Neural Networks

Manuela Veloso

Co-instructor: Pradeep Ravikumar

Machine Learning 10-701

Slides Courtesy: Tom Mitchell

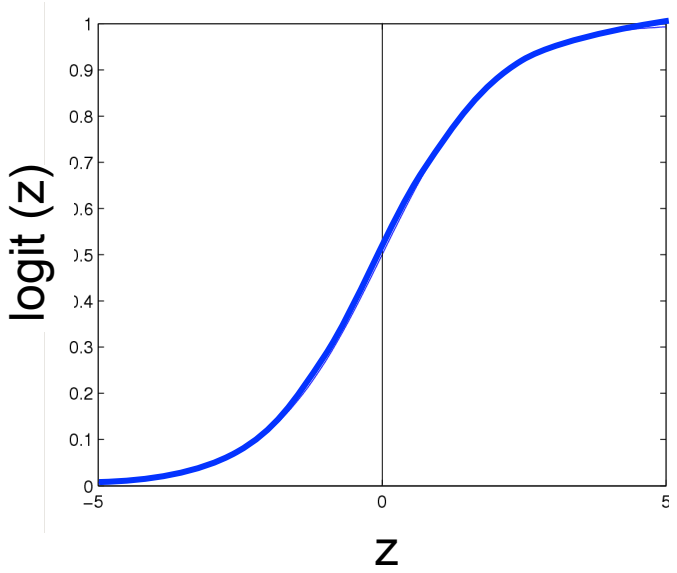
Logistic Regression

Assumes the following functional form for P(Y|X):

$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0+\sum_i w_i X_i))}$$

Logistic function applied to a linear function of the data

Logistic function (or Sigmoid): $\frac{1}{1+exp(-z)}$



Logistic Regression is a Linear Classifier!

Assumes the following functional form for P(Y|X):

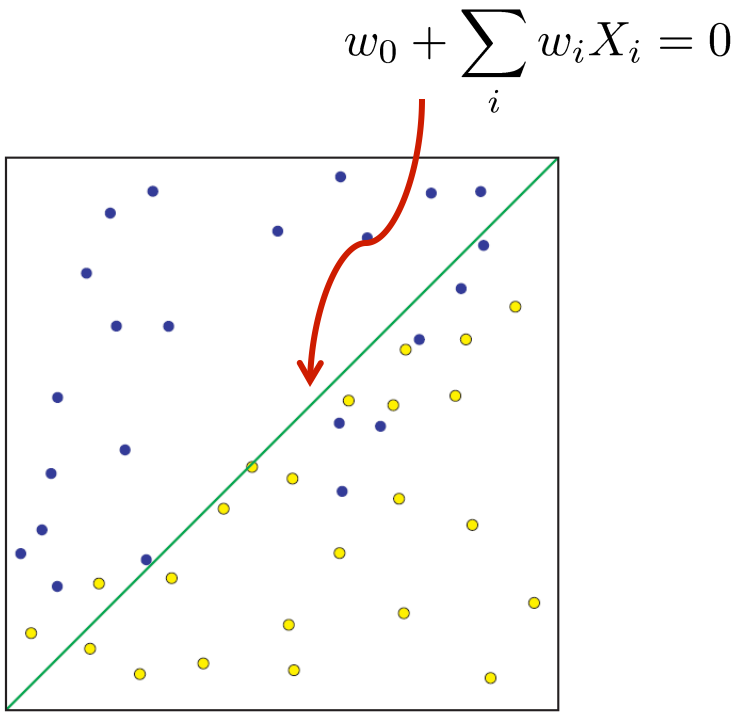
$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0+\sum_i w_i X_i))}$$

Decision boundary:

$$P(Y=0|X) \underset{1}{\overset{0}{\gtrless}} P(Y=1|X)$$

$$0 \underset{1}{\overset{0}{\gtrless}} w_0 + \sum_i w_i X_i$$

(Linear Decision Boundary)



Training Logistic Regression

How to learn the parameters w_0, w_1, … w_d?

Training Data $\{(X^{(j)}, Y^{(j)})\}_{j=1}^{n}$ $X^{(j)} = (X_1^{(j)}, \ldots, X_d^{(j)})$

Maximum (Conditional) Likelihood Estimates

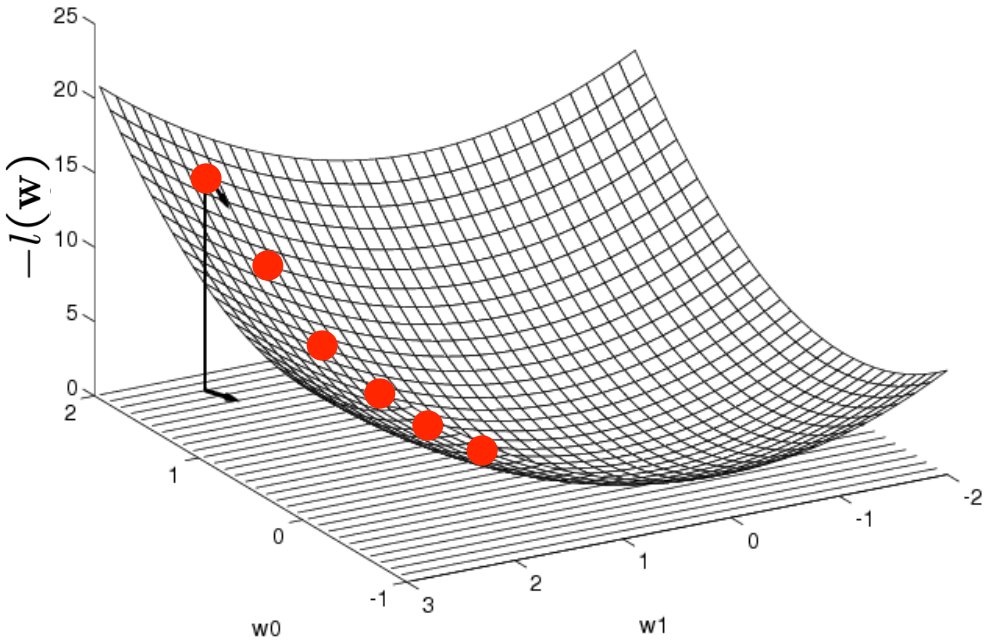
$$\hat{\mathbf{w}}_{MCLE} = \arg\max_{\mathbf{w}} \prod_{j=1}^{n} P(Y^{(j)} \mid X^{(j)}, \mathbf{w})$$

Discriminative philosophy – Don't waste effort learning P(X), focus on P(Y|X) – that's all that matters for classification!

Optimizing convex function

- Max Conditional log-likelihood = Min Negative Conditional log-likelihood
- Negative Conditional log-likelihood is a convex function

Gradient Descent (convex)



Gradient:

$$\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_d}]'$$

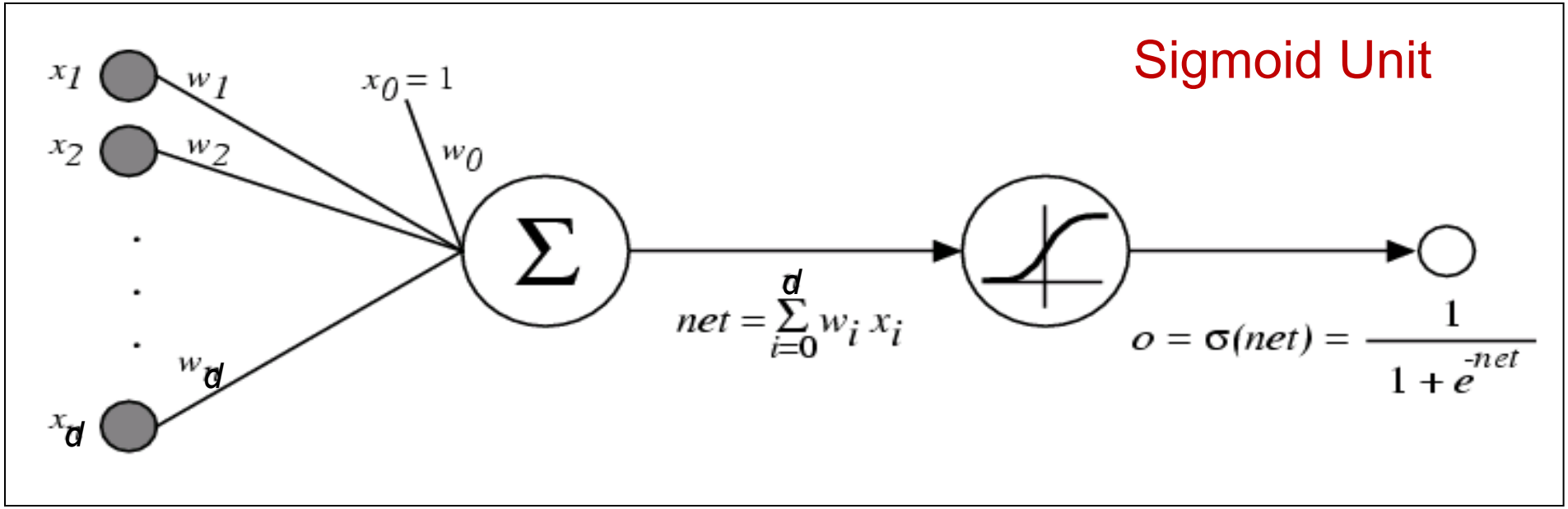
Update rule: **Learning rate, η>0**

$$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left. \frac{\partial l(\mathbf{w})}{\partial w_i} \right|_t$$

Logistic function as a Graph
Single-Layer Perceptron

$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

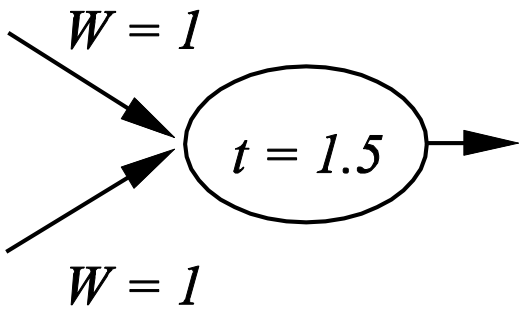


Single Layer: Remarks

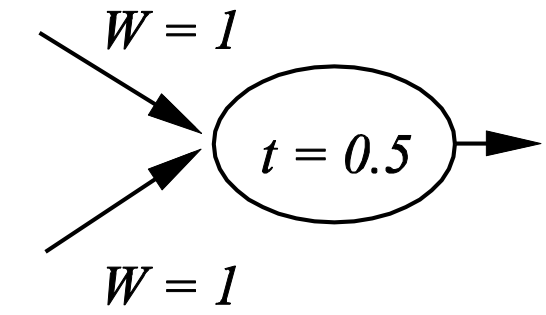
- Good news: Can represent any problem in which the decision boundary is *linear*.

- Boolean functions?
  - And
  - Or
  - Not

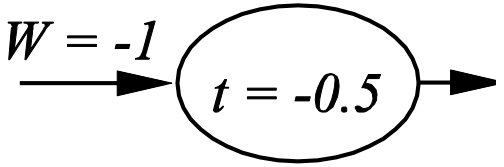
Boolean Functions (-t as bias)



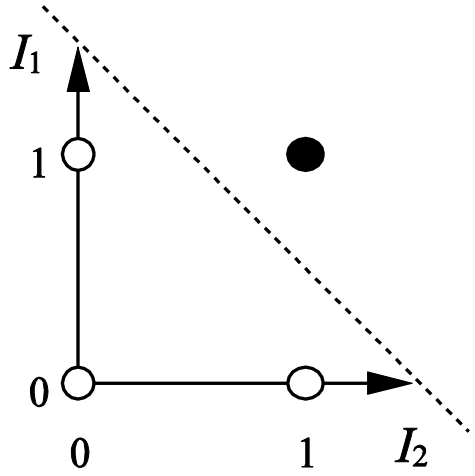
AND



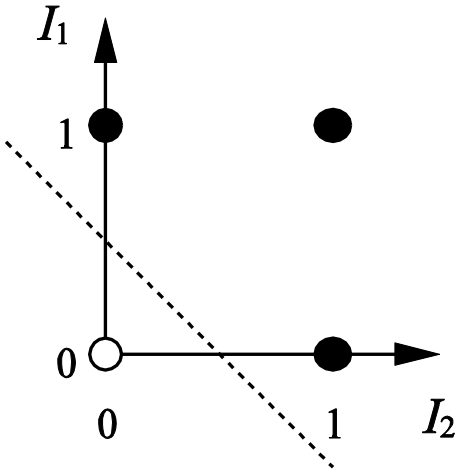
OR



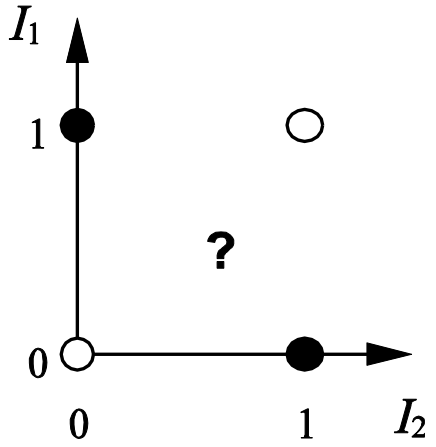
NOT



(a) I_1 and I_2

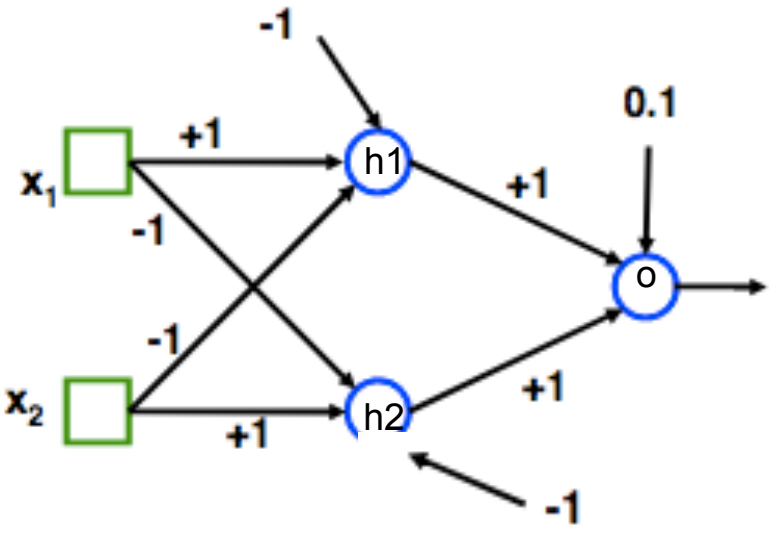


(b) I_1 or I_2



(c) I_1 xor I_2

XOR
Input is -1, 1 (instead of 0, 1)

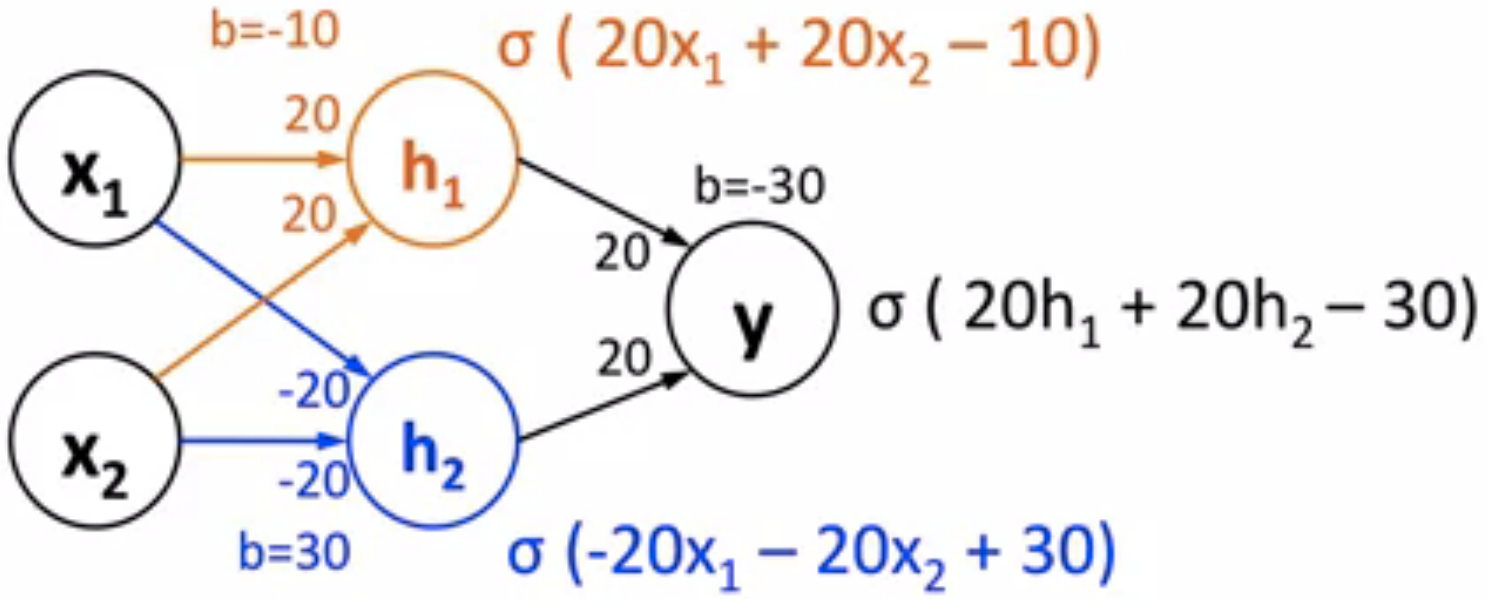


Sign activation function:
-1 if input < 0, 1, if input >=0

xor (-1,-1) and xor (1,1) = -1
xor (-1, 1) and xor (1,-1) = 1

x1	x2	h1 in	h2 out	h2 in	h2 out	o input	o out
-1	-1	(-1)1+(-1)(-1)+(-1) = -1	-1	(-1)1+(-1)(-1)+(-1) = -1	-1	(-1)1 + (-1)1 + 0.1 = -1.9	-1
-1 1	1 -1	(-1)1+1(-1) + (-1) = -3 (-1)(-1)+1*1 + (-1) = 1	-1 1	(-1)(-1)+1*1 + (-1) = 1 (-1)1+1(-1) + (-1) = -3	1 -1	(-1).1+1*1 + 0.1 = 0.1	1
1	1	1*1 +(-1).1 + (-1) = -1	-1	1*1 +(-1)1 + (-1) = -1	-1	(-1)1 + (-1).1 + 0.1 = -1.9	-1

