Lecture 18: HMM & Deep Learning

Winter 2018

Kai-Wei Chang

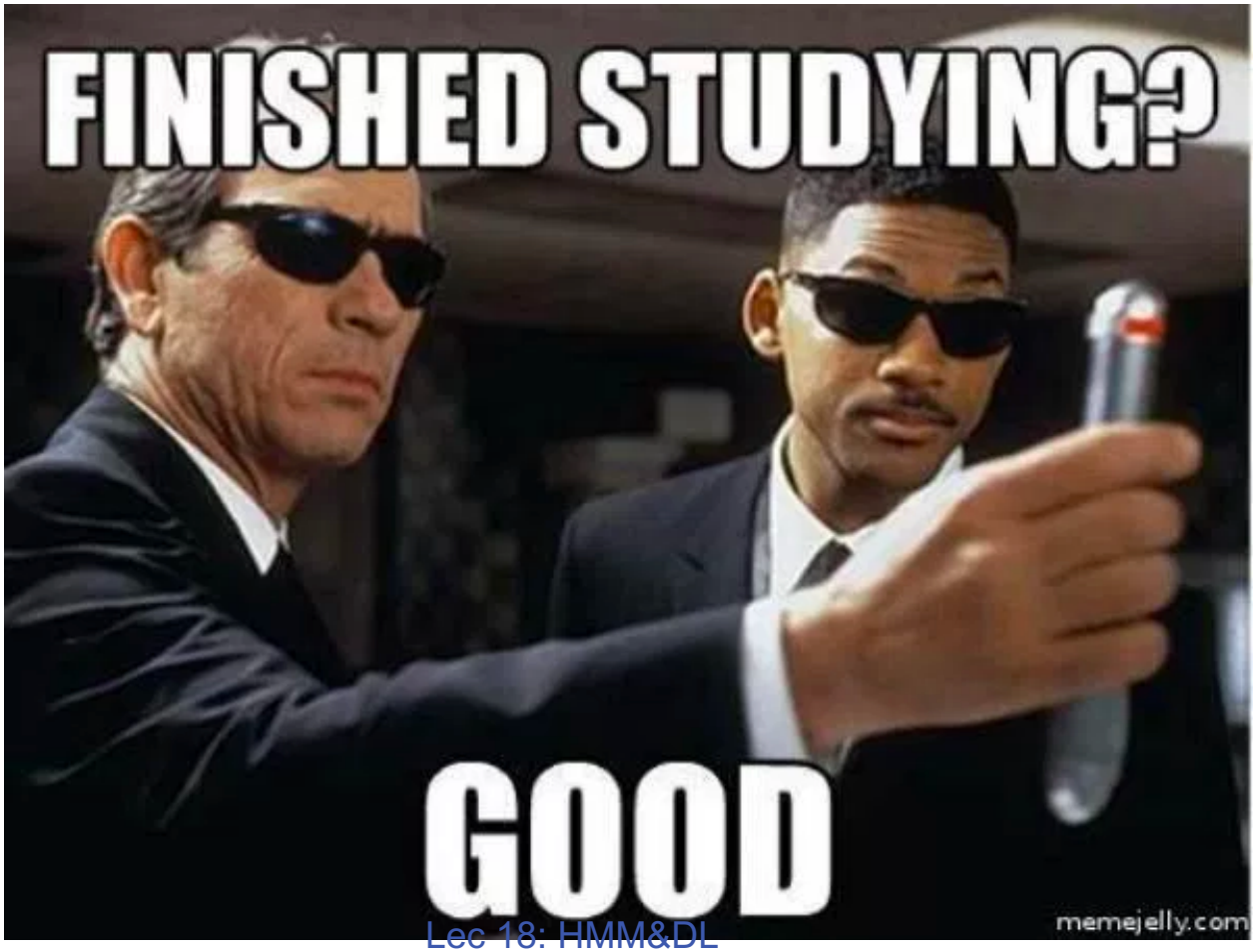
CS @ UCLA

kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

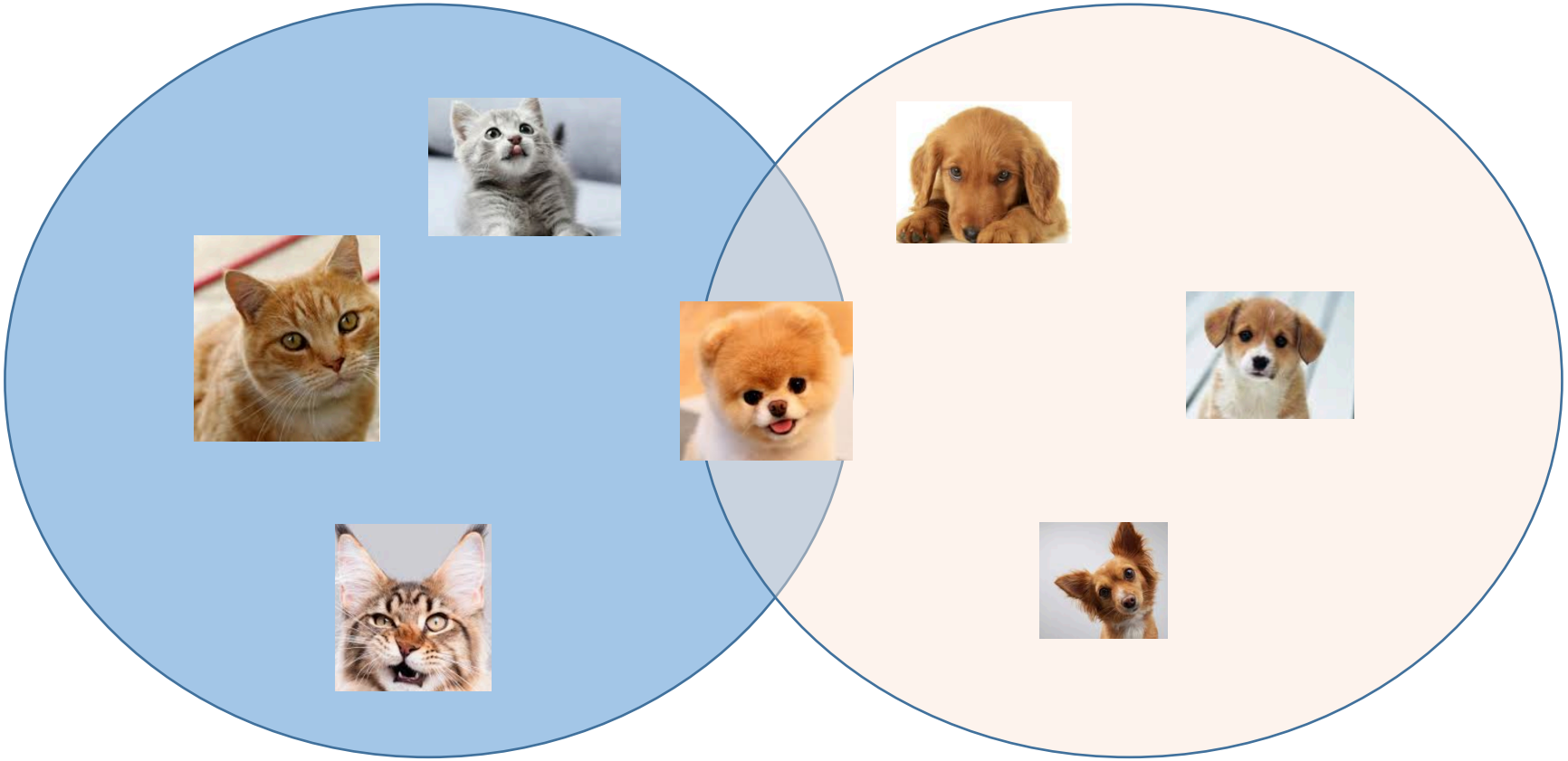
Final Exam 3/22, 11:30am-2:00pm

- Franz Hall: 1178 , 150min exam



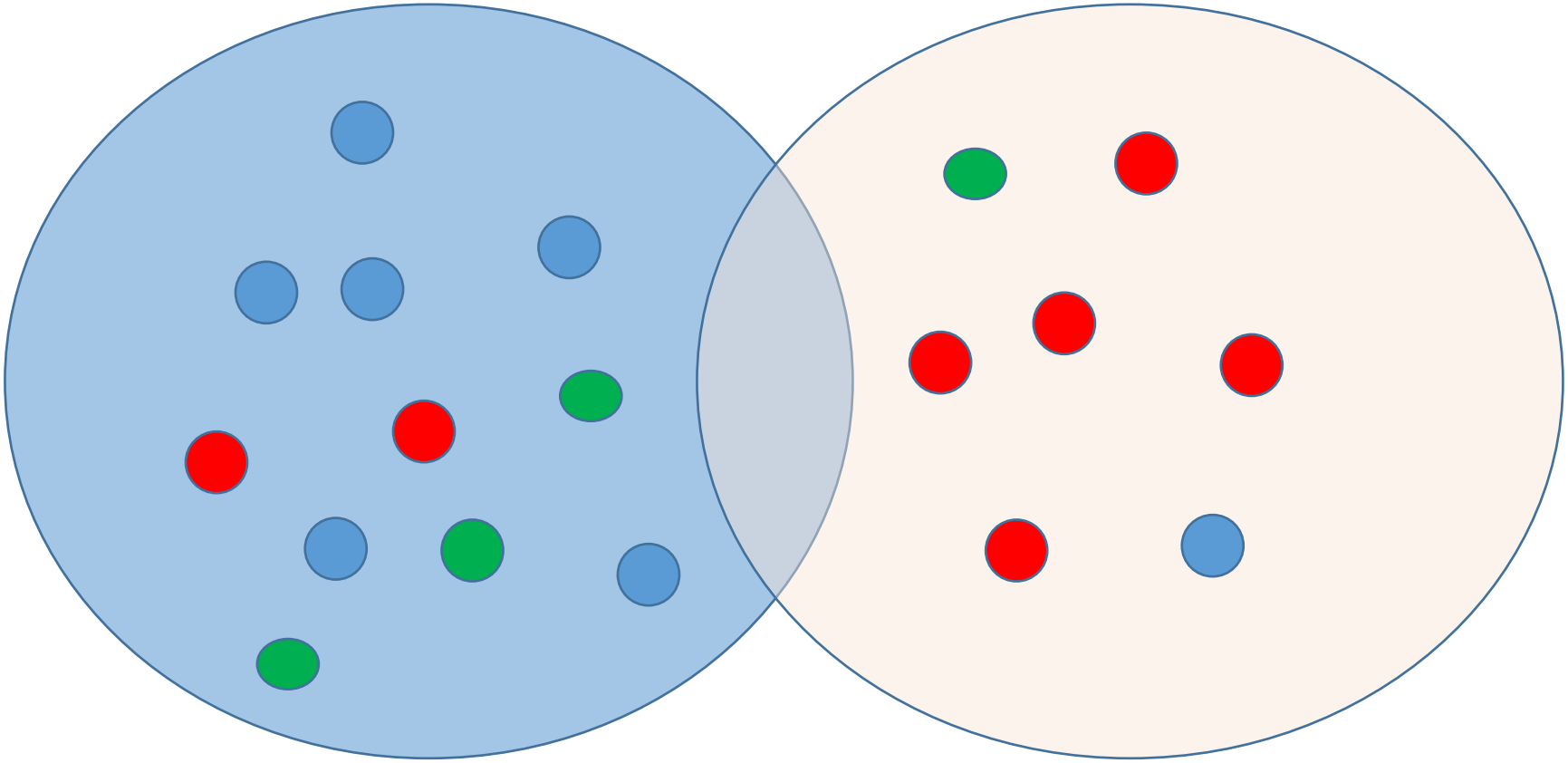
Generative model

- Idea: build a probabilistic model to measure how likely we would see the observations



Generative model

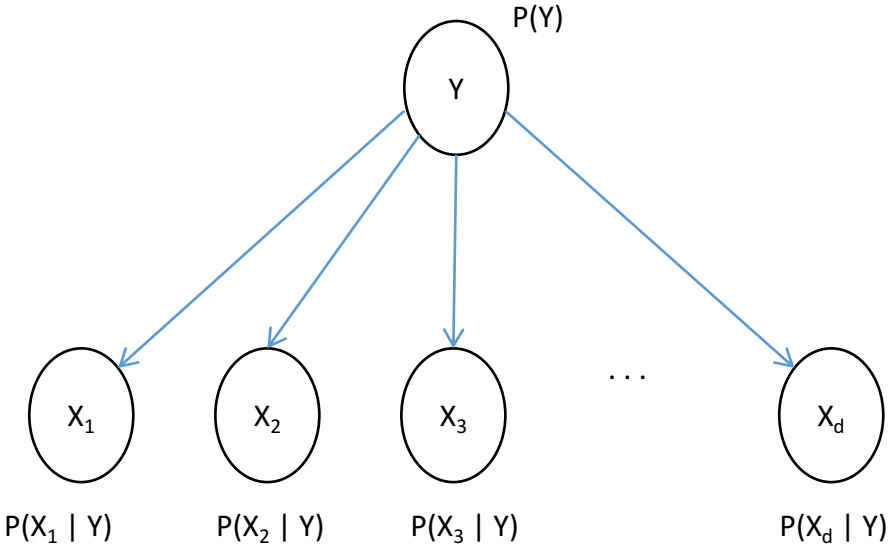
- Idea: build a probabilistic model to measure how likely we would see the observations



Prediction under a generative Model

Given the label, sample the features independently from the conditional distributions

Cat? or Dog?



$$\underset{y}{\operatorname{argmax}} P(y) \prod_j P(x_j|y)$$

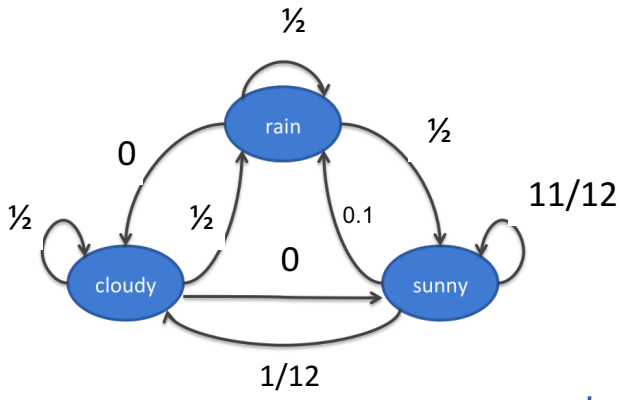
Weather Forecast by Markov Chain

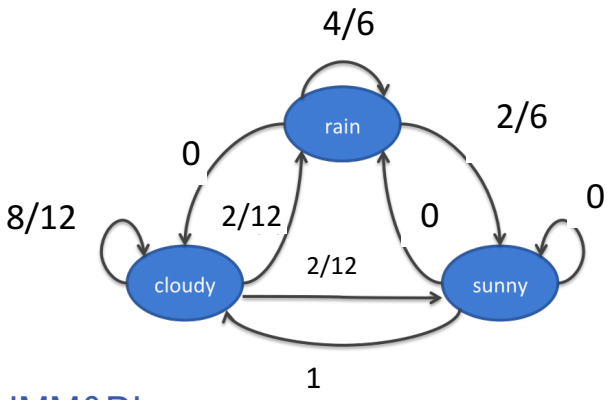
- Observing a sequence of weather events, predict the weather tomorrow

S: Sunny, C: Cloudy, R: rainy

Los Angeles (winter): S S S S S S S S C C R R S S S S S S S S

Seattle (winter): C C C C C C R R C S C C S C R R R R C C C





Discrete Markov Process

- State at time t is x_t
- First-order Markov assumption

  The state of the system at any time is ***independent*** of the full sequence history given the previous state

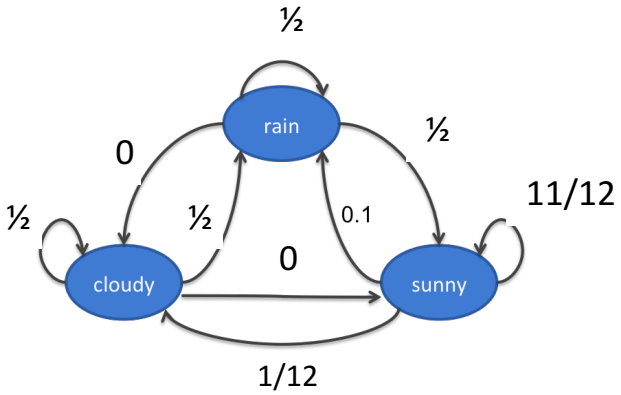
$$P(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} P(x_i | x_{i-1})$$

  - Defined by two sets of probabilities:
    - Initial state distribution: $P(x_1 = S_j)$
    - State transition probabilities: $P(x_i = S_j \mid x_{i-1} = S_k)$

Example: The weather

- Three states: rain, cloudy, sunny

State transitions:



- Observations are Markov chains:

Eg: *cloudy sunny sunny rain*

Probability of the sequence =

P(cloudy) P(sunny|cloudy) P(sunny | sunny) P(rain | sunny)

Initial probability

Transition probabilities

Weather Forecast by Markov Chain

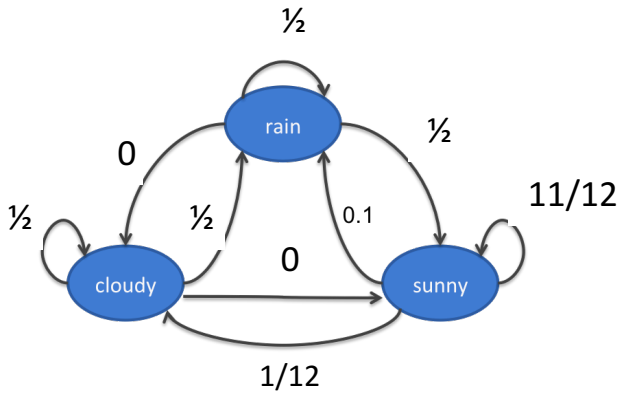
- Maximum Likelihood estimation:

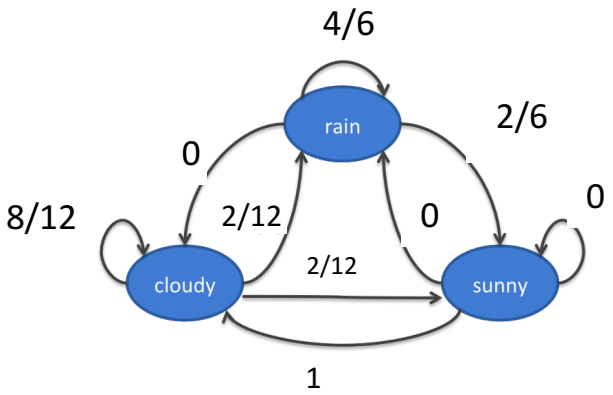
$$P(y_t = \text{Rain} \mid y_{t-1} = \text{Sunny}) = \frac{\#\, S\, R}{\#\, S}$$

S: Sunny, C: Cloudy, R: rainy

Los Angeles (winter): S S S S S S S S C C R R S S S S S S S S S

Seattle (winter): C C C C C C R R C S C C S C R R R R C C C





Example: Another language model

It was a bright cold day in April

$$P(\text{It was a bright cold day in April}) =$$
$$P(\text{It}) \times$$
$$P(\text{was}|\text{It}) \times$$
$$P(\text{a}|\text{was}) \times$$
$$P(\text{bright}|\text{a}) \times$$
$$P(\text{cold}|\text{bright}) \times$$
$$P(\text{day}|\text{cold}) \times \cdots$$

Probability of a word starting a sentence

Probability of a word following "It"

Probability of a word following "was"

Probability of a word following "a"

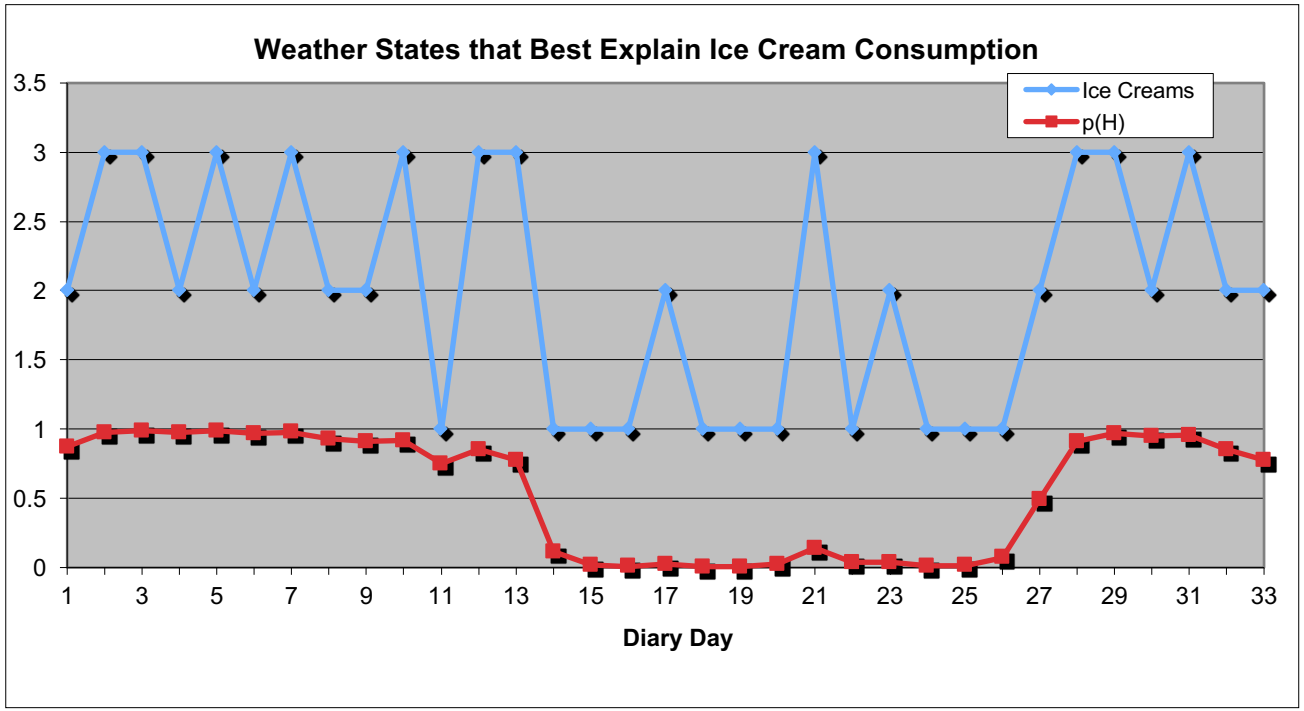
If there are K tokens/states, how many parameters do we need? $O(K^2)$

Hidden Markov Model

(C)old day v.s. (H)ot day

#cones

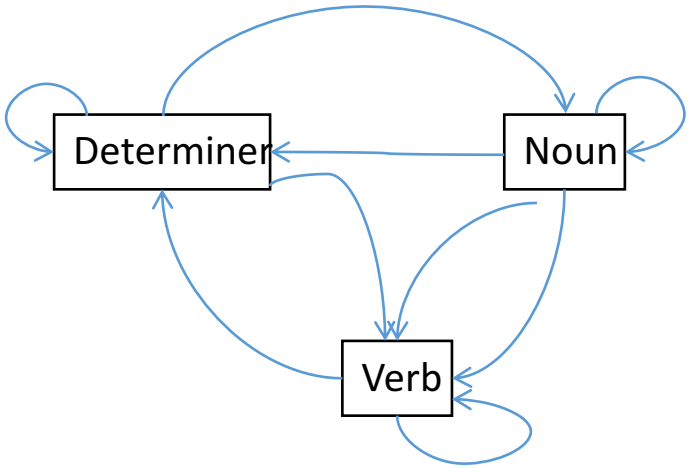
	p(…|C)	p(…|H)	p(…|START)
(1|…)	0.7	0.1	
(2|…)	0.2	0.2	
(3|…)	0.1	0.7	
(C|…)	0.8	0.1	0.5
(H|…)	0.1	0.8	0.5
…)	0.1	0.1	0



Toy part-of-speech example

The Fed raises interest rates

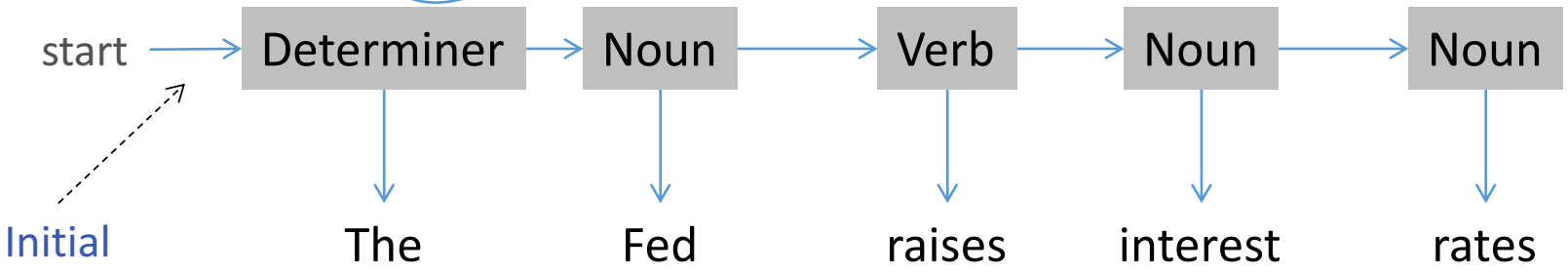
Transitions



Emissions

P(The | Determiner) = 0.5
P(A | Determiner) = 0.3
P(An | Determiner) = 0.1
P(Fed | Determiner) = 0
…

P(Fed| Noun) = 0.001
P(raises| Noun) = 0.04
P(interest| Noun) = 0.07
P(The| Noun) = 0
…



Joint model over states and observations

- Notation
  - Number of states = K, Number of observations = M
  - π: Initial probability over states (K dimensional vector)
  - A: Transition probabilities (K×K matrix)
  - B: Emission probabilities (K×M matrix)

- Probability of states and observations
  - Denote states by $y_1, y_2, \cdots$ and observations by $x_1, x_2, \cdots$

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$= \pi_{y_1} \prod_{i=1}^{n-1} A_{y_i, y_{i+1}} \prod_{i=1}^{n} B_{y_i, x_i}$$

Key questions for HMMs

[Rabiner 1999]

1. Given an observation sequence, $x_1, x_2, \cdots, x_n$ and a model (π, A, B), how to efficiently calculate the most probable state sequence?

