

Neural Networks: Backpropagation

Machine Learning



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

Neural Networks

- What is a neural network?
- Predicting with a neural network
- Training neural networks
- Practical concerns

This lecture

- What is a neural network?
- Predicting with a neural network
- Training neural networks
 - Backpropagation
- Practical concerns

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:** $y = NN(\mathbf{x}, \mathbf{w})$

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

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

Perhaps with a *regularizer*

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

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

Perhaps with a *regularizer*

So far, we saw that this strategy worked for:

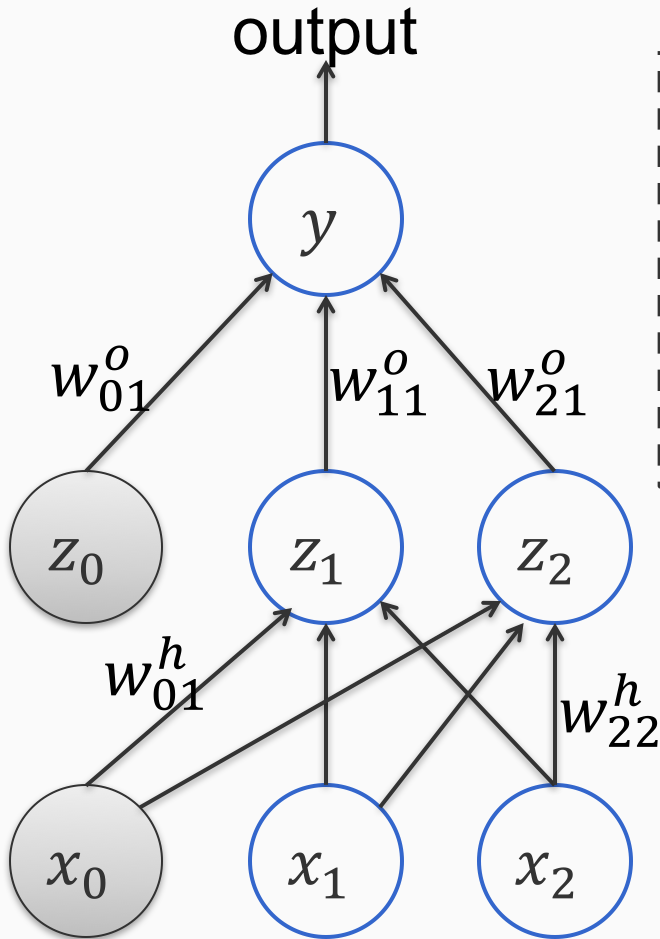
- | | |
|----------------------------|---|
| 1. Logistic Regression | Each
minimizes a
different loss
function |
| 2. Support Vector Machines | |
| 3. Perceptron | |
| 4. LMS regression | |

All of these are linear models

Same idea for non-linear models too!

Back to our running example

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



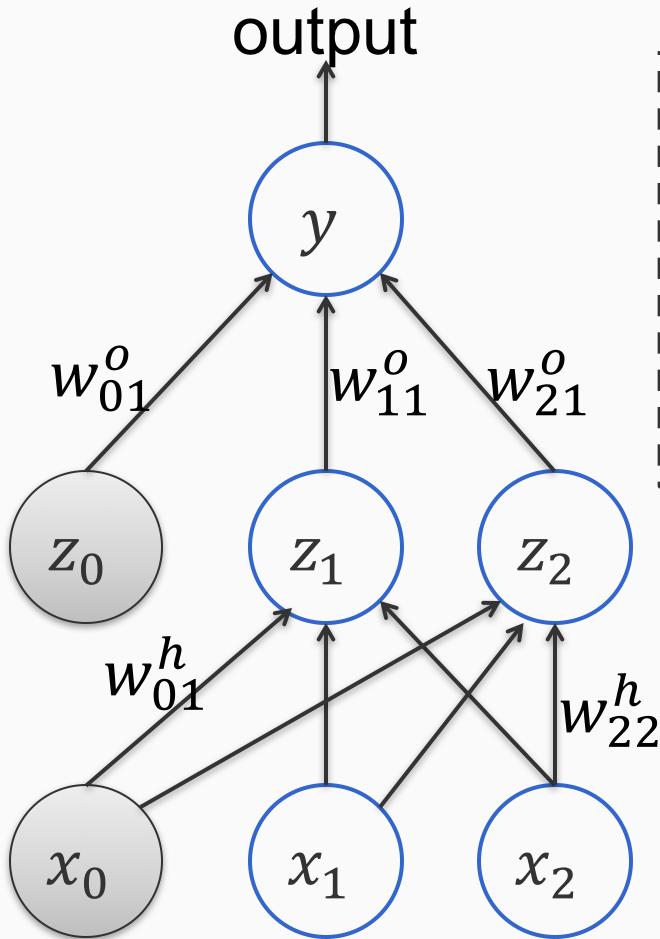
$$\text{output } y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$$

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

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

Back to our running example

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



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

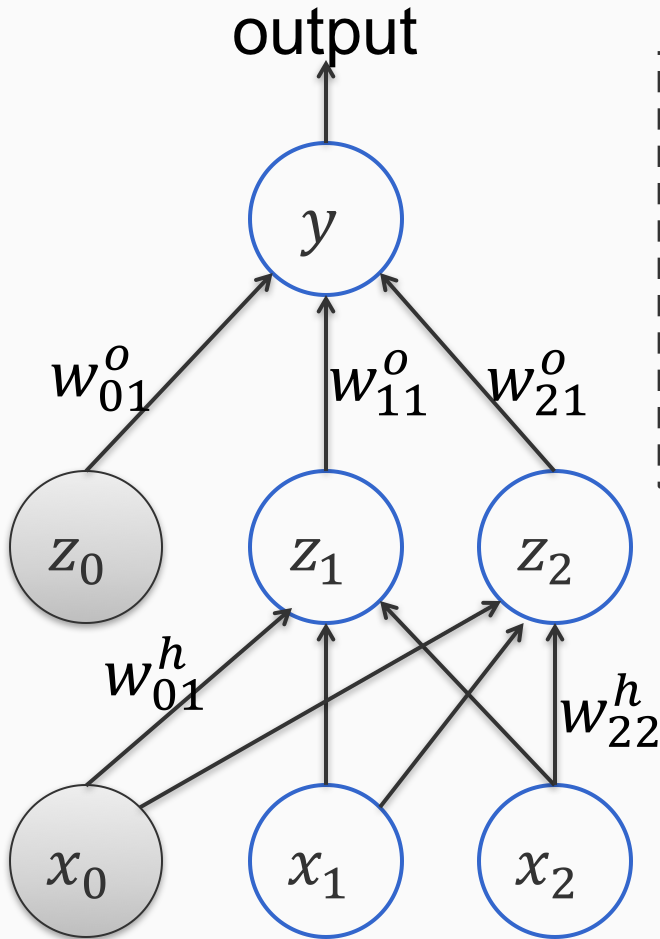
$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

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

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

Back to our running example

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



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

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

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

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 $S = \{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \mathbb{R}^d$

1. Initialize parameters w
2. For epoch = 1 ... T:
 3. Return w

$$\min_w \sum_i L(NN(x_i, w), y_i)$$

Stochastic gradient descent

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

1. Initialize parameters w
2. For epoch = 1 ... T:
 1. Shuffle the training set
3. Return w

$$\min_w \sum_i L(NN(x_i, w), y_i)$$

Stochastic gradient descent

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

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

$$\min_w \sum_i L(NN(x_i, w), y_i)$$

Stochastic gradient descent

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

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

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

Stochastic gradient descent

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

1. Initialize parameters \mathbf{w}
2. For epoch = 1 ... T:
 1. Shuffle the training set
 2. 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: $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$
3. Return \mathbf{w}

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

Stochastic gradient descent

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

1. Initialize parameters \mathbf{w}
2. For epoch = 1 ... T:
 1. Shuffle the training set
 2. 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: $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$
3. Return \mathbf{w}

γ_t : learning rate, many
tweaks possible

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

Stochastic gradient descent

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

1. Initialize parameters \mathbf{w}

2. For epoch = 1 ... T:

1. Shuffle the training set

2. 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: $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$

3. Return \mathbf{w}

The objective is **not convex**.
Initialization can be important

γ_t : learning rate, many
tweaks possible

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

Stochastic gradient descent

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

1. Initialize parameters \mathbf{w}

The objective is not convex.
Initialization can be important

2. For epoch = 1 ... T:

1. Shuffle the training set

2. 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

3. 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
- 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?

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

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

- Or maybe its sub-gradient
- 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!

