Neural networks

Computer Vision (CSCI 5520G)

Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca

Lesson Plan

- Feed-forward neural networks
- Activation functions
- Gradient based learning in neural networks
  - Backpropagation
- Layered view of neural networks

Feed forward neural networks

- Approximate some function $y = f^*(\mathbf{x})$ by learning parameters θ s.t. $\tilde{y} = f(\mathbf{x}; \theta)$
- Feed forward neural networks can be seen as *directed acyclic graphs*

$$y = f(\mathbf{x}) = f^{(L)}(\cdots f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))))$$

- Training examples specify the output of the *last* layer
  - Network needs to figure out the inputs/outputs for the *hidden* layers

Extending linear models

How can we extend linear models?

Extending linear models

How can we extend linear models?

- Specify a very general ϕ s.t. the model becomes $y = \theta^T \phi(\mathbf{x})$
  - Problem with generalization
  - Difficult to encode *prior* information needed to solve AI-level tasks

Extending linear models

How can we extend linear models?

- Specify a very general ϕ s.t. the model becomes $y = \theta^T \phi(\mathbf{x})$
  - Problem with generalization
  - Difficult to encode *prior* information needed to solve AI-level tasks
- Engineer ϕ for the task at hand
  - Tedious
  - Difficult to transfer to new tasks

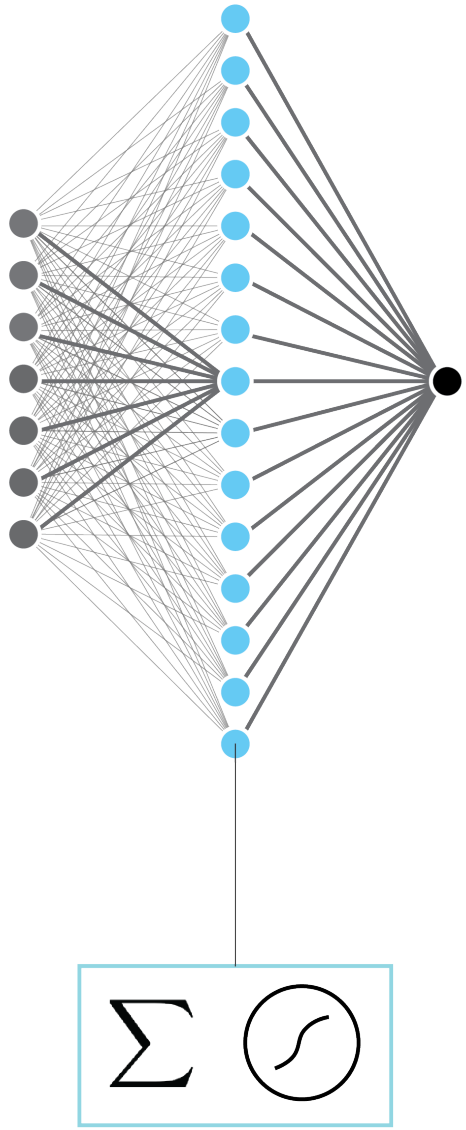
Extending linear models

How can we extend linear models?

- Specify a very general ϕ s.t. the model becomes $y = \theta^T \phi(\mathbf{x})$
  - Problem with generalization
  - Difficult to encode *prior* information needed to solve AI-level tasks
- Engineer ϕ for the task at hand
  - Tedious
  - Difficult to transfer to new tasks
- Neural networks approaches
  - $y = f(\mathbf{x}; \theta, w) = \phi(\mathbf{x}; \theta)^T w$ i.e. use parameters θ to learn ϕ and use w to map $\phi(\mathbf{x})$ to the desired output y
  - The training problem is non-convex
  - Key advantage: a designer just need to specify the right family of functions and not the exact function ϕ

Classical artificial neural networks

- Shallow and wide
- One hidden layer can represent any function
- Focus was on efficient ways to optimize (train)



Classical artificial neural networks

We can represent networks comprising a single hidden layer as

$$y = f_o\left(\mathbf{W}_{oh} f_h\left(\mathbf{W}_{ih}\mathbf{x}\right)\right).$$

Here, $\mathbf{x} \in \mathbb{R}^d$ is the d-dimensional input, $\mathbf{W}_{ih} \in \mathbb{R}^{d_h \times d}$ is the input-to-hidden-layer weight matrix, d_h is the size of the hidden layer, $\mathbf{W}_{ho} \in \mathbb{R}^{1 \times d_h}$ is the hidden-layer-to-output weight matrix, and f_h and f_o are activations functions for hidden layer and output, respectively.

Activation functions

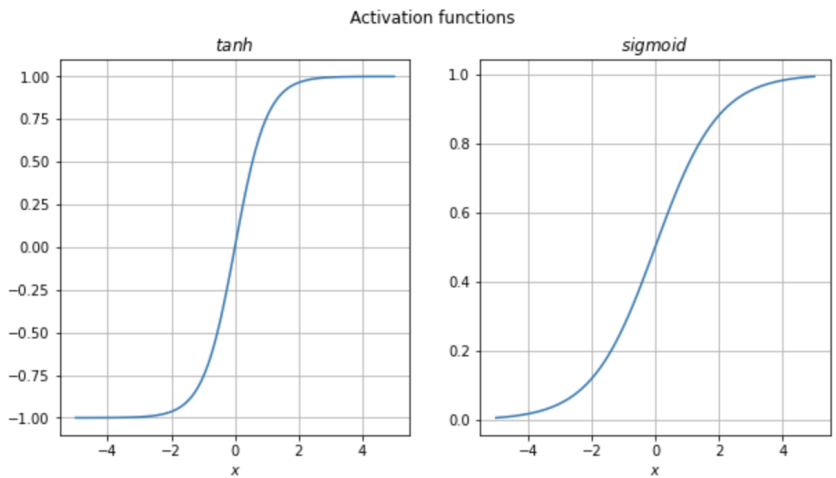
Activation functions take a vector as input and return a vector of the same size. The function is applied element-wise.

Activation functions

Activation functions take a vector as input and return a vector of the same size. The function is applied element-wise.

Traditionally, possible choices for f_h are:

- hyperbolic tangent; and
- sigmoid.

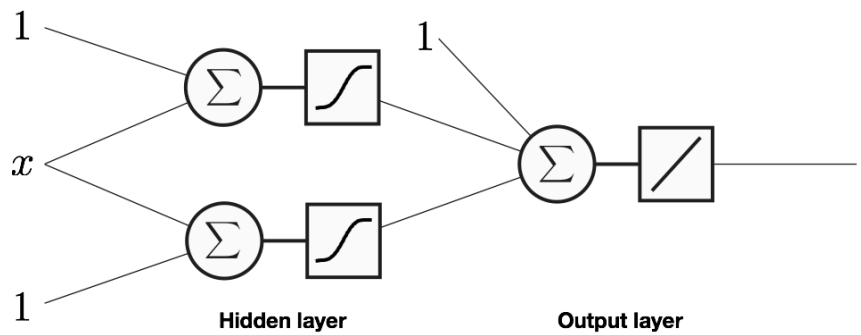


Activation functions

- The activation functions for output:
  - Identity function for regression;
  - Sigmoid for binary classification; and
  - Softmax for multi-class classification.
- The activation functions for hidden layers:
  - tanh (allows for negative output values); and
  - sigmoid.

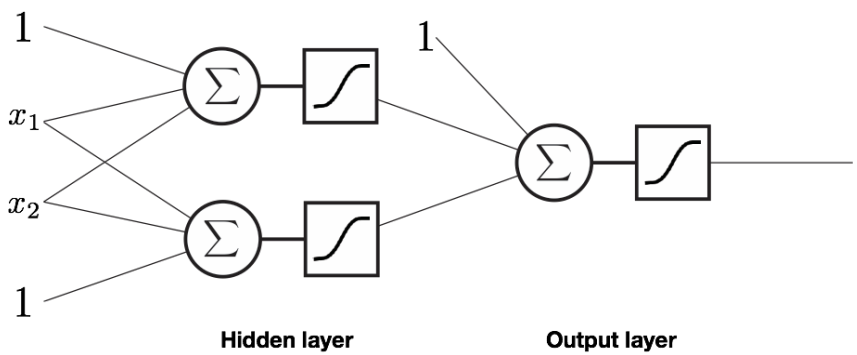
Example: a regression network

- **Input**: 1D, real numbers
- **Ouput**: 1D, real numbers
- **Hidden layer size**: 2
- **Number of weights**: 7
- **Loss**: MSE
- Probabilistic view of line fitting



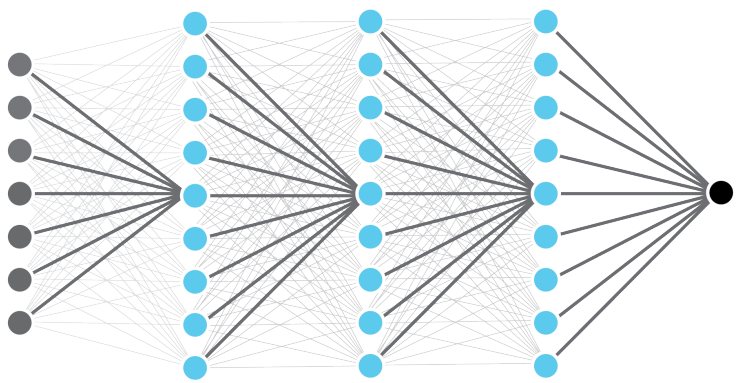
Example: a classification network

- **Input**: 2D, real numbers
- **Output**: 1D, class labels 0 or 1
- **Hidden layer size**: 2
- **Number of weights**: 9
- **Loss**: Cross-entropy
- Data likelihood under Bernoulli distribution



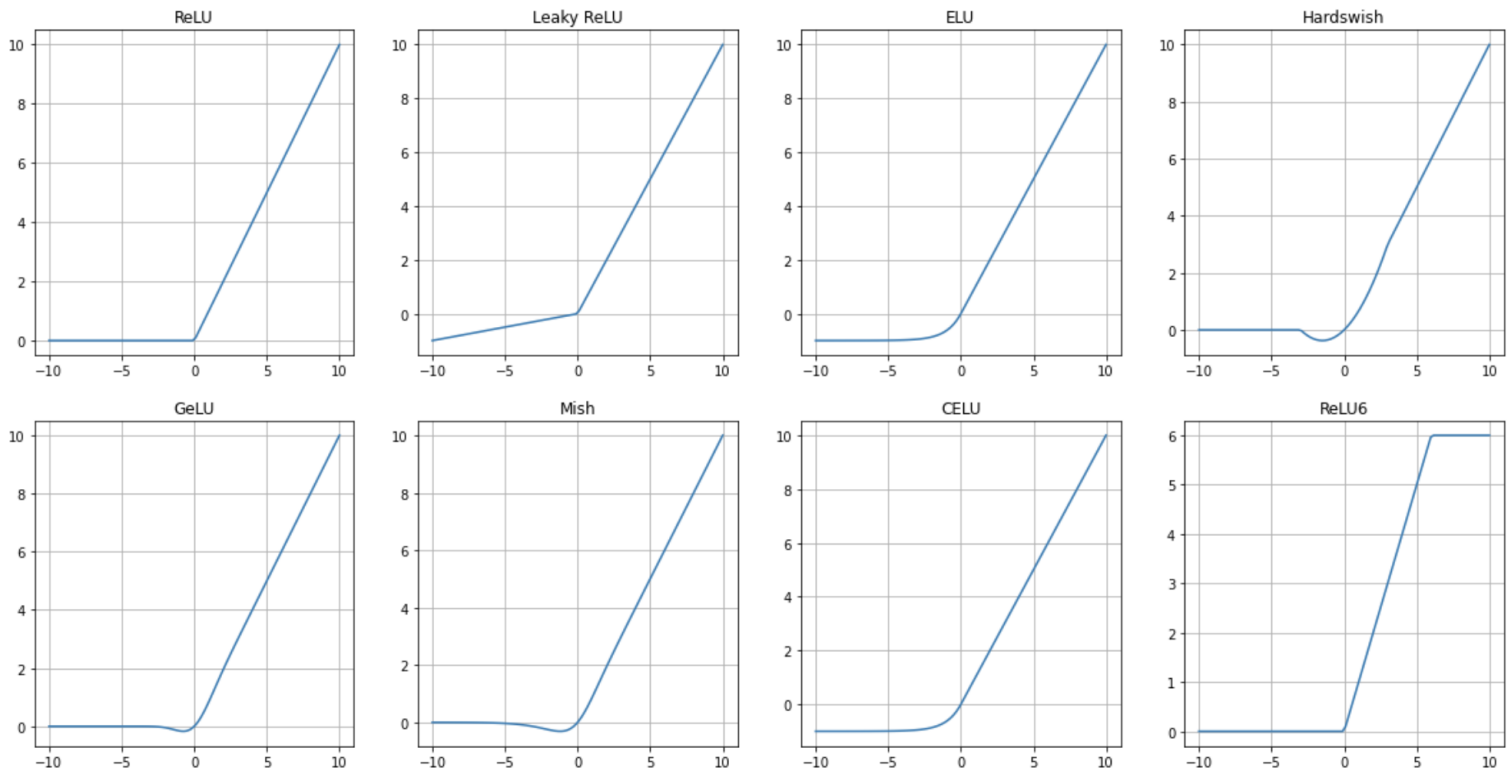
Current view – deep neural networks

- Multi-layer networks
  - These networks are deeper than these are wider
- Hierarchical representation
  - Reduces *semantic gap*
- Deep networks outperform humans on many tasks
- Access to data
- Advances in computer science, physics and engineering



Activation functions

- The activation function for output:
  - Identity function for regression;
  - Sigmoid for binary classification; and
  - Softmax for multi-class classification.
- The following activation functions are typically used for hidden layers.



