Neural Networks

CS 6355: Structured Prediction

Based on slides and material from Geoffrey Hinton, Richard Socher, Dan Roth, Yoav Goldberg, Shai Shalev-Shwartz and Shai Ben-David, and others

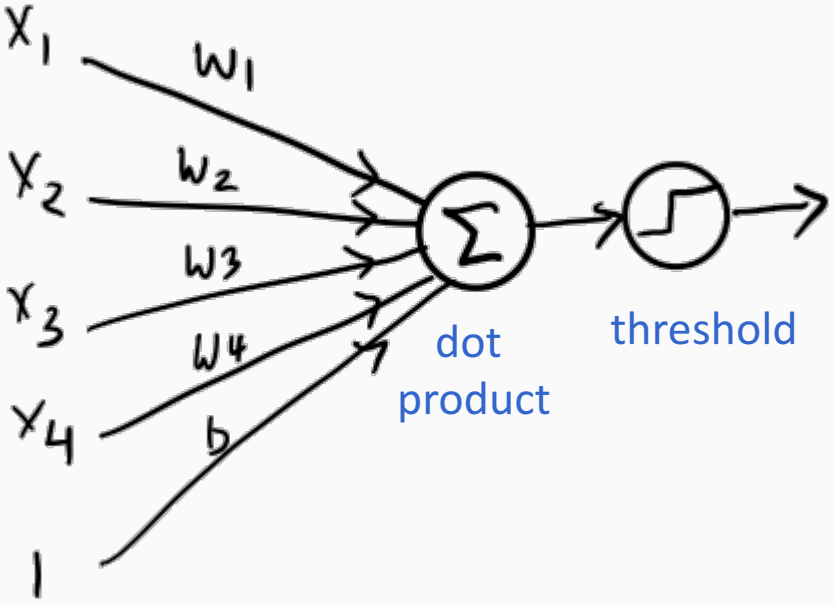
This lecture

- What is a neural network?
- Training neural networks
- Practical concerns
- Neural networks and structures

This lecture

- What is a neural network?
  - The hypothesis class
  - Structure, expressiveness
- Training neural networks
- Practical concerns
- Neural networks and structures

We have seen linear threshold units



Prediction

$$sgn\ (\boldsymbol{w}^T \boldsymbol{x} + b) = sgn(\sum w_i x_i + b)$$

Learning

various algorithms

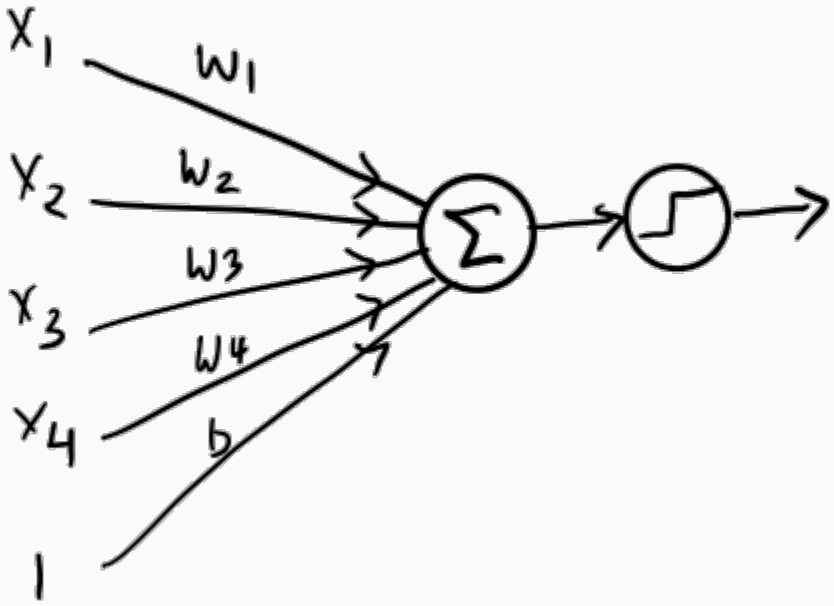
perceptron, SVM, logistic regression,…

in general, minimize loss

But where do these input features come from?

What if the features were outputs of another classifier?