XOR with 0, 1 inputs and outputs and sigmoid activation



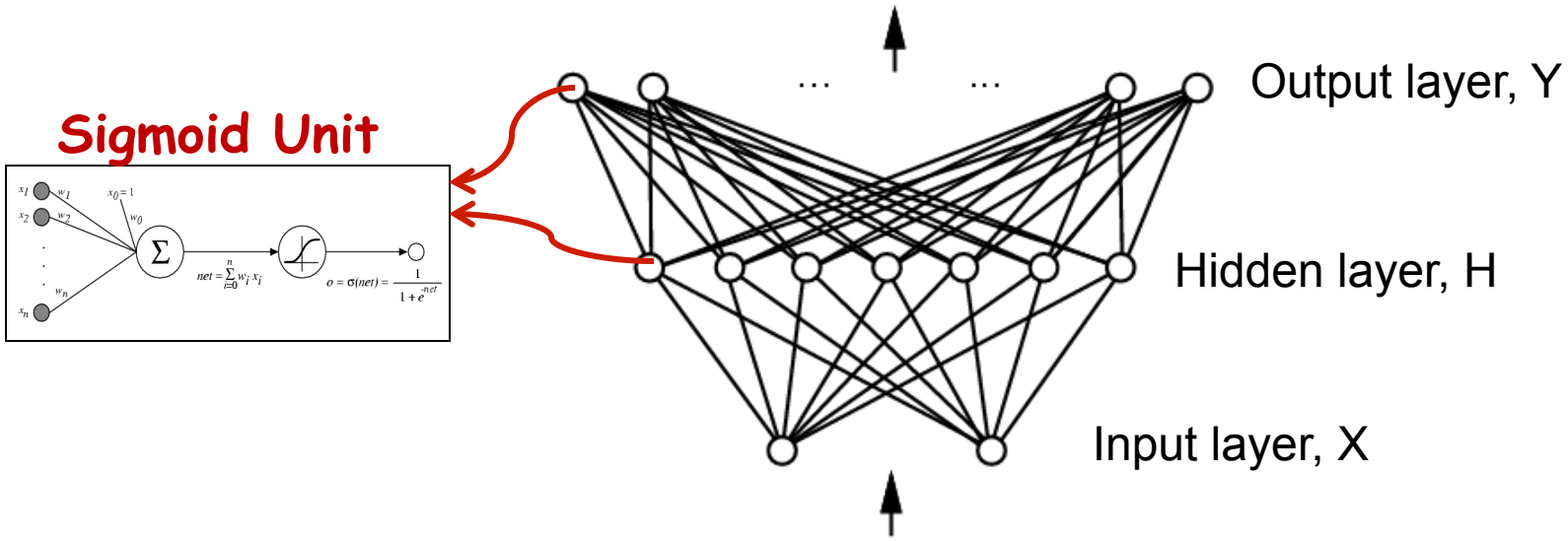
σ(20*0 + 20*0 – 10) ≈ 0 σ (-20*0 – 20*0 + 30) ≈ 1 σ (20*0 + 20*1 – 30) ≈ 0

σ(20*0 + 20*1 – 10) ≈ 1 σ(-20*0 – 20*1 + 30) ≈ 1 σ(20*1 + 20*1 – 30) ≈ 1

Neural Networks to learn f: X → Y

- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

- Neural networks - Represent f by *network* of logistic/sigmoid units:



Two-Layered Networks - Output

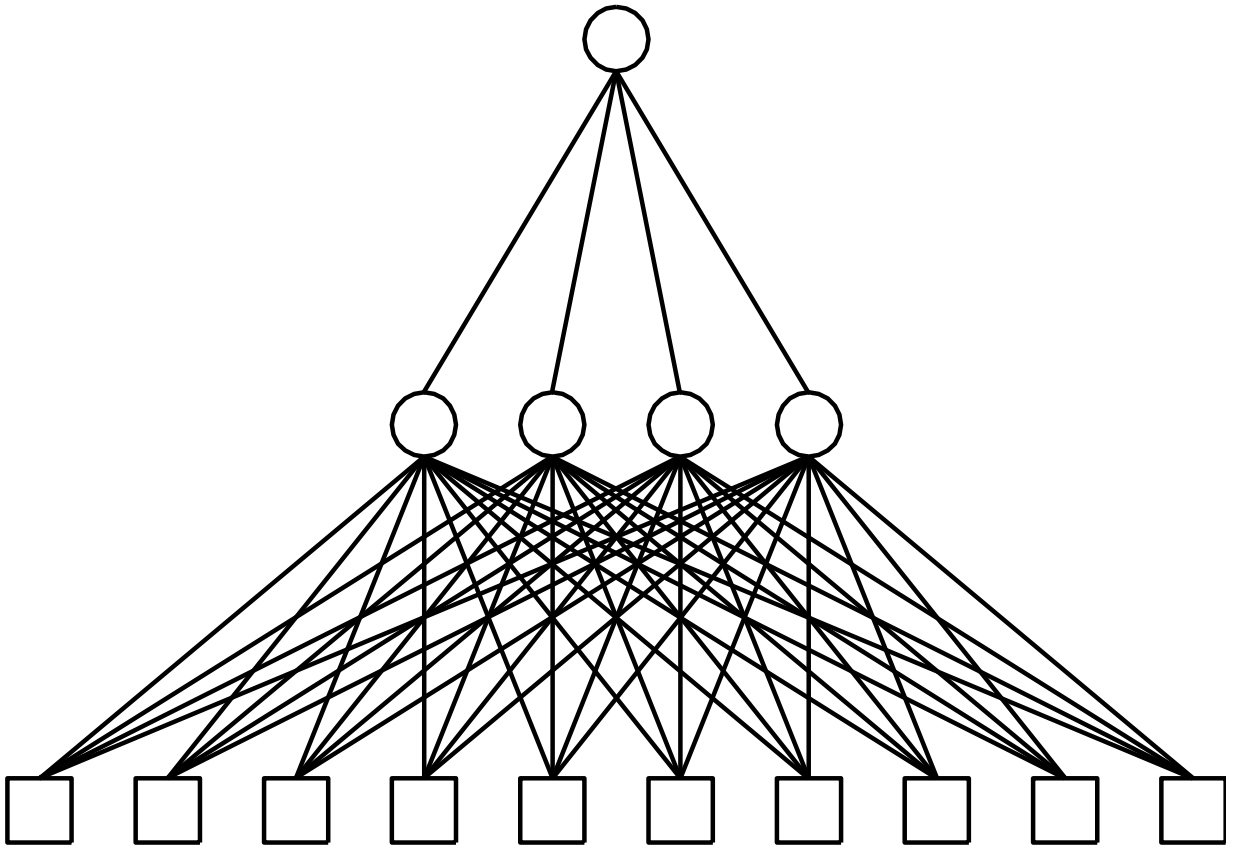
Output units O_i

$W_{j,i}$

Hidden units a_j

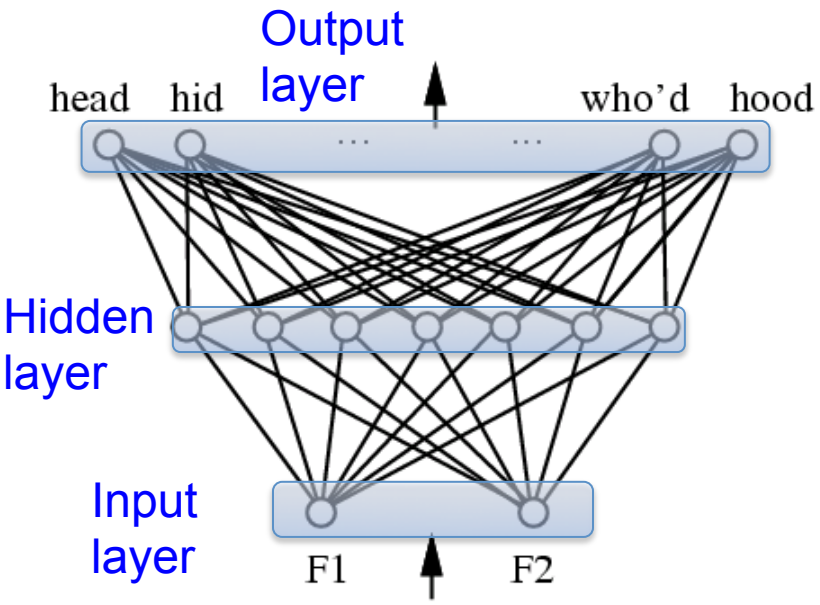
$W_{k,j}$

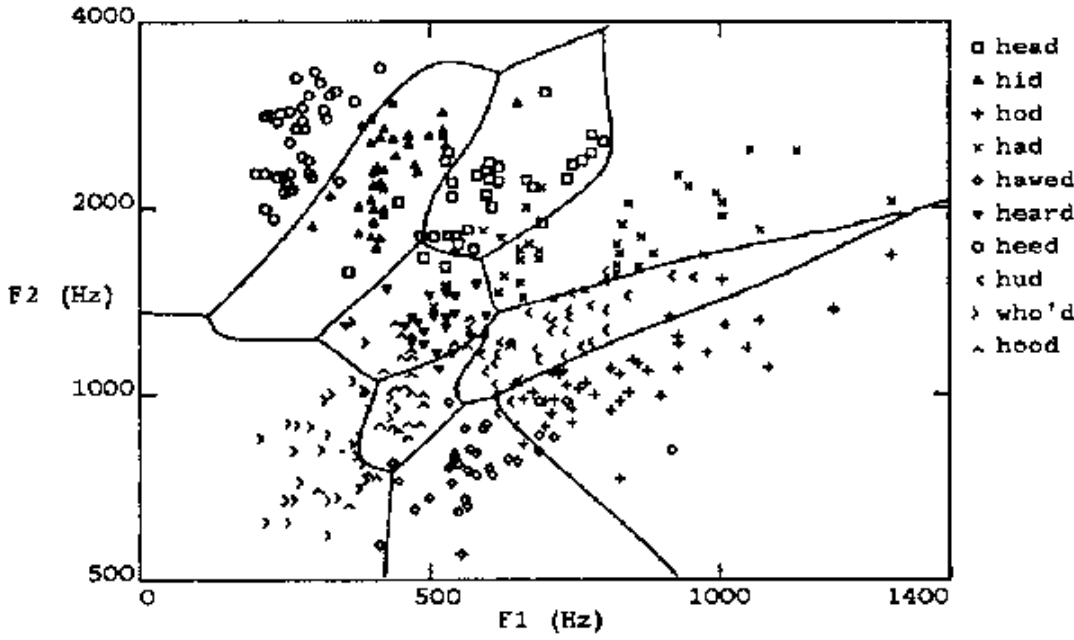
Input units I_k



Multilayer Networks of Sigmoid Units

Neural Network trained to distinguish vowel sounds using 2 formants (features)

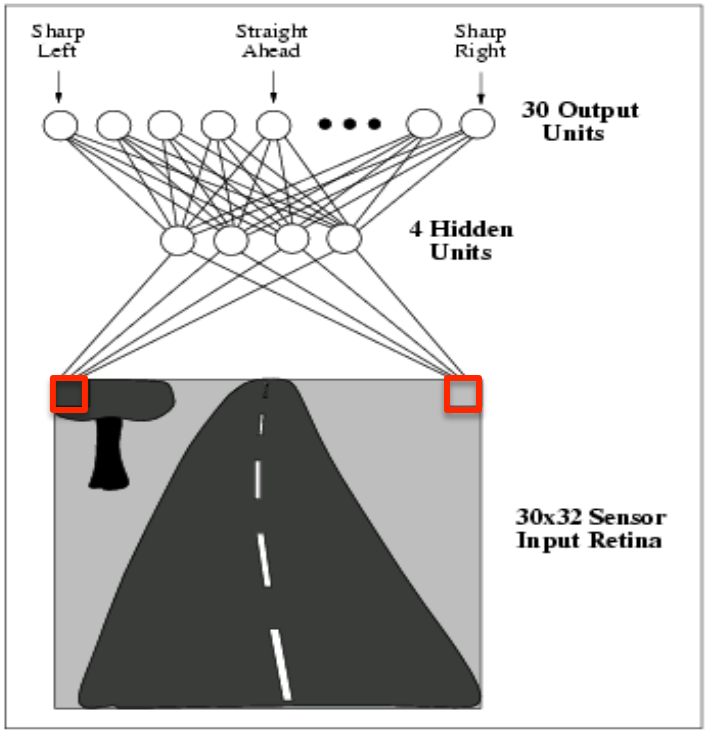




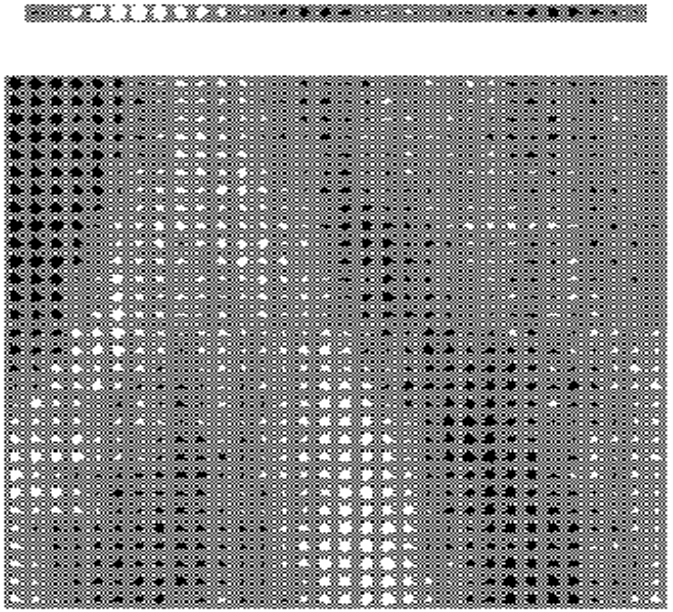
Two layers of logistic units

Highly non-linear decision surface

Neural Network trained to drive a car!

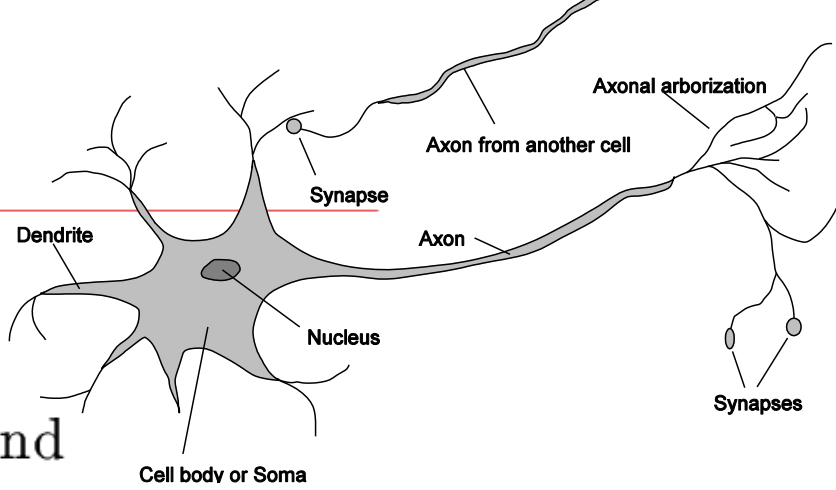


Weights to output units from one hidden unit



Weights of each pixel for one hidden unit

Connectionist Models



Consider humans:

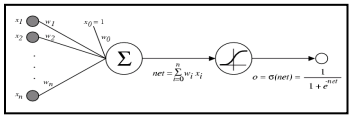
- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

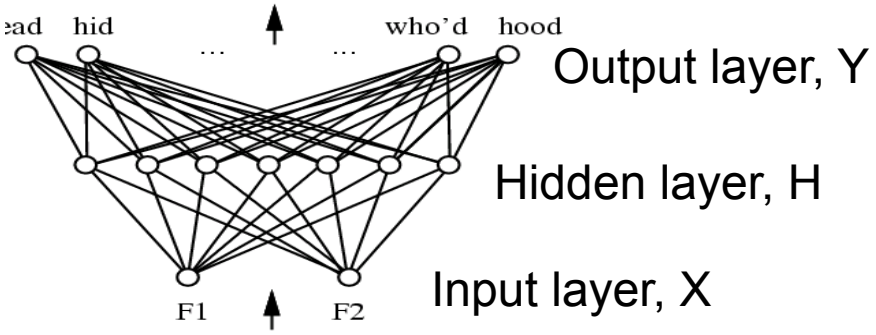
$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

Prediction using Neural Networks





Prediction – Given neural network (hidden units and weights), use it to predict the label of a test point

Forward Propagation – Start from input layer; For each subsequent layer, compute output of sigmoid unit

Sigmoid unit:

$$o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i x_i)$$

1-Hidden layer, 1 output NN:

$$o(\mathbf{x}) = \sigma\left(w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{o_h}\right)$$

M(C)LE Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon$ ← assume noise $N(0,\sigma_\varepsilon)$, iid

deterministic

- Let's maximize the conditional data likelihood

$$W \leftarrow \arg\max_W \ln \prod_l P(Y^l | X^l, W)$$

$$W \leftarrow \arg\min_W \sum_l (y^l - \hat{f}(x^l))^2$$

Learned neural network

Train weights of all units to minimize sum of squared errors of predicted network outputs

MAP Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon$ ← noise $N(0,\sigma_\varepsilon)$

deterministic

Gaussian $P(W) = N(0,\sigma I)$

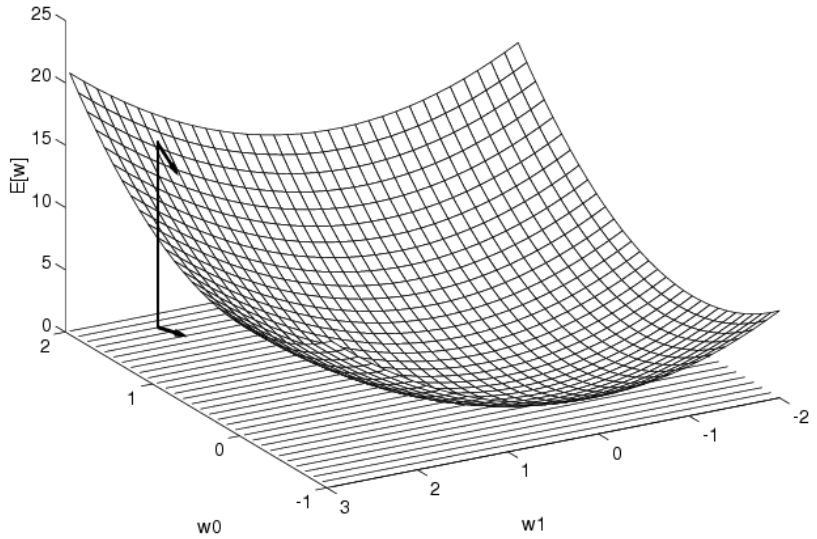
$$W \leftarrow \arg\max_W \ln P(W) \prod_l P(Y^l|X^l, W)$$

$$W \leftarrow \arg\min_W \left[c\sum_i w_i^2\right] + \left[\sum_l (y^l - \hat{f}(x^l))^2\right]$$

$\ln P(W) \leftrightarrow c \sum_i w_i^2$

Train weights of all units to minimize sum of squared errors of predicted network outputs plus weight magnitudes