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
- 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

Checkpoint

Where are we

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

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

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

Checkpoint

Where are we

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

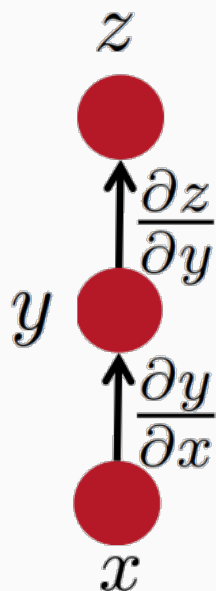
If we had the true label of the input, then we can define the loss for that example

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

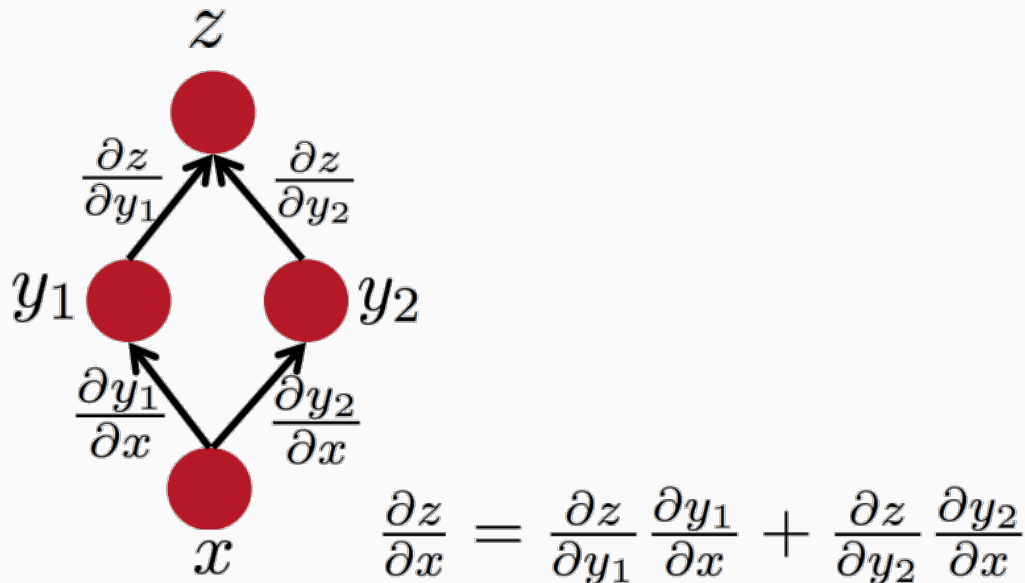
- If z is a function of y and y is a function of x
 - Then z is a function of x , as well
- Question: how to find $\frac{\partial z}{\partial x}$



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

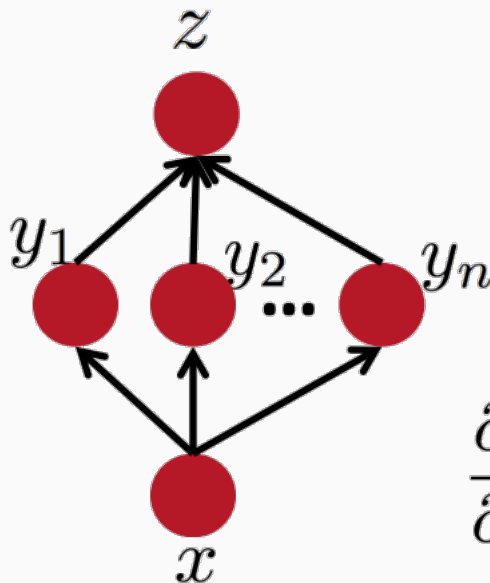
Reminder: Chain rule for derivatives

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



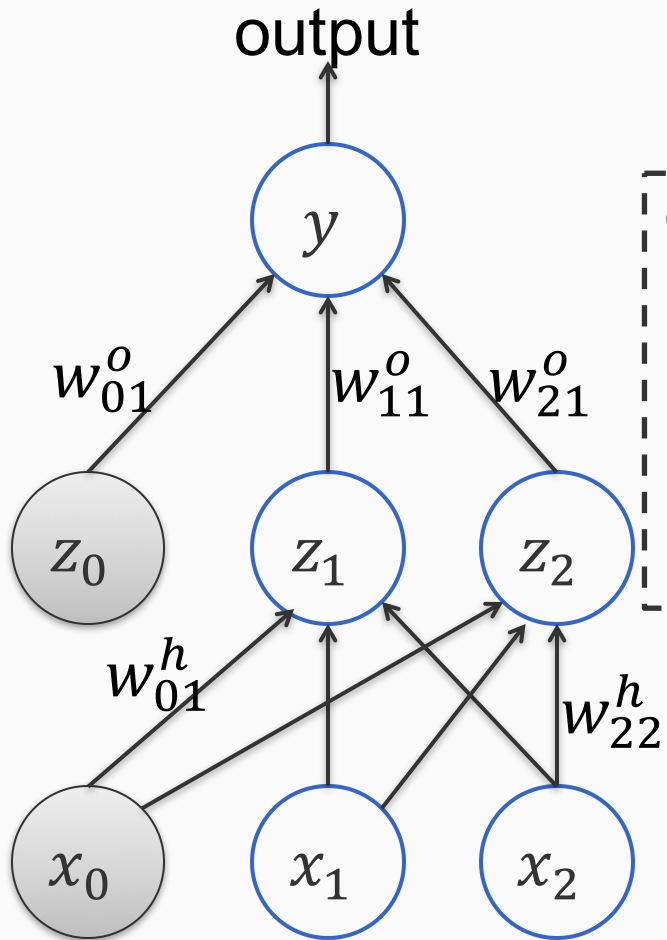
Reminder: Chain rule for derivatives

- 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



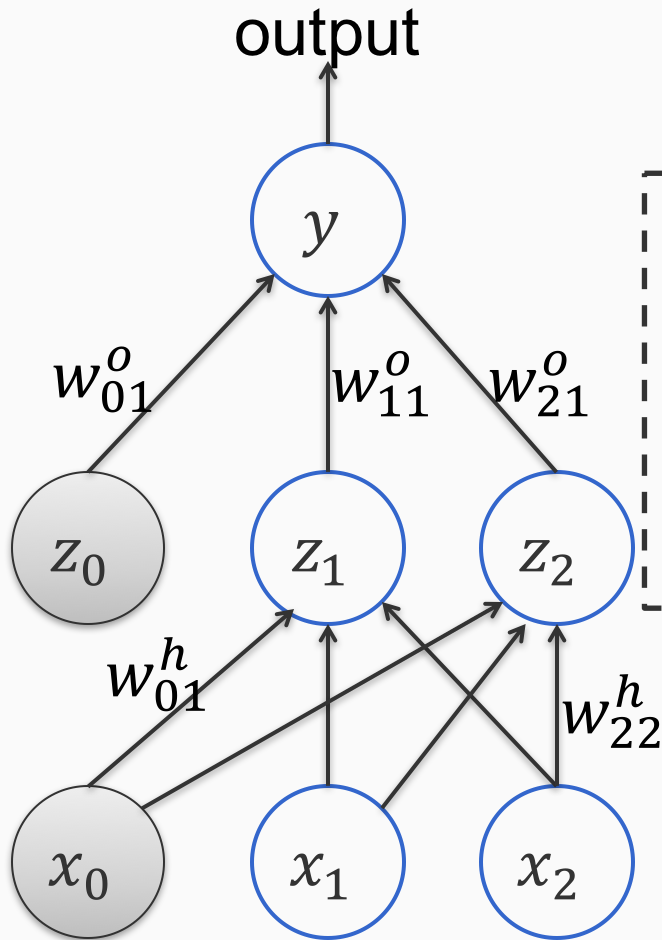
$$L = \frac{1}{2}(y - y^*)^2$$

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

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

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

Backpropagation



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

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

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

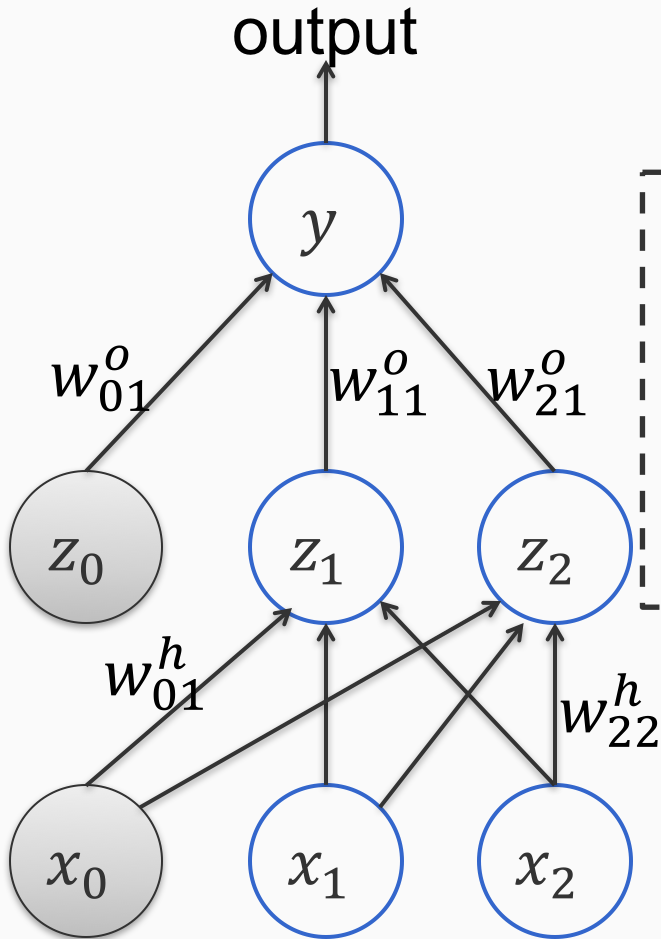
$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

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)