Features from classifiers



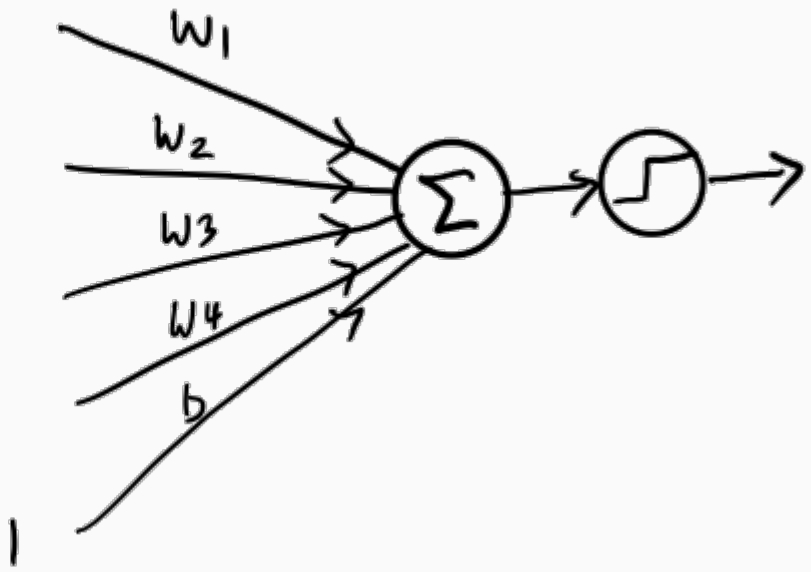
Features from classifiers

x_1

x_2

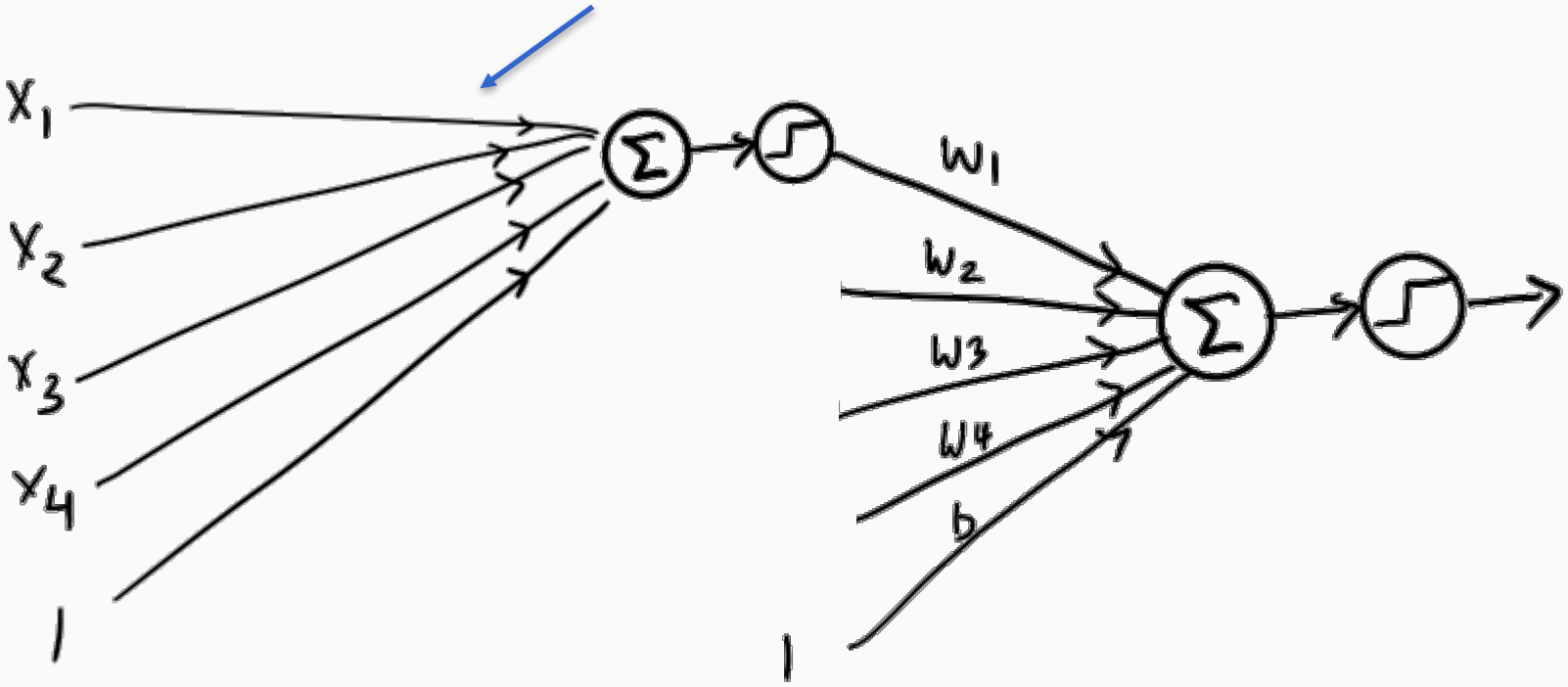
x_3

x_4

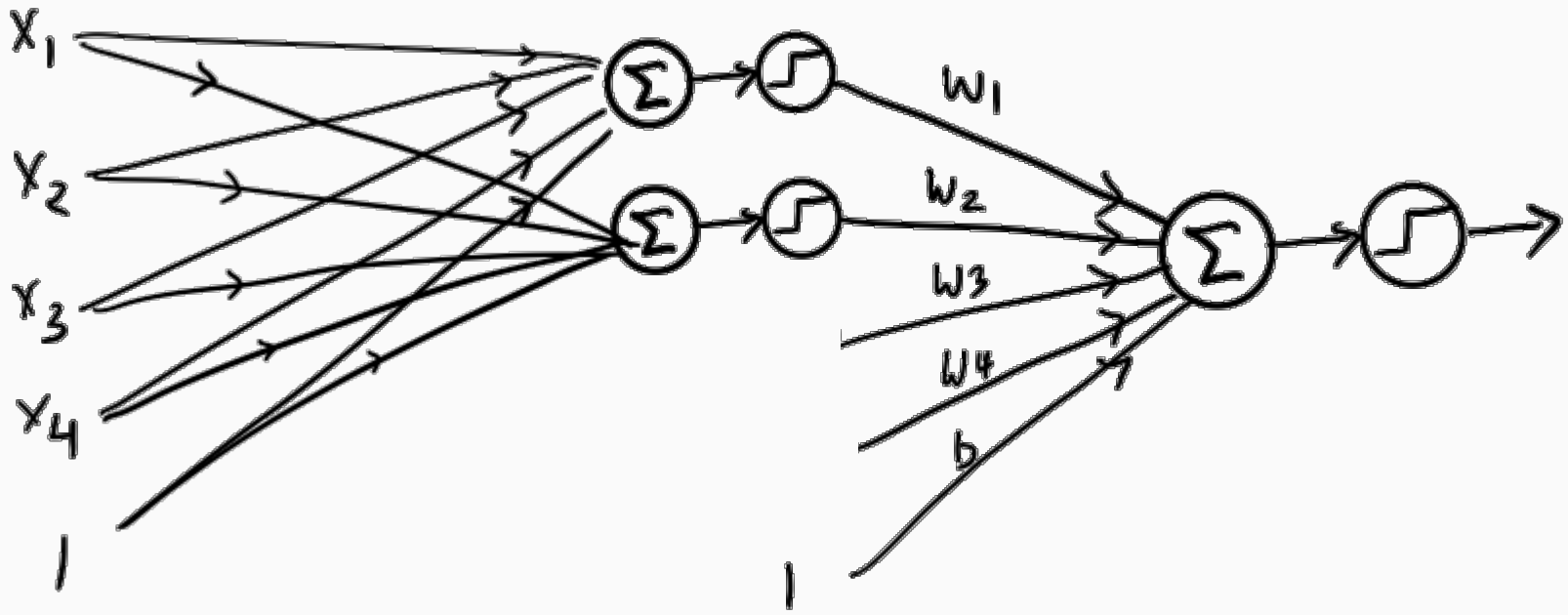


Features from classifiers

Each of these connections have their own weights as well

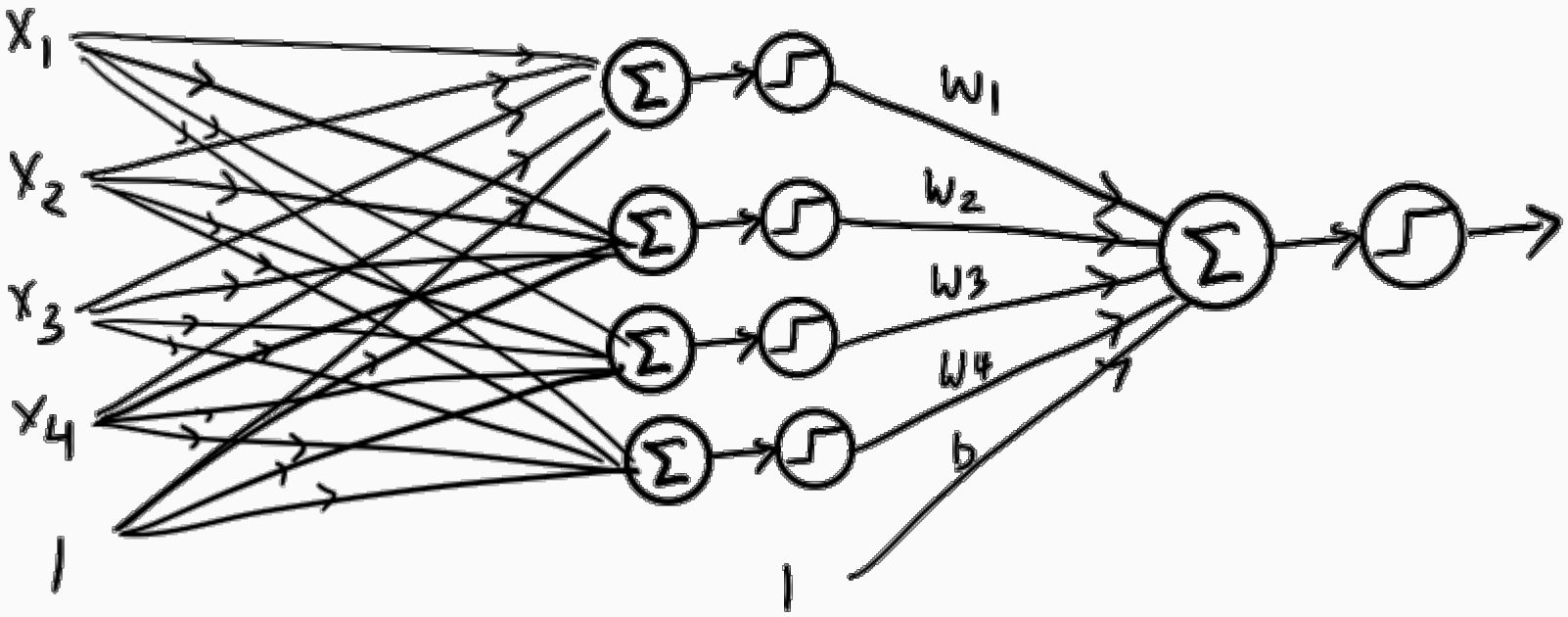


Features from classifiers



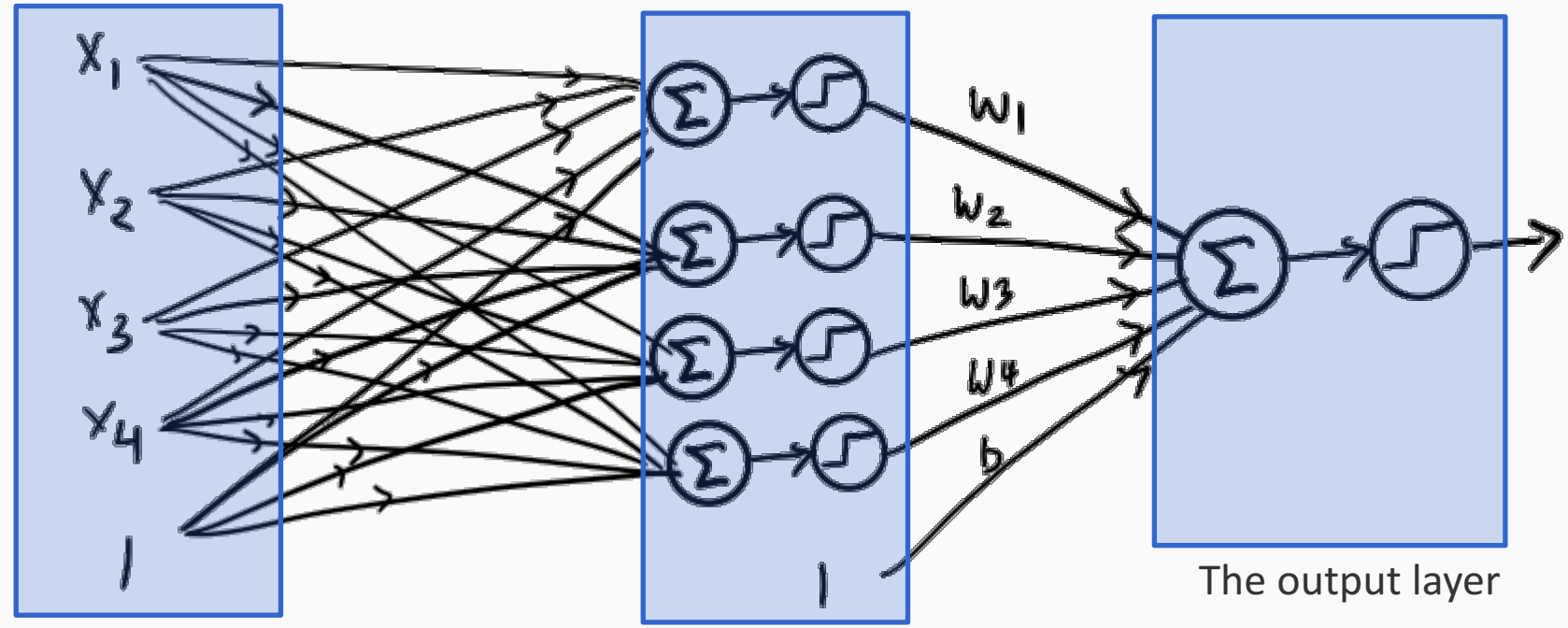
Features from classifiers

This is a **two layer** feed forward neural network



Features from classifiers

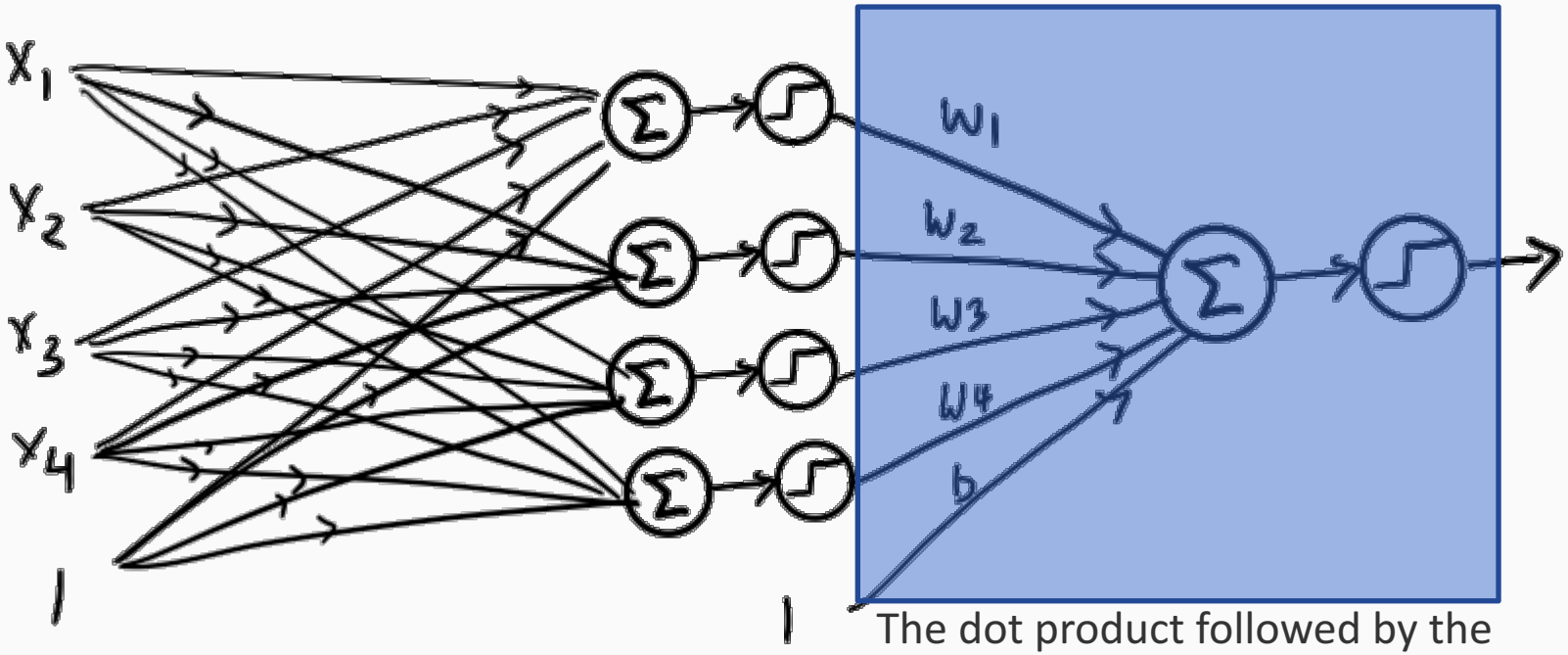
This is a **two layer** feed forward neural network



Think of the hidden layer as learning a good **representation** of the inputs

Features from classifiers

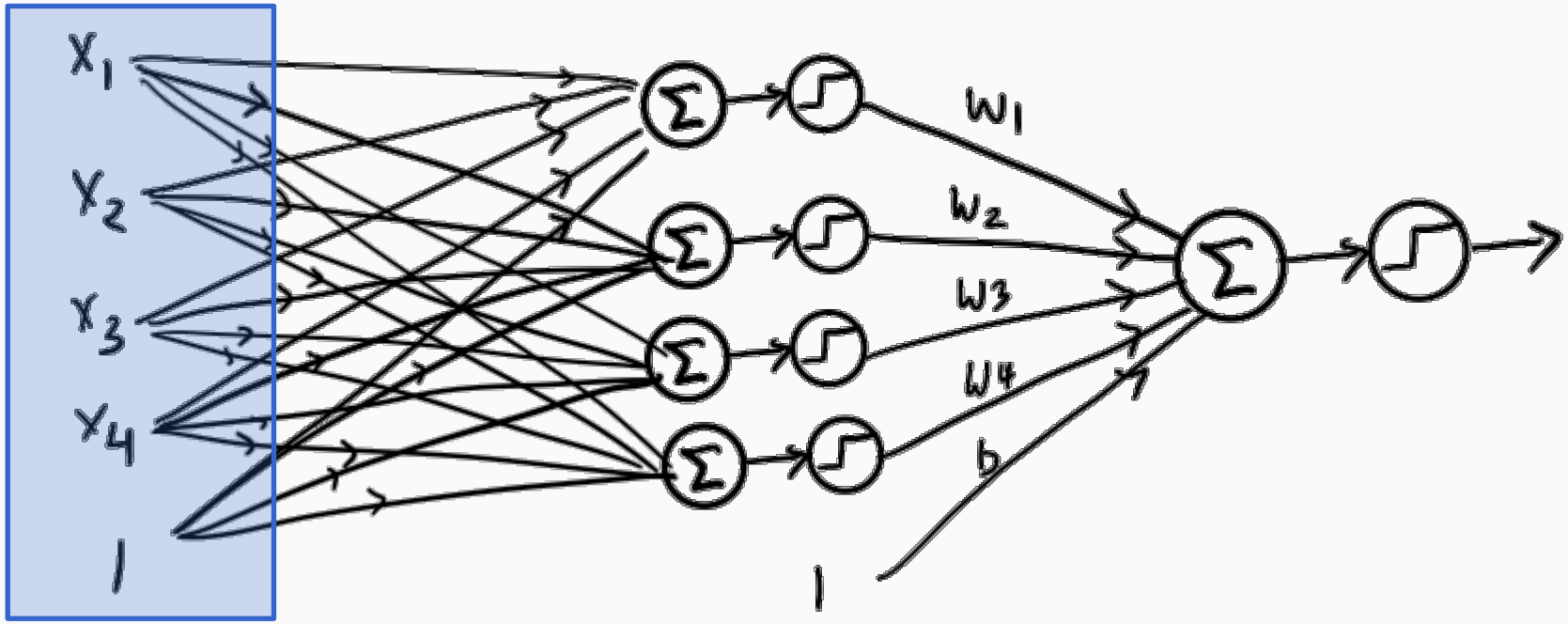
This is a **two layer** feed forward neural network



The dot product followed by the threshold constitutes a ***neuron***

Five neurons in this picture (four in hidden layer and one output)

But where do the inputs come from?



The input layer

What if the inputs were the outputs of a classifier?

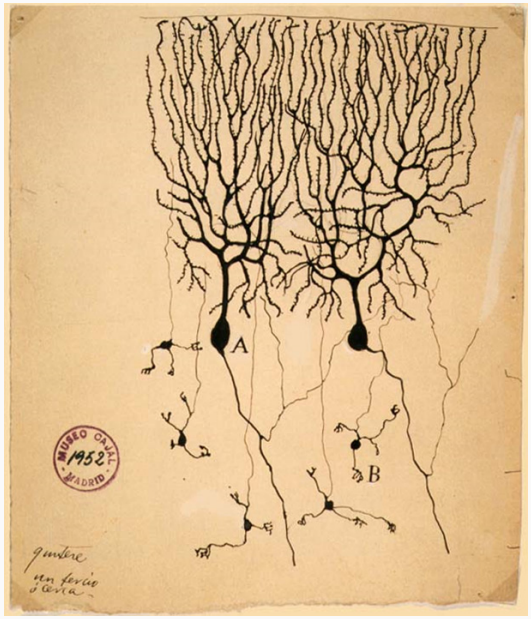
We can make a **three** layer network…. And so on.

Let us try to formalize this

Neural networks

- A robust approach for approximating real-valued, discrete-valued or vector valued functions
- Among the most effective **general purpose** supervised learning methods currently known
  - Especially for *complex and hard to interpret data* such as real-world sensory data
- The Backpropagation algorithm for neural networks has been shown successful in many practical problems
  - handwritten character recognition, speech recognition, object recognition, some NLP problems

Biological neurons

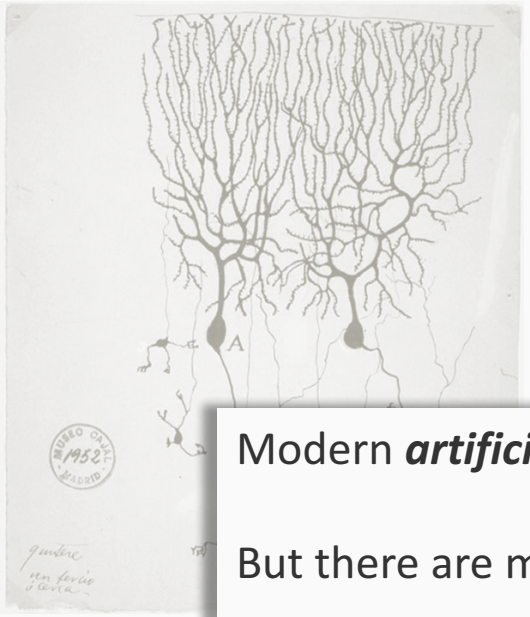


The first drawing of a brain cells by Santiago Ramón y Cajal in 1899

Neurons: core components of brain and the nervous system consisting of

1. Dendrites that collect information from other neurons
2. An axon that generates outgoing spikes

Biological neurons



The first d
cells by Sa
Cajal in 18

Neurons: core components of brain and the nervous system consisting of

1. Dendrites that collect information from other neurons
2. An axon that generates outgoing spikes

Modern ***artificial*** neurons are "inspired" by biological neurons

But there are many, many fundamental differences

Don't take the similarity seriously (as also claims in the news about the "emergence" of intelligent behavior)

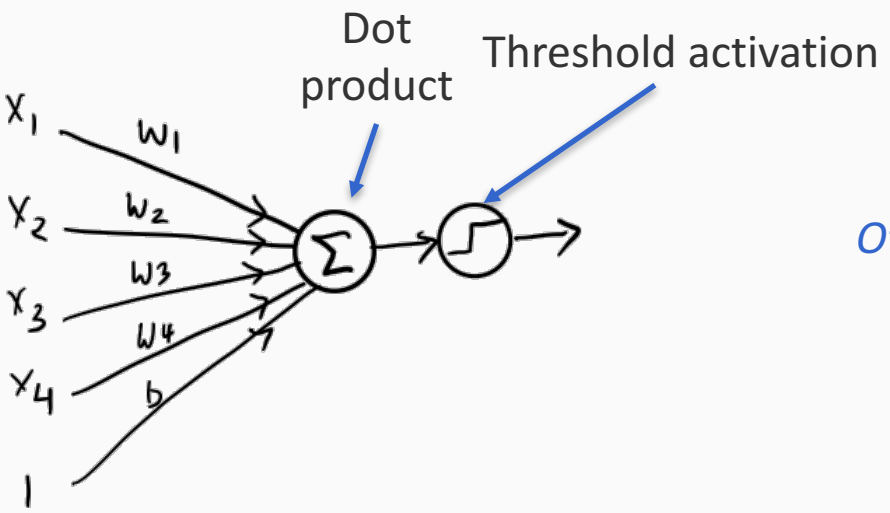
Artificial neurons

*Functions that **very loosely** mimic a biological neuron*

A neuron accepts a collection of inputs (a vector **x**) and produces an output by:

- Applying a dot product with weights **w** and adding a bias b
- Applying a (possibly non-linear) transformation called an ***activation***

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



Other activations are possible

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	$\text{sgn}(z)$
Sigmoid unit	$\dfrac{1}{1+\exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

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

A neural network

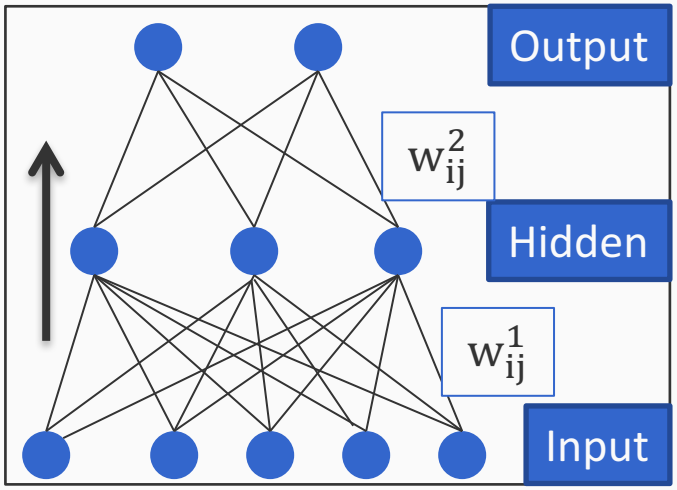
A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

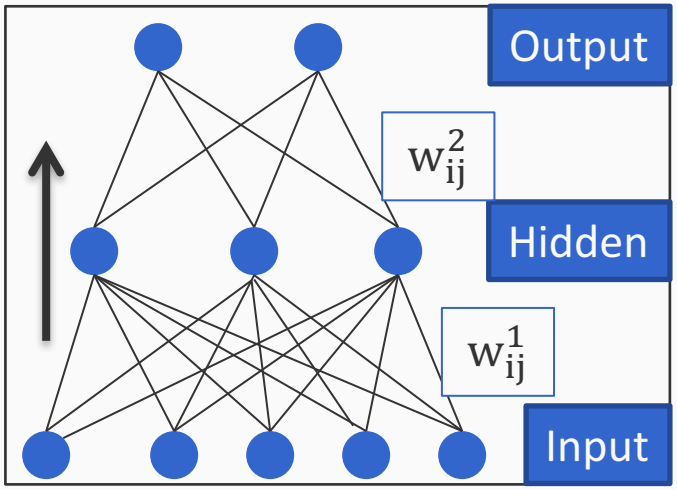
- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

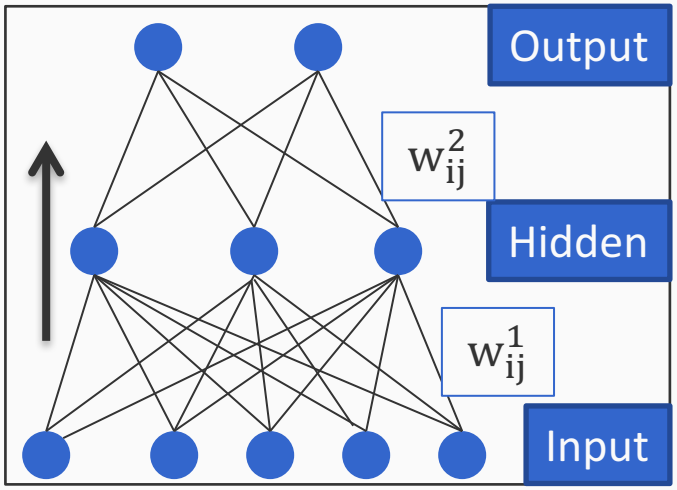


- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

A neural network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node

  Called the *architecture* of the network
  Typically predefined, part of the design of the classifier

