Lecture 14
Neural Networks: Part II

GEOL 4397: Data analytics and machine learning for geoscientists

Jiajia Sun, Ph.D.

April. 11th, 2019

UNIVERSITY of HOUSTON
YOU ARE THE PRIDE
EARTH AND ATMOSPHERIC SCIENCES

Outline

- Vanishing gradients
  - What causes it?
  - Activation function
  - Xavier and He initialization
  - Batch normalization

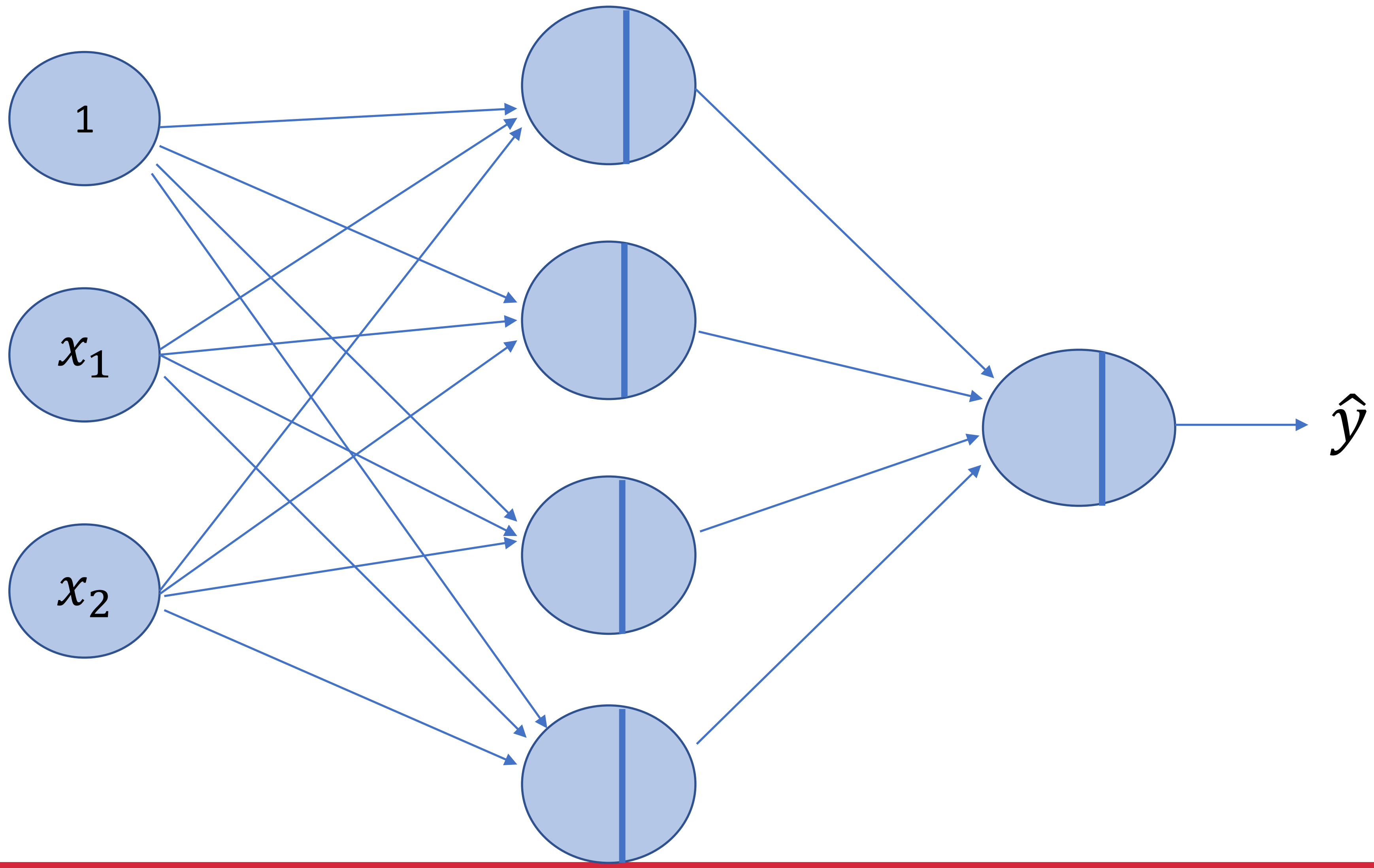
- Optimization algorithms

- Implementing DNN in TensorFlow

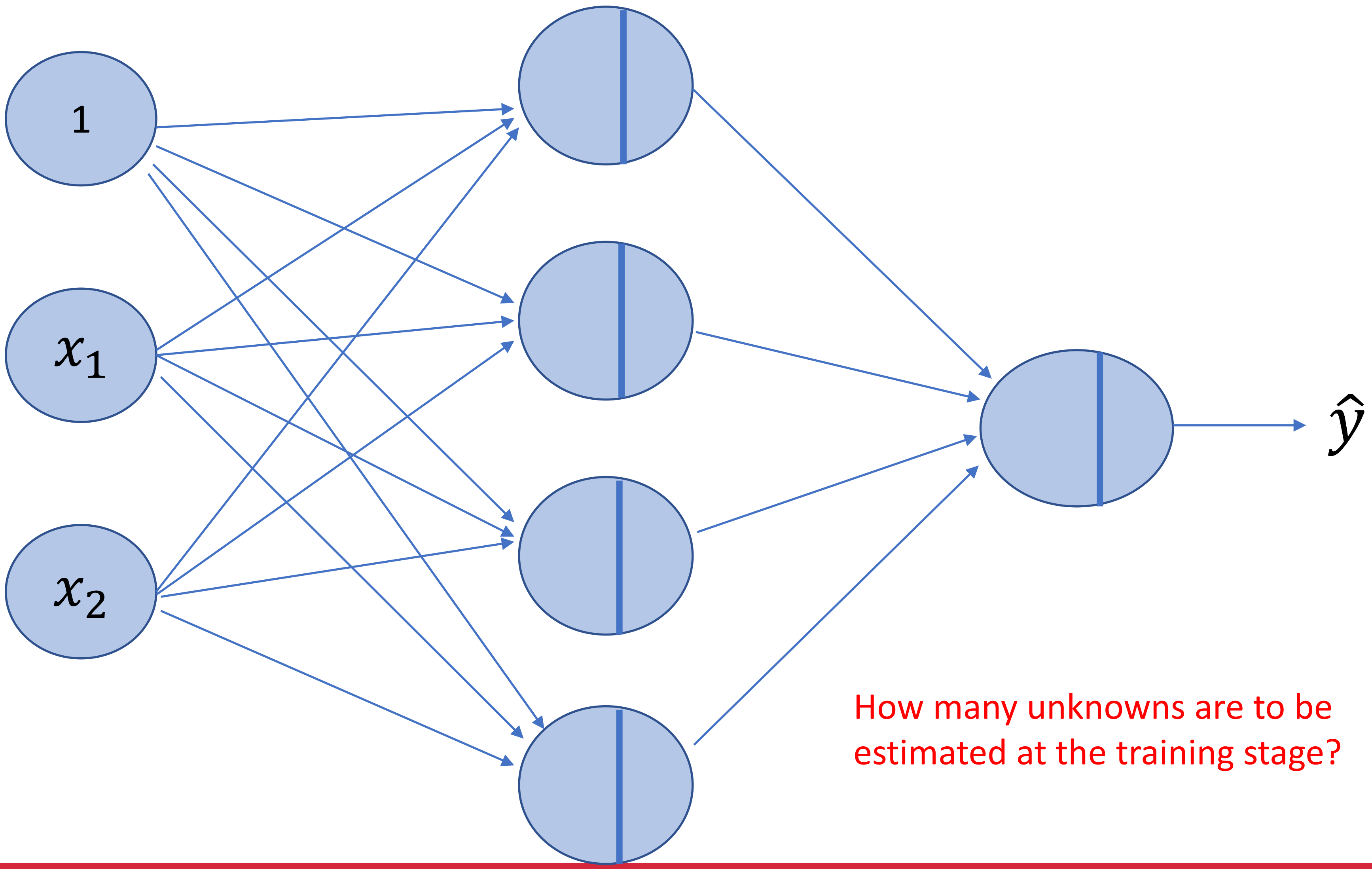
Acknowledgments

- Michael Nielsen for an excellent explanation of the vanishing gradients (http://neuralnetworksanddeeplearning.com/chap5.html)