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

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

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

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

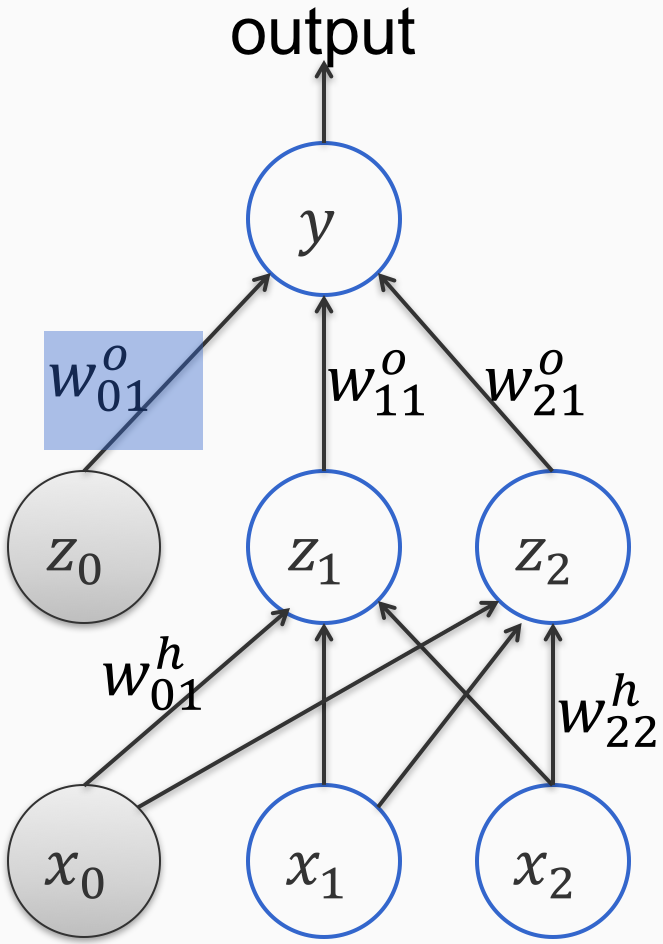
We want to compute

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

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

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

Output layer

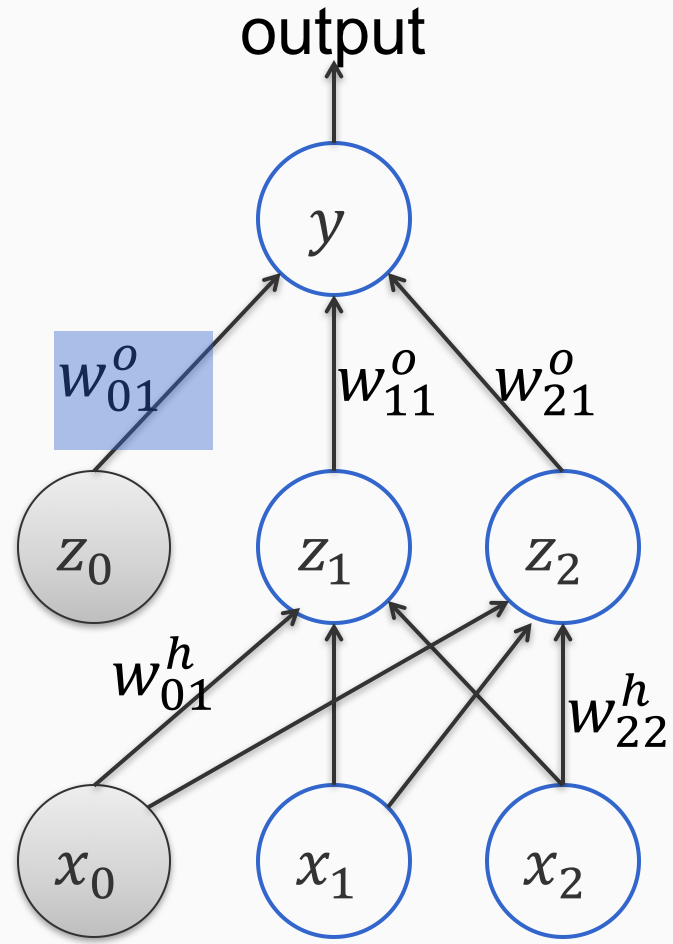


$$\frac{\partial L}{\partial w_{01}^o}$$

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

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

Output layer

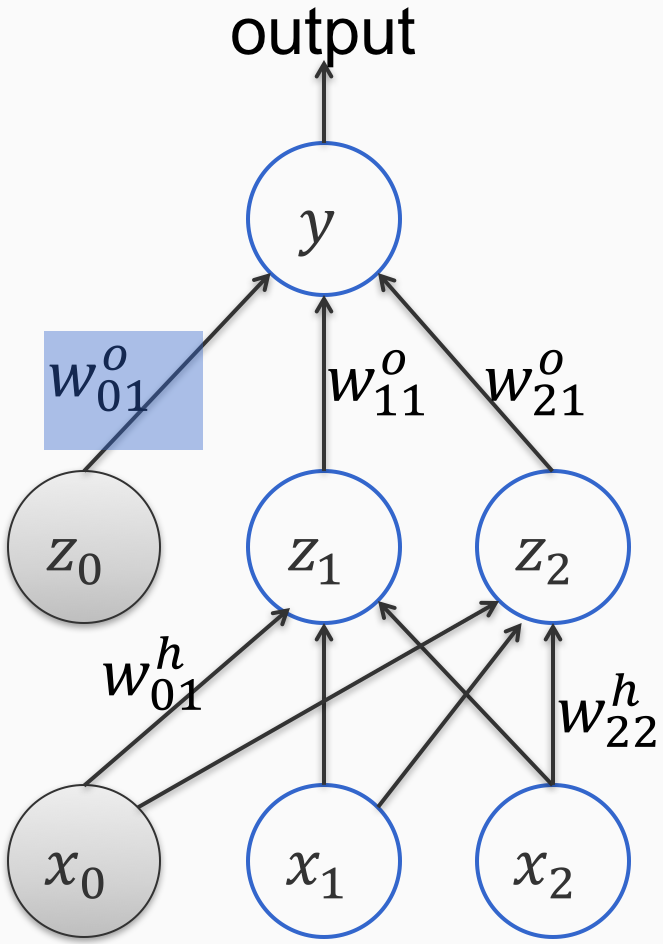


$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{01}^o}$$

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

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

Output layer



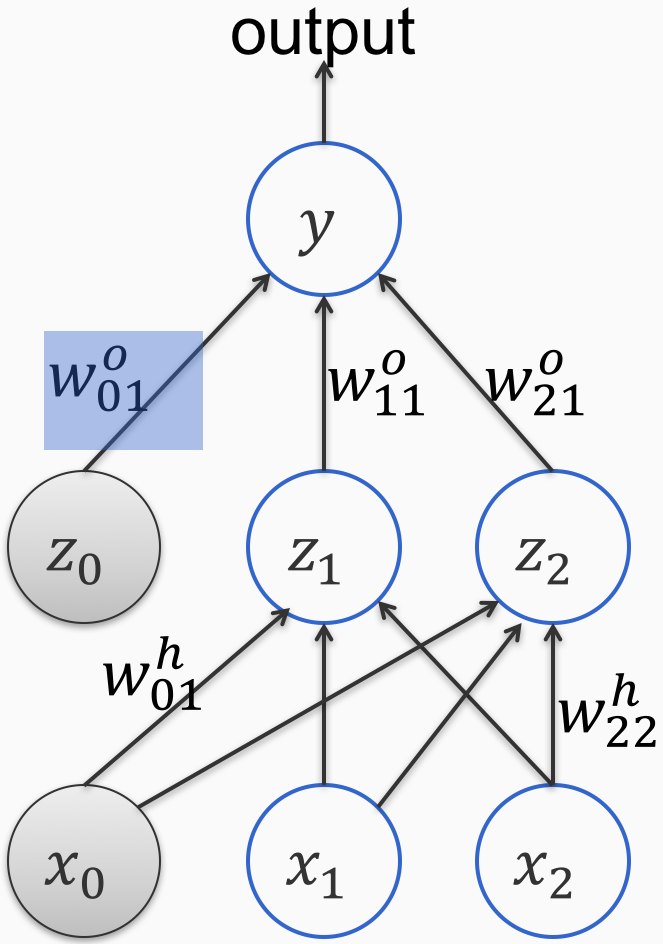
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{01}^o}$$

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

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

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

Output layer



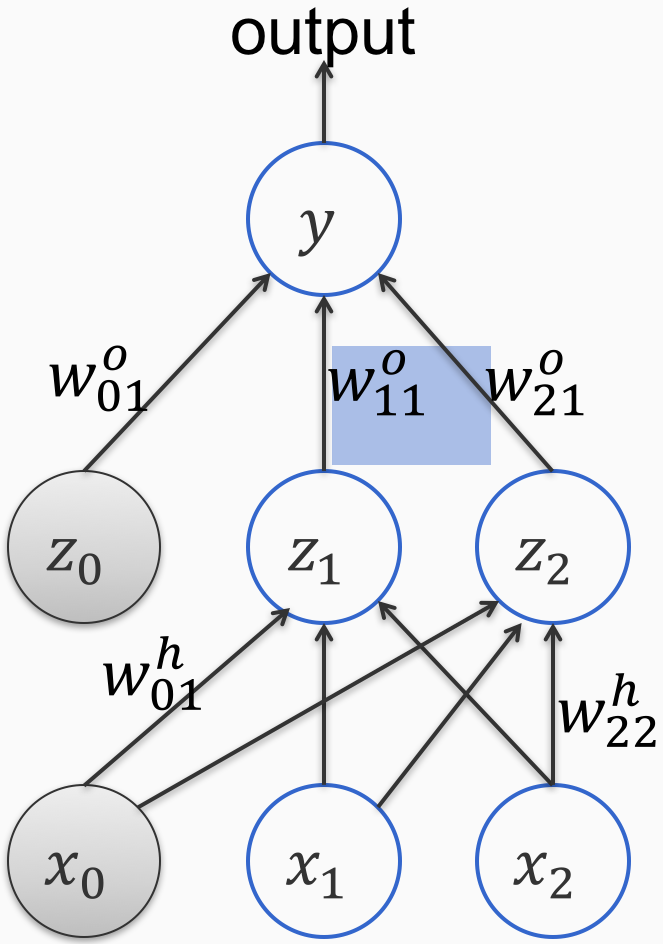
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{01}^o}$$