  - The edge weights

  Learned from data

A brief history of neural networks

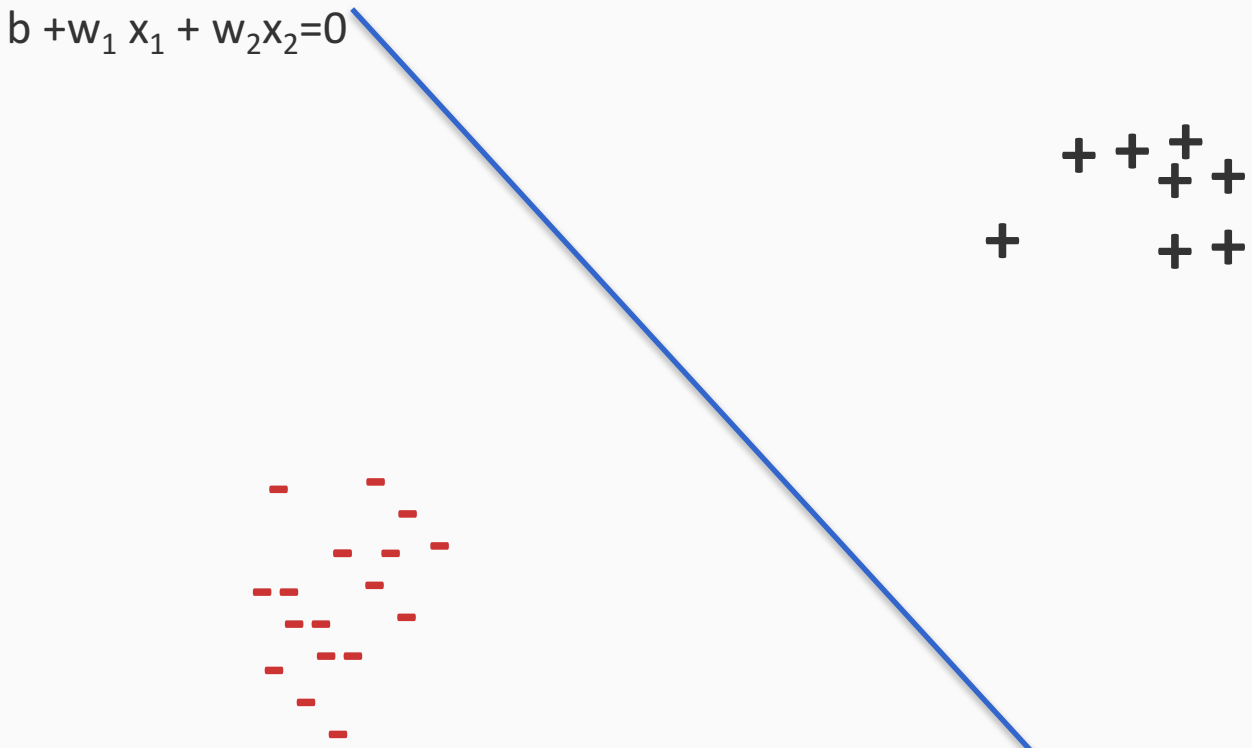
- 1943: McCullough and Pitts showed how linear threshold units can compute logical functions
- 1949: Hebb suggested a learning rule that has some physiological plausibility
- 1950s: Rosenblatt, the Peceptron algorithm for a single threshold neuron
- 1969: Minsky and Papert studied the neuron from a geometrical perspective
- 1980s: Convolutional neural networks (Fukushima, LeCun), the backpropagation algorithm (various)
- 2003-today: More compute, more data, deeper networks

See also: http://people.idsia.ch/~juergen/deep-learning-overview.html

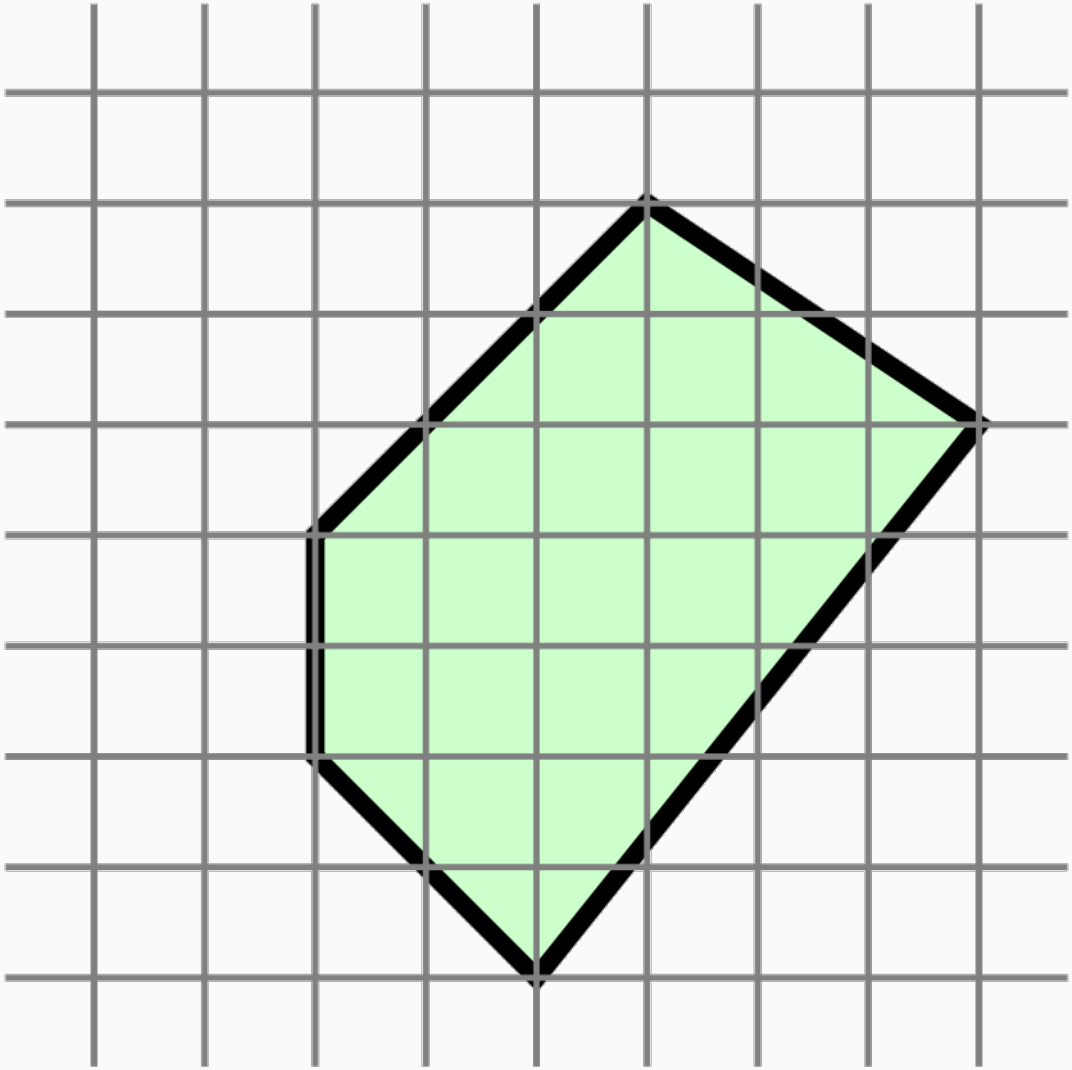
What functions do neural networks express?

A single neuron with threshold activation

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



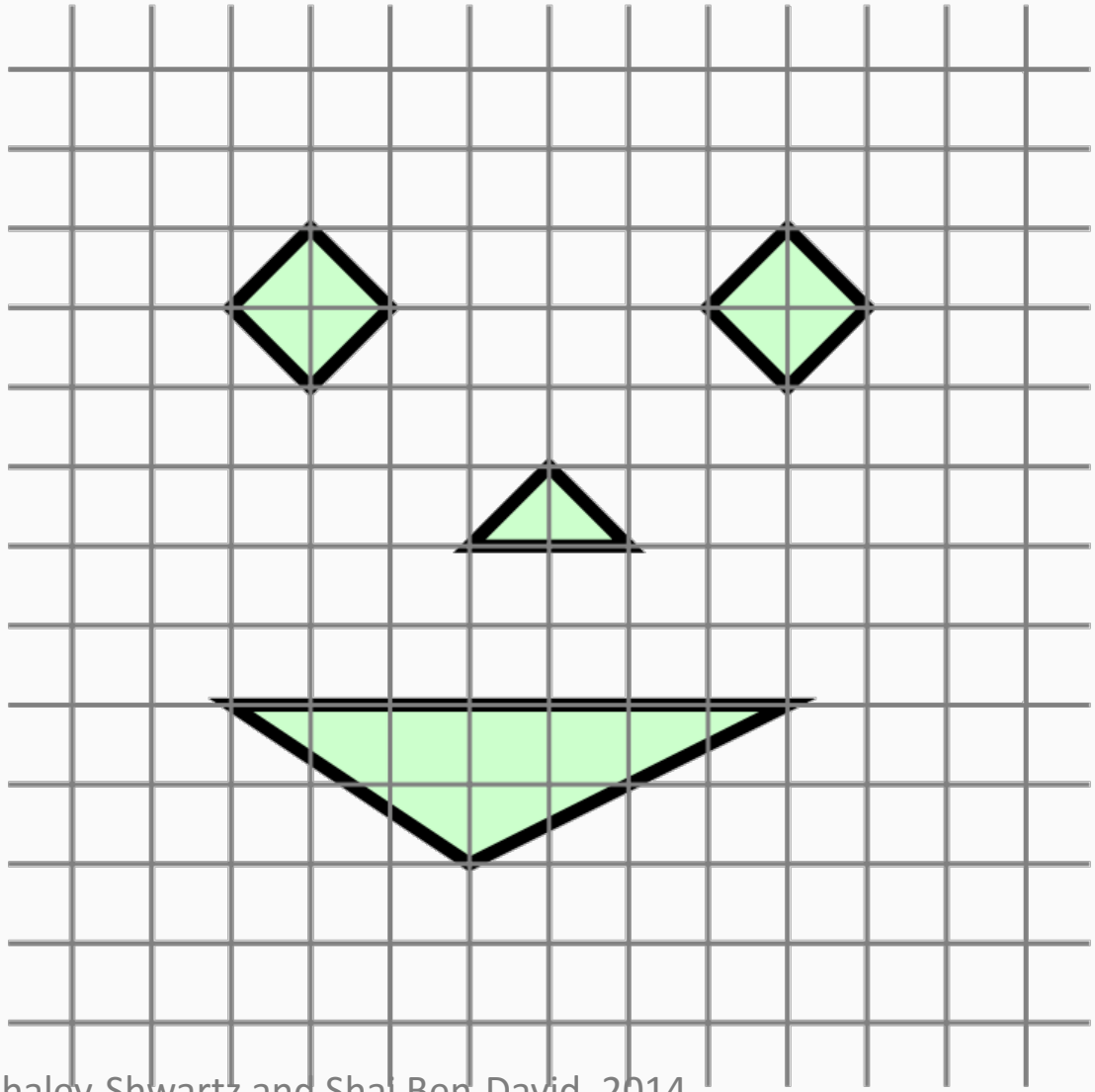
Two layers, with threshold activations



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

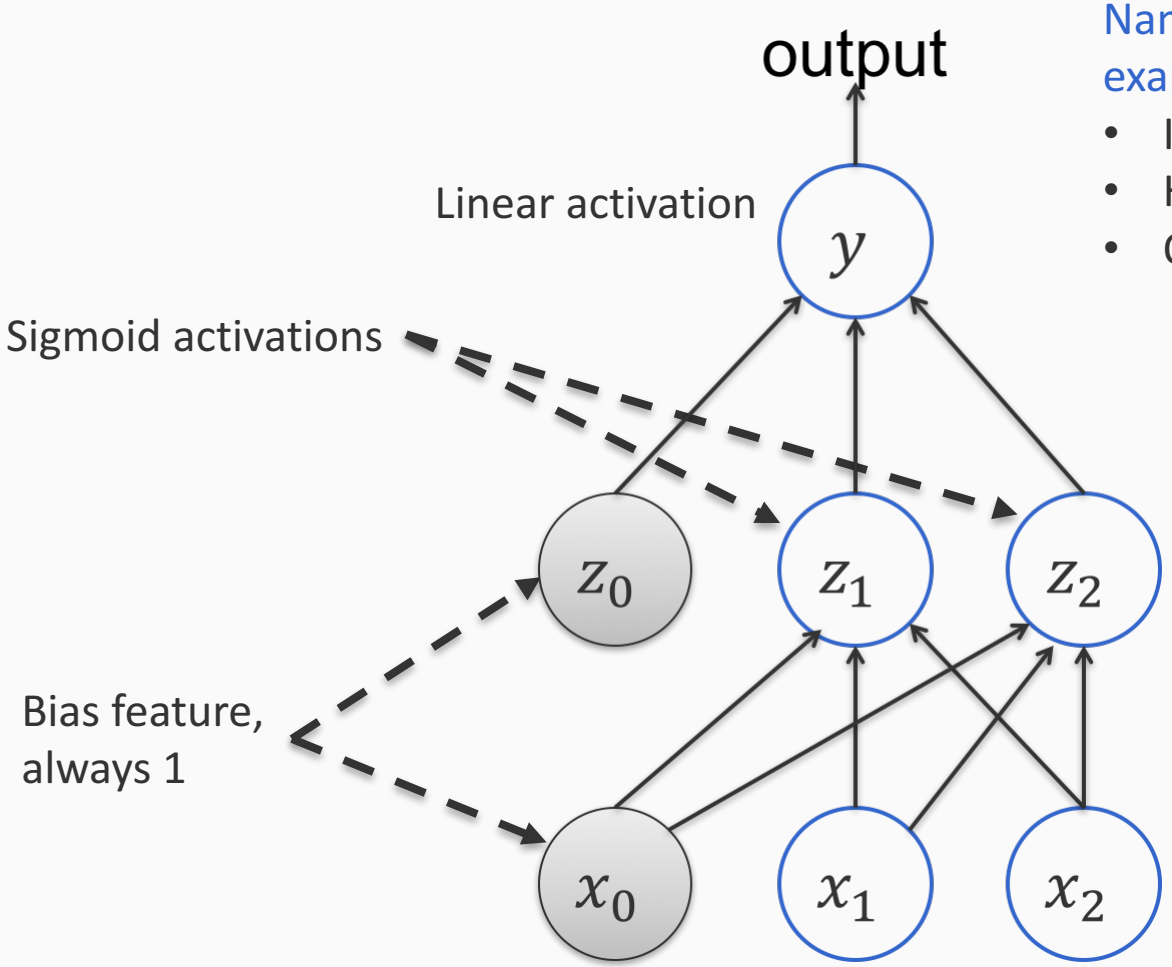
Figure from Shai Shalev-Shwartz and Shai Ben-David, 2014

Neural networks are universal function approximators

- Any continuous function can be approximated to arbitrary accuracy using one hidden layer of sigmoid units [Cybenko 1989]
- Approximation error is insensitive to the choice of activation functions [DasGupta et al 1993]
- Two layer threshold networks can express *any* Boolean function
  - **Exercise**: Prove this
- VC dimension of threshold network with edges E: $VC = O(|E| \log |E|)$
- VC dimension of sigmoid networks with nodes V and edges E:
  - Upper bound: $\mathrm{O}(|V|^2|E|^2)$
  - Lower bound: $\Omega(|E|^2)$

Exercise: Show that if we have only linear units, then multiple layers does not change the expressiveness

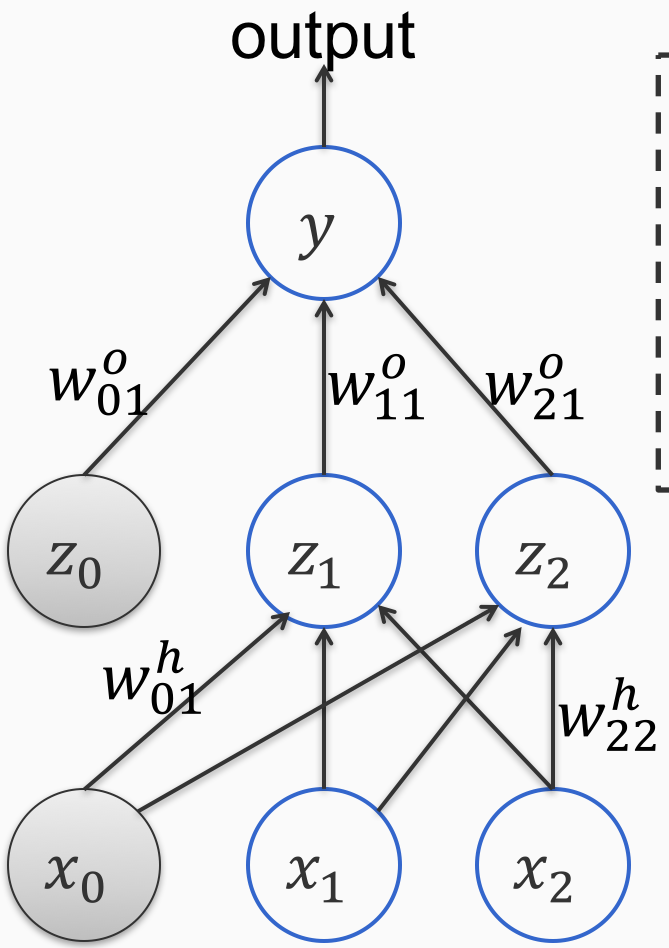
An example network



Naming conventions for this example

- Inputs: x
- Hidden: z
- Output: y

The forward pass



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

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

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

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

Questions?