Gradient Descent



E – Mean Square Error

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_d}\right]$$

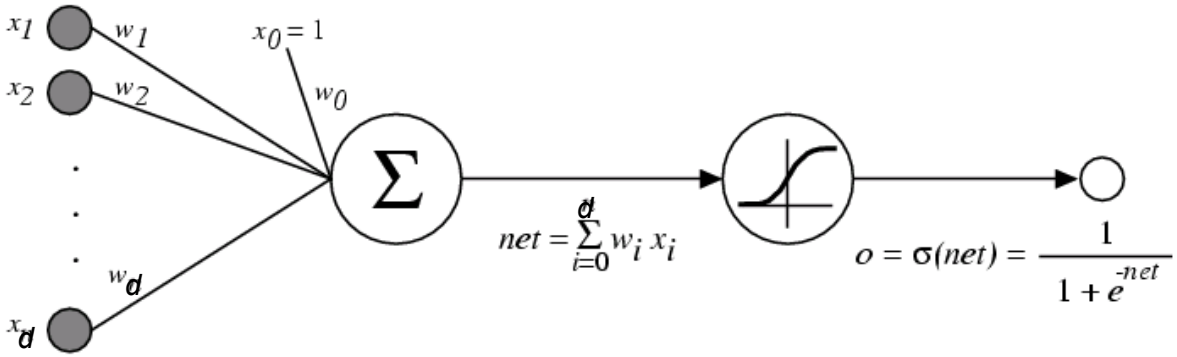
Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Training Neural Networks



$\sigma(x)$ is the sigmoid function

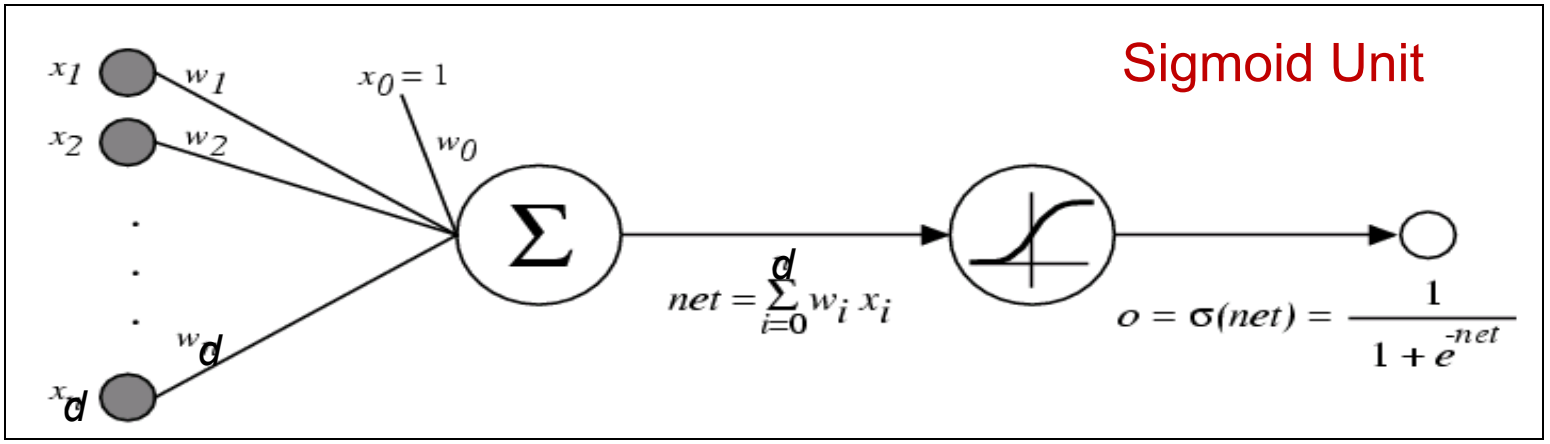
$$\frac{1}{1 + e^{-x}}$$

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

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units → Backpropagation

Error Gradient for a Sigmoid Unit



$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2$$

$$= \frac{1}{2} \sum_l \frac{\partial}{\partial w_i} (y^l - o^l)^2$$

$$= \frac{1}{2} \sum_l 2(y^l - o^l) \frac{\partial}{\partial w_i} (y^l - o^l)$$

$$= \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i} \right)$$

$$= -\sum_l (y^l - o^l) \frac{\partial o^l}{\partial net^l} \frac{\partial net^l}{\partial w_i}$$

But we know:

$$\frac{\partial o^l}{\partial net^l} = \frac{\partial \sigma(net^l)}{\partial net^l} = o^l (1 - o^l)$$

$$\frac{\partial net^l}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}^l)}{\partial w_i} = x_i^l$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{l \in D} (y^l - o^l) o^l (1 - o^l) x_i^l$$

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$

Using all training data *D*

2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2$$

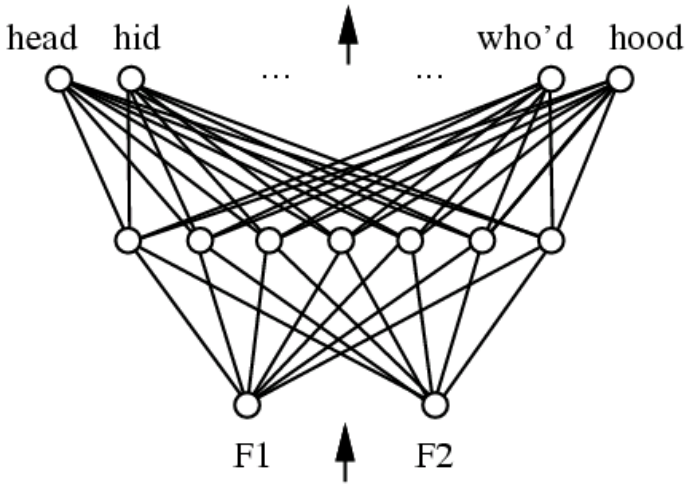
Incremental mode Gradient Descent:
Do until satisfied

- For each training example l in D
  1. Compute the gradient $\nabla E_l[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_l[\vec{w}]$

$$E_l[\vec{w}] \equiv \frac{1}{2}(y^l - o^l)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Backpropagation Algorithm



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 → Using Forward propagation
  2. For each output unit k

$$\delta_k^l \leftarrow o_k^l(1 - o_k^l)(y_k^l - o_k^l)$$

  3. For each hidden unit h

$$\delta_h^l \leftarrow o_h^l(1 - o_h^l) \sum_{k \in outputs} w_{h,k}\delta_k^l$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}^l$$

where

$$\Delta w_{i,j}^l = \eta \delta_j^l o_i^l$$

l = training example

y_k = target output (label) of output unit k

$o_{k(h)}$ = unit output (obtained by forward propagation)

w_{ij} = wt from i to j

Note: if i is input variable, $o_i = x_i$

More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

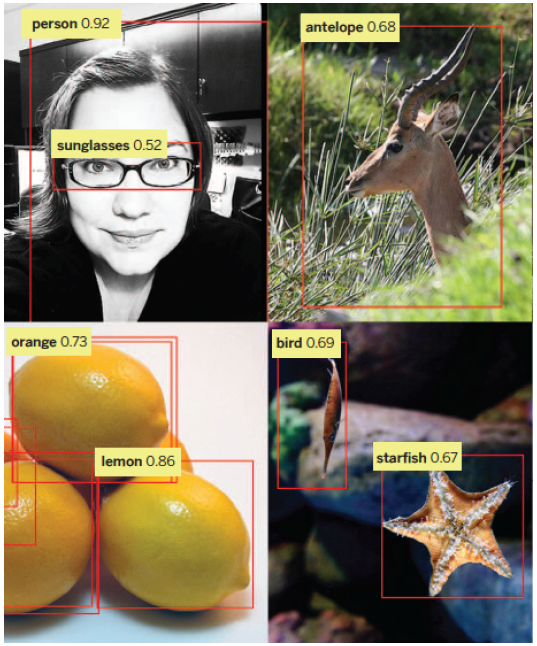
$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

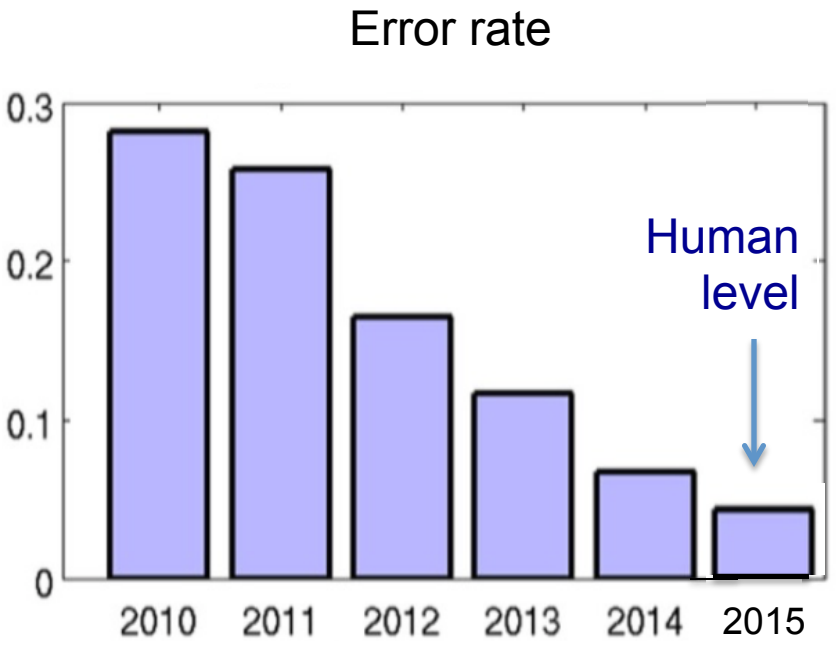
Objective/Error no longer convex in weights

Convolutional Neural Nets

Computer Vision



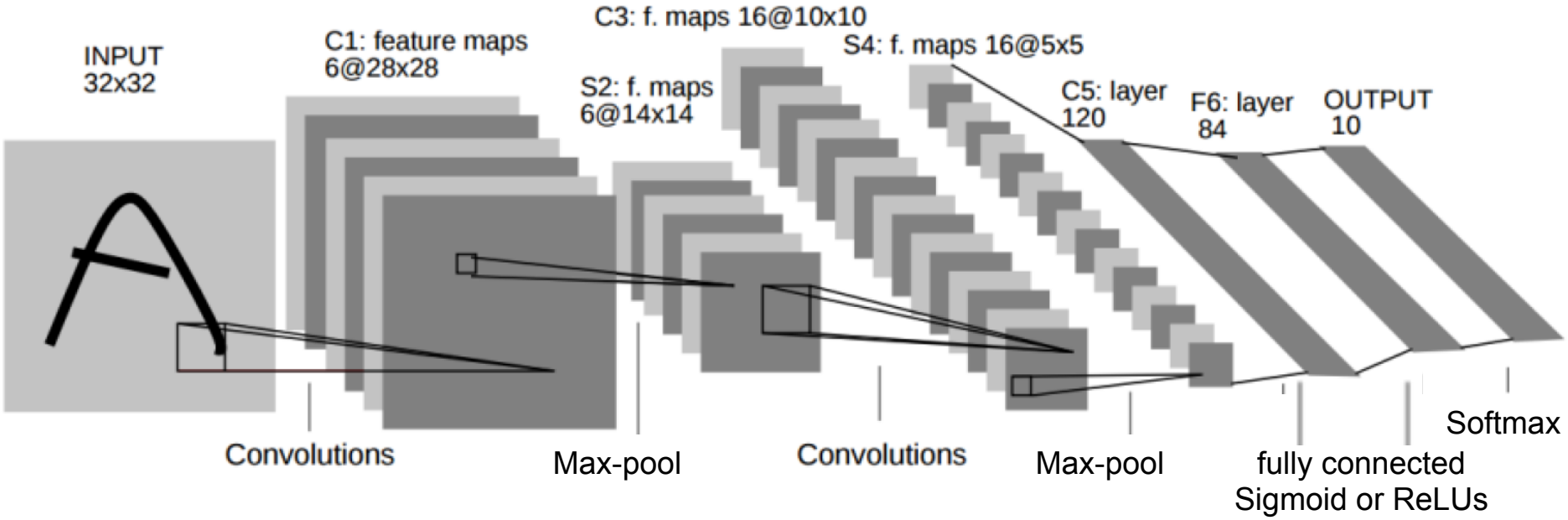
Imagenet Visual Recognition



Challenges:

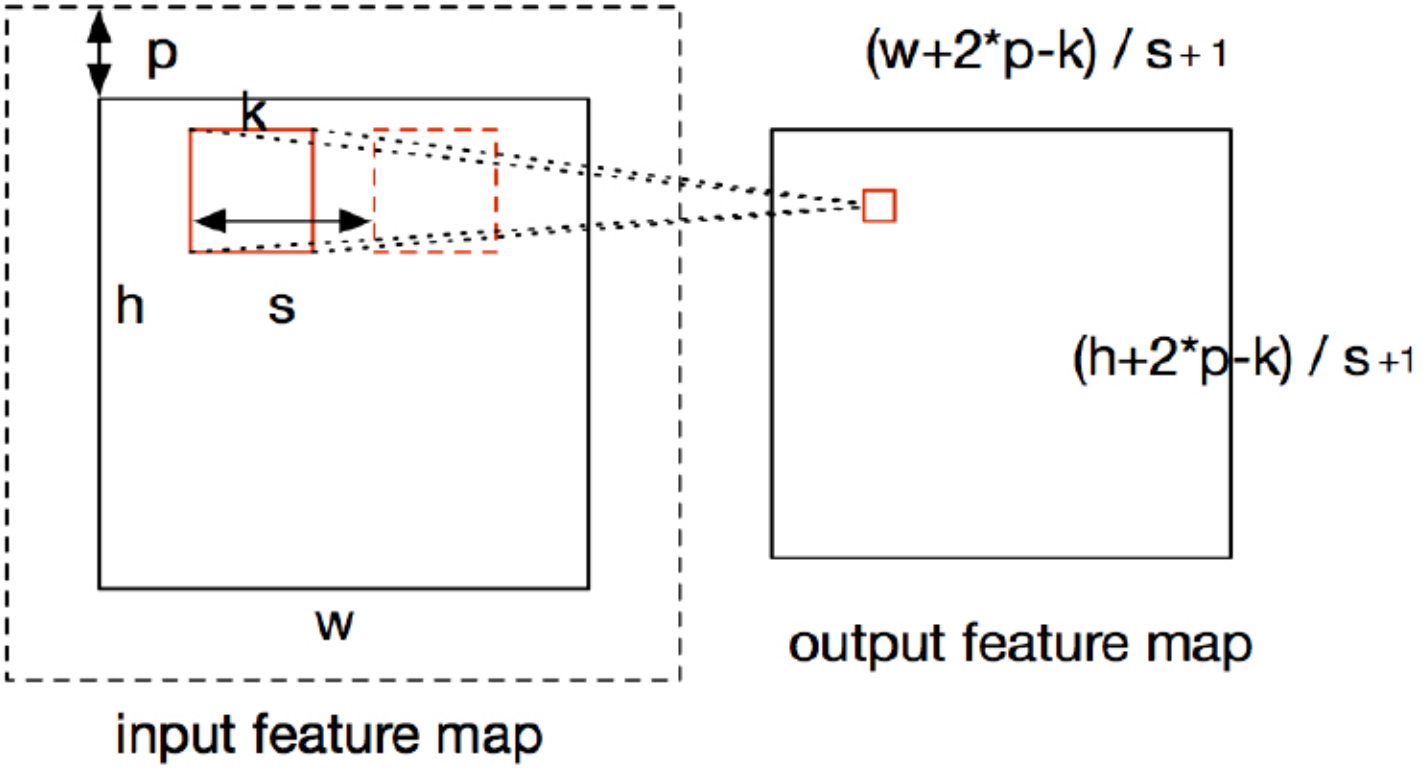
- invariance to translation, lighting, scaling, …
- manual engineering of low-level image features
- ...