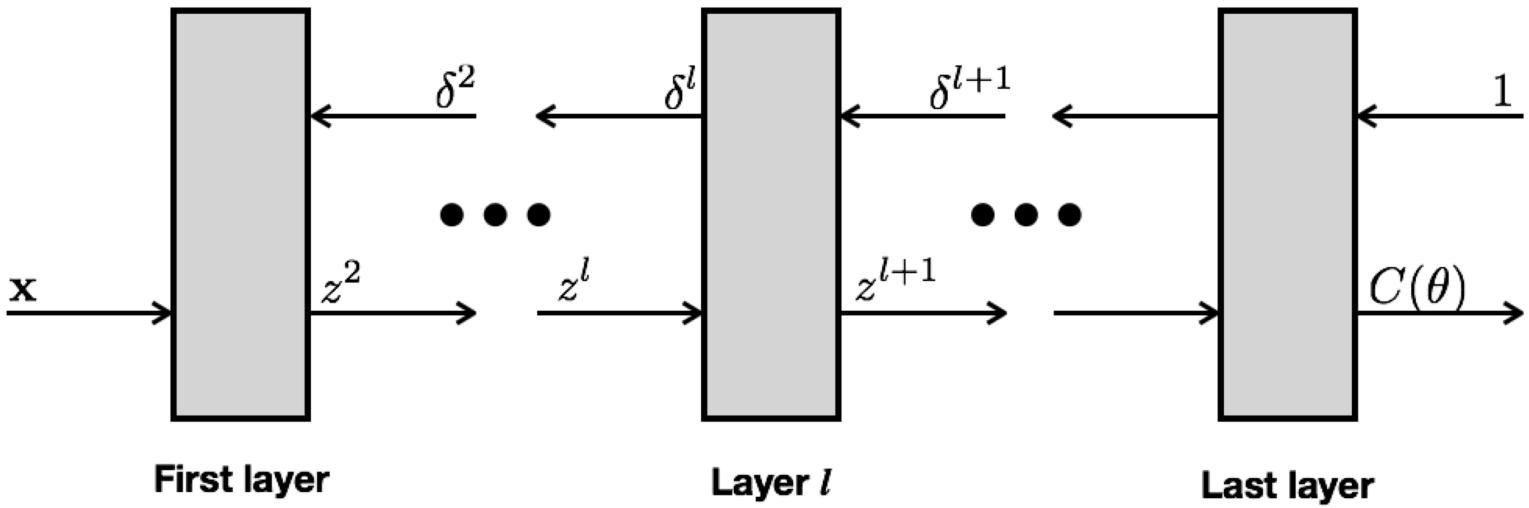
Gradient-based learning in neural networks

- Non-linearities of neural networks render most cost functions non-convex
- Use iterative gradient based optimizers to drive cost function to lower values
- Gradient descent applied to non-convex cost functions has no guarantees is sensitive to initial conditions
  - Initialize weights to small random values
  - Initialize biases to zero or small positive values

Q. How to compute the gradient of cost w.r.t. network weights (parameters)?

Computing gradients

- Backpropagation (Rumelhard et al. 1986) is often used to compute the gradient of the error function w.r.t. to network's weights.
- We will examine backpropagation within a *layered view* of neural networks
  - Treating neural networks as differentiable layers allows a plug-and-play compositional ability enabling us to construct large neural networks



Two-class Softmax classifier

Let's consider a two-class softmax classifier. We define cost $C(\theta)$ that needs to be minimized to train this classifier. As seen earlier, we set this cost equal to the negative log-likelihood for this 2-class softmax classifier.

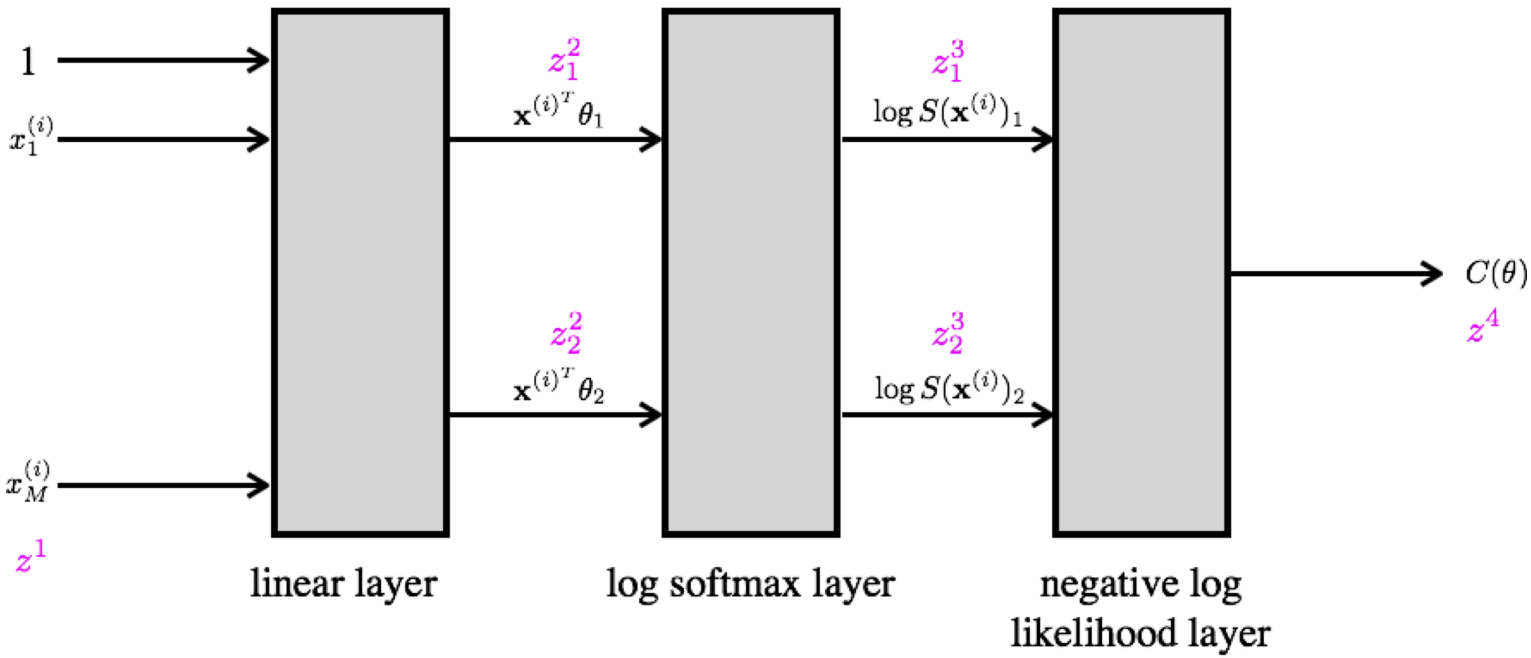
$$C(\theta) = l(\theta)$$
$$= -\sum_{i=1}^{N} \mathbb{I}_0(y^{(i)}) \log \frac{e^{\mathbf{x}^{(i)T}\theta_1}}{e^{\mathbf{x}^{(i)T}\theta_1} + e^{\mathbf{x}^{(i)T}\theta_2}} + \mathbb{I}_1(y^{(i)}) \log \frac{e^{\mathbf{x}^{(i)T}\theta_2}}{e^{\mathbf{x}^{(i)T}\theta_1} + e^{\mathbf{x}^{(i)T}\theta_2}}$$

We need to compute the gradient of this loss w.r.t. network parameters for gradient-based learning.

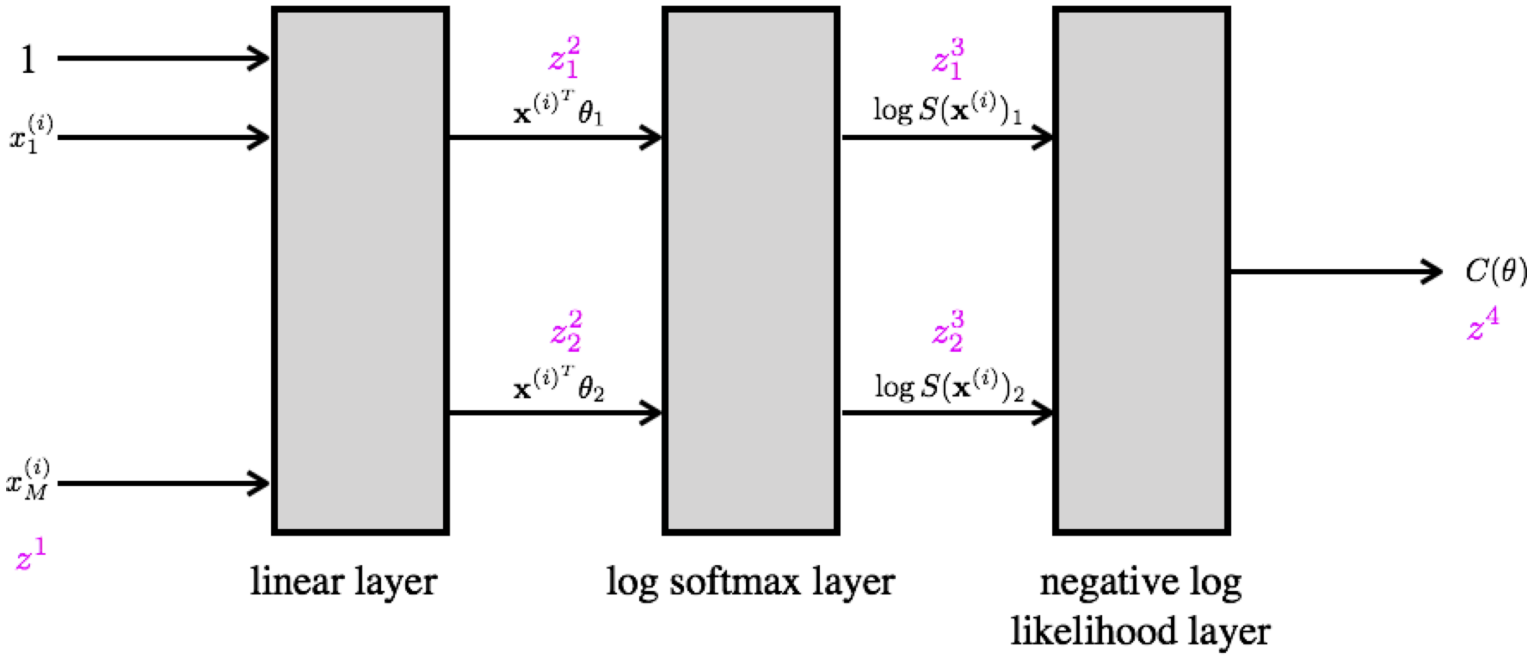
Layer representation

$$C(\theta) = -\sum_{i=1}^{N} \mathbb{I}_0(y^{(i)}) \log \frac{e^{\mathbf{x}^{(i)T}\theta_1}}{e^{\mathbf{x}^{(i)T}\theta_1} + e^{\mathbf{x}^{(i)T}\theta_2}} + \mathbb{I}_1(y^{(i)}) \log \frac{e^{\mathbf{x}^{(i)T}\theta_2}}{e^{\mathbf{x}^{(i)T}\theta_1} + e^{\mathbf{x}^{(i)T}\theta_2}}$$

Layers

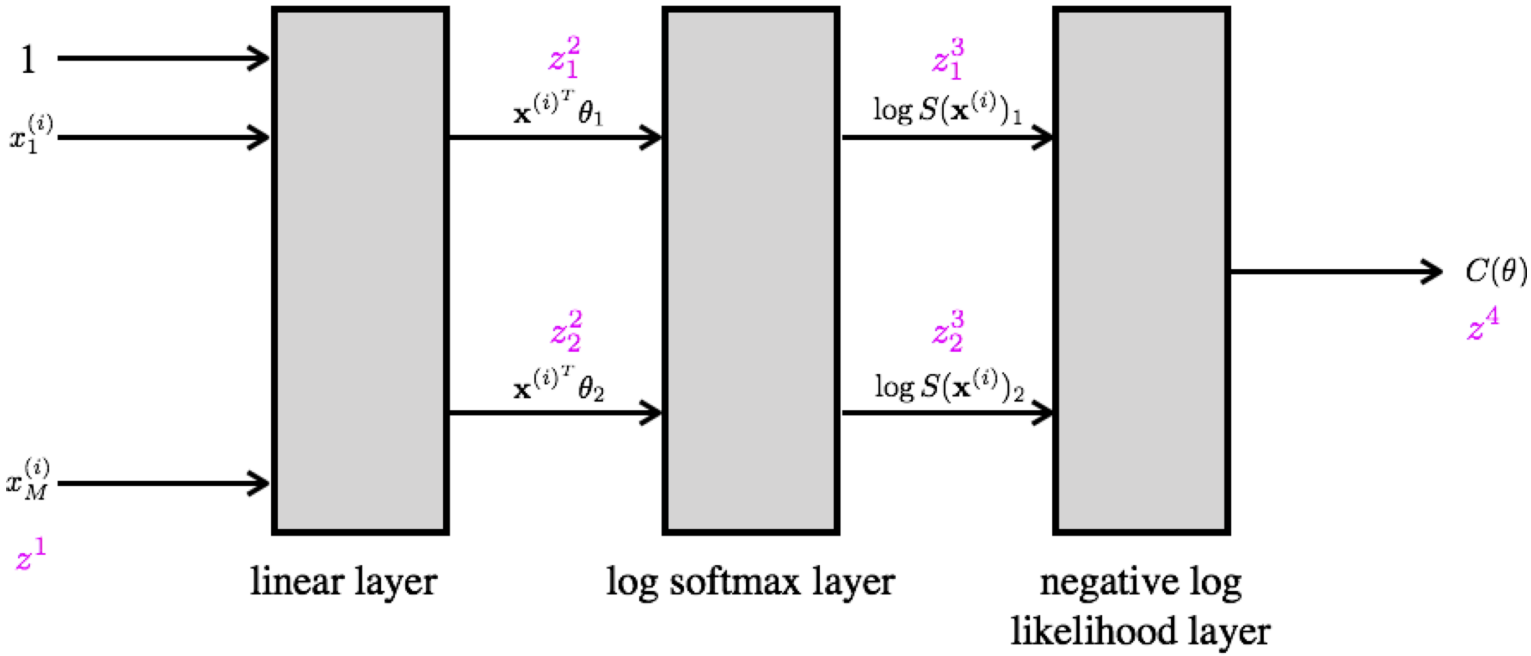


Layer representation



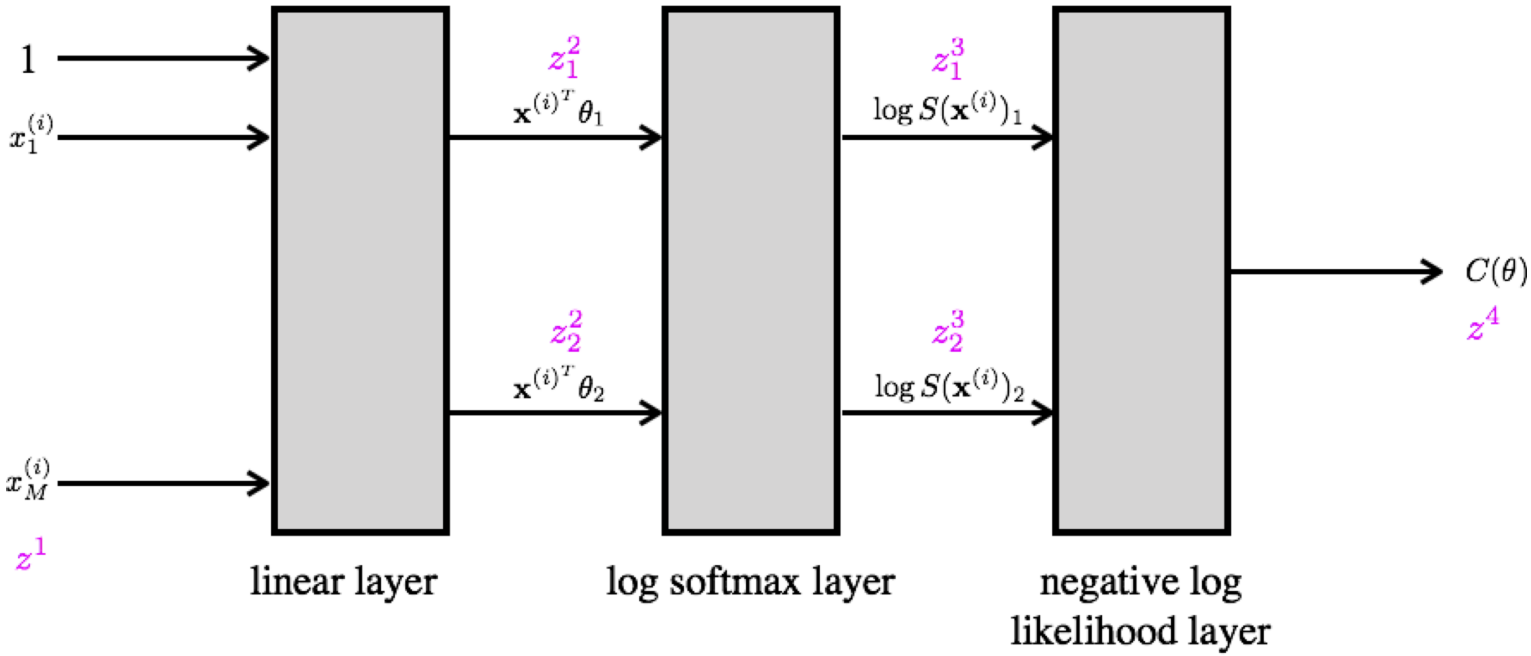
Which are the network parameters?