   Inference

2. How to calculate (π, A, B) from observations?

   Learning

Outline

- *Sequence models*
- Hidden Markov models
  - Inference with HMM
  - Supervised Learning for HMM

Most likely state sequence

- Input:
  - A hidden Markov model (π, A, B)
  - An observation sequence $\mathbf{x} = (x_1, x_2, \cdots, x_n)$

- Output: A state sequence $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ that corresponds to
  - Maximum *a posteriori* inference (MAP inference)

$$\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$$

- Computationally: combinatorial optimization

MAP inference

- We want $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B)$
- We have defined

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

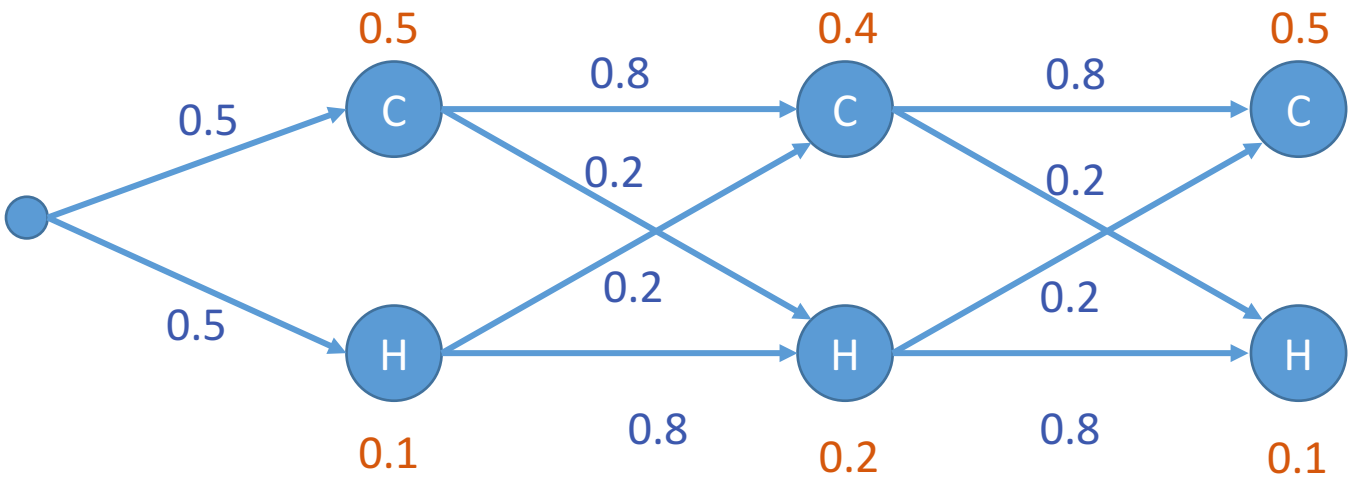
- But $P(\mathbf{y}|\mathbf{x}, \pi, A, B) \propto P(\mathbf{x}, \mathbf{y}|\pi, A, B)$
  - And we don't care about P(**x**) we are maximizing over **y**
- So, $\arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}, \pi, A, B) = \arg\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B)$

Jason's ice cream

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	**0.5**	**0.1**	
(2|…)	**0.4**	**0.2**	
(3|…)	**0.1**	**0.7**	
(C|…)	**0.8**	**0.2**	**0.5**
(H|…)	**0.2**	**0.8**	**0.5**

- Best tag sequence for P("1,2,1")?

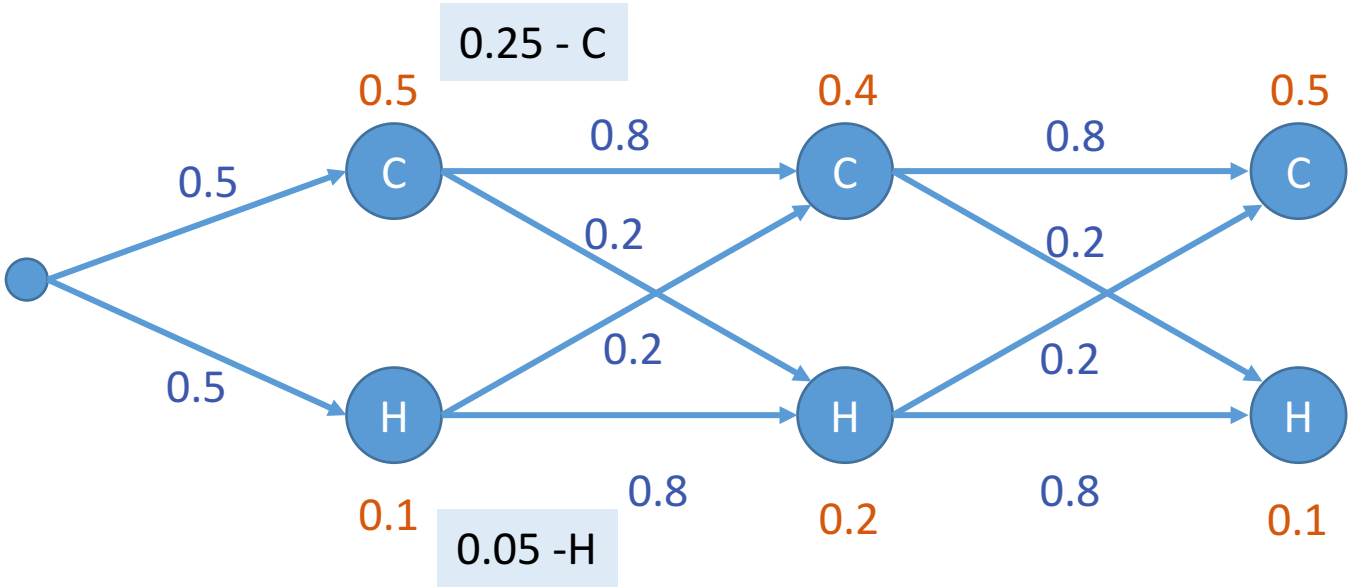


Jason's ice cream

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	**0.5**	**0.1**	
(2|…)	**0.4**	**0.2**	
(3|…)	**0.1**	**0.7**	
(C|…)	**0.8**	**0.2**	**0.5**
(H|…)	**0.2**	**0.8**	**0.5**

- Best tag sequence for P("1,2,1")?

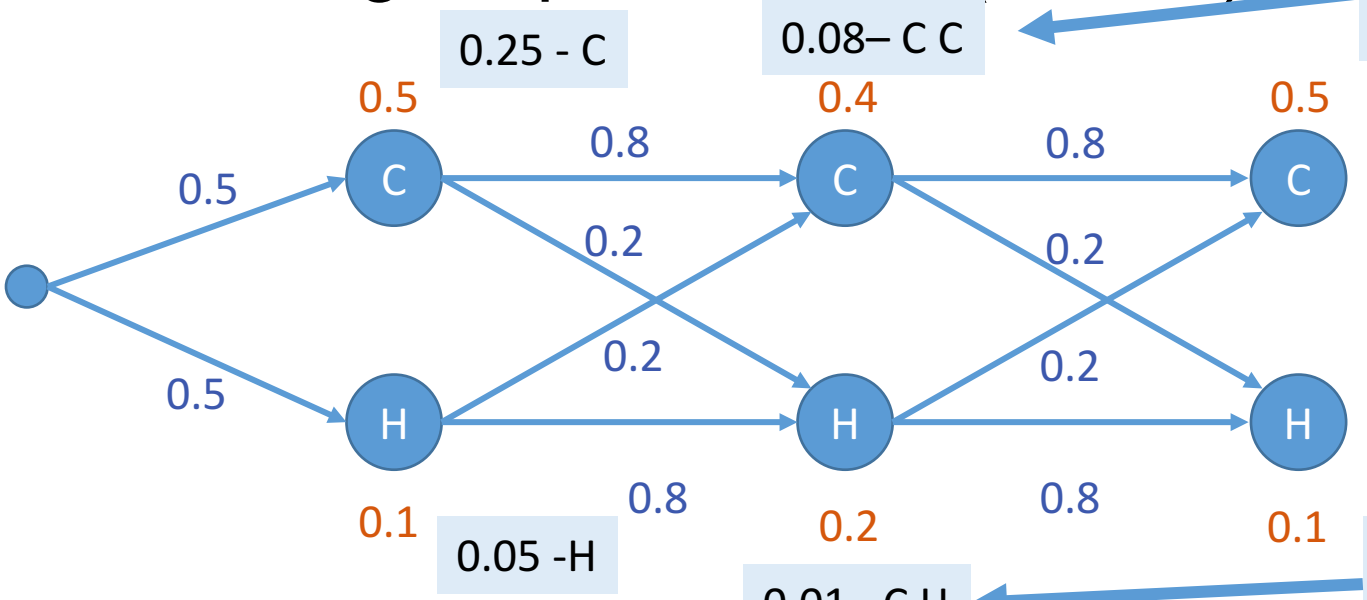


Jason's ice cream

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	**0.5**	**0.1**	
(2|…)	**0.4**	**0.2**	
(3|…)	**0.1**	**0.7**	
(C|…)	**0.8**	**0.2**	**0.5**
(H|…)	**0.2**	**0.8**	**0.5**

- Best tag sequence for P("1,2,1")?

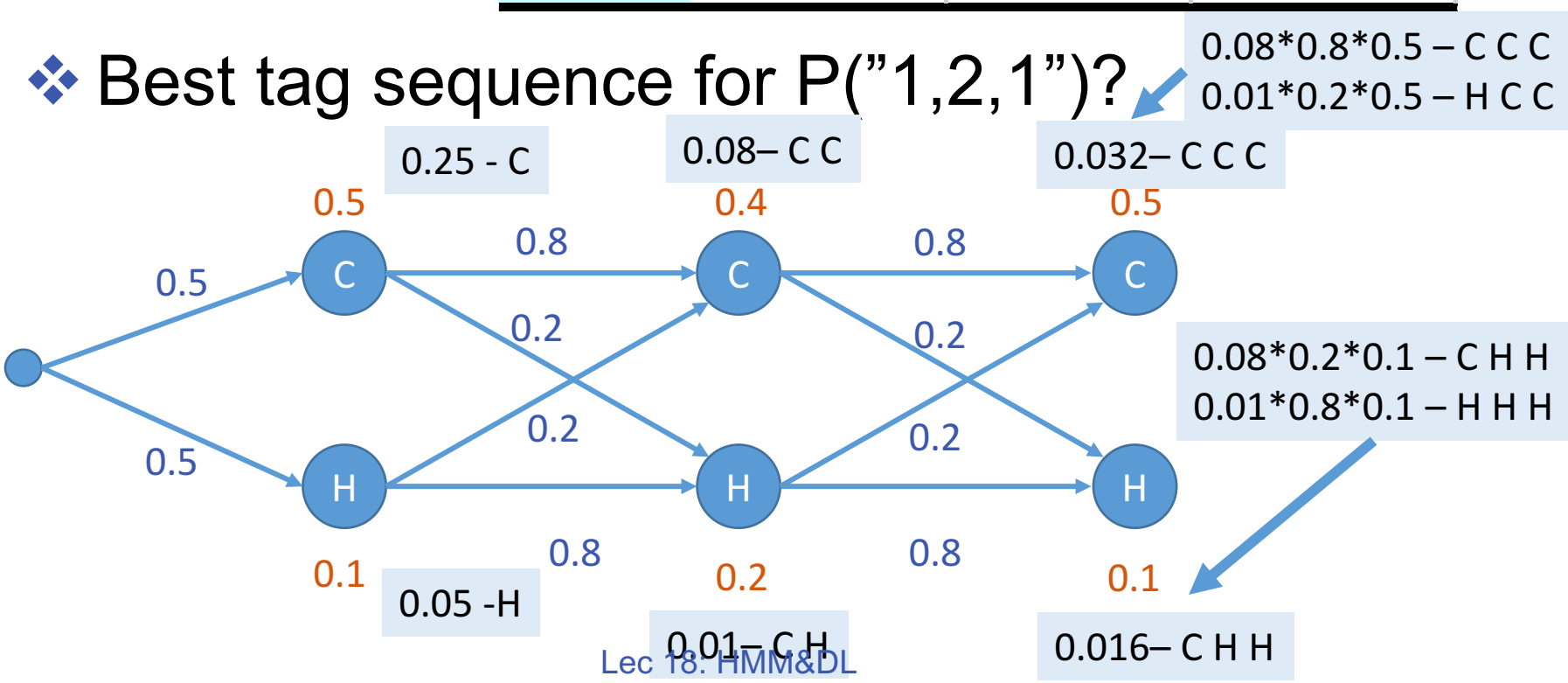


Jason's ice cream

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	**0.5**	**0.1**	
(2|…)	**0.4**	**0.2**	
(3|…)	**0.1**	**0.7**	
(C|…)	**0.8**	**0.2**	**0.5**
(H|…)	**0.2**	**0.8**	**0.5**

- Best tag sequence for P("1,2,1")?

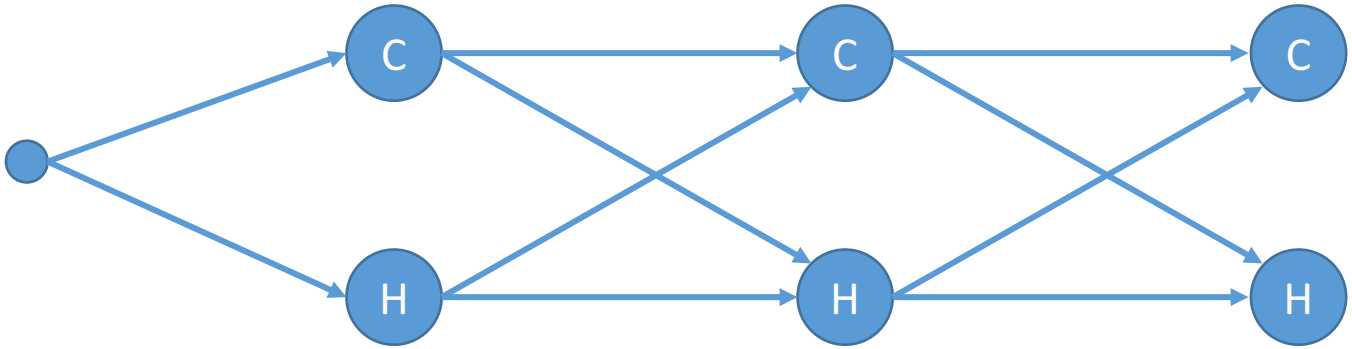


Your turn

#cones

	p(…|C)	p(…|H)	p(…|START)
(1|…)	**0.5**	**0.1**	
(2|…)	**0.4**	**0.2**	
(3|…)	**0.1**	**0.7**	
(C|…)	**0.8**	**0.2**	**0.5**
(H|…)	**0.2**	**0.8**	**0.5**

- Best tag sequence for P(”3,3,2”)?



Viterbi algorithm

π: Initial probabilities
A: Transitions
B: Emissions

Max-product algorithm for first order sequences

1. Initial: For each state s, calculate

$$\text{score}_1(s) = P(s)P(x_1|s) = \pi_s B_{x_1,s}$$

1. Recurrence: For i = 2 to n, for every state s, calculate

$$\begin{aligned}\text{score}_i(s) &= \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1}) \\ &= \max_{y_{i-1}} A_{y_{i-1},s}B_{s,x_i}\text{score}_{i-1}(y_{i-1})\end{aligned}$$

1. Final state: calculate

$$\max_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}|\pi, A, B) = \max_s \text{score}_n(s)$$

This only calculates the max. To get final answer (*argmax*),

- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

Questions?

General idea

- Dynamic programming
  - The best solution for the full problem relies on best solution to sub-problems
  - Memoize partial computation

- Examples
  - Viterbi algorithm
  - Dijkstra's shortest path algorithm
  - …

Complexity of inference

- Complexity parameters
  - Input sequence length: n
  - Number of states: K
- Memory
  - Storing the table: nK (scores for all states at each position)
- Runtime
  - At each step, go over pairs of states
  - $O(nK^2)$

Outline

- *Sequence models*
- Hidden Markov models
  - Inference with HMM
  - Supervised Learning for HMM

Learning HMM parameters

- Assume we know the number of states in the HMM
- Two possible scenarios
  1. We are given a data set D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>} of sequences labeled with states

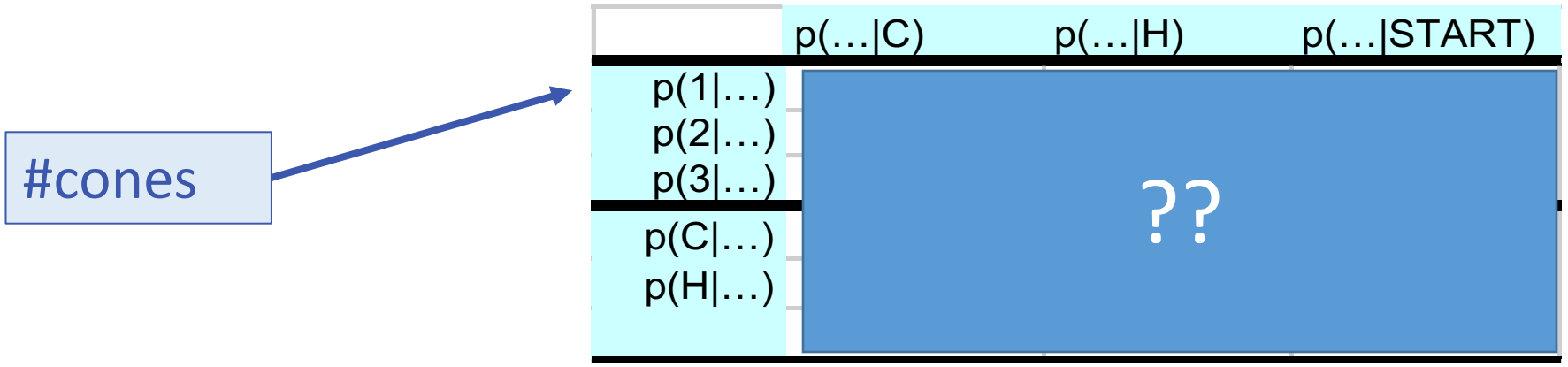
     And we have to learn the parameters of the HMM (π, A, B)

     Supervised learning with complete data
  2. We are given only a collection of sequences D = {$\mathbf{x}_i$}

     And we have to learn the parameters of the HMM (π, A, B)

     Unsupervised learning, with incomplete data

Jason's ice cream



- Can we figure out (π, A, B) from history records?

#cones	1	2	1	1	2	3	3	3	3	2	3	1	2
Hot/Cold	C	C	C	C	H	H	H	H	C	C	H	H	H

#cones	3	2	2	2	2	3	1	1	1	2	3	2	2
Hot/Cold	H	H	H	C	C	H	H	C	C	H	H	H	C

Supervised learning of HMM

- We are given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>}
  - Each $\mathbf{x}_i$ is a sequence of observations and $\mathbf{y}_i$ is a sequence of states that correspond to $\mathbf{x}_i$

  Goal: Learn initial, transition, emission distributions (π, A, B)

- How do we learn the parameters of the probability distribution?
  - The maximum likelihood principle

Where have we seen this before?

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

And we know how to write this in terms of the parameters of the HMM

Supervised learning details

$$(\hat{\pi}, \hat{A}, \hat{B}) = \max_{\pi,A,B} P(D|\pi, A, B) = \max_{\pi,A,B} \prod_i P(\mathbf{x}_i, \mathbf{y}_i|\pi, A, B)$$

π, A, B can be estimated separately just by counting

- Makes learning simple and fast

[Exercise: Derive the following using derivatives of the log likelihood.]