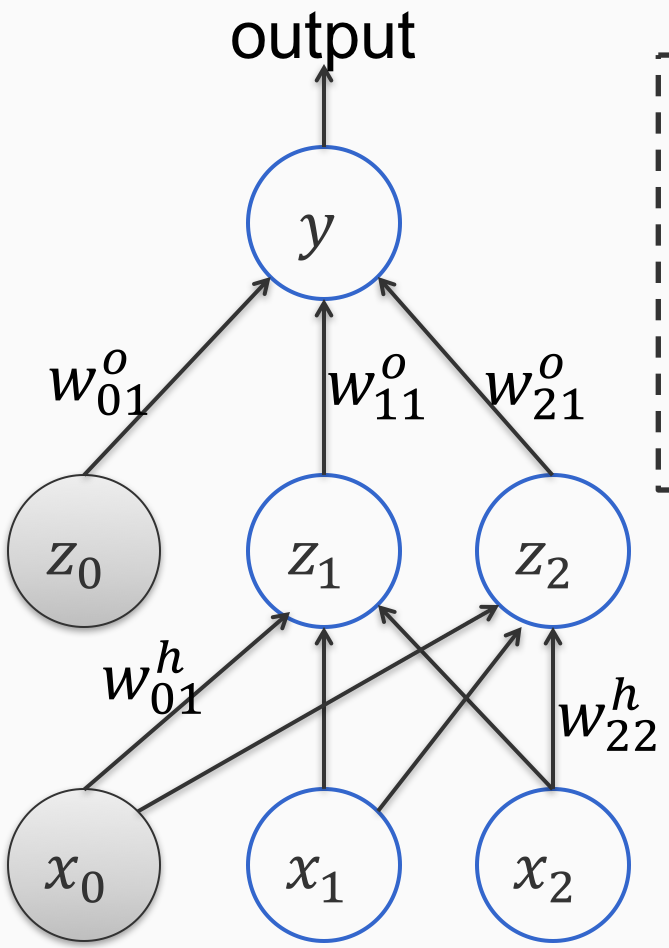
This lecture

- What is a neural network?
- Training neural networks
  - Backpropagation
- Practical concerns
- Neural Networks and Structures

Training a neural network

- Given
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples
    - $S = \{(\mathbf{x}_i, y_i)\}$
- The goal: Learn the weights of the neural network
- *Remember*: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction: $\hat{y} = NN(\boldsymbol{x}, \boldsymbol{w})$

Back to our running example



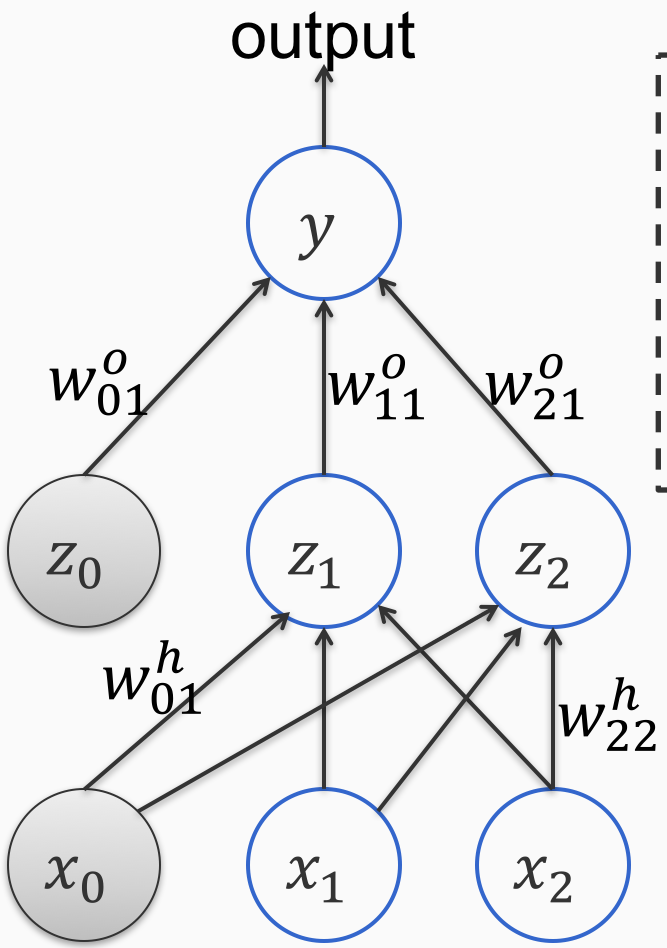
Given an input **x**, how is the output predicted

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^*

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

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

Perhaps with a *regularizer*

How do we solve the optimization problem?

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

Stochastic gradient descent

Given a training set S = $\{(\mathbf{x}_i, \mathrm{y}_i)\}$, $\mathbf{x} \in \Re^{\mathrm{d}}$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, \mathrm{y}_i) \in$ S:
      - 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

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

Stochastic gradient descent

Given a training set S = $\{(\mathbf{x}_i, \mathrm{y}_i)\}$, $\mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, \mathrm{y}_i) \in$ S:
      - 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

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

Checkpoint Where are we

If we have a neural network (structure, activations and weights), we can make a prediction for an input

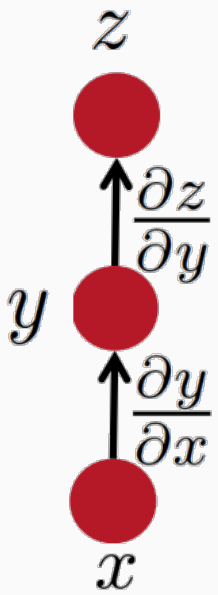
If we had the true label of the input, then we can define the loss for that example

If we can take the derivative of the loss with respect to each of the weights, we can take a gradient step in SGD

Questions?

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

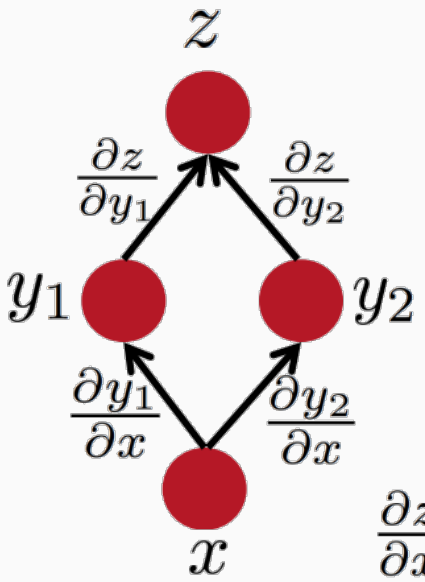


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Slide courtesy Richard Socher

Reminder: Chain rule for derivatives

- If z is (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 $\frac{\partial z}{\partial x}$

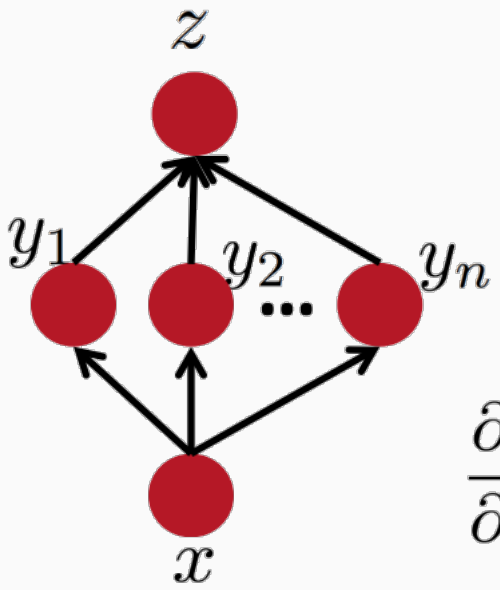


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Slide courtesy Richard Socher

Reminder: Chain rule for derivatives

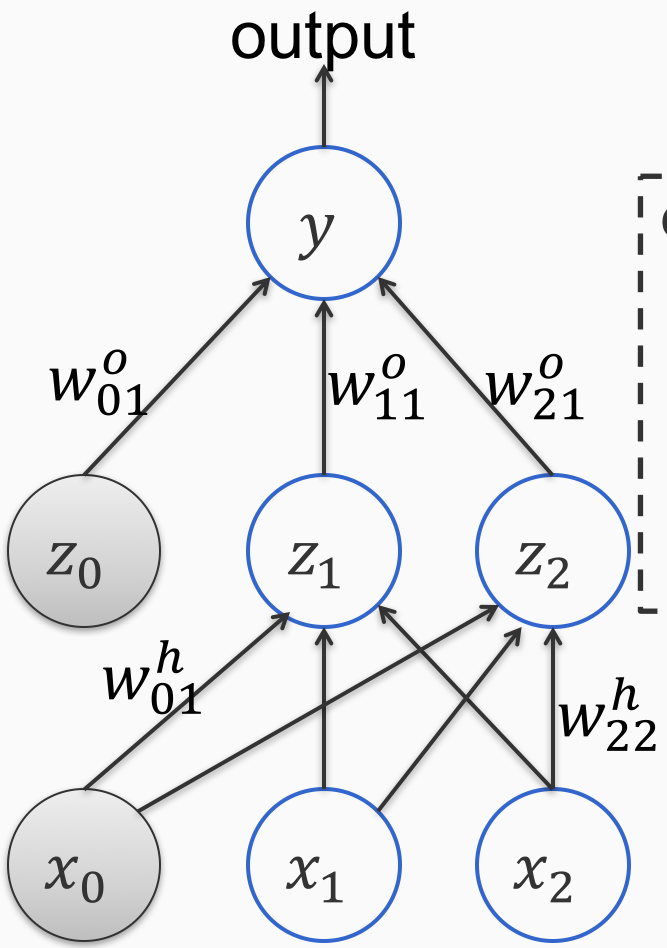
- If z is a sum of functions of y_i's, and the y_i's are functions of x
  - Then z is a function of x, as well
- Question: how to find $\frac{\partial z}{\partial x}$



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

Slide courtesy Richard Socher

Backpropagation



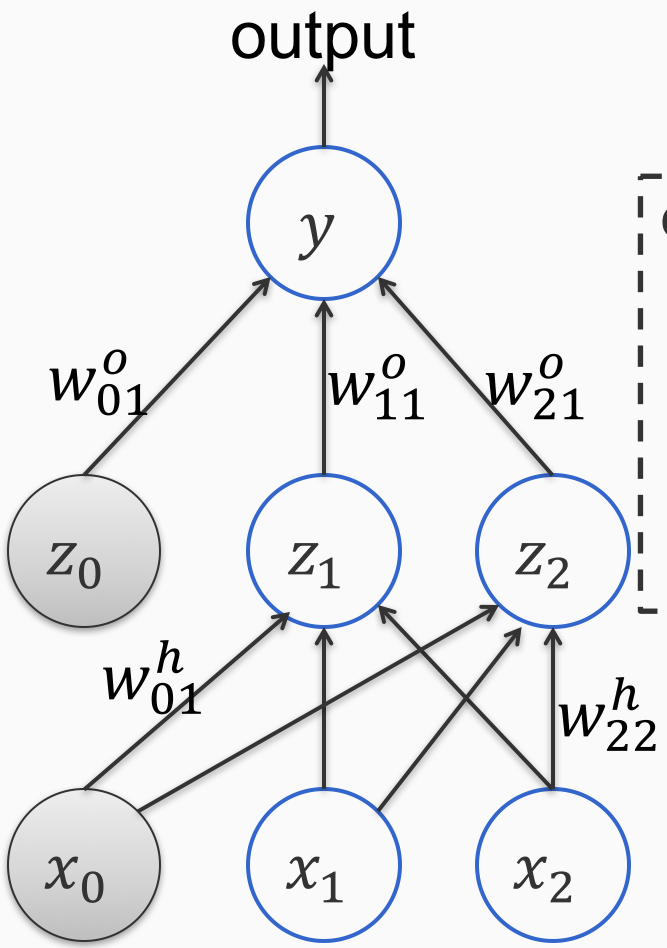
$$L = \frac{1}{2}(y - y^*)^2$$

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

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

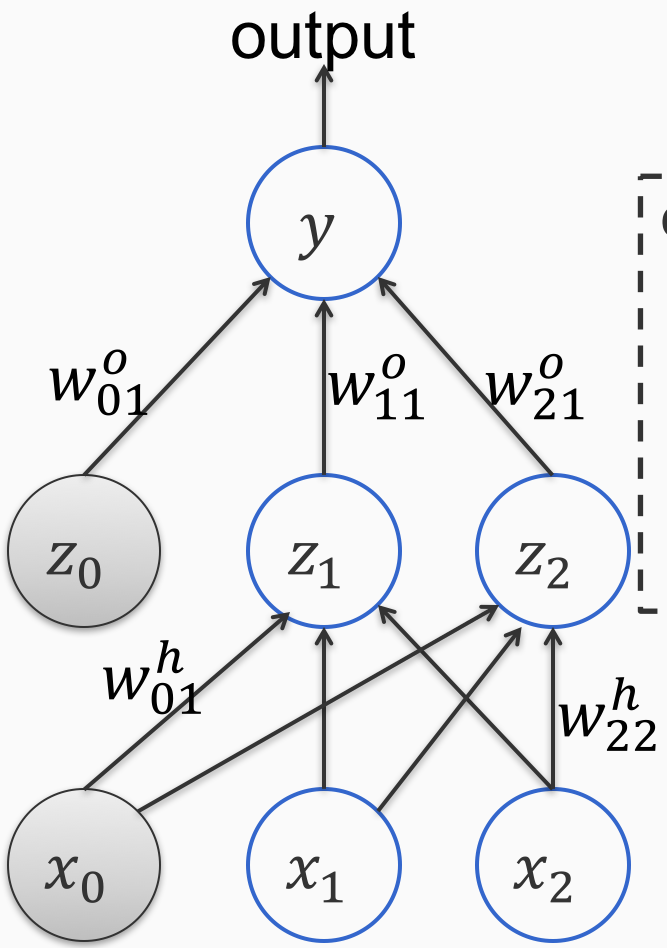
$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

We want to compute

$$\frac{\partial L}{\partial w_{ij}^{o}} \text{ and } \frac{\partial L}{\partial w_{ij}^{h}}$$

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

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

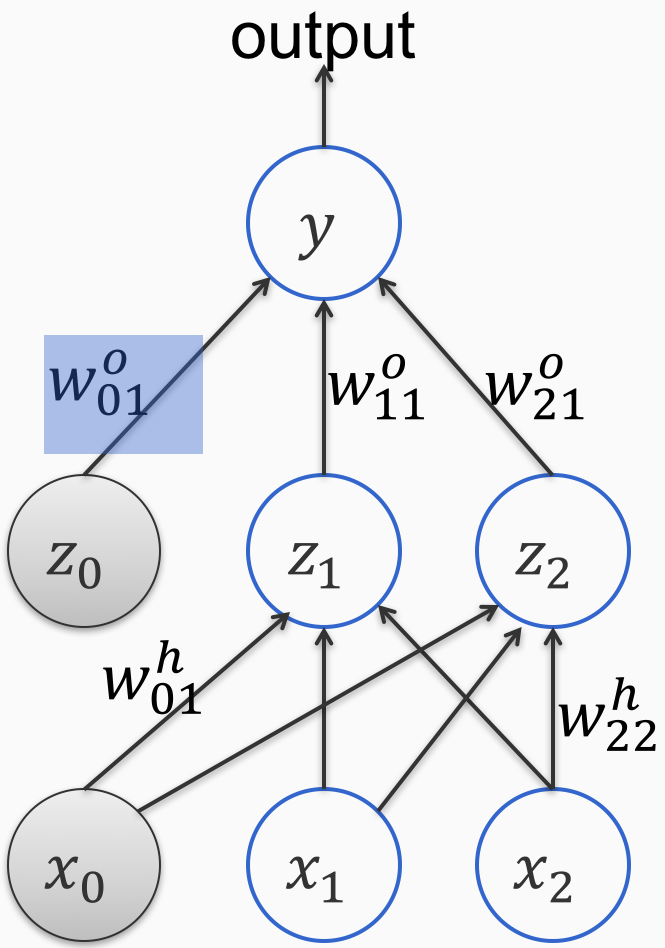
$$z_1 = \sigma(w_{01}^{h} + w_{11}^{h} x_1 + w_{21}^{h} x_2)$$