Layer representation



Which are the network parameters? θ_1 and θ_2

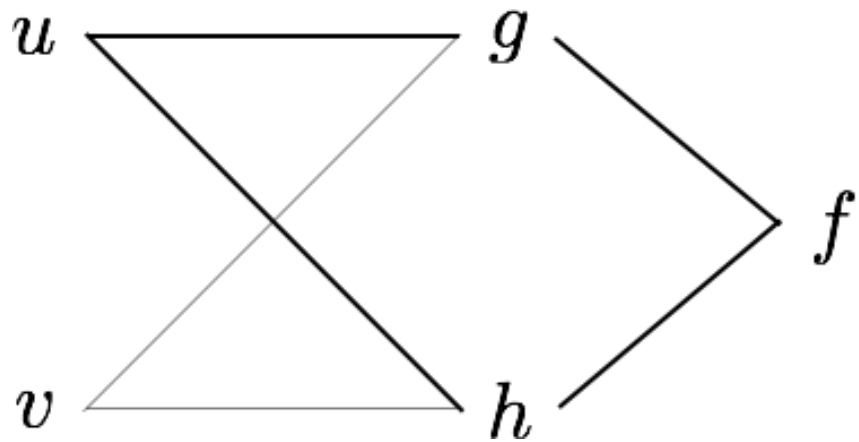
Layer representation



Which are the network parameters? θ_1 and θ_2

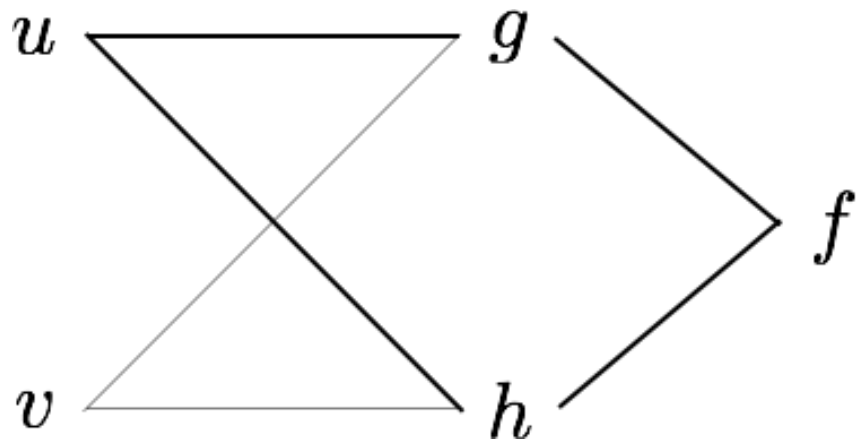
We need to compute $\frac{\partial C(\theta)}{\partial \theta_1}$ and $\frac{\partial C(\theta)}{\partial \theta_2}$.

Chain rule



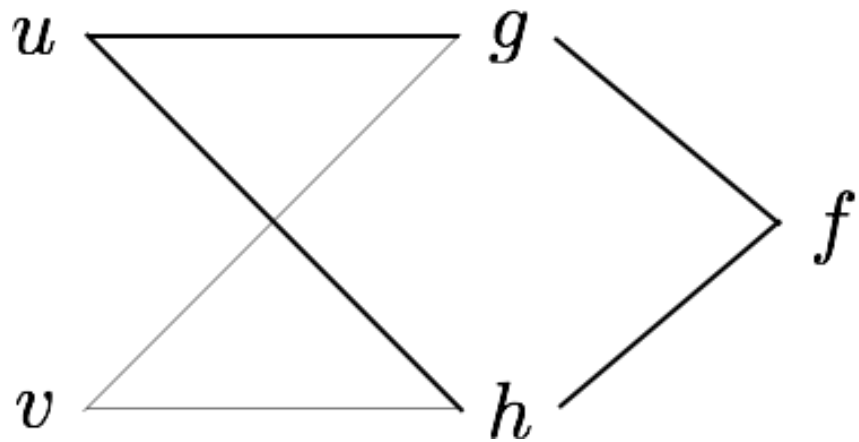
$$\underbrace{\frac{\partial f(g(u,v),h(u,v))}{\partial u}}_{\text{how does } f \text{ changes when } u \text{ is changed?}} =$$

Chain rule



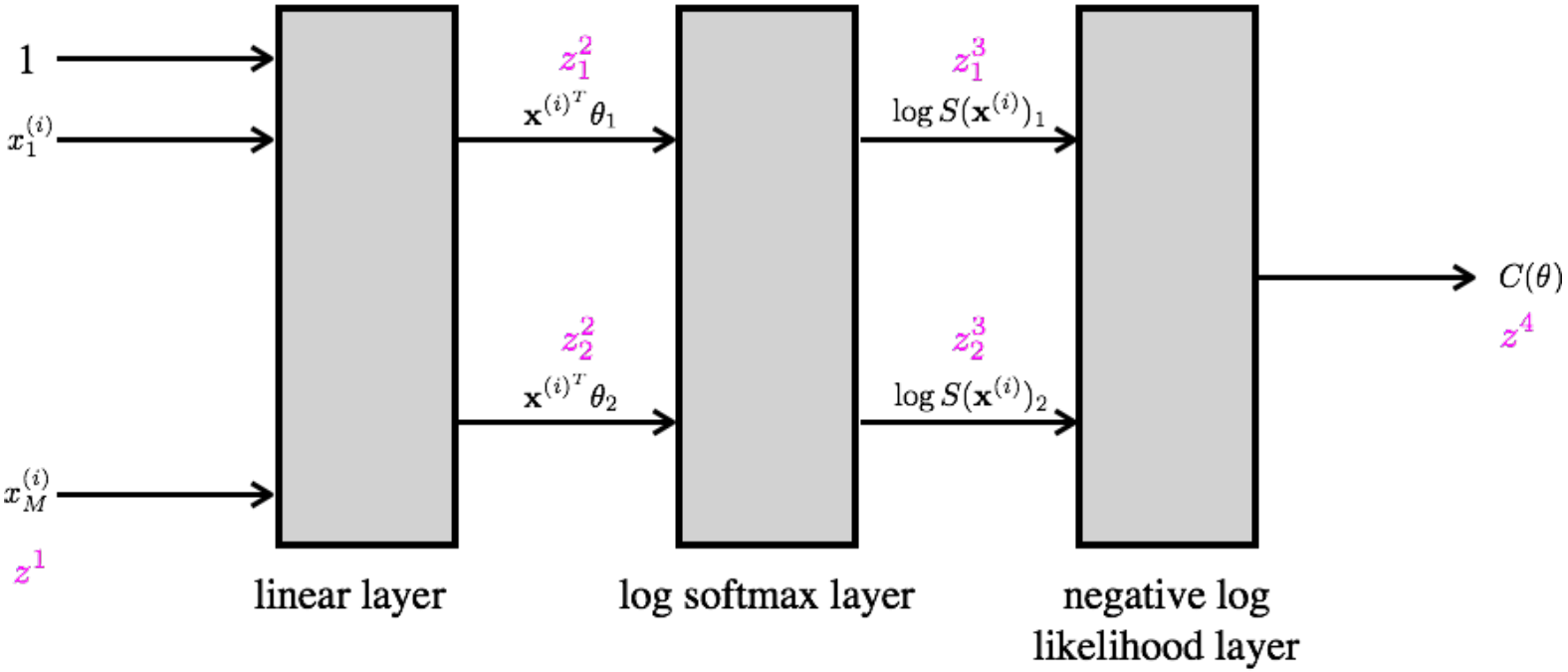
$$\underbrace{\frac{\partial f(g(u,v),h(u,v))}{\partial u}}_{\text{how does } f \text{ changes when } u \text{ is changed?}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial u}$$

Chain rule



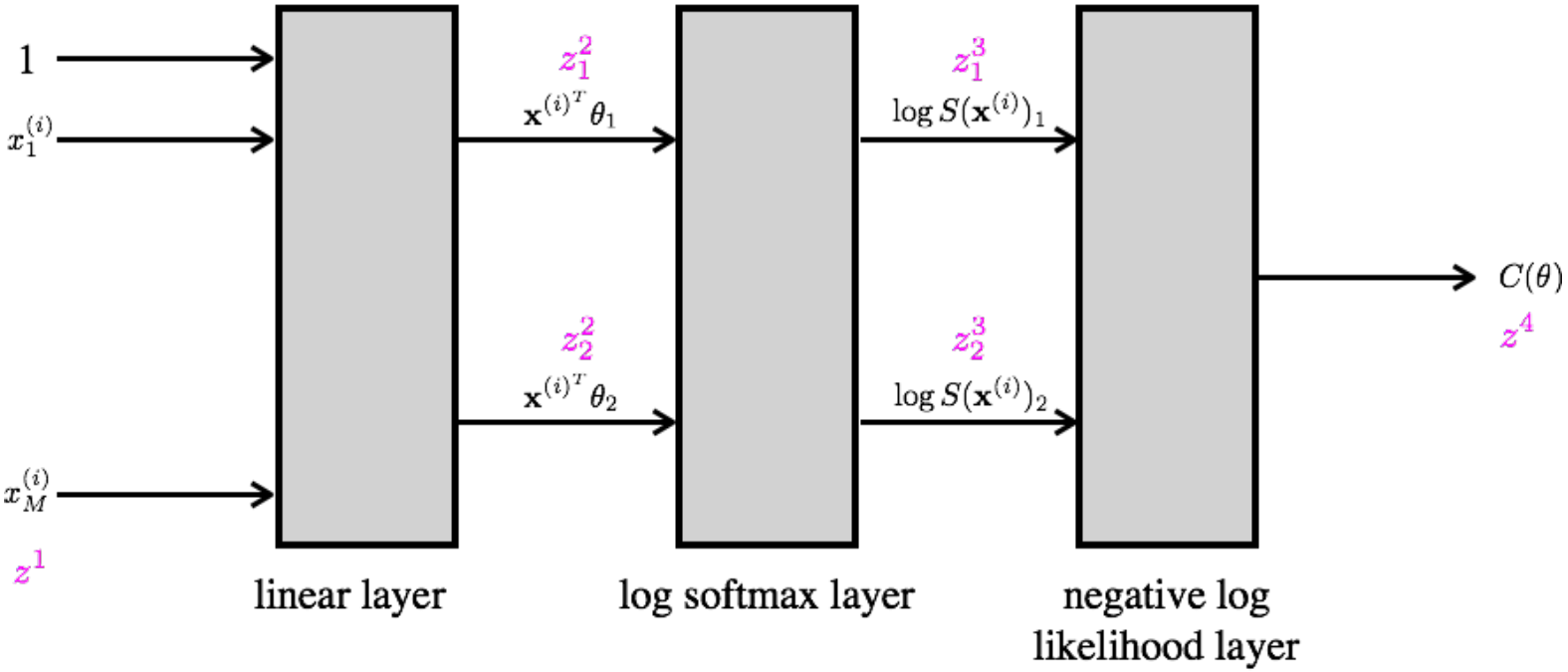
$$\underbrace{\frac{\partial f(g(u,v),h(u,v))}{\partial u}}_{\text{how does } f \text{ changes when } u \text{ is changed?}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial u} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial u}$$

Back to layer representation



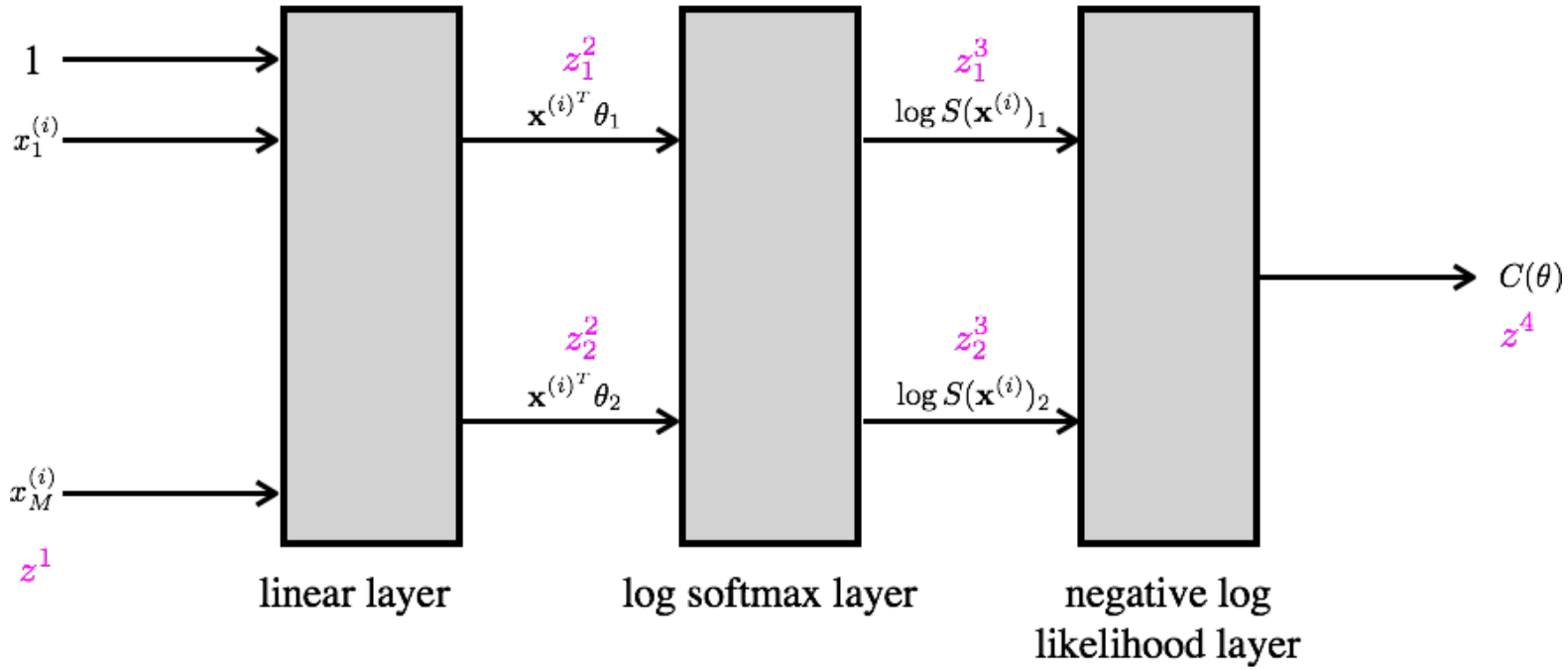
- What are our model paramters?