Number of instances where the first state is s

$$\pi_s = \frac{\text{count}(\text{start} \rightarrow s)}{n}$$

Number of examples

Initial probabilities

$$A_{s',s} = \frac{\text{count}\,(s \rightarrow s')}{\text{count}\,(s)}$$

Transition probabilities

$$B_{s,x} = \frac{\text{count}\begin{pmatrix} s \\ \downarrow \\ x \end{pmatrix}}{\text{count}\,(s)}$$

Emission probabilities

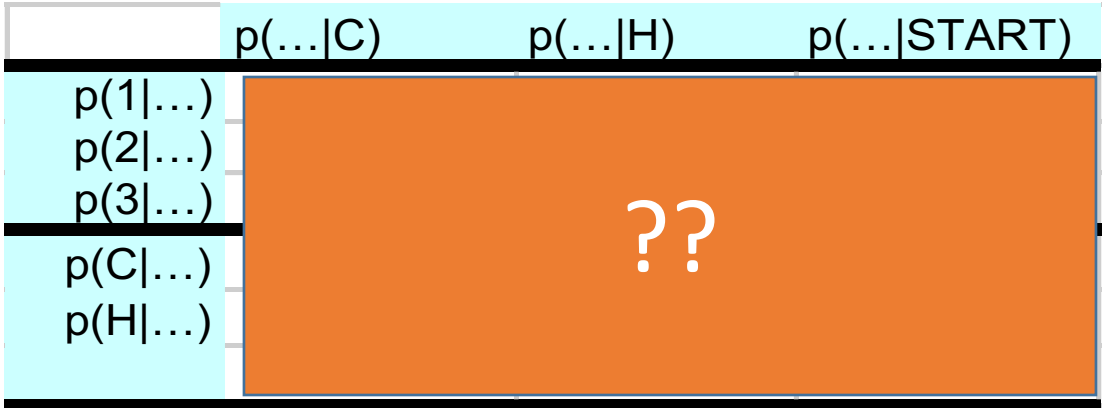
Priors and smoothing

- Maximum likelihood estimation works best with lots of annotated data
  - Never the case
- Priors inject information about the probability distributions
- Effectively additive smoothing
  - Add small constants to the counts

Question we will not answer in this lecture

- Can we figure out (π, A, B) from just observation

#cones



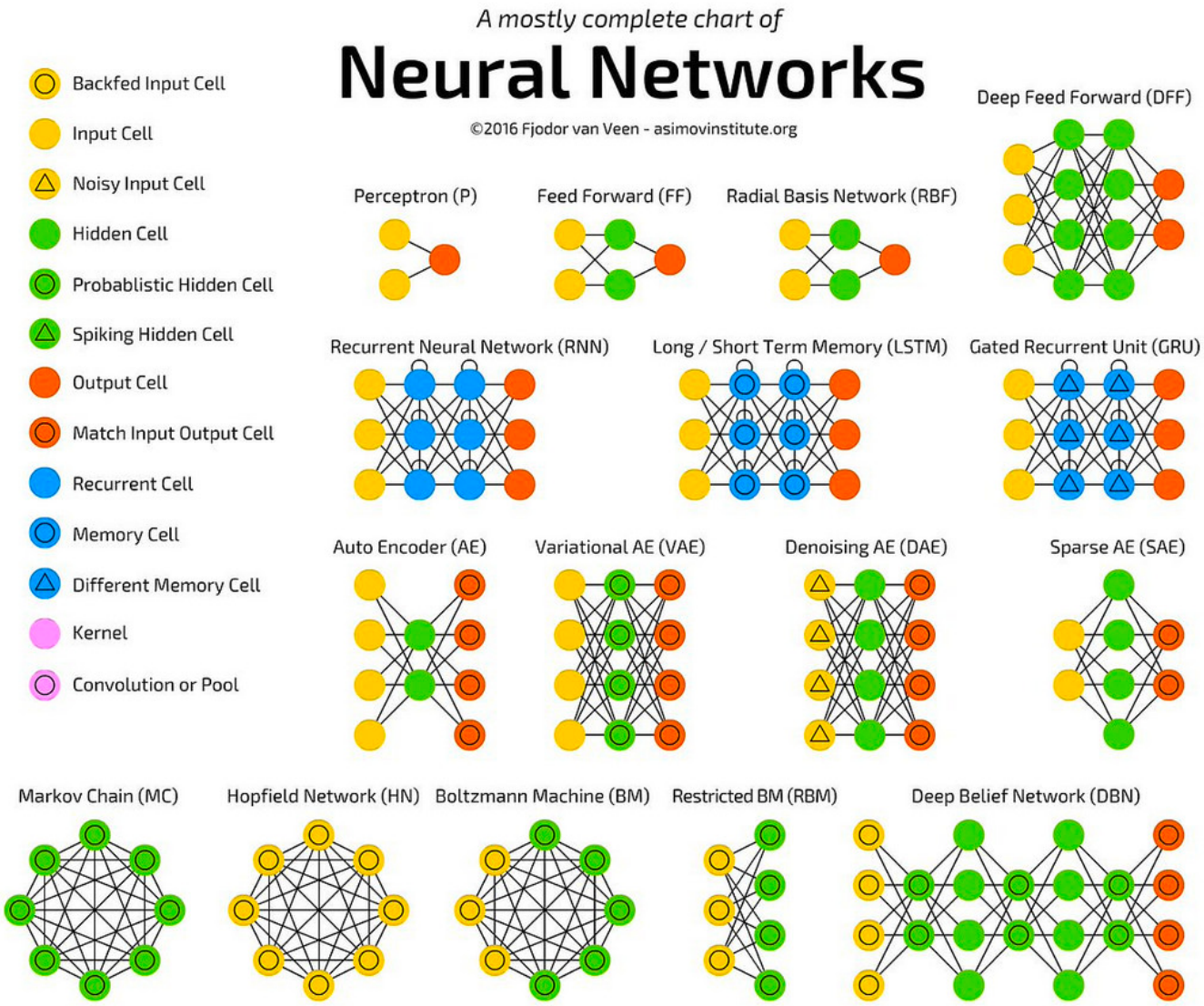
#cones	1	2	1	1	2	3	3	3	3	2	3	1	2
Hot/Cold	??												

#cones	3	2	2	2	2	3	1	1	1	2	3	2	2
Hot/Cold	??												

Hidden Markov Models summary

- Predicting sequences
  - As many output states as observations
- Markov assumption helps decompose the score
- Several algorithmic questions
  - Most likely state (decoding)
  - Learning parameters
    - Supervised, Unsupervised

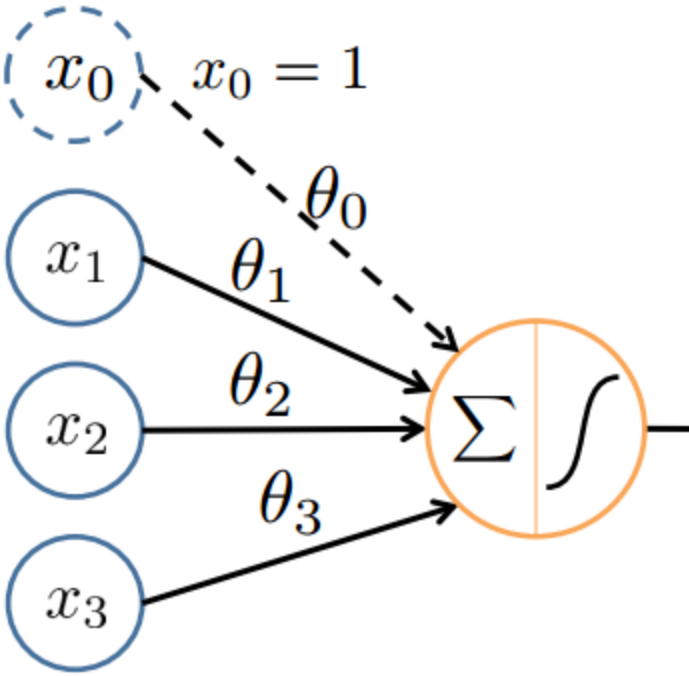
Neural Network



This lecture

- Neural Network
- Recurrent Neural Network

Neural Network

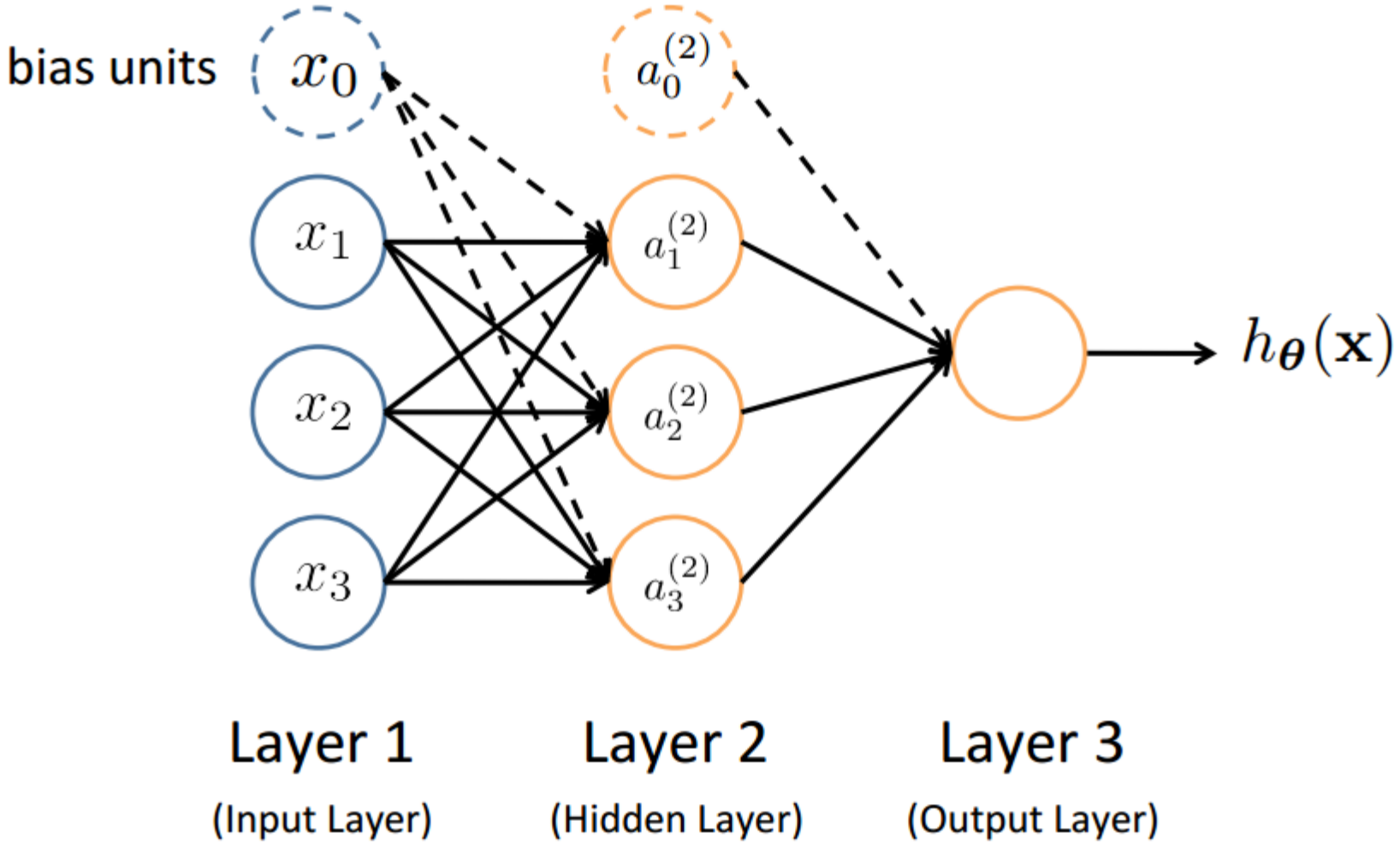


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Neural Network (feed forward)



Slide by Andrew Ng

Feed-Forward Process

- Input layer units are features (e.g., words)
  - Usually, one-hot vector or word embedding
- Working forward through the network, the **input function** is applied to compute the input value
  - E.g., weighted sum of the input
- The **activation function** transforms this input function into a final value
  - Typically a **nonlinear** function (e.g, **sigmoid**)

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Activation functions

Also called transfer functions

$$output = activation(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Name of the neuron	Activation function: $activation(z)$
Linear unit	z
Threshold/sign unit	$sign(z)$
Sigmoid unit	$\frac{1}{1+\exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

Many more activation functions exist (sinusoid, sinc, gaussian, polynomial…)

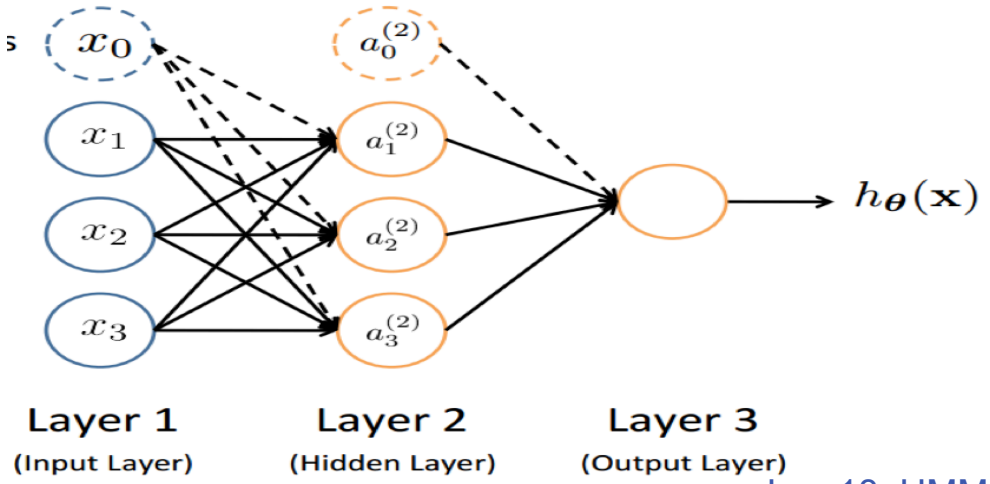
Vector Representation

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

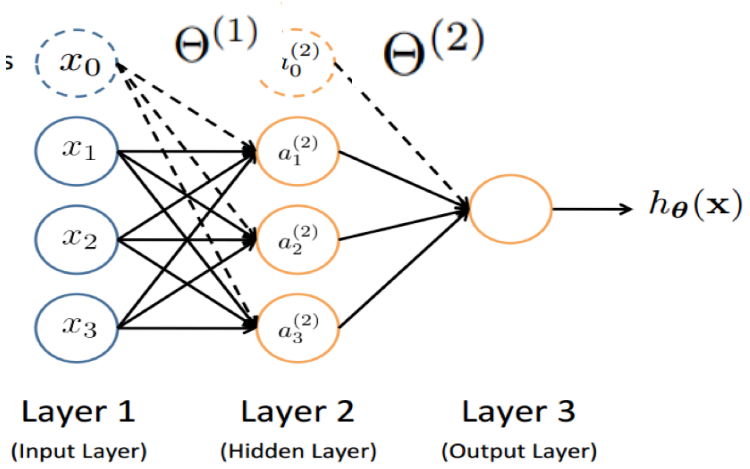
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Based on slide by Andrew Ng



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

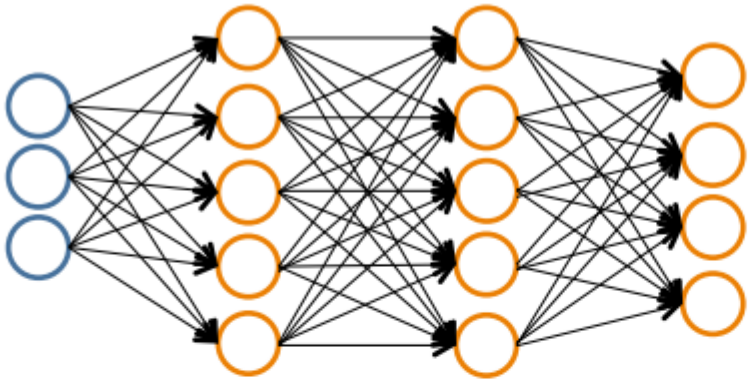
Can extend to multi-class

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

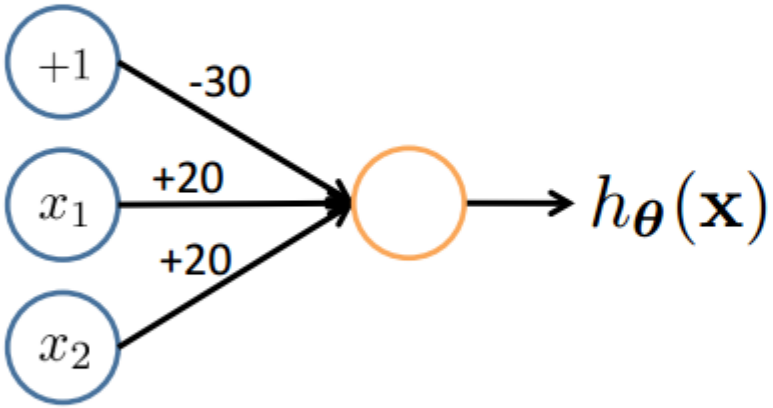
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Why staged predictions?

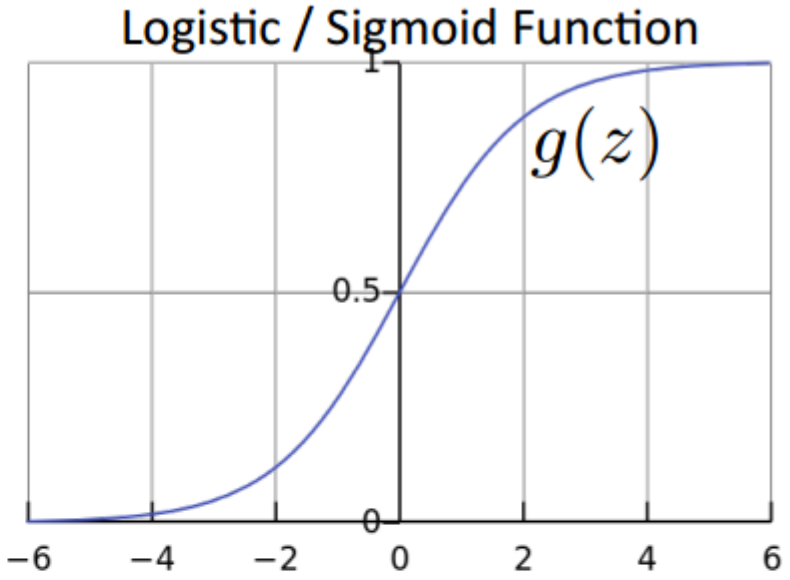
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1$ AND x_2

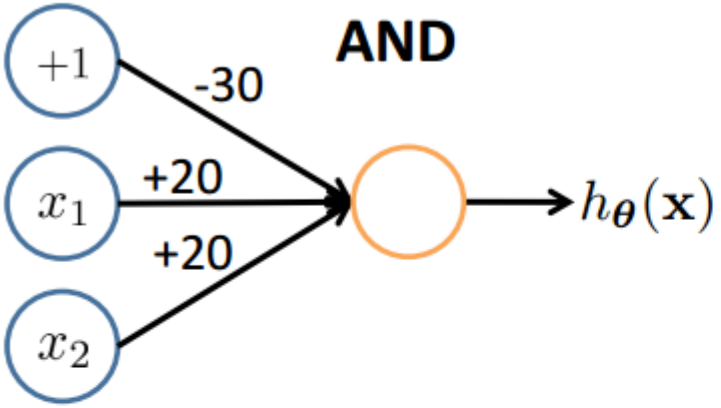


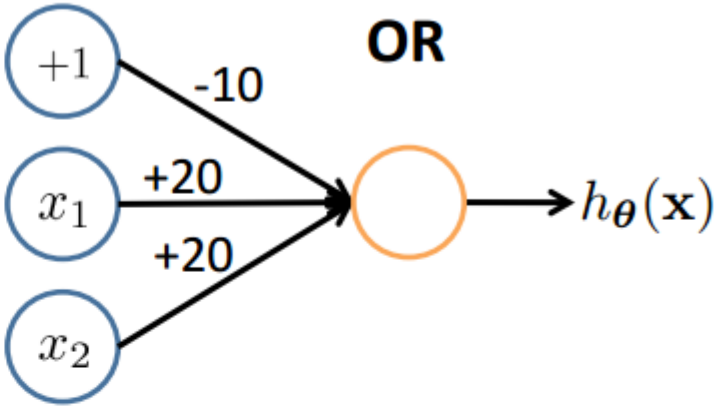
$\mathrm{h}_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$

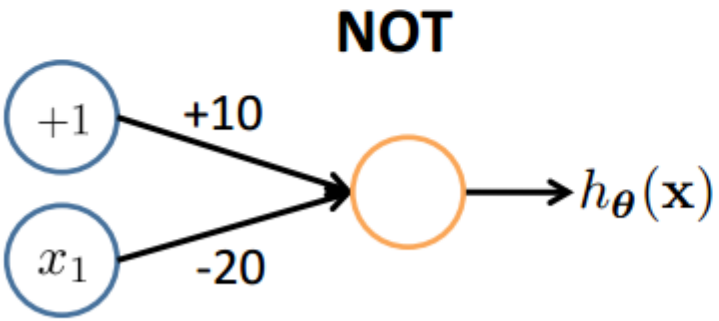


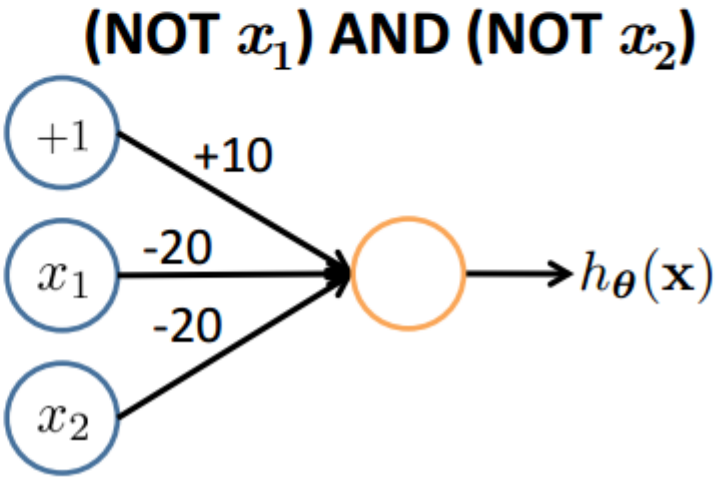
x_1	x_2	$\mathrm{h}_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Based on slide and example by Andrew Ng