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

Output layer

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

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

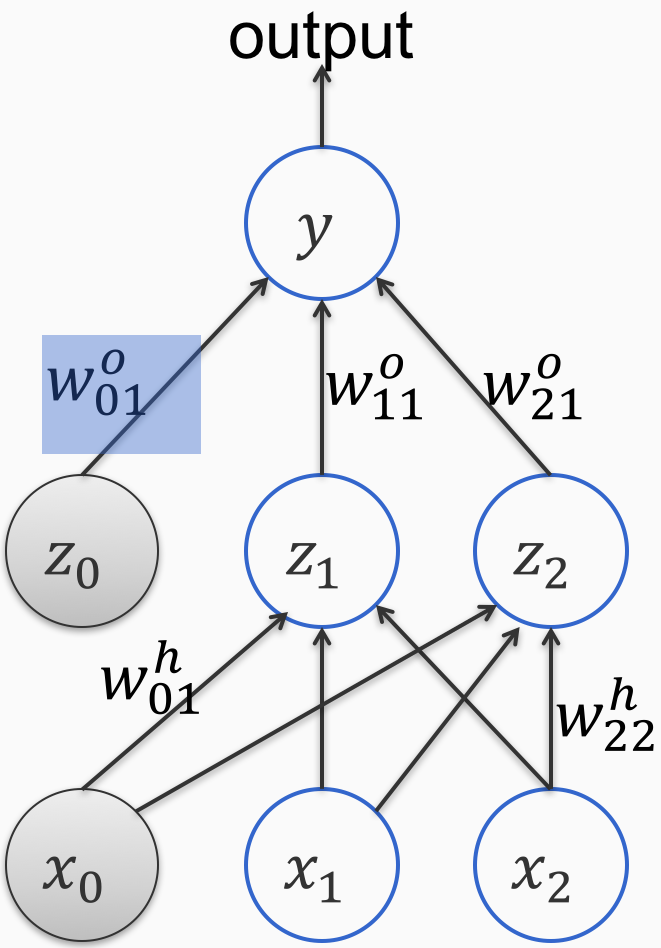


$$\frac{\partial L}{\partial w_{01}^o}$$

Output layer

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

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



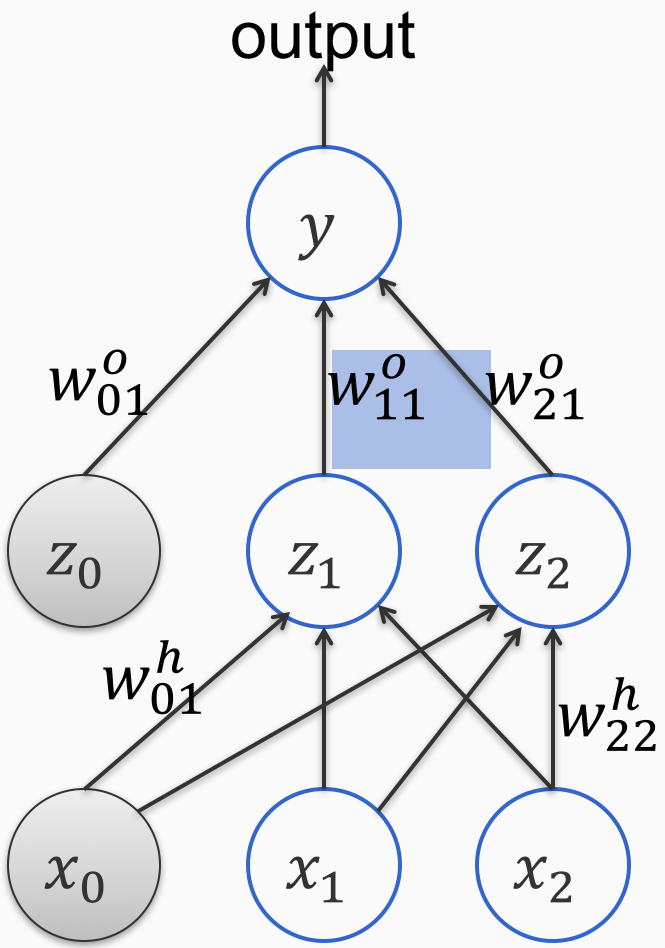
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^o}$$

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

Output layer

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

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

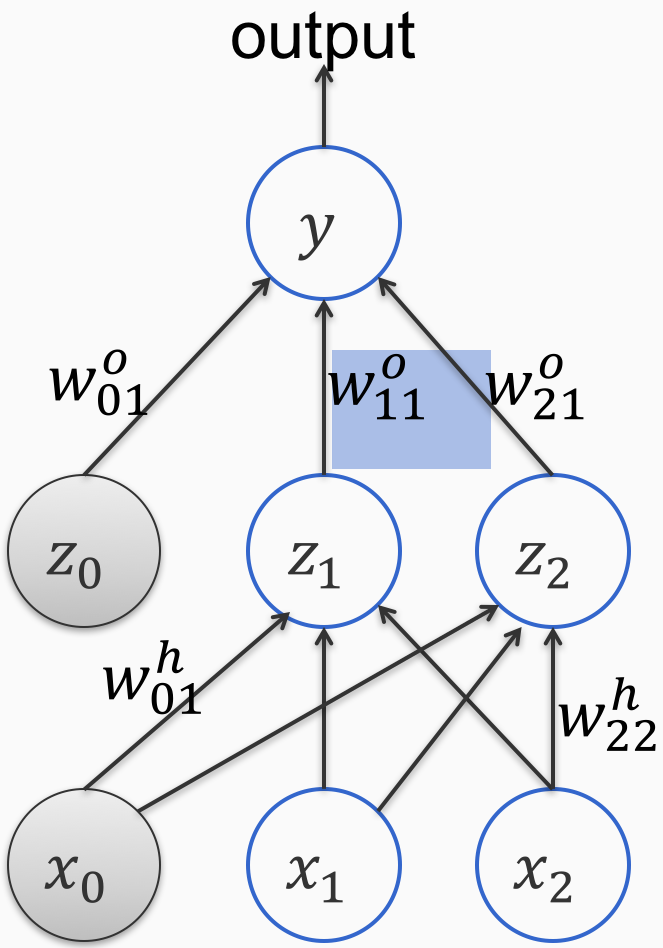


$$\frac{\partial L}{\partial w_{11}^o}$$

Output layer

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

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



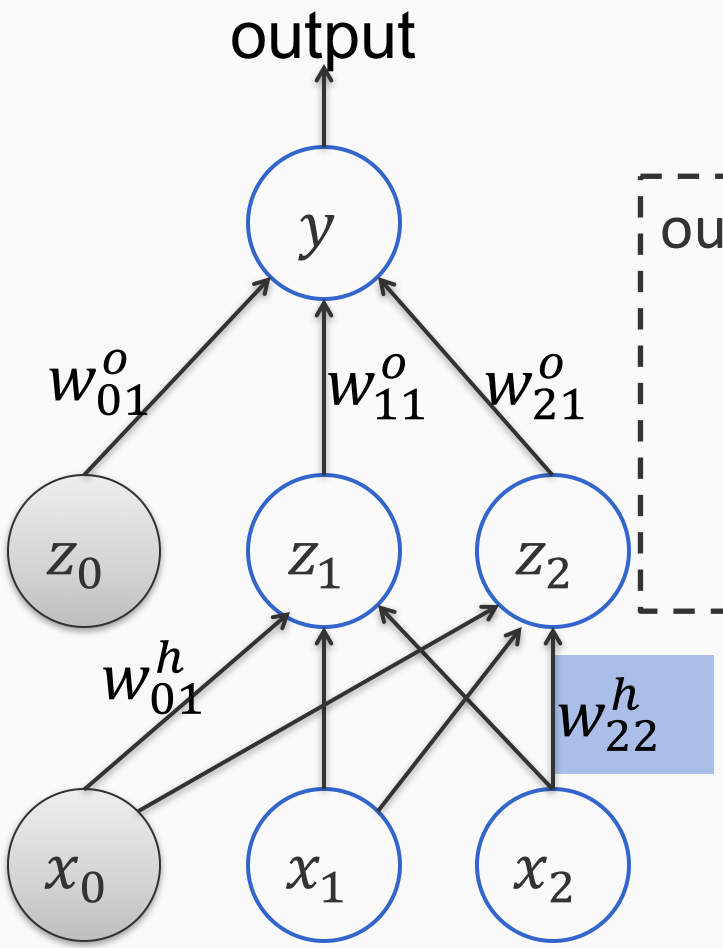
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^0}$$

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

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

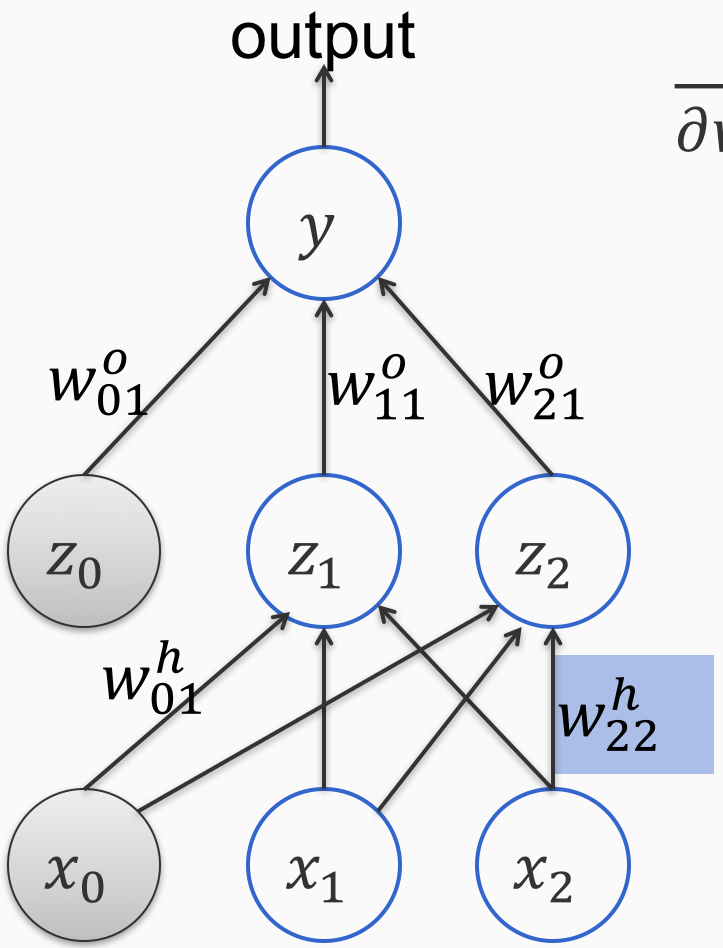
$$z_2 = \sigma(w_{02}^{h} + w_{12}^{h} x_1 + w_{22}^{h} x_2)$$

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

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

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

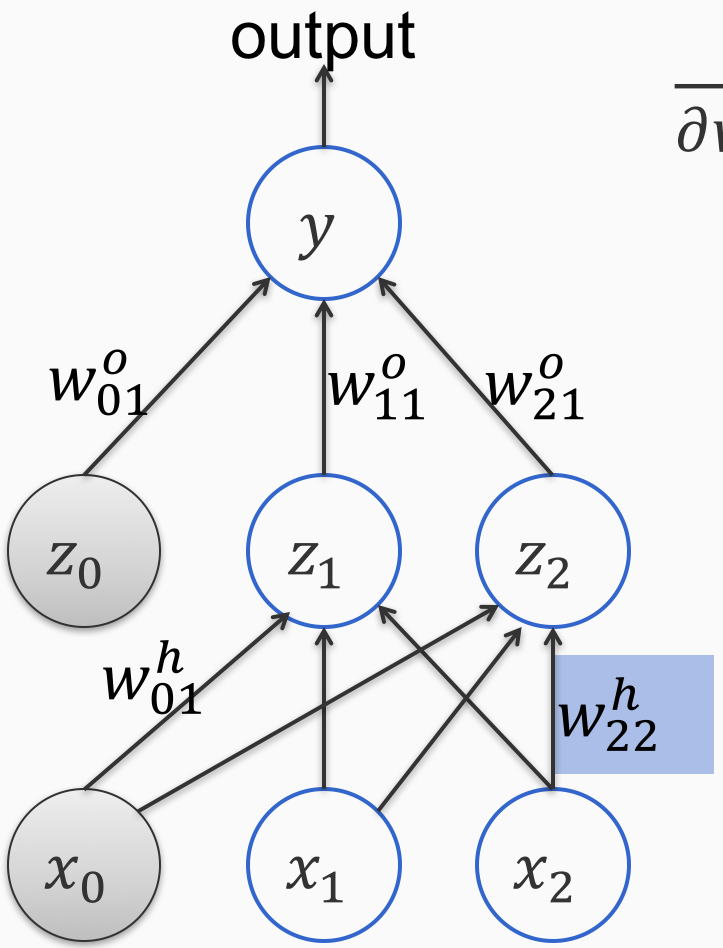
Hidden layer derivatives



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

Hidden layer

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

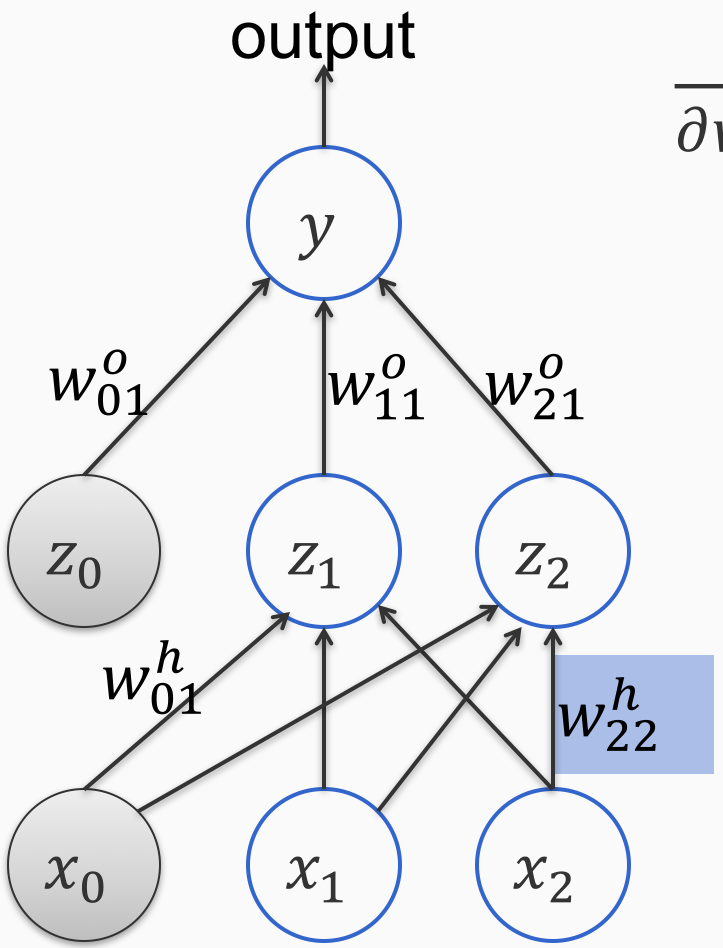


$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

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

Hidden layer

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



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

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

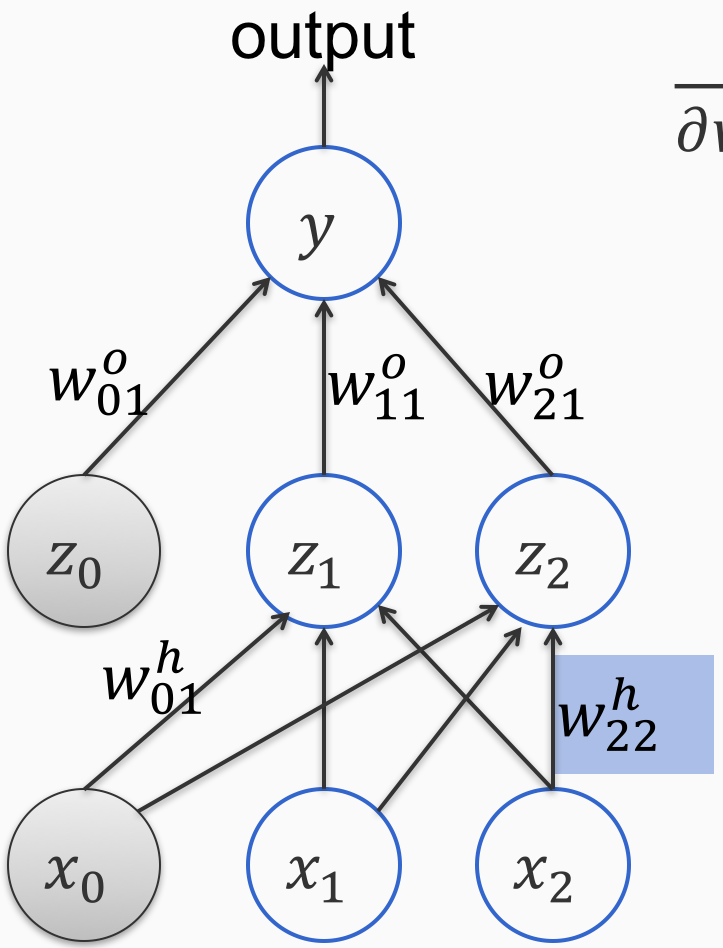
$$= \frac{\partial L}{\partial y}(w_{11}^o \frac{\partial}{\partial w_{22}^h} z_1 + w_{21}^o \frac{\partial}{\partial w_{22}^h} z_2)$$

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

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

Hidden layer

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



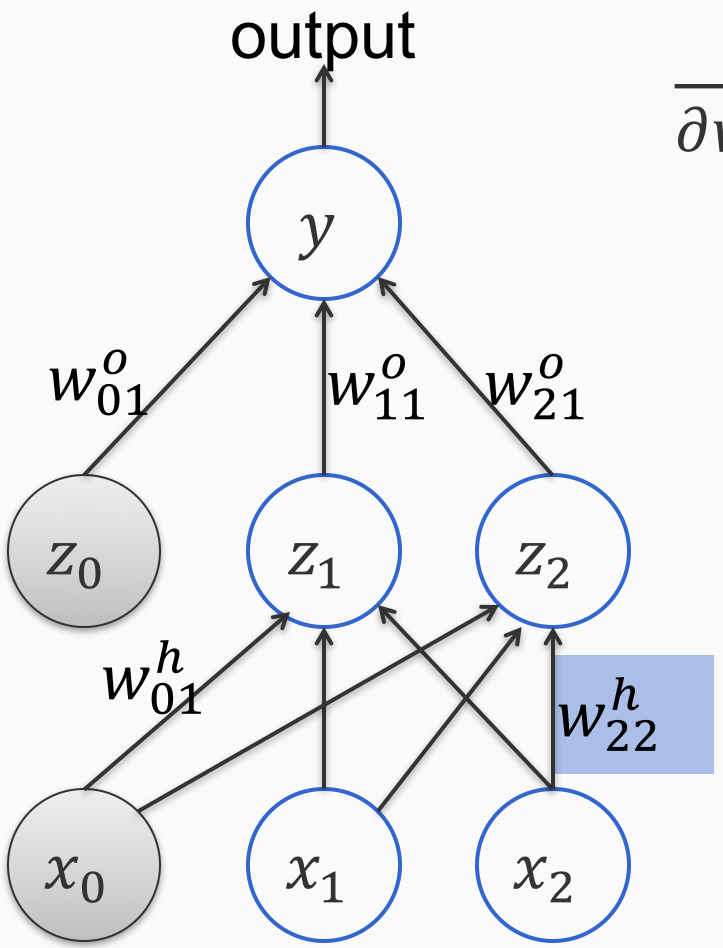
$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