Back to layer representation



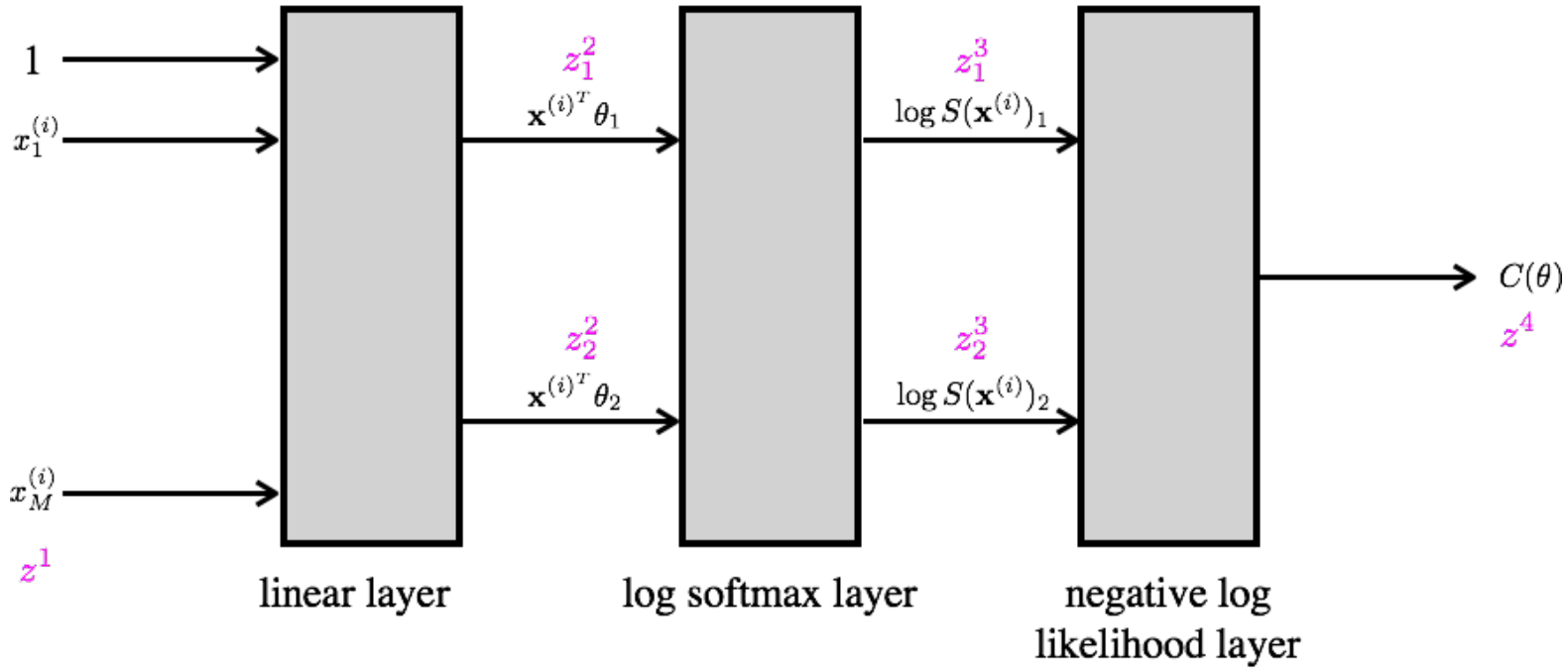
- What are our model paramters? (θ_1 and θ_2)
- We are interested in computing $\frac{\partial z^4}{\partial \theta_1}$ and ($\frac{\partial z^4}{\partial \theta_2}$)
  - Why?

Back to layer representation



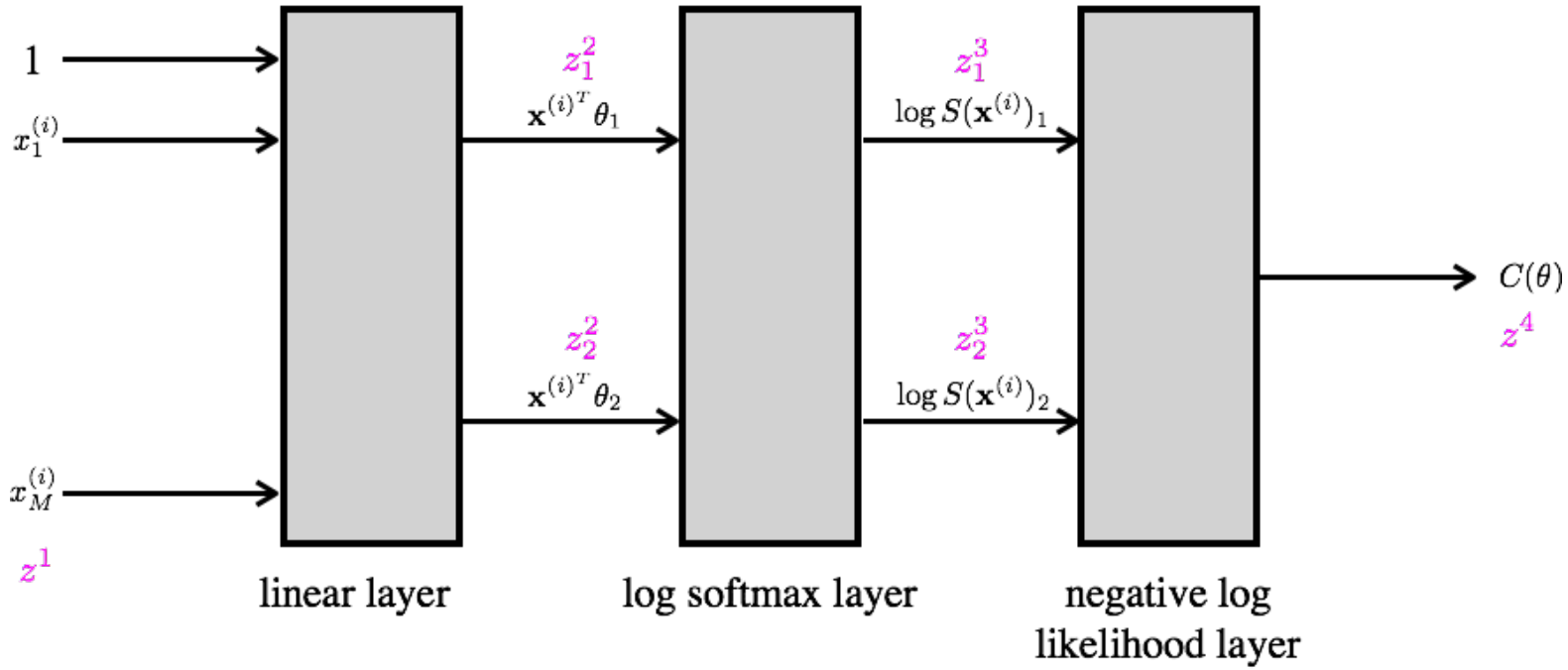
- What are our model paramters? (θ_1 and θ_2)
- We are interested in computing $\frac{\partial z^4}{\partial \theta_1}$ and ($\frac{\partial z^4}{\partial \theta_2}$)
  - Why? (z^4 is the cost that we can minimize using gradient descent using these gradients)

Back to layer representation



- What are our model paramters? (θ_1 and θ_2)
- We are interested in computing $\frac{\partial z^4}{\partial \theta_1}$ and ($\frac{\partial z^4}{\partial \theta_2}$)
  - Why? (z^4 is the cost that we can minimize using gradient descent using these gradients)
  - How do we even compute $\frac{\partial z^4}{\partial \theta_1}$?

Back to layer representation

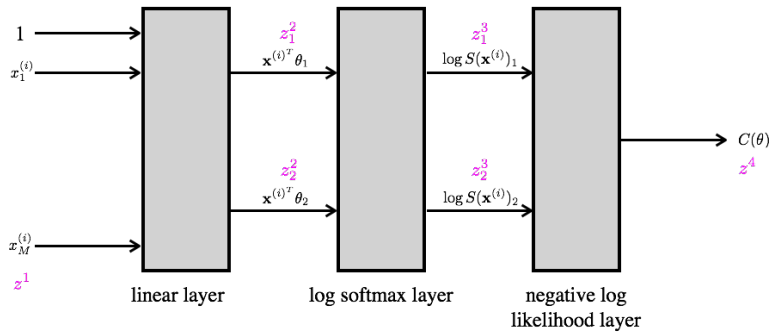


- What are our model paramters? (θ_1 and θ_2)
- We are interested in computing $\frac{\partial z^4}{\partial \theta_1}$ and ($\frac{\partial z^4}{\partial \theta_2}$)
  - Why? (z^4 is the cost that we can minimize using gradient descent using these gradients)
  - How do we even compute $\frac{\partial z^4}{\partial \theta_1}$? (or $\frac{\partial z^4}{\partial \theta_2}$)

Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

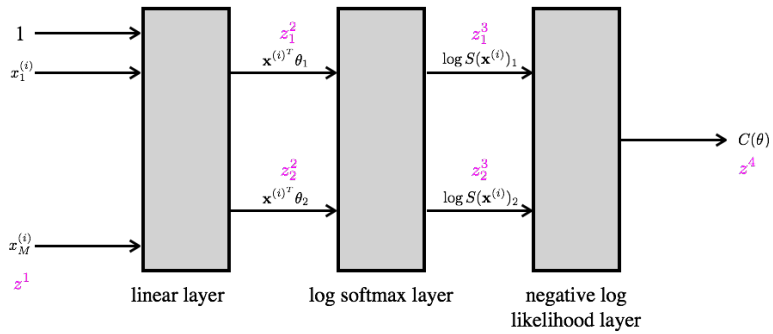
$$\frac{\partial z^4}{\partial \theta_1} =$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

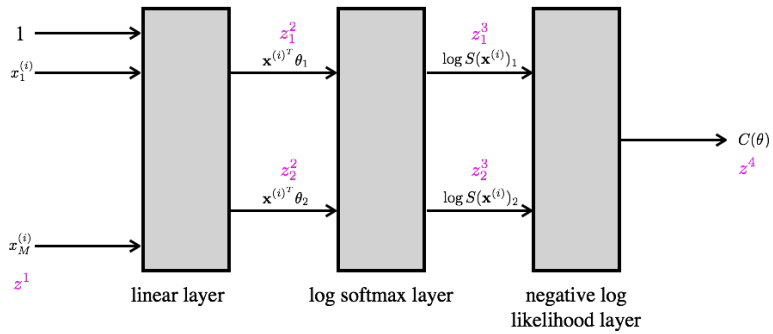
$$\frac{\partial z^4}{\partial \theta_1} = \frac{\partial z^4}{\partial z_1^3} \frac{\partial z_1^3}{\partial \theta_1}$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

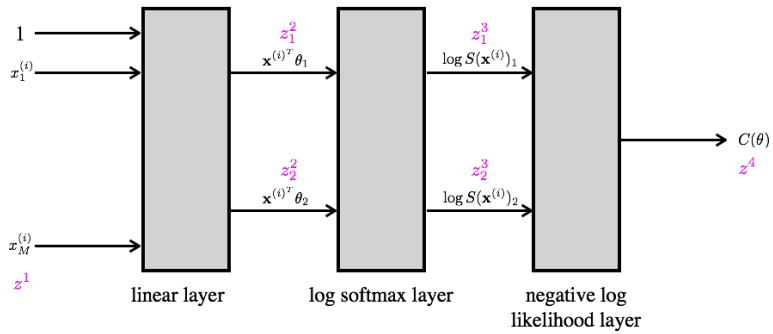
$$\begin{aligned}\frac{\partial z^4}{\partial \theta_1} &= \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial \theta_1} \\ &= \frac{\partial z^4}{\partial z_1^3}\end{aligned}$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

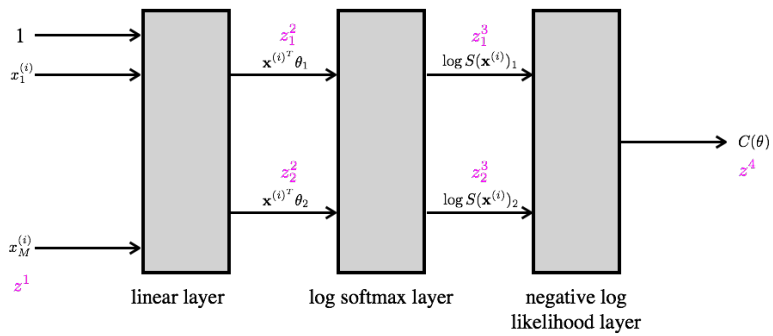
$$\begin{aligned}\frac{\partial z^4}{\partial \theta_1} &= \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial \theta_1} \\ &= \frac{\partial z^4}{\partial z_1^3}\left(\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_1^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right)\end{aligned}$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

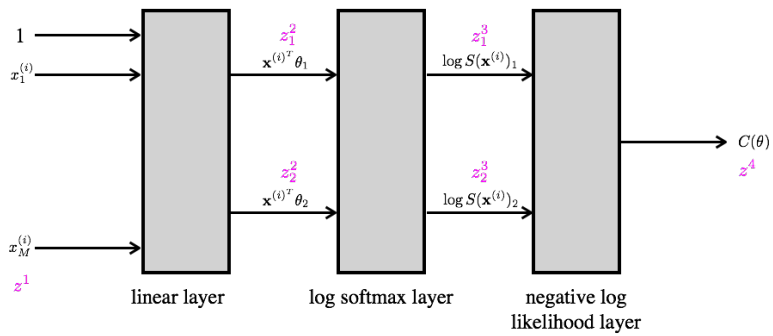
$$\begin{aligned}
\frac{\partial z^4}{\partial \theta_1} &= \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial \theta_1} \\
&= \frac{\partial z^4}{\partial z_1^3}\left(\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_1^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right) \\
&\quad + \frac{\partial z^4}{\partial z_2^3}
\end{aligned}$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