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

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

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

Output layer

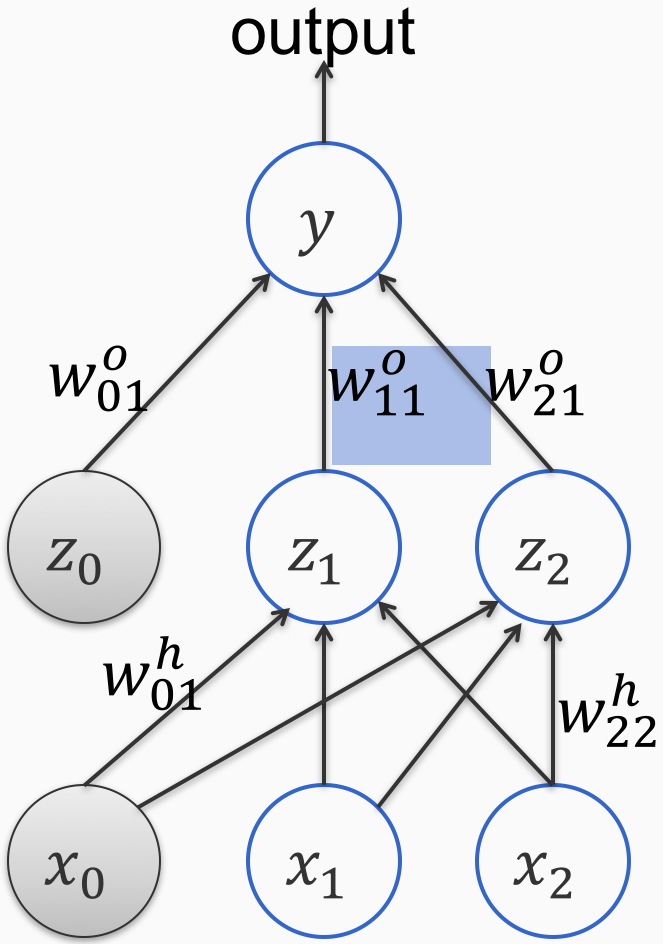


$$\frac{\partial L}{\partial w_{11}^o}$$

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

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

Output layer

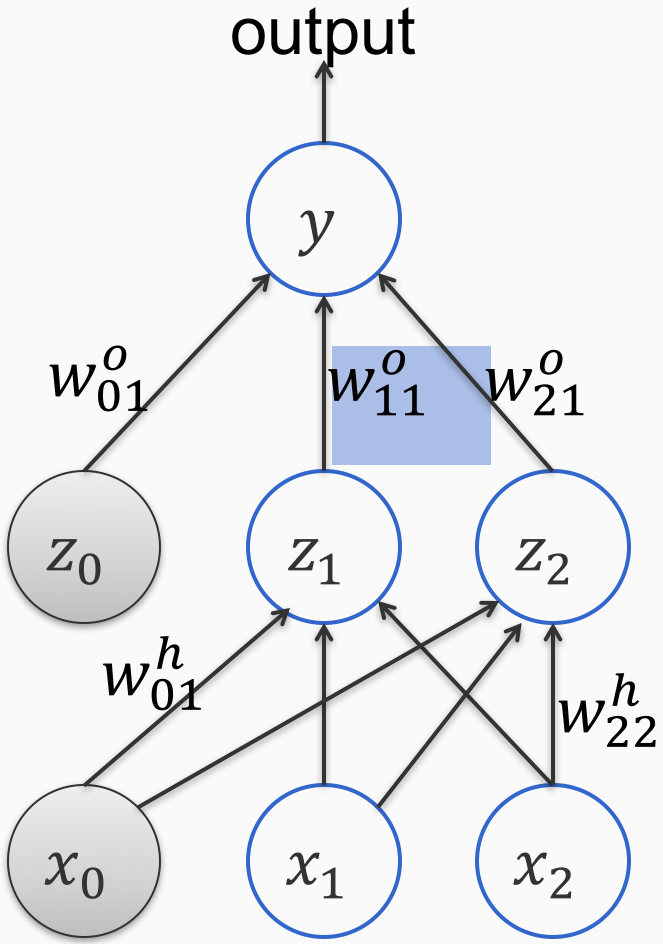


$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{11}^o}$$

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

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

Output layer



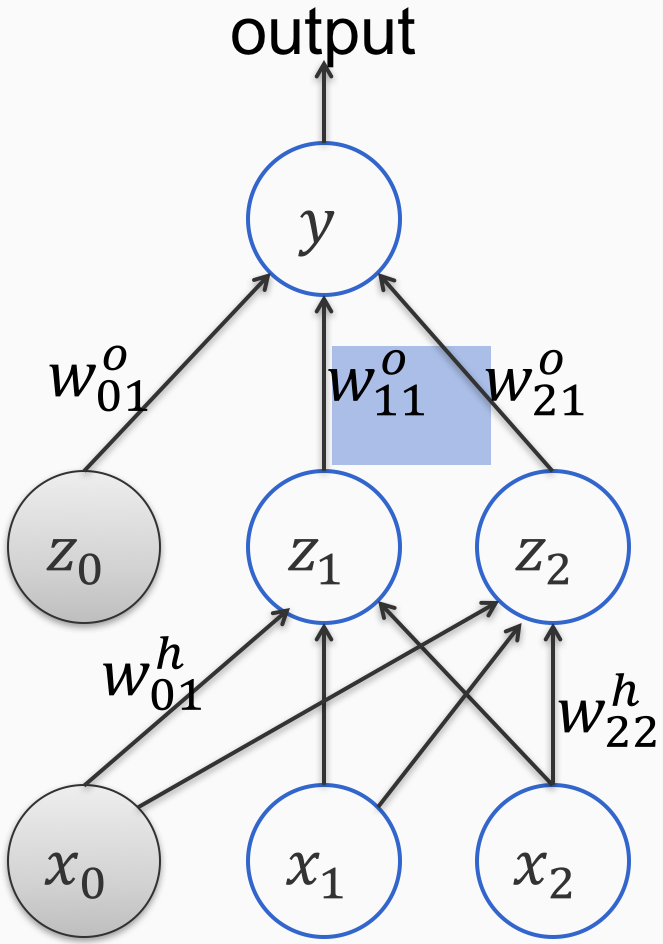
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{11}^o}$$

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

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

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

Output layer



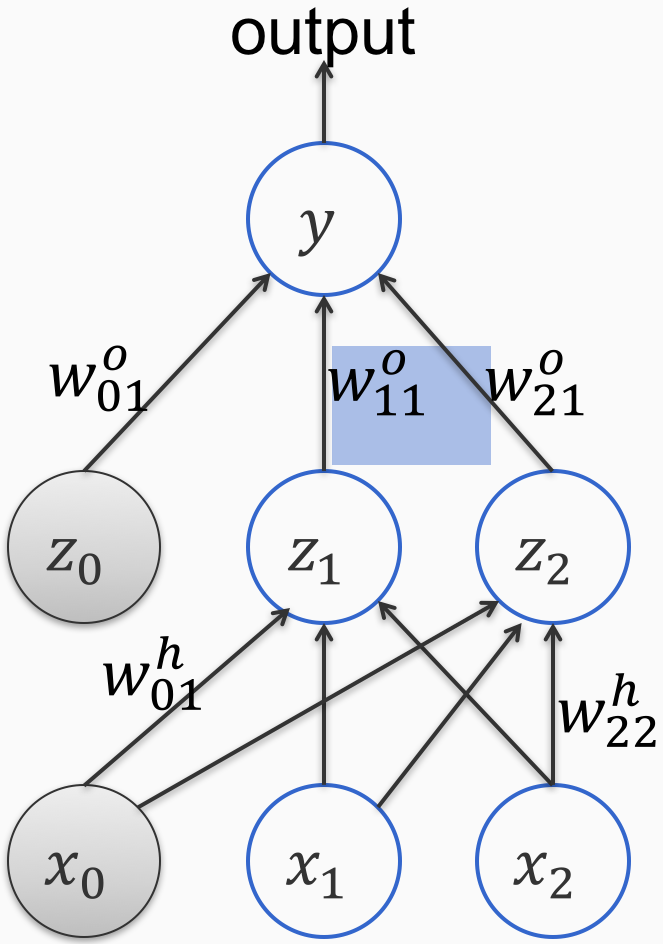
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{11}^o}$$

$$\frac{\partial L}{\partial y} = y - y^* \qquad \frac{\partial y}{\partial w_{11}^o} = z_1$$

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

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

Output layer



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

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

$$\frac{\partial y}{\partial w_{01}^o} = z_1$$

We have already computed this partial derivative for the previous case

Cache to speed up!

