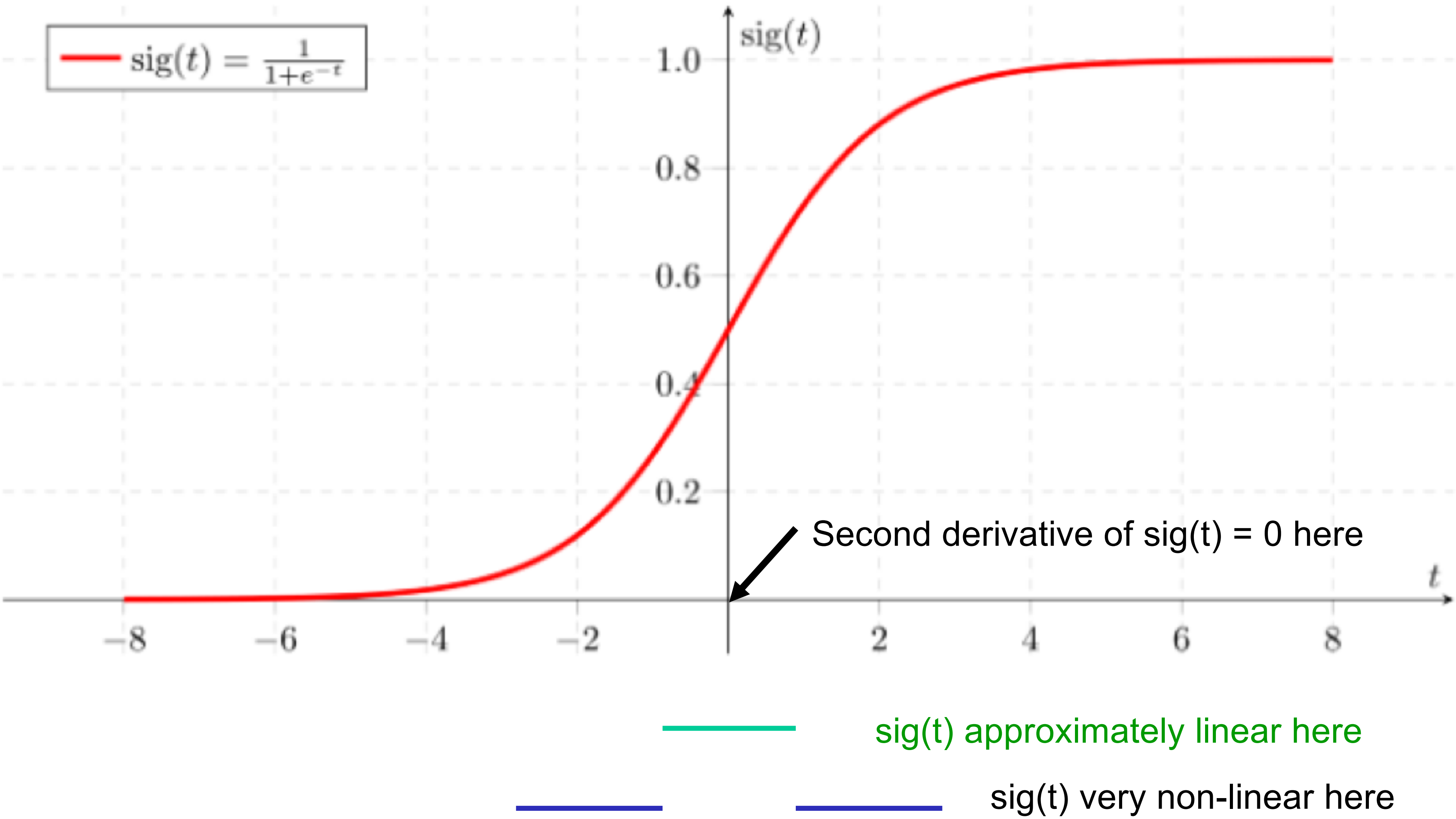
Why? Consider **sigmoid unit**…



Small w's → small net=g(x) → output f(g(x)) nearly linear function of inputs x

As we take more gradient steps, |w| grows → increasingly non-linear

Begin with small weights → Begin with ~linear function



Expressive Capabilities of ANNs

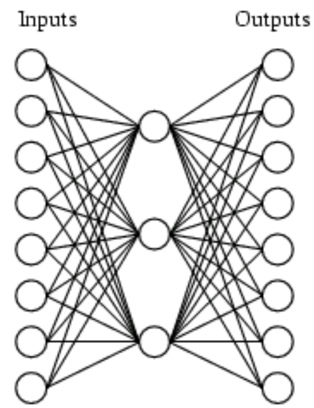
Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Learning Hidden Layer Representations