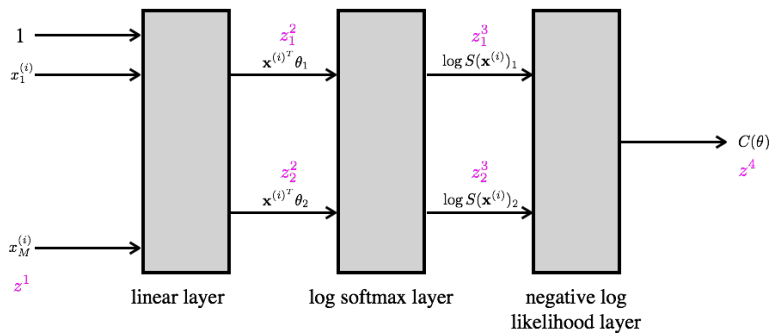
$$
\begin{aligned}
\frac{\partial z^4}{\partial \theta_1} &= \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial \theta_1} \\
&= \frac{\partial z^4}{\partial z_1^3}\left(\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_1^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right) \\
&+ \frac{\partial z^4}{\partial z_2^3}\left(\frac{\partial z_2^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_2^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right)
\end{aligned}
$$



Computing $\frac{\partial z^4}{\partial \theta_1}$

We can use the *chain rule* to compute $\frac{\partial z^4}{\partial \theta_1}$.

$$\begin{aligned}\frac{\partial z^4}{\partial \theta_1} &= \frac{\partial z^4}{\partial z_1^3}\frac{\partial z_1^3}{\partial \theta_1} + \frac{\partial z^4}{\partial z_2^3}\frac{\partial z_2^3}{\partial \theta_1} \\ &= \frac{\partial z^4}{\partial z_1^3}\left(\frac{\partial z_1^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_1^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right) \\ &+ \frac{\partial z^4}{\partial z_2^3}\left(\frac{\partial z_2^3}{\partial z_1^2}\frac{\partial z_1^2}{\partial \theta_1} + \frac{\partial z_2^3}{\partial z_2^2}\frac{\partial z_2^2}{\partial \theta_1}\right)\end{aligned}$$



We can similarly compute $\frac{\partial z^4}{\partial \theta_2}$.

Notation

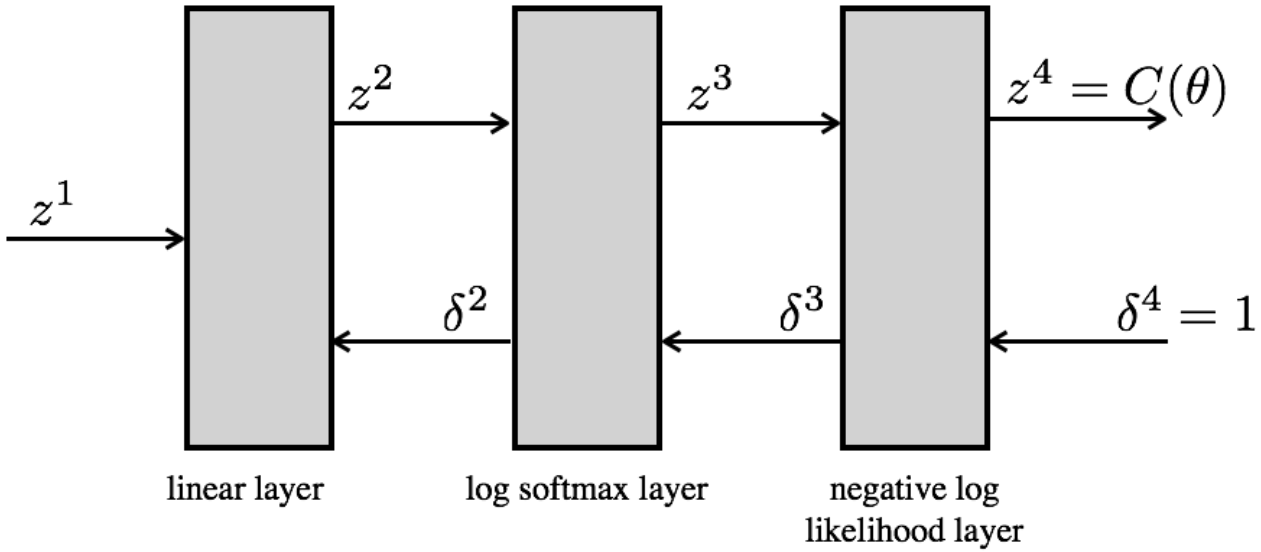
- z^l: input to layer l
- z^{l+1}: output of layer l
- $C(\theta)$: cost (or loss) to be minimized, e.g., negative log-likelihood for the case of the two-class softmax classifier
- Define

$$\delta^l = \frac{\partial C(\theta)}{\partial z^l}$$

- If L denotes the last layer, $\delta^L = 1$, since the output of the last layer is z^L which is equal to the cost $C(\theta)$, i.e.,

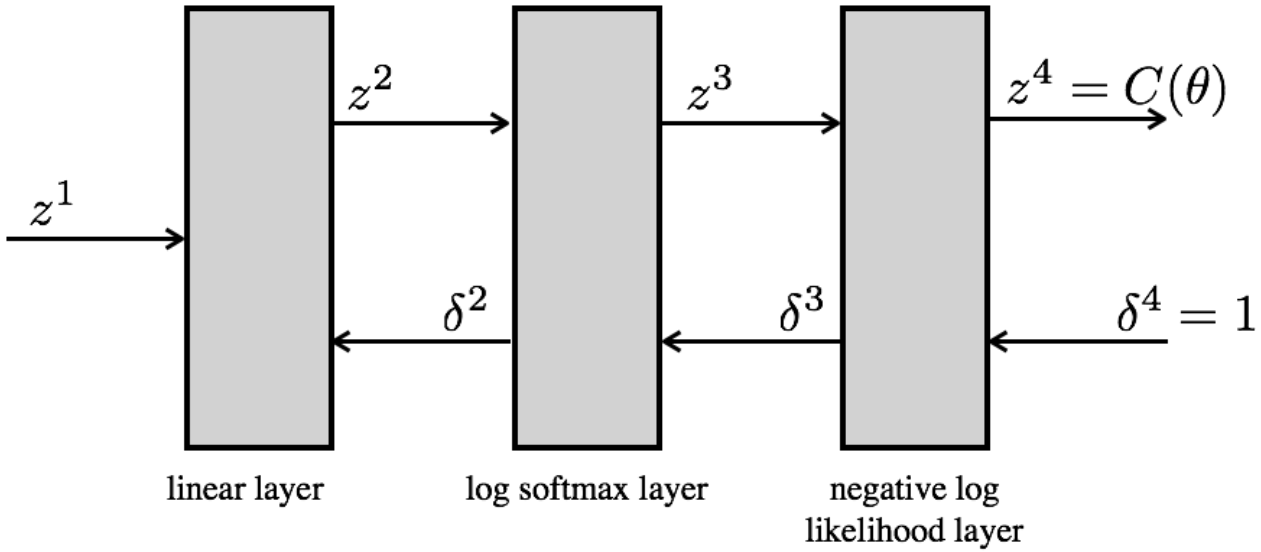
$$\delta^L = \frac{\partial C(\theta)}{\partial z^L} = \frac{\partial z^L}{\partial z^L} = 1$$

Forward pass and backward pass



Forward pass **Backward pass**

Forward pass and backward pass

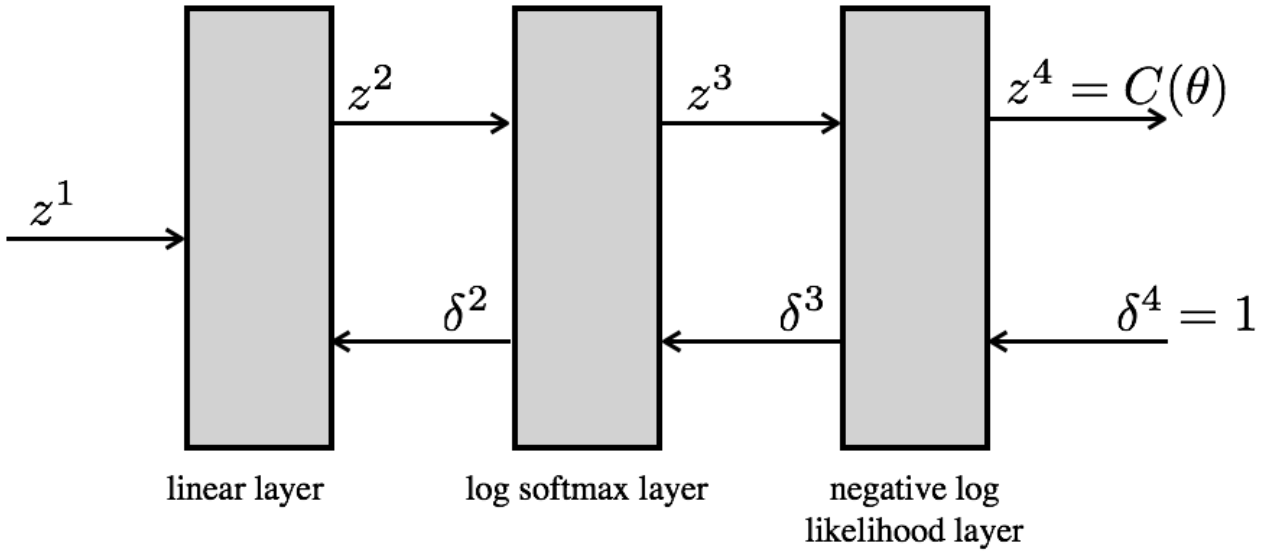


Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

Backward pass

Forward pass and backward pass



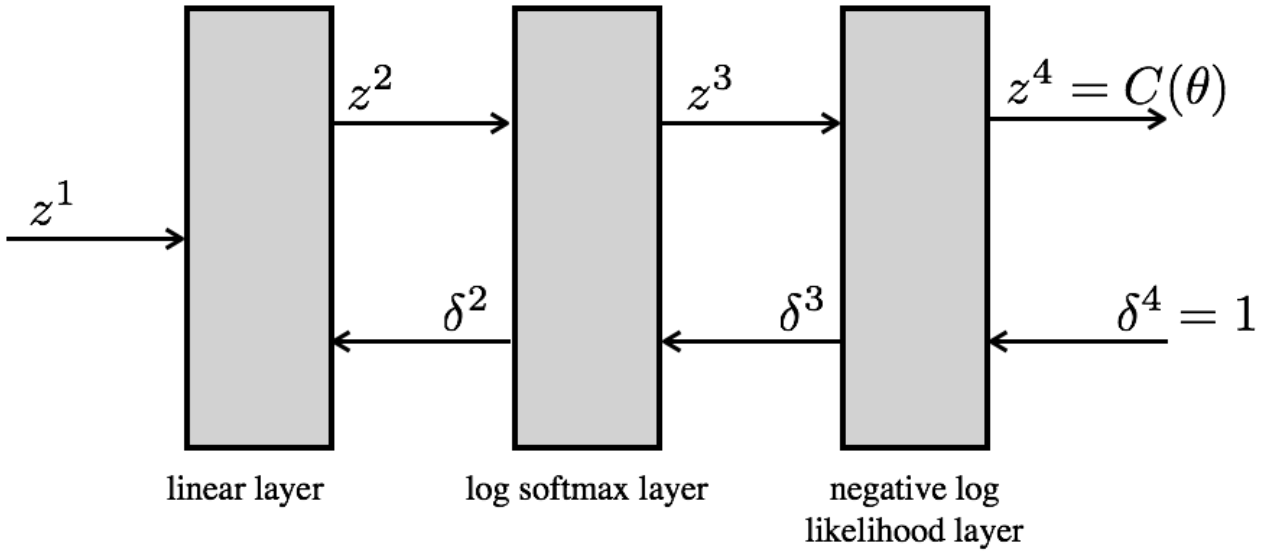
Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

$z^2 = f(z^1)$ (*linear function*)

Backward pass

Forward pass and backward pass



Forward pass

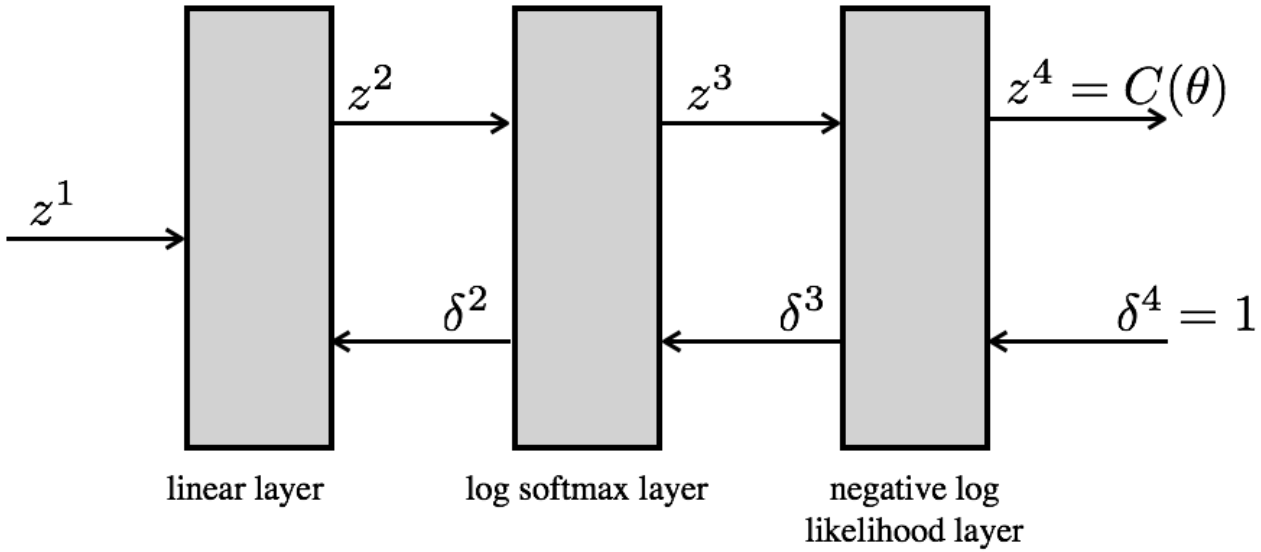
$z^1 = f(\mathbf{x})$ (*input data*)

$z^2 = f(z^1)$ (*linear function*)

$z^3 = f(z^2)$ (*log softmax*)