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

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

Hidden layer

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



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

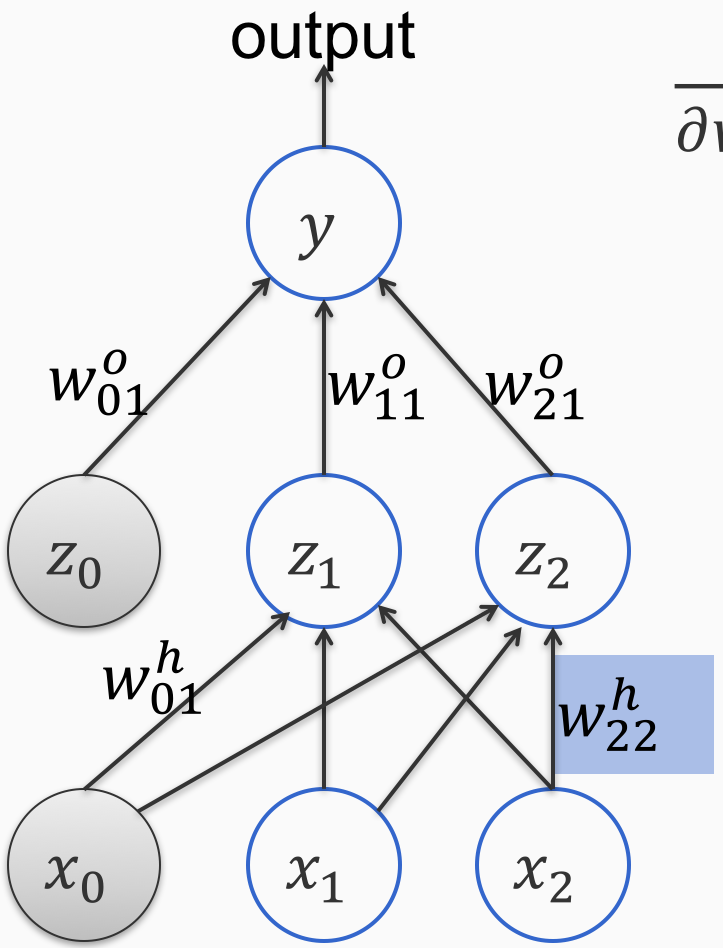
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

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

Hidden layer

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

Call this s



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

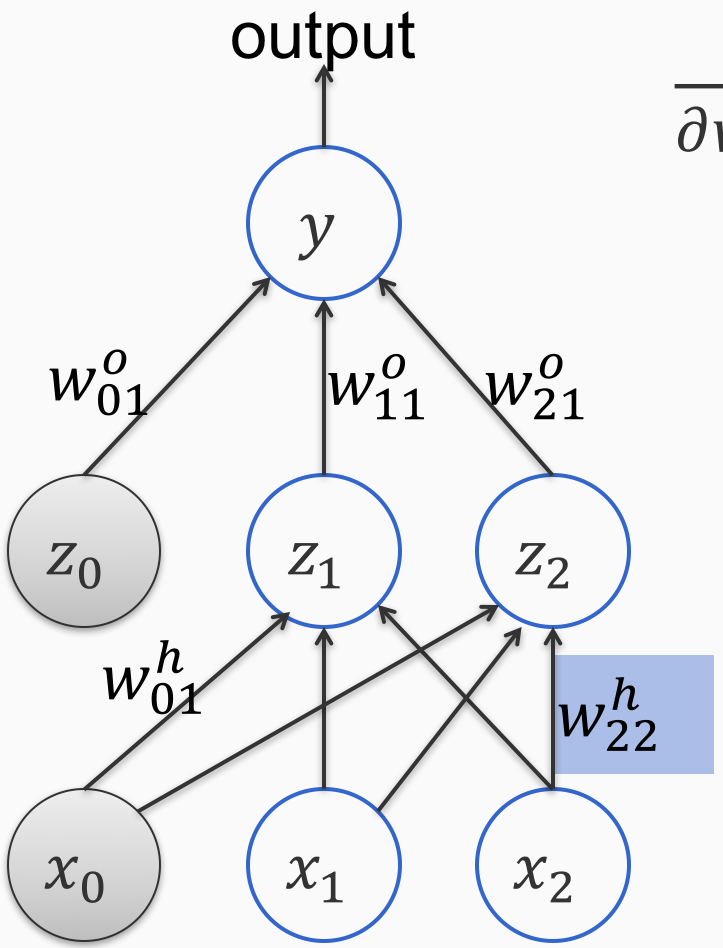
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

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

Hidden layer

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

Call this s



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

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

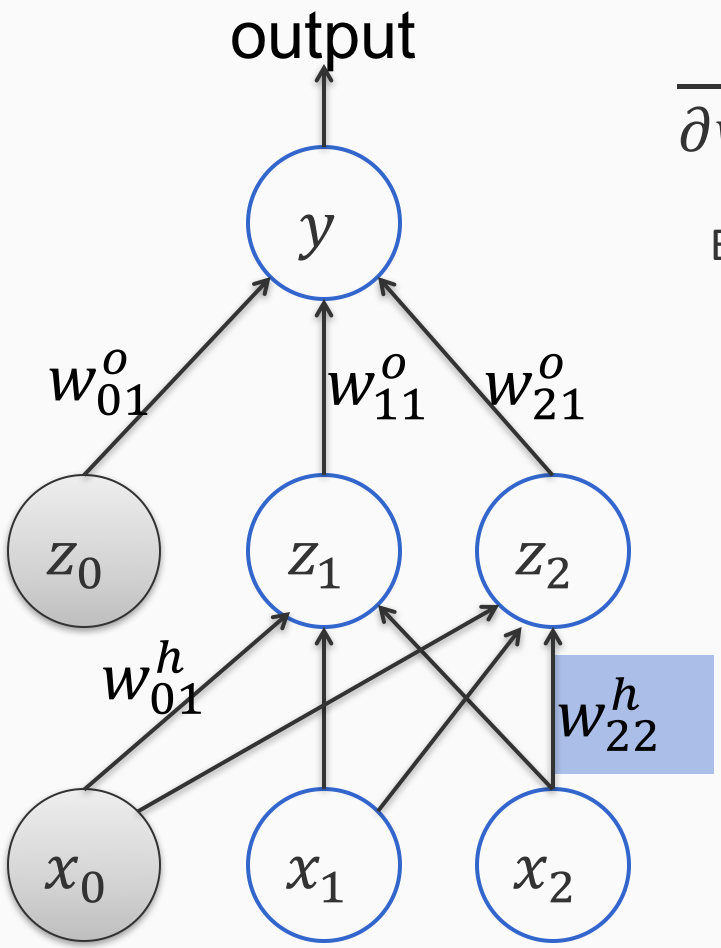
$$= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial w_{22}^h}$$

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

Hidden layer

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

Call this s



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

Each of these partial derivatives is easy

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

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

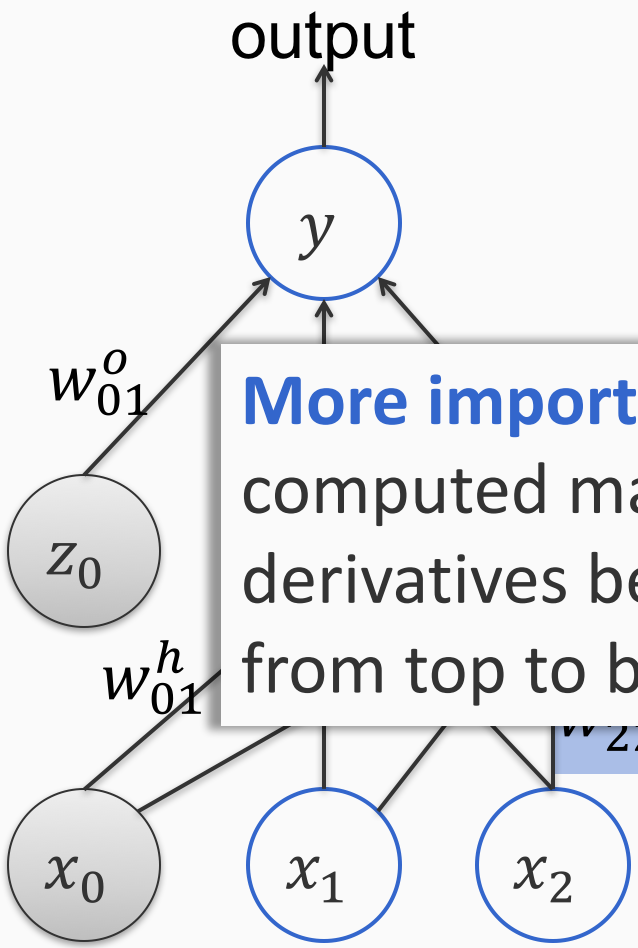
$$\frac{\partial s}{\partial w_{22}^h} = x_2$$

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

Hidden layer

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

Call this s



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

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

cause $z_2(s)$
istic
we have
een

The Backpropagation Algorithm

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

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

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

Stochastic gradient descent

Given a training set S = $\{(\mathbf{x}_i, \mathrm{y}_i)\}$, $\mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, \mathrm{y}_i) \in$ 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

The objective is not convex. Initialization can be important

γ_t: learning rate, many tweaks possible

The usual stochastic gradient descent tricks apply here

This lecture

- What is a neural network?
- Training neural networks
- Practical concerns
- Neural Networks and Structures

Practical concerns

1. Addressing problems with SGD
2. Preventing overfitting
3. Number of hidden layers

Training neural networks with SGD

- **No guarantee** of convergence**,** may oscillate or reach a local minima
- In practice, many large networks are trained on **large amounts of data** for realistic problems
- Many epochs (tens of thousands) may be needed for adequate training
  - Large data sets may require many hours of CPU or GPU time
  - Sometimes specialized hardware even
- **Termination criteria**: Number of epochs, Threshold on training set error, No decrease in error, Increased error on a validation set
- To **avoid local minima**: several trials with different random initial weights with majority or voting techniques

Preventing overfitting

- Running too many epochs may *over-train* the network and result in over-fitting
- Keep a **hold-out validation set** and test accuracy after every epoch
- Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond that
- To avoid losing training data to validation:
  - Use k-fold cross-validation to determine the average number of epochs that optimizes validation performance
  - Train on the full data set using this many epochs to produce the final results

Number of hidden units

- **Too few hidden units** prevent the system from adequately fitting the data and learning the concept.
- **Using too many hidden units** leads to over-fitting.
- Similar cross-validation method can be used to determine an appropriate number of hidden units.

This lecture

- What is a neural network?
- Training neural networks
- Practical concerns
- Neural Networks and Structures

What do neural networks bring us?

"Deep learning" is a combination of various modeling and optimization ideas

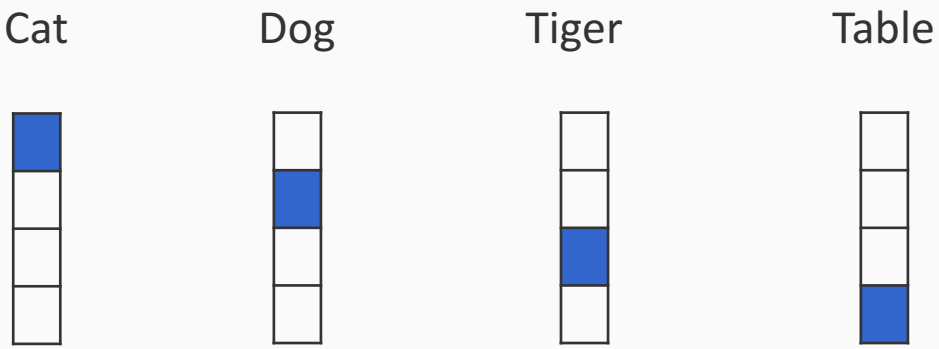
From our perspective, two important ideas stand out:

1. Neural networks for scoring outputs
   - Non-linear scoring functions
   - Much wider design space

2. ***Distributed representations***
   - Learned vector valued representations can coalesce superficially distinct objects
   - Eg: "cat" and "feline" share overlap in meaning, but

Why Distributed Representations

Think about feature representations

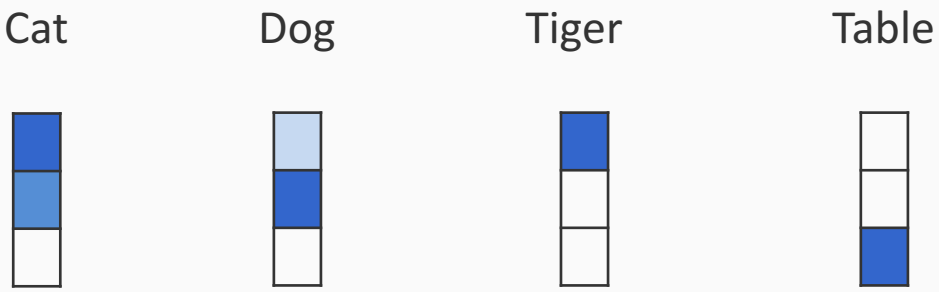


These vectors do not capture inherent similarities

Distances or dot products are all equal

Why Distributed Representations

Think about feature representations



Dense vector (often lower dimensional) representations can capture similarities better

Neural Networks in the age of structures

How can we exploit expressive scoring functions and distributed representations for structures?

Ideas?

Some possible approaches

- Treat neural networks as graphical models
  - Each neuron with a sigmoid activation function expresses a probability distribution over a single bit
  - This approach gives us Restricted Boltzmann Machines
- Adapt standard conditional random fields to use distributed representations
- Treat neural networks as simple scoring functions
  - We can still do inference on over the neural networks
  - For eg: Greedy inference over a sequence
  - Or perhaps more complex inference
    - Open question

Recurrent Neural Networks

Long Short-Term Memories and its siblings

Predicting sequences

https://karpathy.github.io/2015/05/21/rnn-effectiveness/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural networks are prediction machines

We can assign labels to inputs

cat

burrito

But what if the label to an input depends on a previous state of the network?

