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

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

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

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

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

Perhaps with a *regularizer*

So far, we saw that this strategy worked for:

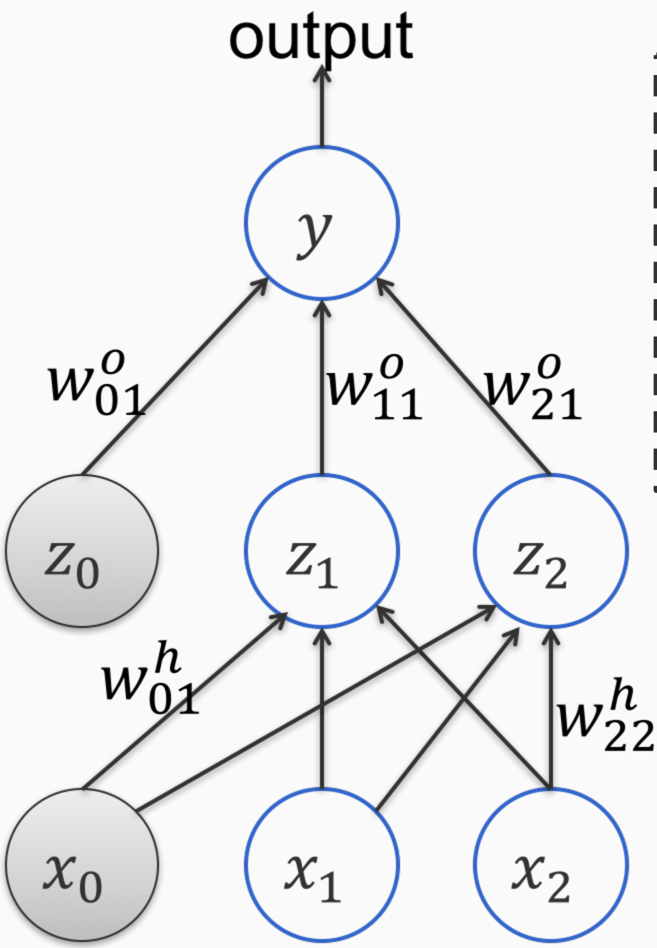
1. Logistic Regression
2. Support Vector Machines
3. Perceptron
4. LMS regression

Each minimizes a different loss function

All of these are linear models

Same idea for non-linear models too!

Back to our running example



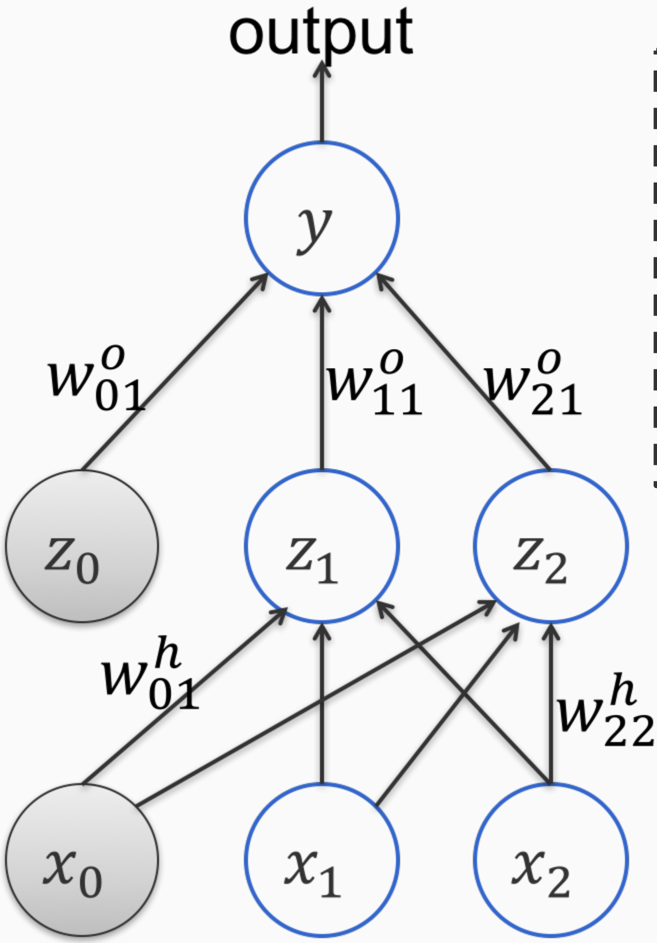
Given an input **x**, how is the output predicted

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

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

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

Back to our running example



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

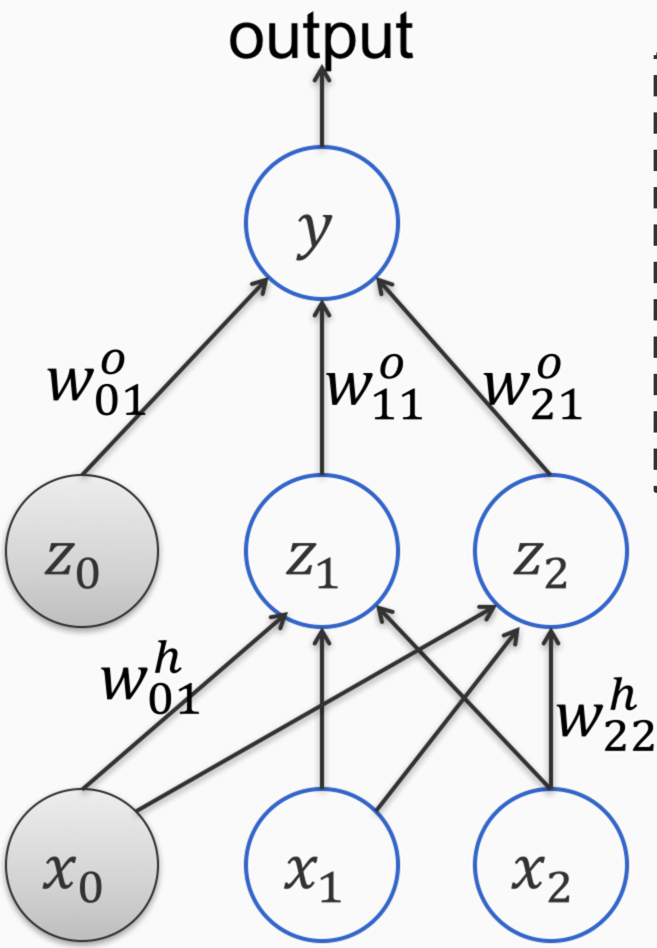
output $y = w_{01}^{o} + w_{11}^{o} z_1 + w_{21}^{o} z_2$

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

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

Suppose the true label for this example is a number y_i

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_i

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

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

Learning as loss minimization

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

Learn the weights by minimizing a loss L

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

Perhaps with a *regularizer*

How do we solve the optimization problem?

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

Stochastic gradient descent

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

1. Initialize parameters w
2. For epoch = 1 … T:

3. Return w

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

Stochastic gradient descent

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

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

3. Return w

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

Stochastic gradient descent

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

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

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

Stochastic gradient descent

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

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

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

Stochastic gradient descent

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

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

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

Stochastic gradient descent

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

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 $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

γ_t: learning rate,
many tweaks possible

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

Stochastic gradient descent

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

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 $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$
      - 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 $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
    1. Shuffle the training set
    2. For each training example $(\mathbf{x}_i, y_i) \in S$:
        - 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 ~150 layer neural network for image classification!

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 ~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

Let's look at some simple expressions

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

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

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

Let's look at some simple expressions

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

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

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

Let's look at some simple expressions

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

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

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

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

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

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

Let’s look at some simple expressions

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

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

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

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

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

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

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

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

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

Let's look at some simple expressions

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

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

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

$f(x, y) = xy$

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

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

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

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

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

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.

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

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

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 $\frac{\partial g}{\partial y}, \frac{\partial g}{\partial z}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial g}$

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**

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

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

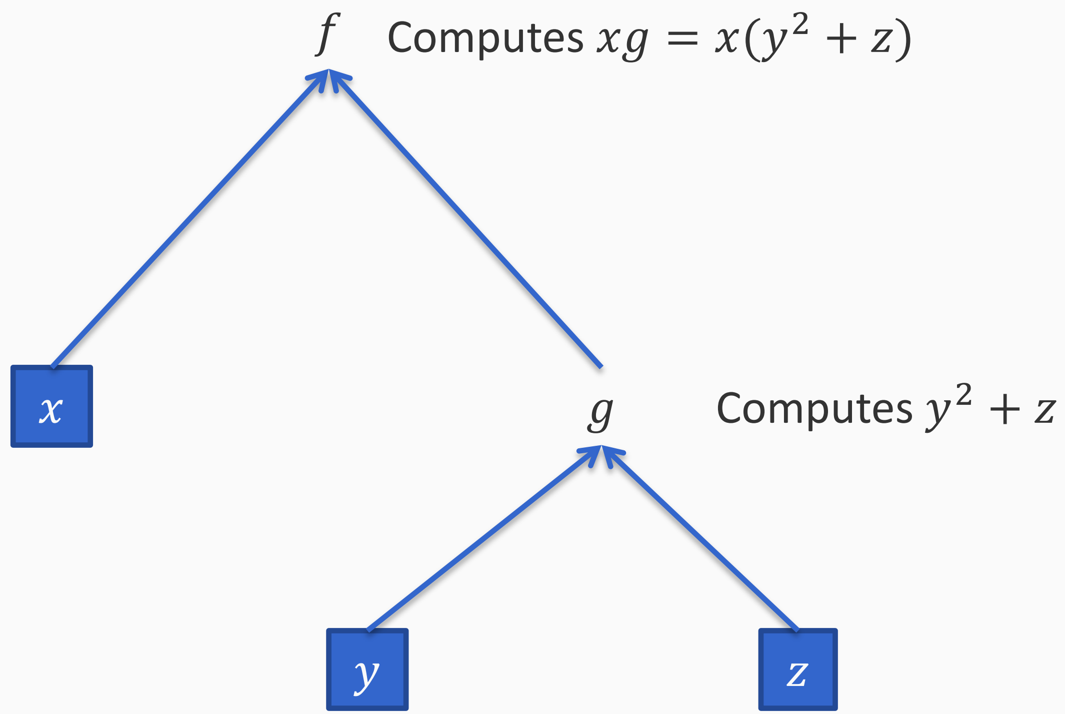
Each of these is a simple form. We know how to compute $\frac{\partial g}{\partial y}, \frac{\partial g}{\partial z}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial g}$

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

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

In terms of “computation graphs”

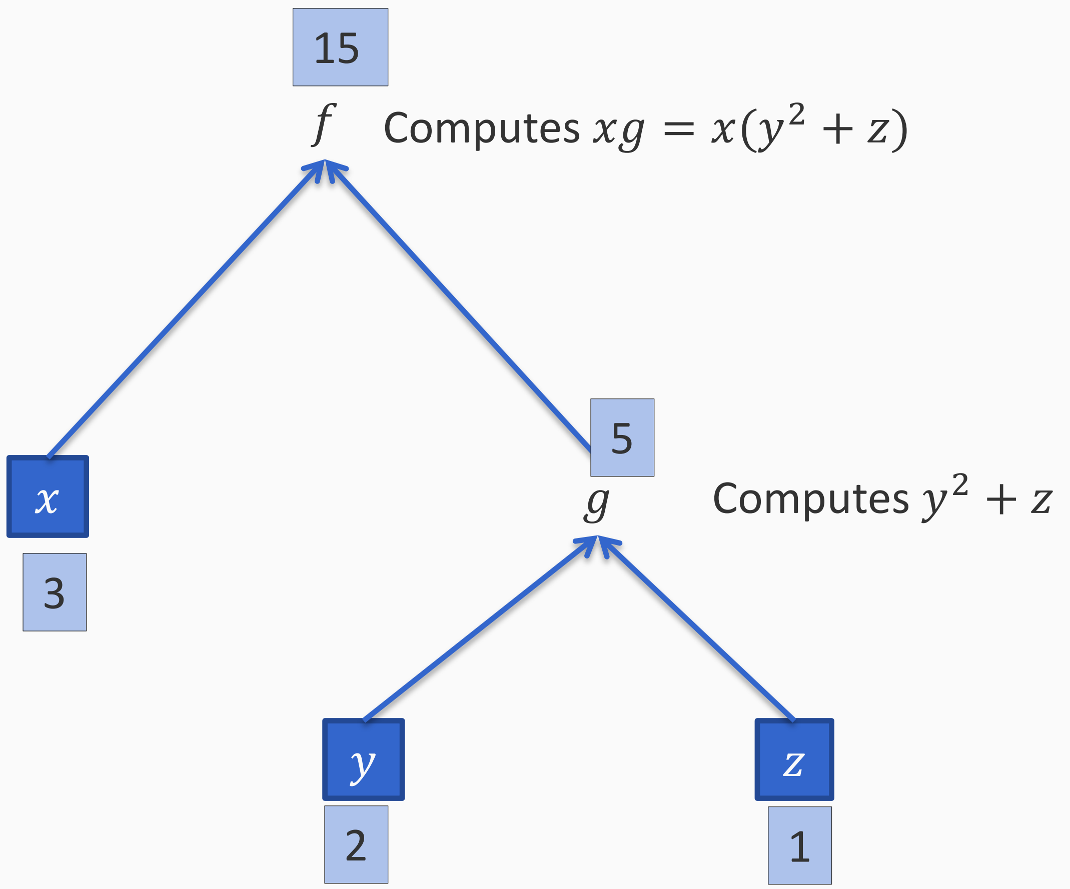
$$f(x, y, z) = x(y^2 + z)$$



In terms of "computation graphs"

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

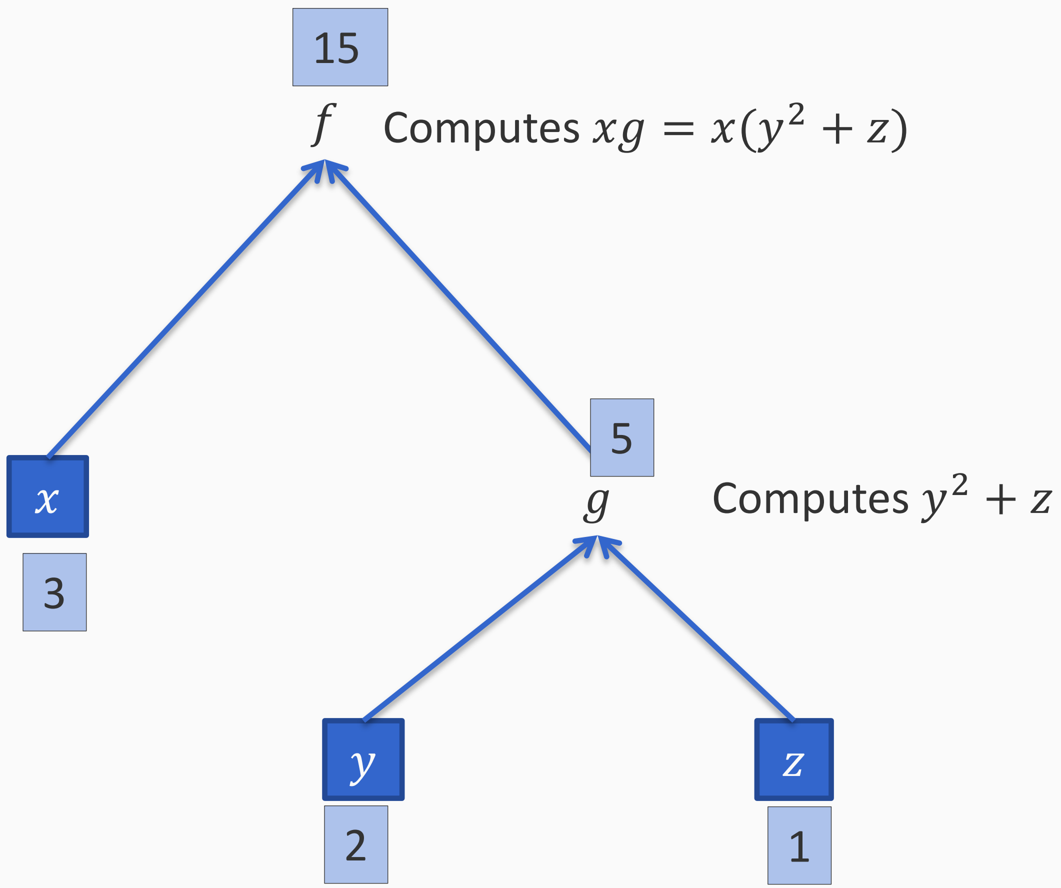
The forward pass: Computes function values for specific inputs