Backward pass

Forward pass and backward pass



Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

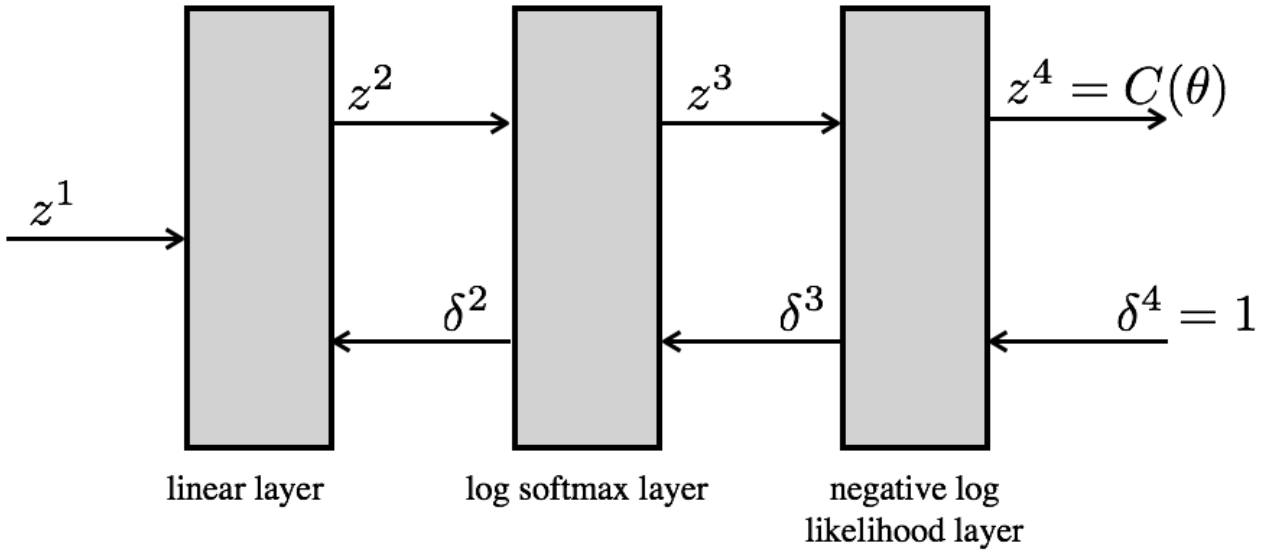
$z^2 = f(z^1)$ (*linear function*)

$z^3 = f(z^2)$ (*log softmax*)

$z^4 = f(z^3) = C(\theta)$ (*negative log likelihood*, cost)

Backward pass

Forward pass and backward pass



Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

$z^2 = f(z^1)$ (*linear function*)

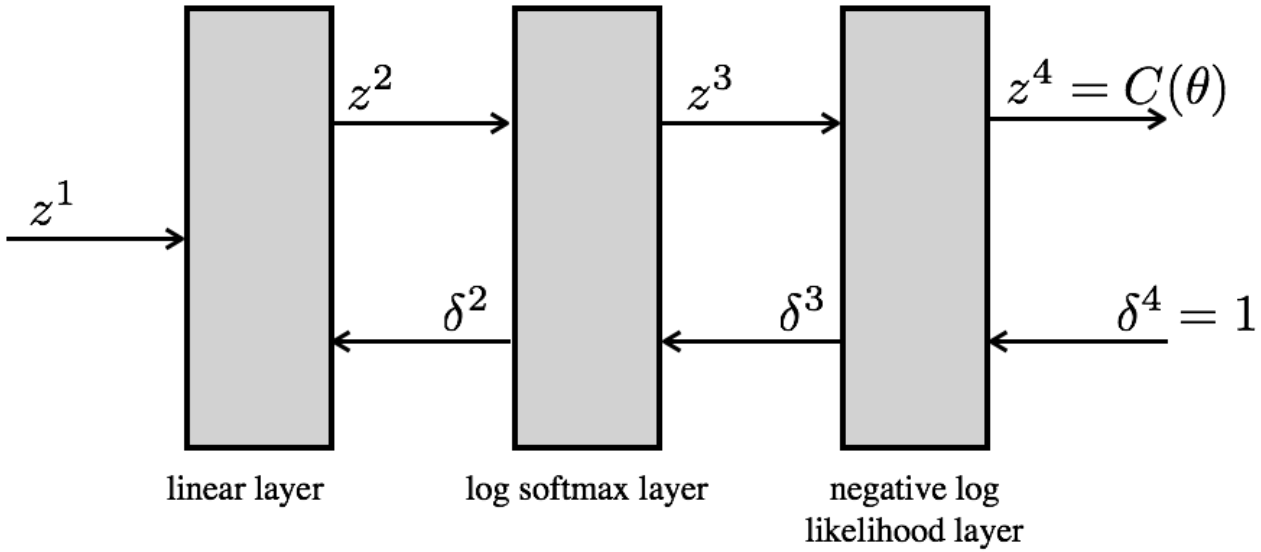
$z^3 = f(z^2)$ (*log softmax*)

$z^4 = f(z^3) = C(\theta)$ (*negative log likelihood*, cost)

Backward pass

$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = 1$

Forward pass and backward pass



Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

$z^2 = f(z^1)$ (*linear function*)

$z^3 = f(z^2)$ (*log softmax*)

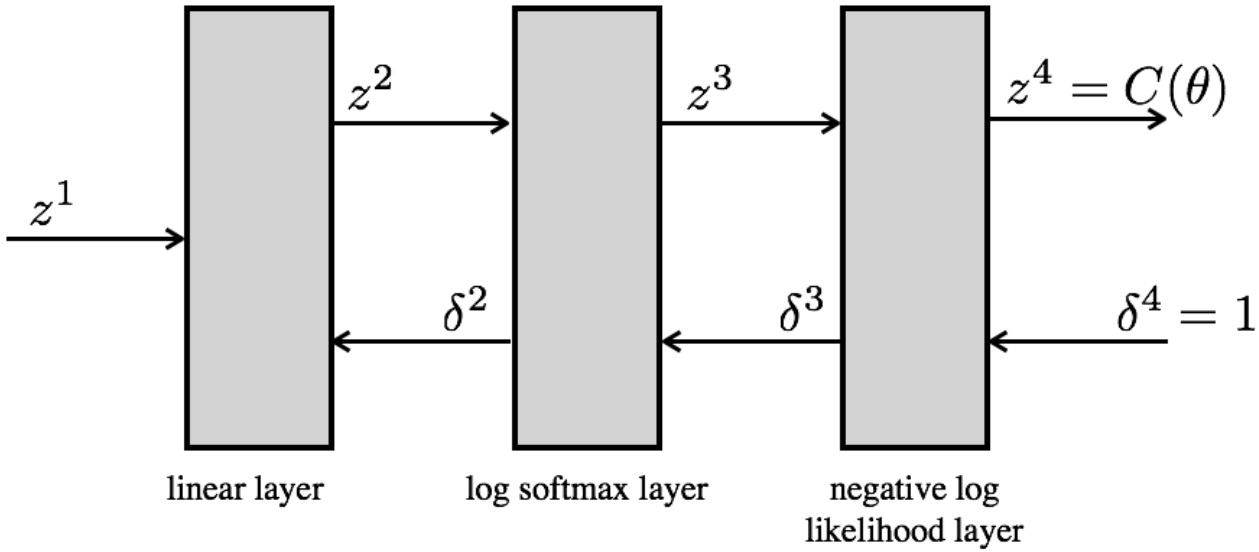
$z^4 = f(z^3) = C(\theta)$ (*negative log likelihood*, cost)

Backward pass

$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = 1$

$\delta^3 = \frac{\partial C(\theta)}{\partial z^3}$

Forward pass and backward pass



Forward pass

$z^1 = f(\mathbf{x})$ (*input data*)

$z^2 = f(z^1)$ (*linear function*)

$z^3 = f(z^2)$ (*log softmax*)

$z^4 = f(z^3) = C(\theta)$ (*negative log likelihood*, cost)

Backward pass

$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = 1$

$\delta^3 = \frac{\partial C(\theta)}{\partial z^3}$

$\delta^2 = \frac{\partial C(\theta)}{\partial z^2}$

Computing δ^l

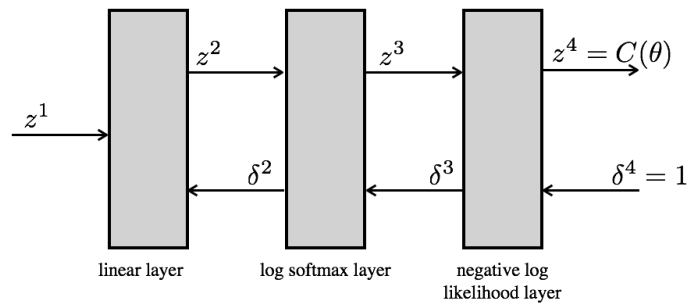
$$\delta^4 =$$

$$\delta_1^3 =$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

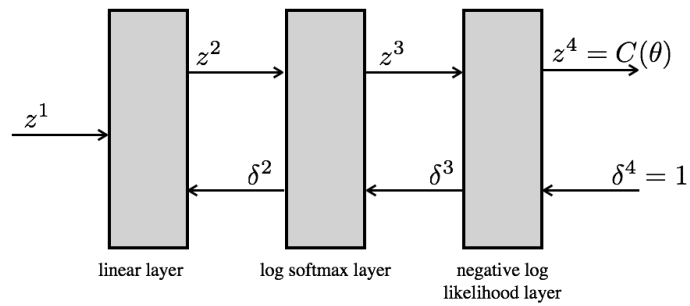
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4}$$

$$\delta_1^3 =$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

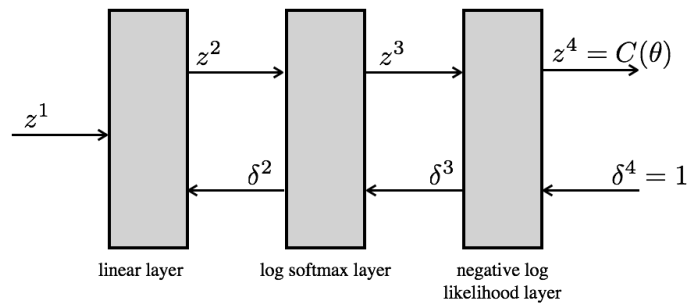
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 =$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

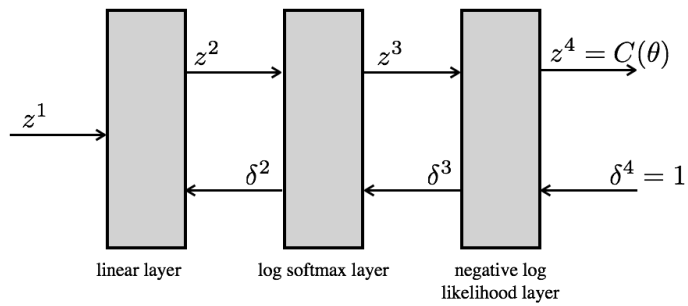
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} =$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

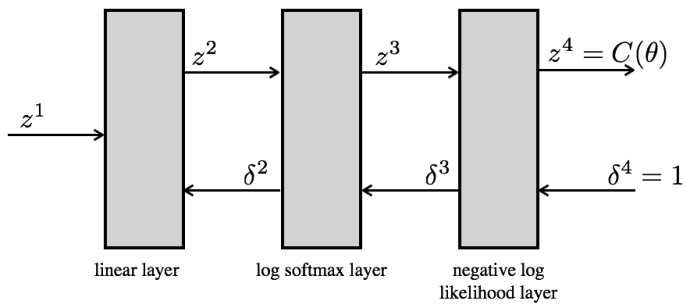
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

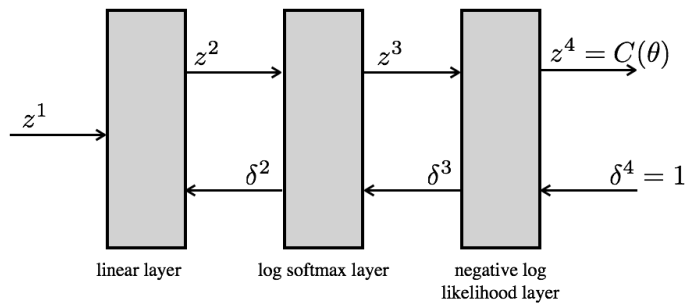
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 =$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

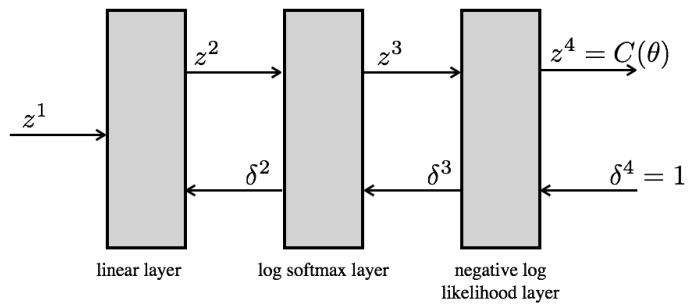
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3}$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

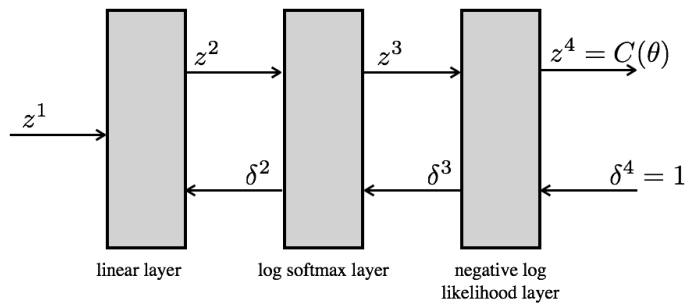
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

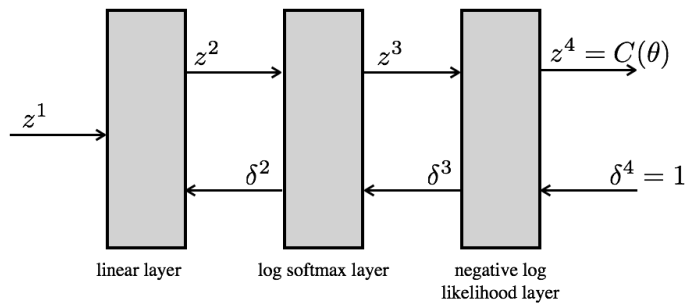
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 =$$

$$\delta_2^2 =$$



Computing δ^l

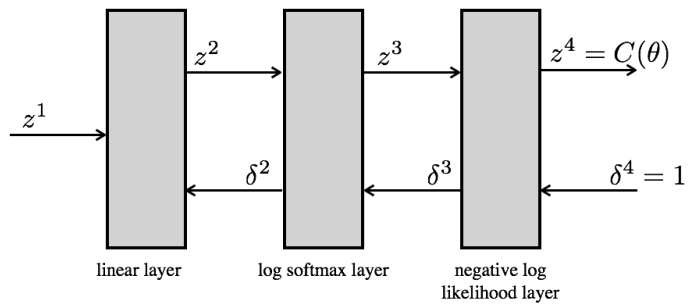
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2}$$

$$\delta_2^2 =$$



Computing δ^l

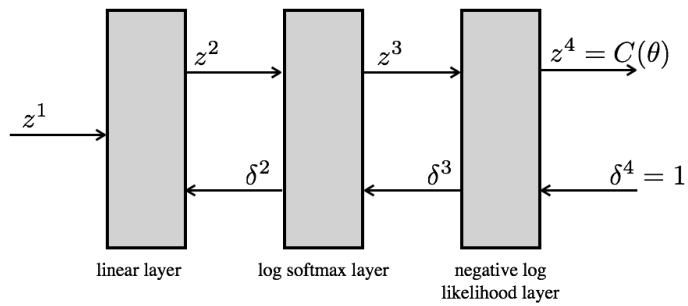
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 =$$