A Convolutional Neural Net for Handwritten Digit recognition: LeNet



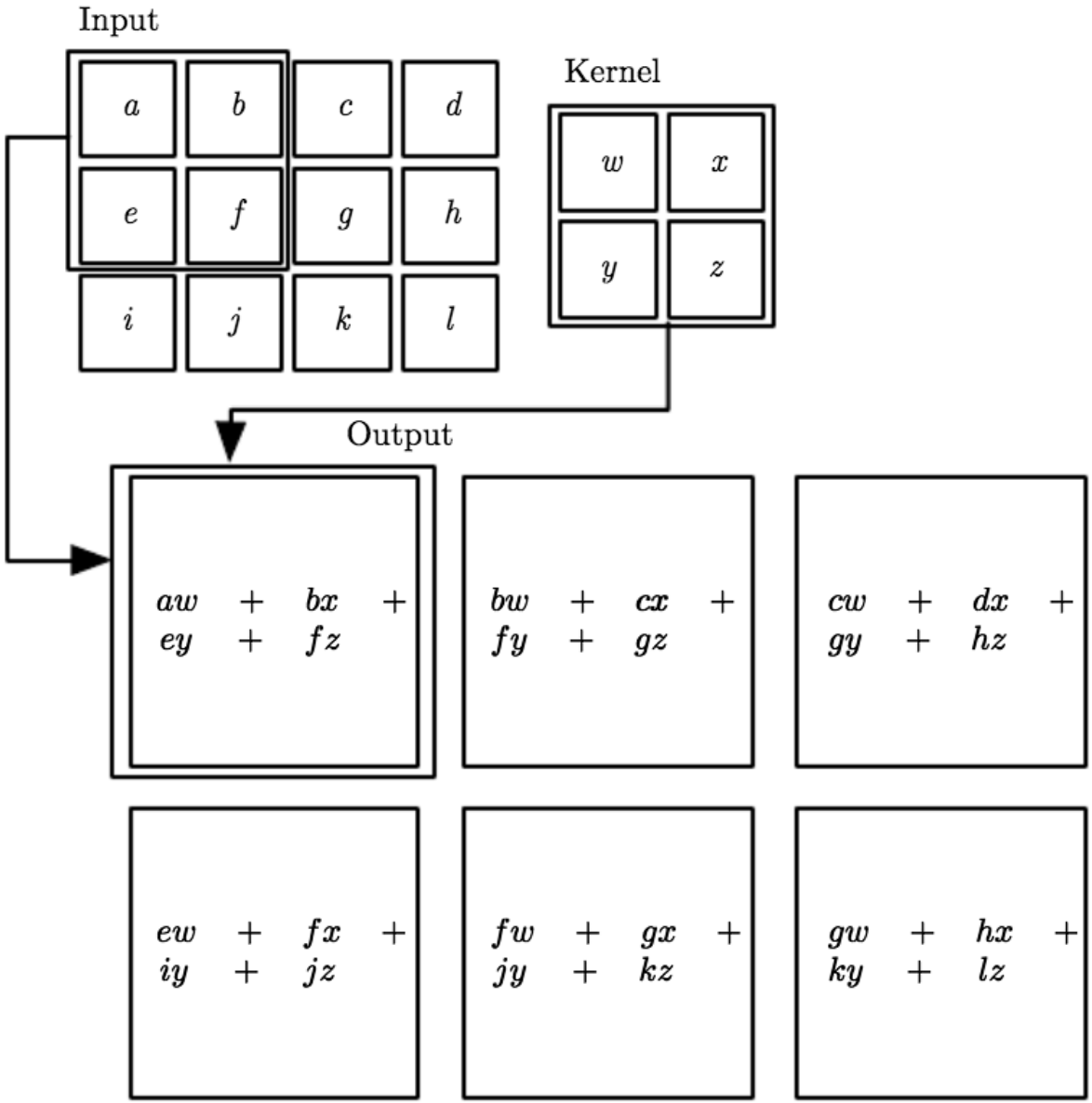
Convolution layer

p = padding
s = stride



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

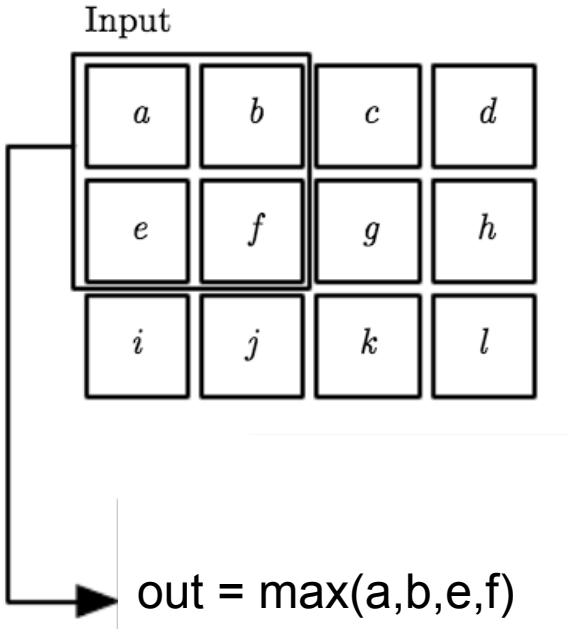
Convolution



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

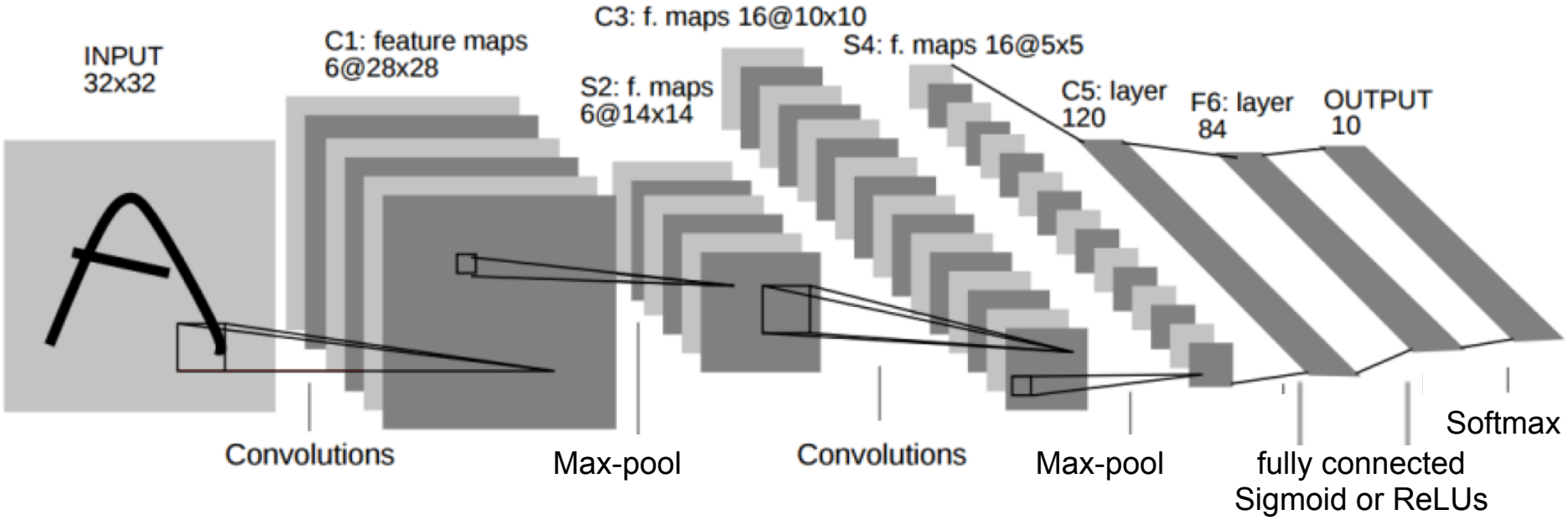
[from Goodfellow et al.]

Maxpool



[from Goodfellow et al.]

A Convolutional Neural Net for Handwritten Digit recognition: LeNet



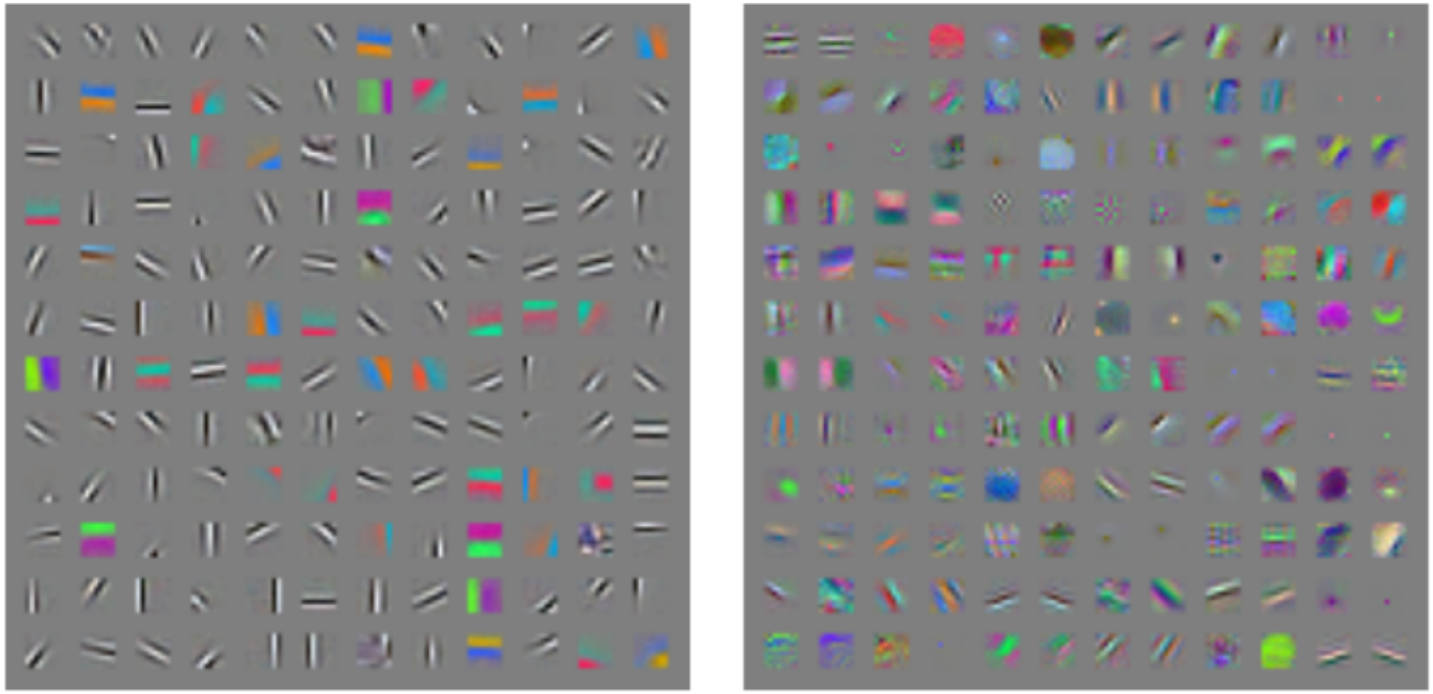
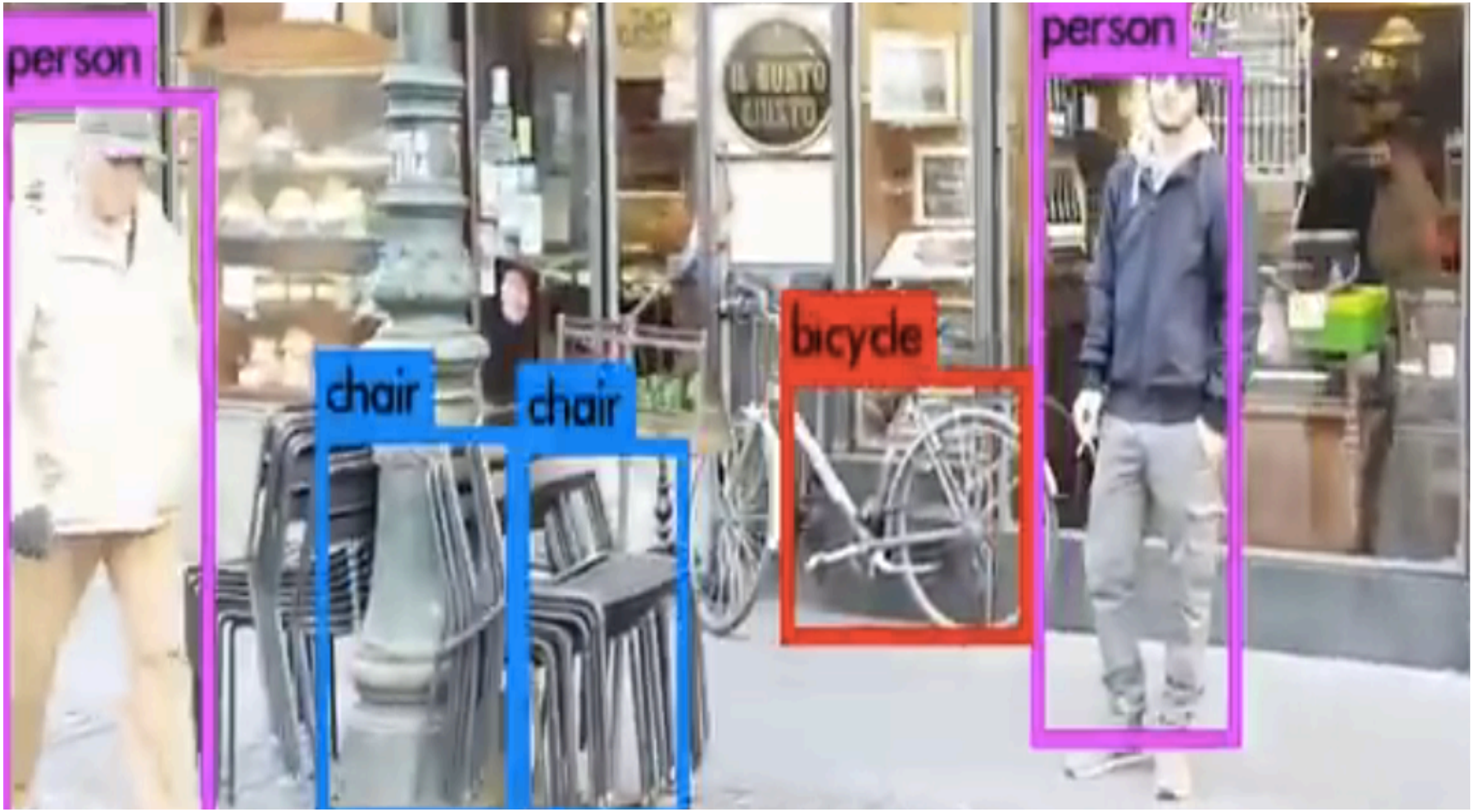


Figure 9.19: Many machine learning algorithms learn features that detect edges or specific colors of edges when applied to natural images. These feature detectors are reminiscent of the Gabor functions known to be present in primary visual cortex. *(Left)*Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches. *(Right)*Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

[from Goodfellow et al.]


person
chair
chair
bicycle
person

man in black shirt is playing guitar.

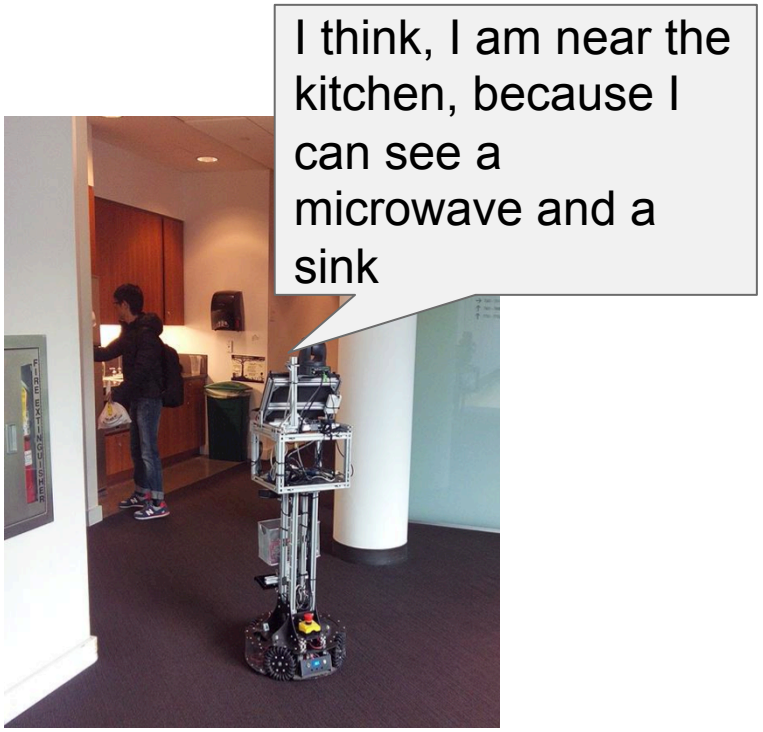
construction worker in orange safety vest is working on road.

Explaining Robot Actions

Sai Selvaraj, RI Master thesis, Stephanie Rosenthal

People's understanding of robots increases when robots explain

We use the important regions in the image to explain the scene classification.



Explaining Robot Actions

Algorithm:

Given an image *I*, algorithm for generating an explanation for a classification is:

- Using all the training images generate $E_{C,R}$ for all the classes
- Find E_R for the test image *I*, belonging to the class *y*
- Intersection of E_R and $E_{C,R}[y]$ to generate an explanation for *I*

Explaining Robot's Actions

Results:

For the image belonging to Floor 3 shown in the example:

E_R = [`(1,1), pot', `(2,1), person', `(2,1), furnishing', `(2,1), chair', `(1,1), chair']

$E_{C,R}$*[Floor 3]* = [‘(1,1), pot’, ‘(2,1), person’, ‘(2,1), furnishing’, ‘(2,1), chair’]

Final sentence: “I am in Floor 3, because I see pot in the center, person in left center, furnishing in left center, and chair in left center.”

Explaining Robot's Actions

Results:

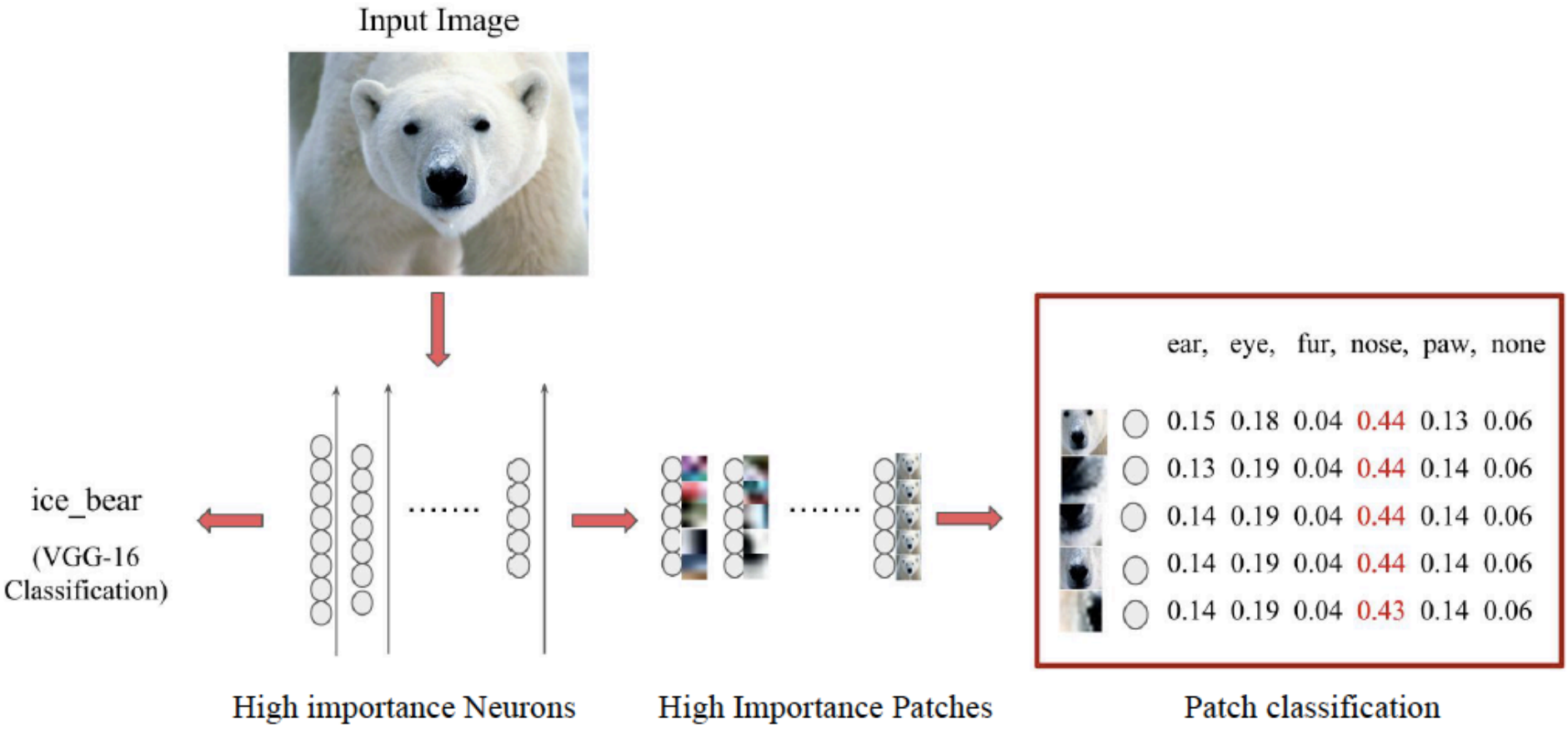
- Left image: "I am in Floor 3, because I see pot in the center, person in left center, furnishing in left center, and chair in left center."
- Center image: "I am in floor 5, because I see chair at right bottom, and furnishing at right bottom."
- Right image: "I am in floor 6, because I see chair at right center, and sofa at center."

Explaining Deep Learning

Automatic Patch Pattern Labeling for Explanation (APPLE)
(Sandeep Konam, RI Master thesis 2017)

- Find high importance neurons within the CNN
- Deconvolve the network to determine the patch of the image that each important neuron looks at
- Manually label some of those patches to learn a classifier
- Automatically label all the important patches using the learned patch classifier to determine the explanation of the important object features.