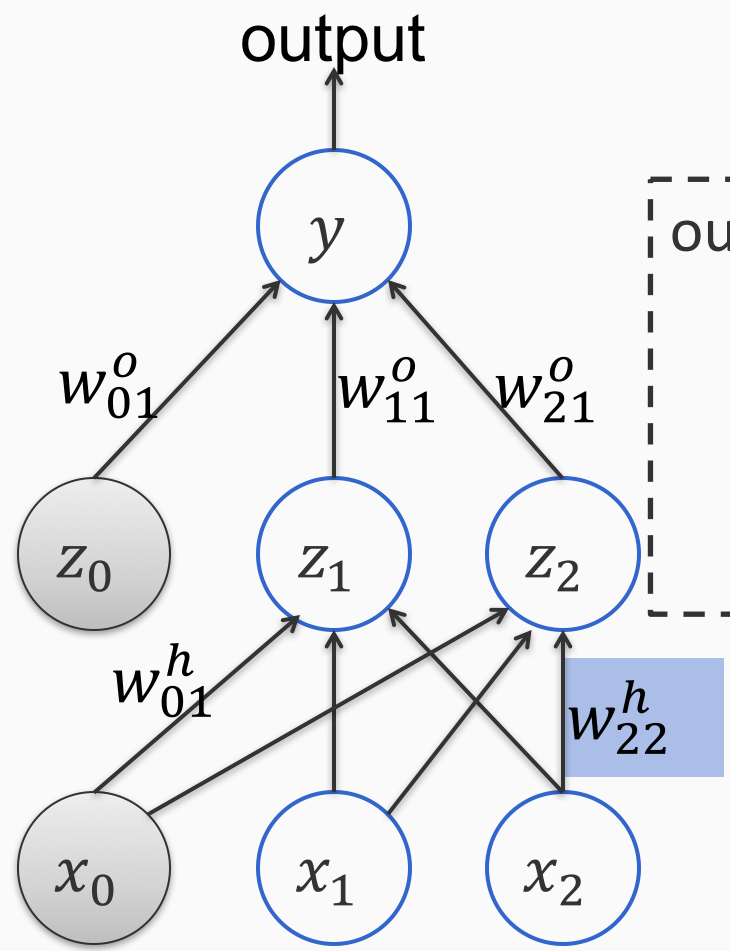
Hidden layer derivatives

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

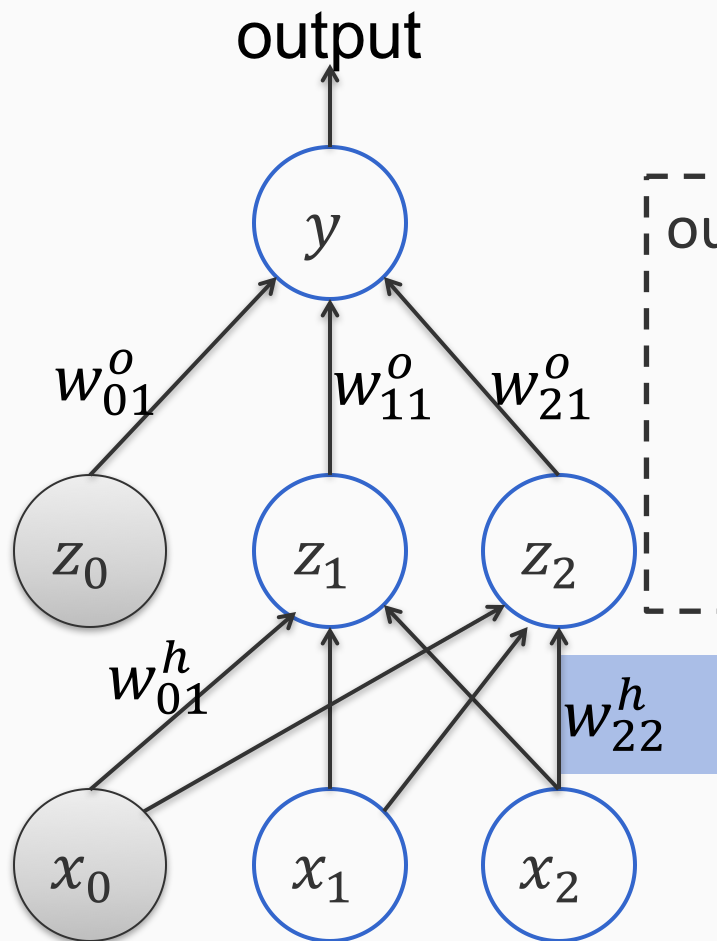
output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

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

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



Hidden layer derivatives



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

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

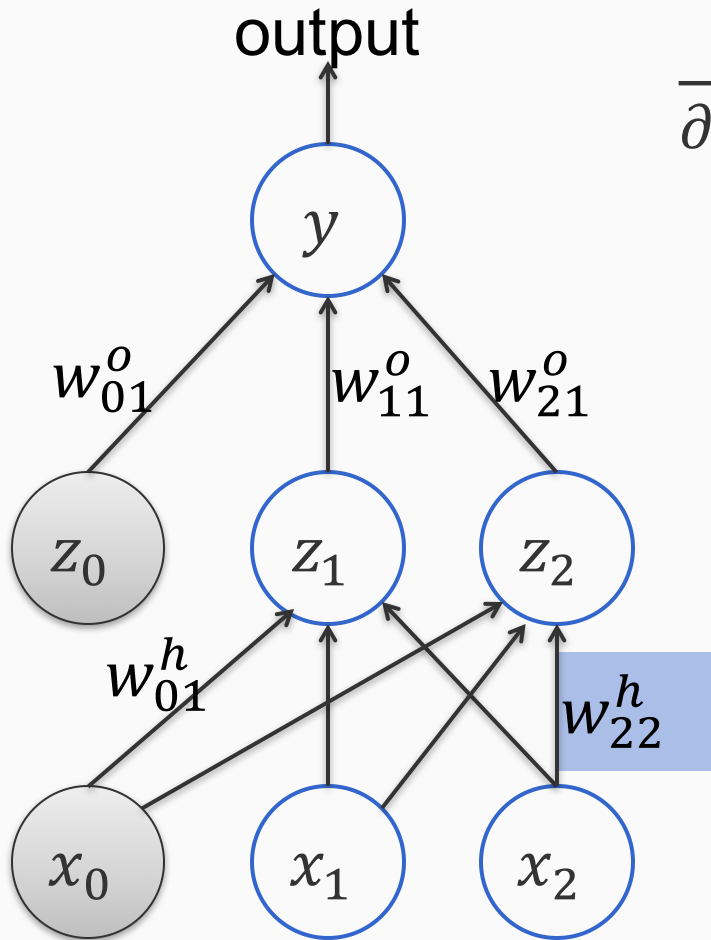
$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