Computing δ^l

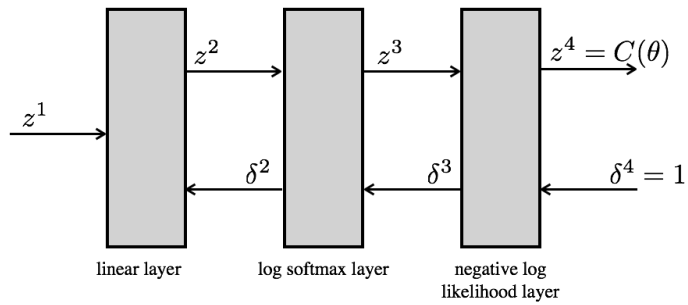
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 =$$



Computing δ^l

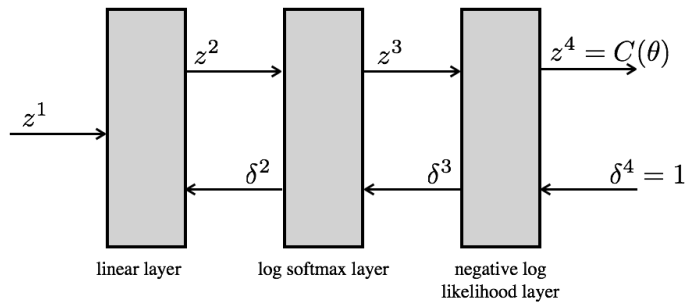
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 = \frac{\partial C(\theta)}{\partial z_2^2}$$



Computing δ^l

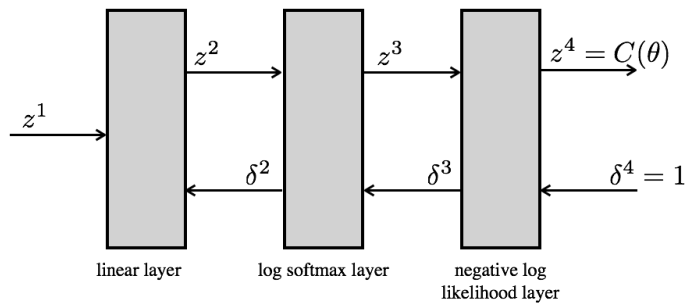
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 = \frac{\partial C(\theta)}{\partial z_2^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_2^2}$$



Computing δ^l

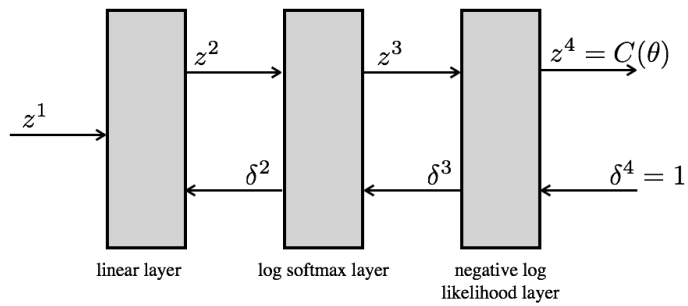
$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

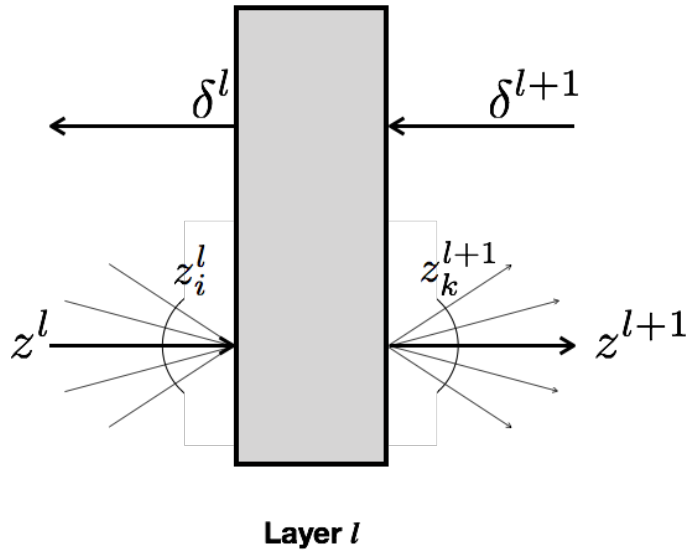
$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 = \frac{\partial C(\theta)}{\partial z_2^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_2^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_2^2}$$



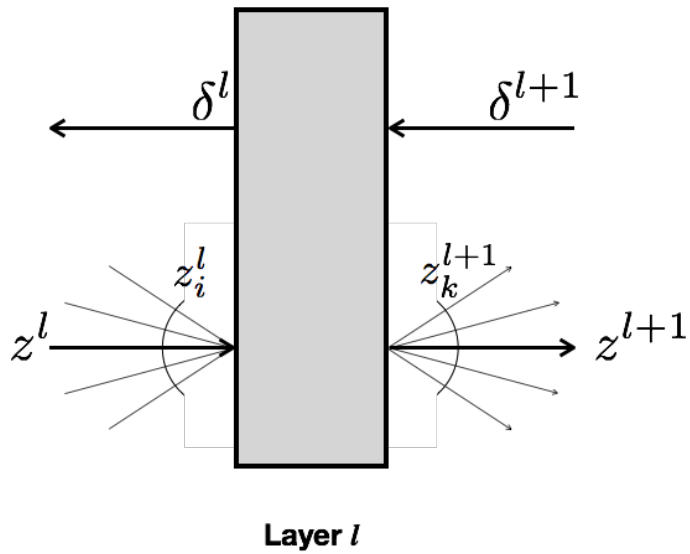
For any differentiable layer l



For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

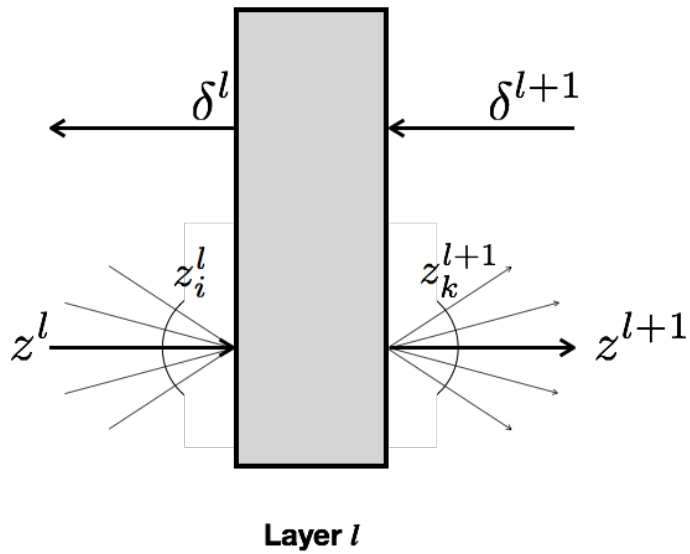
$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



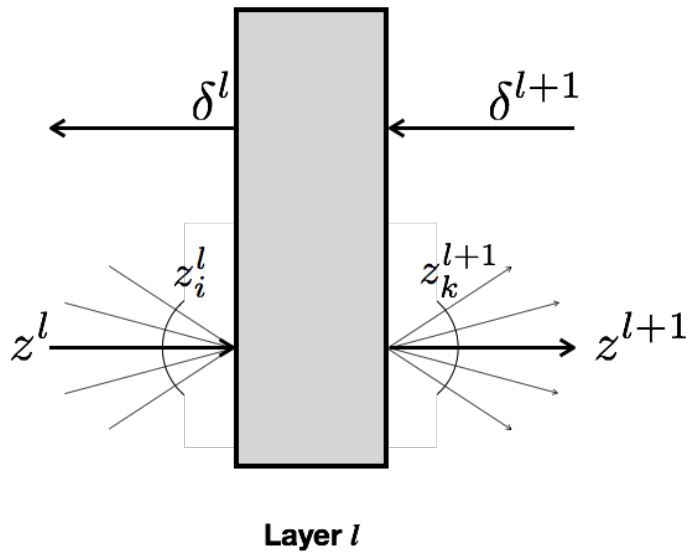
Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} =$$

For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



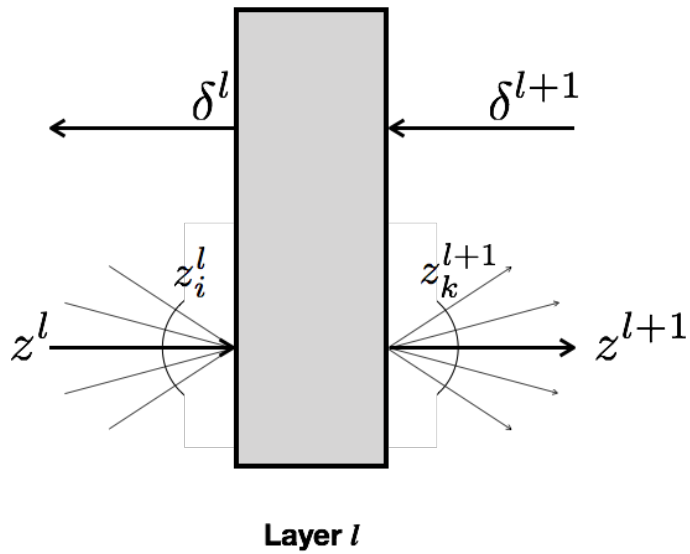
Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} = \sum_k \frac{\partial C(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \theta^l} =$$

For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



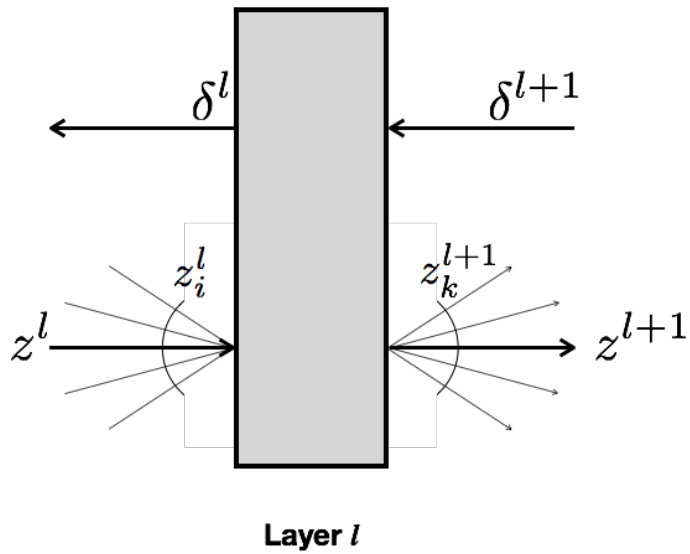
Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} = \sum_k \frac{\partial C(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \theta^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial \theta^l}$$

For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



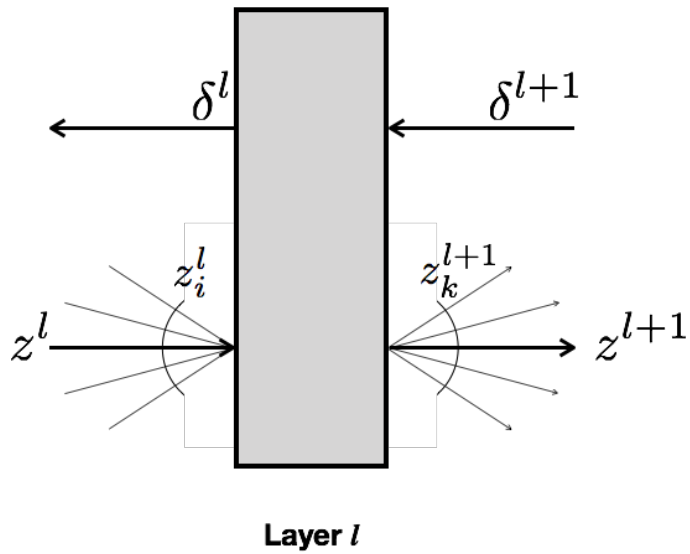
Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} = \sum_k \frac{\partial C(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \theta^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial \theta^l}$$

For any differentiable layer l

For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$



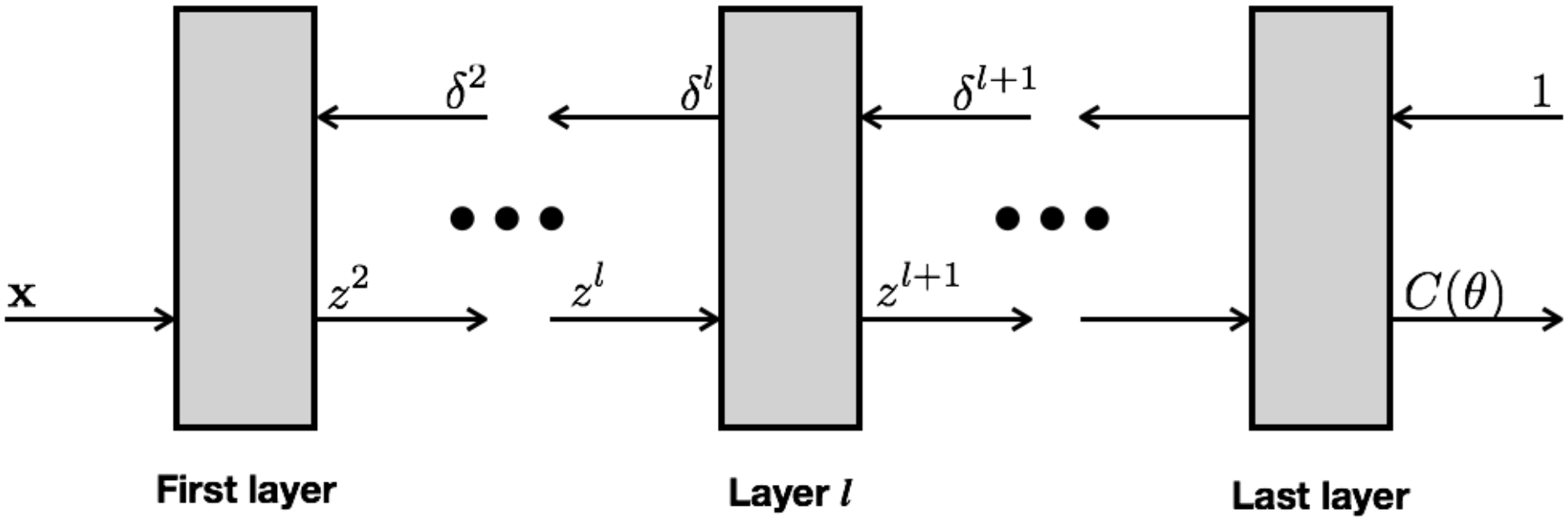
Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} = \sum_k \frac{\partial C(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \theta^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial \theta^l}$$