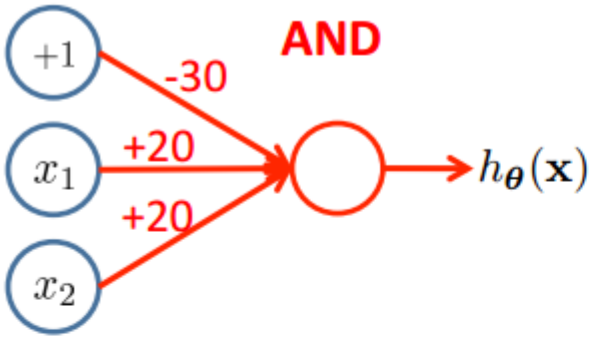

AND
+1
x_1
x_2
-30
+20
+20
$h_\theta(\mathbf{x})$

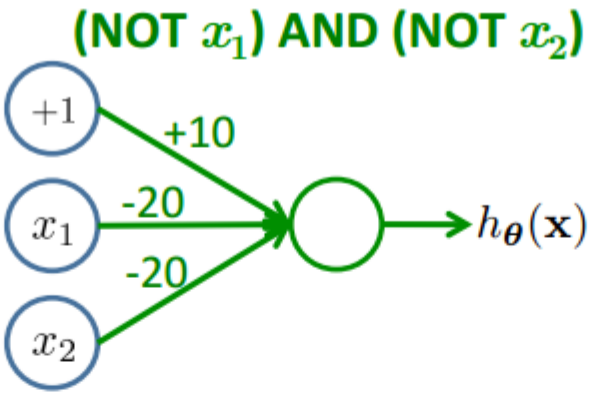

OR
+1
x_1
x_2
-10
+20
+20
$h_\theta(\mathbf{x})$

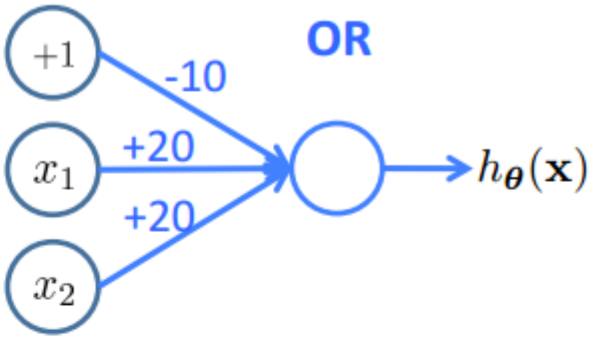

NOT
+1
x_1
+10
-20
$h_\theta(\mathbf{x})$

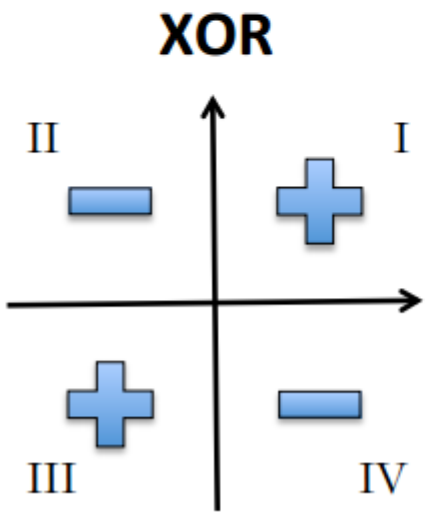

(NOT x_1) AND (NOT x_2)
+1
x_1
x_2
+10
-20
-20
$h_\theta(\mathbf{x})$

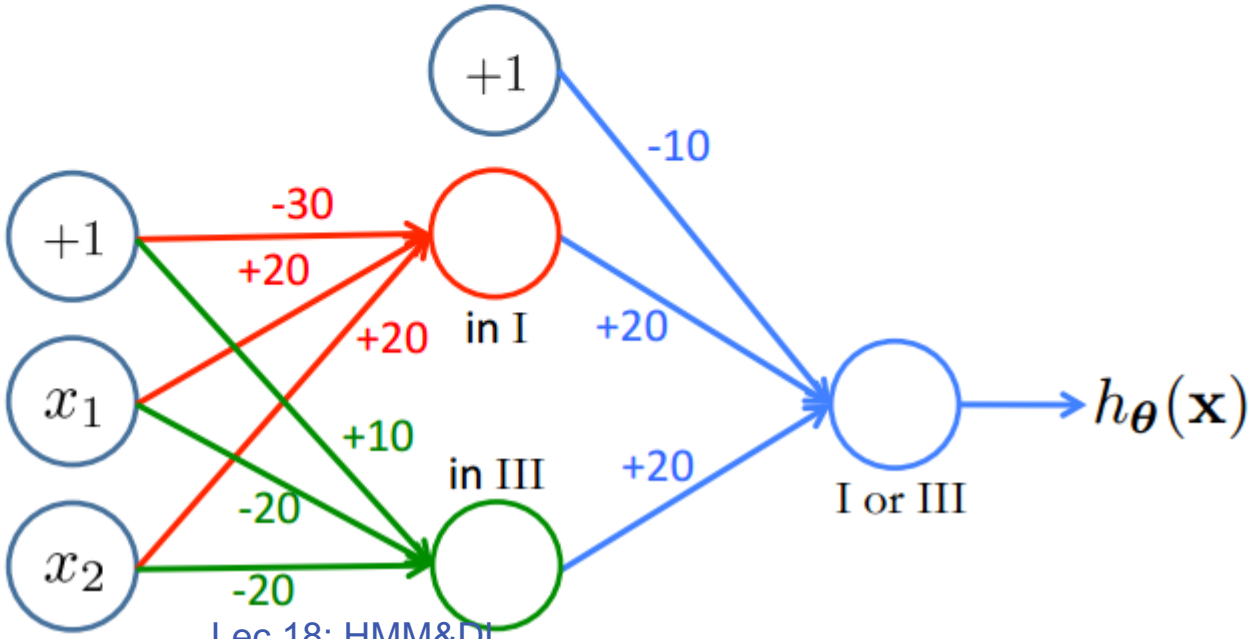
Combining Representations to Create Non-Linear Functions







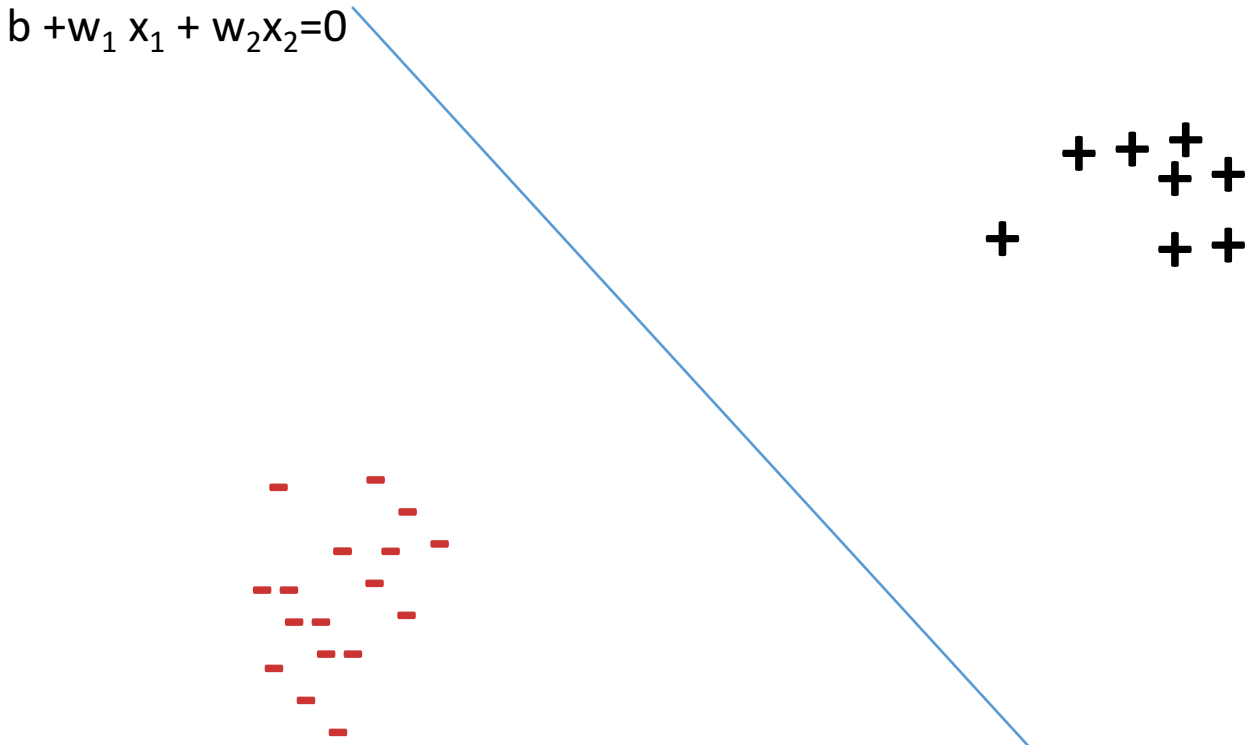




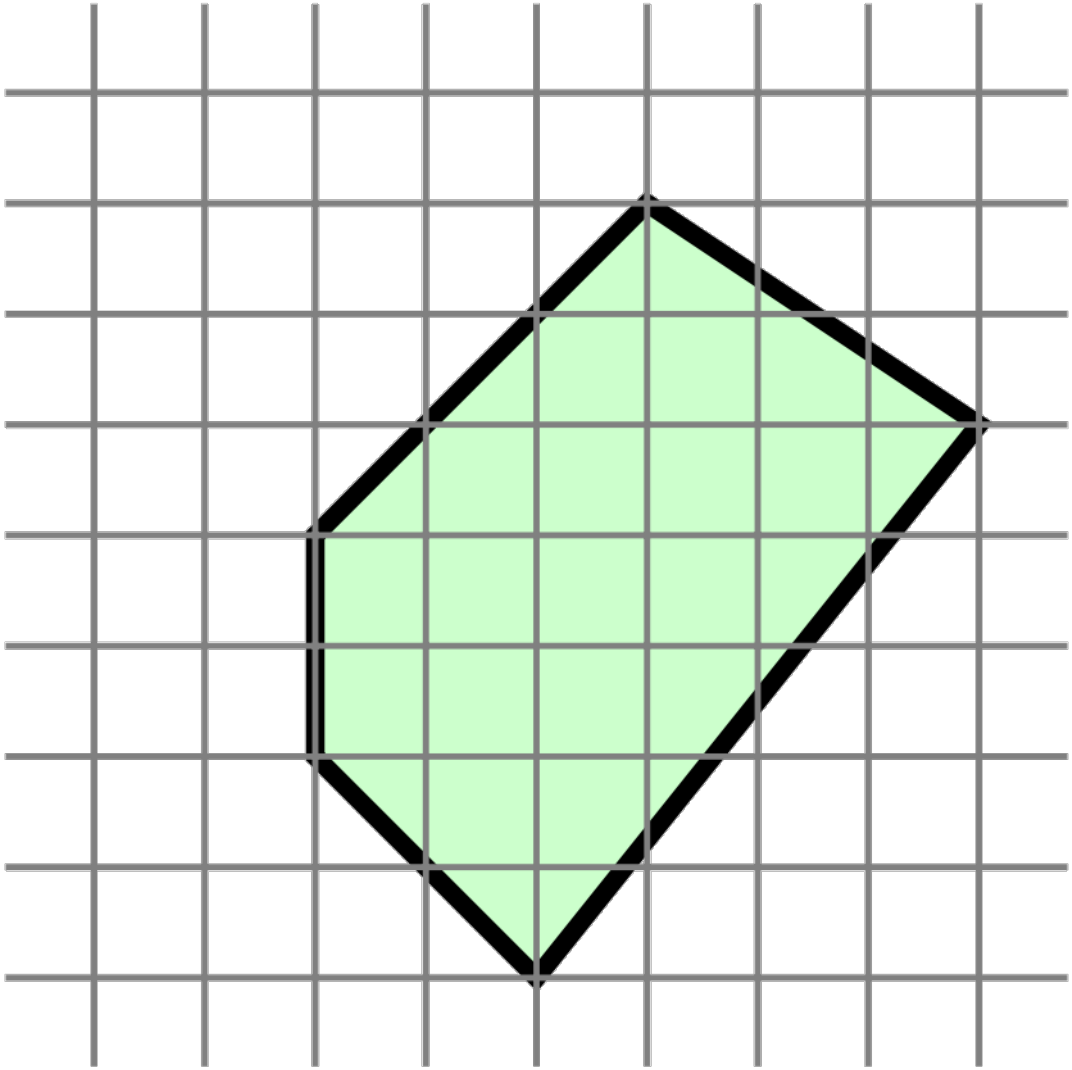
Based on example by Andrew Ng

A single neuron with threshold activation

Prediction = *sgn*$(b + w_1 x_1 + w_2 x_2)$



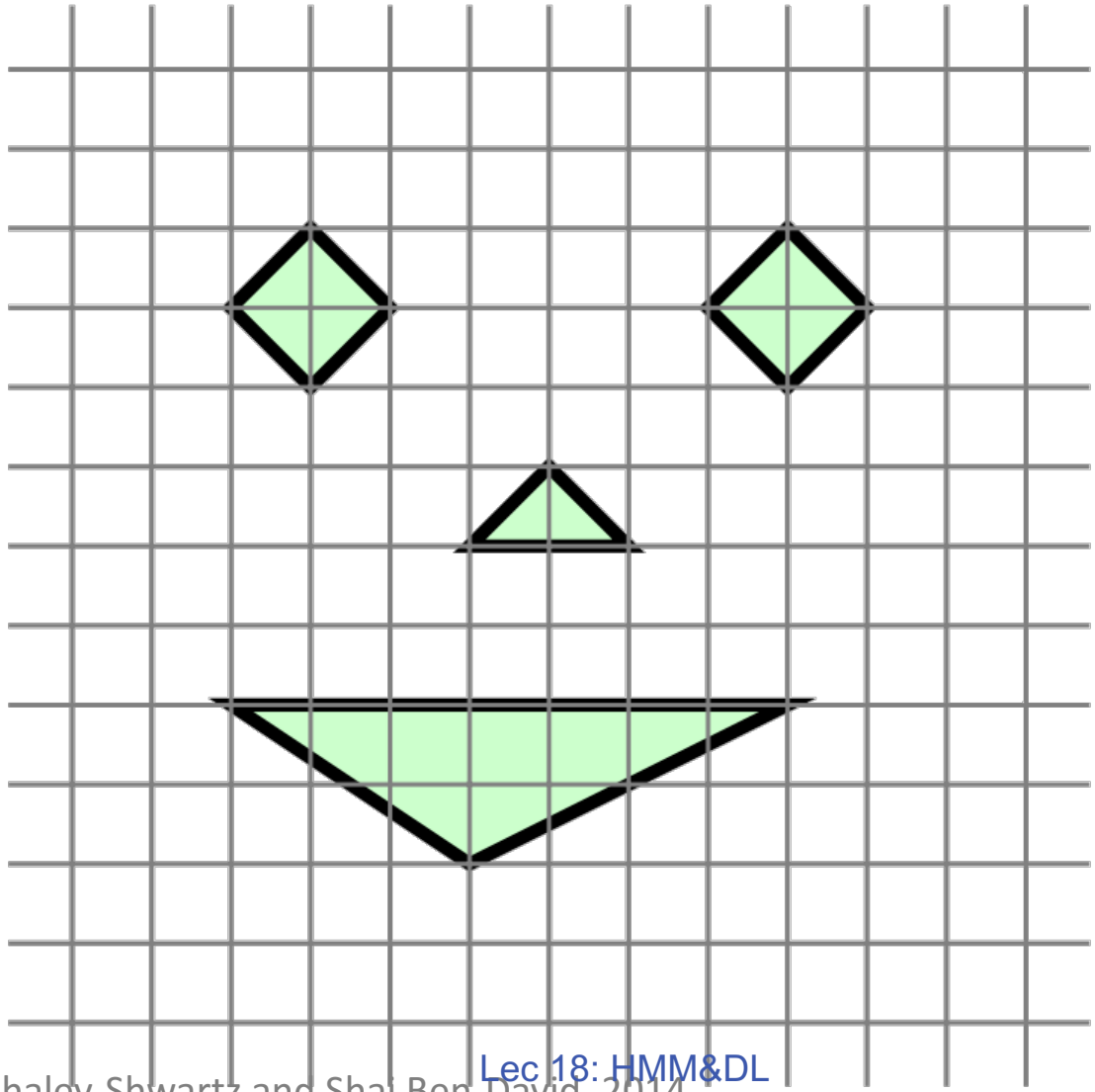
Two layers, with threshold activations



In general, convex polygons

Figure from Shai Shalev-Shwartz and Shai Ben-David, 2014

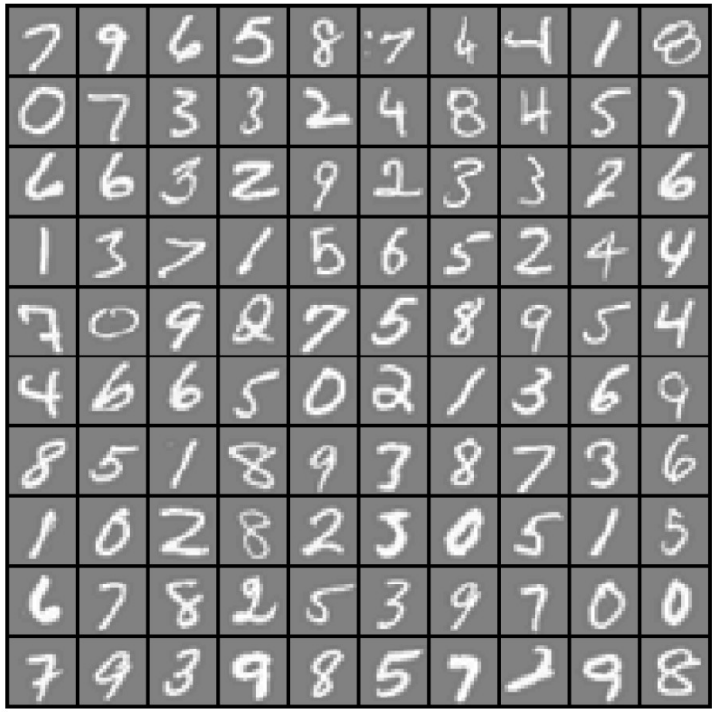
Three layers with threshold activations

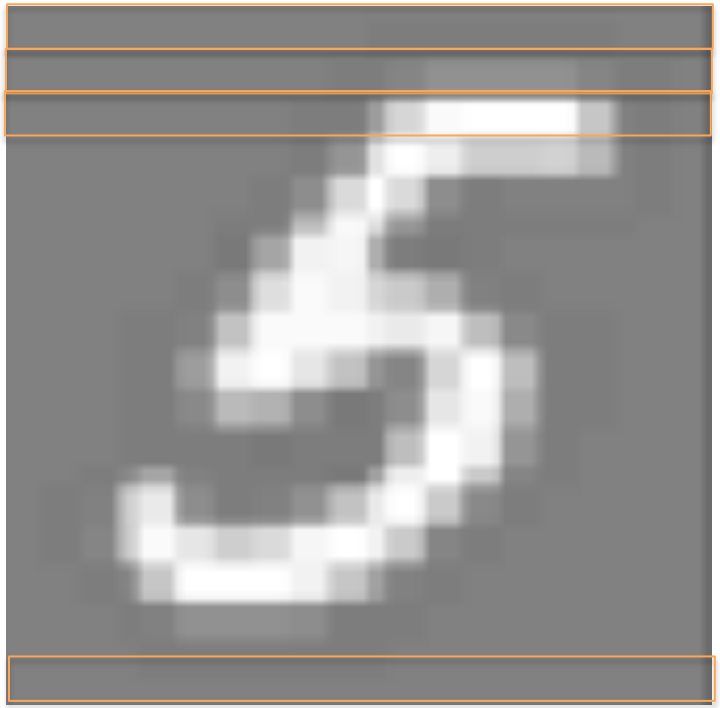


In general, unions of convex polygons

Figure from Shai Shalev-Shwartz and Shai Ben-David, 2014

Layering Representations





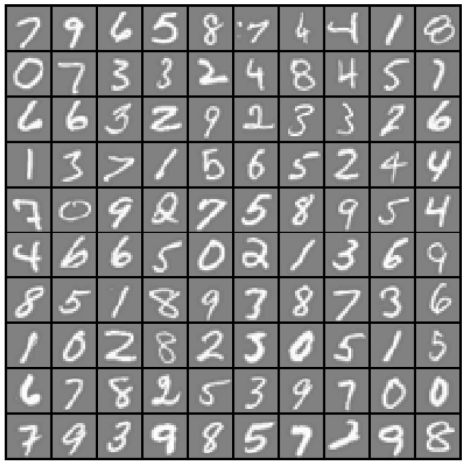
$x_1 \dots x_{20}$
$x_{21} \dots x_{40}$
$x_{41} \dots x_{60}$

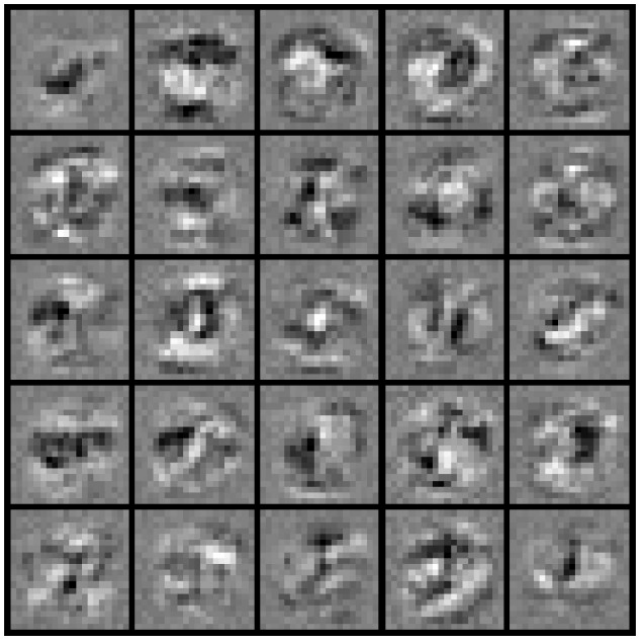
$x_{381} \dots x_{400}$

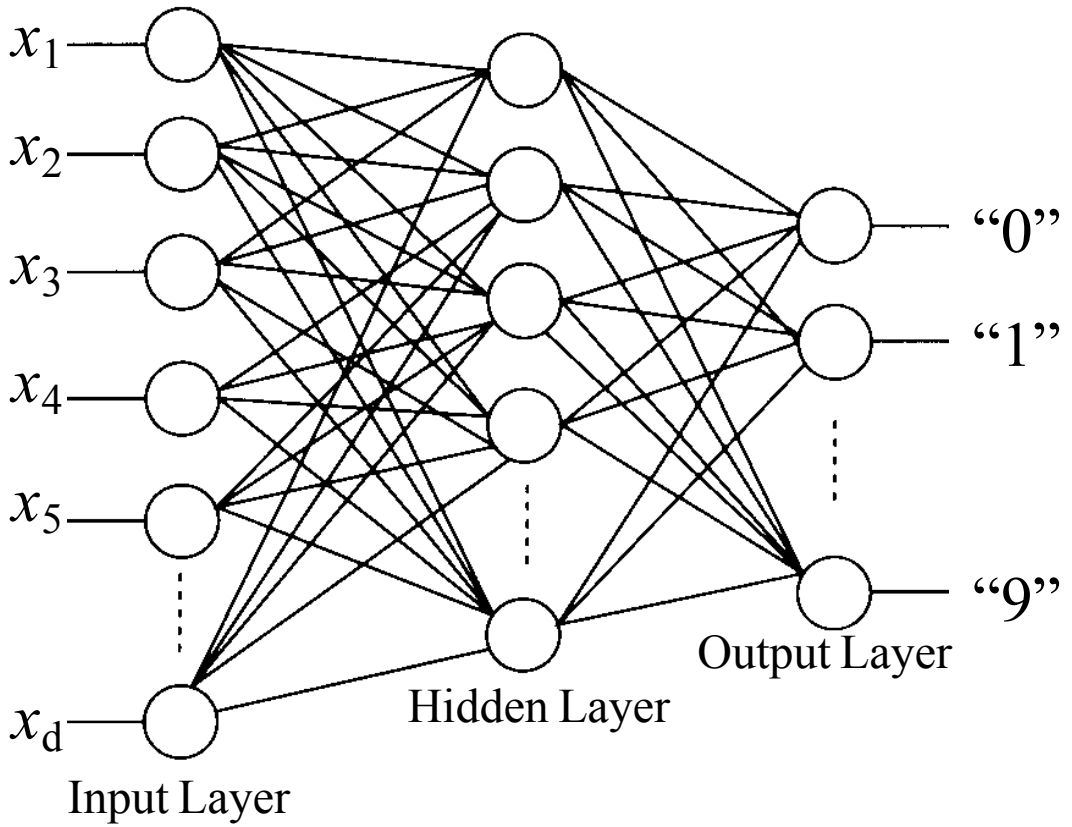
20 × 20 pixel images
$d = 400$ 10 classes