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

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

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

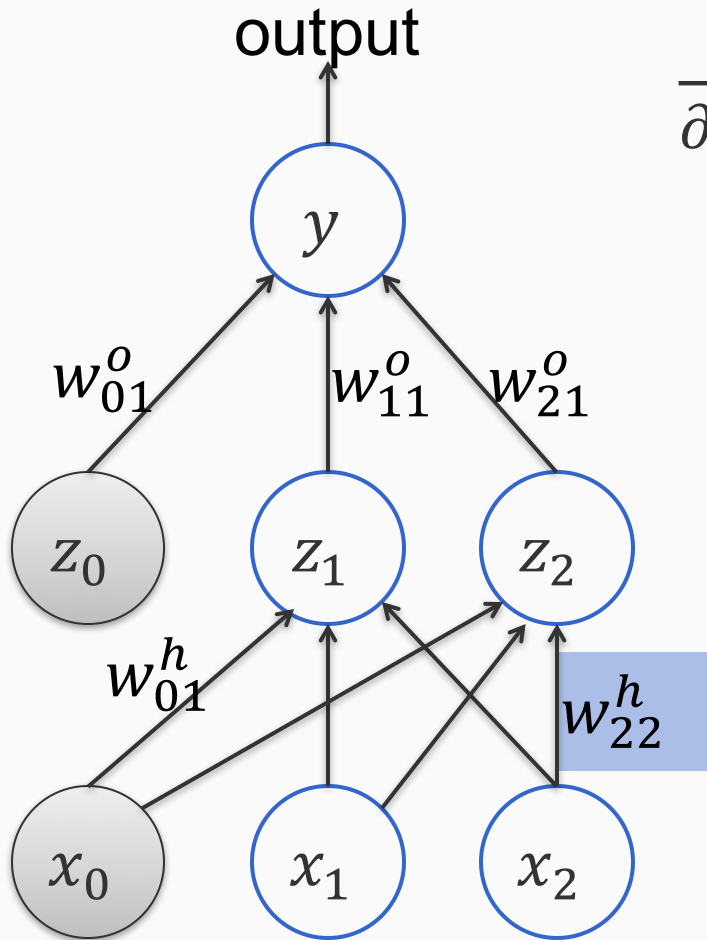
Hidden layer derivatives



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

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

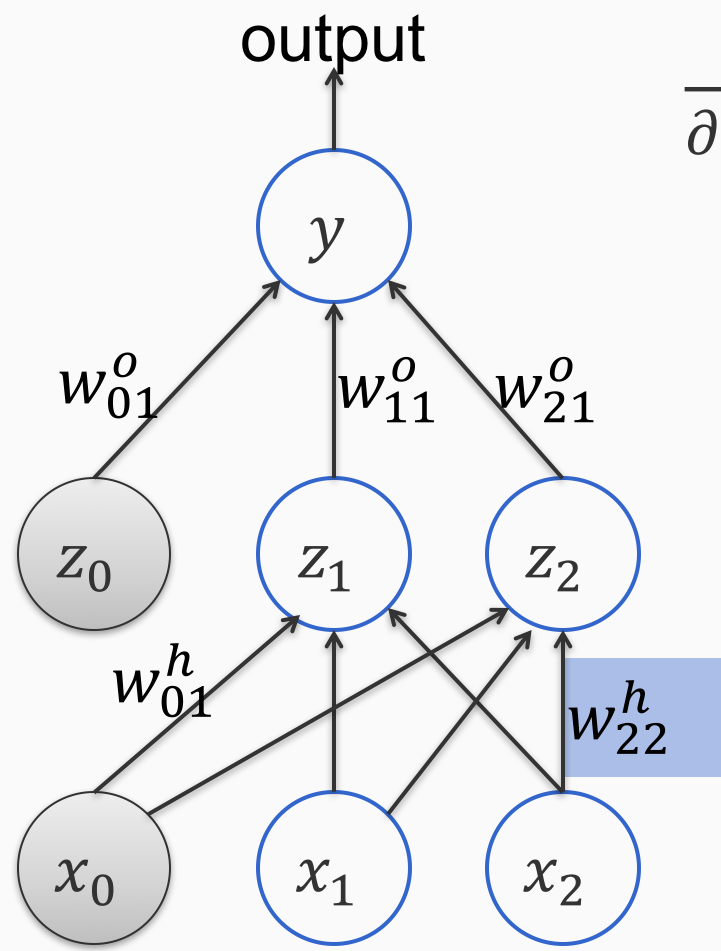
Hidden layer



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

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

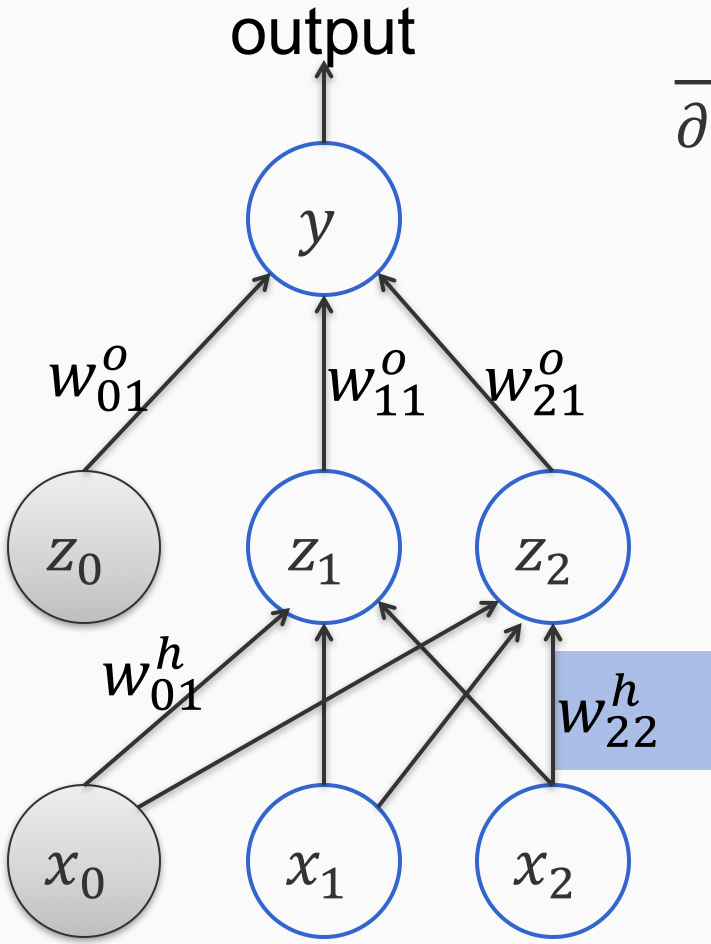
Hidden layer



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

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

Hidden layer



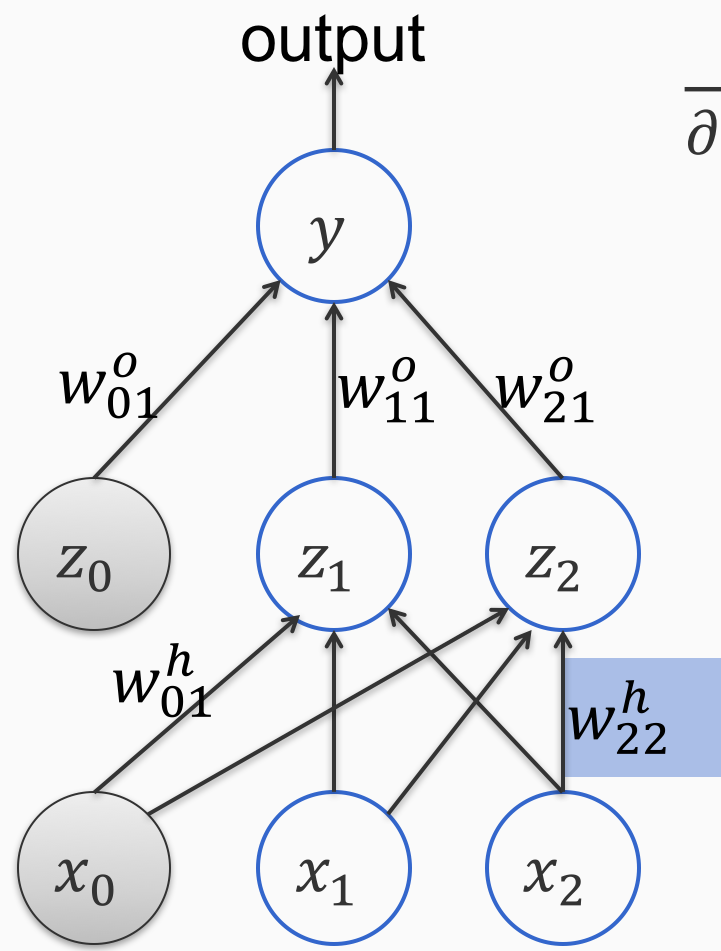
$$\begin{aligned} \frac{\partial L}{\partial w_{22}^h} &= \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{22}^h} \\ &= \frac{\partial L}{\partial y} \frac{\partial}{\partial w_{22}^h} (w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2) \\ &= \frac{\partial L}{\partial y} (w_{11}^o \frac{\partial}{\partial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