In terms of "computation graphs"

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

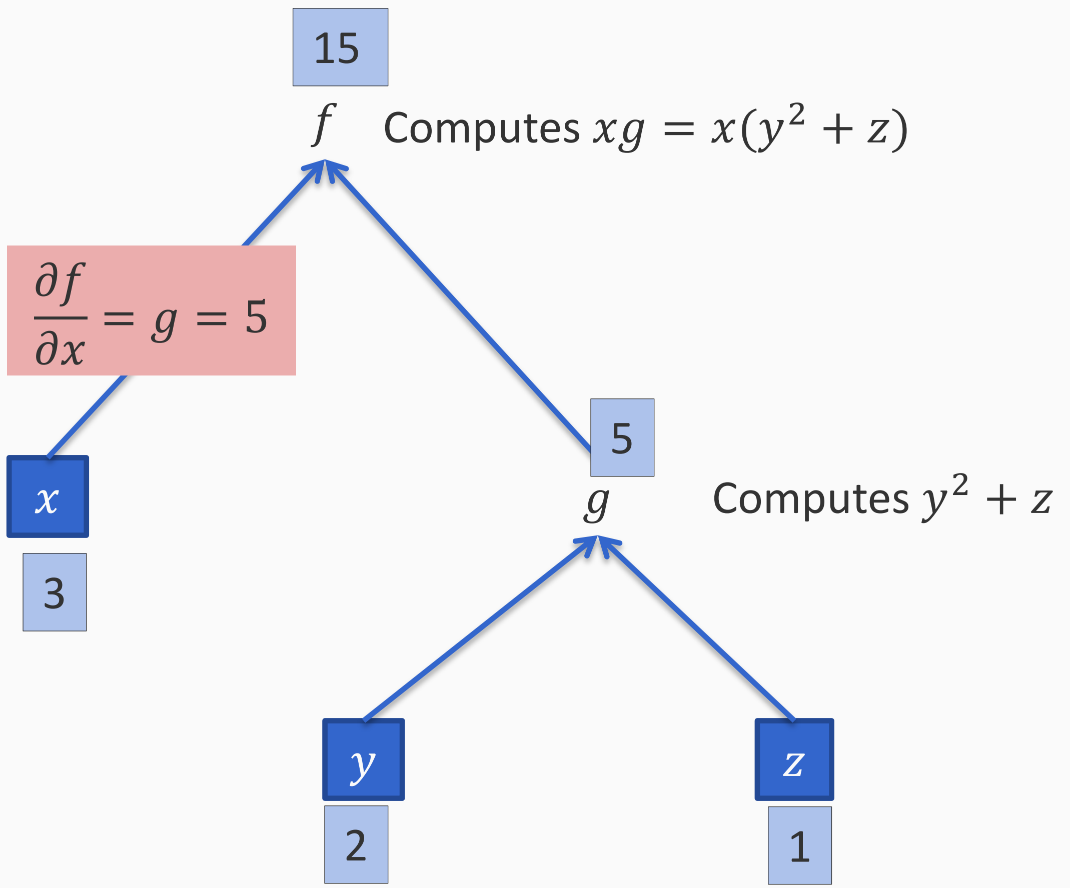
The backward pass: Computes derivatives of each intermediate node



In terms of “computation graphs”

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

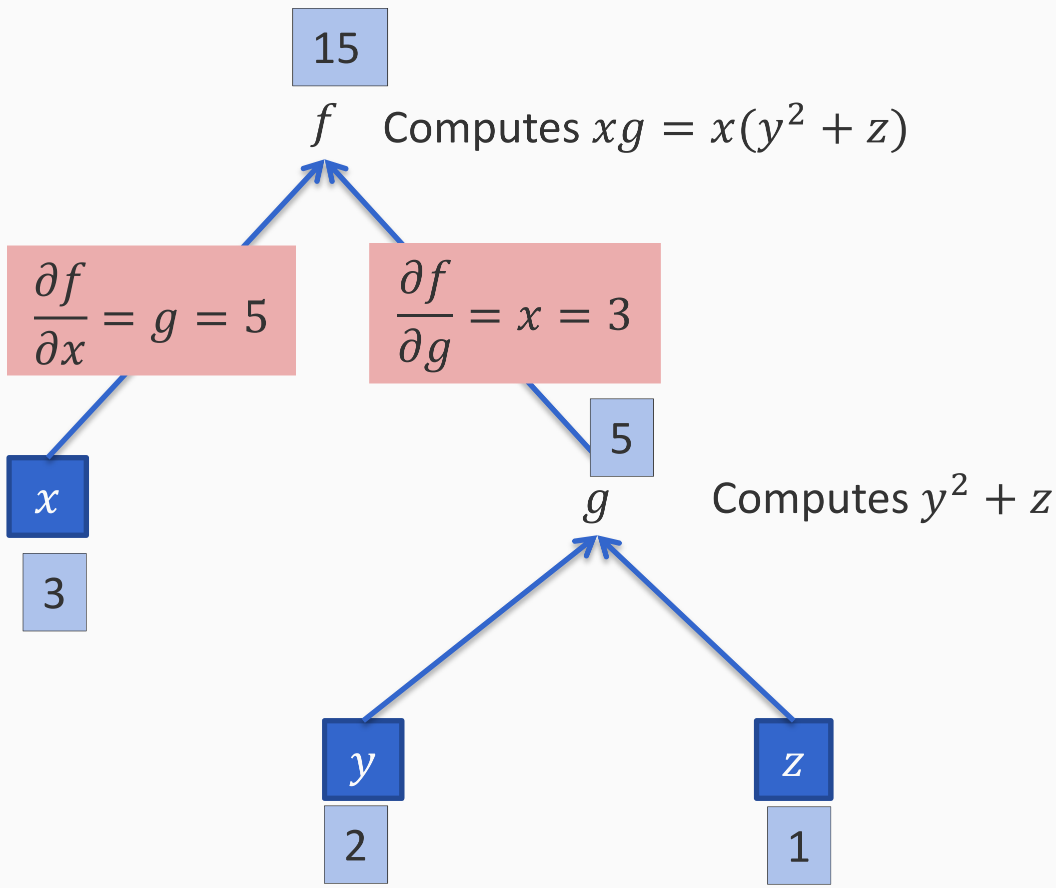
The backward pass: Computes derivatives of each intermediate node



In terms of "computation graphs"

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

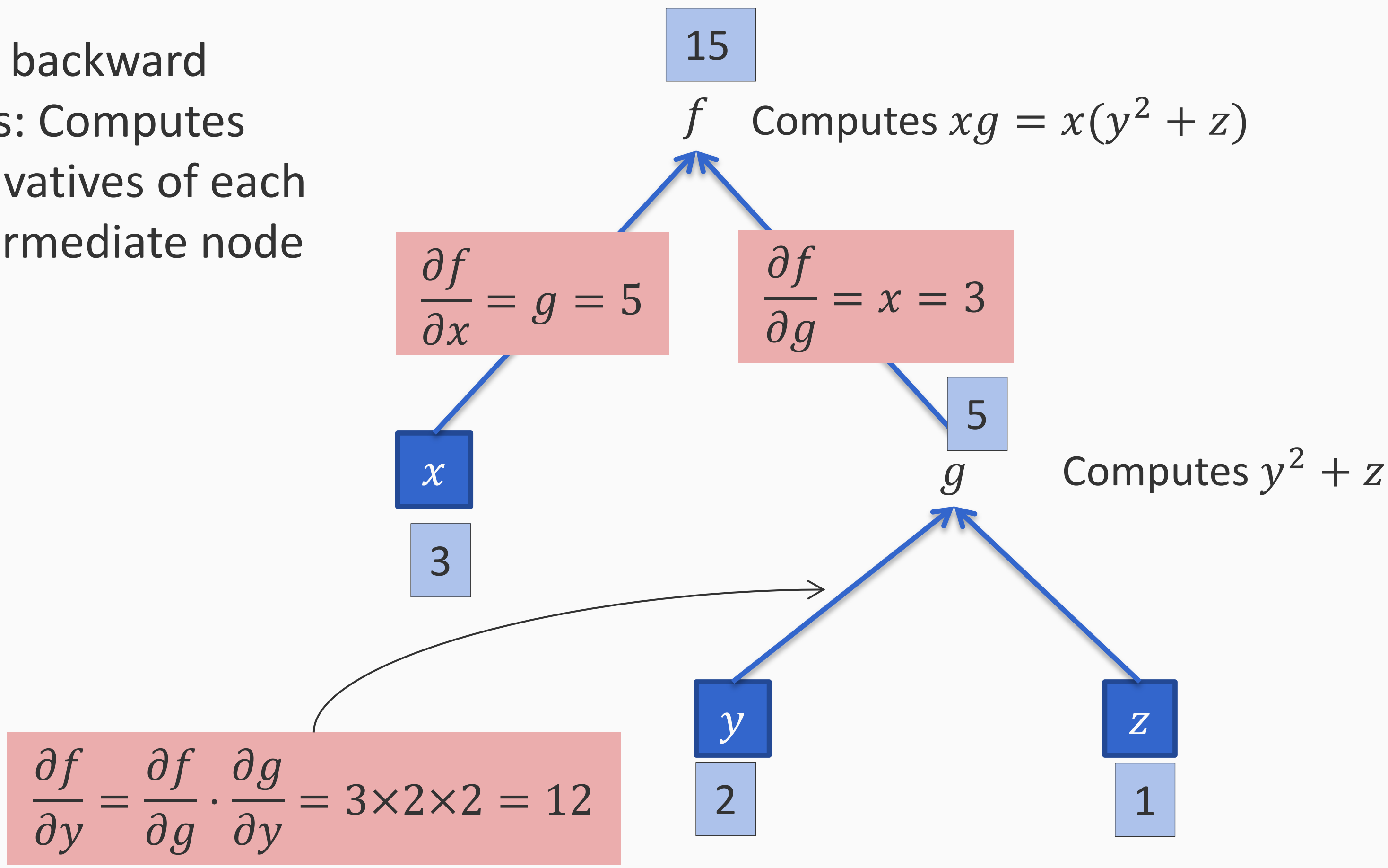
The backward pass: Computes derivatives of each intermediate node



In terms of "computation graphs"

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

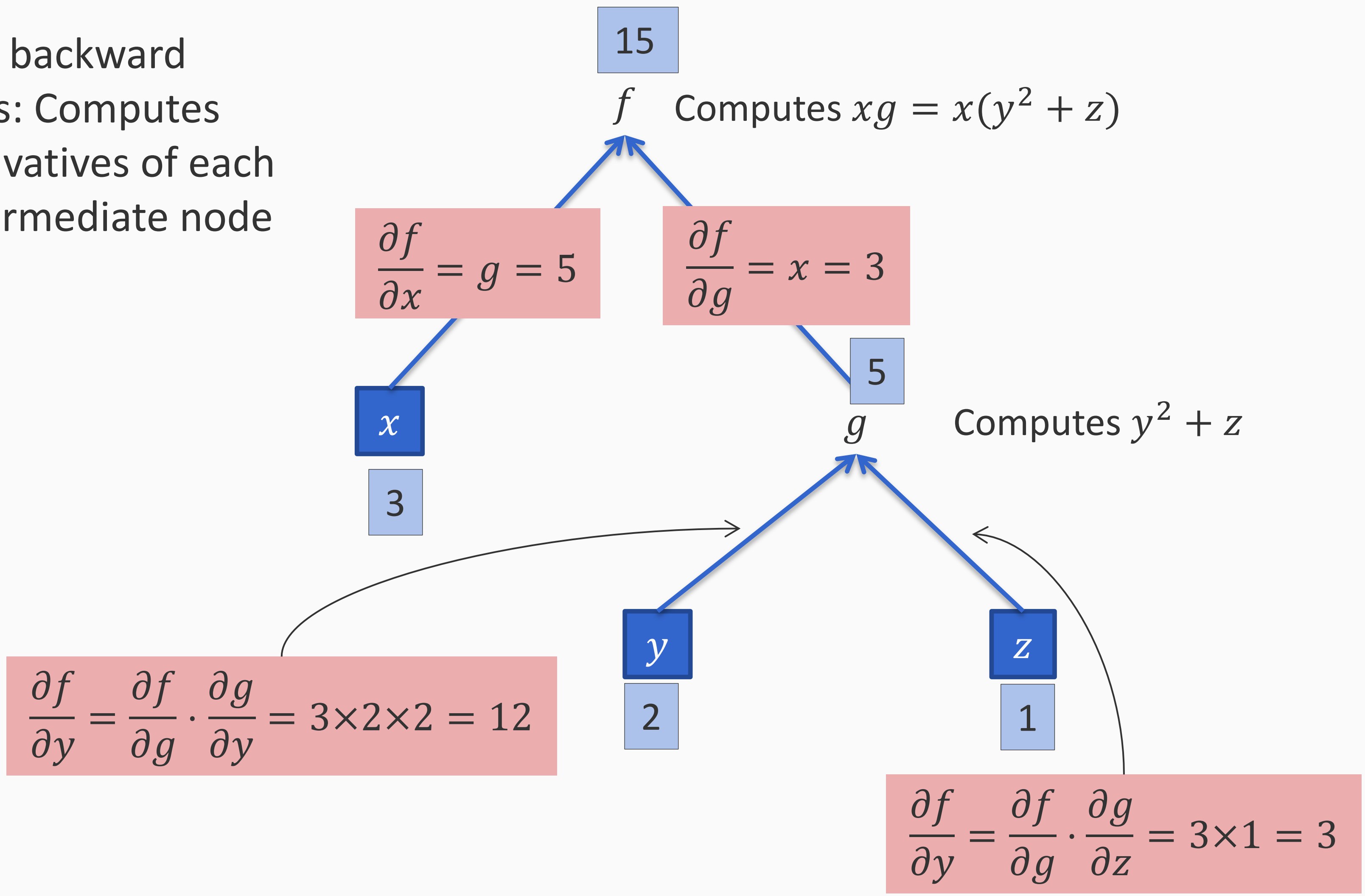
The backward pass: Computes derivatives of each intermediate node



In terms of “computation graphs”