A shallow neural network

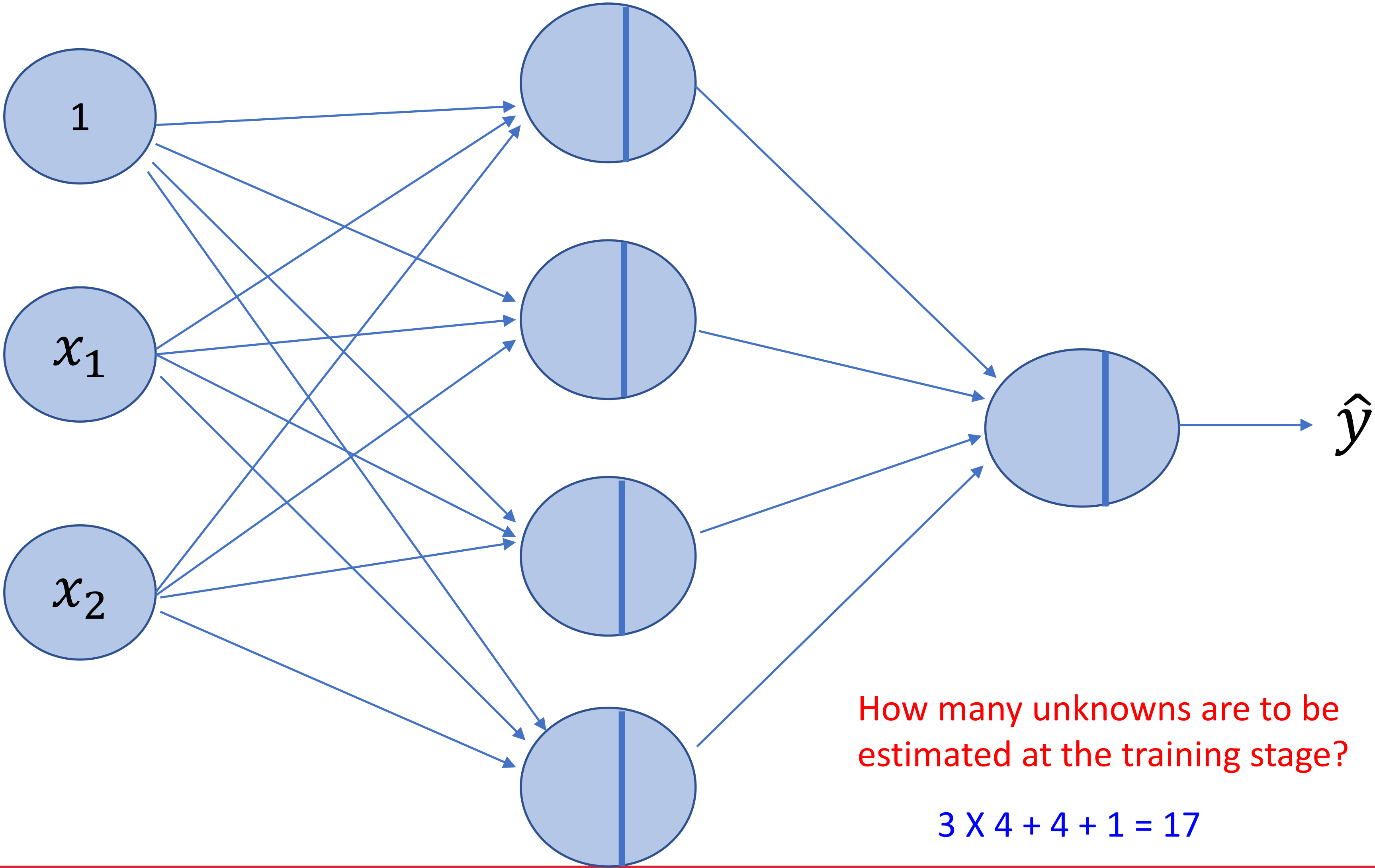


A shallow neural network



How many unknowns are to be estimated at the training stage?