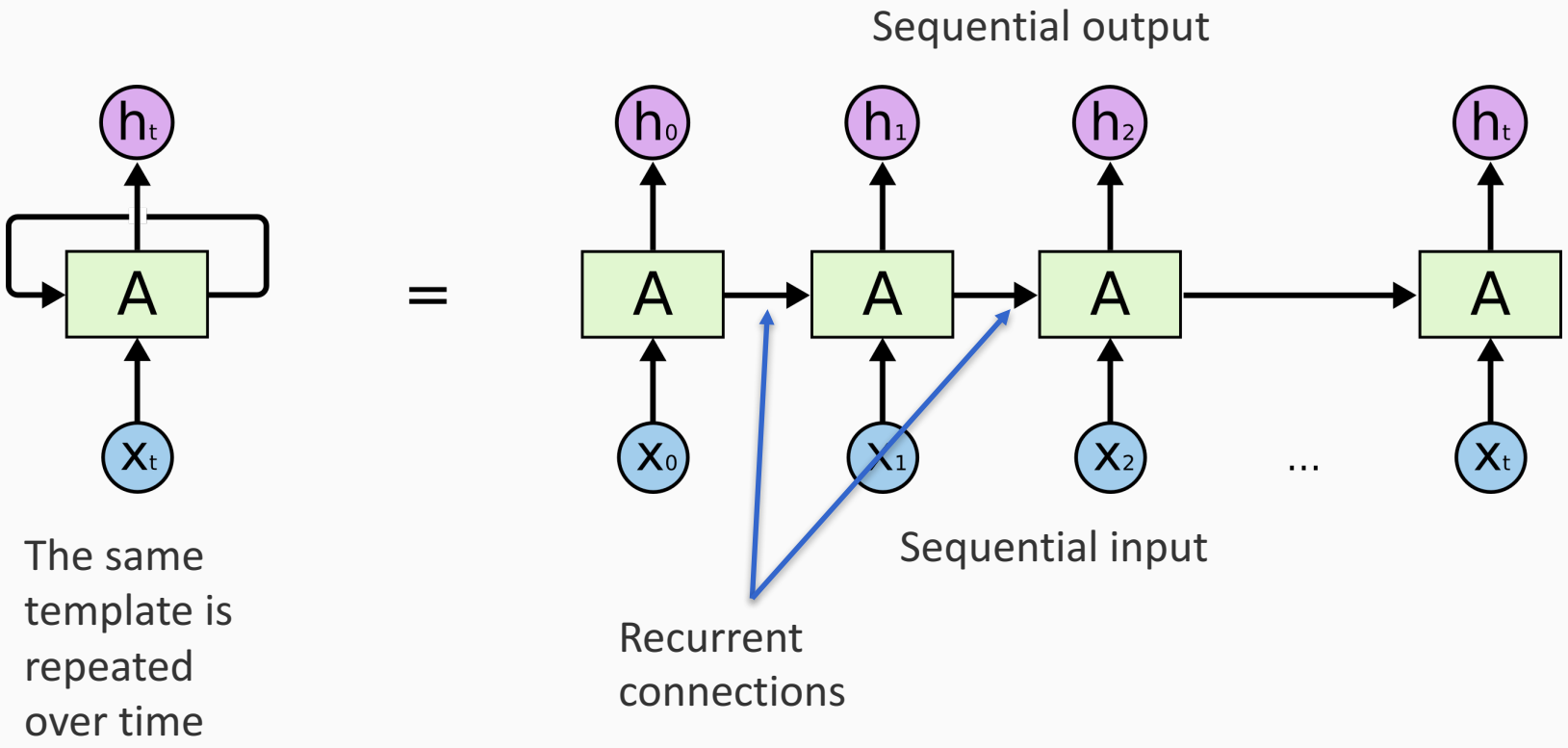
Vanilla neural networks

1. Do not have persistent memory
2. Can not deal with varying sized inputs

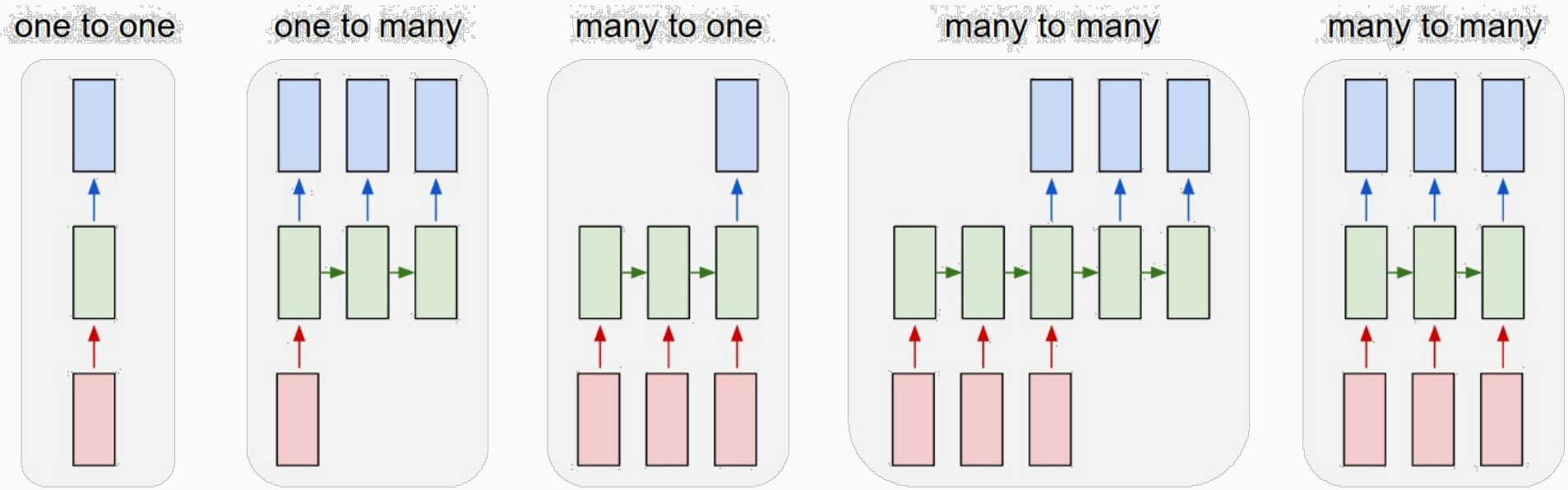
Sequential prediction: Examples

- Language models: “It was a dark and stormy _____”
  - Constructing sentences automatically requires us to remember what we constructed before
- Speech recognition
  - Convert a sequence of audio signals to words
  - The word at time t may depend on what word was predicted at time (t-1)
- Event extraction from movies
  - Watch a movie and predict what events are happening
  - The events at a particular scene probably depends on both the video signal ***and*** the events that were predicted in the previous scene
- ….. Many more examples

Recurrent Neural Networks: Networks with "loops"



Various configurations possible



Vanilla networks

Sequence output (eg: image captioning)

Sequence input (eg: sentiment analysis)

Seq2seq (eg: translation)

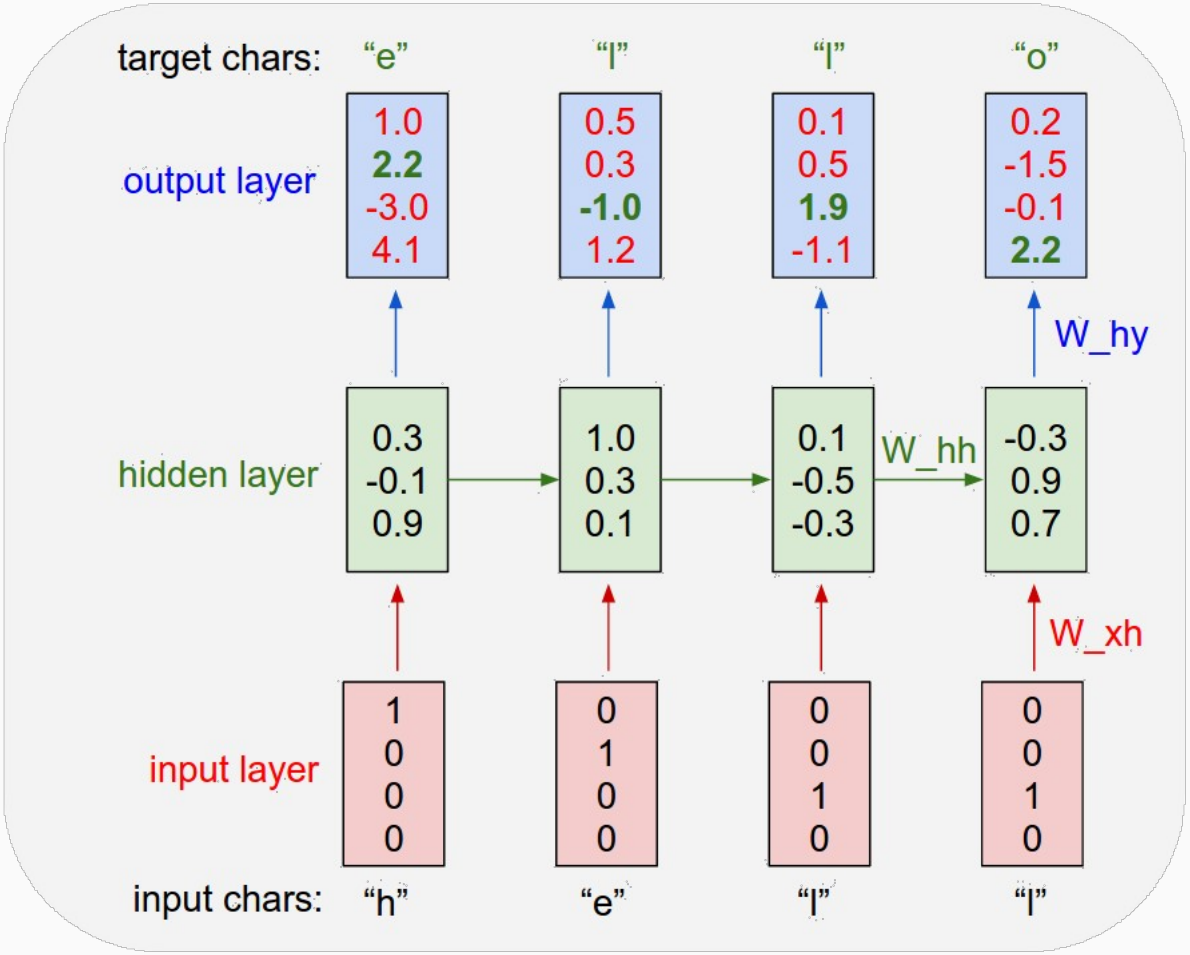
Insides of an RNN

Each recurrent neuron has maintains a **state** vector (h) that it updates

Forward pass:

1. Accept input **x**
2. Update $\mathbf{h}^{t+1} = activation(\boldsymbol{w_1} \cdot \boldsymbol{h}^t + \boldsymbol{w_2} \cdot \boldsymbol{x})$
3. Produce $output = activation(\boldsymbol{w_o} \cdot \boldsymbol{h^t})$

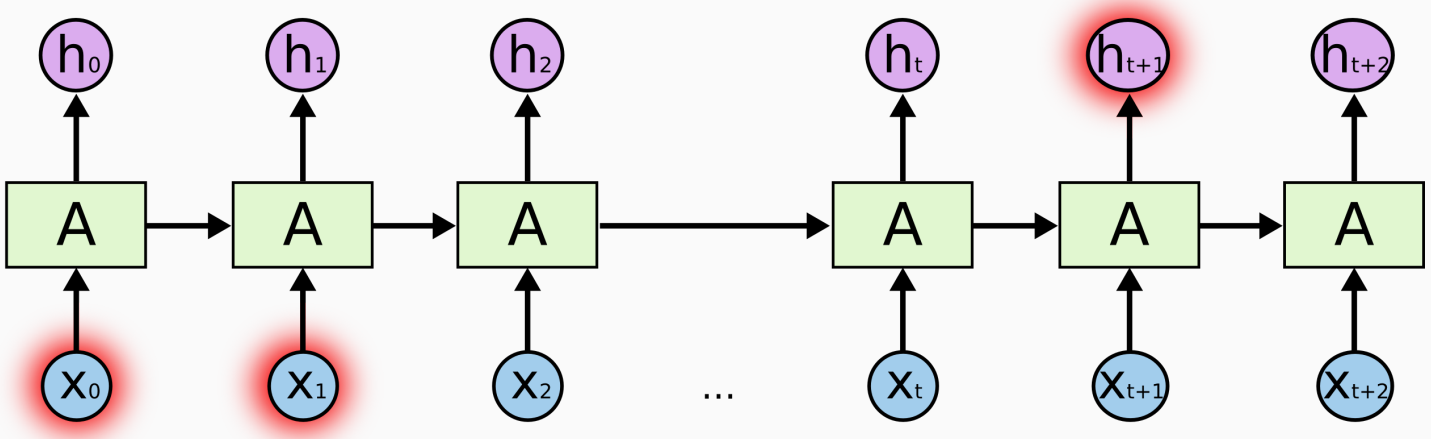
An example: Character level language model



The problem: Vanishing gradient

RNNs are particularly prone to the vanishing gradient problem

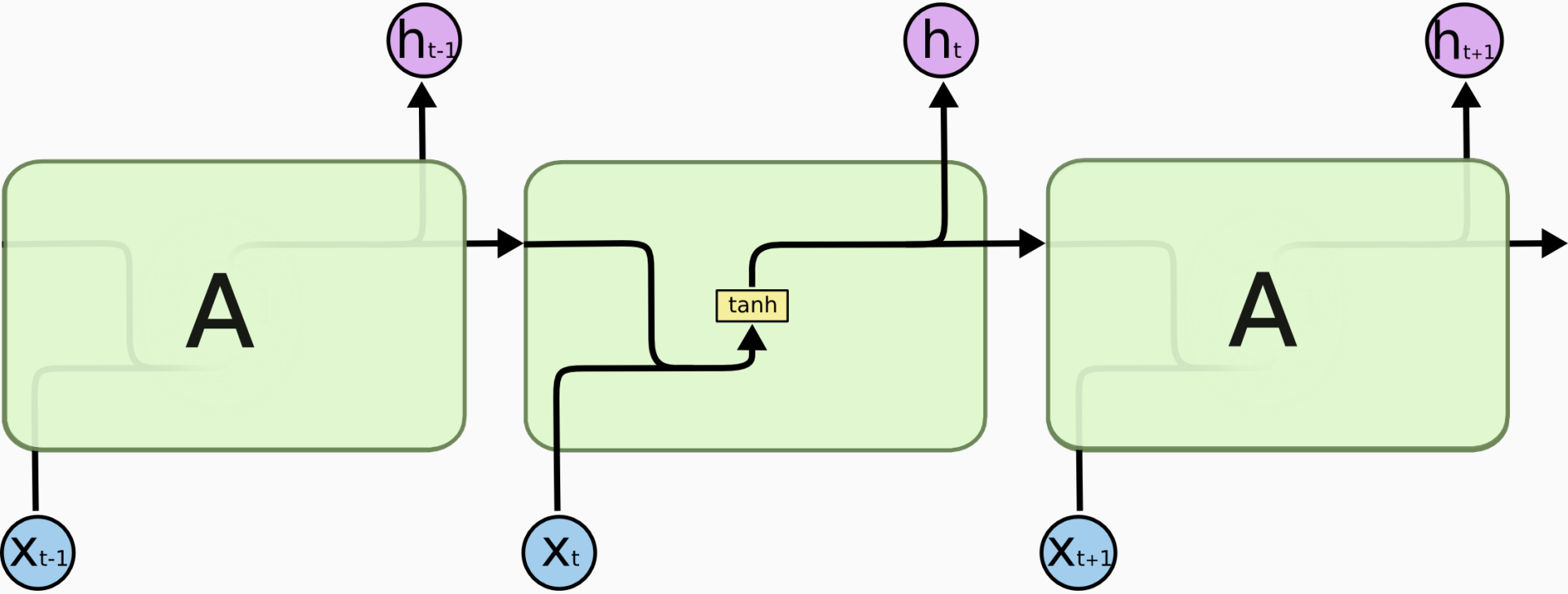
I grew up in France…. I speak ____



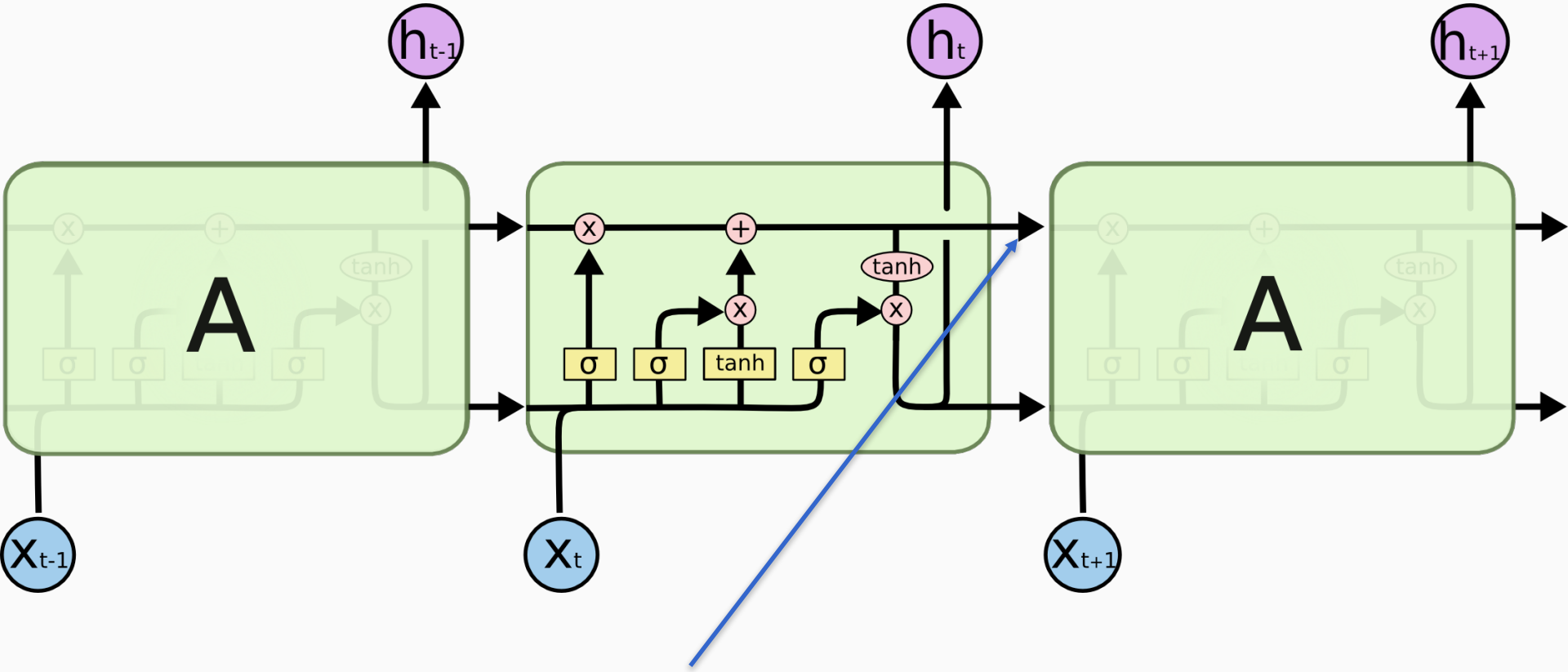
RNNs don't seem to be able to learn long range dependencies [Hochreiter 1991, Bengio et al 1994]

The answer: Better control over the memory via **Long Short-term Memory** (LSTM) units