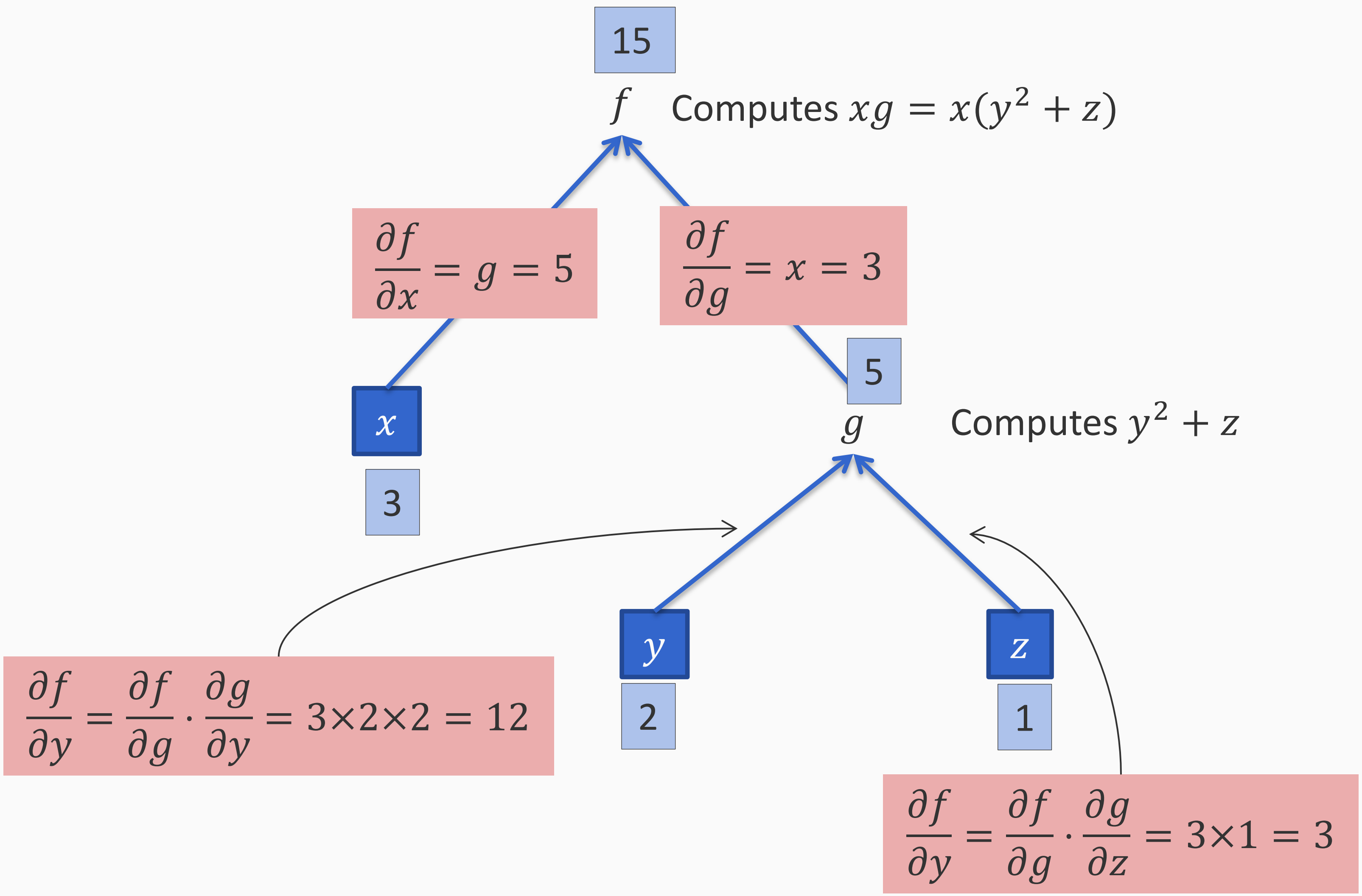
$$f(x, y, z) = x(y^2 + z)$$

The backward pass: Computes derivatives of each intermediate node



In terms of "computation graphs"

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



The abstraction

- Each node in the graph knows two things:
  1. How to compute the value of a function with respect to its inputs (forward)

  2. How to compute the partial derivative of its output with respect to each of its inputs (backward)

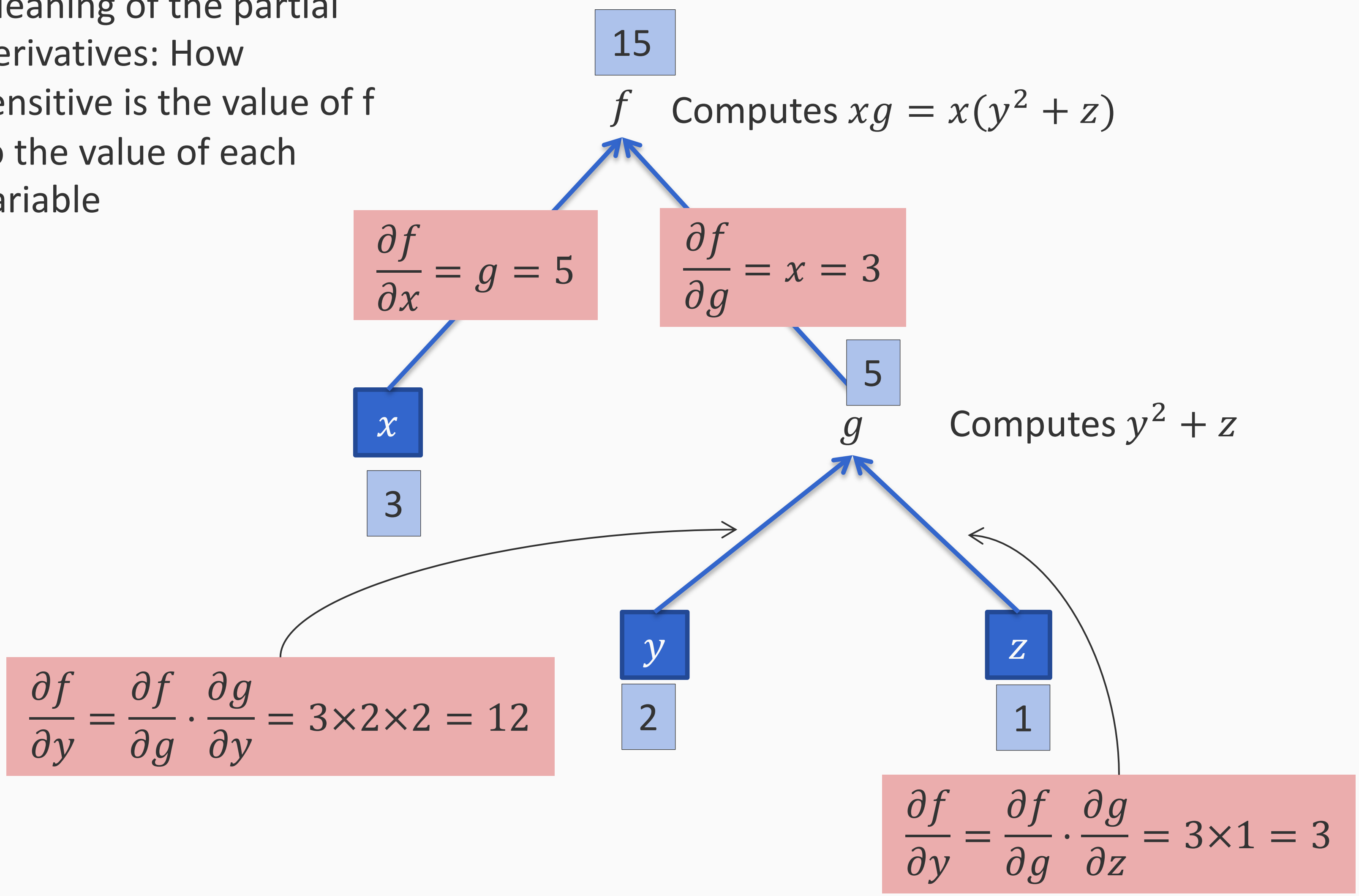
- These can be defined independently of what happens in the rest of the graph

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

In terms of “computation graphs”

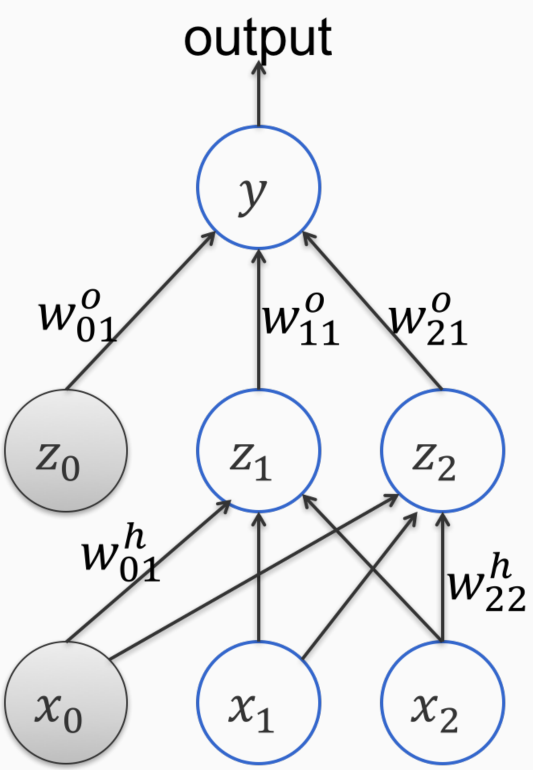
$$f(x, y, z) = x(y^2 + z)$$

Meaning of the partial derivatives: How sensitive is the value of f to the value of each variable



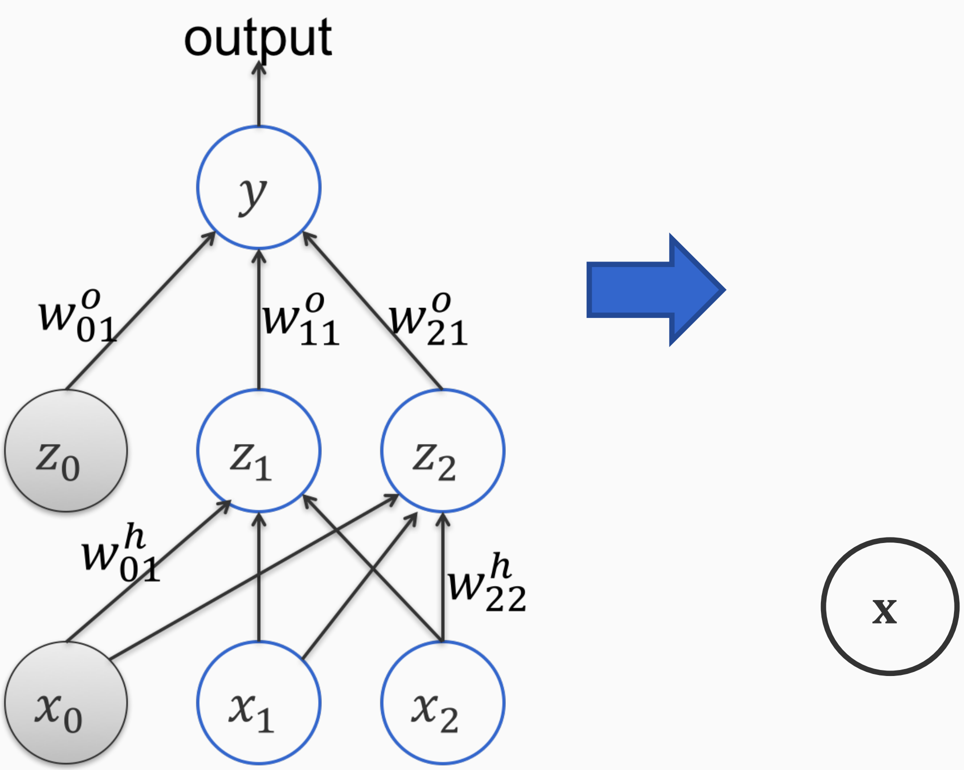
A notational convenience

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



A notational convenience

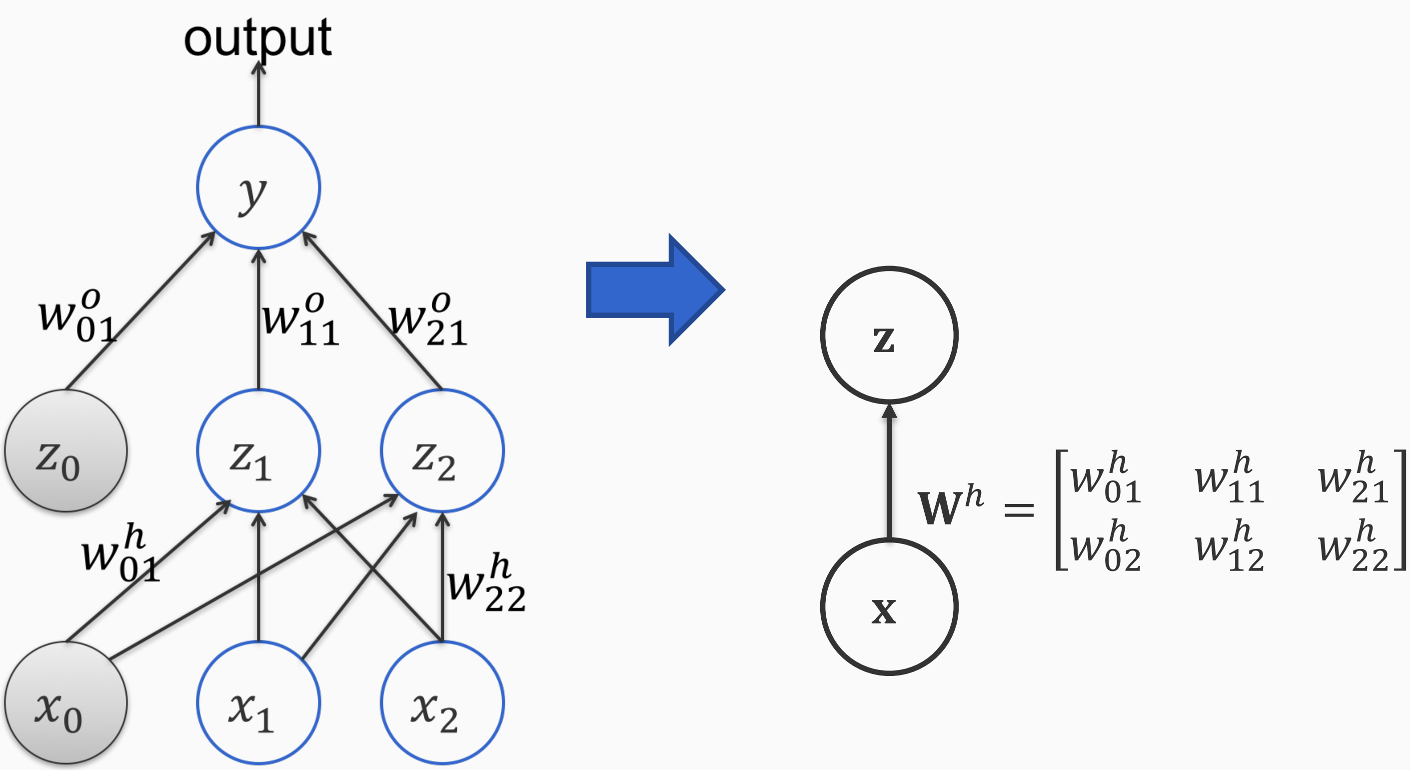
Commonly nodes in the networks represent not only single numbers (e.g. features, outputs) but also entire *vectors* (an array of numbers), *matrices* (a 2d array of numbers) or *tensors* (an n-dimensional array of numbers).



