Lecture 19: Neural Networks Backward Pass

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: These slides are based primarily on content from the book “Machine Learning” by Tom Mitchell and slides created by Vivek Srikumar (Utah) and Dan Roth (Penn) and Dan Klein and Pieter Abbeel (UC Berkeley)

Today's Topics

Homework 7

- out on Wednesday

Neural networks

- Prediction using a neural network
- Training neural networks
- Practical concerns

vumanfredi@wesleyan.edu

Neural Networks

HOW DO WE COMPUTE OUTPUT OF NEURAL NETWORK?

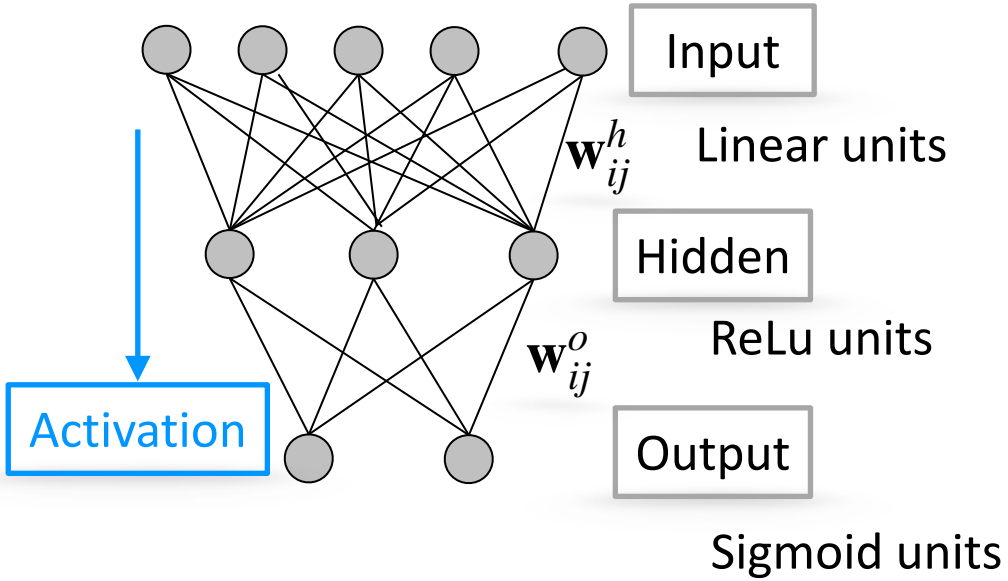
How do we define a neural network?

To define a neural network, we need to specify

- The structure of the graph: how many nodes (what is the type of each node) and how are they organized (what is the connectivity among nodes)
- The activation function on each node
- The edge weights → Learned from data, assuming structure of graph is fixed

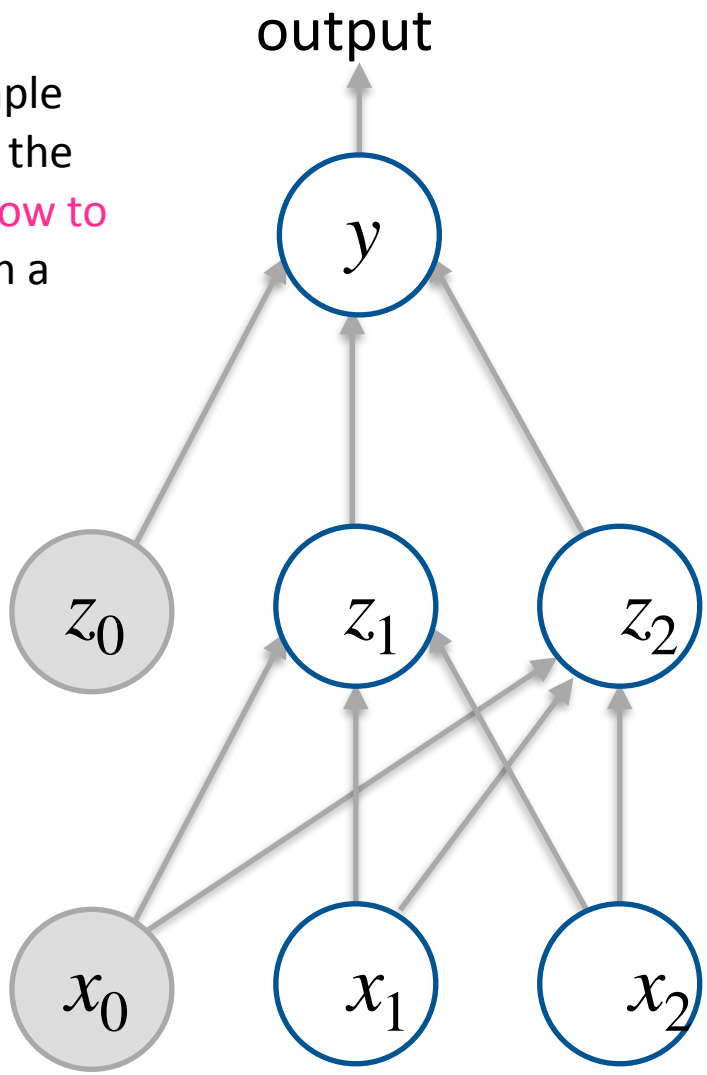
Called the architecture of network. Typically predefined, part of the design of the classifier.

Specialized architectures for different problems like vision or text



Consider an example network

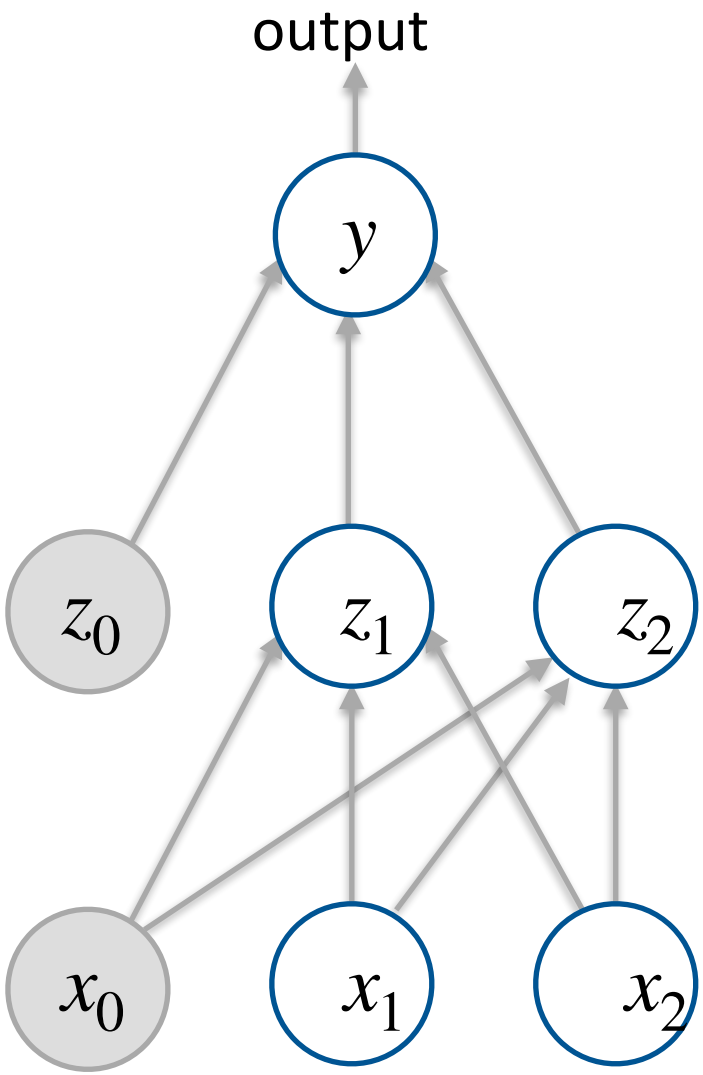
We will use this example network to introduce the general principle of how to make predictions with a neural network



How many layers does this neural network have?

2 layers

What is all this notation?