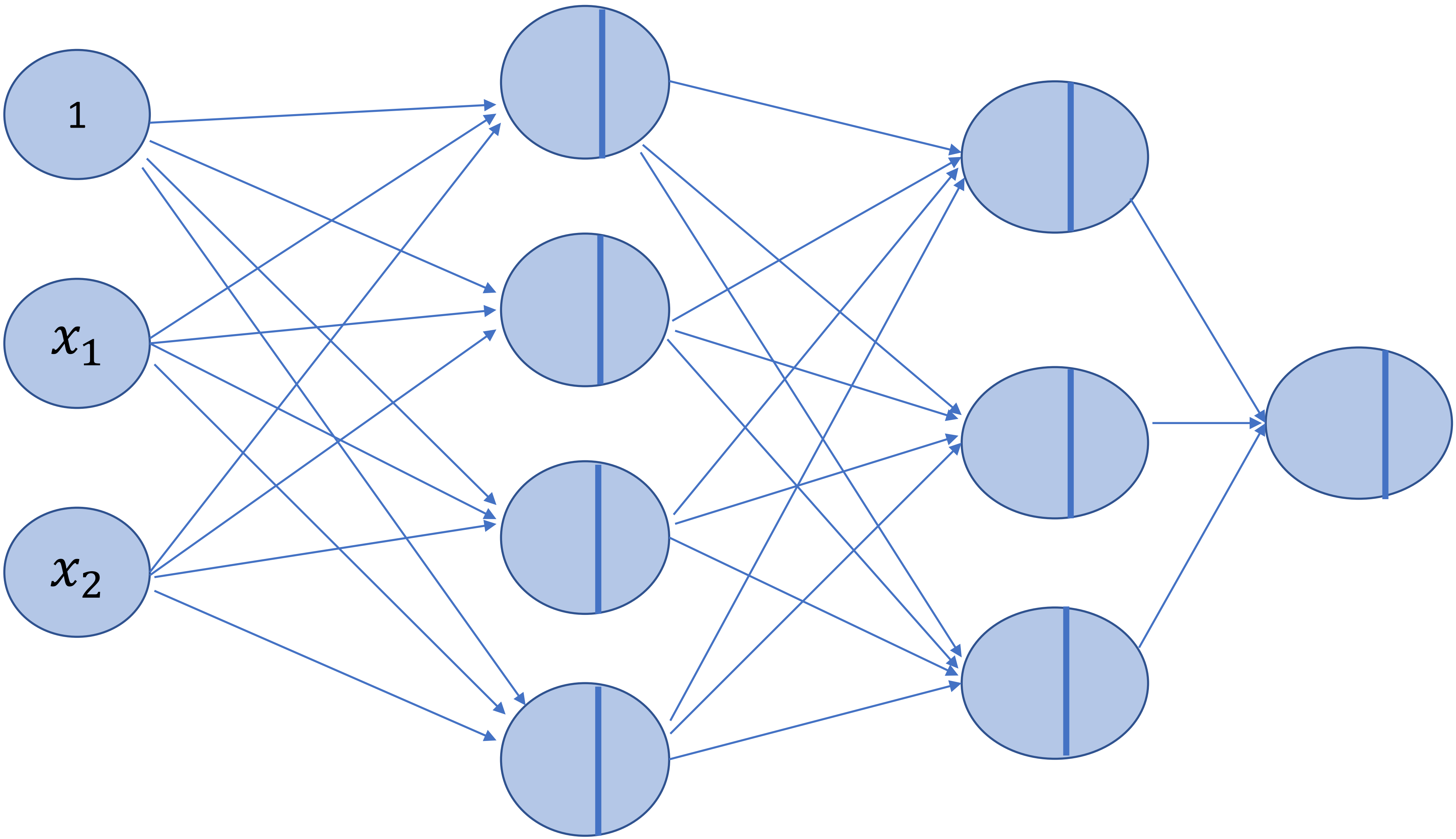
A shallow neural network



How many unknowns are to be estimated at the training stage?