Represents $[x_0, x_1, x_2]$

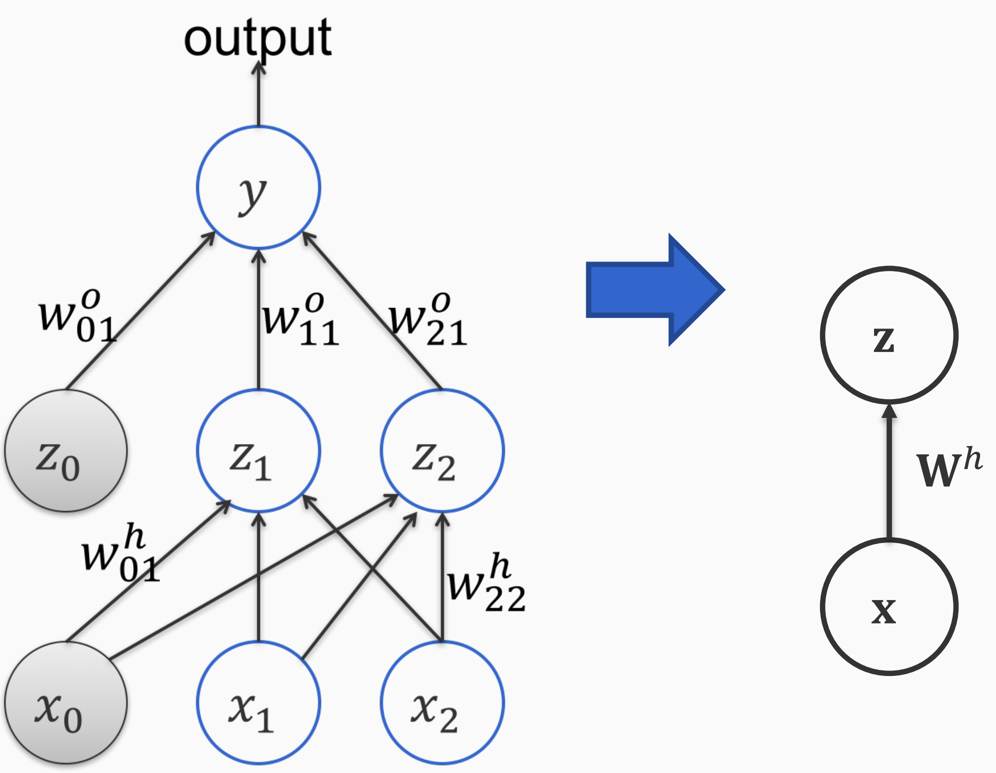
A notational convenience

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



A notational convenience

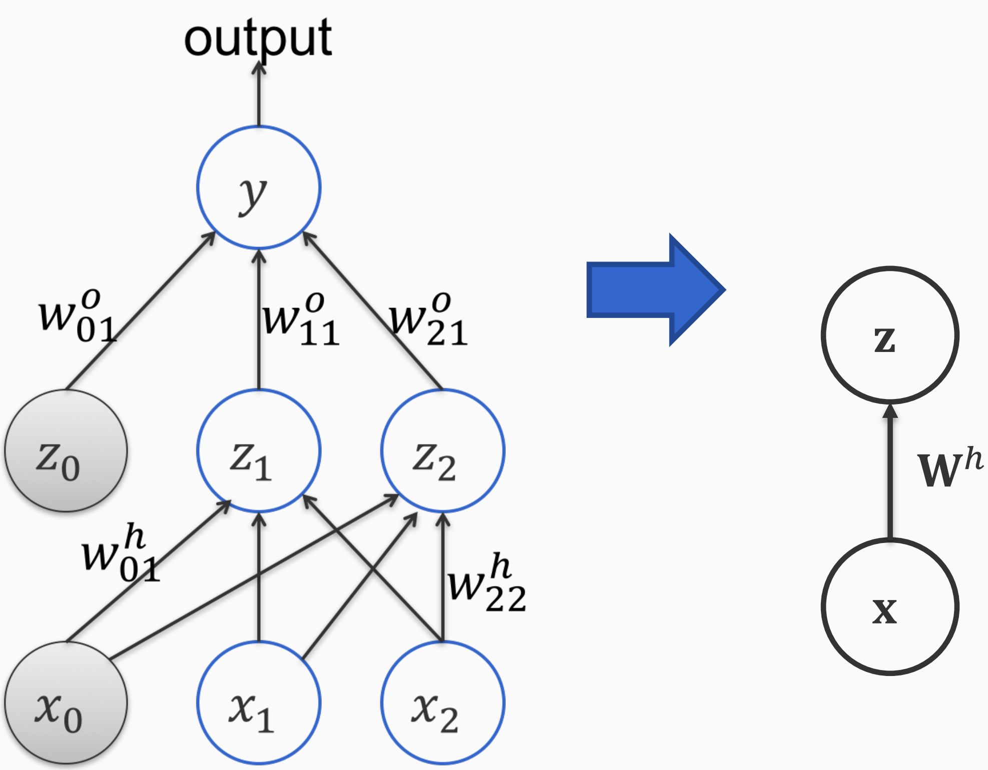
Commonly nodes in the networks represent not only single numbers (e.g. features, outputs) but also entire *vectors* (an array of numbers), *matrices* (a 2d array of numbers) or *tensors* (an n-dimensional array of numbers).



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

A notational convenience

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

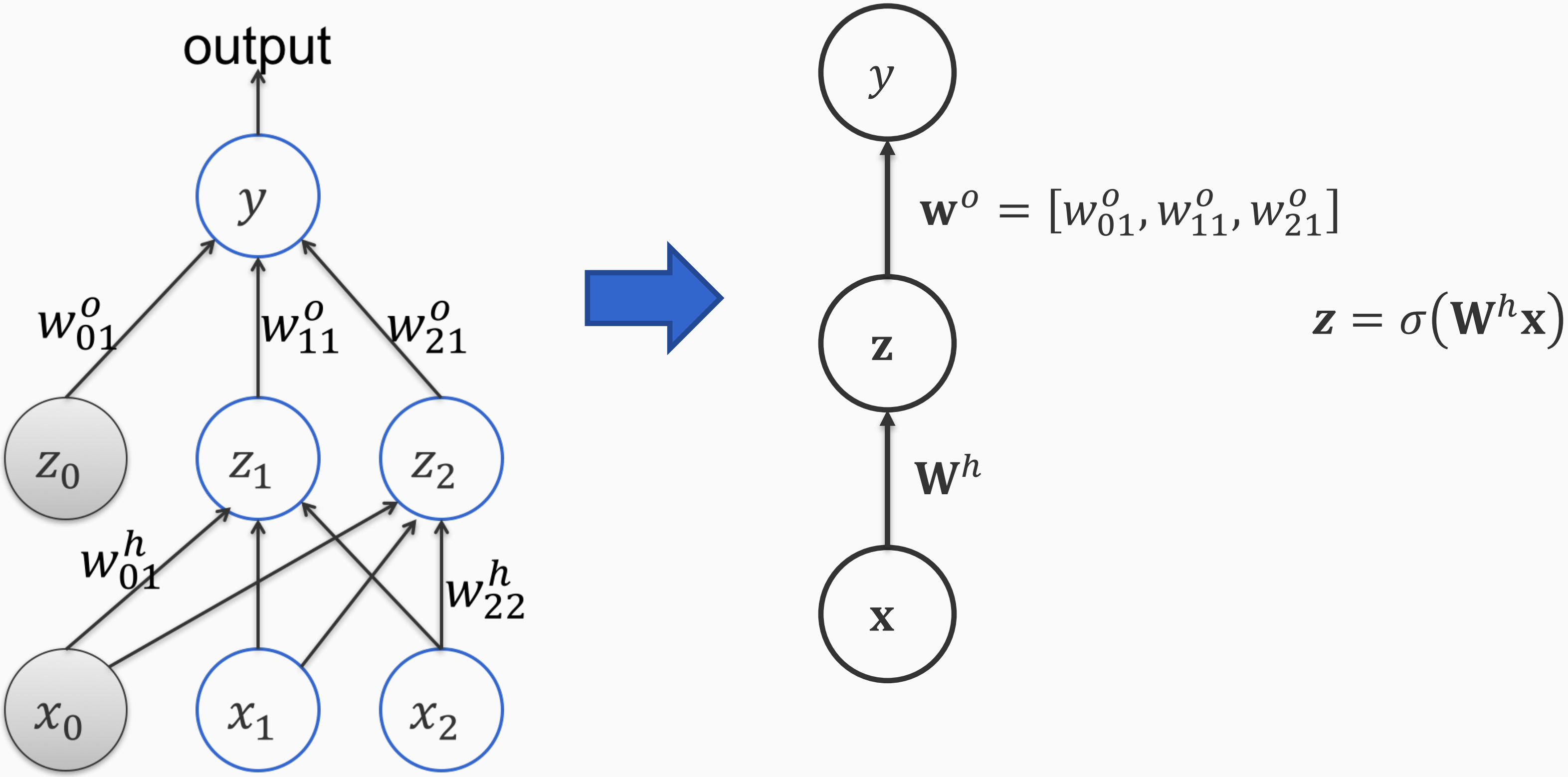


$$\mathbf{z} = \sigma(\mathbf{W}^h\mathbf{x})$$

Each element of $\mathbf{z}$ is z_i, and is generated by the sigmoid activation to each element of $\mathbf{W}^h\mathbf{x}$.

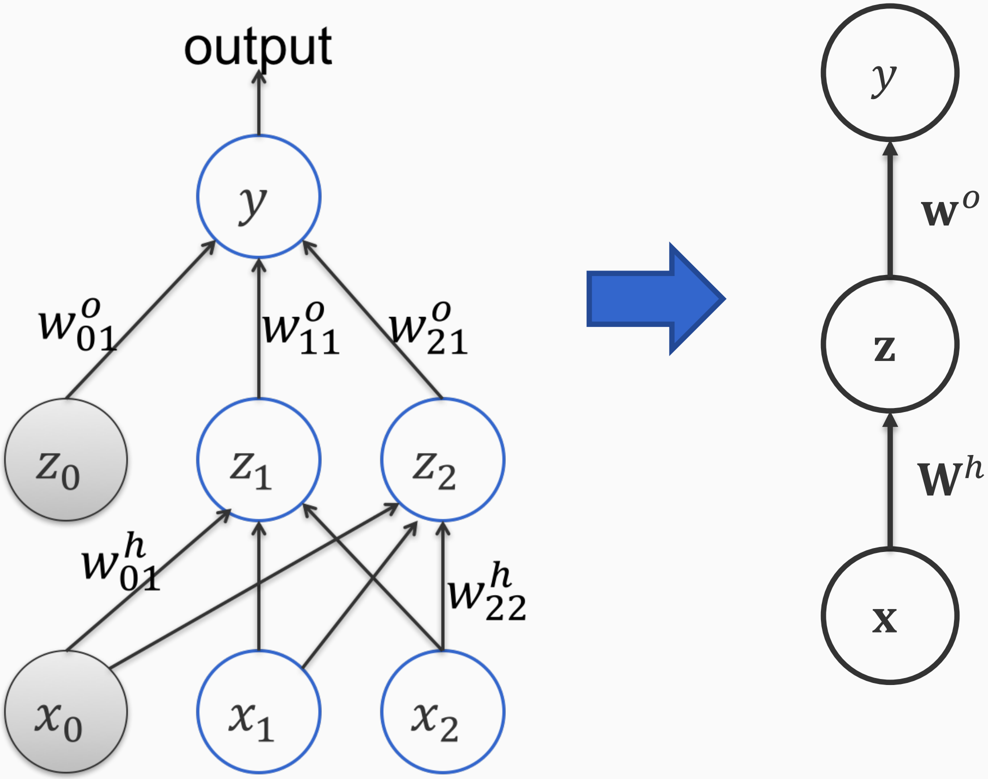
A notational convenience

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



A notational convenience

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

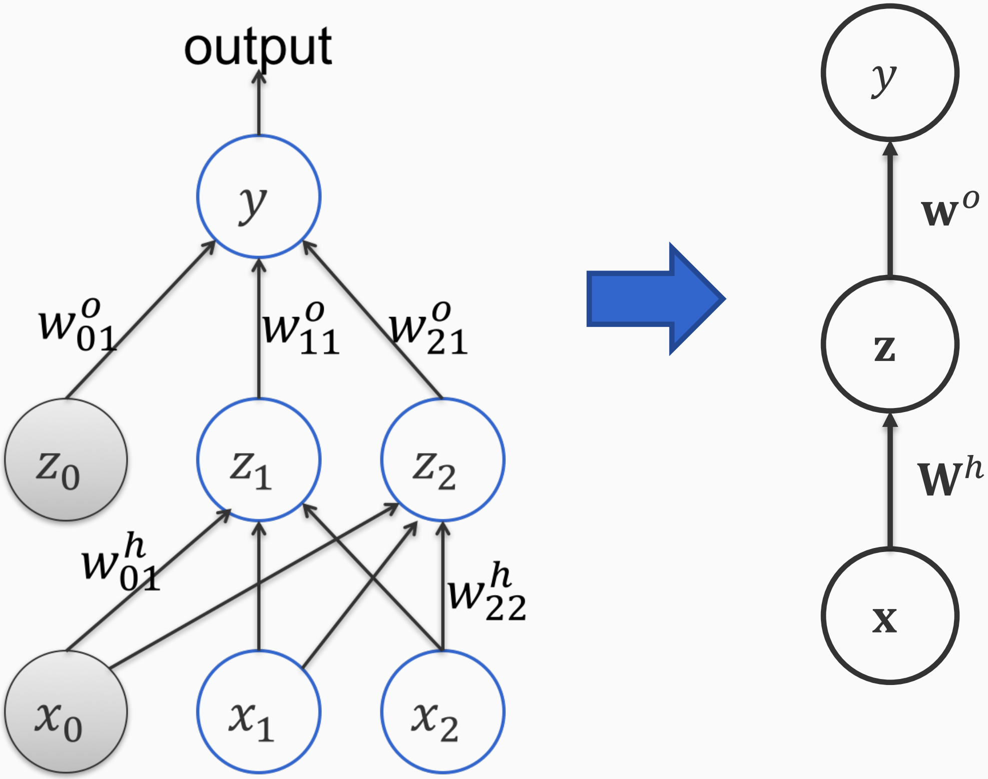


$$y = \mathbf{w}^o \mathbf{z}$$

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

A notational convenience

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



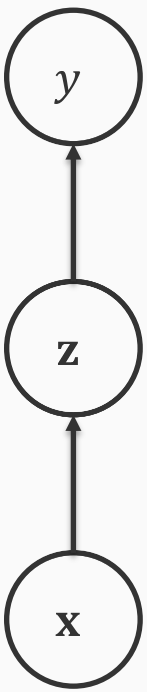
$$y = \mathbf{w}^{o}\mathbf{z}$$

No activation because the output is defined to be linear

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

Reminder: Chain rule for derivatives

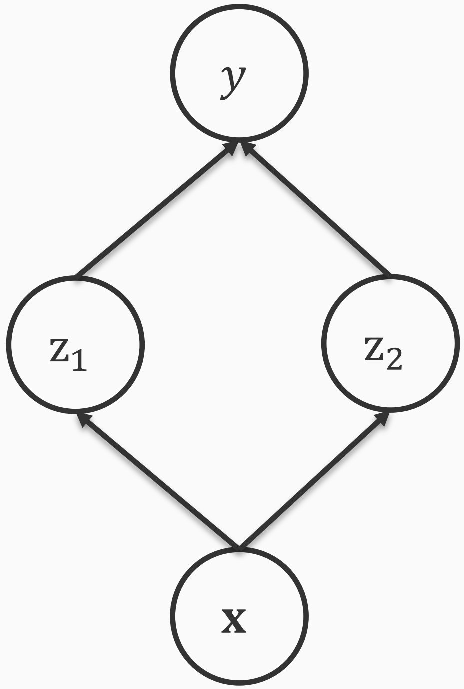
- If y is a function of $\mathbf{z}$ and $\mathbf{z}$ is a function of $\mathbf{x}$
  - Then y is a function of $\mathbf{x}$, as well
- Question: how to find $\frac{\partial y}{\partial \mathbf{x}}$



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

Reminder: Chain rule for derivatives

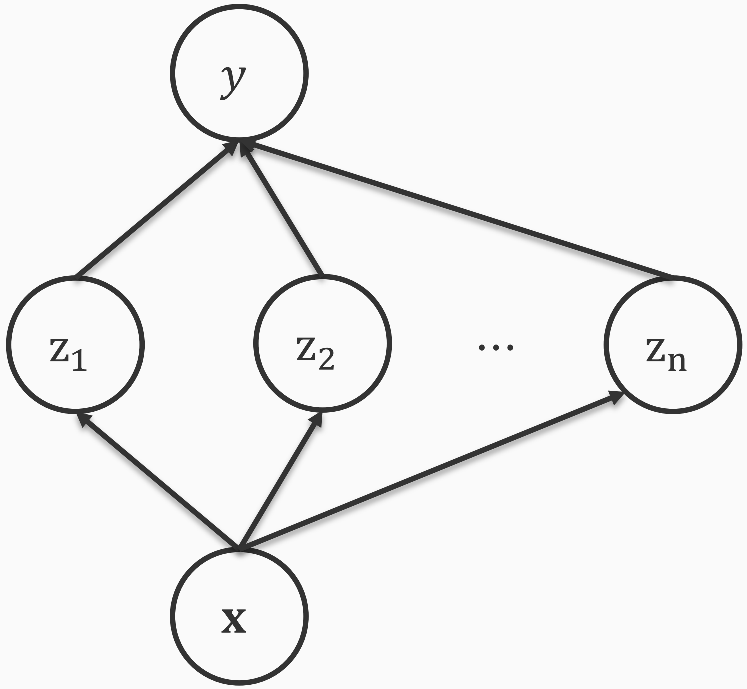
- If $y =$ a function of z_1 + a function of z_2, and the z_i's are functions of x
  - Then y is a function of x, as well
- Question: how to find $\frac{\partial y}{\partial x}$