Naming conventions for this example

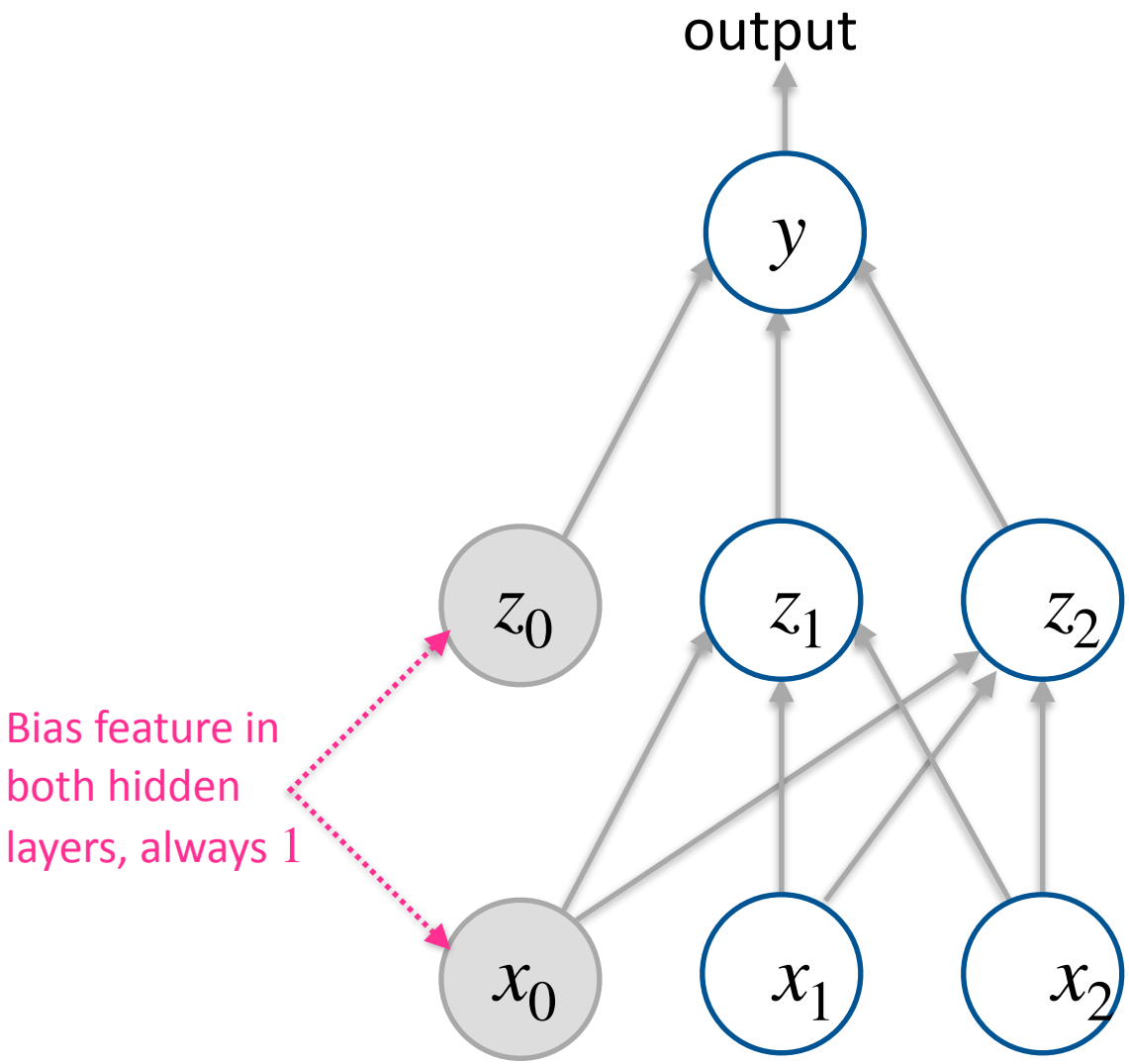
- inputs: x
- hidden: z
- output: y

What are some examples of x and y?

xs are features, could be any features we've used on homework

ys are what we are trying to predict, e.g., house price, spam or not spam, etc.

Bias features are part of network

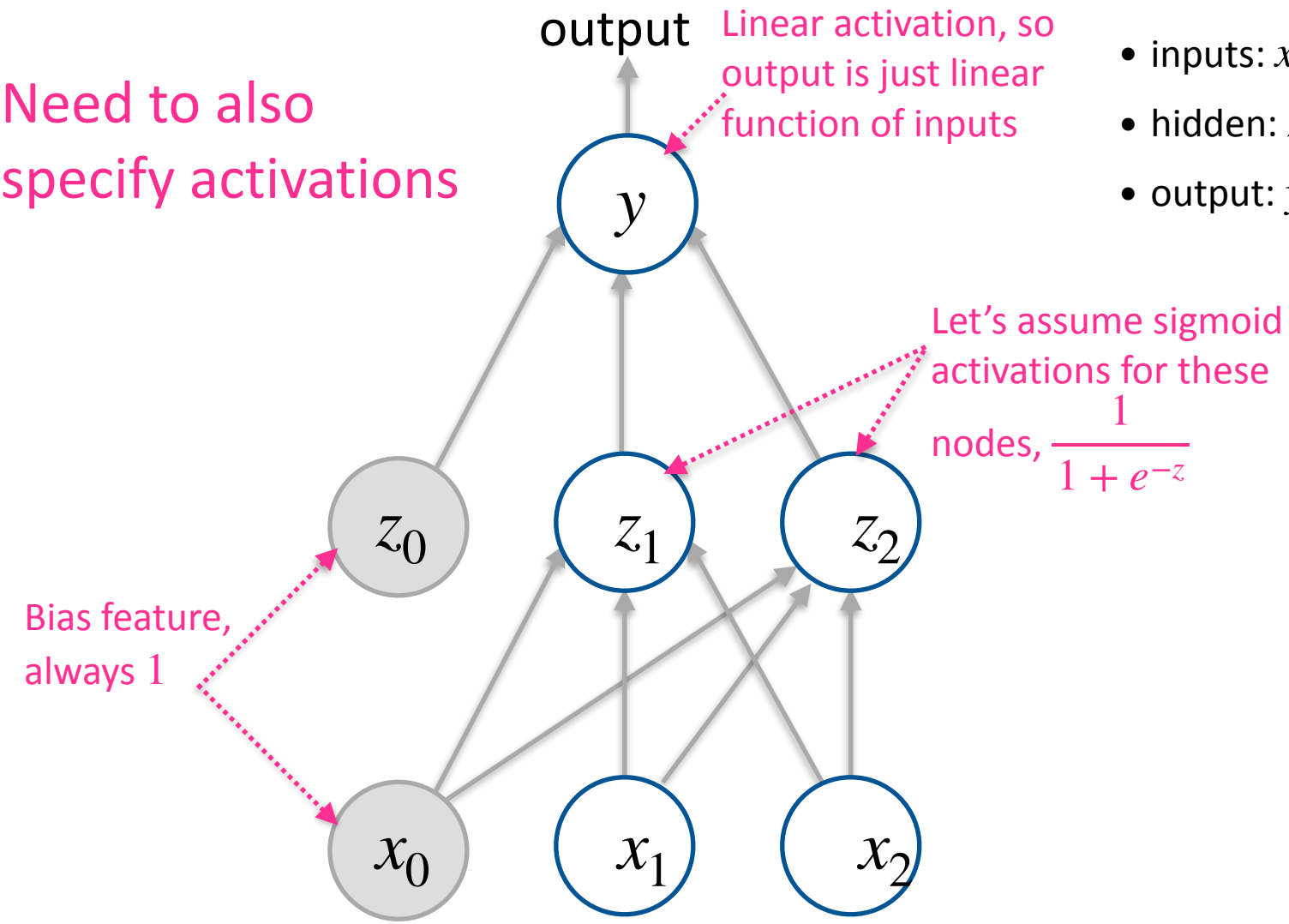


Naming conventions for this example

- inputs: x
- hidden: z
- output: y

What activation functions to use?