In our 2-class softmax classifier only layer 1 has parameters (θ_0 and θ_1)

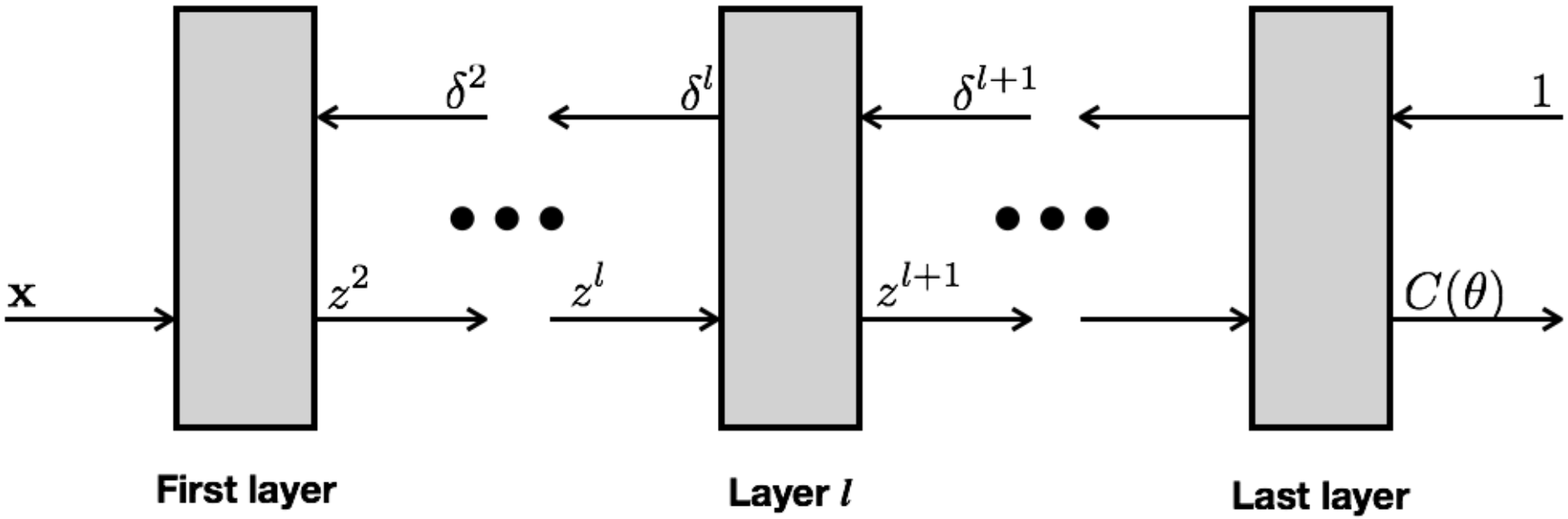
Layered architectures

As long as we have differentiable layers, i.e., we can compute $\frac{\partial z_k^{l+1}}{\partial z_i^l}$, we can use *backpropagation* to compute $\frac{\partial C(\theta)}{\partial \theta^l}$ and use it to update the parameters θ to minimize the cost $C(\theta)$.



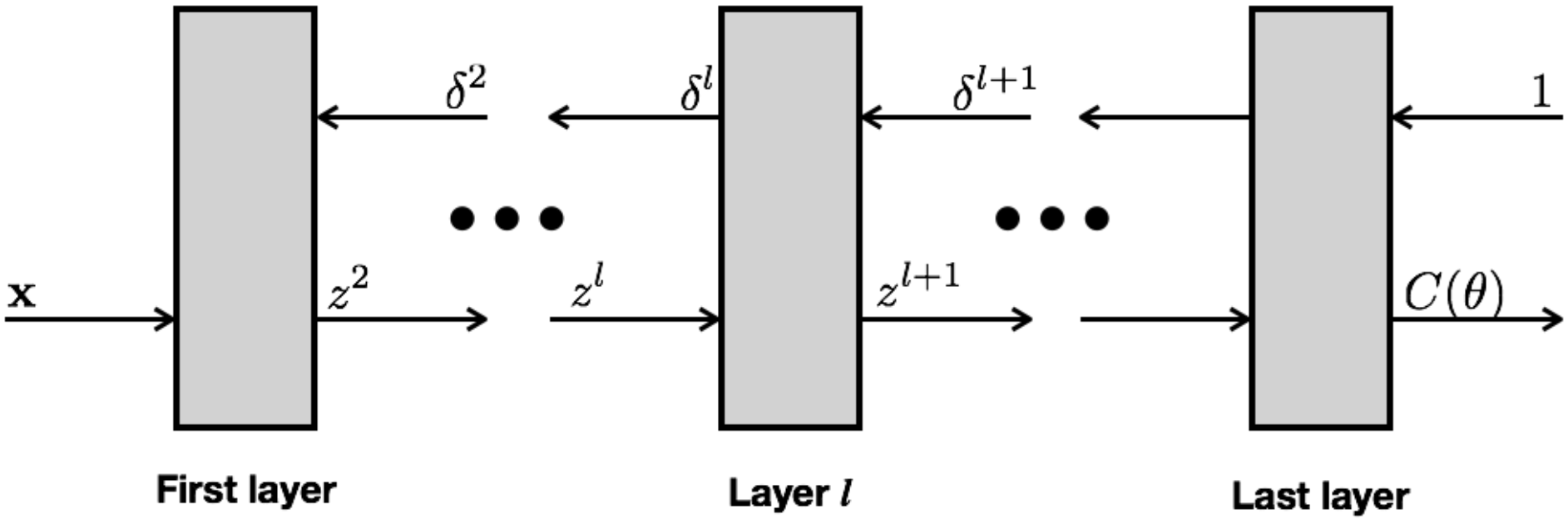
Layered architectures

As long as we have differentiable layers, i.e., we can compute $\frac{\partial z_k^{l+1}}{\partial z_i^l}$, we can use *backpropagation* to compute $\frac{\partial C(\theta)}{\partial \theta^l}$ and use it to update the parameters θ to minimize the cost $C(\theta)$.



Layered architectures

As long as we have differentiable layers, i.e., we can compute $\frac{\partial z_k^{l+1}}{\partial z_i^l}$, we can use *backpropagation* to compute $\frac{\partial C(\theta)}{\partial \theta^l}$ and use it to update the parameters θ to minimize the cost $C(\theta)$.



Backpropagation

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1
- **Backward pass**:

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1
- **Backward pass**:
  - Backpropagate $\delta^L, \delta^{L-1}, \cdots, \delta^2$ (all the way to first layer)

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1
- **Backward pass**:
  - Backpropagate $\delta^L, \delta^{L-1}, \cdots, \delta^2$ (all the way to first layer)
  - Compute $\nabla_\theta C(\theta)$

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1
- **Backward pass**:
  - Backpropagate $\delta^L, \delta^{L-1}, \cdots, \delta^2$ (all the way to first layer)
  - Compute $\nabla_\theta C(\theta)$
- Update θ

Backpropagation

- Set z^1 equal to input $\mathbf{x}$.
- **Forward pass**:
  - Compute $z^2, z^3, \cdots$ layers $1, 2, \cdots$ activations
  - Set δ at the last layer equal to 1
- **Backward pass**:
  - Backpropagate $\delta^L, \delta^{L-1}, \cdots, \delta^2$ (all the way to first layer)
  - Compute $\nabla_\theta C(\theta)$
- Update θ
- Repeat

Layered architecture: consequences

- Compositionality
- Reuse
- Ease of constructing **your own layers**

Layered architecture: consequences

- Compositionality
- Reuse
- Ease of constructing **your own layers**

A neural network can itself be treated as a layer within another neural network (recursion). This allows us to build new neural networks using exisitng (and sometimes pre-trained) models.

This is all good in theory, but what about practice

GPUs

- Support fast vectorized processing

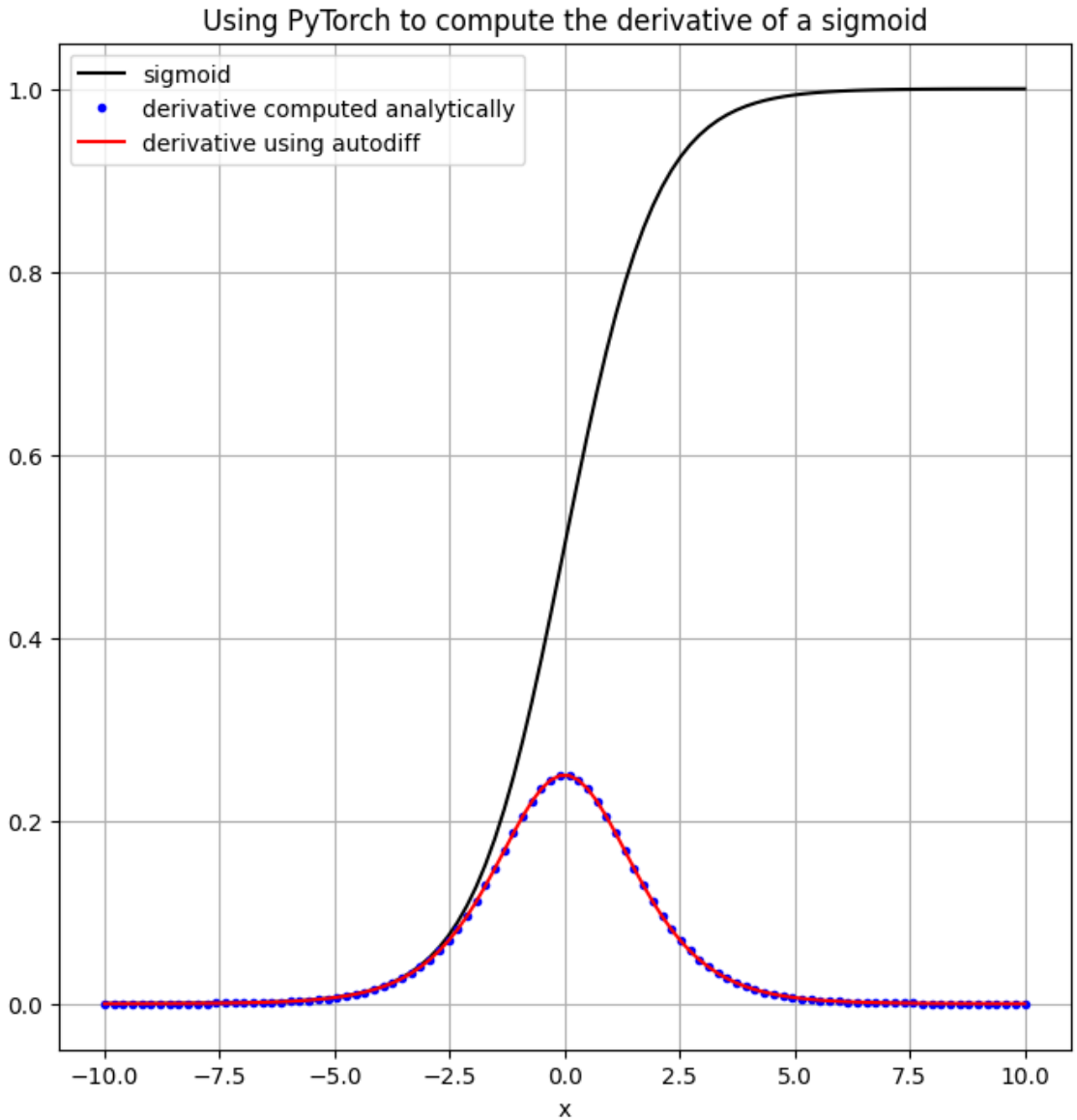
Autodiff

- Techniques to evaluate the derivative of a computer program
- Autodiff example on Google Colab

Autodiff example

```
import torch
import numpy as np

def sigmoid(x):
    return 1. / (1. + torch.exp(-x))

def derivative_of_sigmoid(x):
    "Derivative of a sigmoid (analytical)"
    return sigmoid(x) * (1 - sigmoid(x))

# input
x = torch.linspace(-10,10,100, requires_grad=True)

# derivative of a sigmoid
dx = derivative_of_sigmoid(x)

# PyTorch program that implements sigmoid
z = sigmoid(x)

# using PyTorch autodiff to compute the derivative of the sigmoid
z_ = torch.sum(z)   # because backward can only be called on scalers
z_.backward()       # the backward pass

plt.figure(figsize=(8,8))
plt.title('Using PyTorch to compute the derivative of a sigmoid')
plt.plot(x.detach().numpy(), z.detach().numpy(), 'k', label='sigmoid')
plt.grid()
plt.plot(x.detach().numpy(), dx.detach().numpy(), 'b.', label='derivative computed analytically')
plt.plot(x.detach().numpy(), x.grad.detach().numpy(), 'r', label='derivative using autodiff')
plt.xlabel('x')
plt.legend();
```

Autodiff example



Implication

- It is easy to construct your own layers
  - Provide ways to compute the derivative of the outputs w.r.t. the inputs
  - Provide ways to compute the derivative of the outputs w.r.t. layer parameters
    - Many layers do not have any parameters
- See above: using autodiff, it is possible to simply "program" a layer
  - Derivatives come for free

Regularization for deep networks

Regularization: any modification to reduce generalization error but not the training errors:

- extra constraints and penalties
- prior knowledge

Deep learning is applied to extremely complex tasks. Consequently, regularization is not as simple as controlling the number of parameters

Regularization for deep networks

- Parameter norm penalties
- Data augmentation
  - Fake data
  - Successful in classification/object recognition tasks
- Noise injection
  - Applying random noise to the inputs
  - Applying random noise to hidden layers' inputs
    - Data augmentation at multiple levels of abstraction
  - Data augmentation almost always improves the performance of a neural network
  - Noise added to the weights
    - Recurrent neural networks
    - A practical stochastic implementation of Bayesian inference over weights
  - Noise can also bve added to target outputs

Summary

- Different ways to interpret a neural network
  - Compositions of non-linear functions
  - Computational graphs
  - Comprised of differentiable layers
    - Where possible compose new networks using existing networks

Summary

- Backpropagation: strategy for computing gradients for gradient-based learning
  - Use autodiff to automatixally compute gradients for each layer
    - Vast number of "deep learning" frameworks (e.g., TensorFlow, Theano, PyTorch, etc.); start with those first

Summary

- Controlling model complexity
- Deep learning
  - Loosely speaking, neural networks with several hidden layers
    - Convolutional layers: used for image processing
    - Fully connected layers: often used at the end for regression or classification

Readings

- Ch. 6-9 of Deep Learning by I. Goodfellow et al.

Copyright and License