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1} \cdot \frac{\partial z_1}{\partial x} + \frac{\partial y}{\partial z_2} \cdot \frac{\partial 2}{\partial x}$$

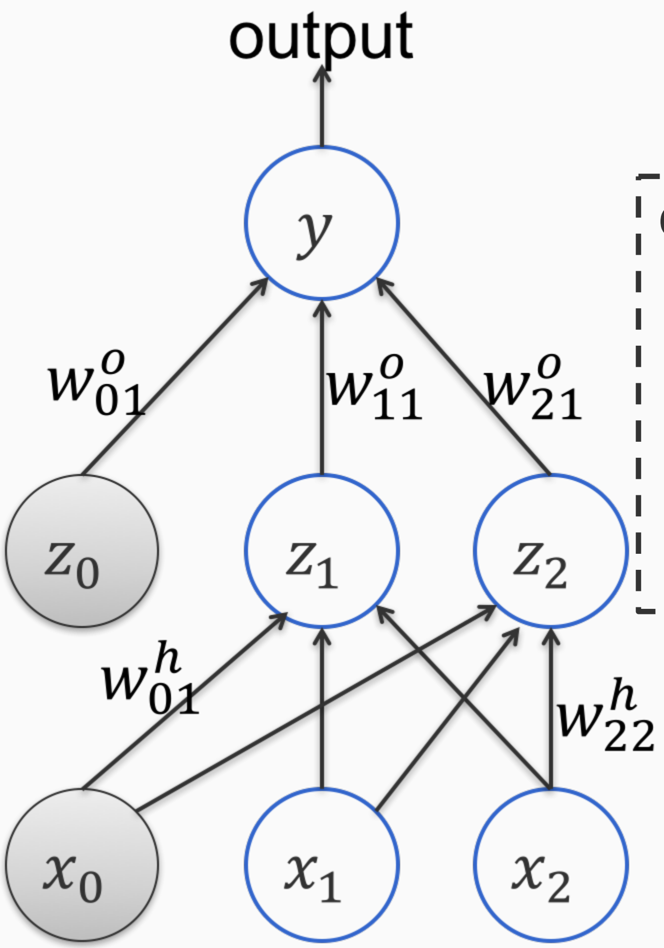
Reminder: Chain rule for derivatives

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



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

Backpropagation



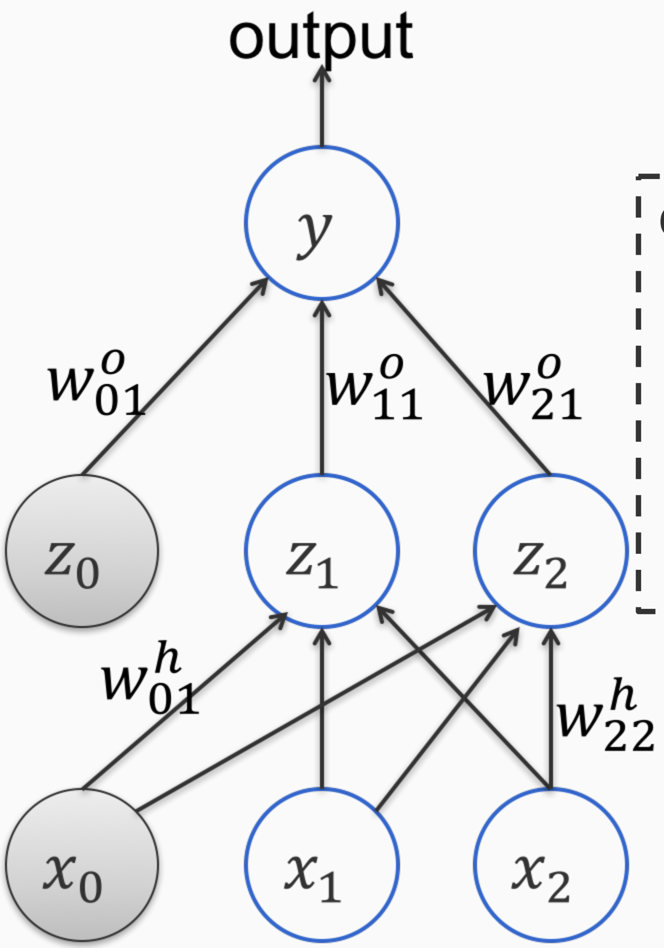
$$L = \frac{1}{2}(y - y^*)^2$$

output $\mathrm{y} = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$

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

$$\mathrm{z}_1 = \sigma(w^h_{01} + w^h_{11} x_1 + w^h_{21} x_2)$$

Backpropagation



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

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

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

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

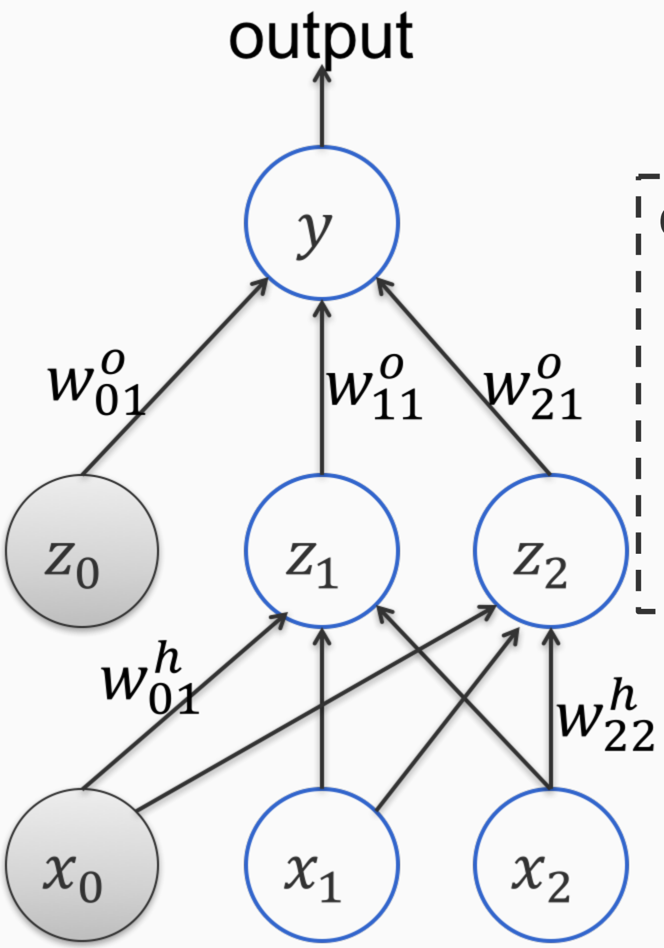
We want to compute

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

Important: L is a differentiable function of all the weights

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 $y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$

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

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

We want to compute

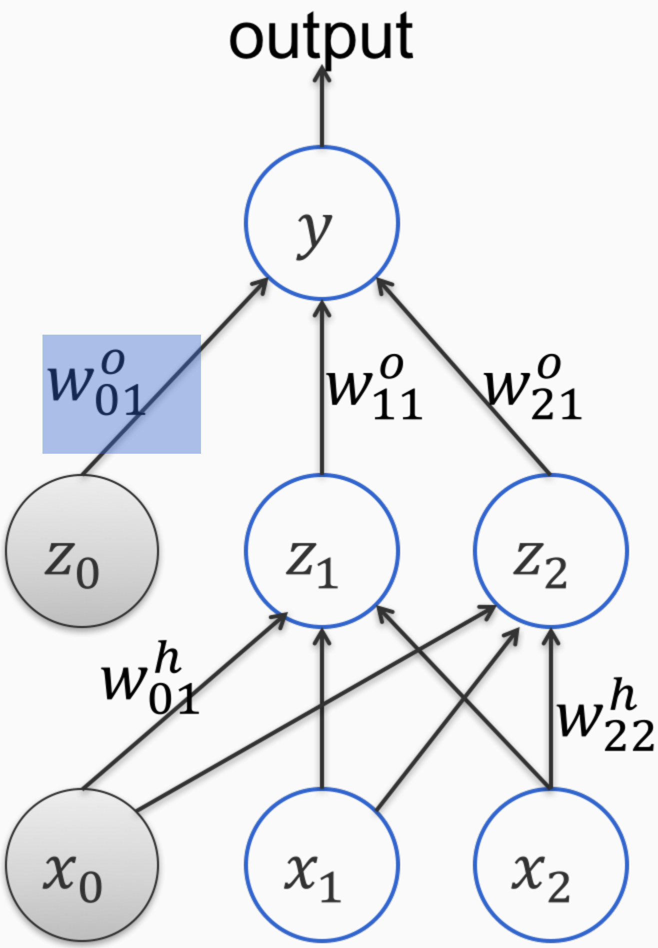
$\frac{\partial L}{\partial w^o_{ij}}$ and $\frac{\partial L}{\partial w^h_{ij}}$

Important: L is a differentiable function of all the weights

Output layer

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

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

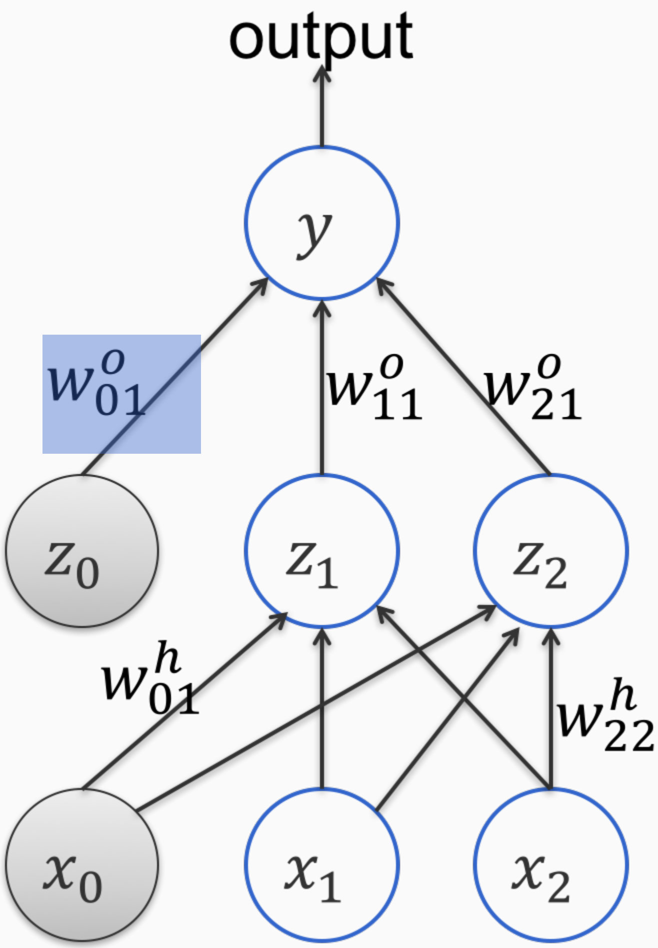


$$\frac{\partial L}{\partial w_{01}^o}$$

Output layer

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

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

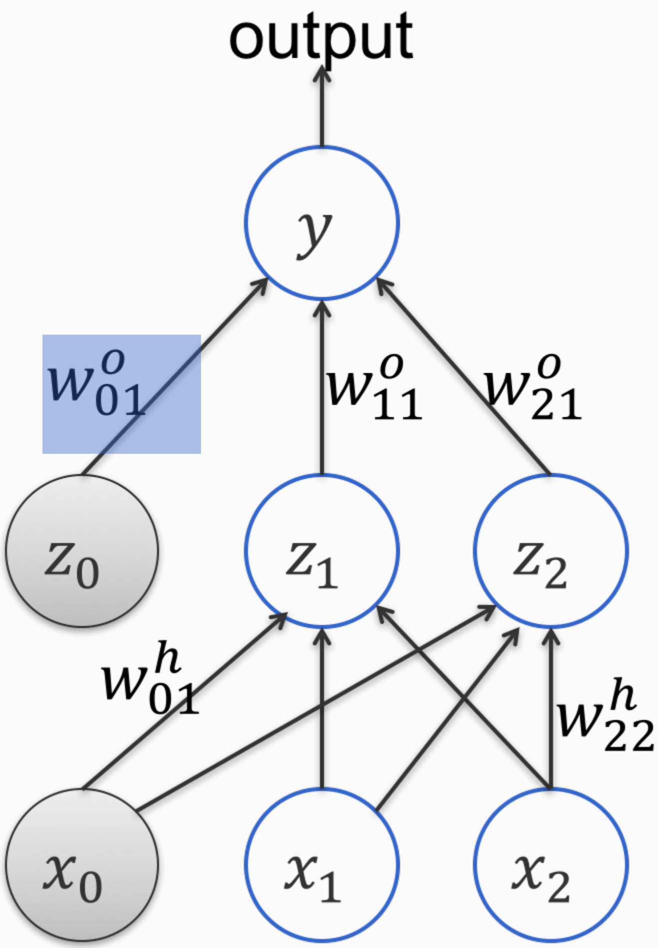


$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{01}^o}$$

Output layer

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

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



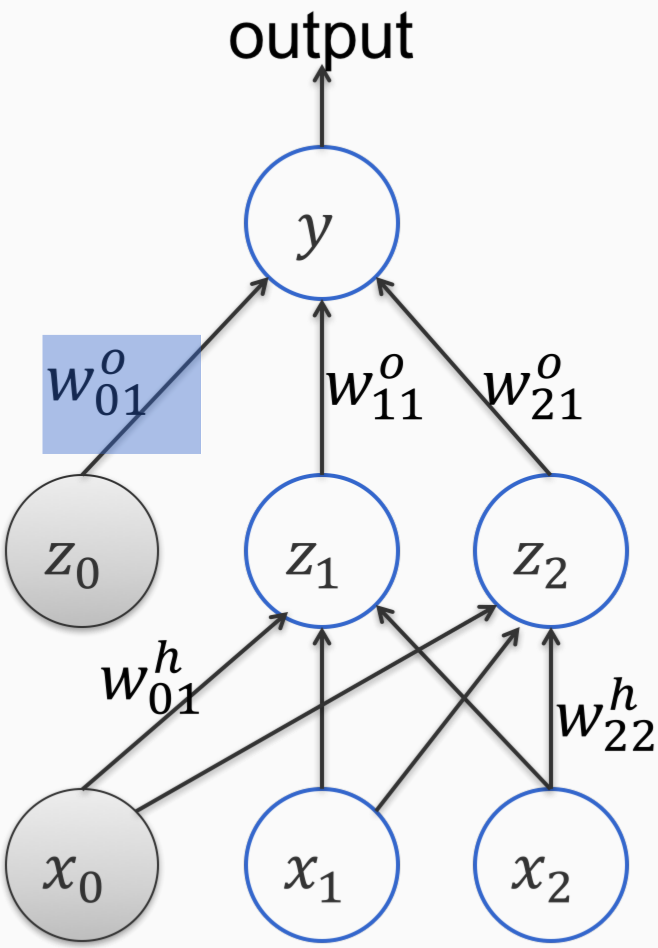
$$\frac{\partial L}{\partial w_{01}^o} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{01}^o}$$