Need to also specify activations



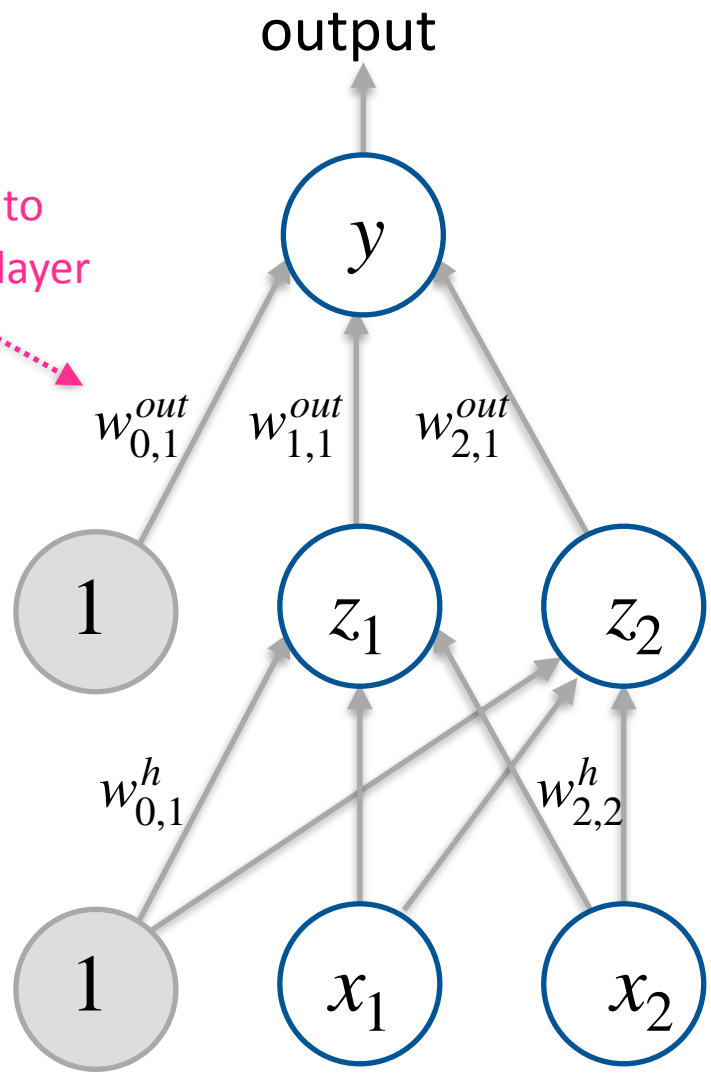
Naming conventions for this example

- inputs: x
- hidden: z
- output: y

Every edge has a weight

Naming conventions for weights: $w^{target-layer}_{from,to}$

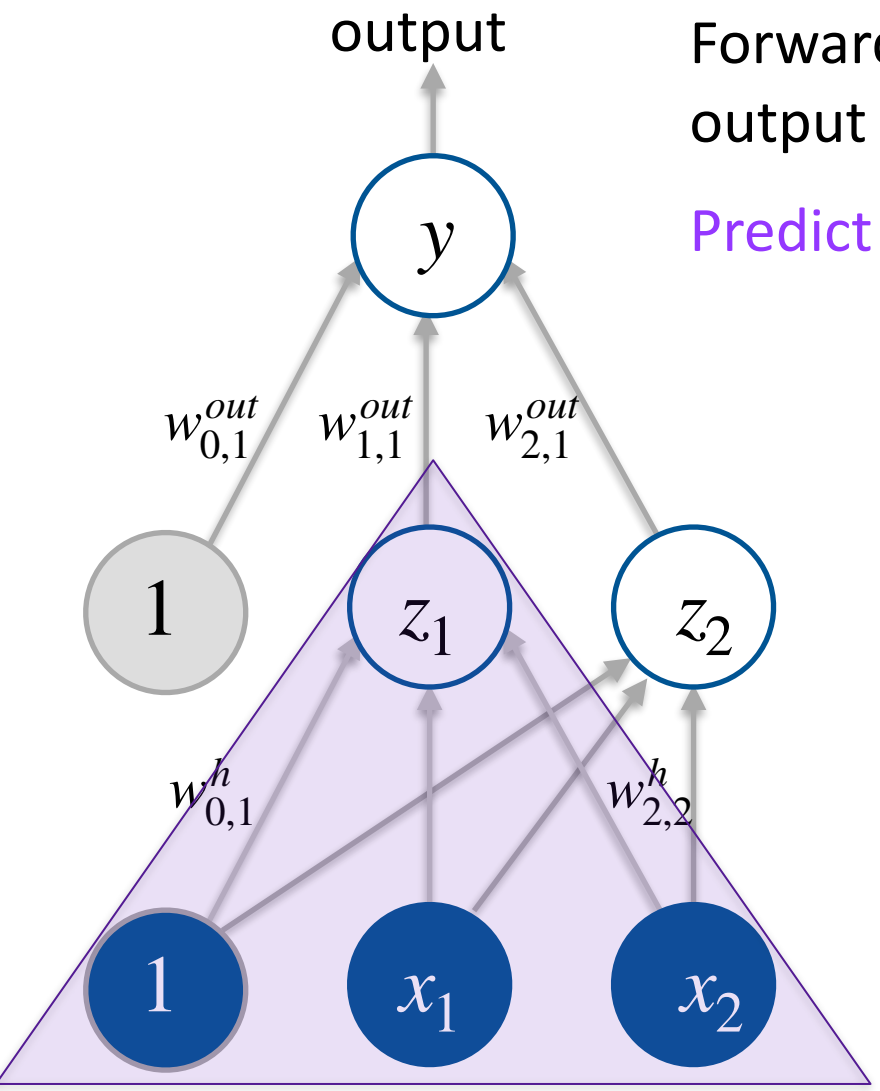
E.g., from neuron 0 to neuron 1 in output layer



Need to specify weights in order to make predictions

One weight for every edge in graph

How to do prediction?



Forward pass: given an input $\mathbf{x}$, how is the output predicted using a neural network?