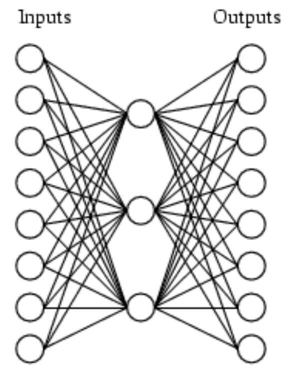
A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Learning Hidden Layer Representations

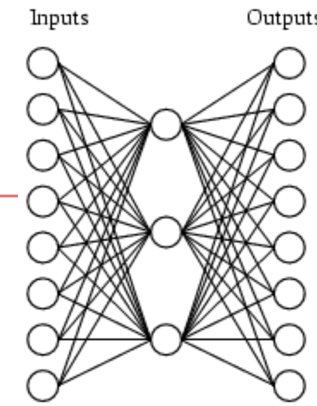
A network:

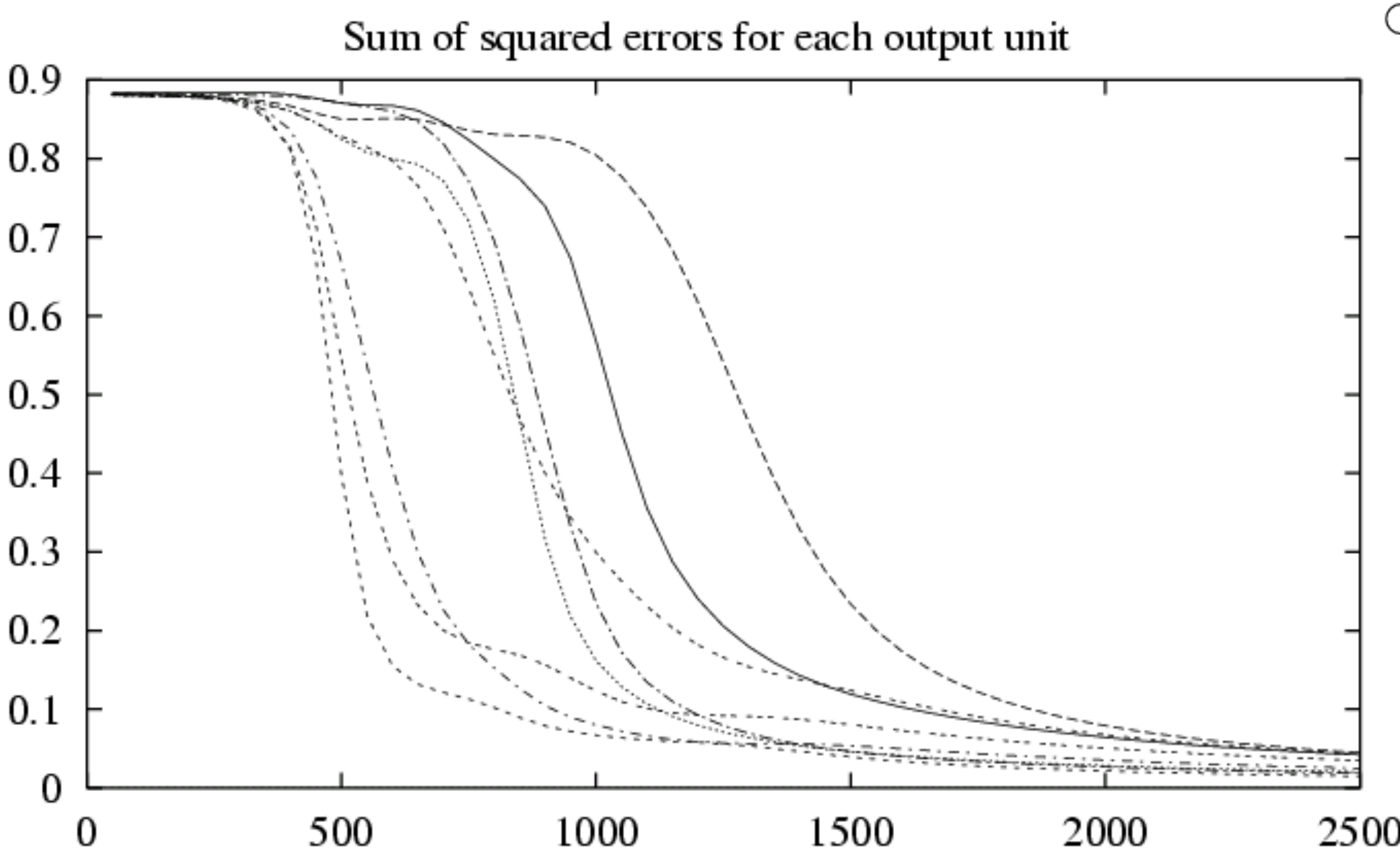


Learned hidden layer representation:

Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

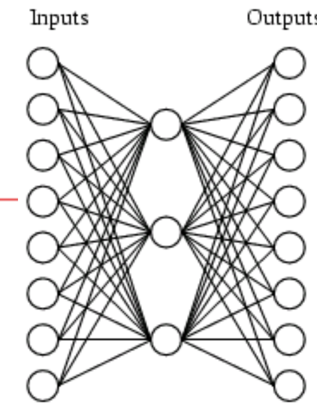
Training

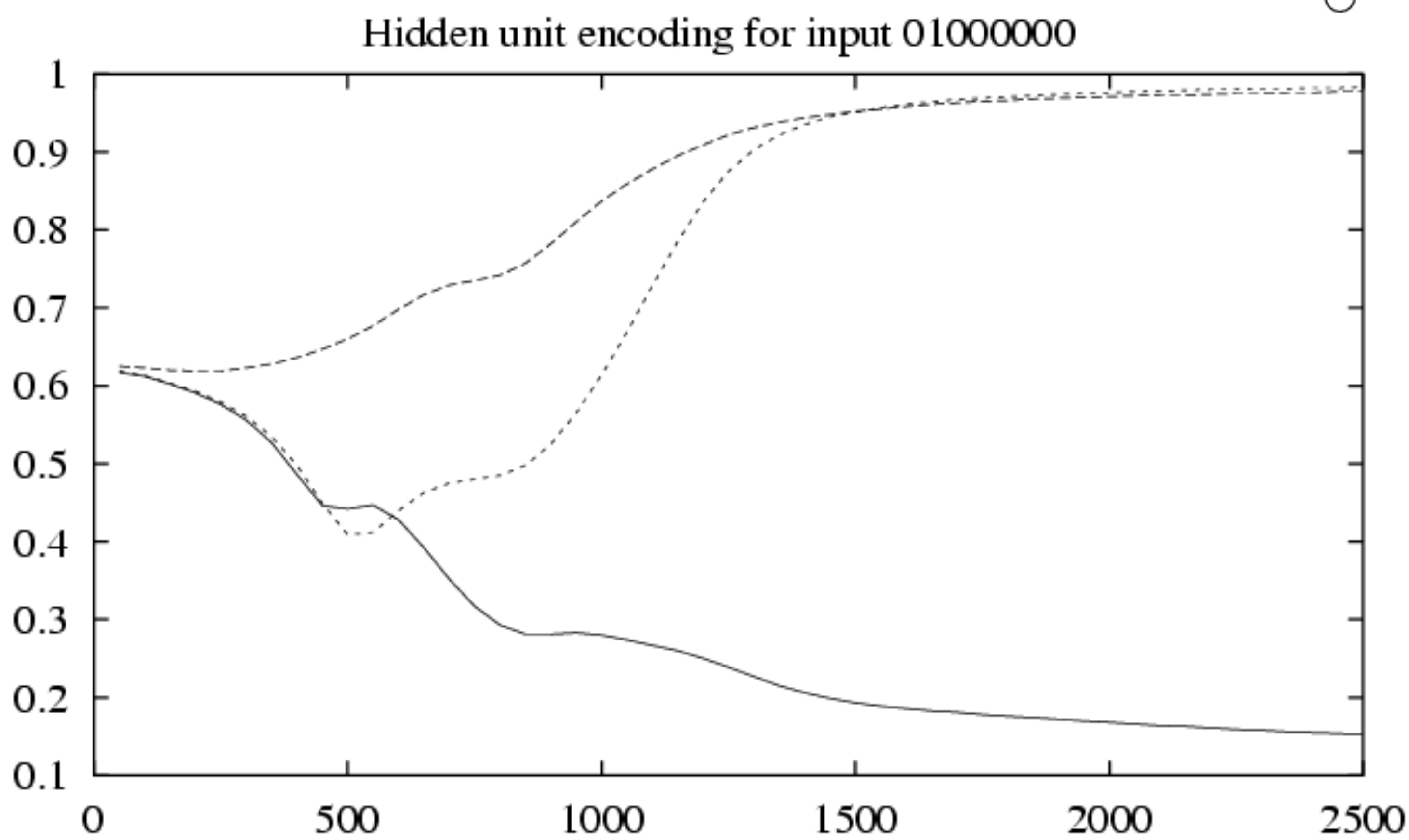




Gradient descent steps →

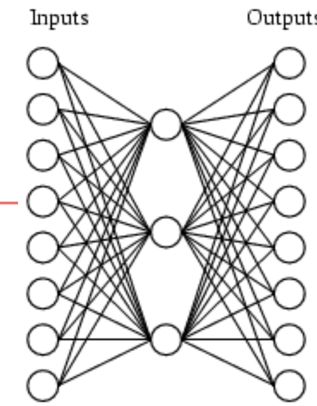
Training

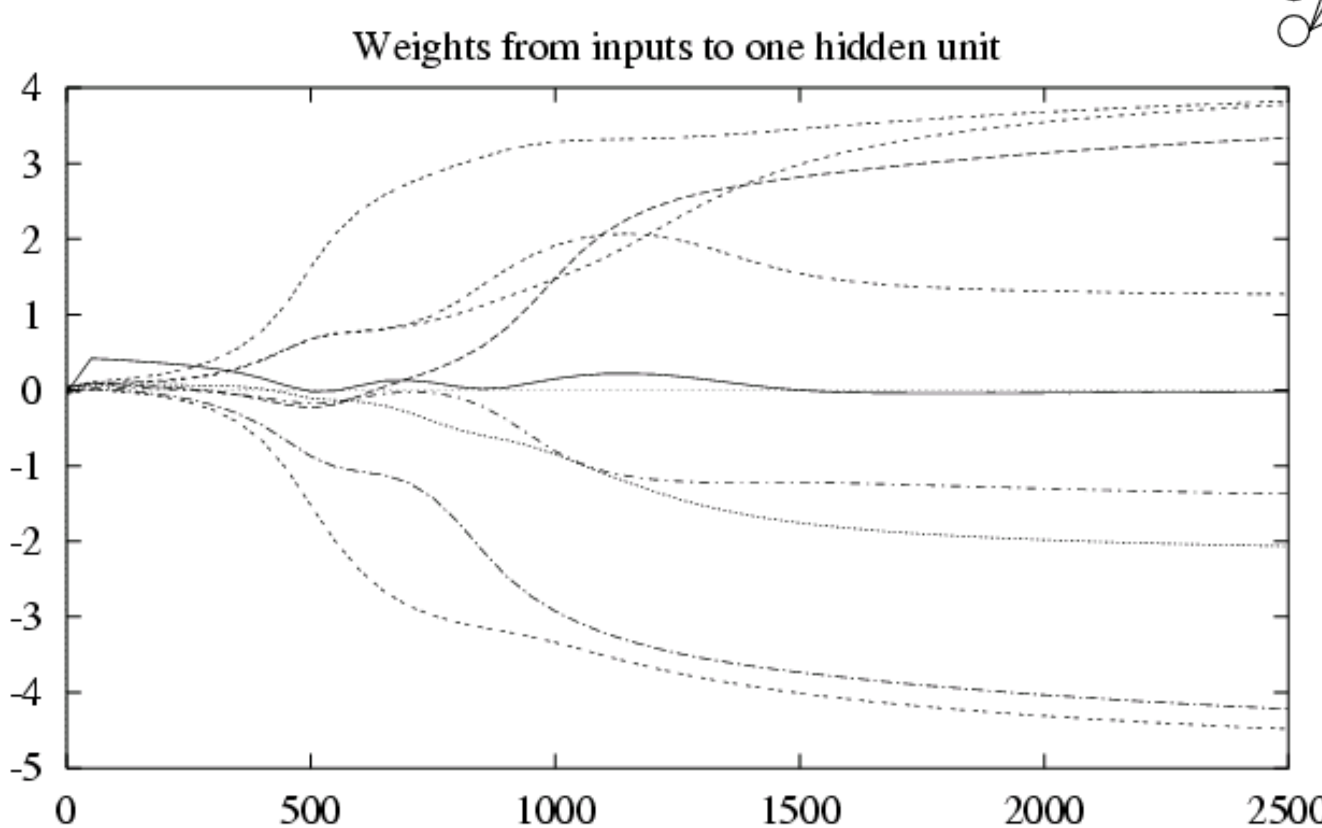




Gradient descent steps →

Training





Gradient descent steps →

What you should know: Artificial Neural Networks

- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially billions of parameters to estimate
- Hidden layers learn intermediate representations

- Directed acyclic graph, trained by gradient descent
- Chain rule over this DAG allows computing all derivatives
- Can use any differentiable loss function
  – we used neg. log likelihood in order to learn outputs P(Y|X)

- Gradient descent, local minima problems
- Overfitting and how to deal with it