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

Output layer

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

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



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

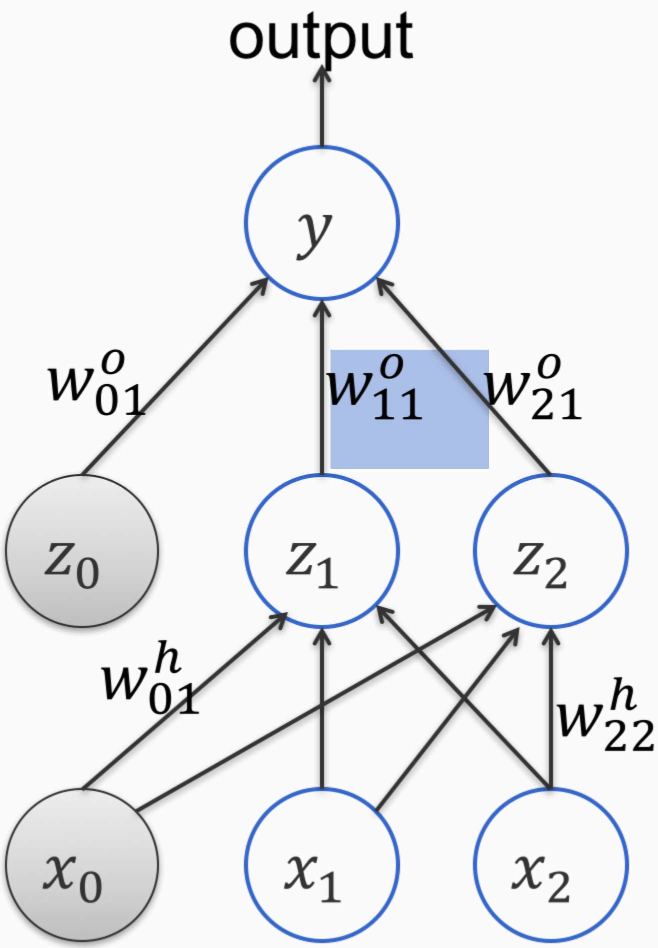
$$\frac{\partial L}{\partial y} = y - y^*$$

$$\frac{\partial y}{\partial w_{01}^o} = 1$$

Output layer

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

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

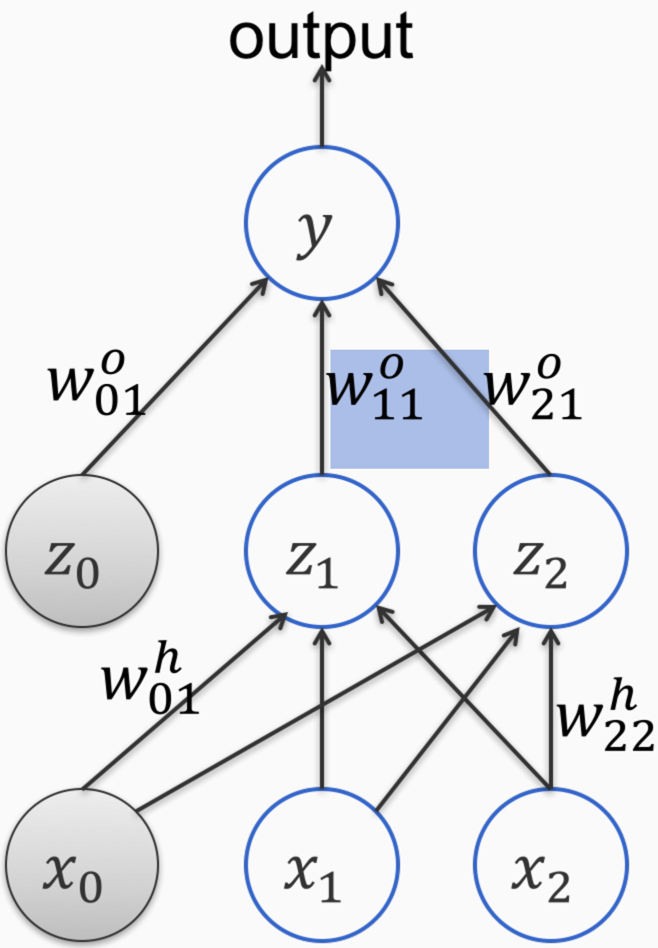


$$\frac{\partial L}{\partial w_{11}^o}$$

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

Output layer

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

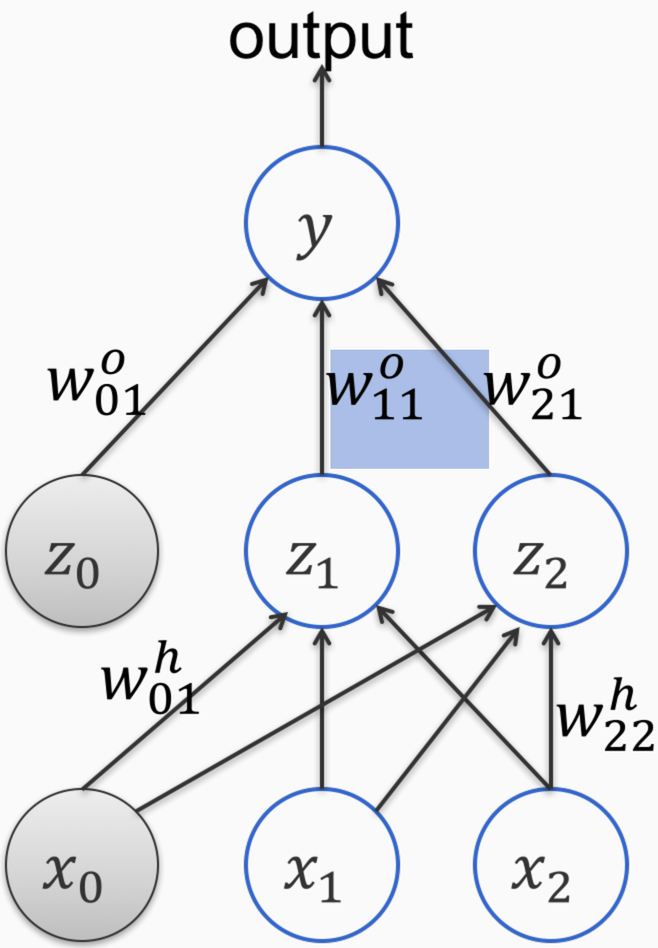


$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^o}$$

Output layer

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

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



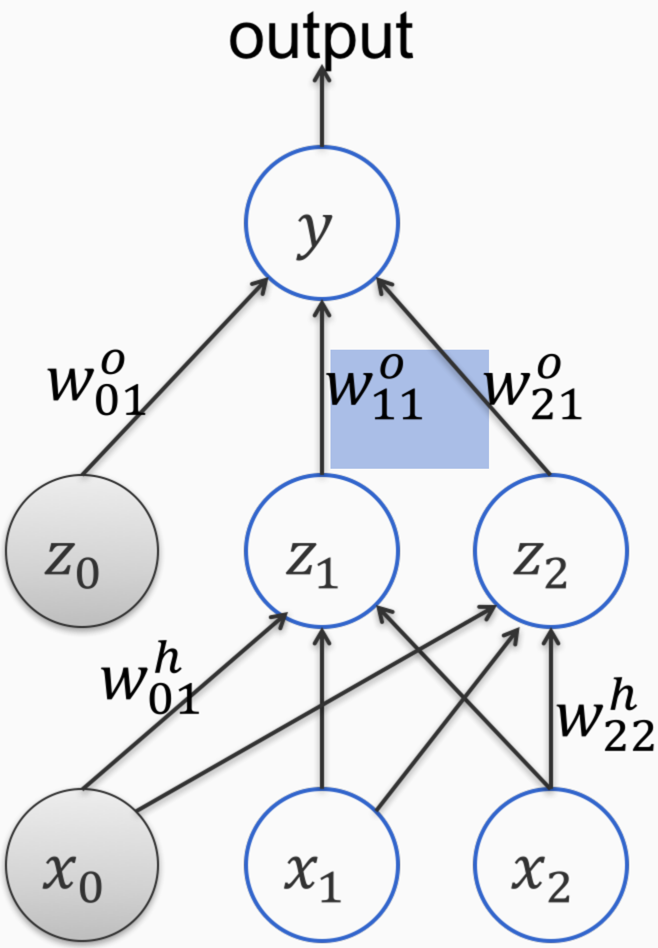
$$\frac{\partial L}{\partial w_{11}^o} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{11}^o}$$

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

Output layer

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

output $\quad y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$



$$\frac{\partial L}{\partial w^o_{11}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w^o_{11}}$$

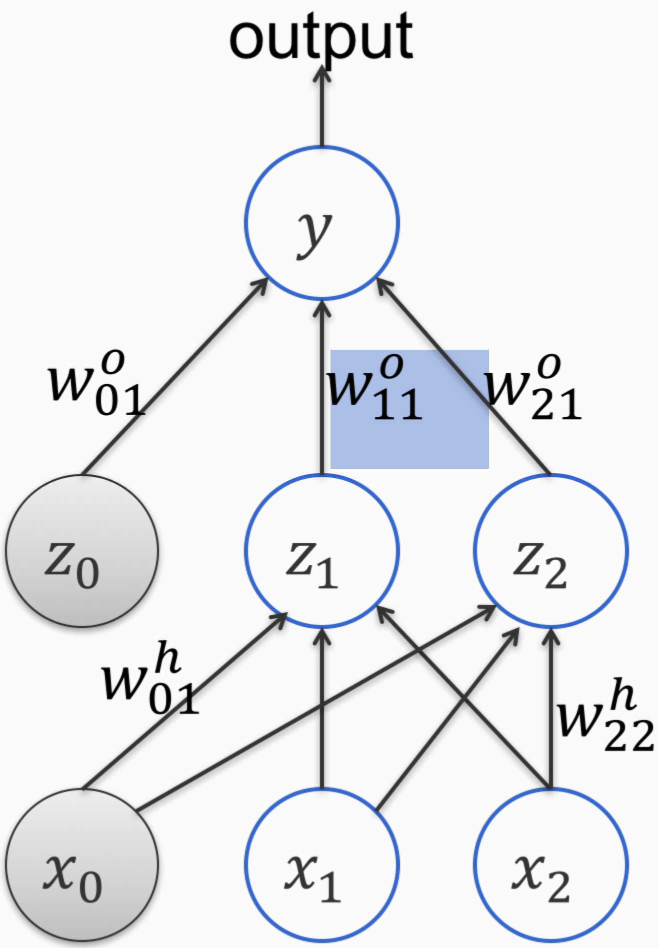
$$\frac{\partial L}{\partial y} = y - y^*$$

$$\frac{\partial y}{\partial w^o_{01}} = z_1$$

Output layer

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

output $\quad y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$



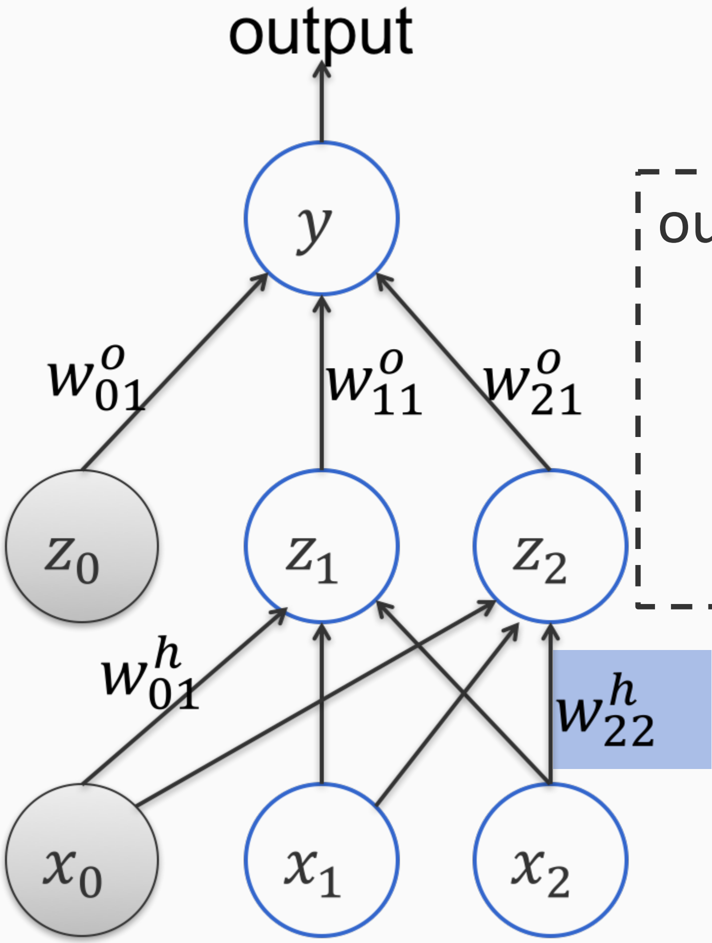
$$\frac{\partial L}{\partial w^o_{11}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w^o_{11}}$$

$$\frac{\partial L}{\partial y} = y - y^* \qquad \frac{\partial y}{\partial w^o_{01}} = z_1$$

We have already computed this partial derivative for the previous case

Cache to speed up!

Hidden layer derivatives



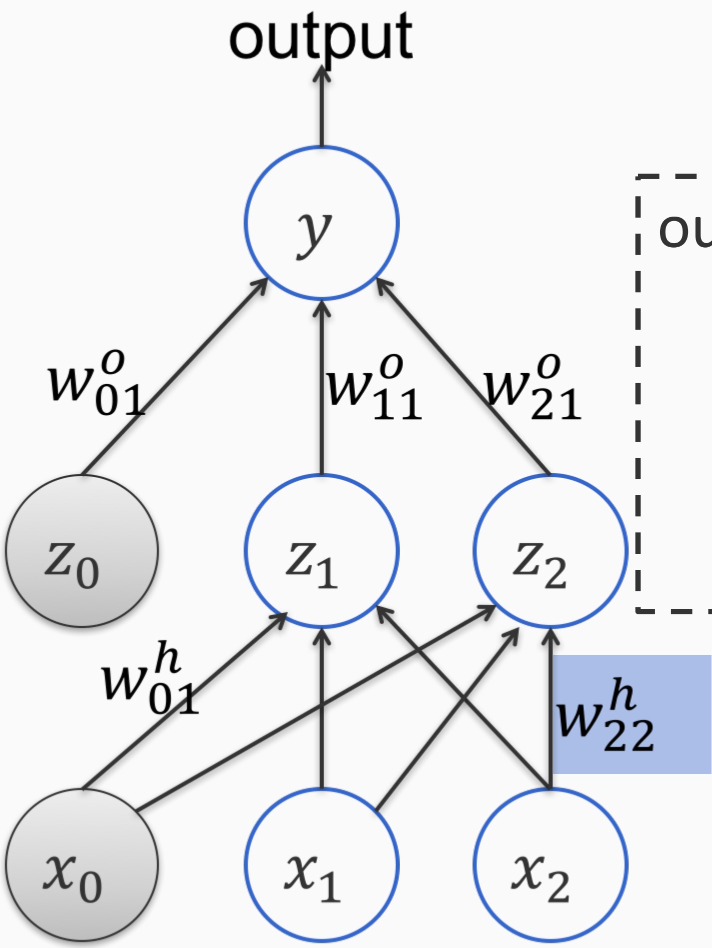
$$L = \frac{1}{2}(y - y^*)^2$$

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

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

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

Hidden layer derivatives



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

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

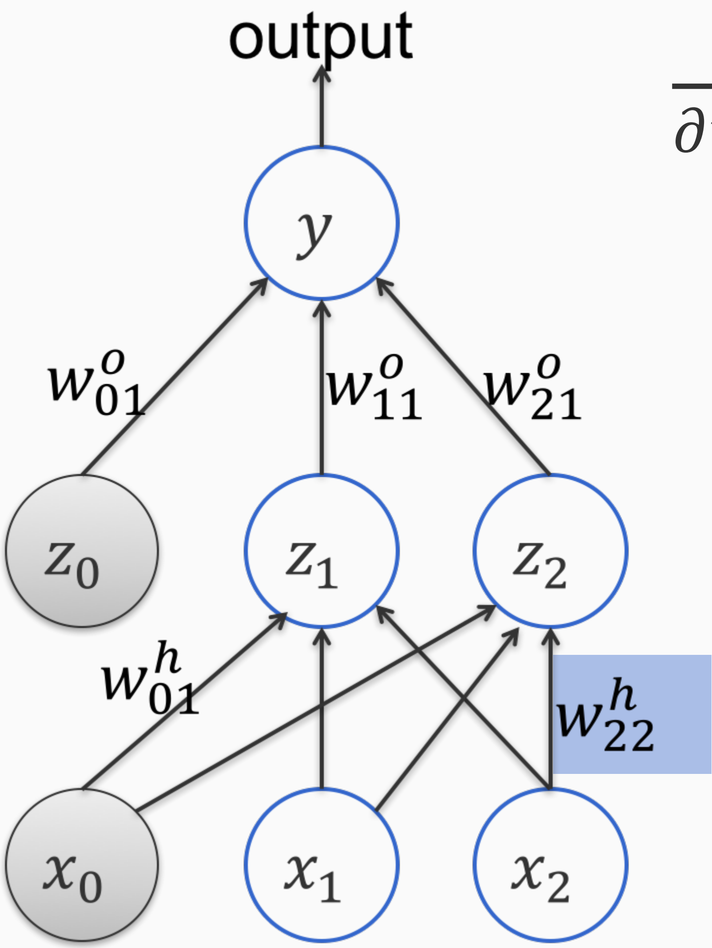
$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