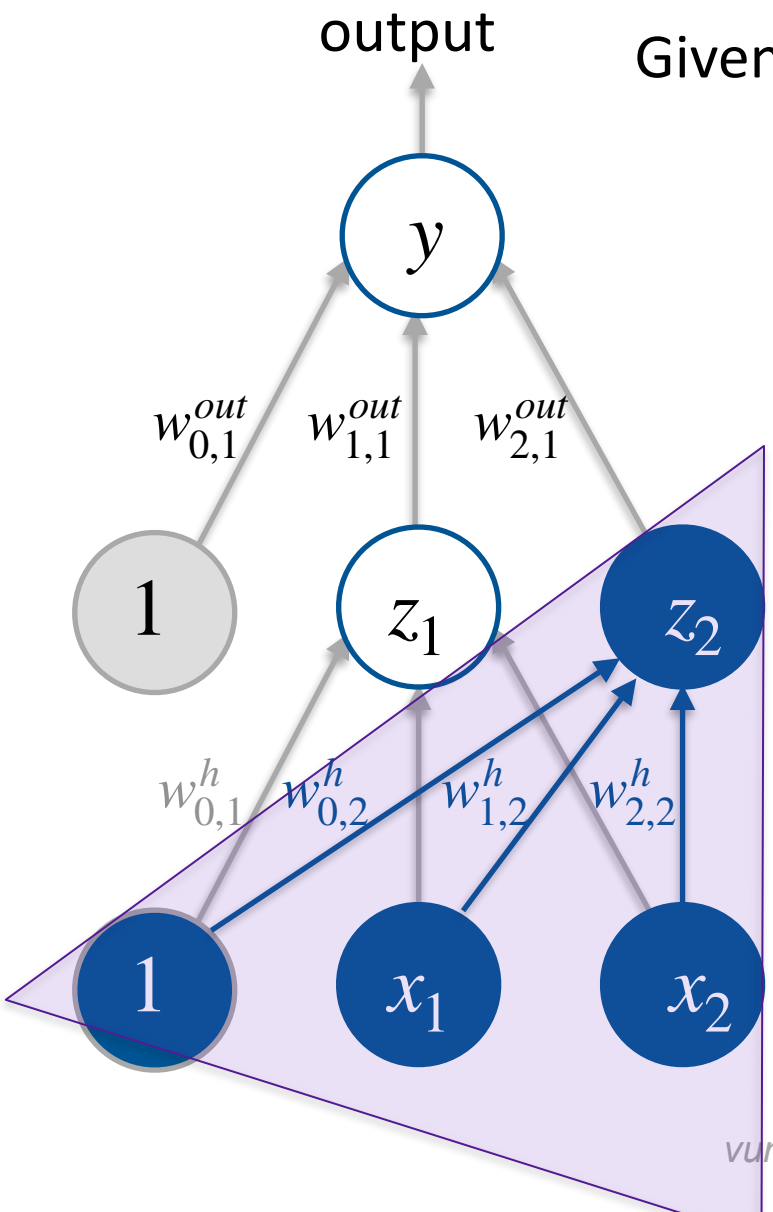
Predict from the bottom up …

$$z_1 = \sigma(w_{0,1}^h + w_{1,1}^h x_1 + w_{2,1}^h x_2)$$

To compute output need to know each input and associated weight

Question: what is activation at hidden nodes, z? Sigmoid

The forward pass for prediction

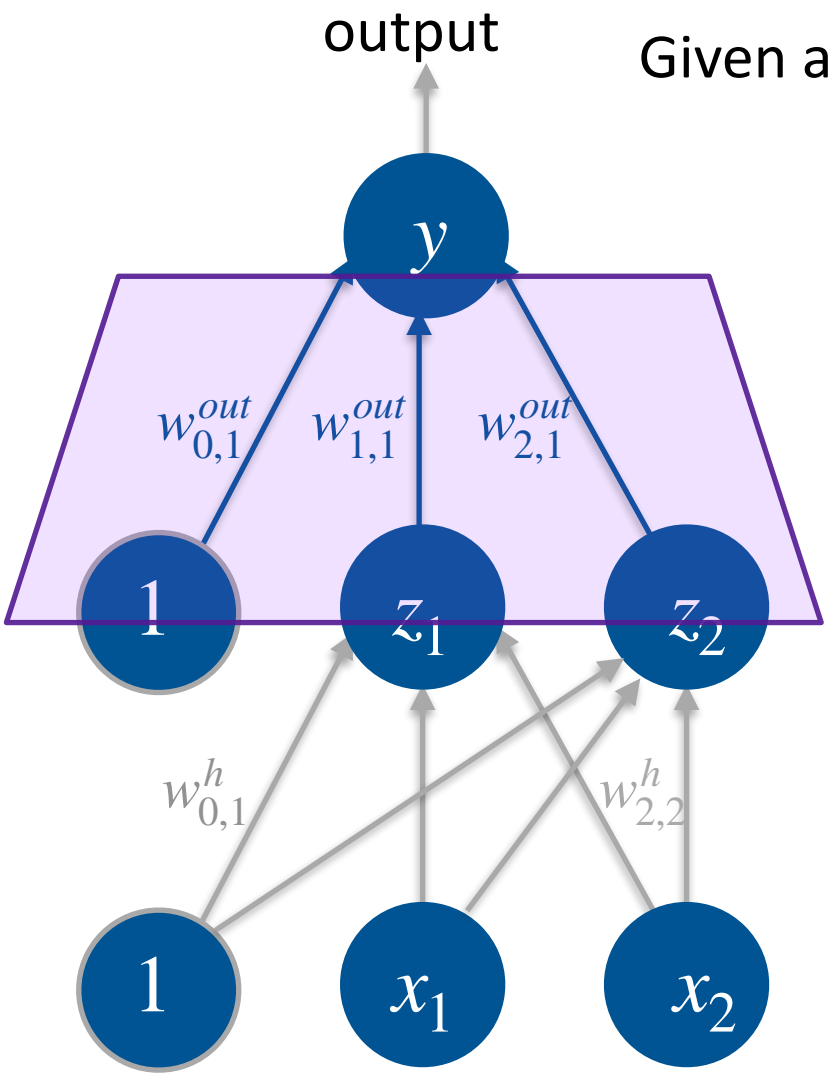


Given an input $\mathbf{x}$, how is the output predicted?

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2}x_1 + w^h_{2,2}x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1}x_1 + w^h_{2,1}x_2)$$

The forward pass for prediction



Given an input $\mathbf{x}$, how is the output predicted?

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

In general, to predict output, just compute value. But before visiting (i.e., computing) the value of a node, need to visit all nodes that serve as inputs to it

Neural Networks

HOW DO WE TRAIN NEURAL NETWORKS?

Training a neural network

Given

A network architecture

- layout of neurons, neuron connectivity, and neuron activations

A dataset of labeled examples

- $S = \{(\mathbf{x_i}, y_i)\}$

Goal

- Learn the weights of the neural network

Remember

- For a fixed architecture, a neural network is a function parameterized by its weights
- Prediction: $y = NN(\mathbf{x}, \mathbf{w})$

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

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

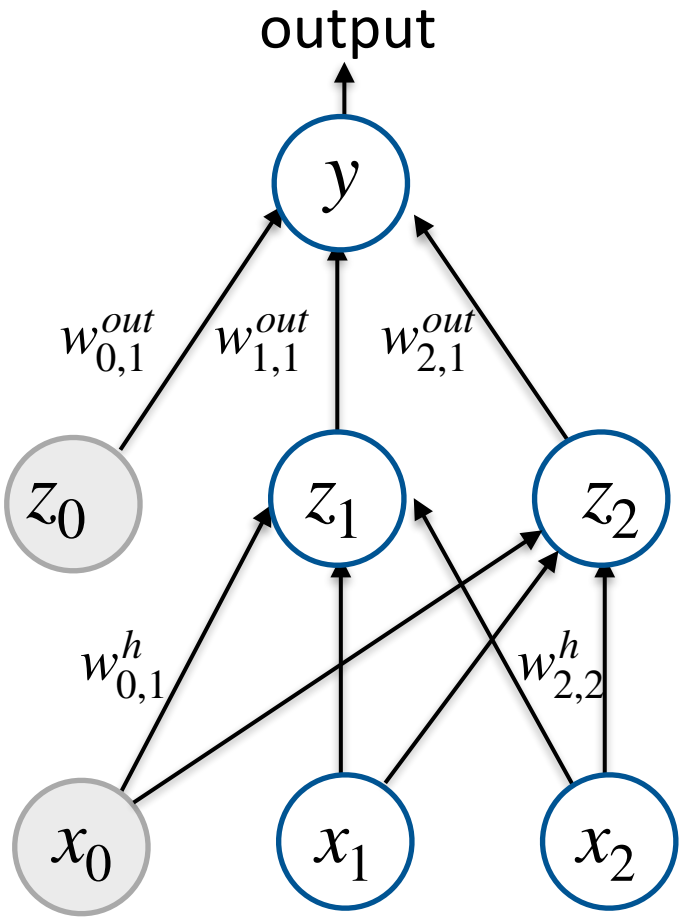
perhaps with a regularizer

How do we solve the optimization problem?

Saw strategy that worked for perceptron and LMS regression: each minimizes a different loss function using (stochastic) gradient descent algorithm.

Same idea for non-linear models too!

Back to our running example



Given an input $\mathbf{x}$, how is the output predicted?

$$y = w_{0,1}^{o} + w_{1,1}^{o} z_1 + w_{2,1}^{o} z_2$$

$$z_2 = \sigma(w_{0,2}^{h} + w_{1,2}^{h} x_1 + w_{2,2}^{h} x_2)$$

$$z_1 = \sigma(w_{0,1}^{h} + w_{1,1}^{h} x_1 + w_{2,1}^{h} x_2)$$

Suppose we have set of weights and the true label for example is real number y_i.

Question: What kind of learning problem do we have and what is reasonable loss function?

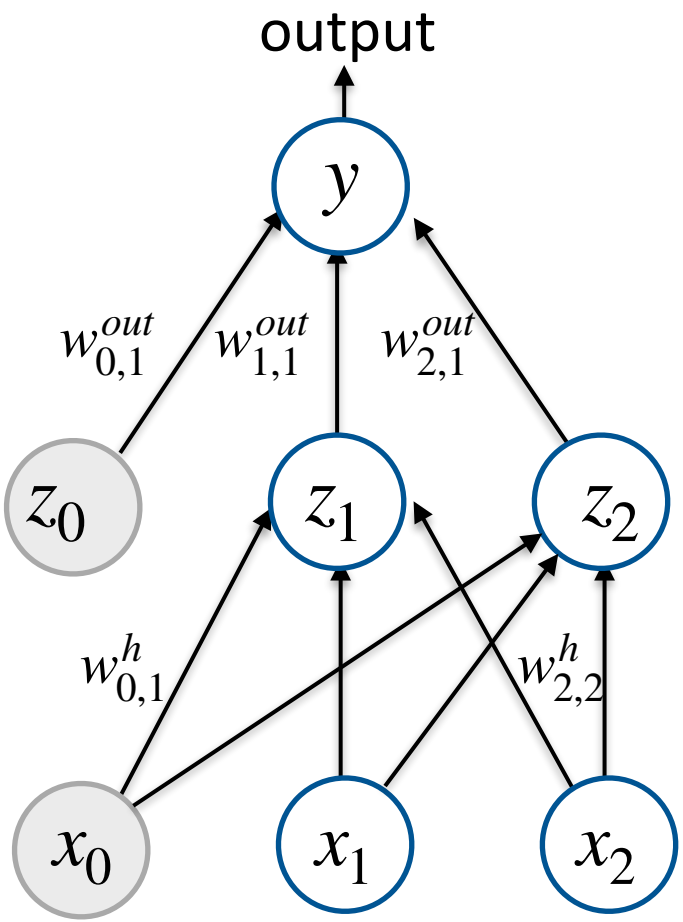
Regression problem, and square loss is reasonable loss function to use. We can write square loss for example as:

$$L = \frac{1}{2}(y_{pred} - y_i)^2$$

Error is a function of all weights!

A notational convenience

Commonly nodes in the network represent not only single numbers (e.g., features, outputs) but also ***vectors*** (an array of numbers), ***matrices*** (a 2d array of numbers) or ***tensors*** (an n-dimensional array of numbers)



No activation because the output is defined to be linear

$y = \mathbf{w}^o\mathbf{z}$ z here doesn't include bias z_0

$\mathbf{W}^o = \begin{bmatrix} w^o_{0,1} & w^o_{1,1} & w^o_{2,1} \end{bmatrix}$

$\mathbf{z} = \sigma(\mathbf{W}^h\mathbf{x})$

$\mathbf{W}^h = \begin{bmatrix} w^h_{0,1} & w^h_{1,1} & w^h_{2,1} \\ w^h_{0,2} & w^h_{1,2} & w^h_{2,2} \end{bmatrix}$

$\mathbf{x}$ Represents $[x_0, x_1, x_2]$

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

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

How do we solve the optimization problem?