Each image is "unrolled" into a vector x of pixel intensities

Layering Representations







Visualization of Hidden Layer

This lecture

- Review: Neural Network
  - Learning NN
- Recursive and Recurrent NN

Why we need to learn this:
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Recall: Learning as loss minimization

We have a classifier *NN* that is completely defined by its weights

Learn the weights by minimizing a loss L

$$\min_{\boldsymbol{w}} \sum_{i} L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$$

Perhaps with a *regularizer*

So far, we saw that this strategy worked for:

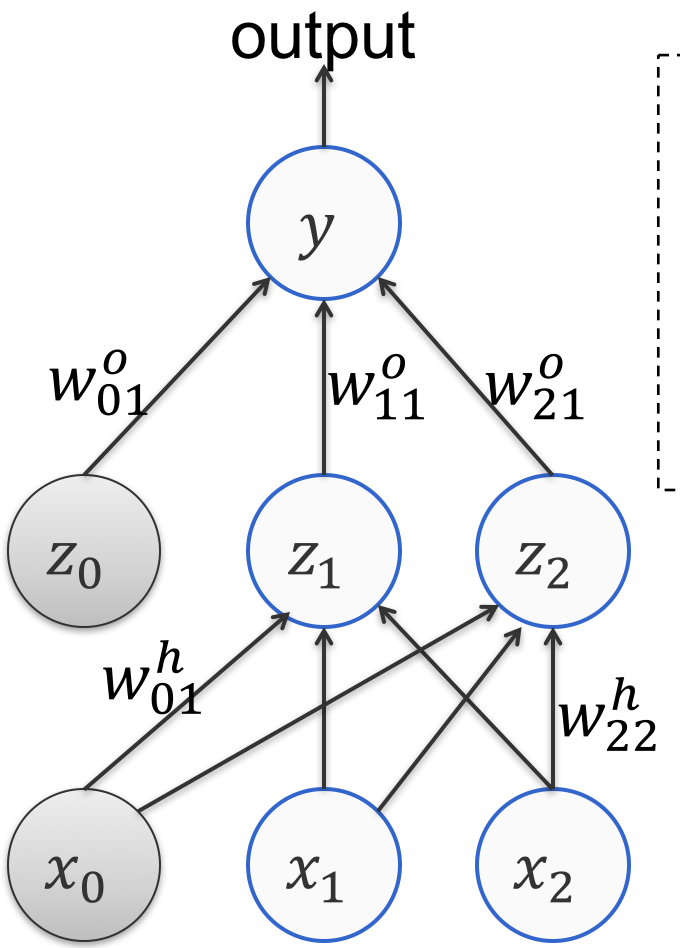
1. Logistic Regression
2. Support Vector Machines
3. Perceptron
4. LMS regression

Each minimizes a different loss function

All of these are linear models

Same idea for non-linear models too!

Back to our running example



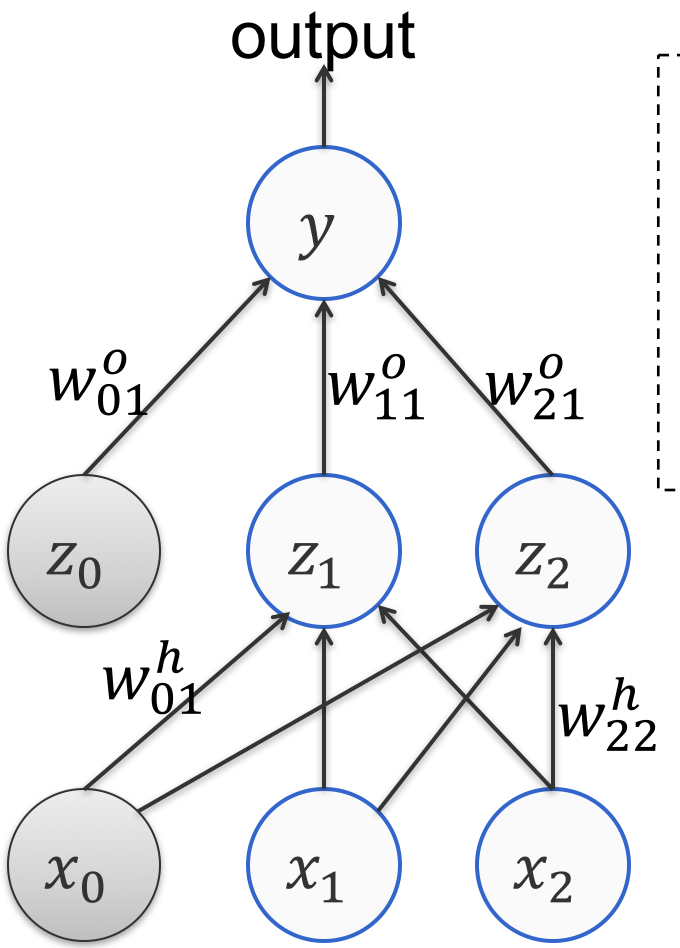
Given an input **x**, how is the output predicted

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

Back to our running example



Given an input **x**, how is the output predicted

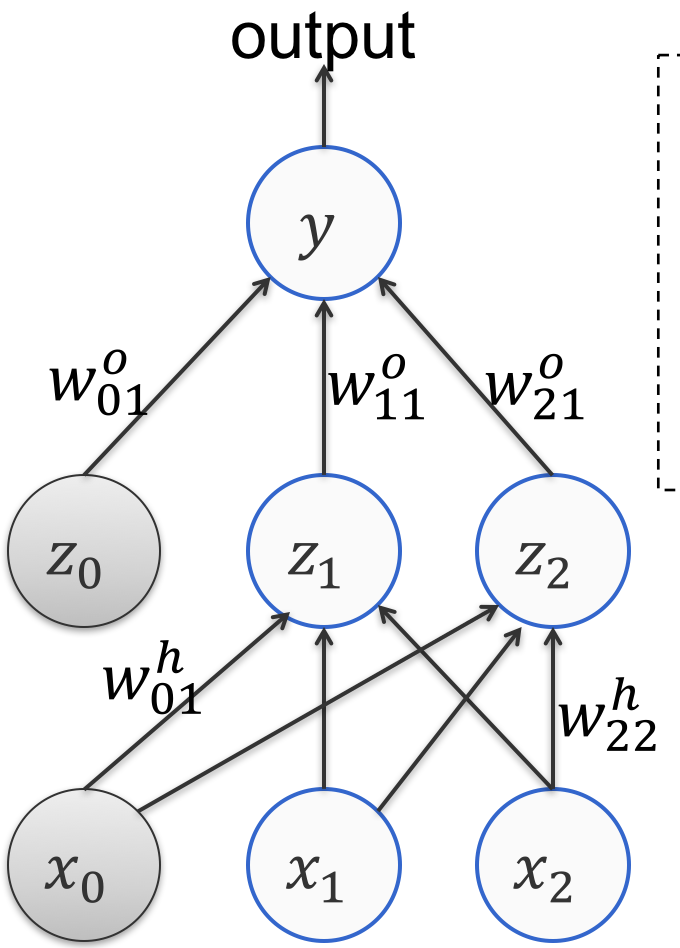
output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

Suppose the true label for this example is a number y^*

Back to our running example



Given an input **x**, how is the output predicted

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

Suppose the true label for this example is a number y^*

We can write the *square loss* for this example as:

$$L = \frac{1}{2}(y - y^*)^2$$

Learning as loss minimization

We have a classifier *NN* that is completely defined by its weights

Learn the weights by minimizing a loss L

$$\min_{\boldsymbol{w}} \sum_{i} L(NN(x_i, w), y_i)$$

Perhaps with a *regularizer*

How do we solve the optimization problem?

Stochastic gradient descent

$$\min_{w} \sum_{i} L(NN(x_i, w), y_i)$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i)$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

γ_t: learning rate, many tweaks possible

Stochastic gradient descent

$$\min_{w} \sum_{i} L(NN(x_i, w), y_i)$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i)$:
      - Treat this example as the entire dataset
        Compute the gradient of the loss $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

The objective is not convex. Initialization can be important

γ_t: learning rate, many tweaks possible

Have we solved everything?

The derivative of the loss function?

$$\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$$

If the neural network is a differentiable function, we can find the gradient

- Or maybe its sub-gradient
- This is decided by the activation functions and the loss function

It was easy for SVMs and logistic regression

- Only one layer

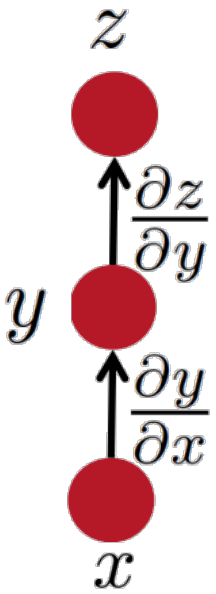
But how do we find the sub-gradient of a more complex function?

- Eg: A recent paper used a ~150 layer neural network for image classification!

We need an efficient algorithm: Backpropagation

Reminder: Chain rule for derivatives

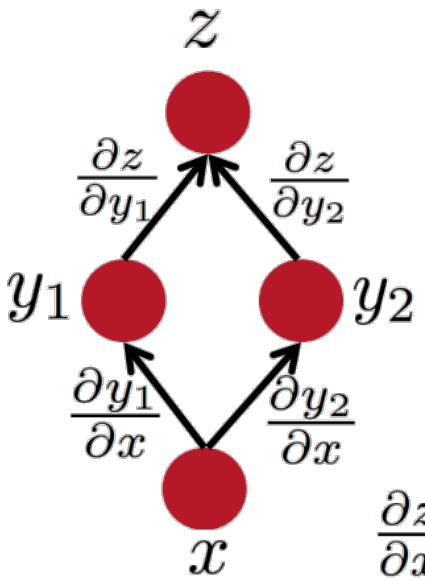
- If z is a function of y and y is a function of x
  - Then z is a function of x, as well
- Question: how to find $\frac{\partial z}{\partial x}$



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Reminder: Chain rule for derivatives

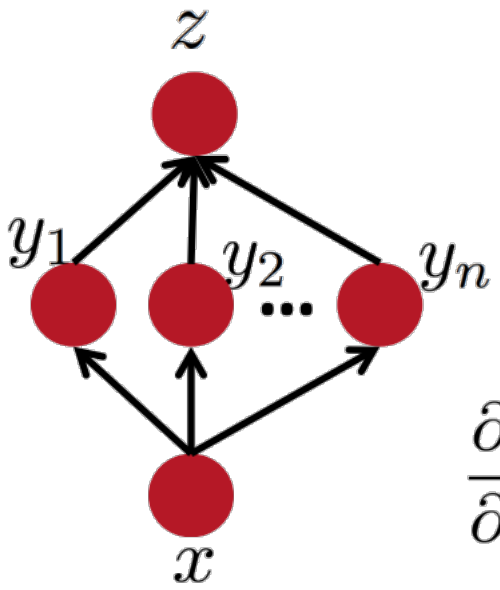
- If $z =$ a function of y_1 + a function of y_2, and the y_i's are functions of x
  - Then z is a function of x, as well
- Question: how to find $\frac{\partial z}{\partial x}$



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

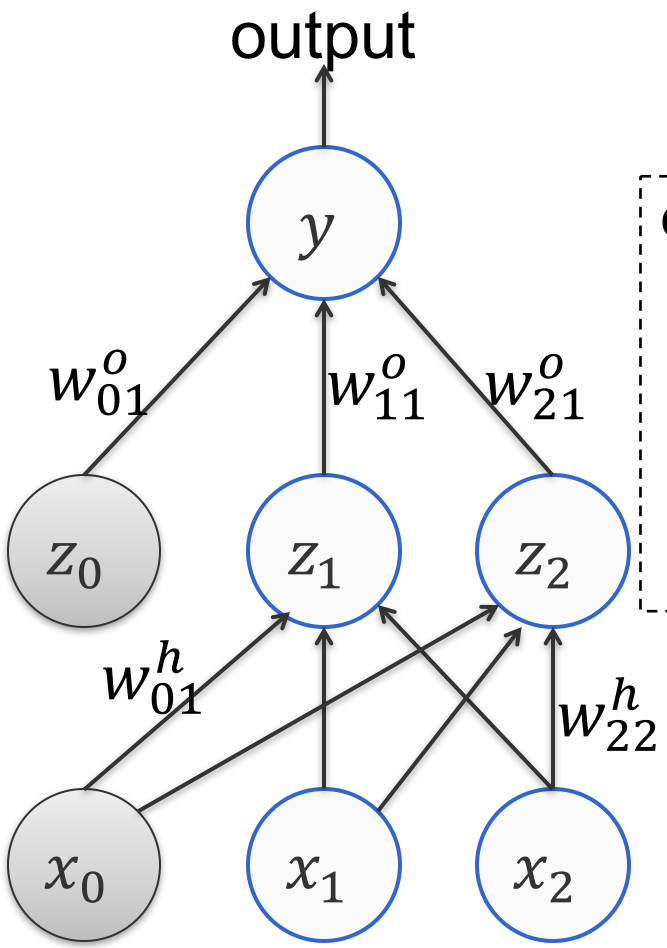
Reminder: Chain rule for derivatives

- If z is a sum of functions of y_i's, and the y_i's are functions of x
  - Then z is a function of x, as well
- Question: how to find $\frac{\partial z}{\partial x}$



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Backpropagation



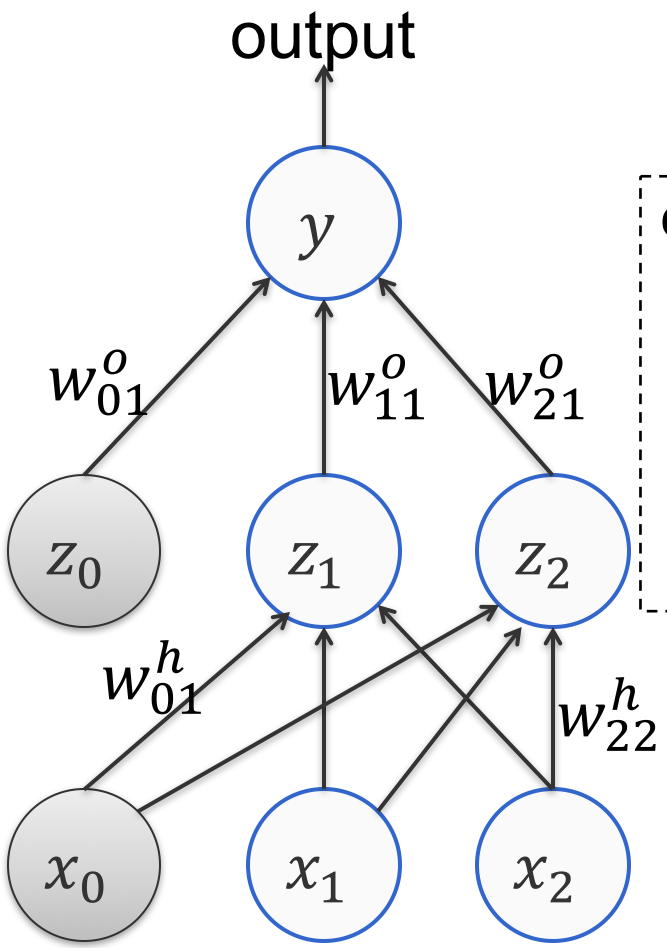
$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

Backpropagation



$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

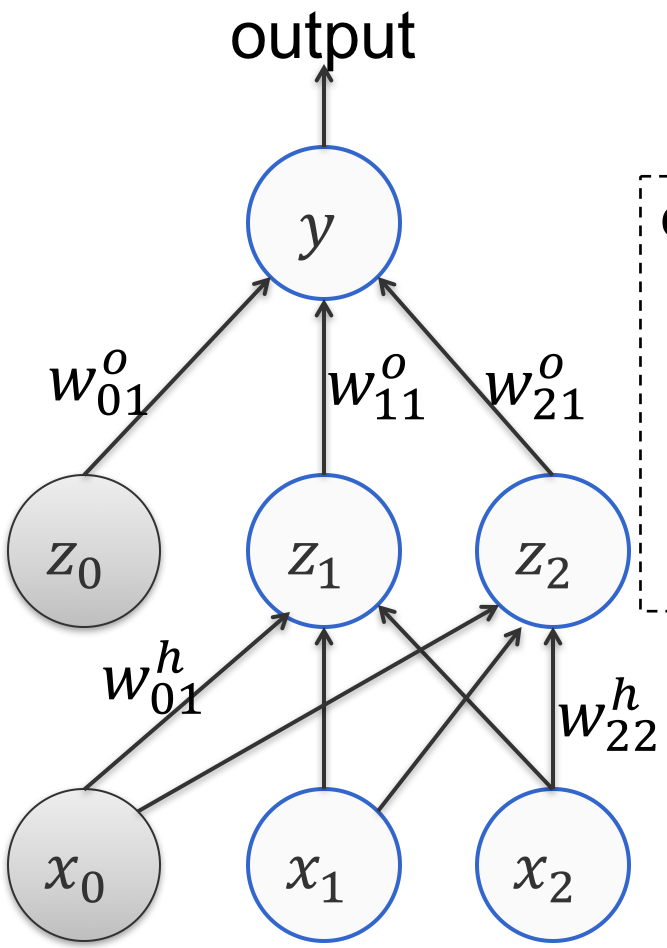
$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

We want to compute $\frac{\partial L}{\partial w_{ij}^{o}}$ and $\frac{\partial L}{\partial w_{ij}^{h}}$

Backpropagation

Applying the chain rule to compute the gradient (And remembering partial computations along the way to speed up things)



$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

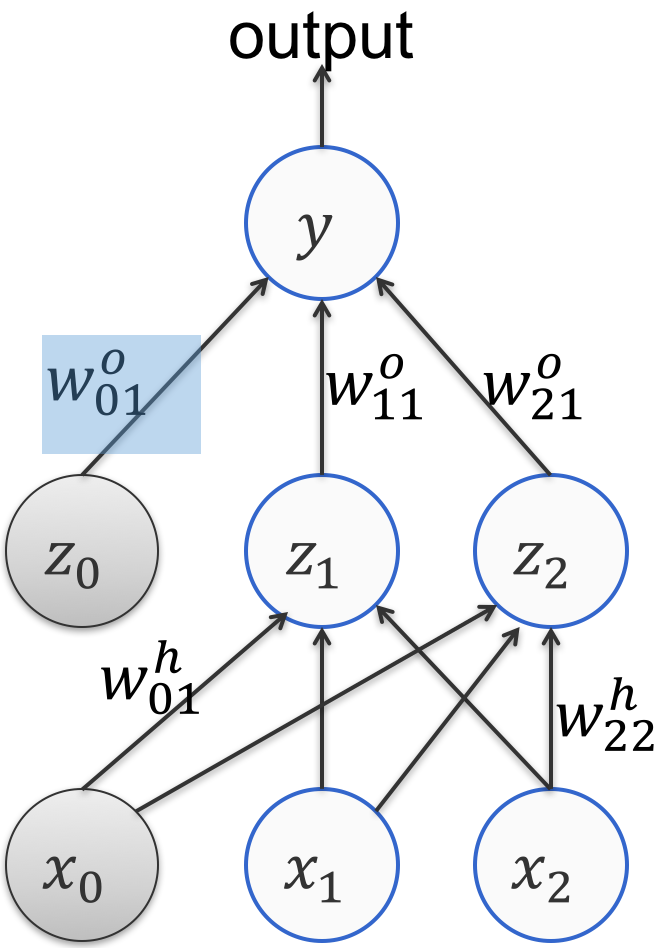
$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

We want to compute $\frac{\partial L}{\partial w_{ij}^{o}}$ and $\frac{\partial L}{\partial w_{ij}^{h}}$

$$L = \frac{1}{2}(y - y^*)^2$$

output $\quad y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer

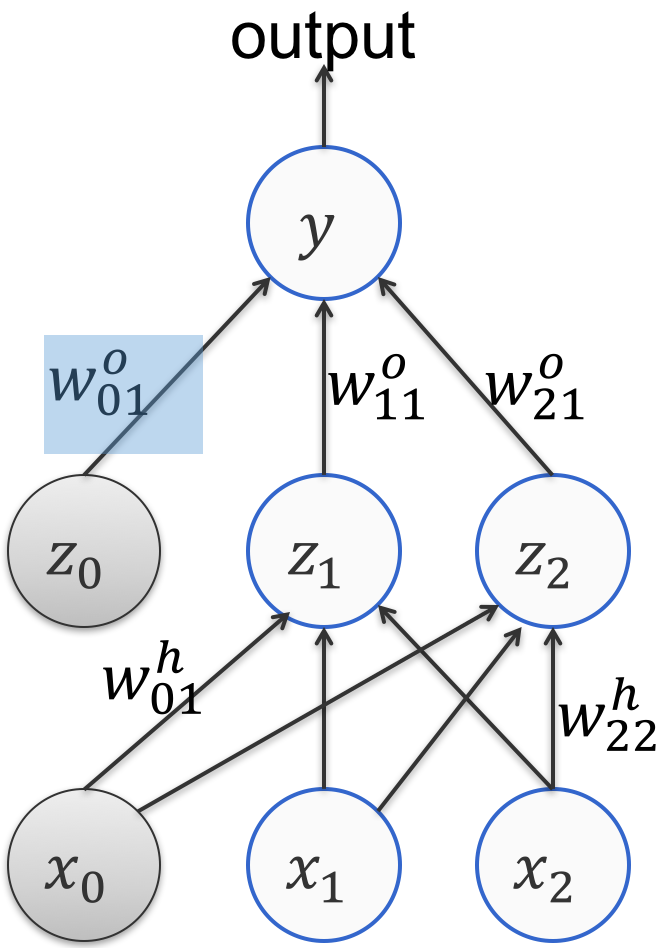


$$\frac{\partial L}{\partial w_{01}^o}$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $\quad y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer

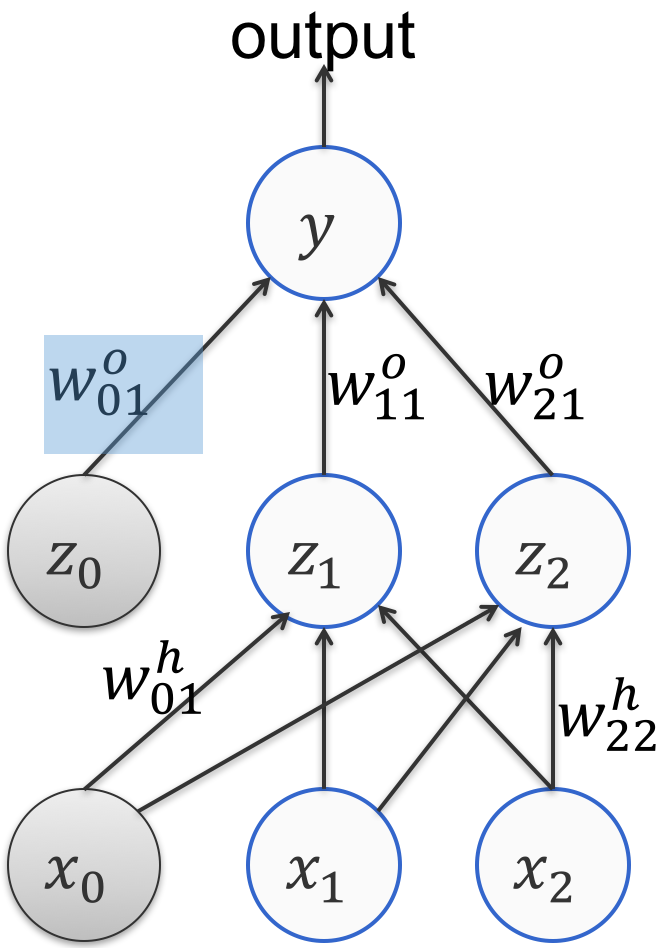


$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^o}$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer



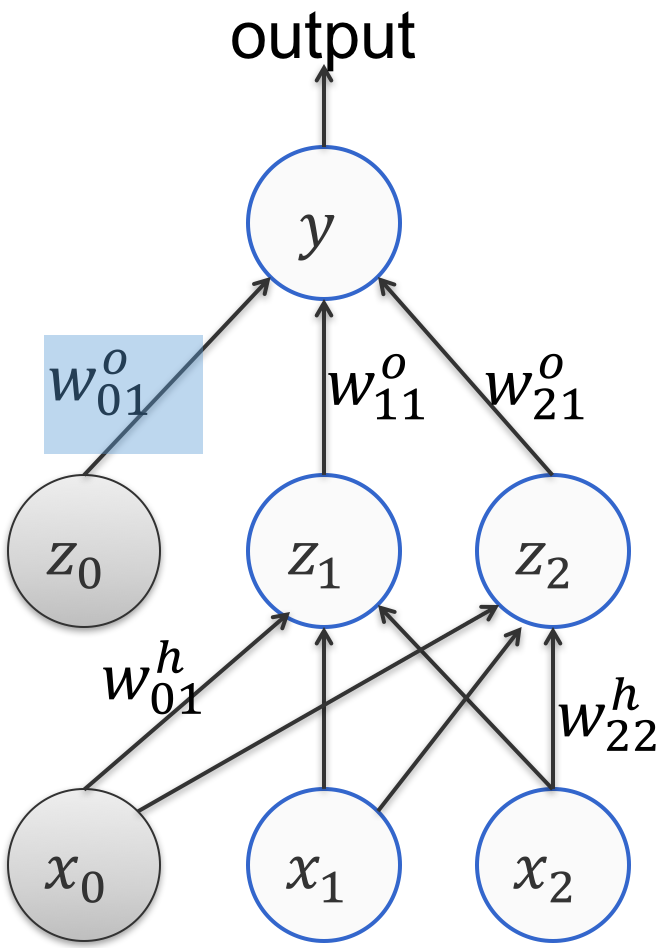
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^o}$$

$$\frac{\partial L}{\partial y} = y - y^*$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer



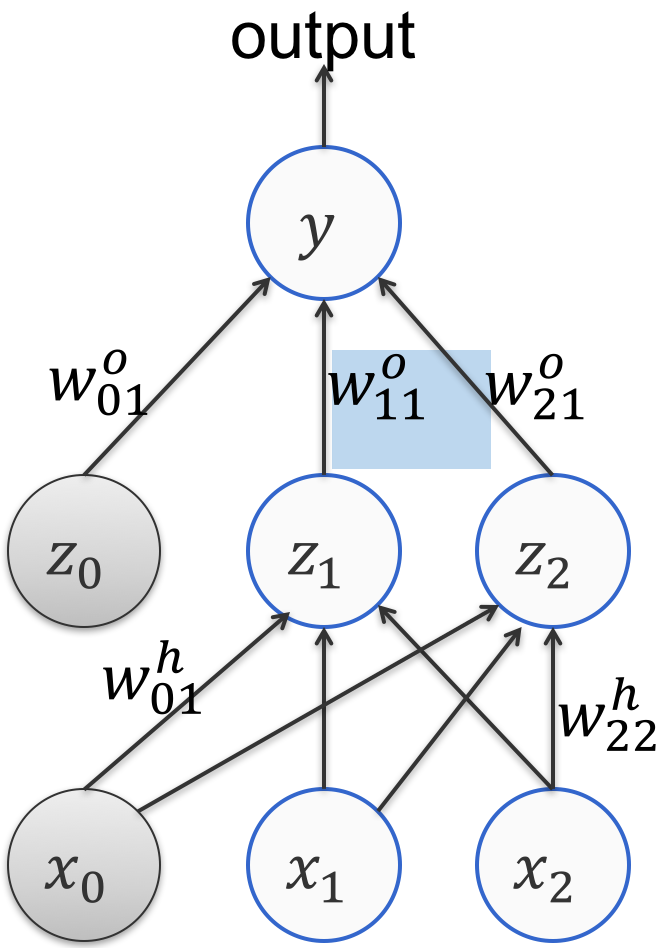
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^o}$$

$$\frac{\partial L}{\partial y} = y - y^* \qquad \frac{\partial y}{\partial w_{01}^o} = 1$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $\quad y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer

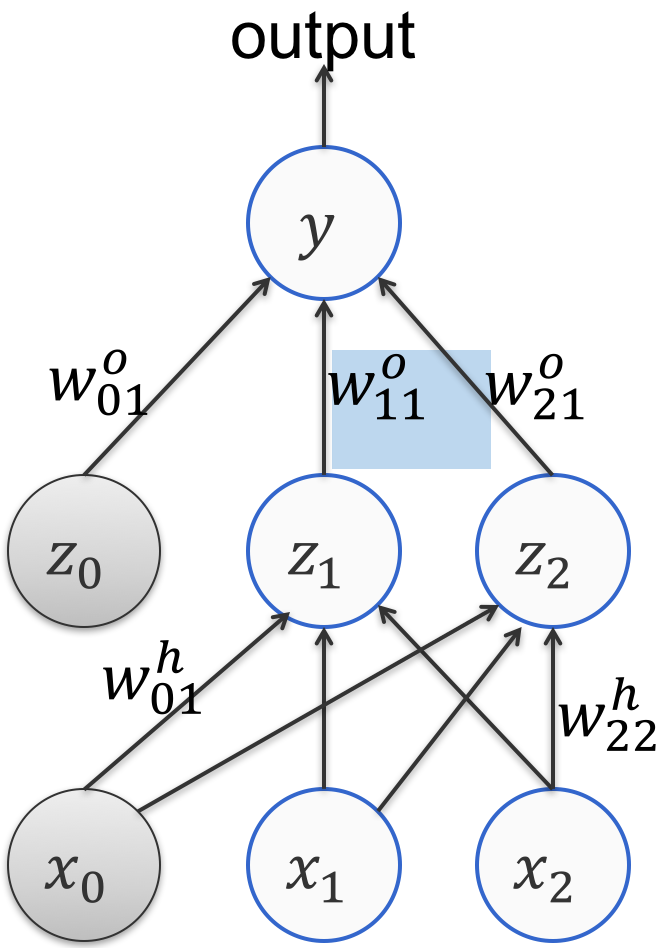


$$\frac{\partial L}{\partial w_{11}^o}$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer

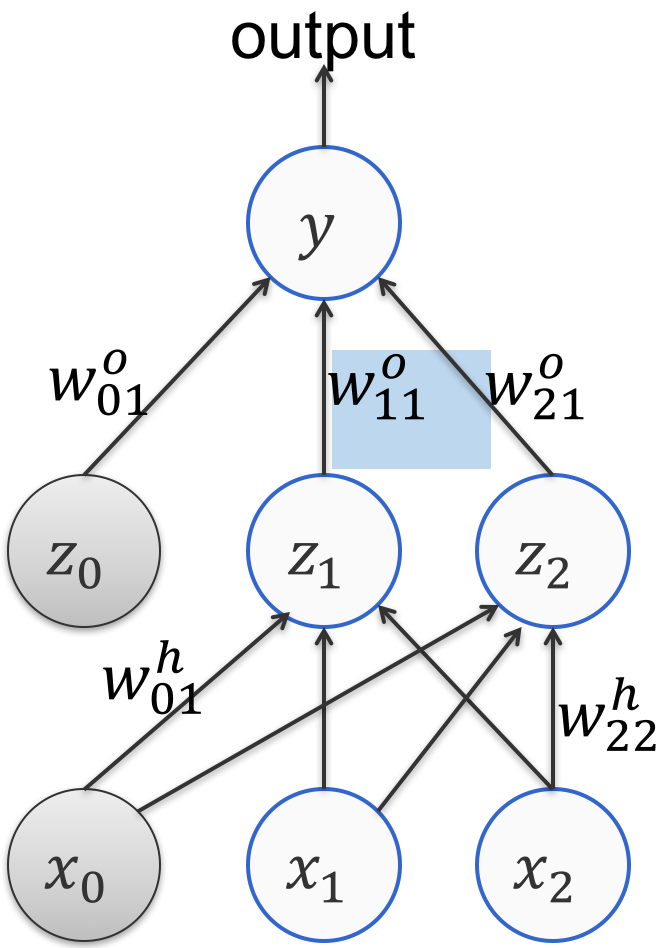


$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^0}$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer



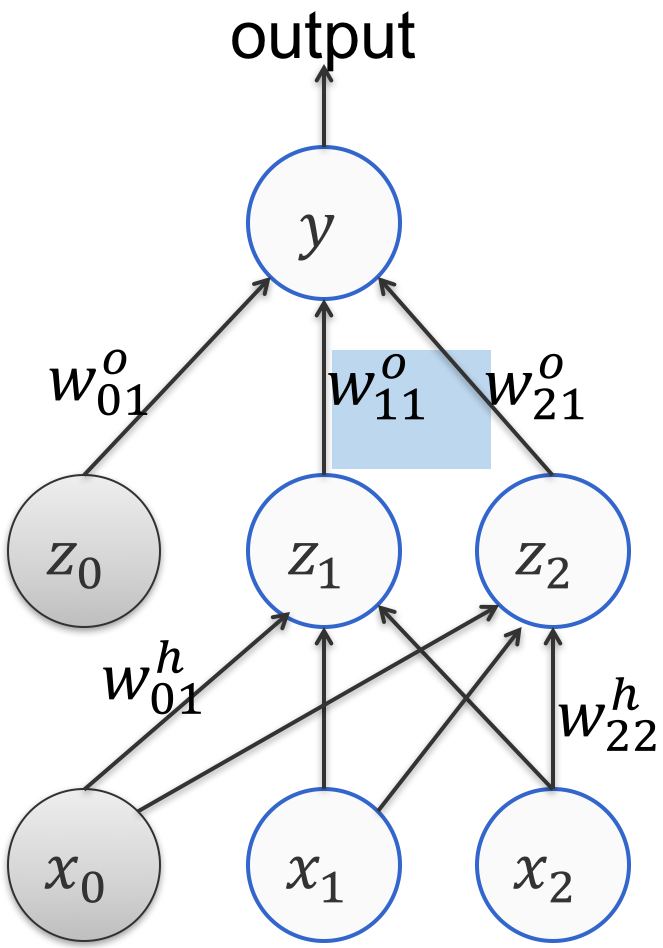
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^0}$$

$$\frac{\partial L}{\partial y} = y - y^*$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer



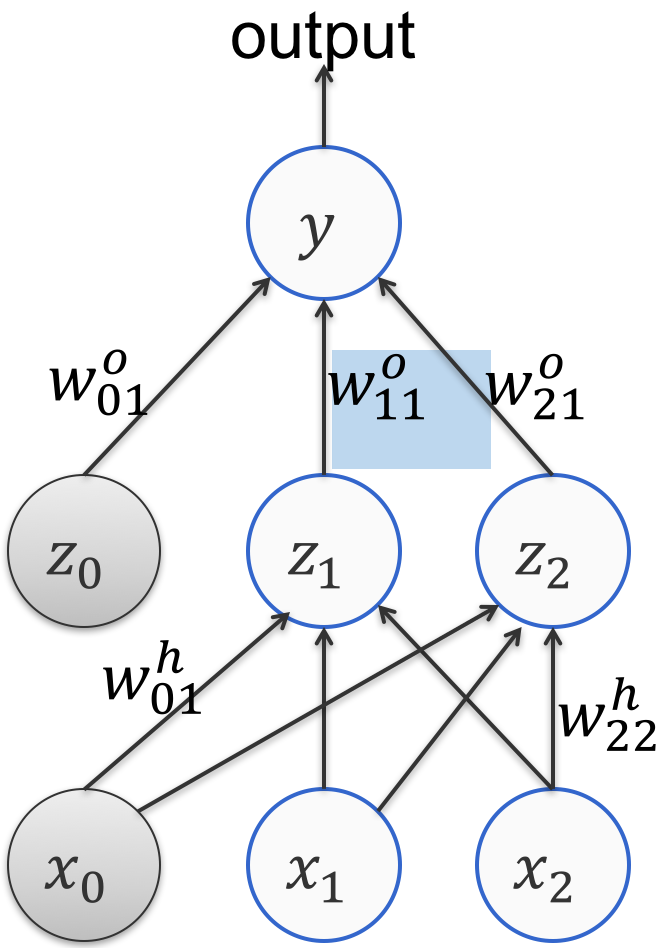
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^o}$$

$$\frac{\partial L}{\partial y} = y - y^* \qquad \frac{\partial y}{\partial w_{01}^o} = z_1$$

$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

Output layer



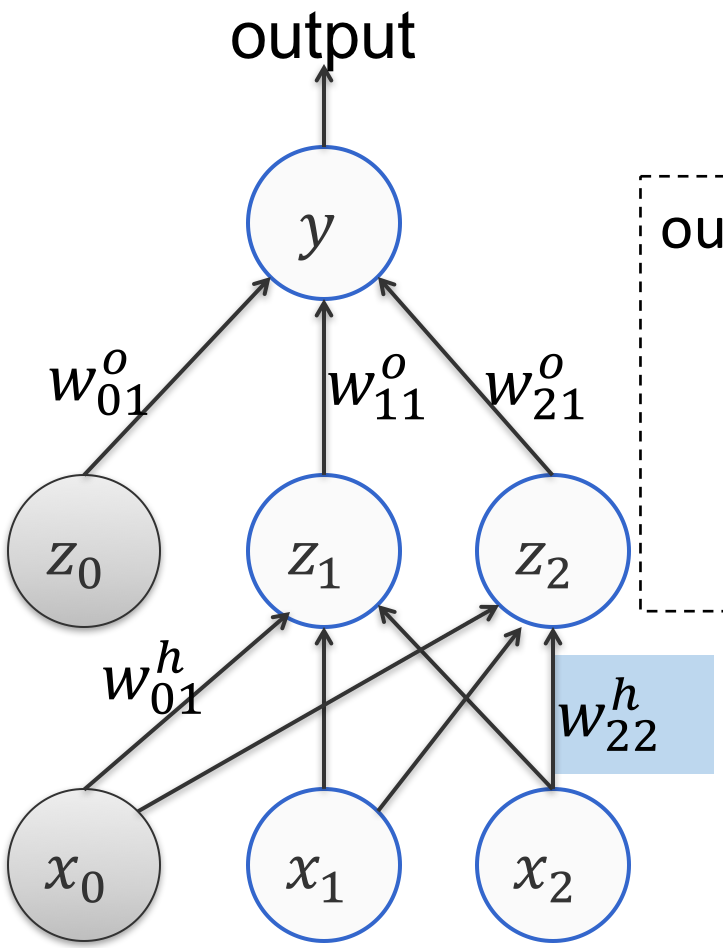
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^o}$$

$$\frac{\partial L}{\partial y} = y - y^* \qquad \frac{\partial y}{\partial w_{01}^o} = z_1$$

We have already computed this partial derivative for the previous case

Cache to speed up!

Hidden layer derivatives



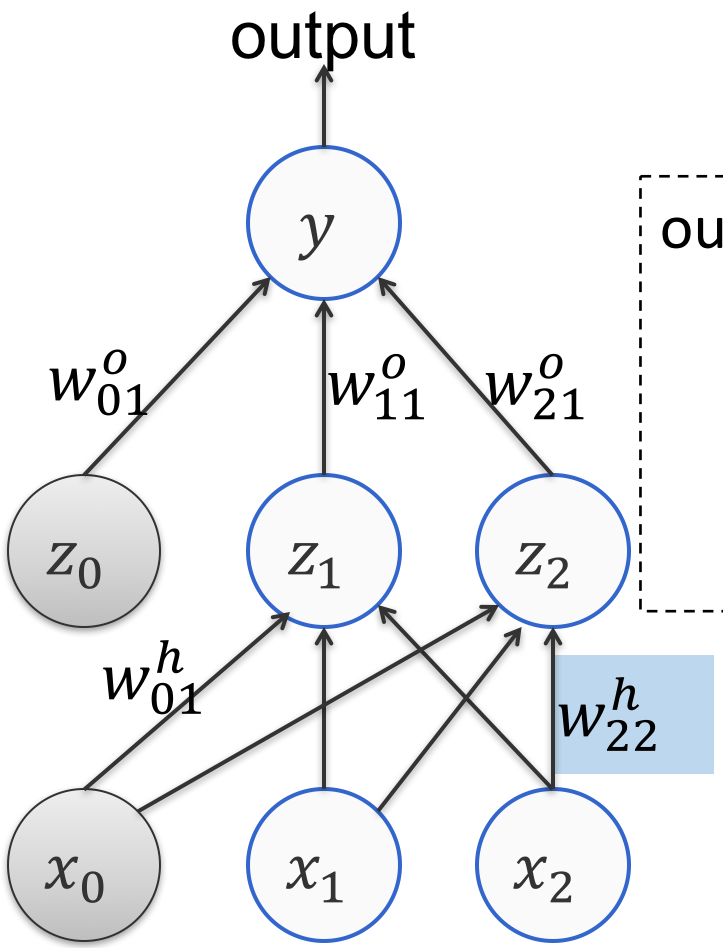
$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

Hidden layer derivatives



$$L = \frac{1}{2}(y - y^*)^2$$

output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

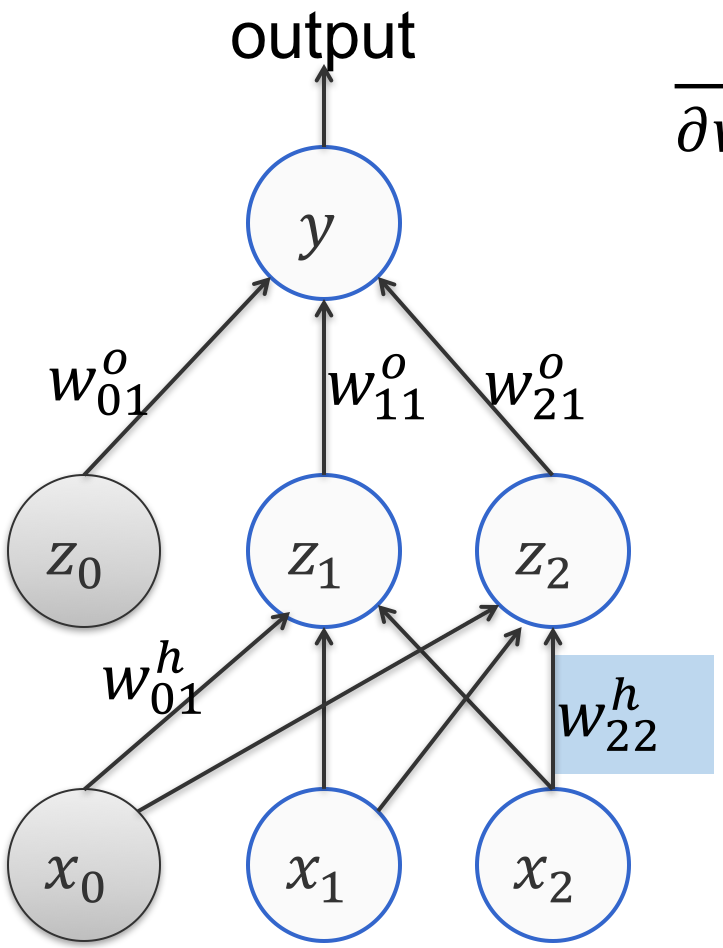
$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

We want $\frac{\partial L}{\partial w_{22}^{h}}$

$$L = \frac{1}{2}(y - y^*)^2$$

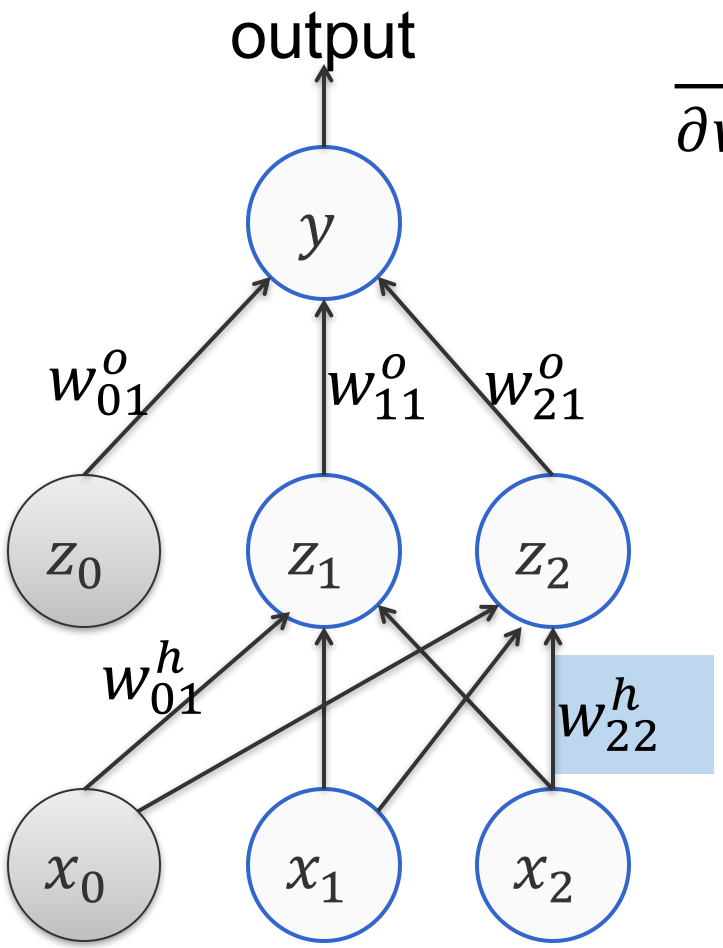
Hidden layer derivatives



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

$$y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$$

Hidden layer

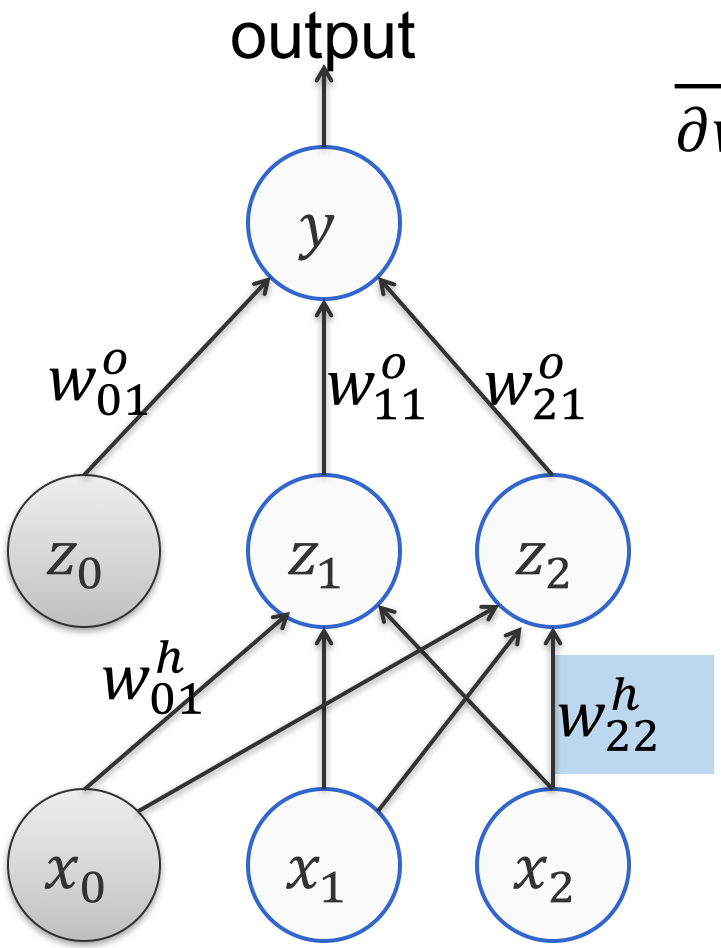


$$\frac{\partial L}{\partial w_{22}^{h}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^{h}}$$