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

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

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

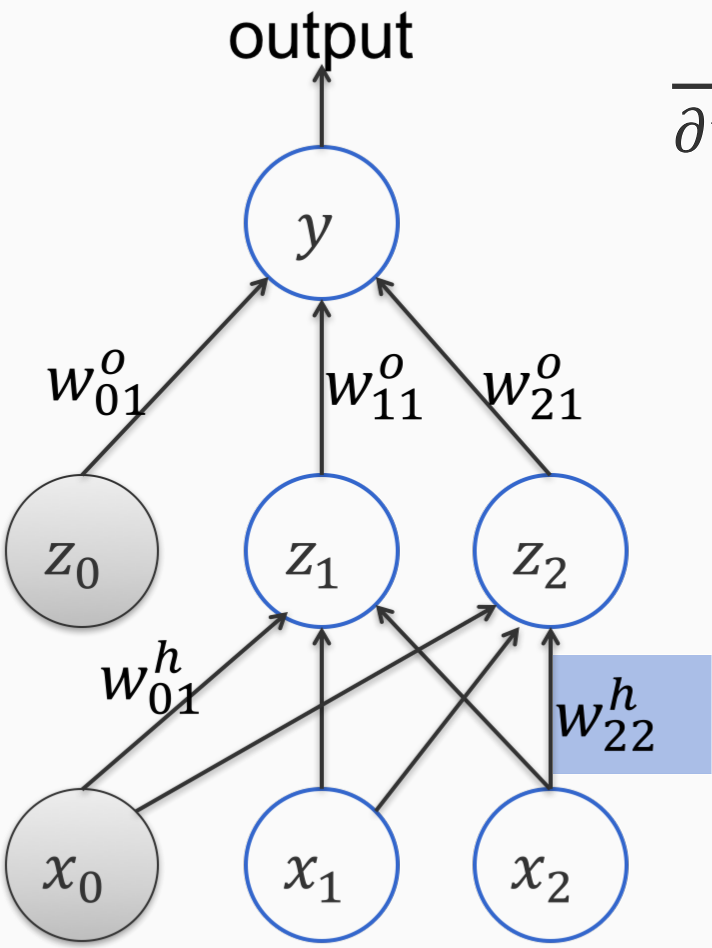
Hidden layer derivatives



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

Hidden layer

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

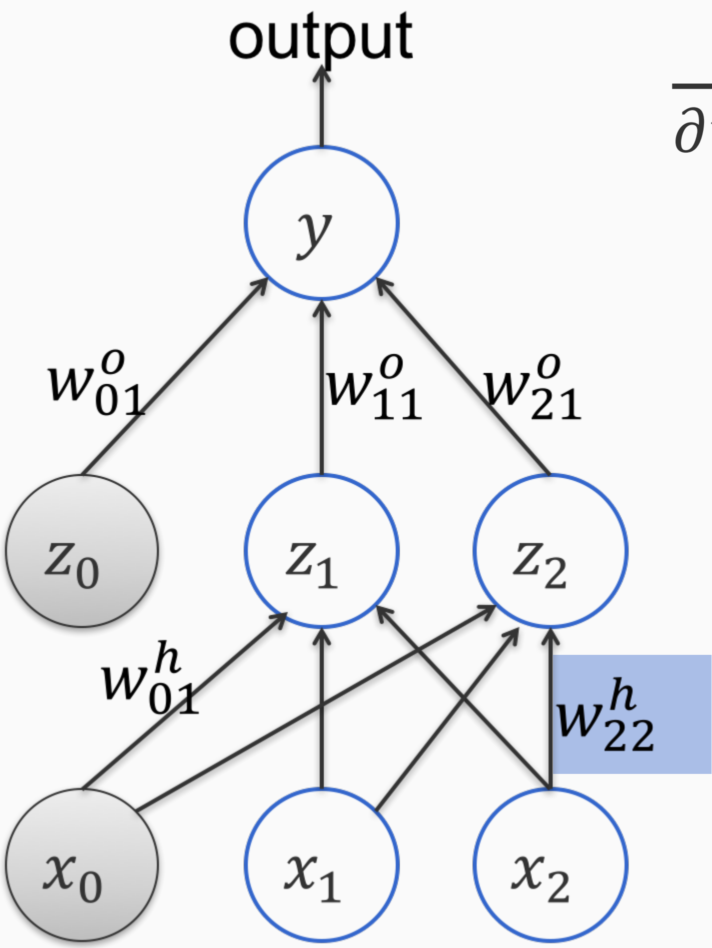


$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{22}^h}$$

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

Hidden layer

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



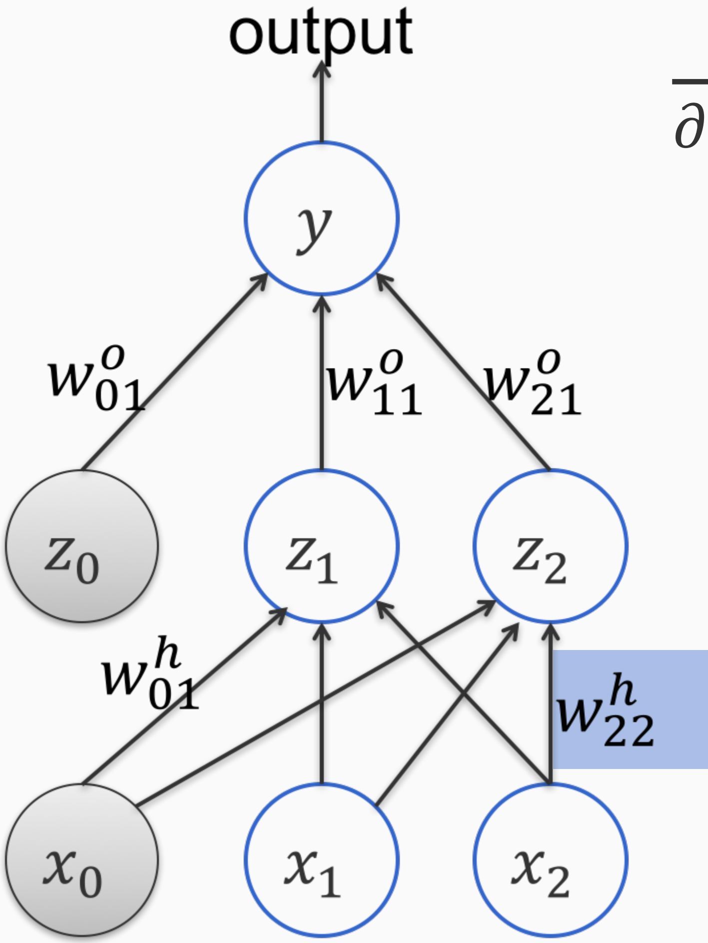
$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

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

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

Hidden layer

$$y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$$

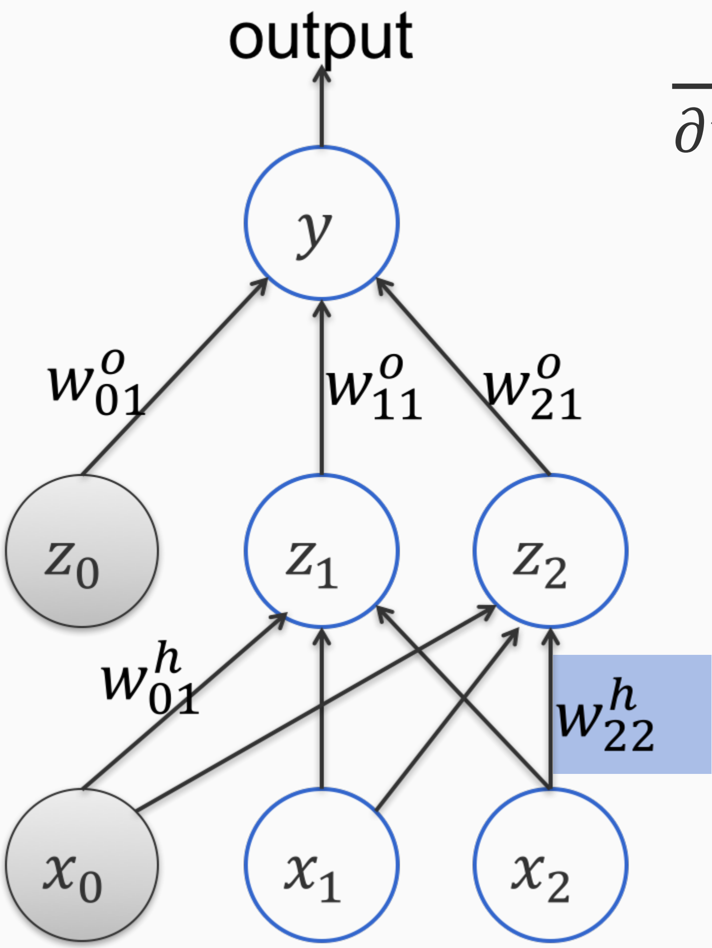


$$\begin{aligned}
\frac{\partial L}{\partial w^h_{22}} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial w^h_{22}} \\
&= \frac{\partial L}{\partial y}\frac{\partial}{\partial w^h_{22}}\left(w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2\right) \\
&= \frac{\partial L}{\partial y}\left(w^o_{11}\underbrace{\frac{\partial}{\partial w^h_{22}} z_1}_{0} + w^o_{21}\frac{\partial}{\partial w^h_{22}} z_2\right)
\end{aligned}$$

z_1 is not a function of w^h_{22}

Hidden layer

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



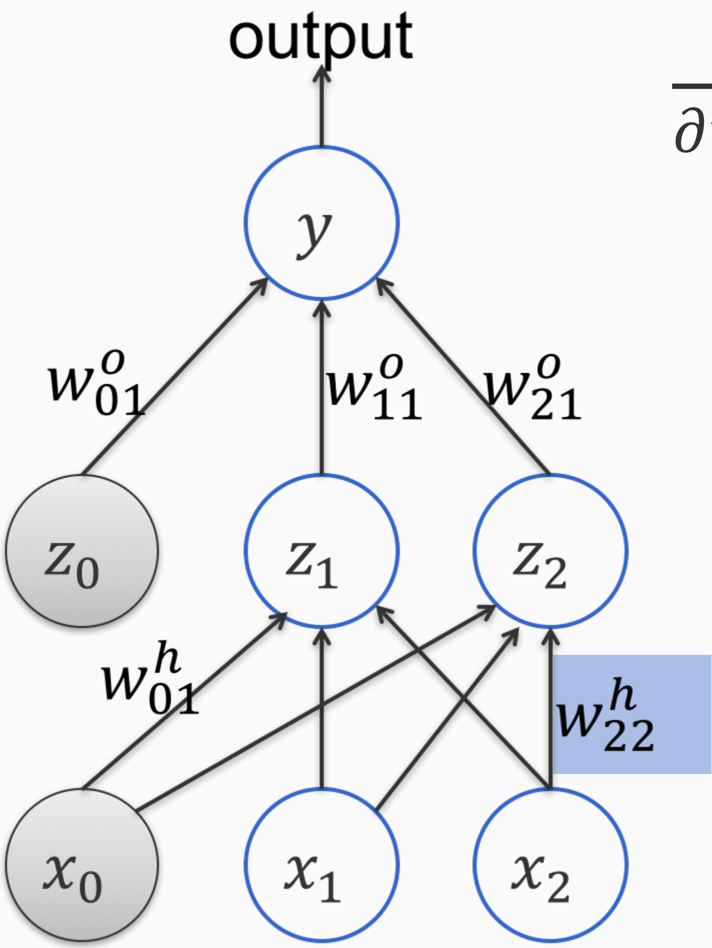
$$\frac{\partial L}{\partial w_{22}^{h}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_{22}^{h}}$$

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

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

Hidden layer

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



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

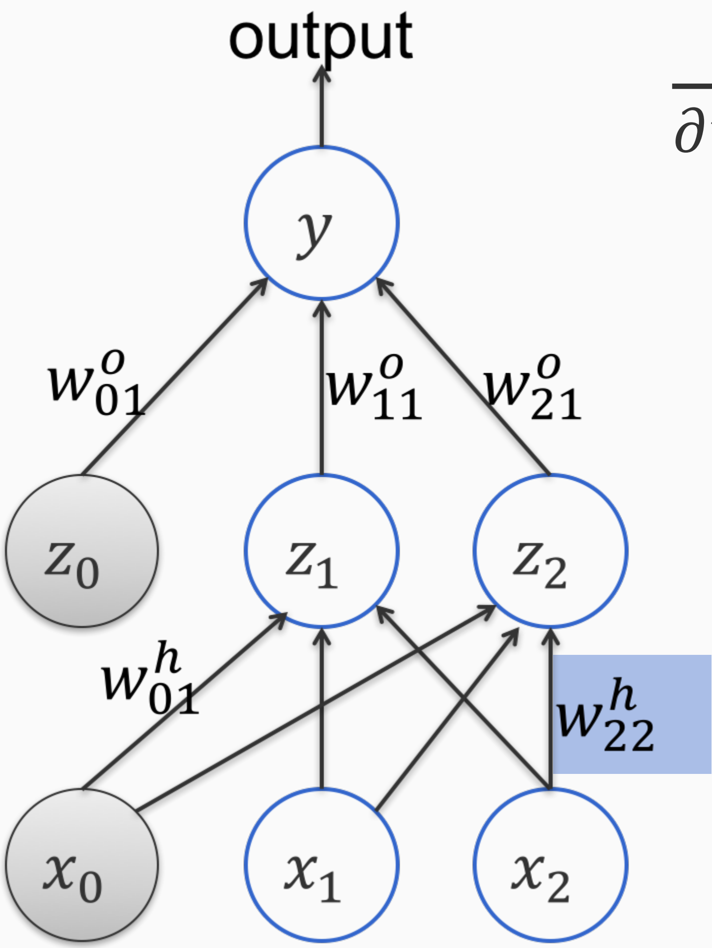
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