Explaining DL: APPLE



High Importance Neurons

Signal propagation equation $z^l_{x,y} = \sum_{i,j} w^l_{i,j} \phi(z^{l-1}_{i,j}) + b^l_{x,y}$

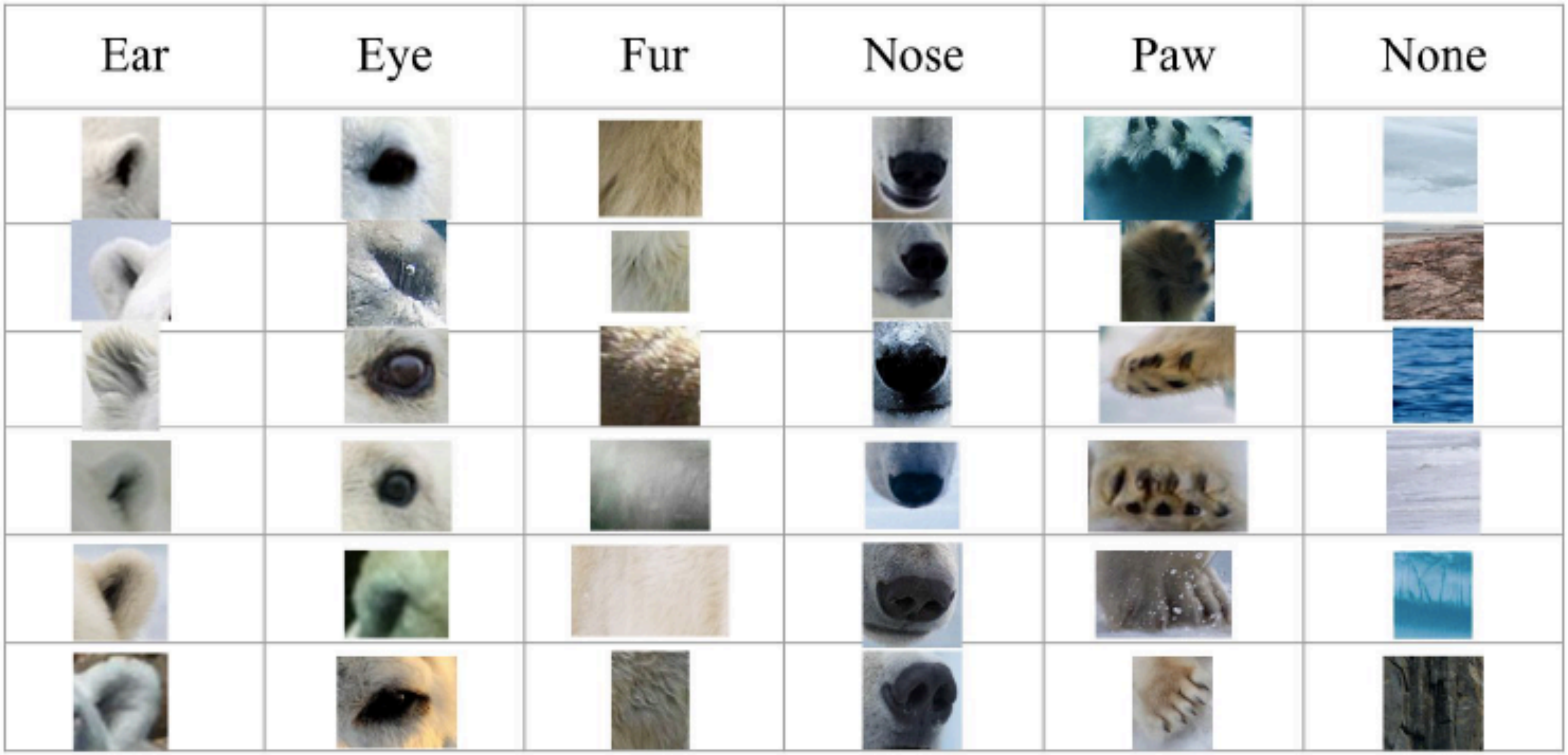
Activation	Activation Matrix Sum	$\sum_{row,col} z^l_{x,y}[row][col]$
	Activation Matrix Variance	$Var_{row,col}\ z^l_{x,y}[row][col]$
Weight	Weight Matrix Sum	$\sum_{row,col} w^{l+1}_{x,y}[row][col]$
	Weight Matrix Variance	$Var_{row,col}\ w^{l+1}_{x,y}[row][col]$

High Importance Patches

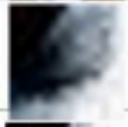
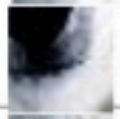
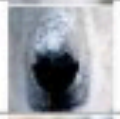
- Deconcolve the network

Patch Classifier

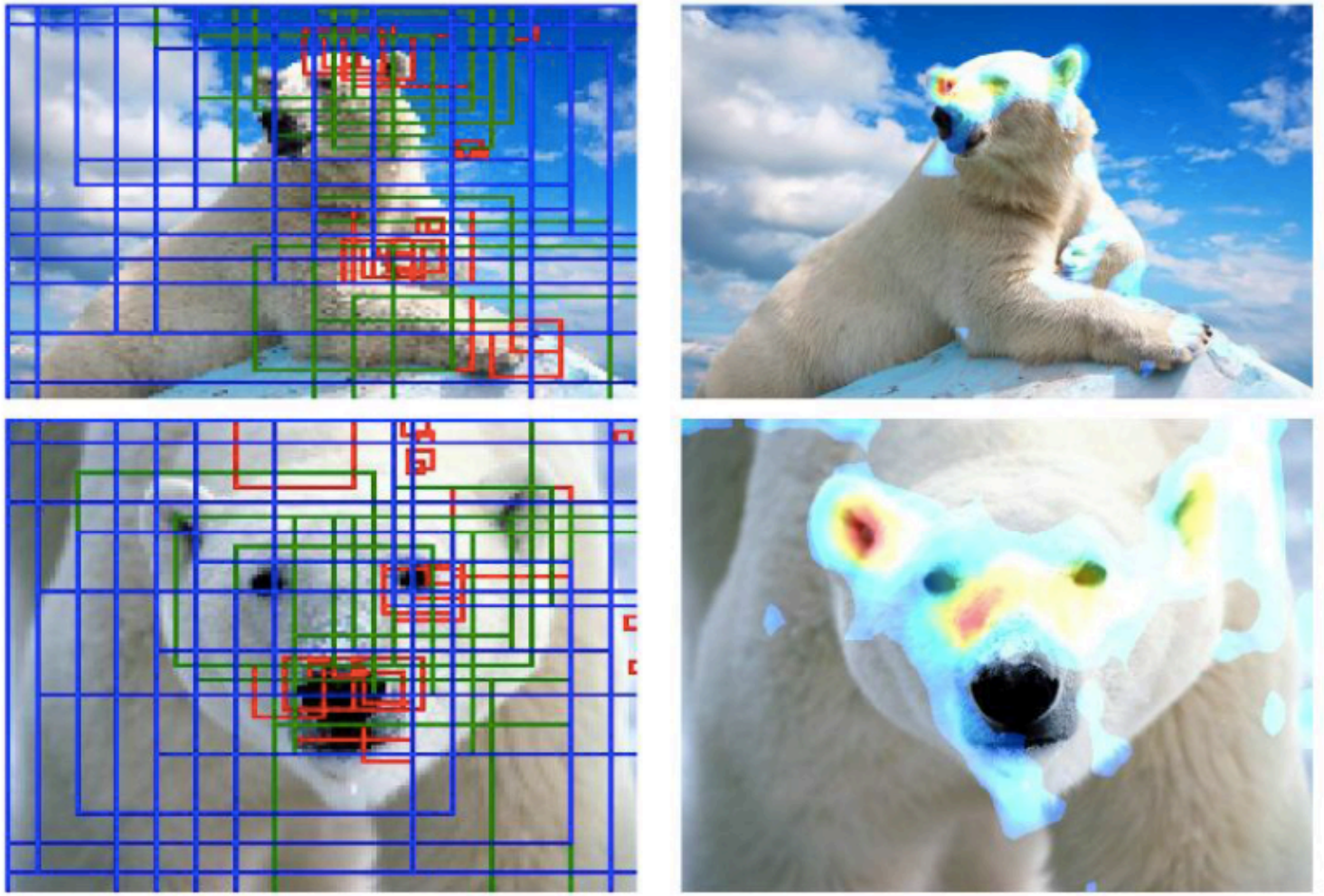
A set of object features as classifier labels (e.g., eyes, nose, ear, fur, and paws for polar bears)



is classified as **ice_bear** because of

Layer	Neuron	Patches	Predictions: ear, eye, fur, nose, paw, none
Layer : 7	Neuron : 4		0.15 0.18 0.04 **0.44** 0.13 0.06
Layer : 3	Neuron : 2		0.13 0.19 0.04 **0.44** 0.14 0.06
Layer : 4	Neuron : 2		0.14 0.19 0.04 **0.44** 0.14 0.06
Layer : 6	Neuron : 3		0.14 0.19 0.04 **0.44** 0.14 0.06
Layer : 3	Neuron : 0		0.14 0.19 0.04 **0.43** 0.14 0.06

- In APPLE (left), red boxes indicate layers 3 and 4, green indicates layers 5 - 7 and blue indicates layers 8 and 9. On the CAM images (right), the heatmap visualizes its important pixels.



Weakly supervised localization of APPLE

Action : Stop
Classification : Person

Layer	Neuron	Patches	Predictions: head, torso, hand, leg, foot, None
Layer : 3	Neuron : 0		0.94 0.04 0.01 0.01 0.00 0.00
Layer : 6	Neuron : 3		0.19 0.62 0.04 0.03 0.08 0.03
Layer : 4	Neuron : 2		0.53 0.34 0.02 0.05 0.05 0.02
Layer : 3	Neuron : 2		0.22 0.30 0.05 0.27 0.08 0.08
Layer : 7	Neuron : 4		0.22 0.30 0.07 0.21 0.08 0.12

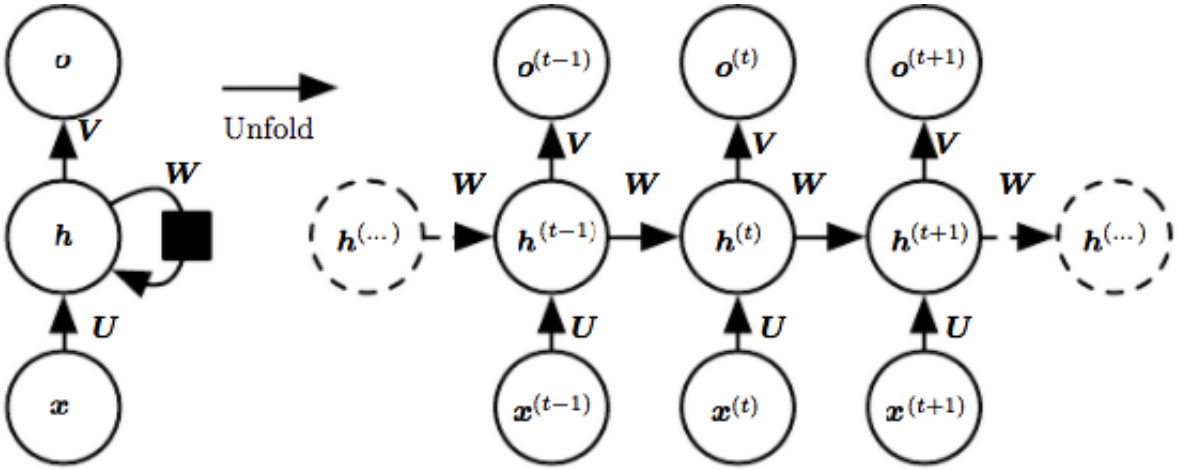
APPLE sorts the classified labeled patches by confidence

Sequential Neural Nets

Recurrent Networks

- Many tasks involve sequential data
  - predict stock price at time t+1 based on prices at t, t-1, t-2, …
  - translate sentences: (word sequences) from Spanish to English
  - speech recognition: (sound sequences) to text (word sequences)

- Key idea: recurrent network uses (part of) its state at *t* as its input at *t+1*

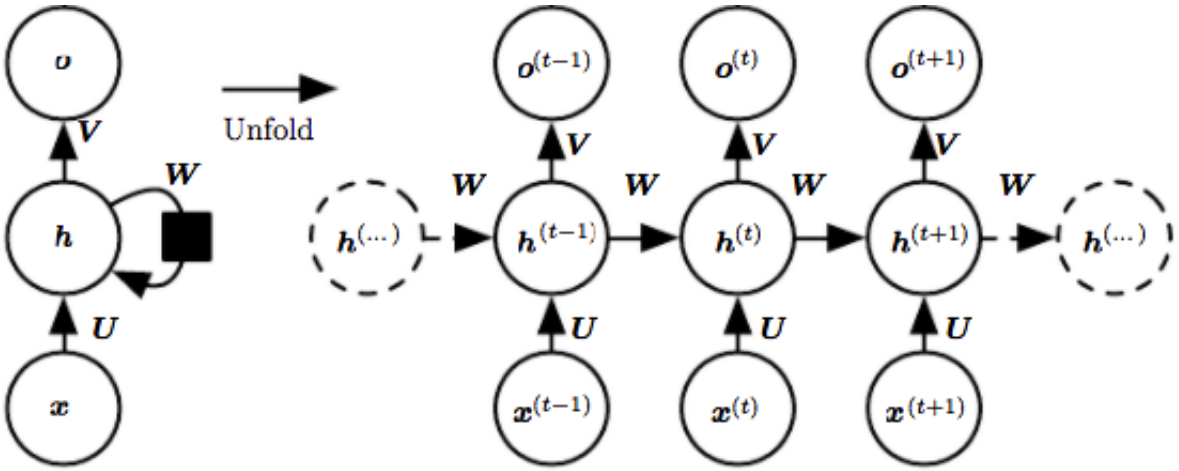


[Goodfellow et al., 2016]

Training Recurrent Networks

Key principle for training:

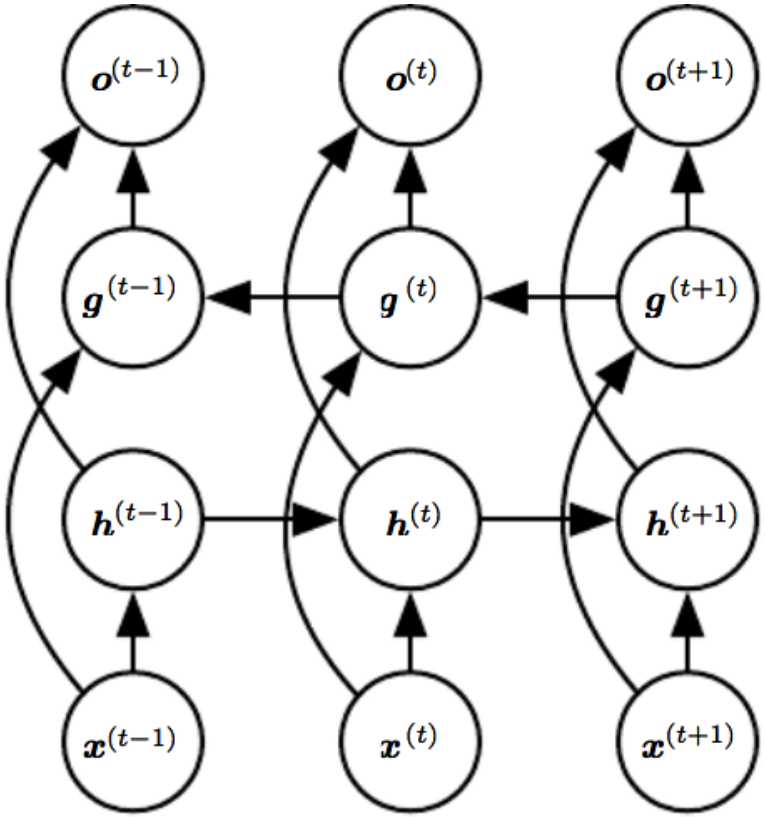
1. Treat as if unfolded in time, resulting in directed acyclic graph
2. Note shared parameters in unfolded net → sum the gradients



[Goodfellow et al., 2016]

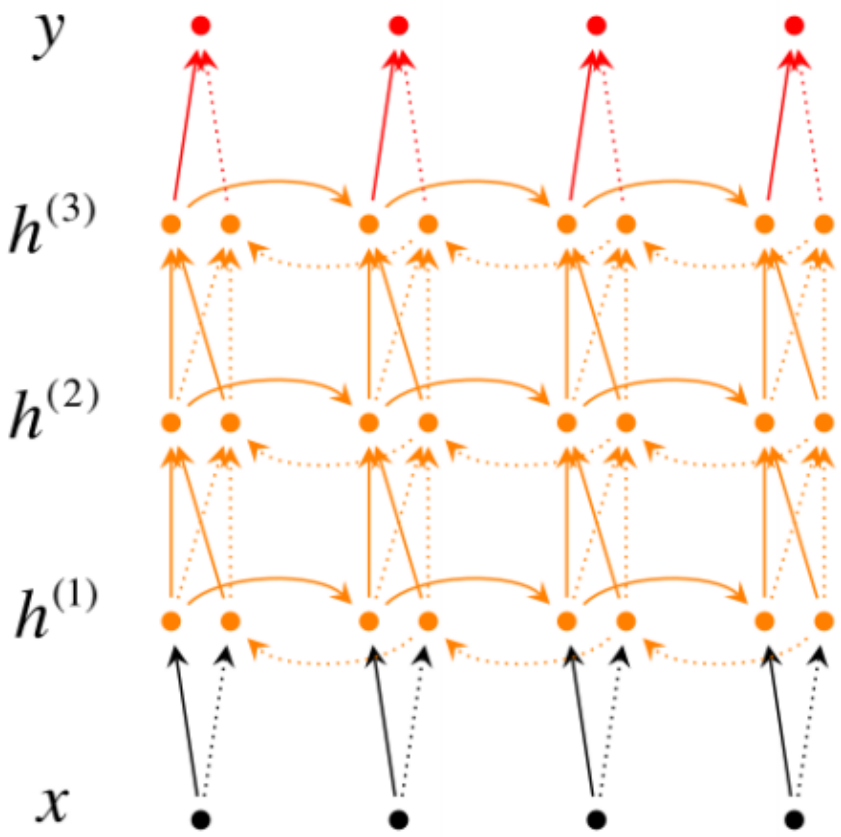
Bi-directional Recurrent Neural Networks

- Key idea: processing of word at position t can depend on following words too, not just preceding words



[Goodfellow et al., 2016]