(Stochastic) gradient descent

Stochastic gradient descent

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

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \mathfrak{R}^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$ ← Parameters $\mathbf{w}$ are the only unknown
2. For epoch $= 1 \ldots T$:

Return $\mathbf{w}$ ← Return final weights once converged

Stochastic gradient descent

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

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \mathfrak{R}^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$
2. For epoch $= 1 \ldots T$:
   - Shuffle the training set

Return $\mathbf{w}$

Stochastic gradient descent

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

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \Re^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$
2. For epoch $= 1 \ldots T$:
   - Shuffle the training set
   - For each training example $(\mathbf{x_i}, y_i) \in S$:

Recall shuffling with perceptron (which uses stochastic gradient descent though we didn't call it that)

Return $\mathbf{w}$

Stochastic gradient descent

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

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \mathfrak{R}^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$
2. For epoch $= 1 \ldots T$:
   - Shuffle the training set
   - For each training example $(\mathbf{x_i}, y_i) \in S$:

     ➡ Treat this example as the entire dataset

     Compute the gradient of the loss $\nabla L(NN(\mathbf{x_i}, \mathbf{w}), y_i)$

Return $\mathbf{w}$

Stochastic gradient descent

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

The objective is not convex (unlike with linear classifiers/ regressors). Initialization is now extremely important!

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \Re^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$
2. For epoch $= 1 \ldots T$:
   - Shuffle the training set
   - For each training example $(\mathbf{x_i}, y_i) \in S$:

     ➡ Treat this example as the entire dataset

     Compute the gradient of the loss $\nabla L(NN(\mathbf{x_i}, \mathbf{w}), y_i)$

     ➡ Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$

γ_t: learning rate, many tweaks possible

Return $\mathbf{w}$

Have we solved everything?

The derivative of the loss function?

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

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

- Or maybe its sub-gradient (to minimize non-differentiable function)
- This is decided by the activation functions and the loss function

Easy if one layer. But how to find sub-gradient of more complex function?

- E.g., 150 layer neural network for image classification!

Even worse, every time we change neural network graph, we have a different gradient to compute, since graph represents function

We need an efficient algorithm

Backpropagation: computes gradients of functions

Checkpoint

If we have 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

So how do we compute the derivative?

Some simple expressions

$f(x, y) = x + y$

$$\frac{\partial f}{\partial x} = 1$$

$$\frac{\partial f}{\partial y} = 1$$

Some simple expressions

$f(x, y) = x + y$

$$\frac{\partial f}{\partial x} = 1$$

$$\frac{\partial f}{\partial y} = 1$$

$f(x, y) = xy$

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

$$\frac{\partial f}{\partial x} = x$$

Some simple expressions

$f(x, y) = x + y$

$$\frac{\partial f}{\partial x} = 1$$

$$\frac{\partial f}{\partial y} = 1$$

$f(x, y) = xy$

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

$$\frac{\partial f}{\partial x} = x$$

$f(x, y) = \max(x, y)$

$$\frac{\partial f}{\partial x} = 1, \text{ if } x \geq y, 0 \text{ otherwise}$$

$$\frac{\partial f}{\partial y} = 1, \text{ if } y \geq x, 0 \text{ otherwise}$$

Useful to keep in mind what these derivatives represent in these (and all other) cases:

$$\frac{\partial f}{\partial x}$$

Represents the rate of change of the function f with respect to a small change in x

More complicated cases?

$$f(x, y, z) = x(y^2 + z)$$

This is still simple enough to manually take derivatives, but let us work through this in a slightly different way

Break down the function in terms of simple forms

$$g = y^2 + z$$
$$f = xg$$

Each of these is a simple form. We know how to compute $\frac{\partial g}{\partial y}, \frac{\partial g}{\partial z}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial g}$

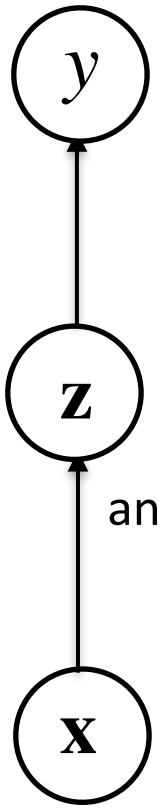
Key idea: build up derivatives of compound expressions by breaking it down into simpler pieces, and applying the chain rule

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial y} = x \cdot 2y = 2xy$$

Reminder: chain rule of derivatives

If y is a function of $\mathbf{z}$ and $\mathbf{z}$ is a function of $\mathbf{x}$ then y is a function of $\mathbf{x}$ as well

How to find gradient of y with respect to $\mathbf{x}$, $\frac{\partial y}{\partial \mathbf{x}}$?



If y is a function of $\mathbf{z}$, $y = f(\mathbf{z})$

$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ by chain rule (or $\frac{\partial y}{\partial \mathbf{x}} = f'(g(\mathbf{x}))g'(\mathbf{x})$)

and $\mathbf{z}$ is a function of $\mathbf{x}$, $\mathbf{z} = g(\mathbf{x})$

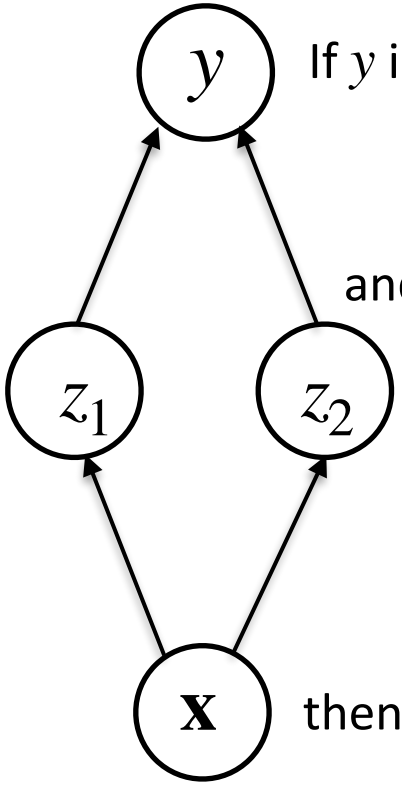
then y is a function of $\mathbf{x}$ as well, $y = f(g(\mathbf{x}))$

Reminder: chain rule of derivatives

If y is a function of z_1 + a function of z_2, and the z_i's are functions of $\mathbf{x}$

- then y is a function of $\mathbf{x}$ as well

How to find $\frac{\partial y}{\partial \mathbf{x}}$?



If y is a sum of functions of z_i's, $y = f_1(z_1) + f_2(z_2)$

and z_i's are functions of $\mathbf{x}$, $z_i = g_i(\mathbf{x})$

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial z_1} \cdot \frac{\partial z_1}{\partial \mathbf{x}} + \frac{\partial y}{\partial z_2} \cdot \frac{\partial z_2}{\partial \mathbf{x}}$$

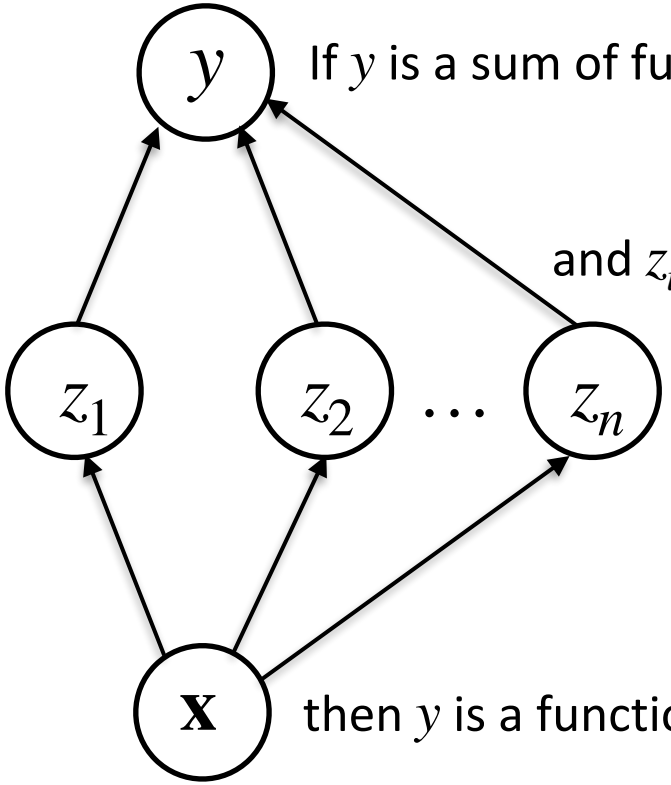
then y is a function of $\mathbf{x}$ as well, $y = f_1(g_1(\mathbf{x})) + f_2(g_2(\mathbf{x}))$

Reminder: chain rule of derivatives

If y is a sum of functions of z_i, and z_i is a function of $\mathbf{x}$

- then y is a function of $\mathbf{x}$ as well

How to find $\frac{\partial y}{\partial \mathbf{x}}$?