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

Hidden layer

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

Call this s



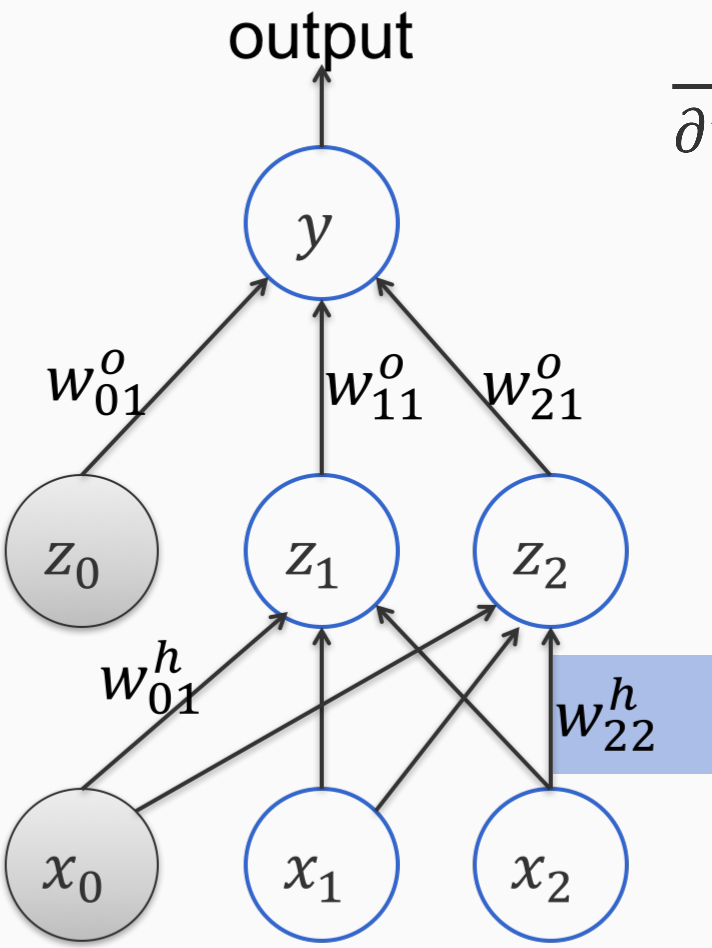
$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_{22}^h}$$

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

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

Hidden layer

$$z_2 = \sigma(\underbrace{w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2}_{\text{Call this s}})$$



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

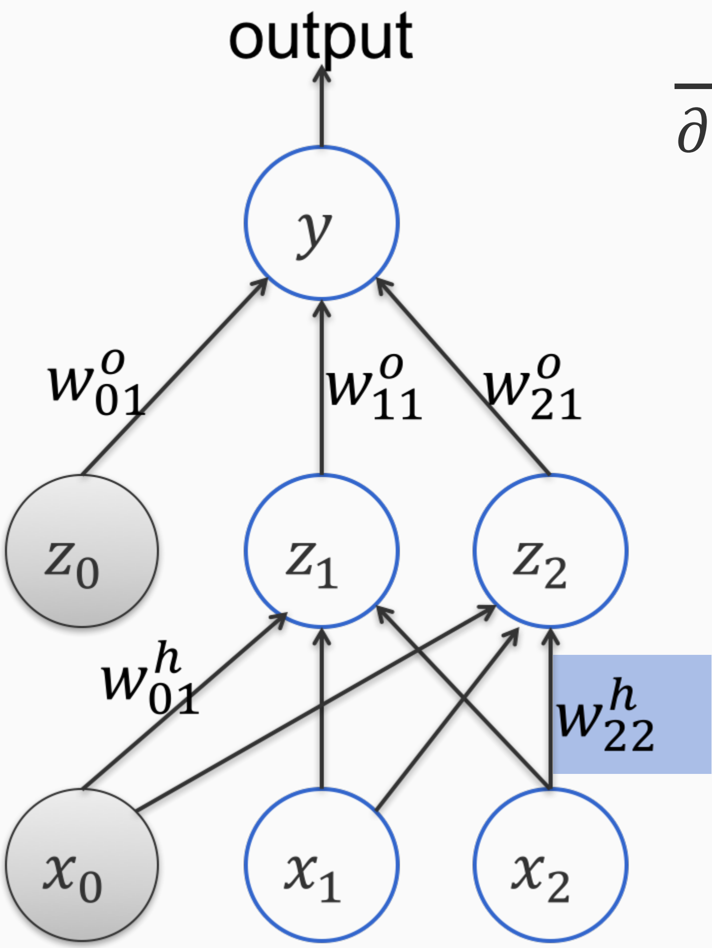
$$= \frac{\partial L}{\partial y}\frac{\partial}{\partial w_{22}^h}(w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2)$$

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

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

Hidden layer

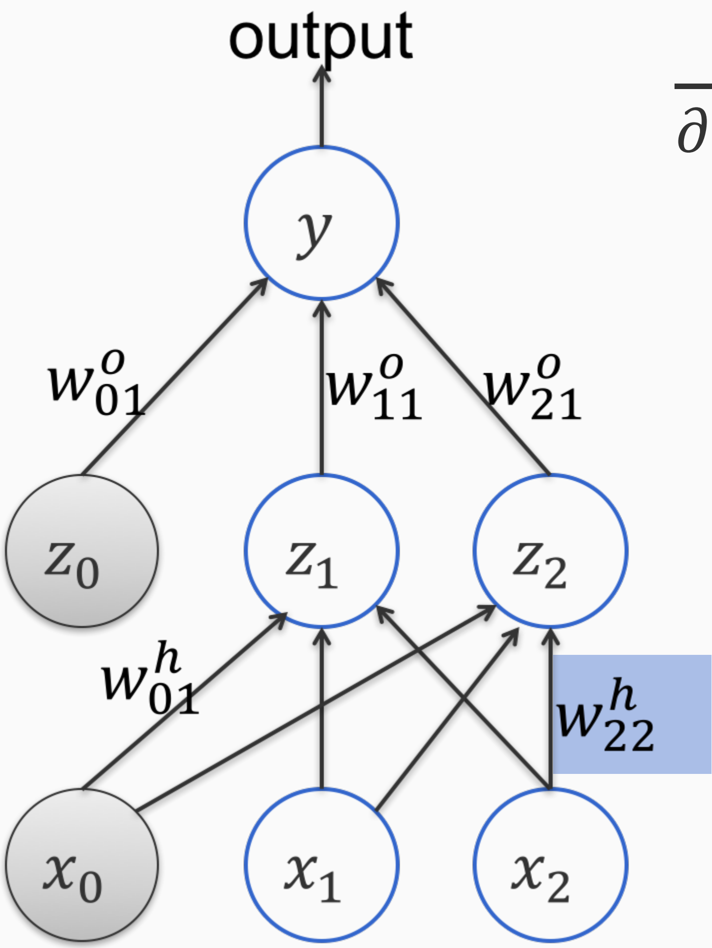
$$z_2 = \sigma(\underbrace{w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2}_{\text{Call this s}})$$



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

Hidden layer

$$z_2 = \sigma(\underbrace{w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2}_{\text{Call this } s})$$

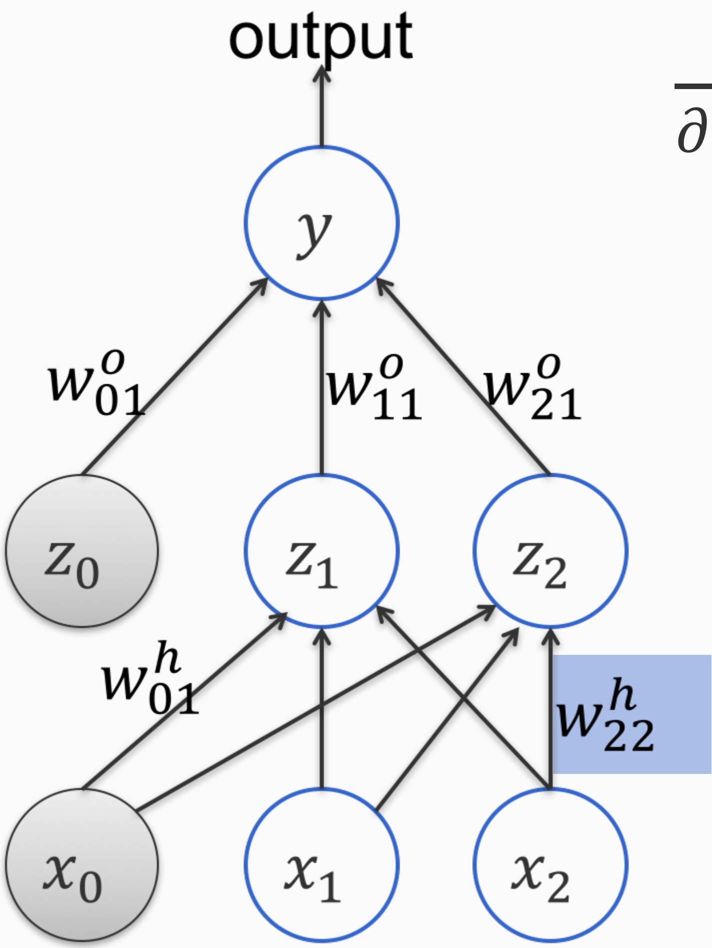


$$\frac{\partial L}{\partial w_{22}^h} = \frac{\partial L}{\partial y} w_{21}^o \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w_{22}^h}$$

Each of these partial derivatives is easy

Hidden layer

$$z_2 = \sigma(\underbrace{w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2}_{\text{Call this s}})$$



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

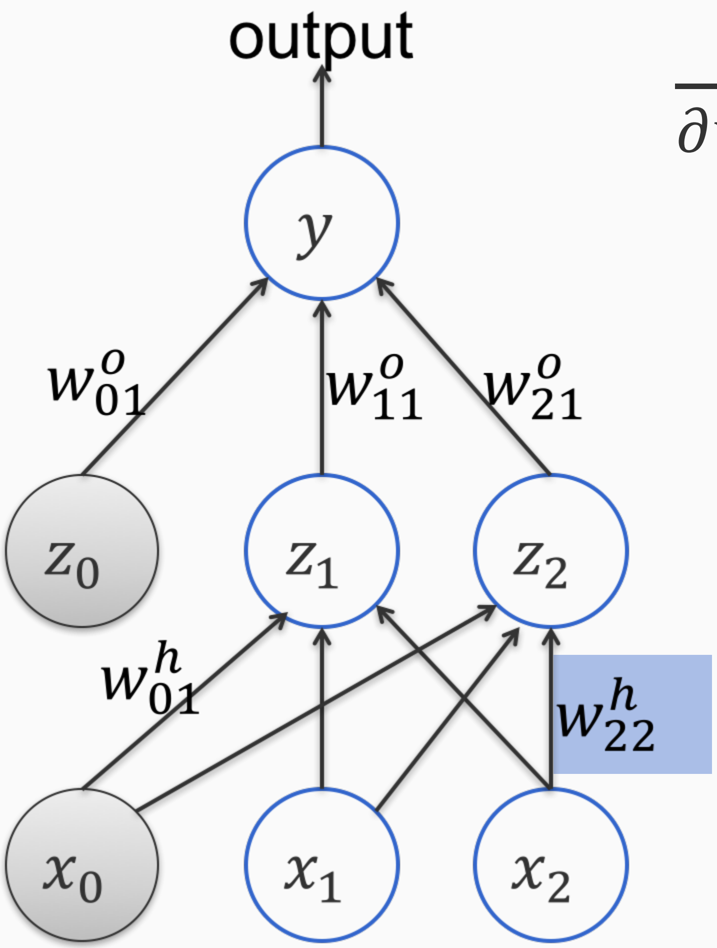
Each of these partial derivatives is easy

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

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

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

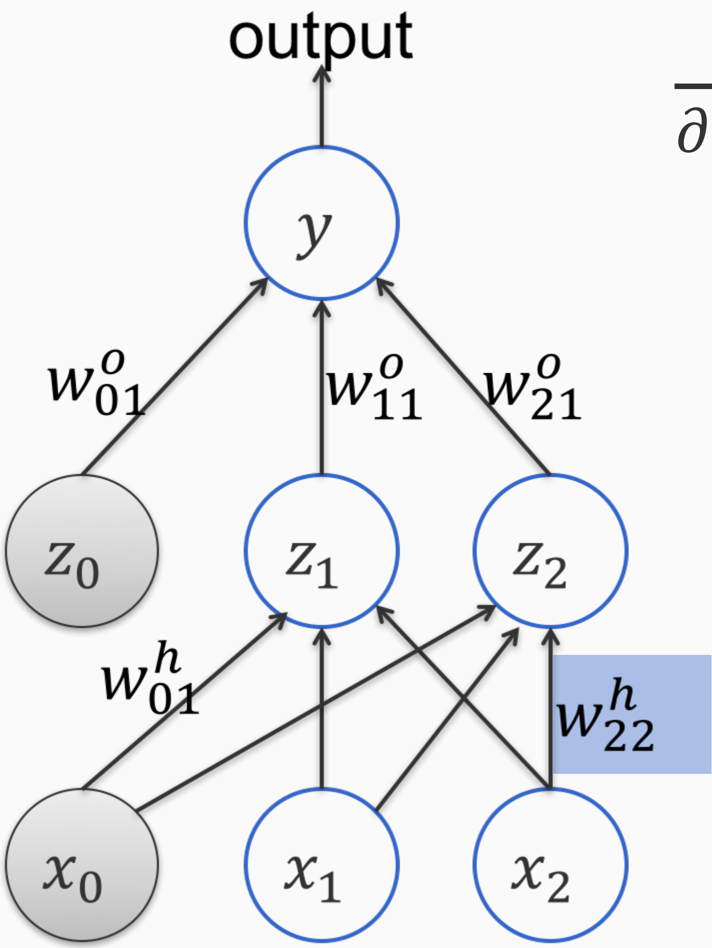
$$\frac{\partial z_2}{\partial s} = z_2(1 - z_2)$$

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

Hidden layer

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

Call this s



$$\frac{\partial L}{\partial w^h_{22}} = \frac{\partial L}{\partial y} w^o_{21} \frac{\partial z_2}{\partial s} \frac{\partial s}{\partial w^h_{22}}$$

Each of these partial derivatives is easy

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

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

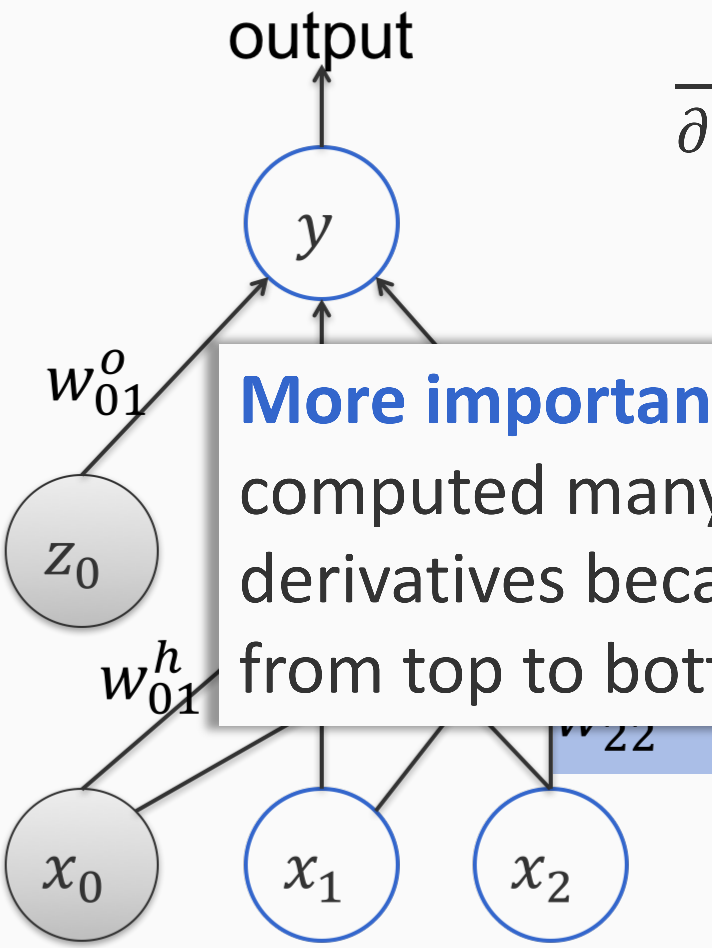
$$\frac{\partial s}{\partial w^h_{22}} = x_2$$

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 (i.e. backwards)

cause $z_2(s)$
istic
we have
een

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

The Backpropagation Algorithm

The same algorithm works for multiple layers, 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 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 $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

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