Deep Bidirectional Recurrent Network



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

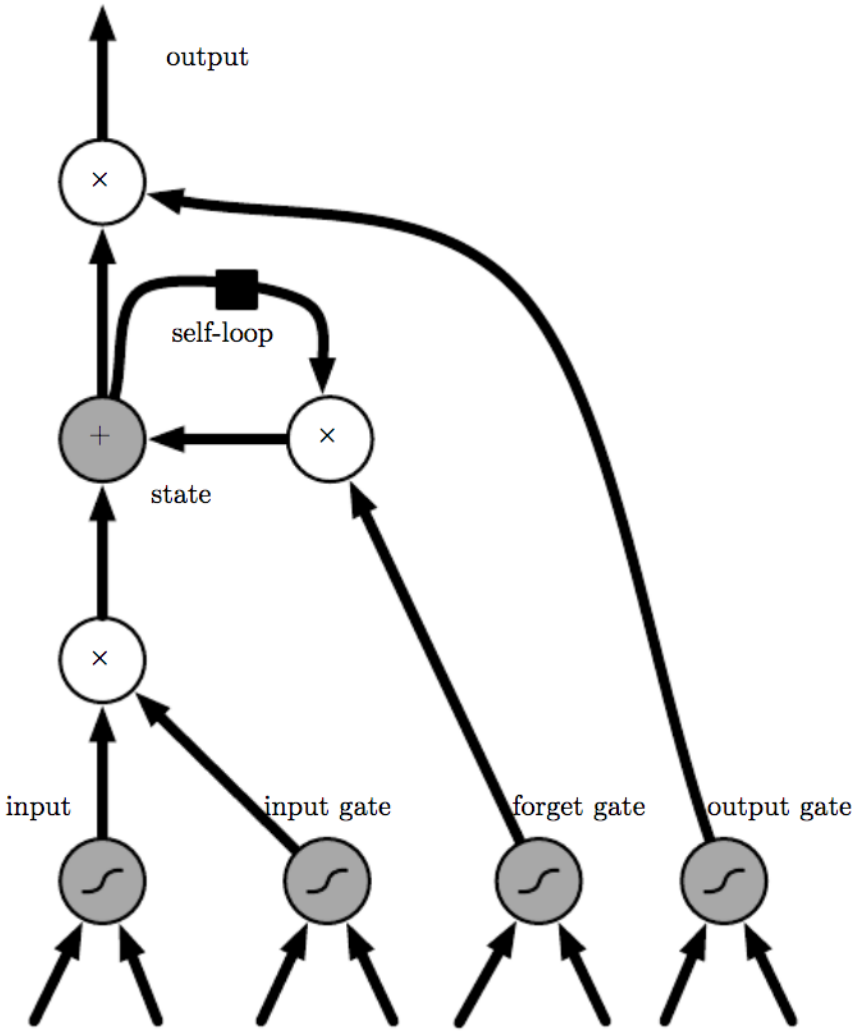
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each bidirectional layer builds on the one below

[Irsoy & Cardie, 2014]

Long Short Term Memory (LSTM) unit



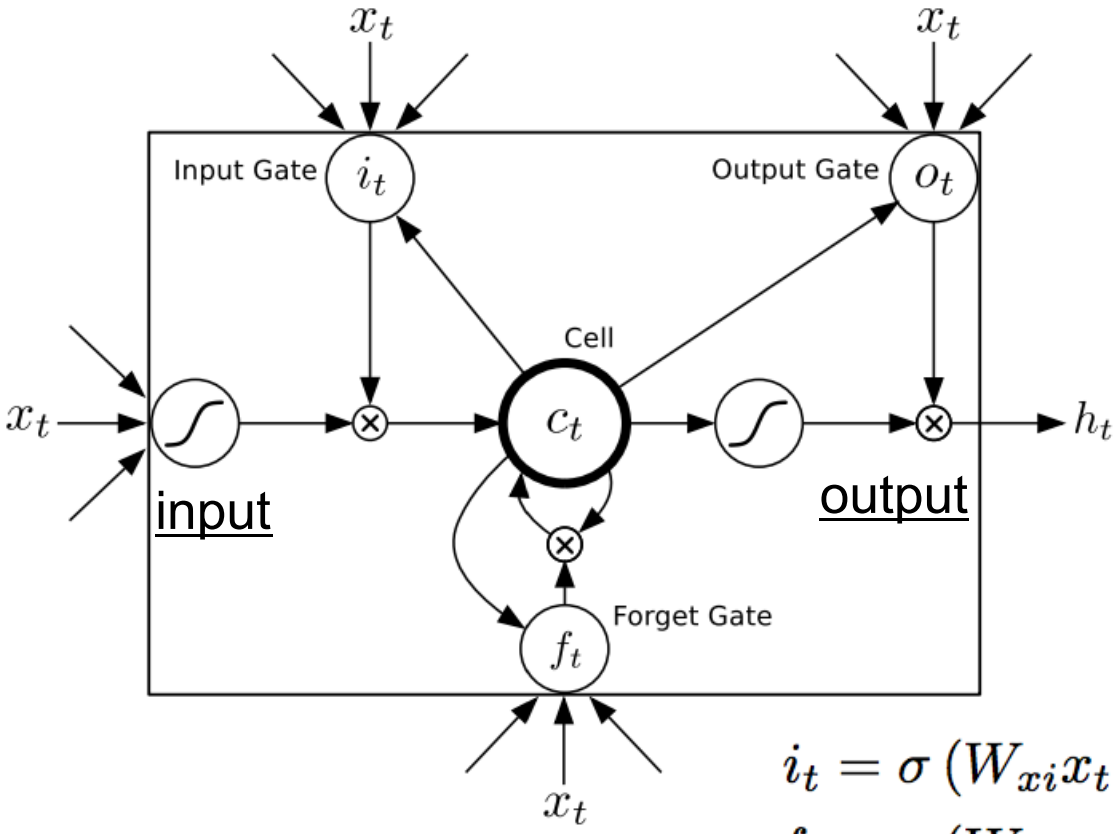
input gate: if 1, add input to memory state

forget gate: if 0, zero out memory state, else retain (partially)

output gate: if 1, read memory to output

[Goodfellow et al., 2016]

Long Short Term Memory (LSTM) unit



input gate: if outputs 1, add input to memory cell

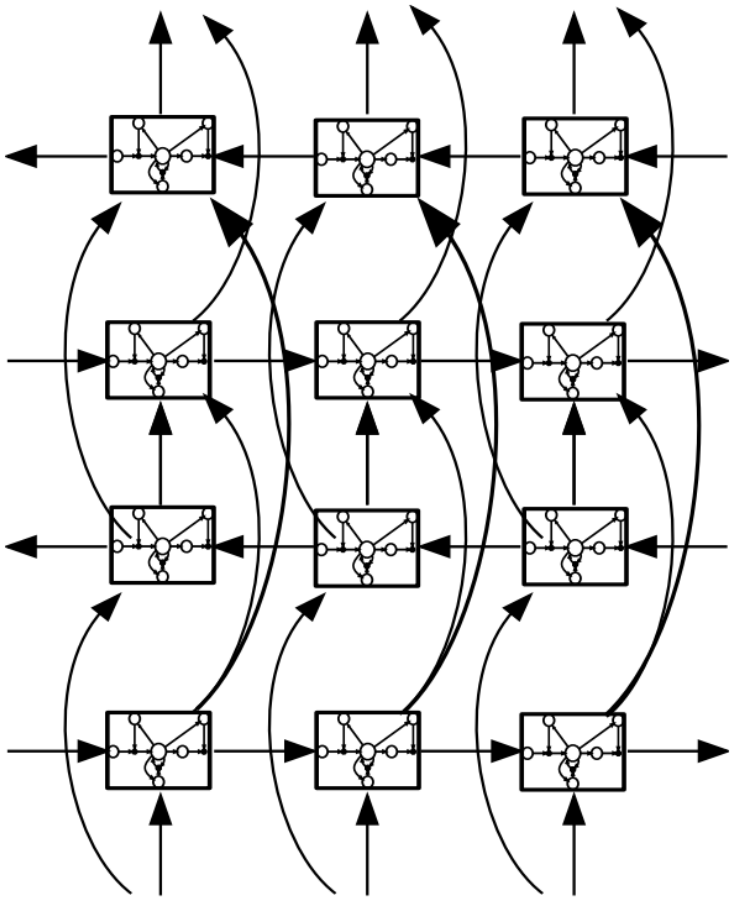
forget gate: if 0, zero out memory cell, else retain

output gate: if 1, read memory to output

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

[Graves et al., 2014]

Deep Bidirectional LSTM Network



[“Hybrid Speech Recognition with Deep Bidirectional LSTM,” Graves et al., 2013]

Programming Frameworks for Deep Nets

- TensorFlow (Google)
- TFLearn (runs on top of TensorFlow, but simpler to use)
- Theano (University of Montreal)
- Pytorch (Facebook)
- CNTK (Microsoft)
- Keras (can run on top of Theano, CNTK, TensorFlow)

Many support use of Graphics Processing Units (GPU's)

Major factor in dissemination of Deep Network technology

```
# Specify that all features have real-value data
feature_columns = [tf.feature_column.numeric_column("x", shape=[4])]

# Build 3 layer DNN with 10, 20, 10 units respectively.
classifier = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                        hidden_units=[10, 20, 10],
                                        n_classes=3,
                                        model_dir="/tmp/iris_model")
# Define the training inputs
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(training_set.data)},
    y=np.array(training_set.target),
    num_epochs=None,
    shuffle=True)

# Train model.
classifier.train(input_fn=train_input_fn, steps=2000)

# Define the test inputs
test_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(test_set.data)},
    y=np.array(test_set.target),
    num_epochs=1,
    shuffle=False)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(input_fn=test_input_fn)["accuracy"]

print("\nTest Accuracy: {0:f}\n".format(accuracy_score))
```

TensorFlow example

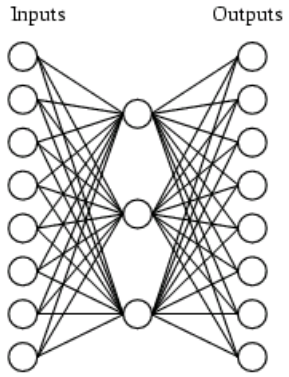
What you should know:

- Backpropagation algorithm
  - training network with gradient descent
  - how to derive gradient for simple cost functions, units
  - relationship to logistic regression
- Neural nets learn internal representations
- Convolutional neural networks
  - convolution operation

Limited by amount of labeled data.
What about unsupervised problems?

Auto-Encoders
Deep Generative Models

Learning Hidden Layer Representations

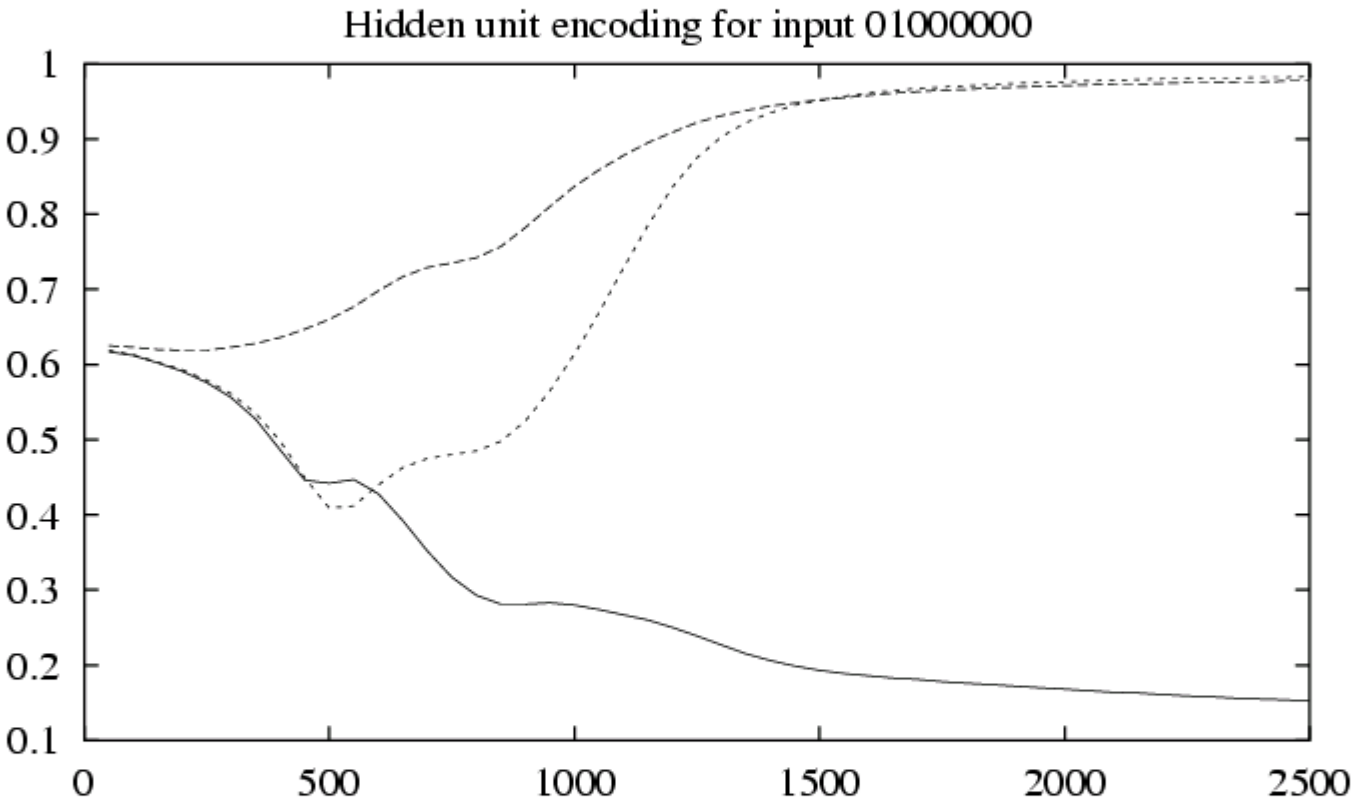


A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

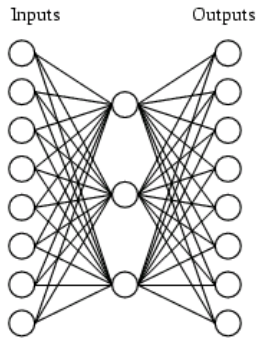
Can this be learned??

Training


Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

Learning Hidden Layer Representations

A network:

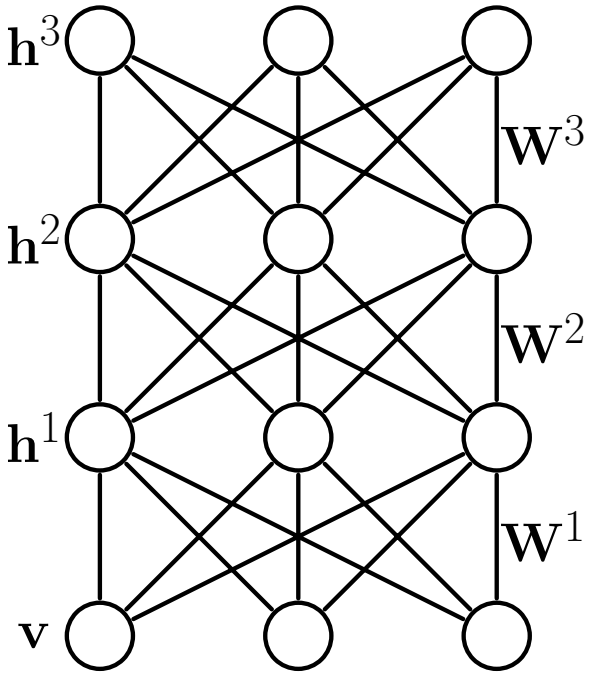


Learned hidden layer representation:

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

Deep Boltzmann Machine (DBM)

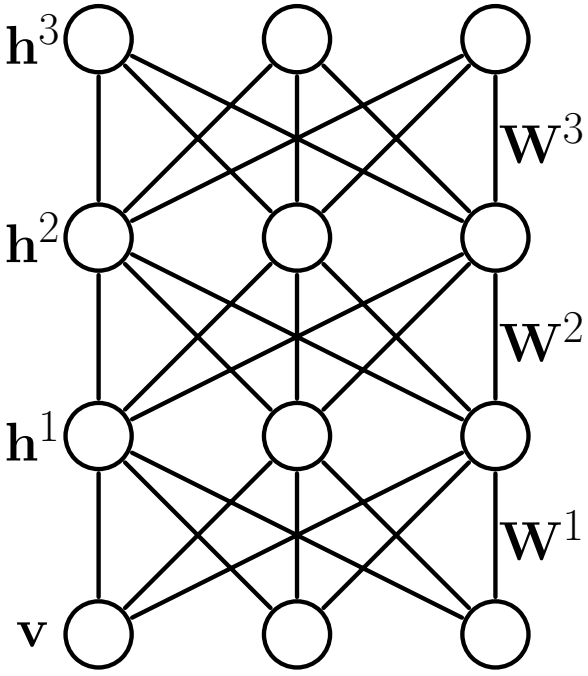
Deep Generative Model (Salakhutdinov 2008, Salakhutdinov & Hinton 2012)



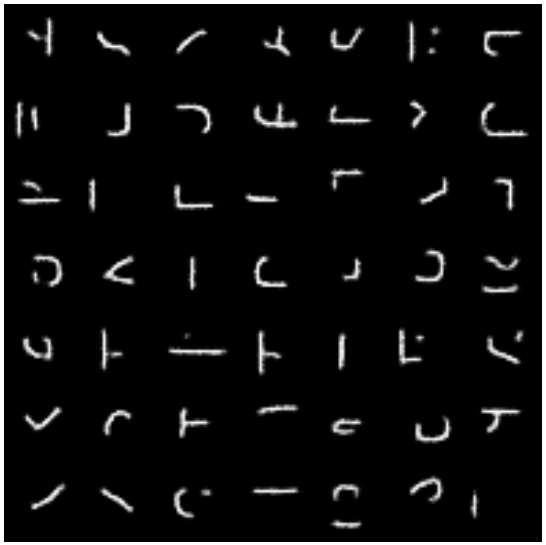
$$P_\theta(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left[\mathbf{v}^\top W^{(1)} \mathbf{h}^{(1)} + {\mathbf{h}^{(1)}}^\top W^{(2)} \mathbf{h}^{(2)} + {\mathbf{h}^{(2)}}^\top W^{(3)} \mathbf{h}^{(3)}\right]$$