If y is a sum of functions of z_i's, $z = \sum f_i(y_i)$

and z_i's are functions of $\mathbf{x}$, $z_i = g_i(\mathbf{x})$

$$\frac{\partial y}{\partial \mathbf{x}} = \sum_{i=1}^{n} \frac{\partial y}{\partial z_i} \cdot \frac{\partial z_i}{\partial \mathbf{x}}$$

then y is a function of $\mathbf{x}$ as well, $y = \sum f_i(g_i(\mathbf{x}))$

Neural Networks

BACKPROPAGATION ALGORITHM

The abstraction

Each node in the graph knows 2 things:

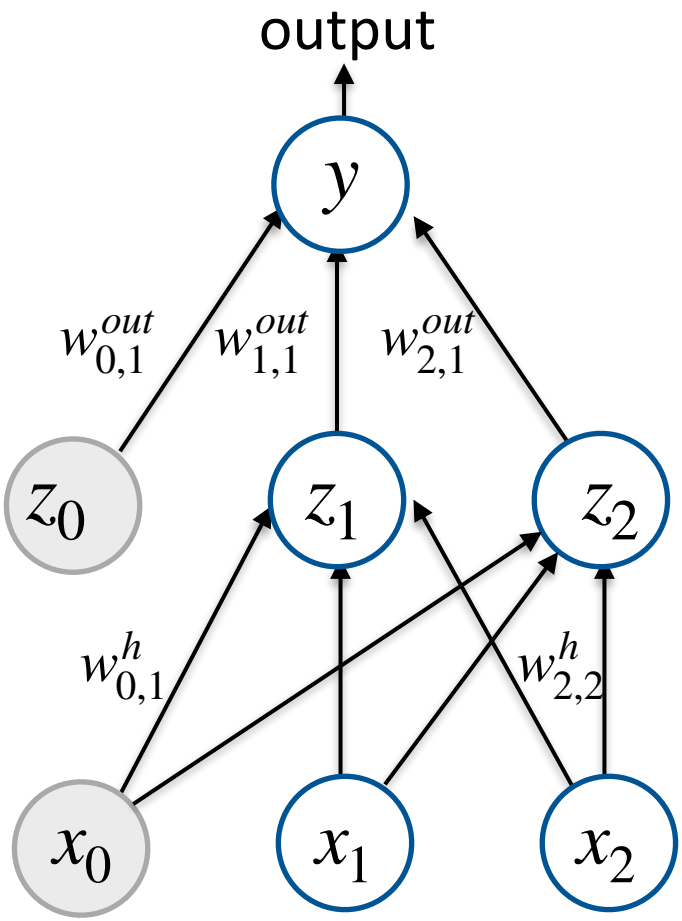
1. Forward pass: how to compute value of a function with respect to its inputs
2. Backward pass: how to compute partial derivative of the output with respects to its inputs

These can be defined independently of what happens in the rest of the graph

We can build up complicated functions using simple nodes and compute values and partial derivatives of these as well

Backpropagation (of errors)

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



$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

We want to compute $\frac{\partial L}{\partial w^o_{ij}}$ and $\frac{\partial L}{\partial w^h_{ij}}$

Important: L is a differentiable function of all of the weights

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

Reminder

Why do we compute gradient of loss function with respect to weights?

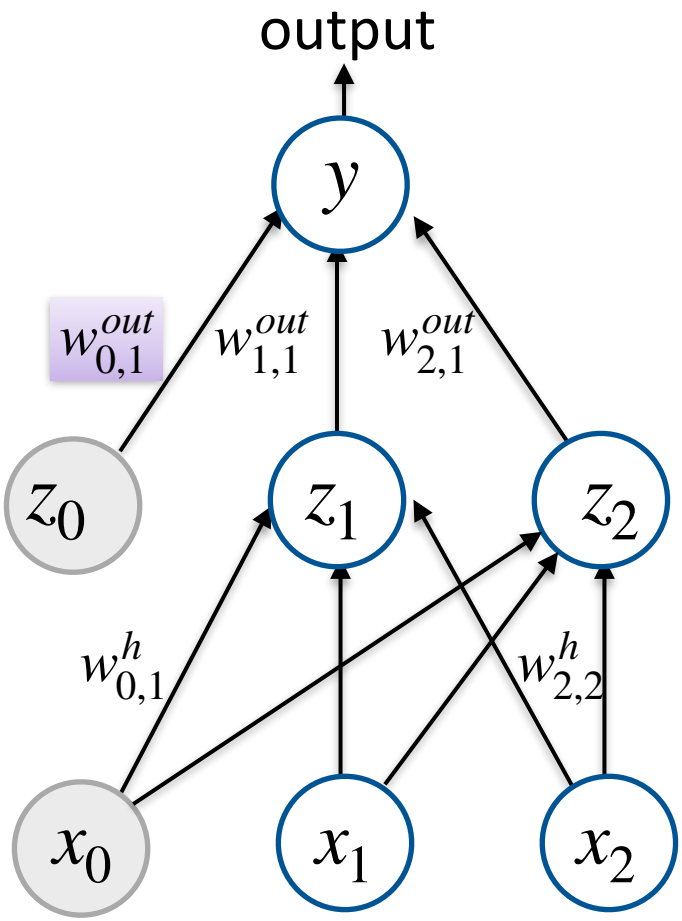
Want to find values of weights that minimize loss function.

By updating weights in opposite direction of this gradient, takes a step closer to minimum

Backpropagation: output layer

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

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$



$$\frac{\partial L}{\partial w^o_{0,1}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w^o_{0,1}}$$

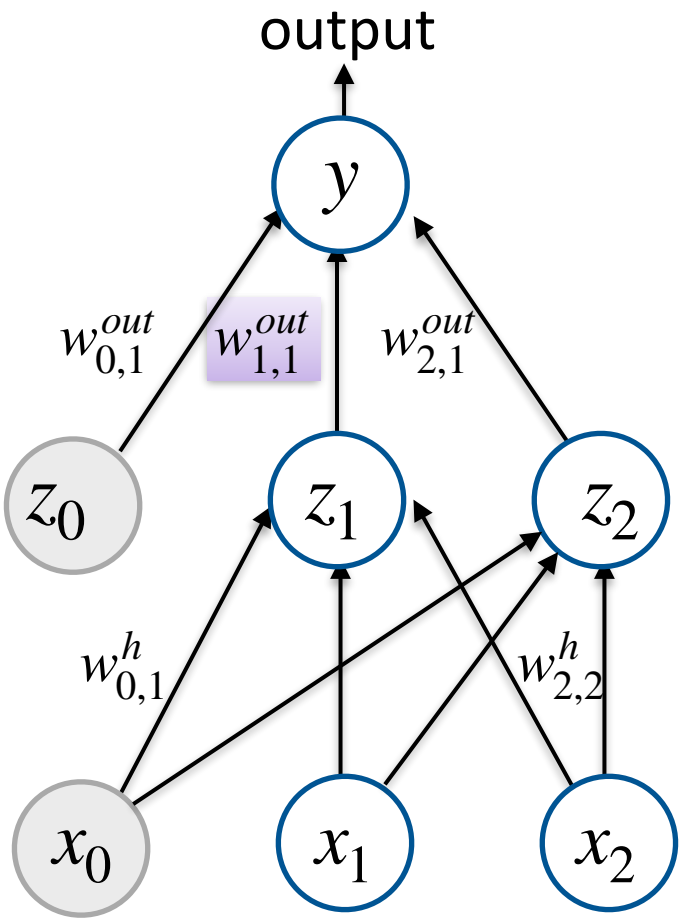
$$\frac{\partial L}{\partial y} = y - y^*$$

$$\frac{\partial y}{\partial w^o_{0,1}} = 1$$

Backpropagation: output layer

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

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$



$$\frac{\partial L}{\partial w^o_{1,1}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w^o_{1,1}}$$

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

$$\frac{\partial y}{\partial w^o_{1,1}} = z_1$$

We have already computed this partial derivative for the previous case. Cache to speed up!

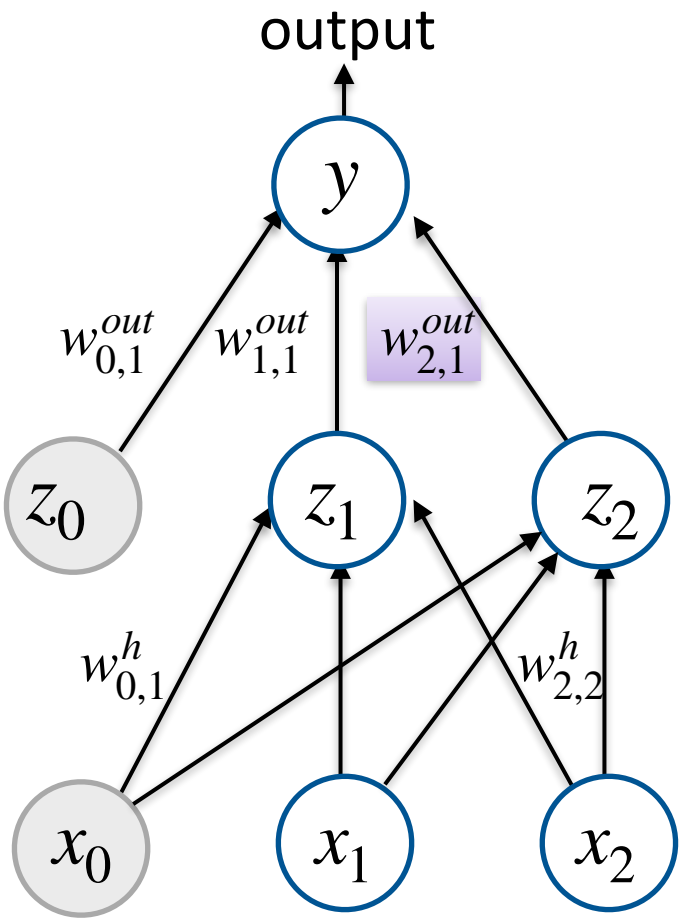
Where do we get value of z_1 from?