0

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

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

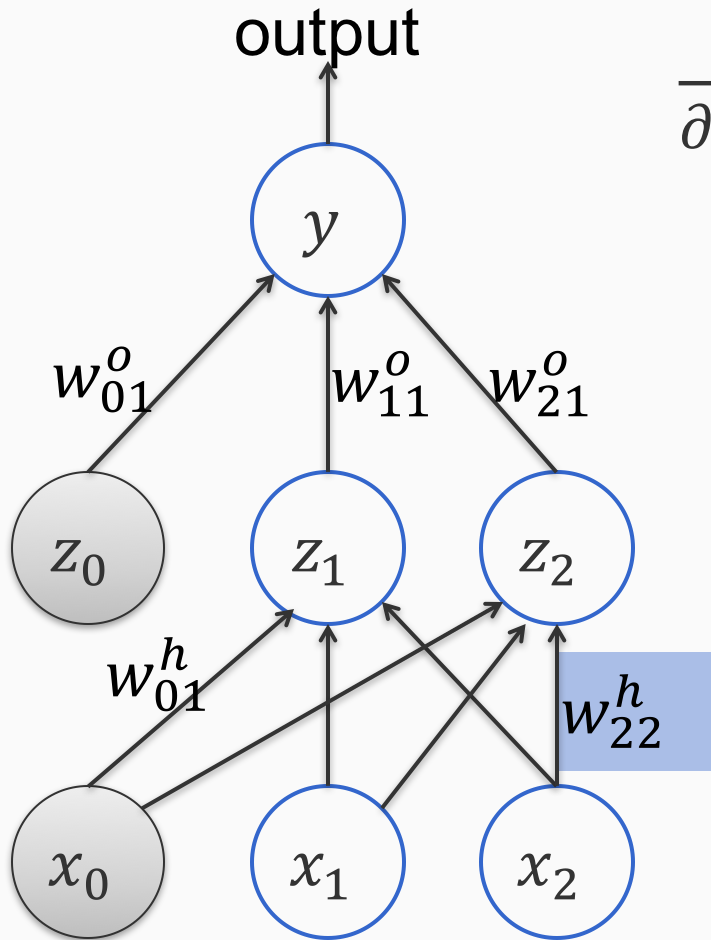
Hidden layer



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

Hidden layer

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

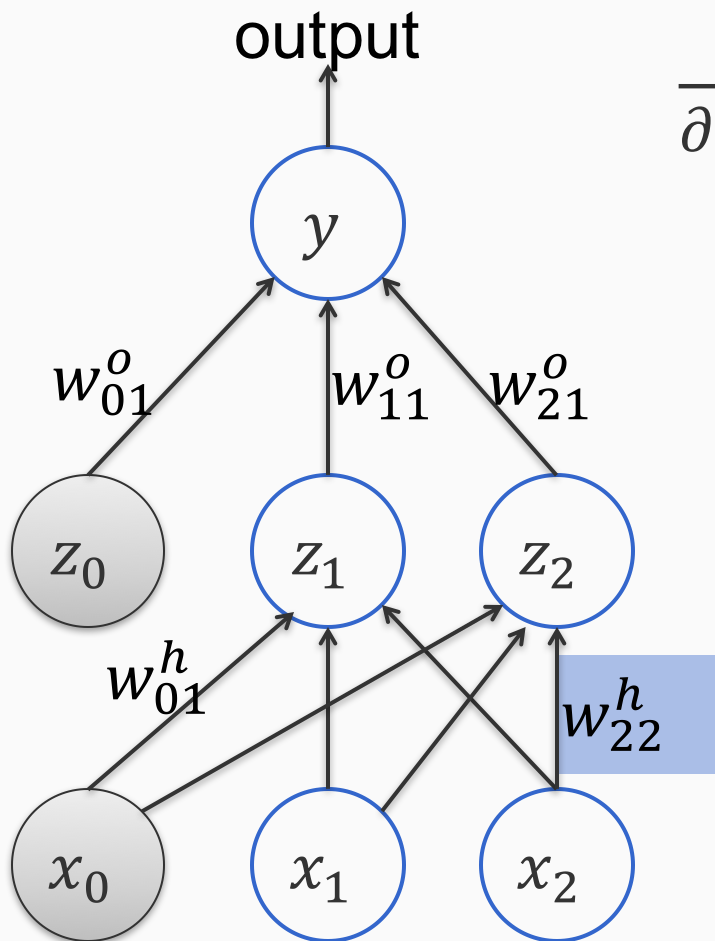


$$\begin{aligned} \frac{\partial L}{\partial w_{22}^h} &= \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{22}^h} \\ &= \frac{\partial L}{\partial y} \frac{\partial}{\partial w_{22}^h} (w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2) \\ &= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial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

Hidden layer

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

Call this s



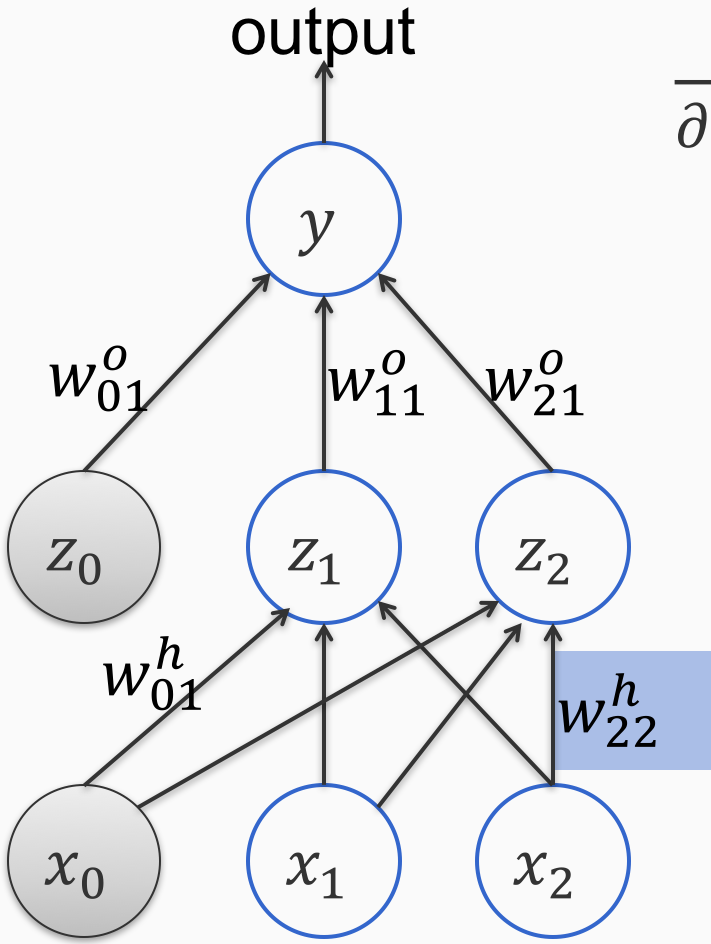
$$\begin{aligned} \frac{\partial L}{\partial w_{22}^h} &= \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{22}^h} \\ &= \frac{\partial L}{\partial y} \frac{\partial}{\partial w_{22}^h} (w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2) \\ &= \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial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