Bernoulli Markov Random Field

Hand-written Character Recognition

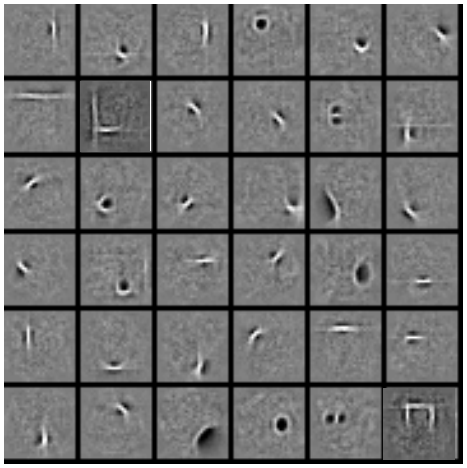


Learned 2nd-layer features



Strokes

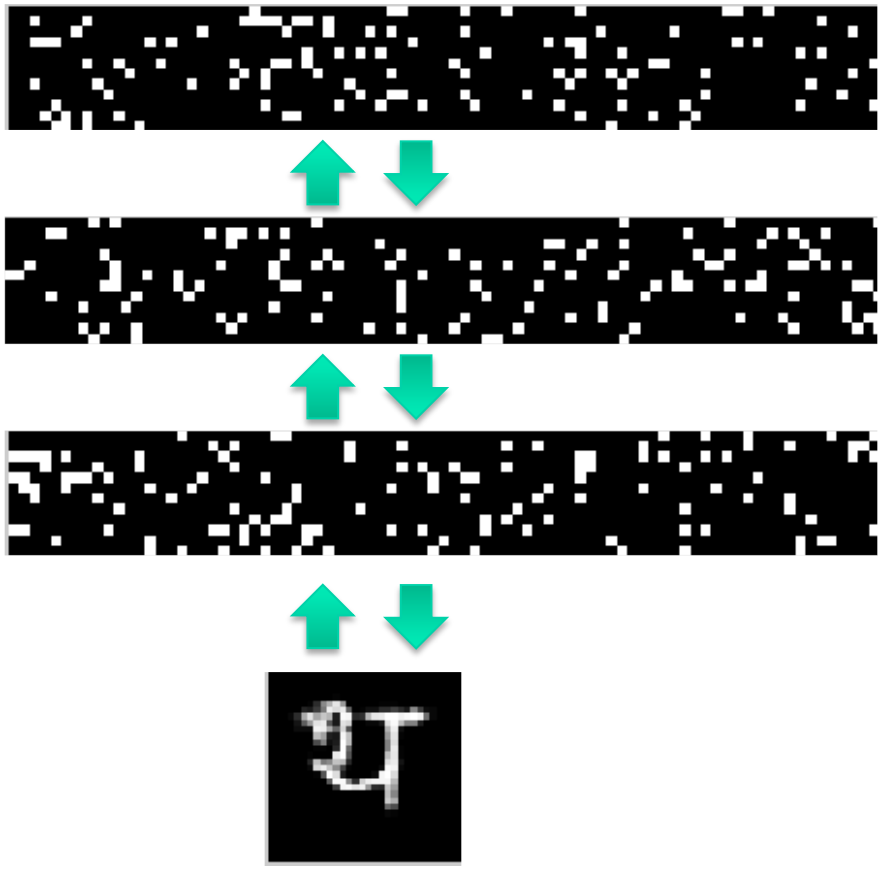
Learned 1st layer features



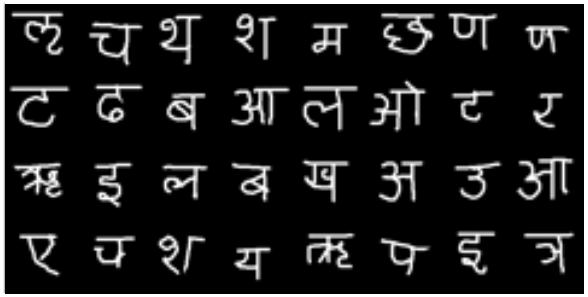
Edges

Deep Boltzmann Machines for Text Characters

Sanskrit



Model P(image)



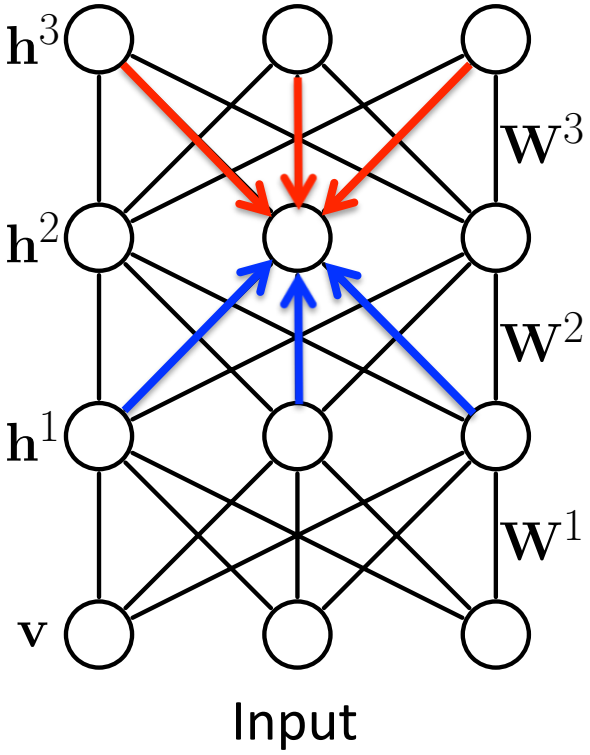
Subset of 25,000 characters from 50 alphabets around the world.

- 3,000 hidden variables
- 784 observed variables (28 by 28 images)
- About 2 million parameters

Bernoulli Markov Random Field

DBMs Model Formulation

$$P_\theta(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left[\mathbf{v}^\top W^{(1)} \mathbf{h}^{(1)} + \mathbf{h}^{(1)\top} W^{(2)} \mathbf{h}^{(2)} + \mathbf{h}^{(2)\top} W^{(3)} \mathbf{h}^{(3)}\right]$$



$\theta = \{W^1, W^2, W^3\}$ model parameters

- Dependencies between hidden variables.
- All connections are undirected.
- Bottom-up and Top-down:

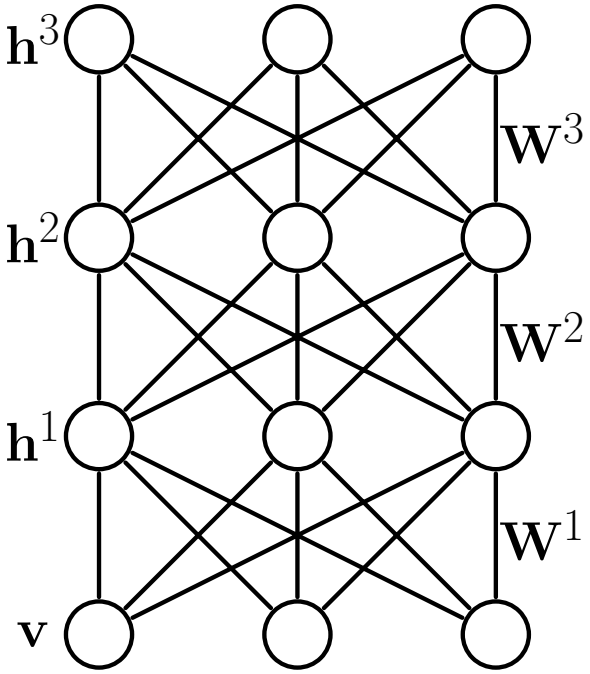
$$P(h_j^2 = 1 | \mathbf{h}^1, \mathbf{h}^3) = \sigma\left(\sum_k W_{kj}^3 h_k^3 + \sum_m W_{mj}^2 h_m^1\right)$$

Top-down Bottom-up

- Hidden variables are dependent even when **conditioned on the input**.

Parameter Learning for DBMs

$$P_\theta(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left[\mathbf{v}^\top W^{(1)} \mathbf{h}^{(1)} + \mathbf{h}^{(1)\top} W^{(2)} \mathbf{h}^{(2)} + \mathbf{h}^{(2)\top} W^{(3)} \mathbf{h}^{(3)}\right]$$



(Approximate) Maximum Likelihood:

$$\frac{\partial \log P_\theta(\mathbf{v})}{\partial W^1} = \mathbb{E}_{P_{data}}[\mathbf{v}\mathbf{h}^{\mathbf{1}\top}] - \mathbb{E}_{P_\theta}[\mathbf{v}\mathbf{h}^{\mathbf{1}\top}]$$

Variational Inference

Stochastic Approximation (MCMC-based)

$$P_{data}(\mathbf{v}, \mathbf{h}^{\mathbf{1}}) = P_\theta(\mathbf{h}^{\mathbf{1}}|\mathbf{v}) P_{data}(\mathbf{v})$$

$$P_{data}(\mathbf{v}) = \frac{1}{N} \sum_{n=1}^{N} \delta(\mathbf{v} - \mathbf{v_n})$$

Can not be computed exactly

Artificial Neural Networks: Summary

- Actively used to model distributed computation in brain
- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate - overfitting
- Hidden layers learn intermediate representations – how many to use?

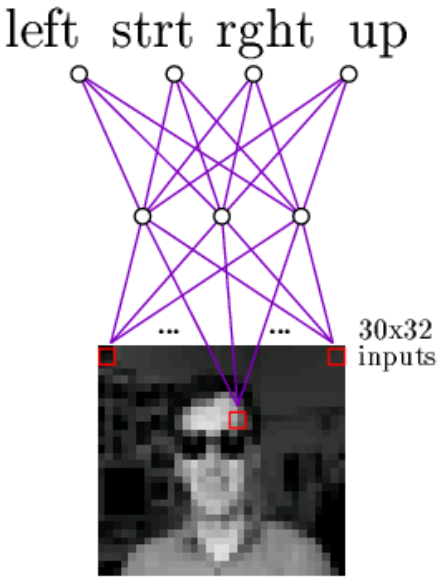
- Prediction – Forward propagation
- Gradient descent (Back-propagation), local minima problems

- Coming back in new form as deep networks

Summary

- Neural networks used for
  - Approximating *y* as function of input *x* (regression)
  - Predicting (discrete) class *y* as function of input *x* (classification)
- Key Concepts:
  - Difference between linear and sigmoid outputs
  - Gradient descent for training
  - Backpropagation for general networks
  - Use of validation data for avoiding overfitting
- Good:
  - "simple" framework
  - Direct procedure for training (gradient descent…)
  - Convergence guarantees in the linear case
- Not so good:
  - Many parameters (learning rate, etc.)
  - Need to design the architecture of the network (how many units? How many layers? What transfer function at each unit? Etc.)
  - Requires a substantial amount of engineering in designing the network
  - Training can be very slow and can get stuck in local minima

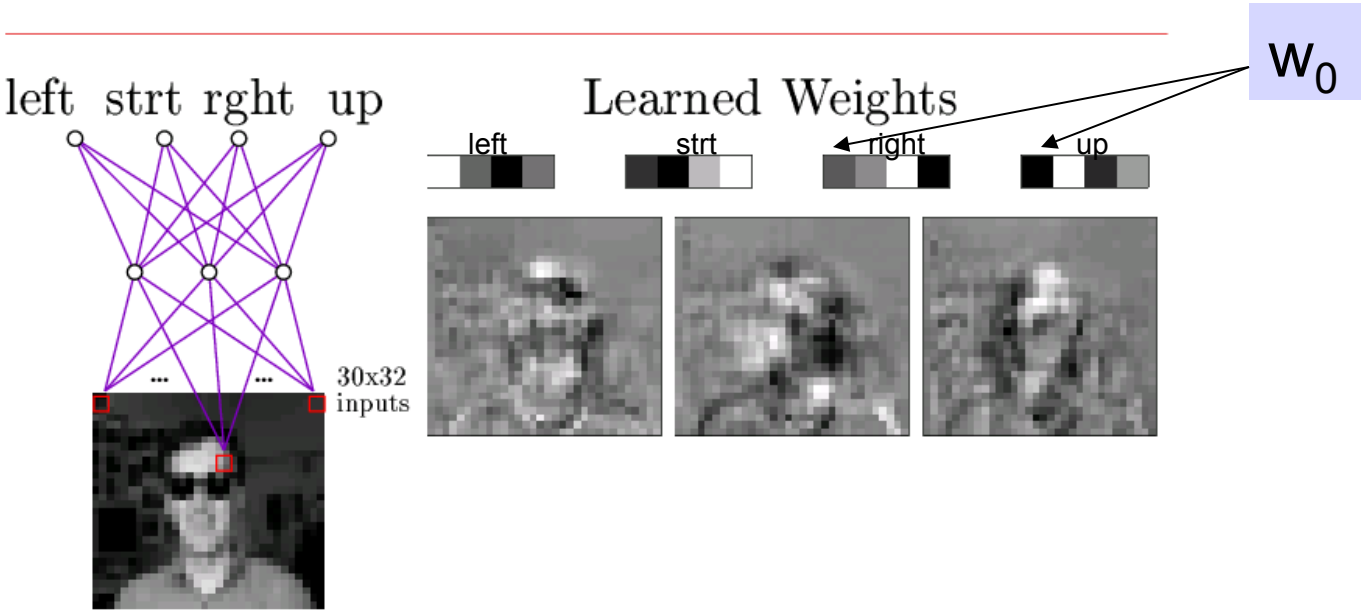
Neural Nets for Face Recognition



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Hidden Unit Weights



Typical input images

http://www.cs.cmu.edu/~tom/faces.html

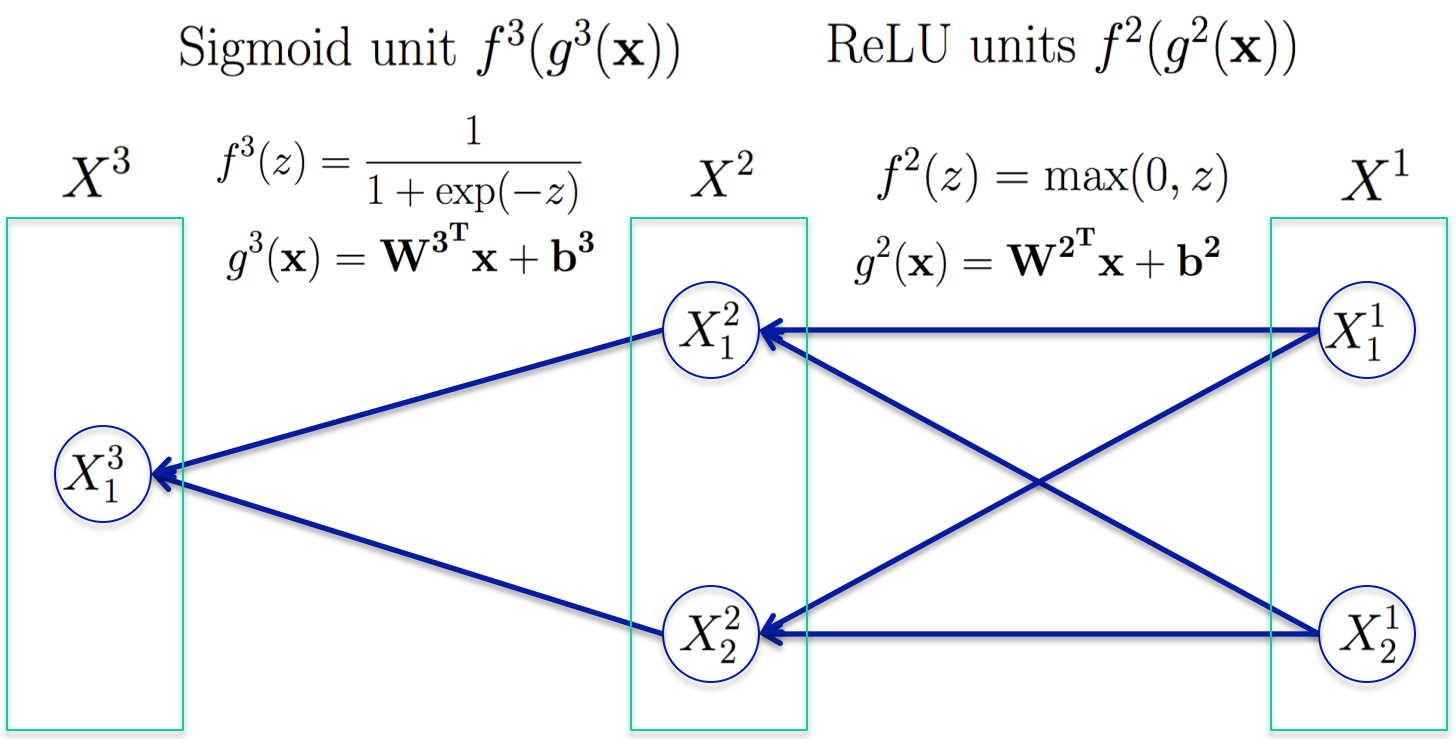
Training Deep Nets

1. Choose loss function J(θ) to optimize
   - sum of squared errors for y continuous: $\Sigma\ (y - h(x; \theta))^2$
   - maximize conditional 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
   - Experiment with alternative architectures

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



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

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

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

Example: Learn probabilistic XOR

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

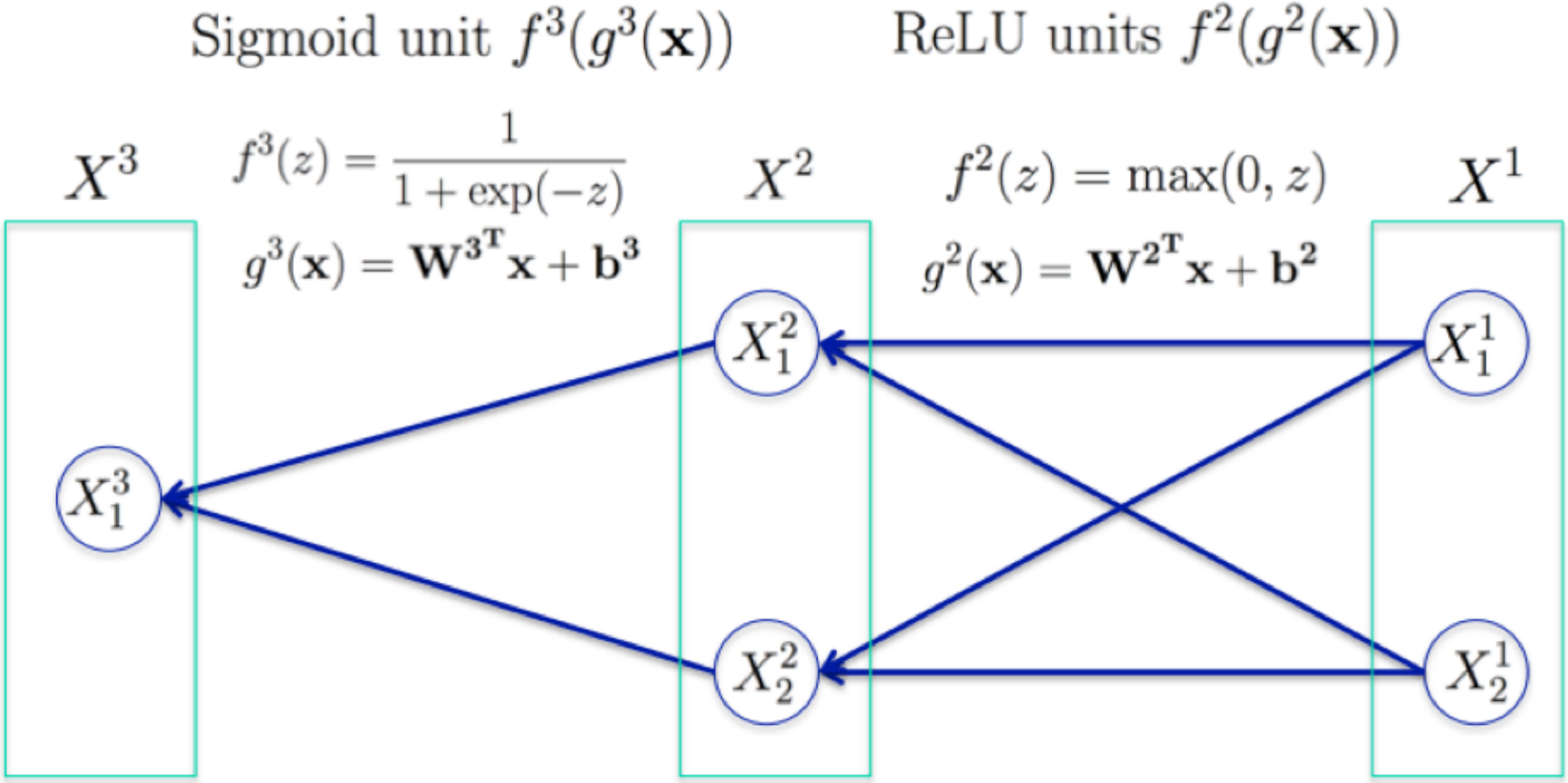
$$P(Y = 0|X_1 = X_2) = 0.9$$

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

- Choose max conditional likelihood,
  equally, minimize negative log conditional likelihood