Compute during forward pass!
If you compute something keep it around, may be useful

3 min: what is $\partial L / \partial w^o_{2,1}$?

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

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$



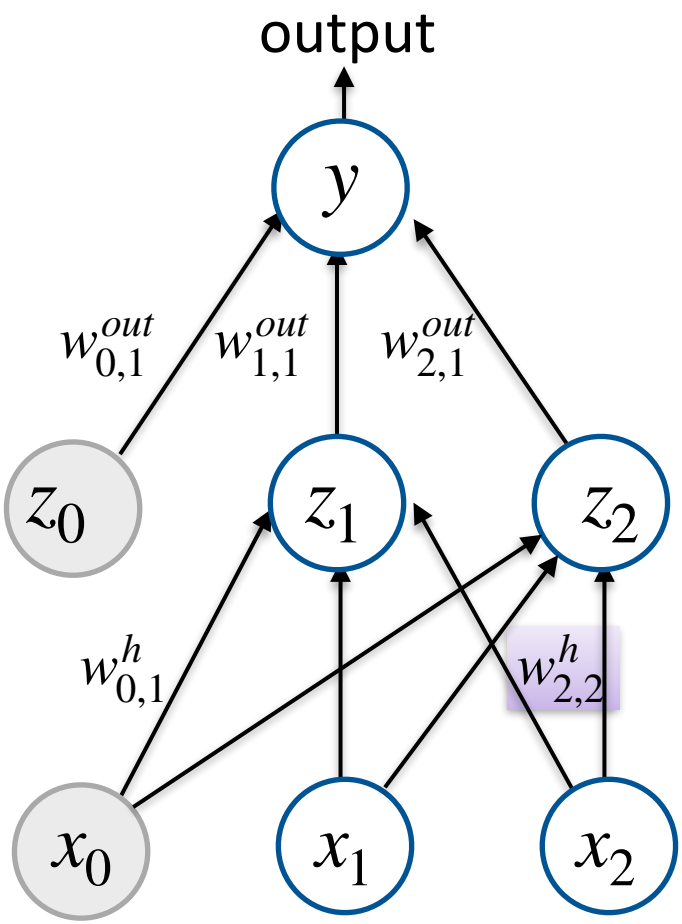
$$\frac{\partial L}{\partial w^o_{2,1}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w^o_{2,1}}$$

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

$$\frac{\partial y}{\partial w^o_{2,1}} = z_2$$

Computed during forward pass!

Backpropagation: hidden layer



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

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

We want to compute $\frac{\partial L}{\partial w^h_{22}}$

But L is not a function of w^h_{22}, is only a function of y

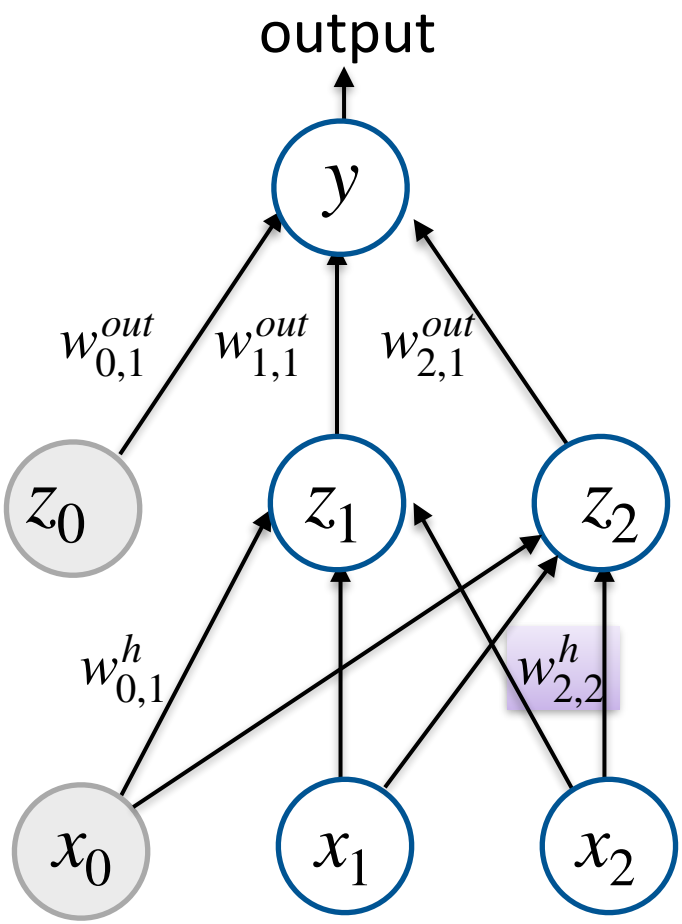
Backpropagation: hidden layer

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

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$



$$\frac{\partial L}{\partial w^h_{2,2}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w^h_{2,2}}$$

So apply chain rule

$$= \frac{\partial L}{\partial y} \cdot \frac{\partial}{\partial w^h_{2,2}} (w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2)$$

Substitute in for y

$$= \frac{\partial L}{\partial y} \cdot (w^o_{1,1} \frac{\partial}{\partial w^h_{2,2}} z_1 + w^o_{2,1} \frac{\partial}{\partial w^h_{2,2}} z_2)$$

Derivative of sum is sum of derivatives

$\frac{\partial}{\partial w^h_{2,2}} z_1 \rightarrow 0$

z_1 is not a function of $w^h_{2,2}$ so can eliminate term (like a constant)

Backpropagation: hidden layer

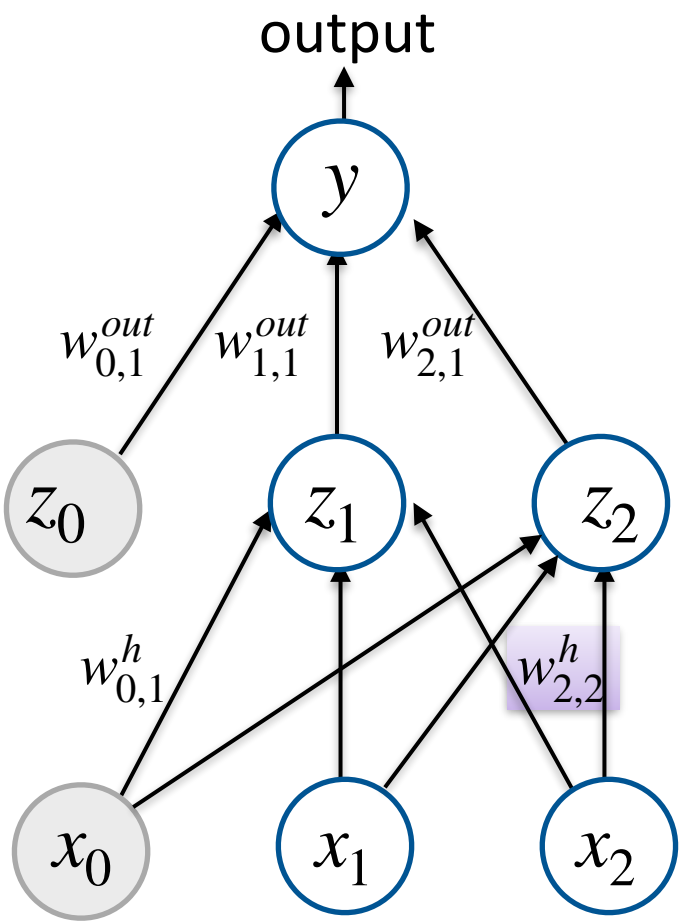
$$L = \frac{1}{2}(y - y^*)^2$$

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

$\sigma(s)$



$$\frac{\partial L}{\partial w^h_{2,2}} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w^h_{2,2}}$$

$$= \frac{\partial L}{\partial y} \cdot \frac{\partial}{\partial w^h_{2,2}}(w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2)$$

$$= \frac{\partial L}{\partial y} \cdot w^o_{2,1} \frac{\partial z_2}{\partial w^h_{2,2}}$$

z_2 is neuron with sigmoid activation applied to linear transformation

$$= \frac{\partial L}{\partial y} \cdot w^o_{2,1} \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w^h_{2,2}}$$