3 X 4 + 4 + 1 = 17

A deep neural network



Another deep neural network

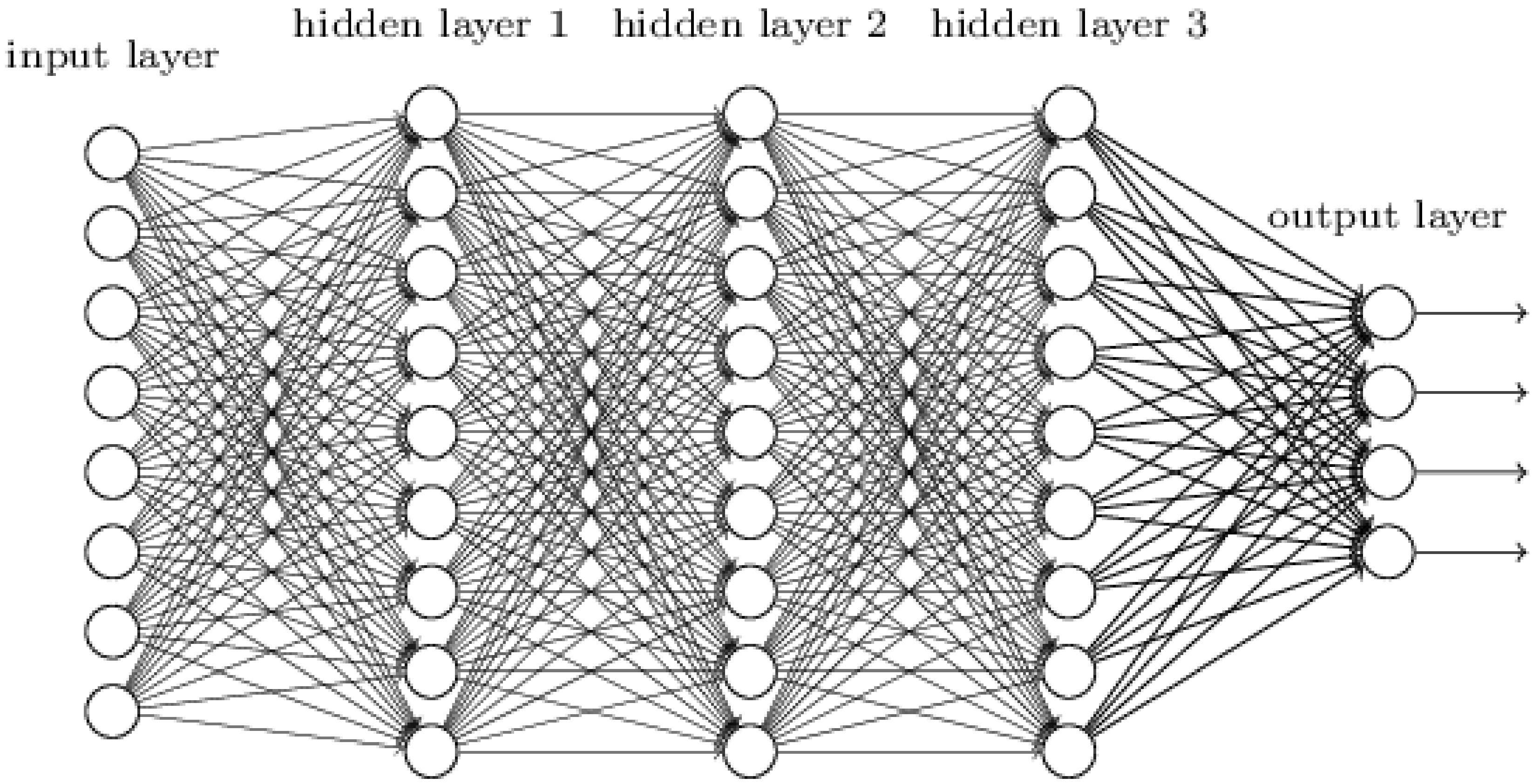
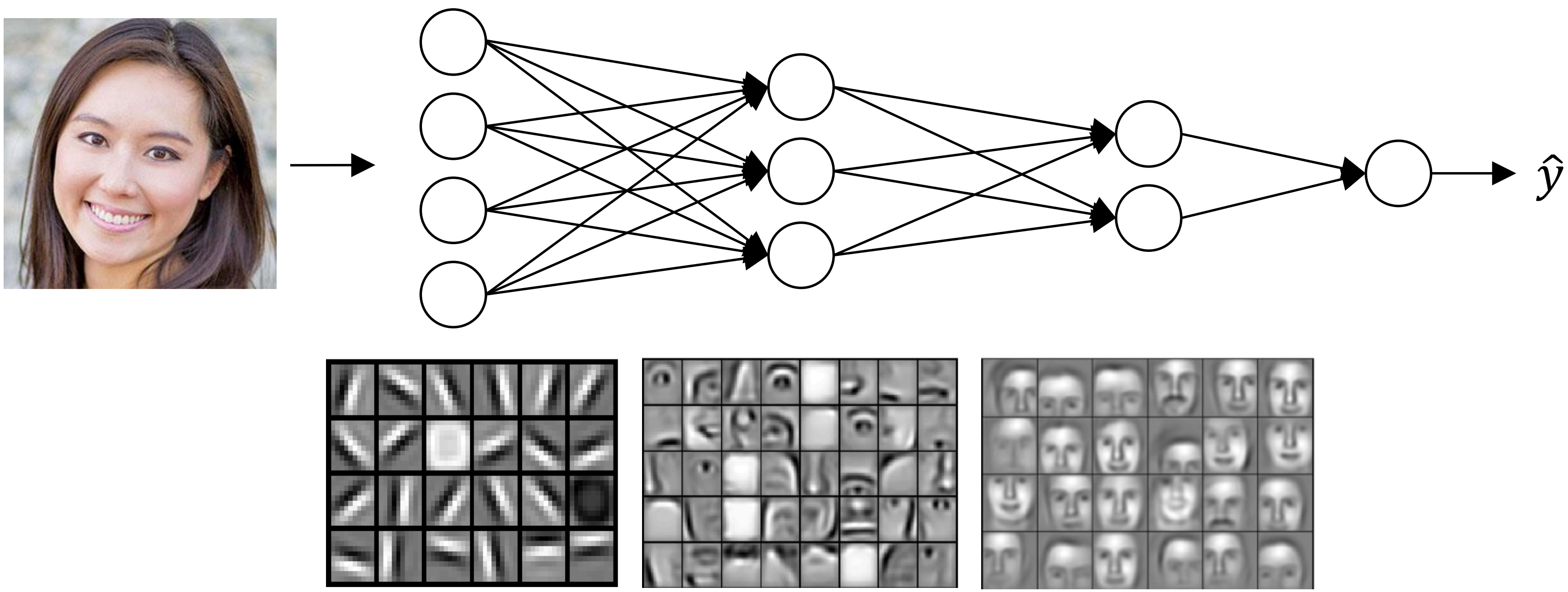