Inside an Recurrent neuron

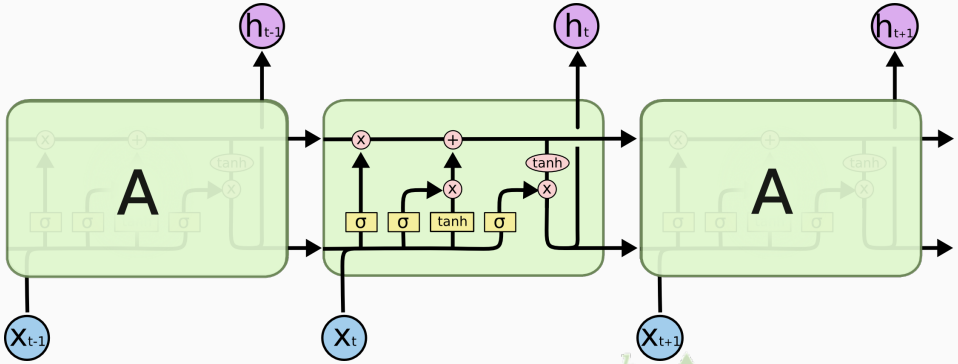


Inside a Long Short Term Memory unit

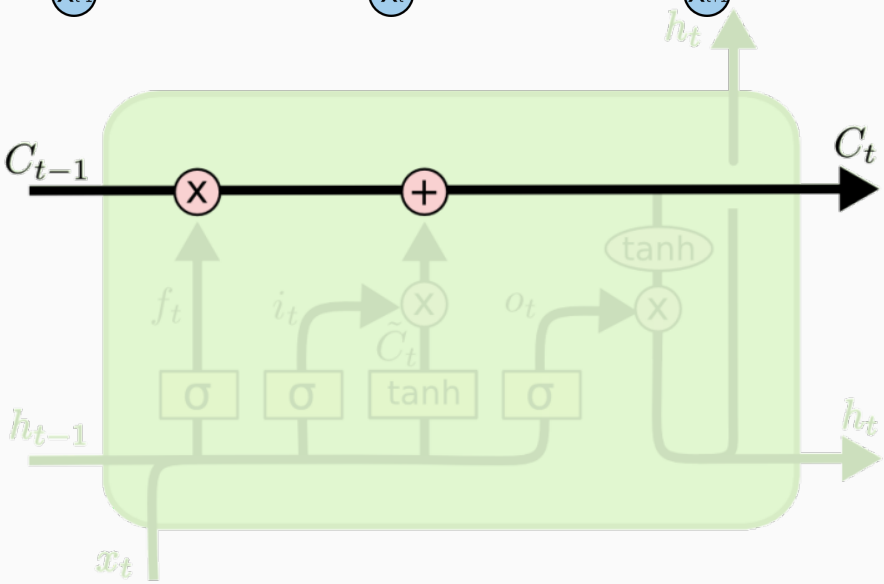


Adds an additional memory to the cell

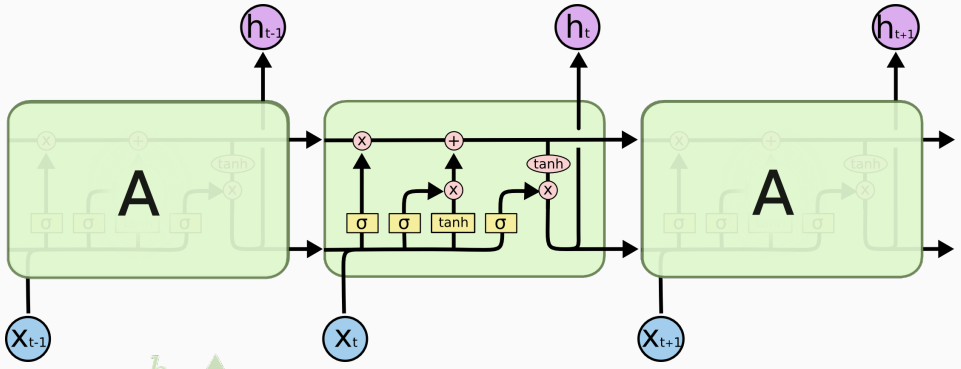
Let us zoom in

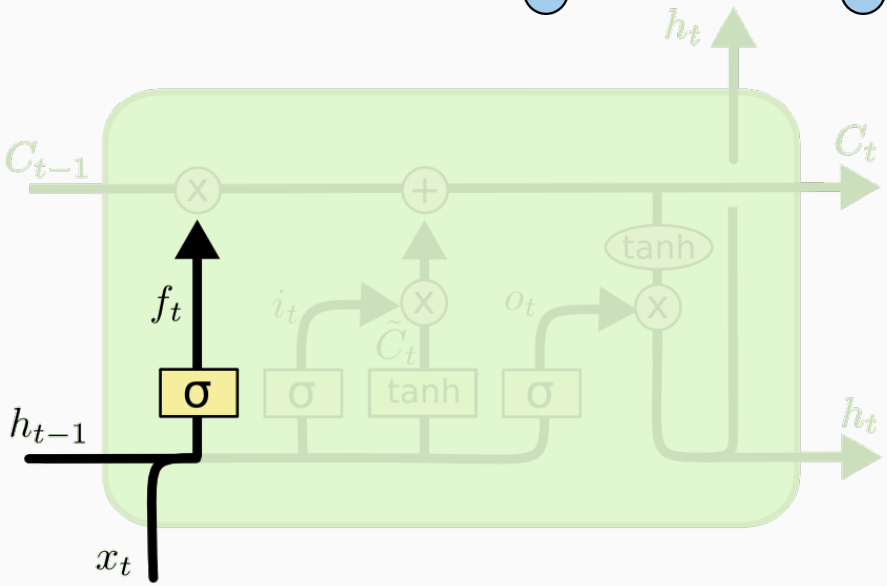


Cell state



Let us zoom in

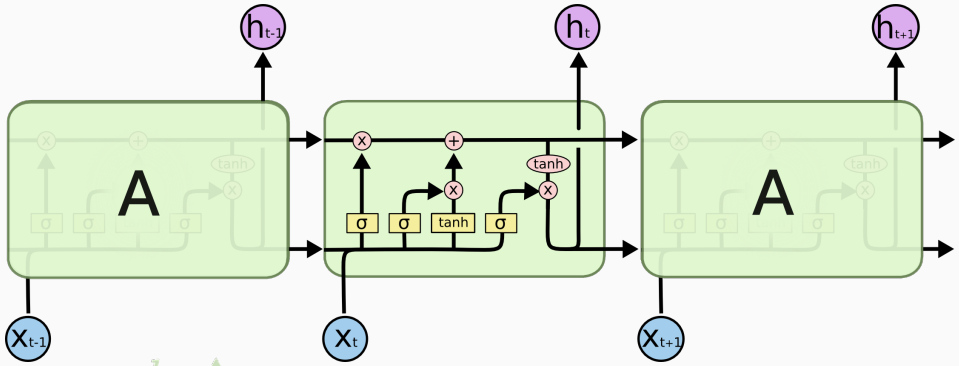


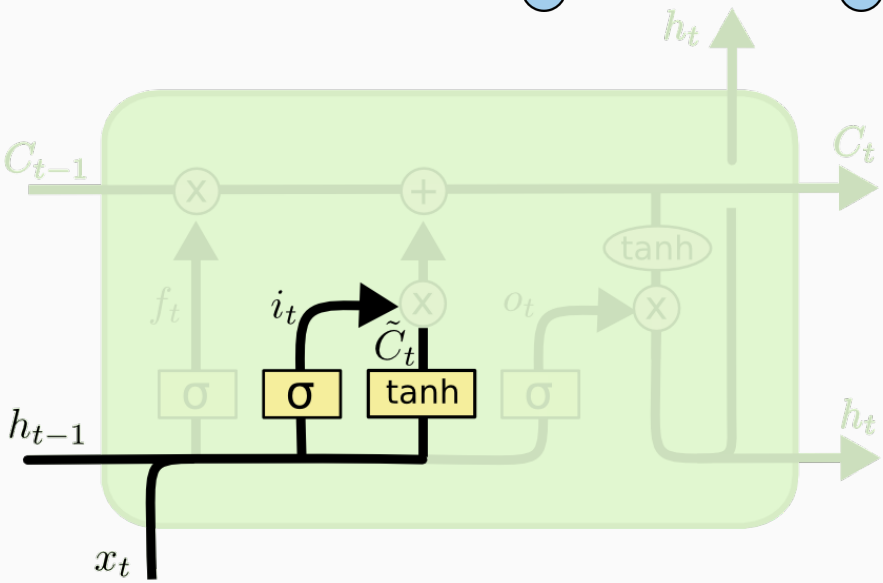


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The “forget gate”: Use the current input to decide what to erase in the cell state

Let us zoom in



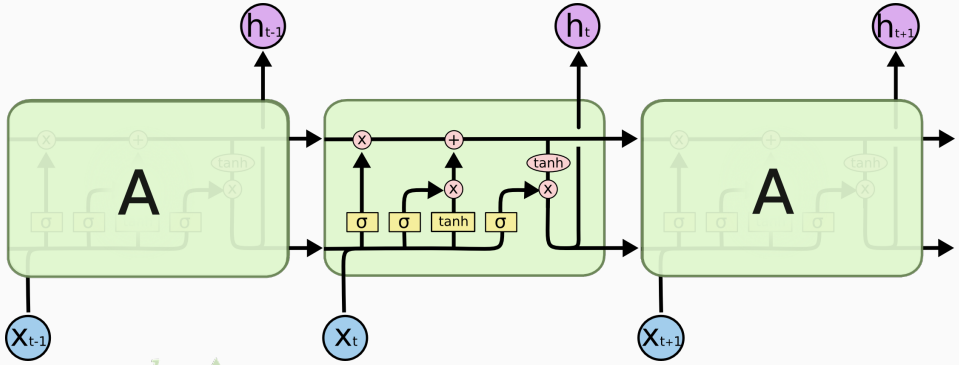


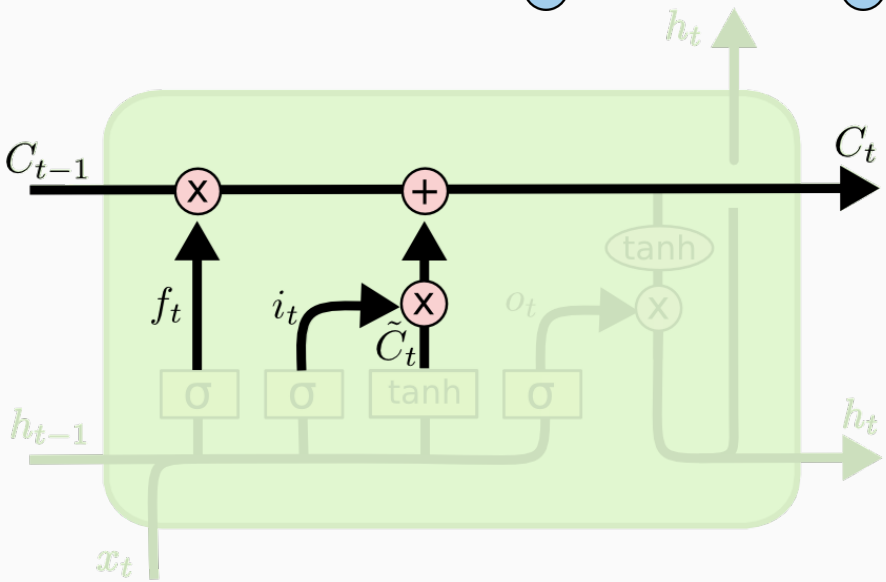
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Create a new cell state and also a filter that decides what part of the newly created cell state should be remembered

Let us zoom in

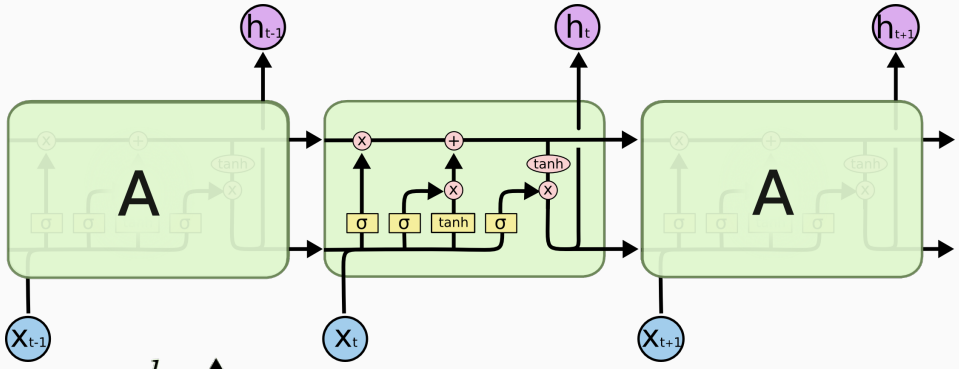


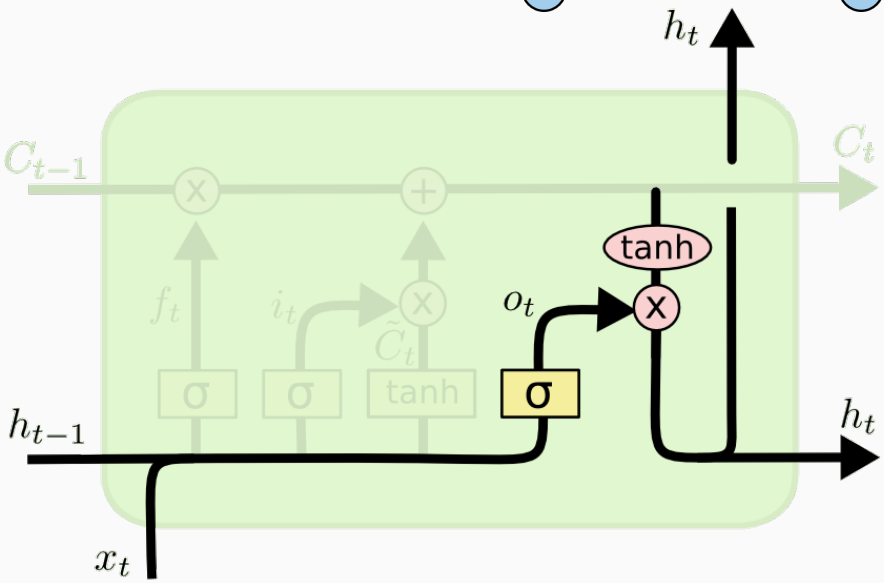


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

New cell state = remaining part of previous state + newly computed information

Let us zoom in





$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Finally, output = filtered version of the new cell state

Examples: Generating Shakespeare

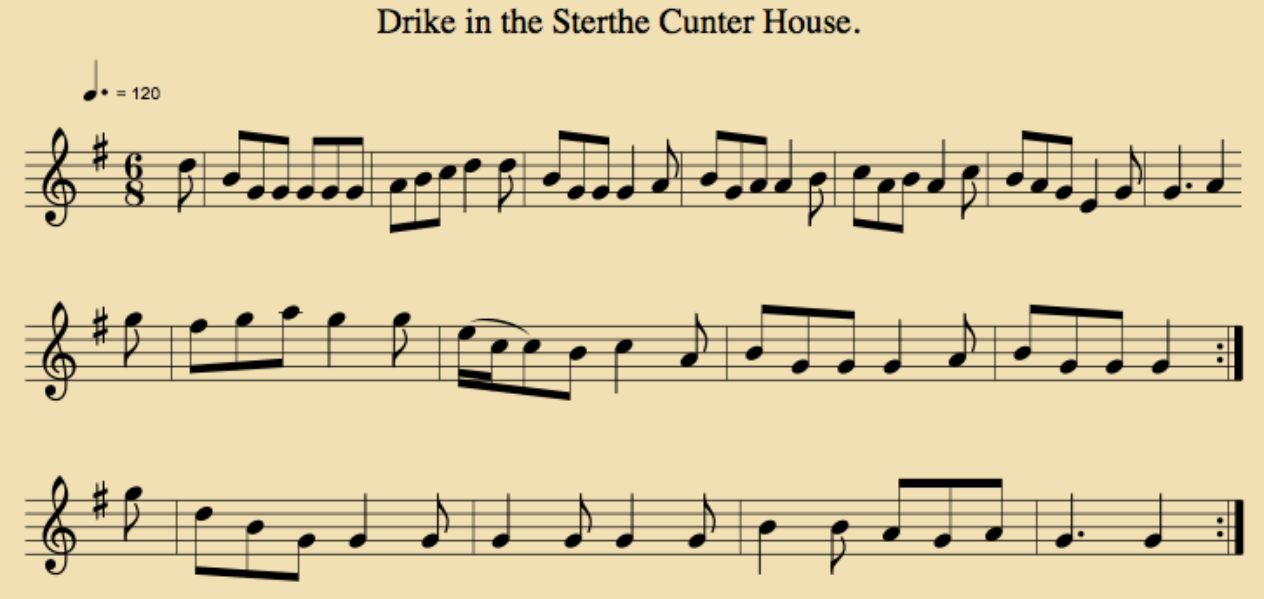
```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

A three layer RNN, 512 hidden nodes in each layer

Millions of parameters

https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Examples: Generating Audio



https://highnoongmt.wordpress.com/2015/05/22/lisls-stis-recurrent-neural-networks-for-folk-music-generation/

Predicting sequences

LSTMs are a fundamental unit of recurrent neural networks

- They are here to stay
- Essential component of sequence-to-sequence models
- Massive in terms of the number of parameters
  - The Google neural language model has billions of parameters
- Several variants exist, but all have a similar flavor
  - Eg: The gated recurrent unit is a simpler variant

Summary

- Neural networks combine expressive scoring functions with distributed input representations

- Several open questions still remain. Some examples:
  - How do we incorporate output dependencies between vector valued representations?
  - Structures offer a clean approach for modeling compositionality. How do we compose distributed representations that are scored with neural networks?
  - Incorporating inference and domain knowledge within neural networks. Perhaps to guide training or for improved predictions?