Compute gradient of z_2 with respect to s

Backpropagation: hidden layer

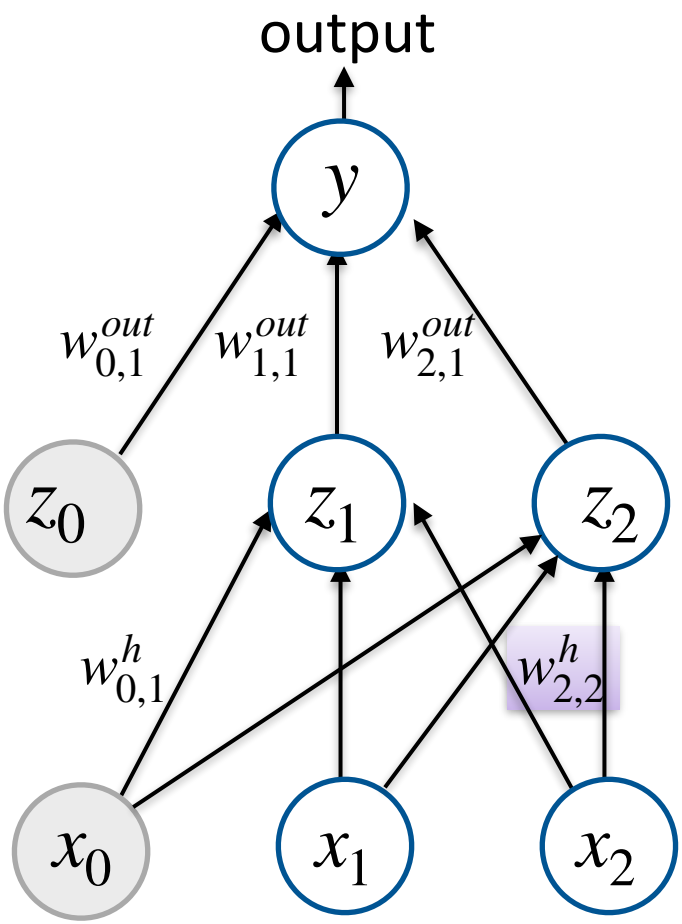
$$L = \frac{1}{2}(y - y^*)^2$$

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

$\sigma(s)$



from previous slide

$$\frac{\partial L}{\partial w^h_{2,2}} = \frac{\partial L}{\partial y} \cdot w^o_{2,1} \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w^h_{2,2}}$$

Compute gradient of z_2 with respect to s

Each of these partial derivatives is easy!

Backpropagation: hidden layer

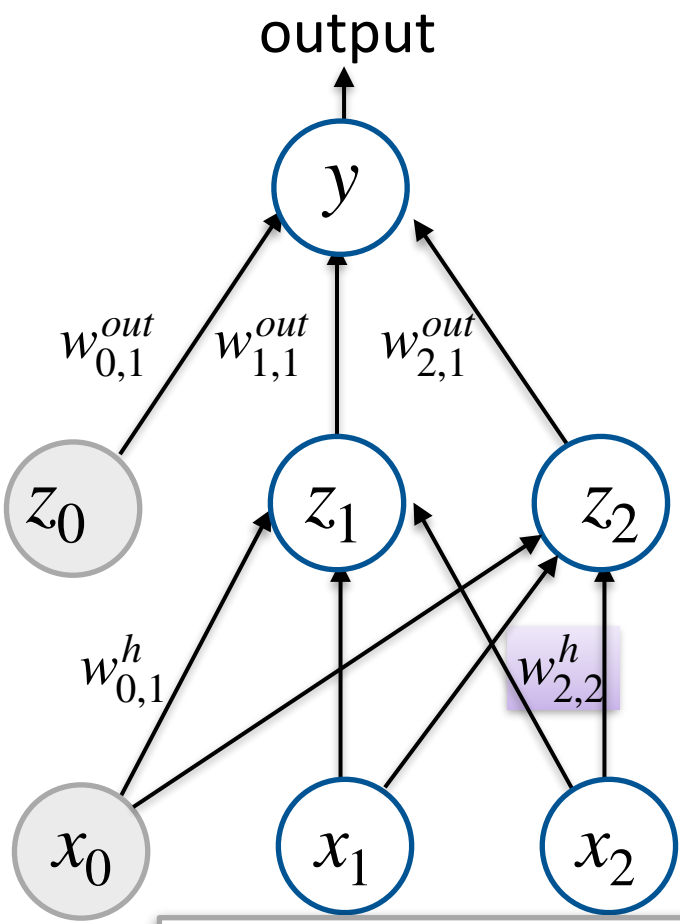
$$L = \frac{1}{2}(y - y^*)^2$$

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

$\sigma(s)$



from previous slide

$$\frac{\partial L}{\partial w^h_{2,2}} = \frac{\partial L}{\partial y} \cdot w^o_{2,1} \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w^h_{2,2}}$$

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

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

$$\frac{\partial s}{\partial w^h_{2,2}} = x_2$$

Because $z_2 = \sigma(s) = \frac{1}{1 + e^{-s}}$ is the sigmoid function which has the derivative $\sigma(s)(1 - \sigma(s))$

Importantly: we have already computed many of these partial derivatives because we are proceeding from top to bottom (i.e., backwards). And calculations can be vectorized for efficient computation on GPUs

Backpropagation: hidden layer

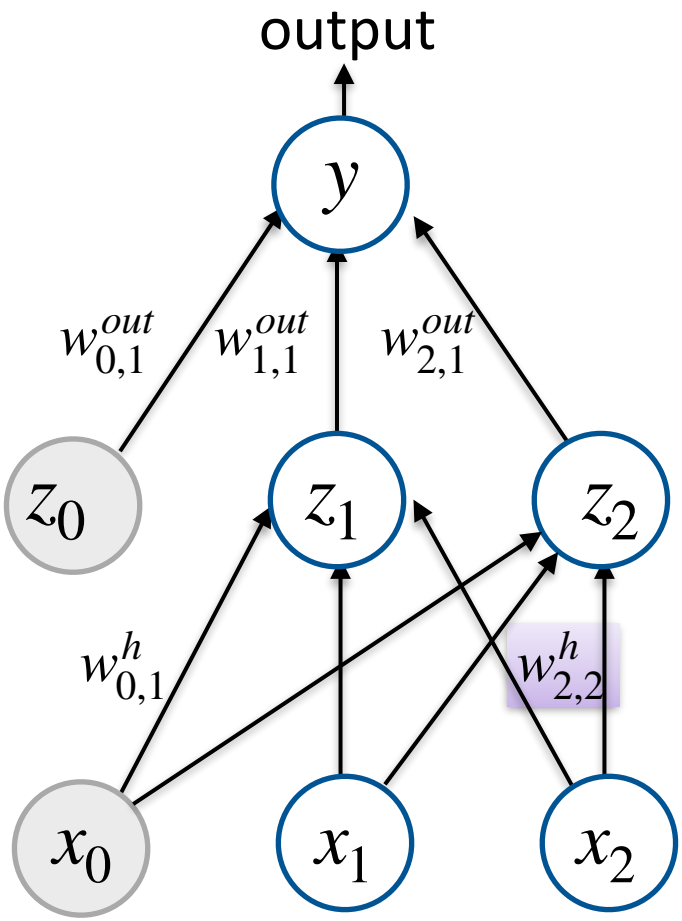
$$L = \frac{1}{2}(y - y^*)^2$$

$$y = w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

$\sigma(s)$



from previous slide

$$\frac{\partial L}{\partial w^h_{2,2}} = \frac{\partial L}{\partial y} \cdot w^o_{2,1} \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w^h_{2,2}}$$

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

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

$$\frac{\partial s}{\partial w^h_{2,2}} = x_2$$

Because $z_2 = \sigma(s) = \frac{1}{1 + e^{-s}}$ is the sigmoid function which has the derivative $\sigma(s)(1 - \sigma(s))$

Question: what gets computed if predicted output y perfectly?

Multiply by 0 so no update!

Stochastic gradient descent

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

The objective is not convex
Initialization can be important

Given a training set $S = \{(\mathbf{x_i}, y_i)\}, \mathbf{x} \in \Re^{\mathbf{d}}$

1. Initialize parameters $\mathbf{w}$
2. For epoch $= 1 \ldots T$:
   - Shuffle the training set
   - For each training example $(\mathbf{x_i}, y_i) \in S$:
     - ➡ Treat this example as the entire dataset

       Compute the gradient of the loss $\nabla L(NN(\mathbf{x_i}, \mathbf{w}), y_i)$ using backpropagation

     - ➡ Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$

γ_t: learning rate, many tweaks possible

Return $\mathbf{w}$

Backpropagation algorithm

The same algorithm works for multiple layers and more complicated architectures

Repeated application of the chain rule for partial derivatives

- First perform forward pass from inputs to the output
- Compute loss
- From loss, proceed backwards to compute partial derivatives using chain rule
- Cache partial derivatives as you compute them to use for lower layers

Mechanizing learning

Backpropagation gives gradient that will be used for gradient descent

- SGD gives a generic learning algorithm
- Backpropagation is a generic method for computing partial derivatives

A recursive algorithm that works from top of neural network to bottom

Modern neural network libraries implement automatic differentiation using backpropagation

- Allows easy exploration of network architectures
- Don’t have to keep deriving gradients by hand each time