$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^{h}}\left(w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2\right)$$

$$y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$$

Hidden layer



$$\frac{\partial L}{\partial w_{22}^{h}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^{h}}$$

$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^{h}}(w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2)$$

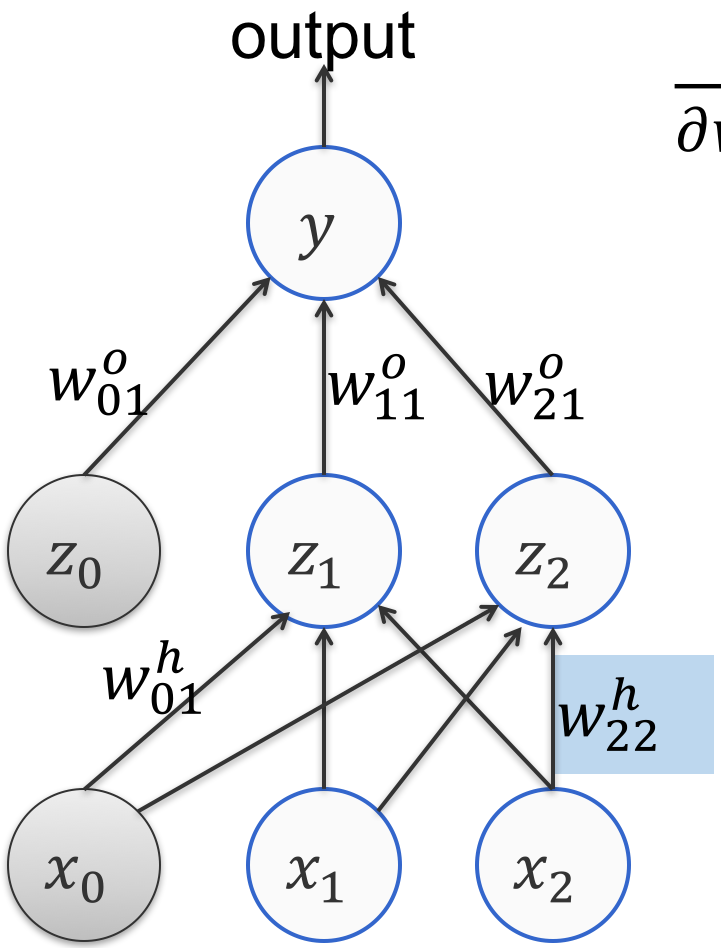
$$= \frac{\partial L}{\partial y}(w_{11}^{o}\frac{\partial}{\partial w_{22}^{h}} z_1 + w_{21}^{o}\frac{\partial}{\partial w_{22}^{h}} z_2)$$

$\frac{\partial}{\partial w_{22}^{h}} z_1 \rightarrow 0$

z_1 is not a function of w_{22}^{h}

$$y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$$

Hidden layer



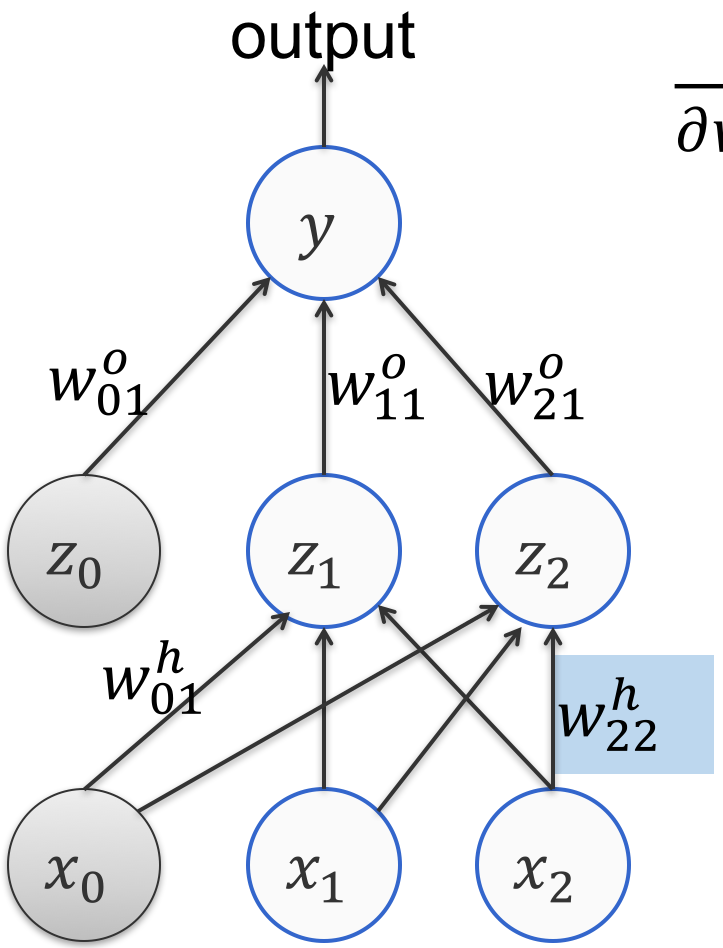
$$\frac{\partial L}{\partial w_{22}^{h}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^{h}}$$

$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^{h}}\left(w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2\right)$$

$$= \frac{\partial L}{\partial y} w_{21}^{o} \frac{\partial z_2}{\partial w_{22}^{h}}$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

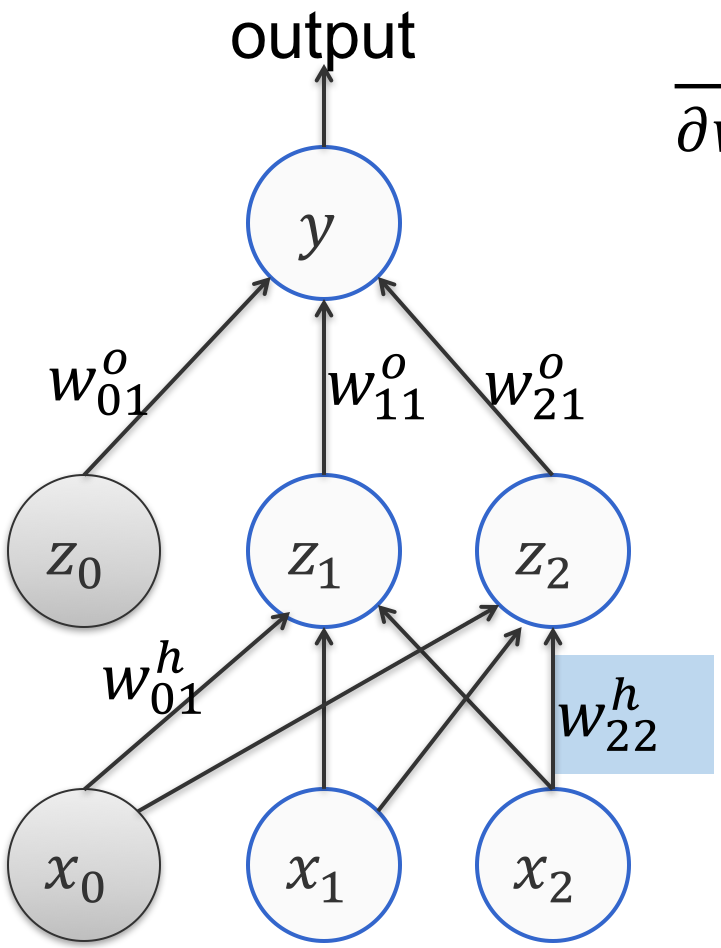
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

$$= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial w_{22}^h}$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

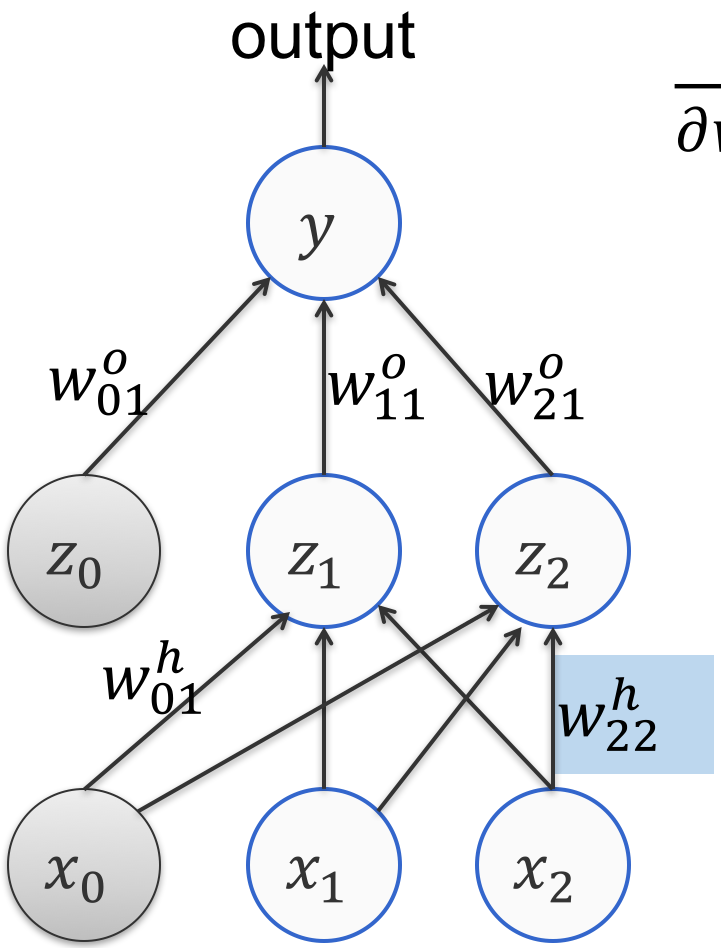
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

$$= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial w_{22}^h}$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

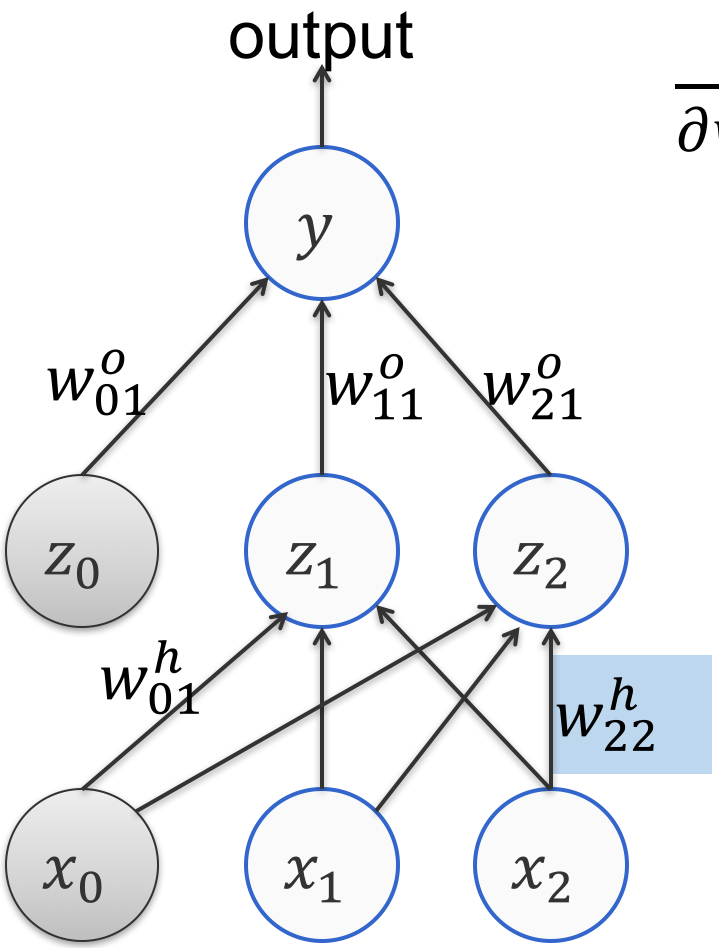
$$= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial w_{22}^h}$$

$$= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s}\frac{\partial s}{\partial w_{22}^h}$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer

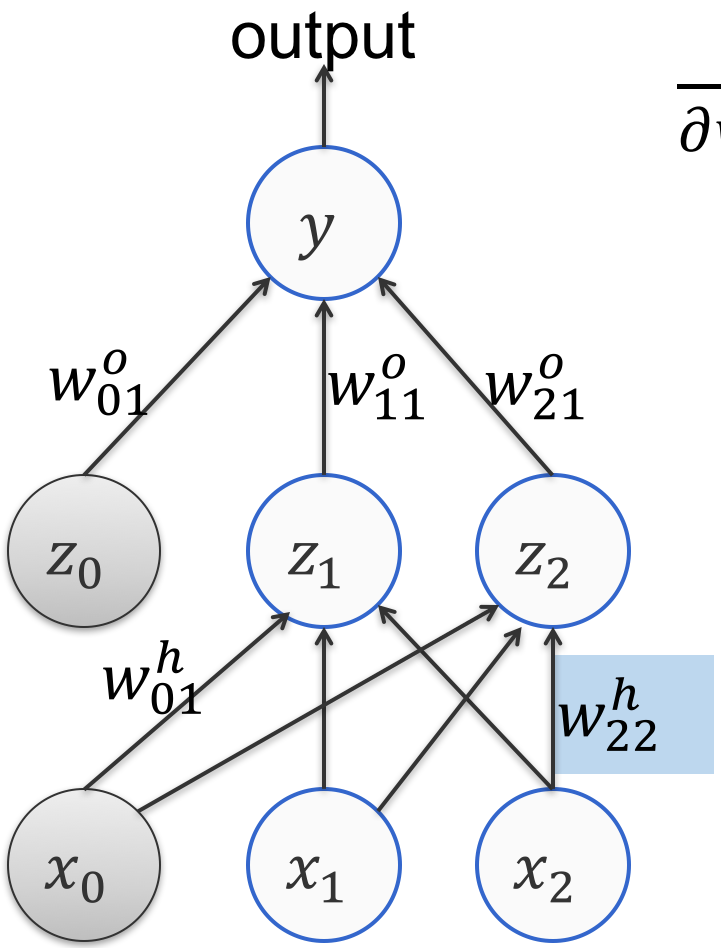


$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$ (From previous slide)

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



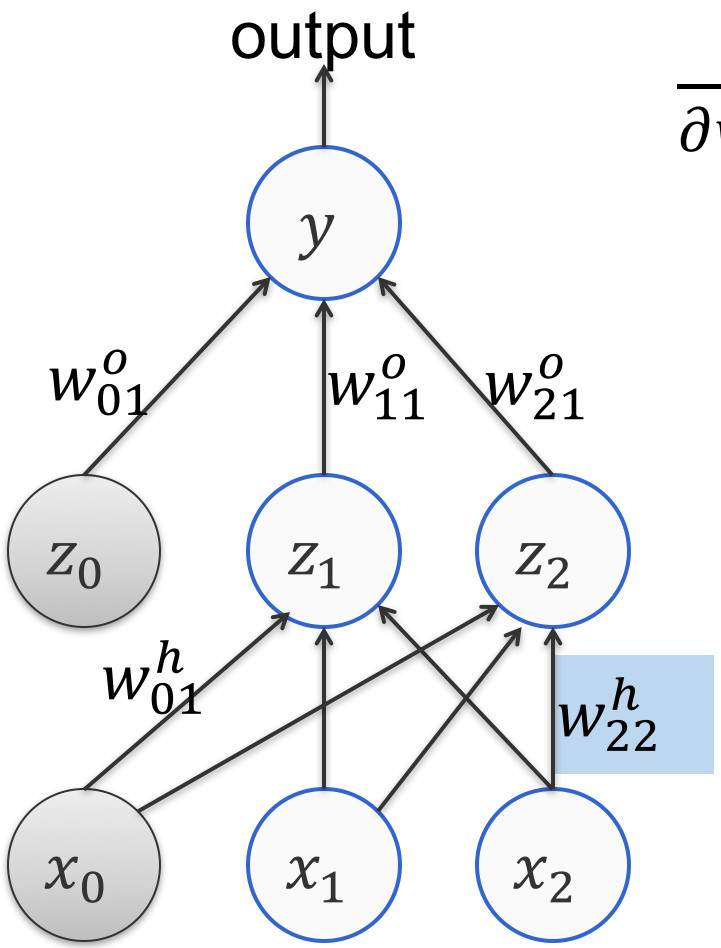
$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

Each of these partial derivatives is easy

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

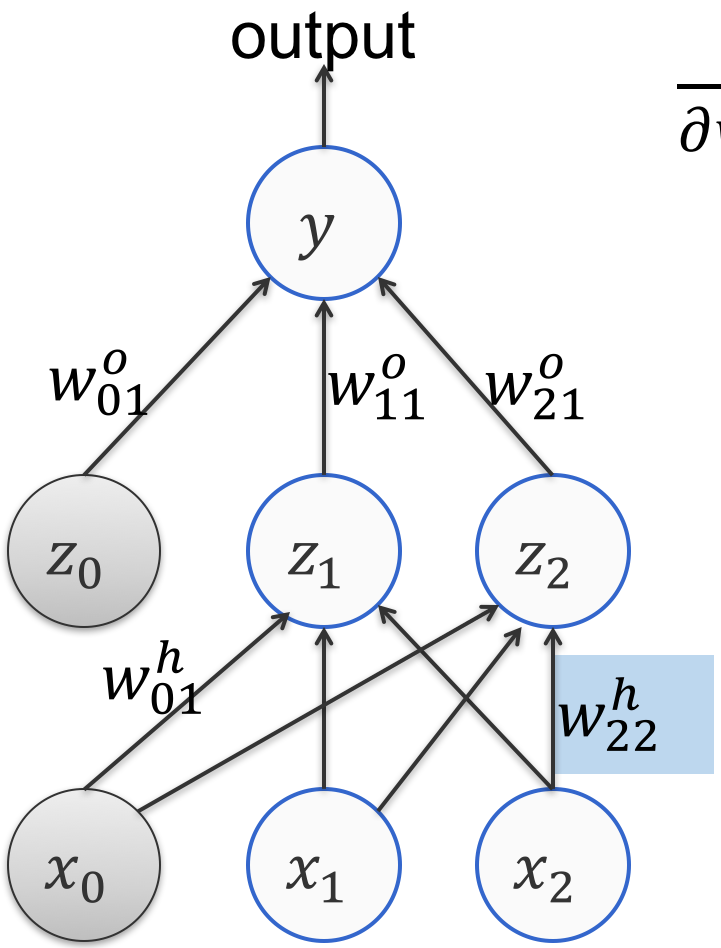
Each of these partial derivatives is easy

$$\frac{\partial L}{\partial y} = y - y^*$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

Each of these partial derivatives is easy

$$\frac{\partial L}{\partial y} = y - y^*$$

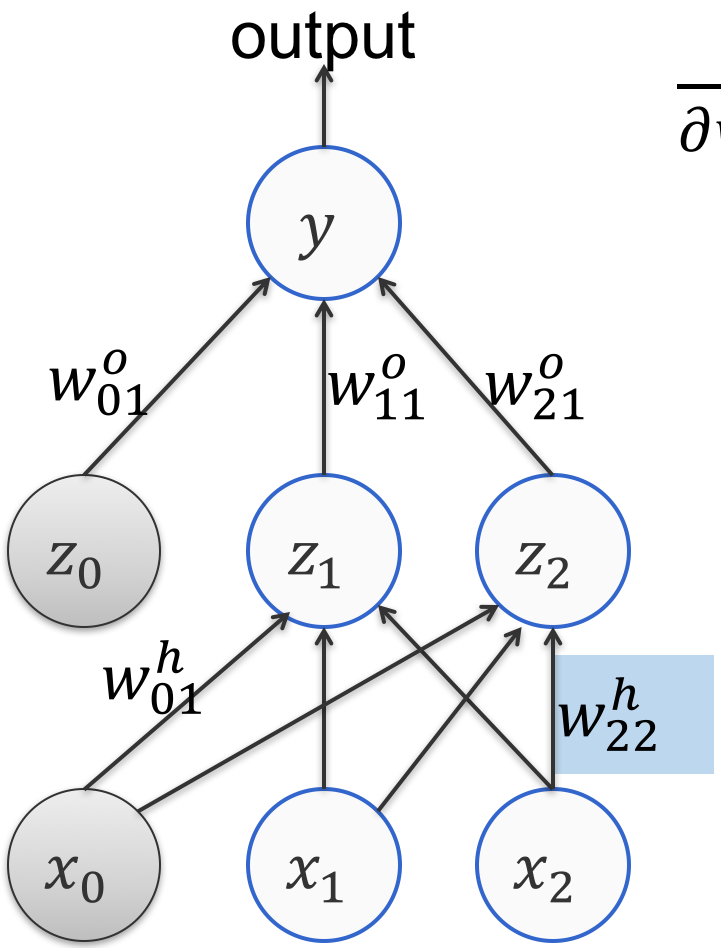
$$\frac{\partial z_2}{\partial s} = z_2(1 - z_2)$$