Hidden layer

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



Call this s



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

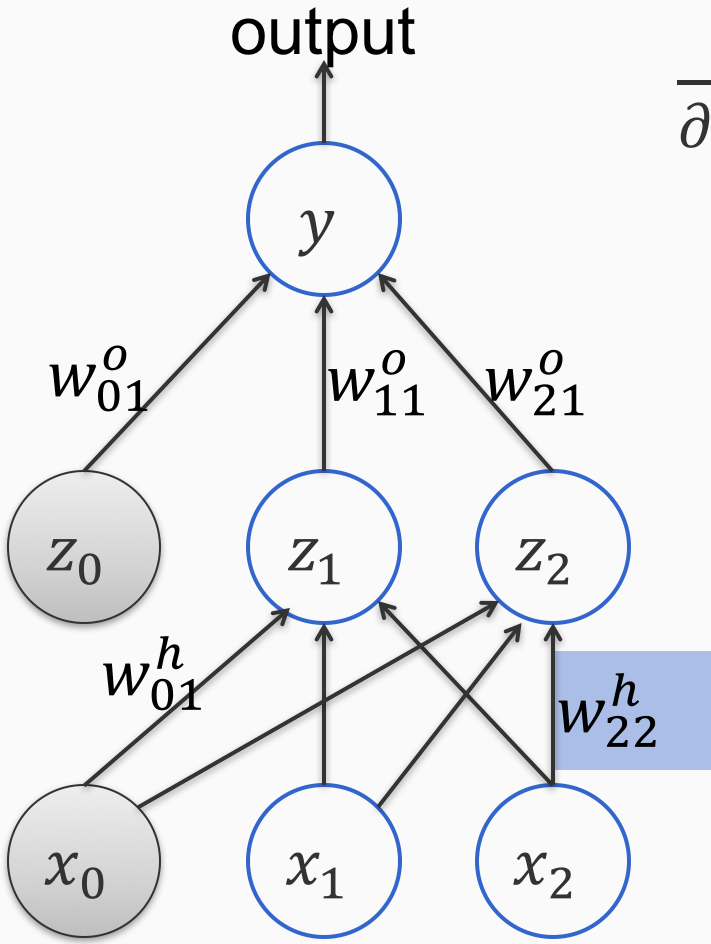
Hidden layer

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



Call this s

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

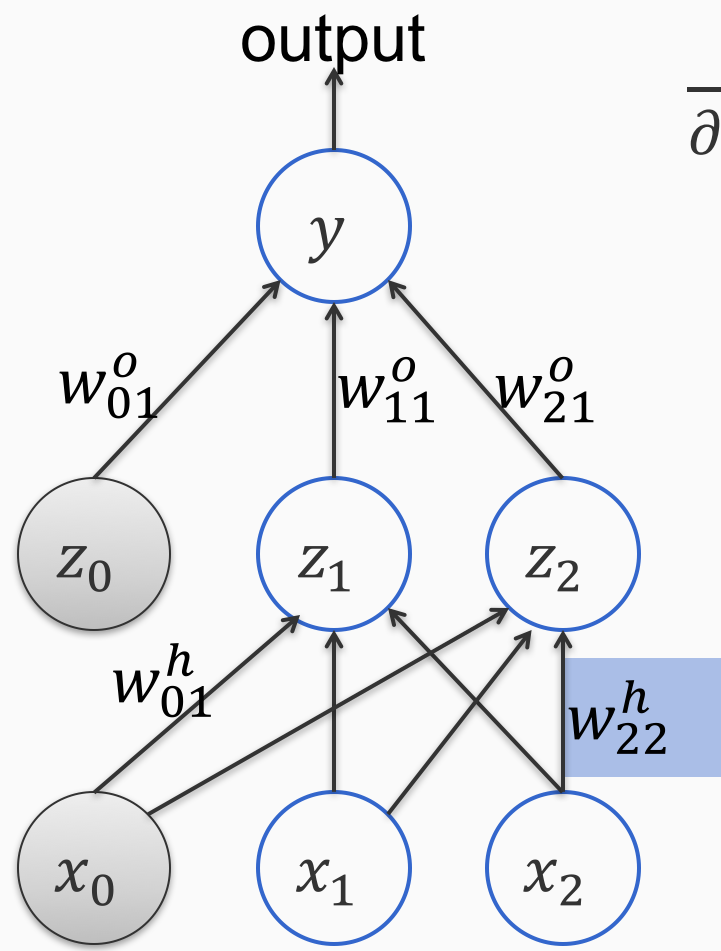


$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$



Call this s

Hidden layer



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

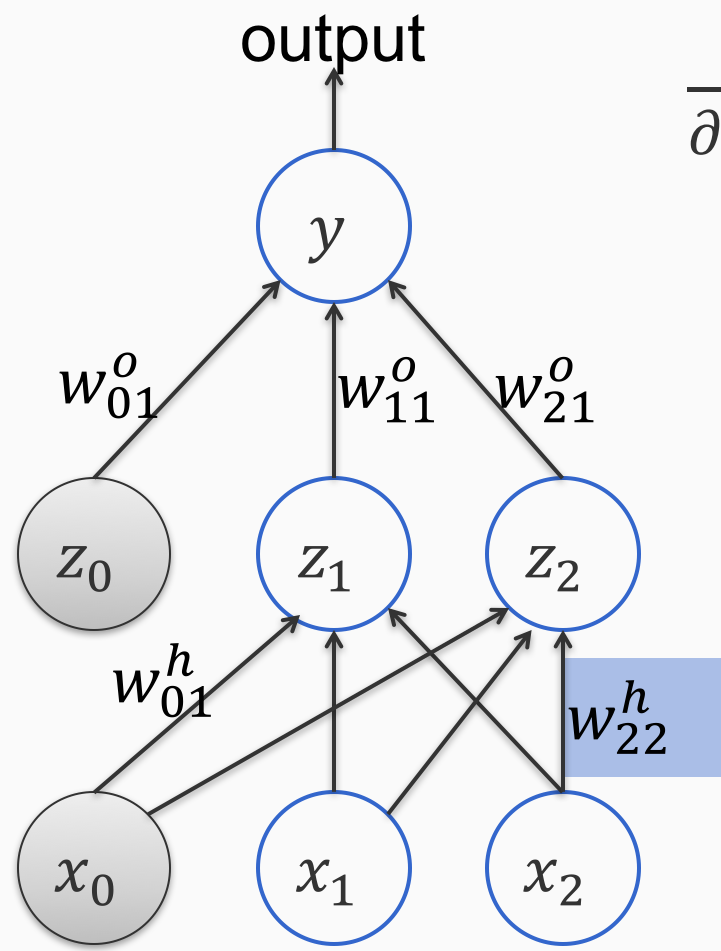
Each of these partial derivatives is easy

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



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

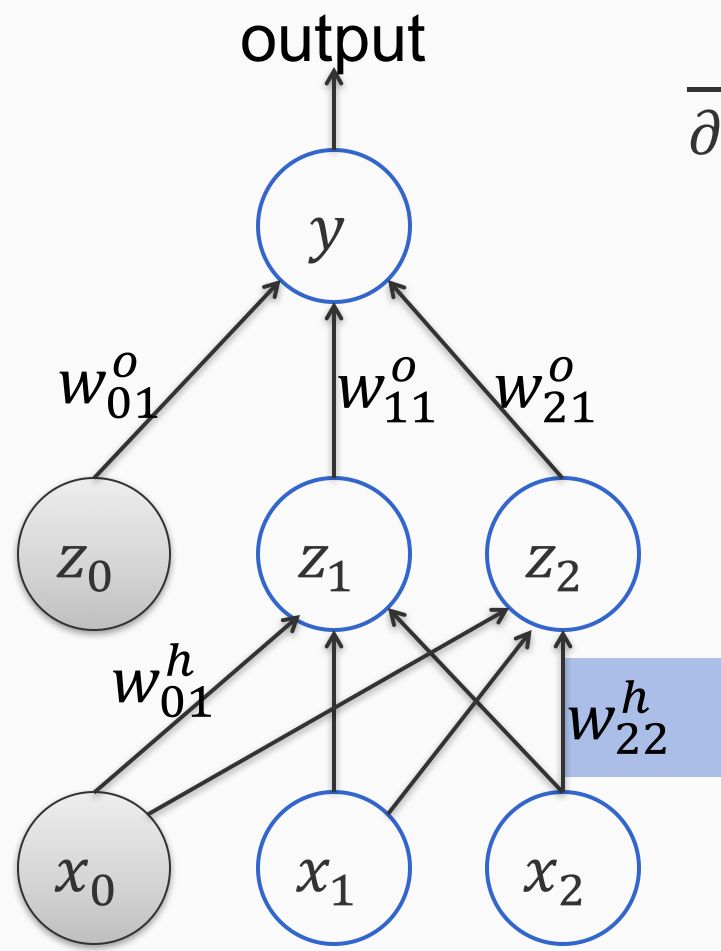
$$\frac{\partial L}{\partial y} = y - y^*$$

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



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

Hidden layer

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



Call this s

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

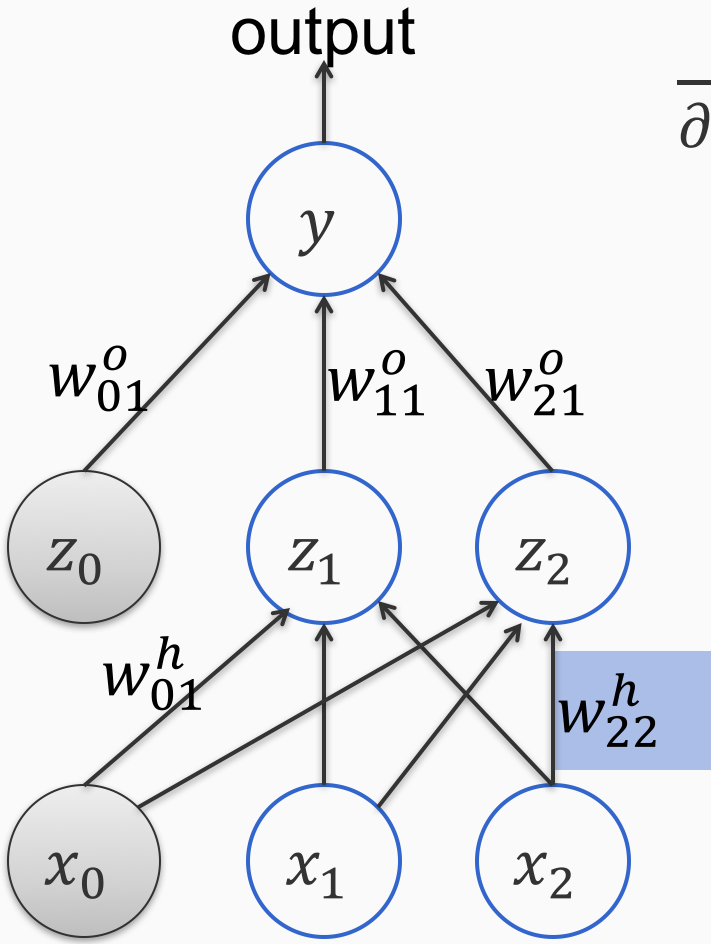
Each of these partial derivatives is easy

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

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

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

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



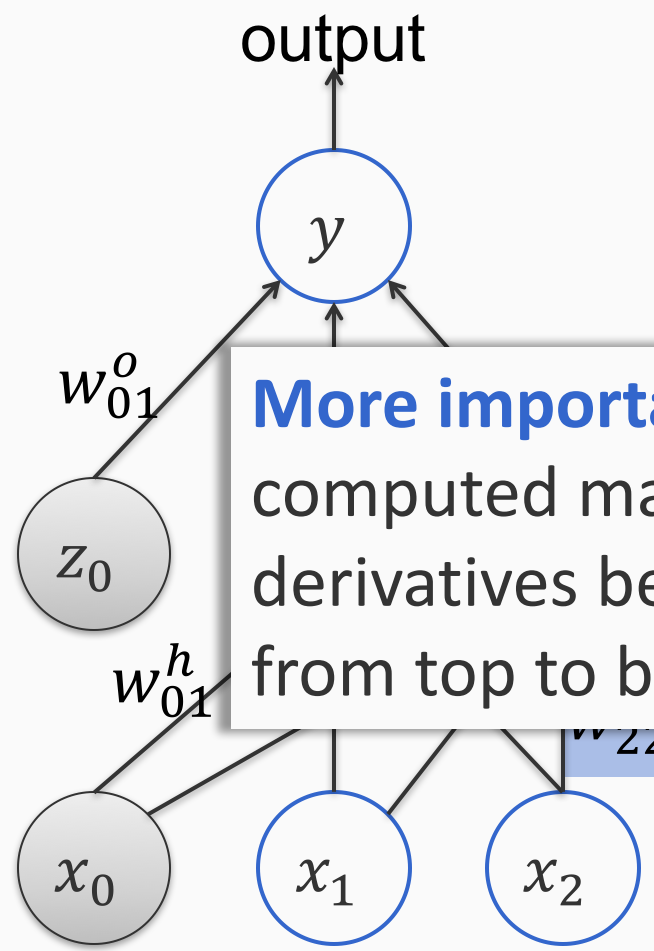
$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

Hidden layer

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 (i.e. backwards)

cause $z_2(s)$
 istic
 we have
 een

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

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

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

Stochastic gradient descent

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

1. Initialize parameters \mathbf{w}
2. For epoch = 1 ... T:
 1. Shuffle the training set
 2. 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: $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla L(NN(\mathbf{x}_i, \mathbf{w}), y_i)$
3. Return \mathbf{w}

The objective is **not convex**.
Initialization can be important

γ_t : learning rate, many
tweaks possible