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

Feed Forward



X^3	$g^3(X^2)$	W^{3T}		b^3
0.53	0.12	0.10	-0.09	0.10

X^2	$g^2(X^1)$	W^{2T}		b^2
0.20	0.20	0.10	-0.10	0.10
0.00	-0.15	-0.20	0.10	0.05
1				

X^1
1
0
1

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

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

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

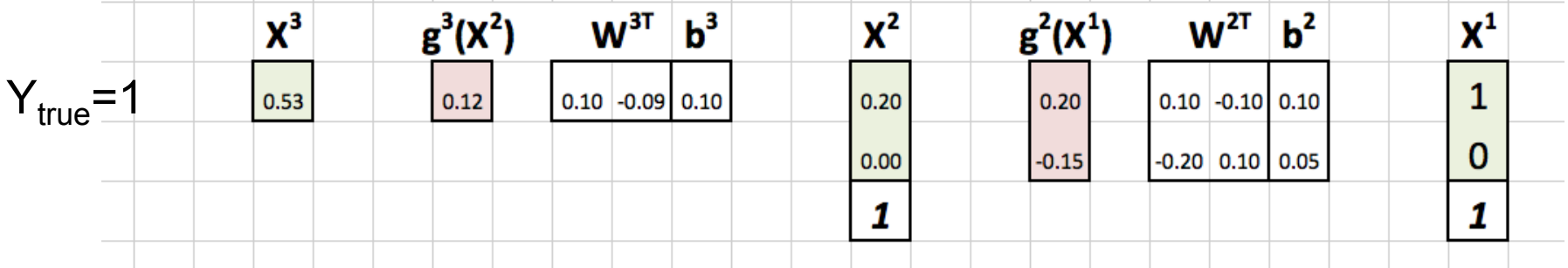
Derive the gradient we need

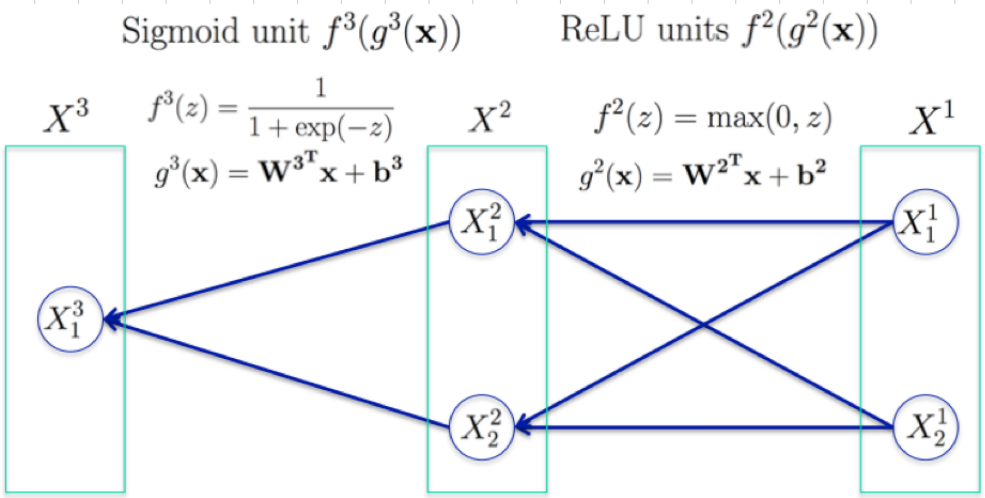
$$J(\theta) = -\sum_k \log P(Y = y_k | X = \mathbf{x}_k)$$
$$= -\sum_k y_k \log P(Y = 1 | X = x_k) + (1 - y_k) \log(1 - P(Y = 1 | X = x_k))$$

simplify notation by considering just one training example

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

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



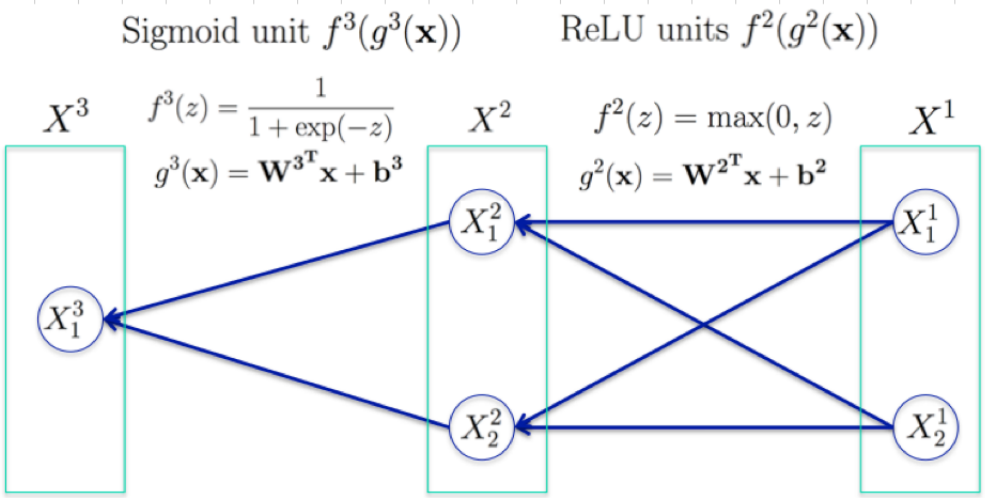


$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} =$$

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

$Y_{true}=1$

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*

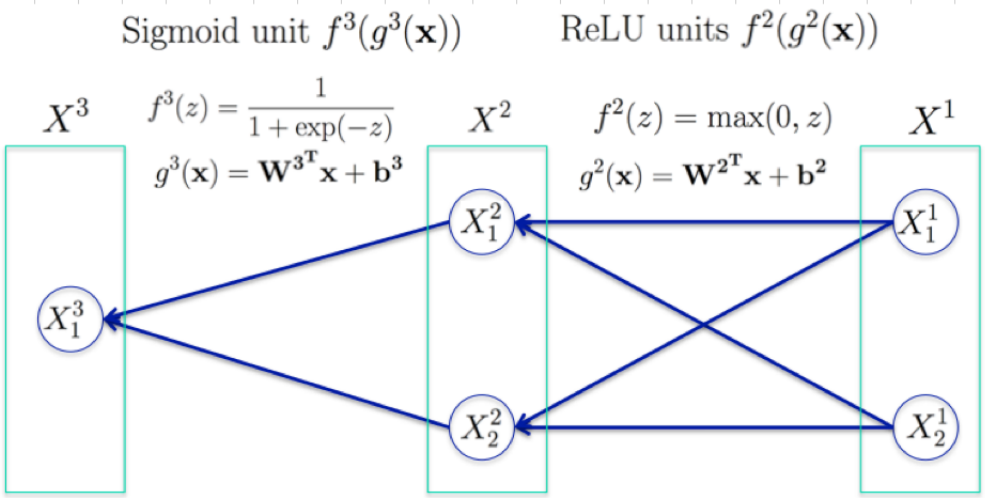


$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1-X_1^3) =$$

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

$Y_{true}=1$

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*



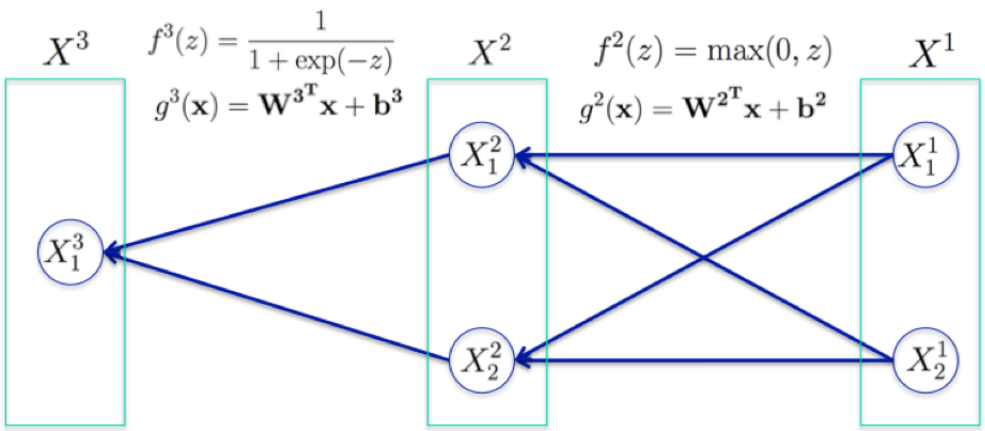
$$\frac{\partial J(\theta)}{\partial g_1^3} = \frac{\partial J(\theta)}{\partial X_1^3}\frac{\partial X_1^3}{\partial g^3} = \frac{\partial J(\theta)}{\partial X_1^3} X_1^3(1-X_1^3) \quad = -0.47$$

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

$Y_{true}=1$

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*

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



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

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

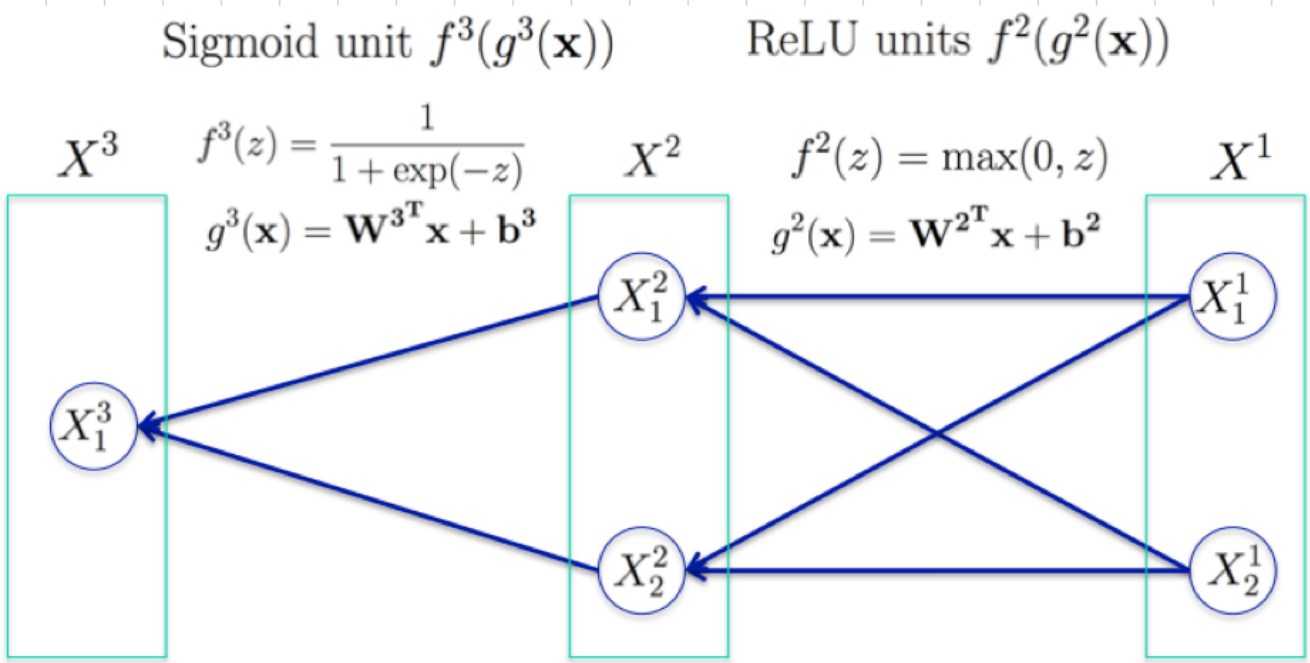
$$\frac{\partial J(\theta)}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3}\frac{\partial g^3}{\partial X_i^2} = \frac{\partial J(\theta)}{\partial g^3} w_i^3$$

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

$$\frac{\partial J(\theta)}{\partial w_{ik}^2} = \frac{\partial J(\theta)}{\partial g_i^2}\frac{\partial g_i^2}{\partial X_k^1} = \frac{\partial J(\theta)}{\partial g^3} X_k^1$$

Back propagation

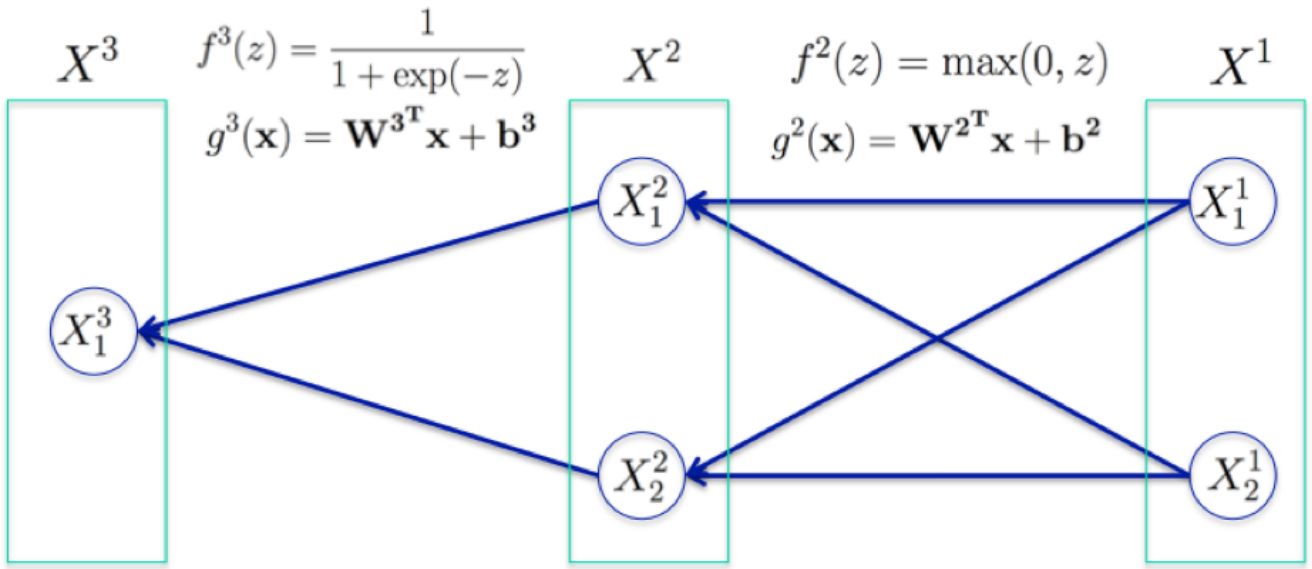
Ytrue	1		
dJ/dX3	-1.89		
dJ/dg3	-0.47		
dJ/dW3T	-0.09	0.00	-0.47
dJ/dX2	-0.05		
	0.04		
dJ/dg2	-0.05		
	0.00		
dJ/dW2T	-0.05	0.00	-0.05
	0.00	0.00	0.00



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*

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

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*

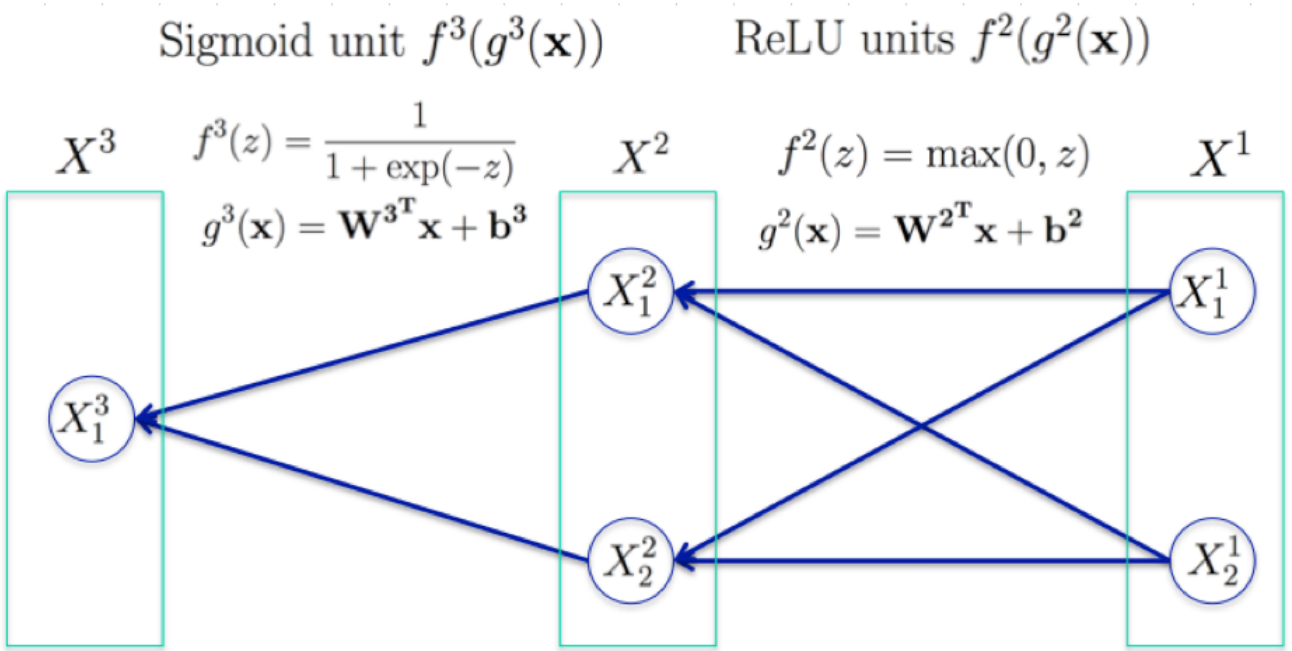
update each parameter according to $\theta_i \leftarrow \quad \theta_i - \eta \frac{\partial J(\theta)}{\partial \theta_i}$

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*

What you should know:

- Representation learning in neural networks
  - hidden layers re-represent inputs to predict outputs
  - auto-encoders
  - word embeddings
- Sequential models
  - recurrent networks, and unfolding them
  - memory units: LSTM, etc.
  - deep sequential neural networks
- Pragmatics of training deep nets
  - vanishing gradients and exploding gradients
  - gradient clipping, batch normalization, …
  - frameworks such as TensorFlow, Pytorch, ...