Why? Because $z_2(s)$ is the logistic function we have already seen

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

Each of these partial derivatives is easy

$$\frac{\partial L}{\partial y} = y - y^*$$

$$\frac{\partial z_2}{\partial s} = z_2(1 - z_2)$$

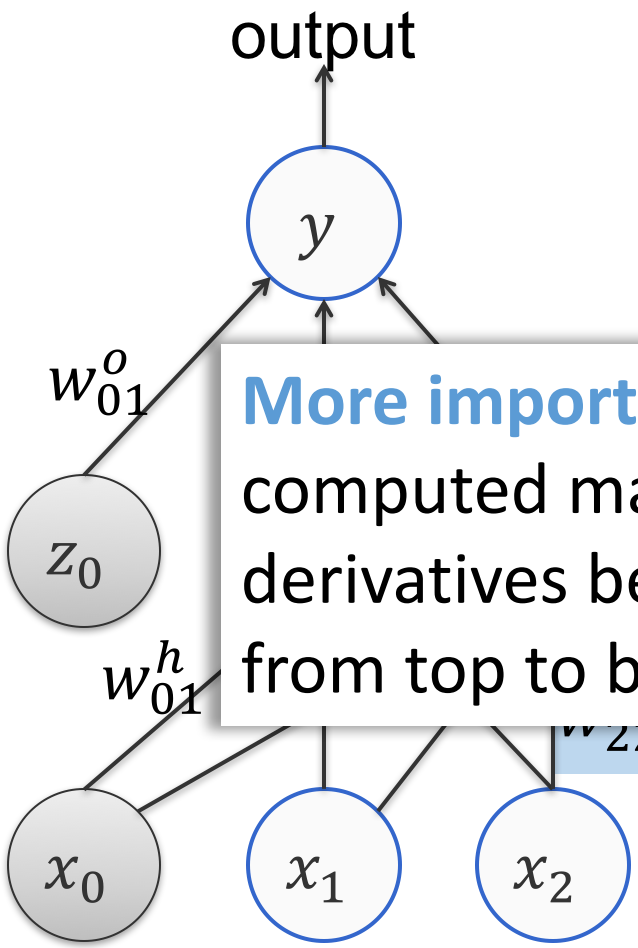
$$\frac{\partial s}{\partial w_{22}^h} = x_2$$

Why? Because $z_2(s)$ is the logistic function we have already seen

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Call this s

Hidden layer



$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

Each of these partial derivatives is easy

More important: We have already computed many of these partial derivatives because we are proceeding from top to bottom (i.e. backwards)

cause $z_2(s)$
istic
we have
een

$$\frac{\partial s}{\partial w_{22}^h} = x_2$$

The Backpropagation Algorithm

The same algorithm works for multiple layers

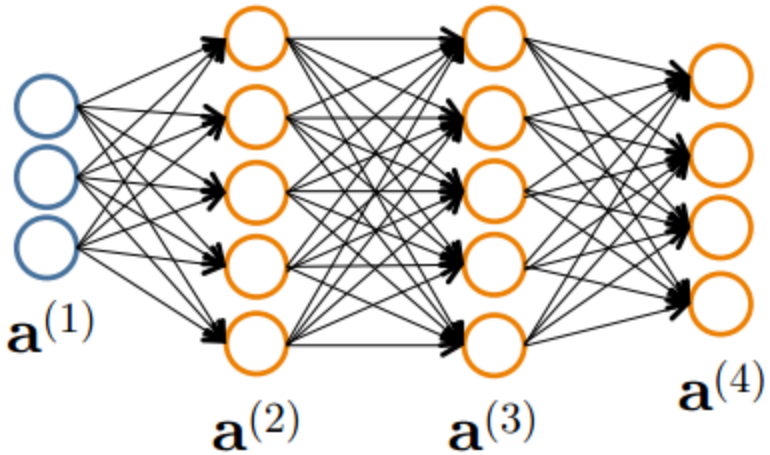
Repeated application of the chain rule for partial derivatives

- First perform forward pass from inputs to the output
- Compute loss

- From the loss, proceed backwards to compute partial derivatives using the chain rule
- Cache partial derivatives as you compute them
  - Will be used for lower layers

Forward Propagation

- Given one labeled training instance $(\mathbf{x}, y)$:

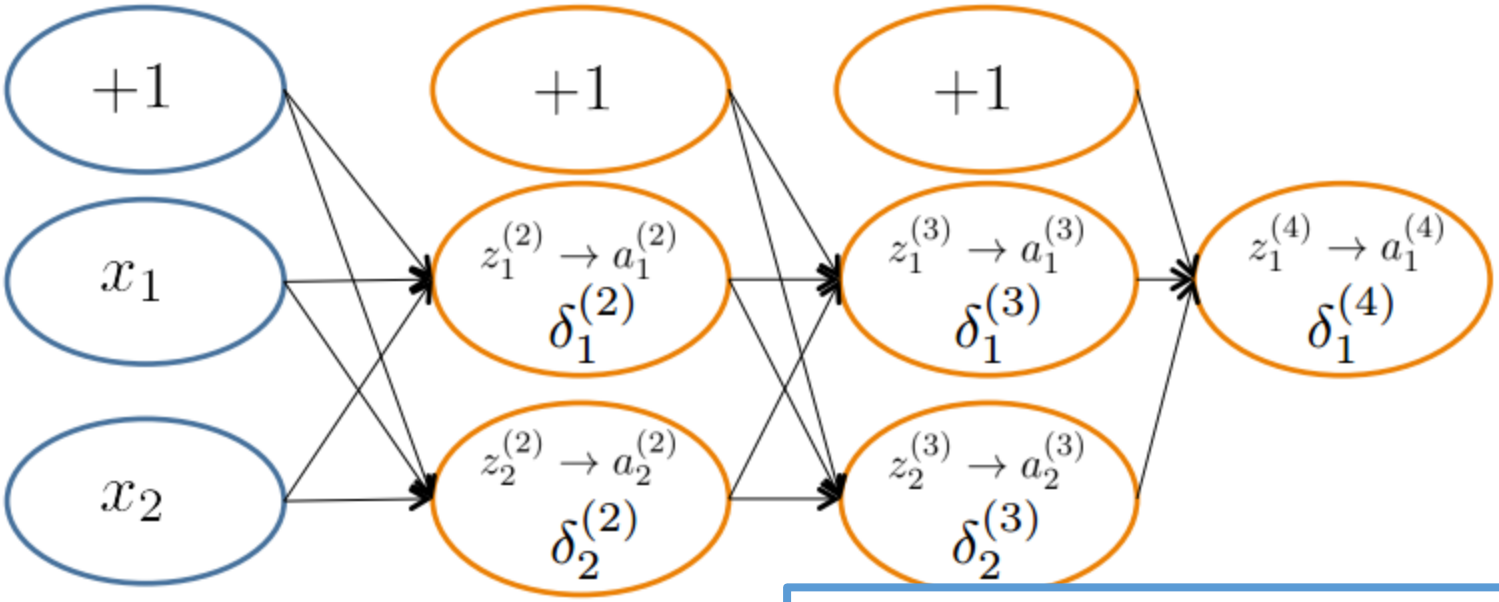


Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Based on slide by Andrew Ng

Backpropagation: Compute Gradient



$$\frac{d}{dt} f(g(t)) = f'(g(t))g'(t) = \frac{df}{dg} \cdot \frac{dg}{dt}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Based on slide by Andrew Ng

Mechanizing learning

- Backpropagation gives you the gradient that will be used for gradient descent
  - SGD gives us a generic learning algorithm
  - Backpropagation is a generic method for computing partial derivatives

- A recursive algorithm that proceeds from the top of the network to the bottom

- Modern neural network libraries implement automatic differentiation using backpropagation
  - Allows easy exploration of network architectures
  - Don’t have to keep deriving the gradients by hand each time

Stochastic gradient descent

$$\min_{w} \sum_{i} L(NN(x_i, w), y_i)$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$

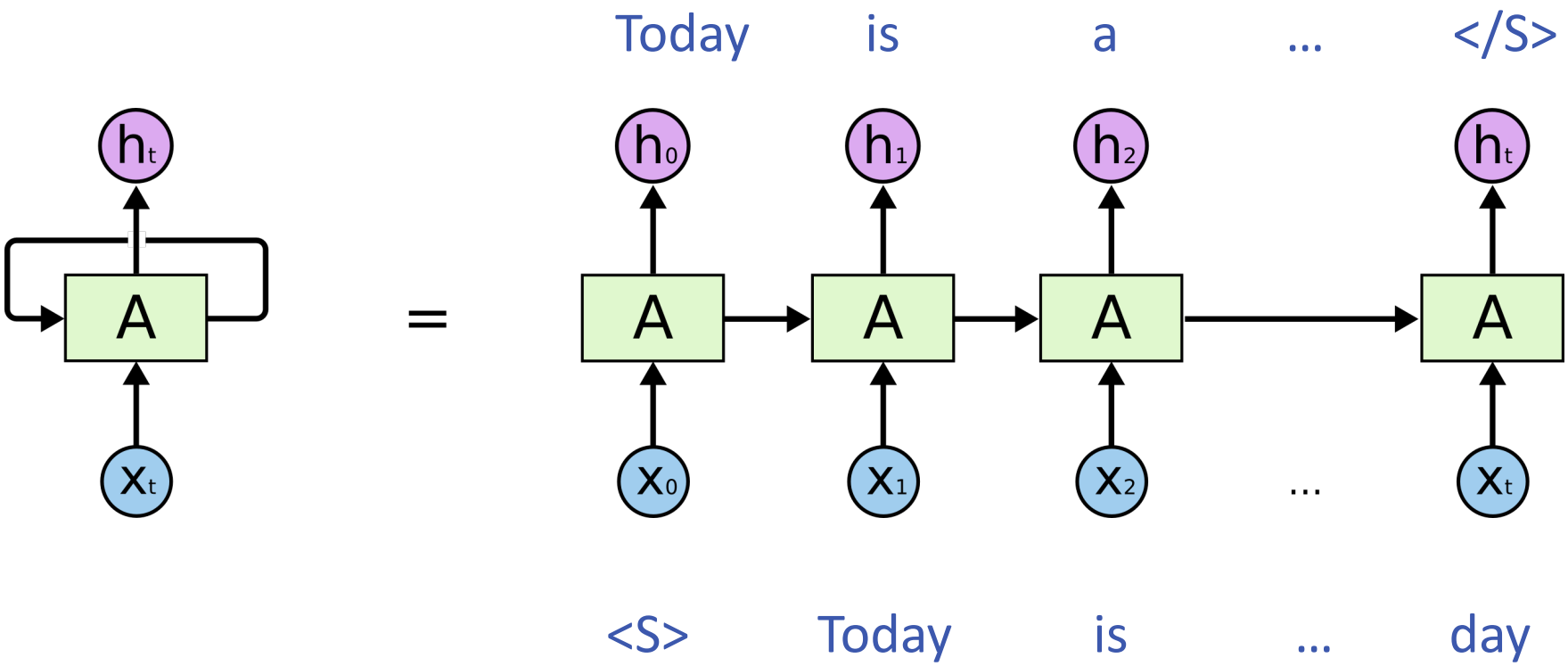
1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i)2$ S:
      - Treat this example as the entire dataset
      - Compute the gradient of the loss $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$ using backpropagation
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

This lecture

- Neural Network
- Recurrent NN

How to deal with input with variant size?

- Use same parameters

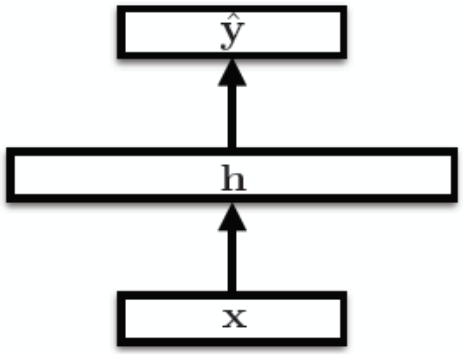


Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

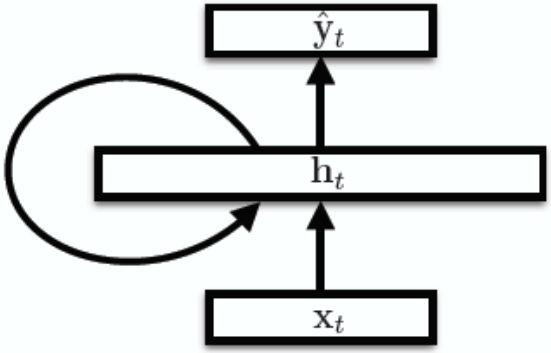
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$



Recurrent NN

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$

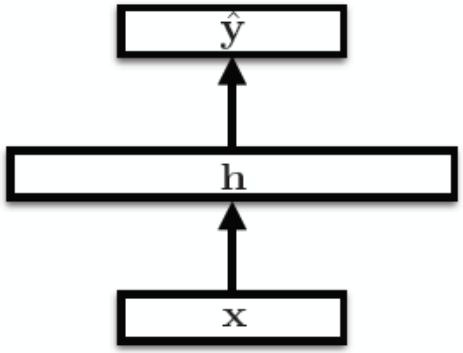


Recurrent Neural Networks

Feed-forward NN

$$\mathbf{h} = g(\mathbf{V}\mathbf{x} + \mathbf{c})$$

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}$$

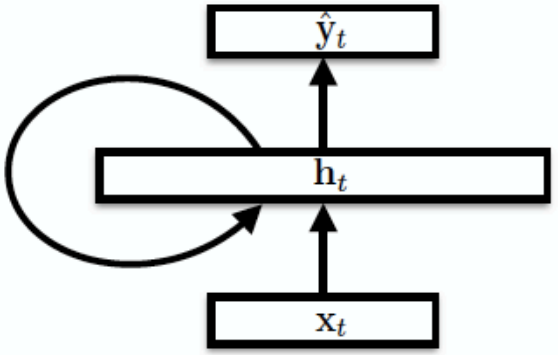


Recurrent NN

~~$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$~~

$$\mathbf{h}_t = g(\mathbf{V}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{c})$$

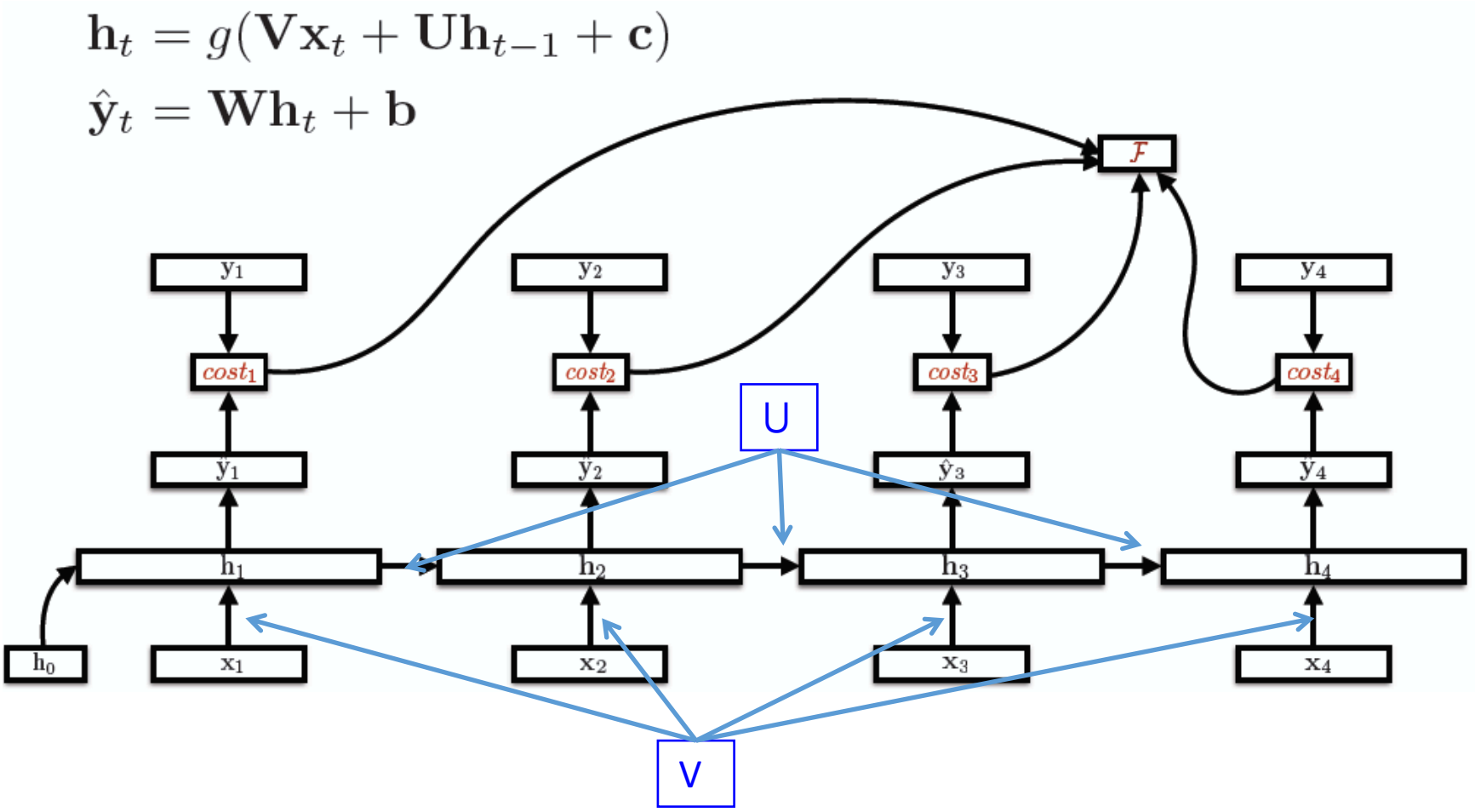
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



Unroll RNNs

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$

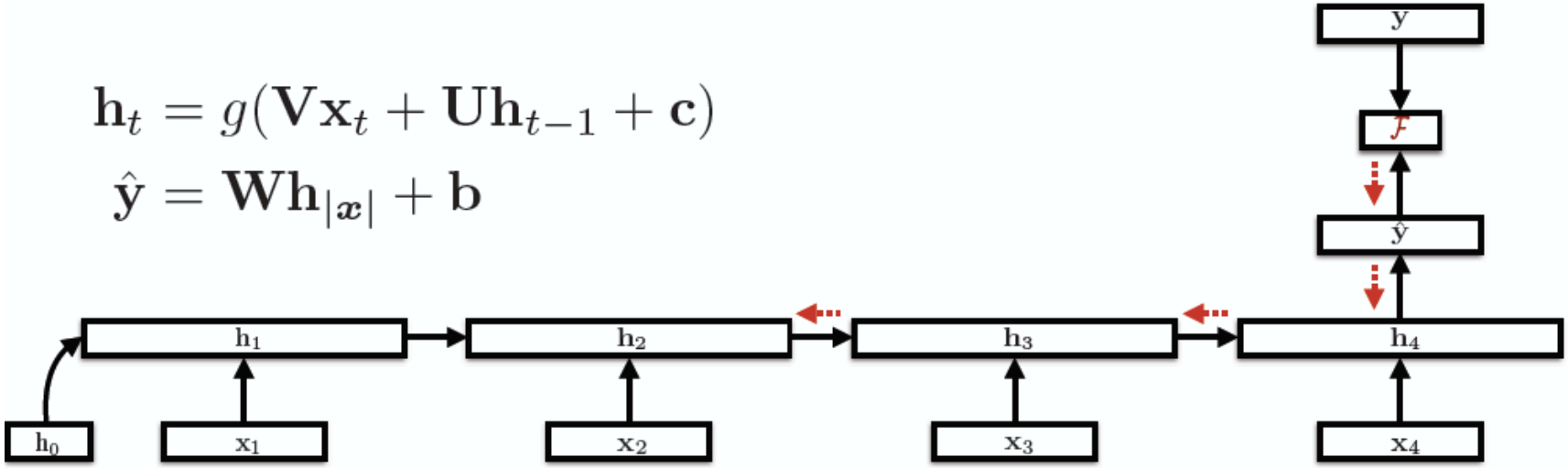
$$\hat{\mathbf{y}}_t = \mathbf{W}\mathbf{h}_t + \mathbf{b}$$



RNN training

- Back-propagation over time

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



What happens to gradients as you go back in time?

$$\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}_4}\frac{\partial \mathcal{F}}{\partial \hat{\mathbf{y}}}\frac{\partial \mathcal{F}}{\partial \mathcal{F}}$$

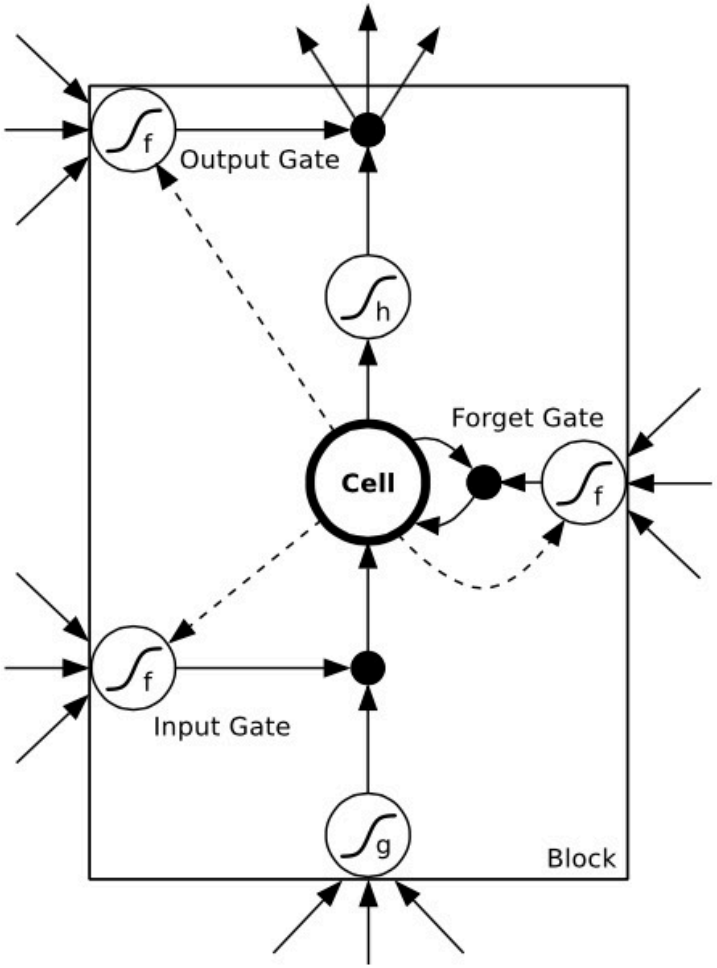
Vanishing Gradients

- For the traditional activation functions, each gradient term has the value in range (-1, 1).
- Multiplying *n* of these small numbers to compute gradients
- The longer the sequence is, the more severe the problems are.

RNNs characteristics

- Model hidden states (input) dependencies
- Errors “back propagation over time”
- Feature learning methods
- Vanishing gradient problem:
  cannot model long-distant dependencies of the hidden states.

Long-Short Term Memory Networks (LSTMs)



$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma(W_i[x_t, h_t] + b_i) \\ \sigma(W_f[x_t, h_t] + b_f) \\ \sigma(W_o[x_t, h_t] + b_o) \\ f(W_g[x_t, h_t] + b_g) \end{pmatrix}$$

$$c_t = f_t * c_{t-1} + i_t * g_t$$

$$h_t = o_t * f(c_t)$$

Use gates to control the information to be added from the input, forgot from the previous memories, and outputted.
σ and f are *sigmoid* and *tanh* function respectively, to map the value to [-1, 1]