Image source: http://neuralnetworksanddeeplearning.com/chap5.html

Why deep representation?

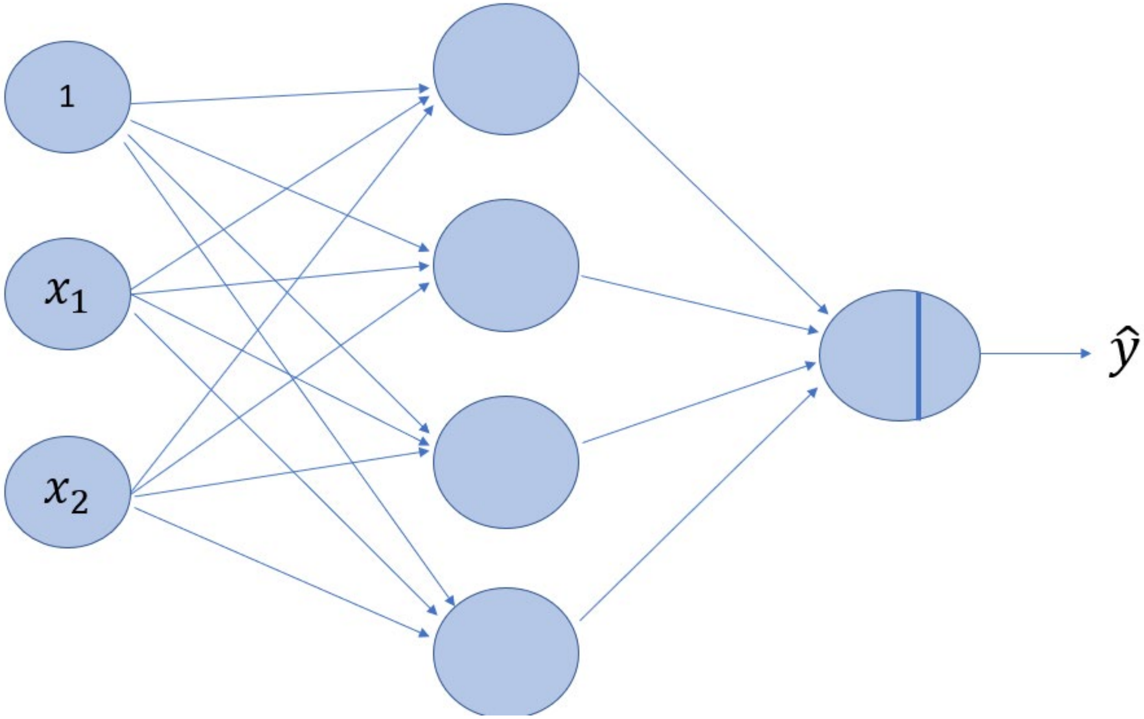


Credit: Andrew Ng

Forward propagation

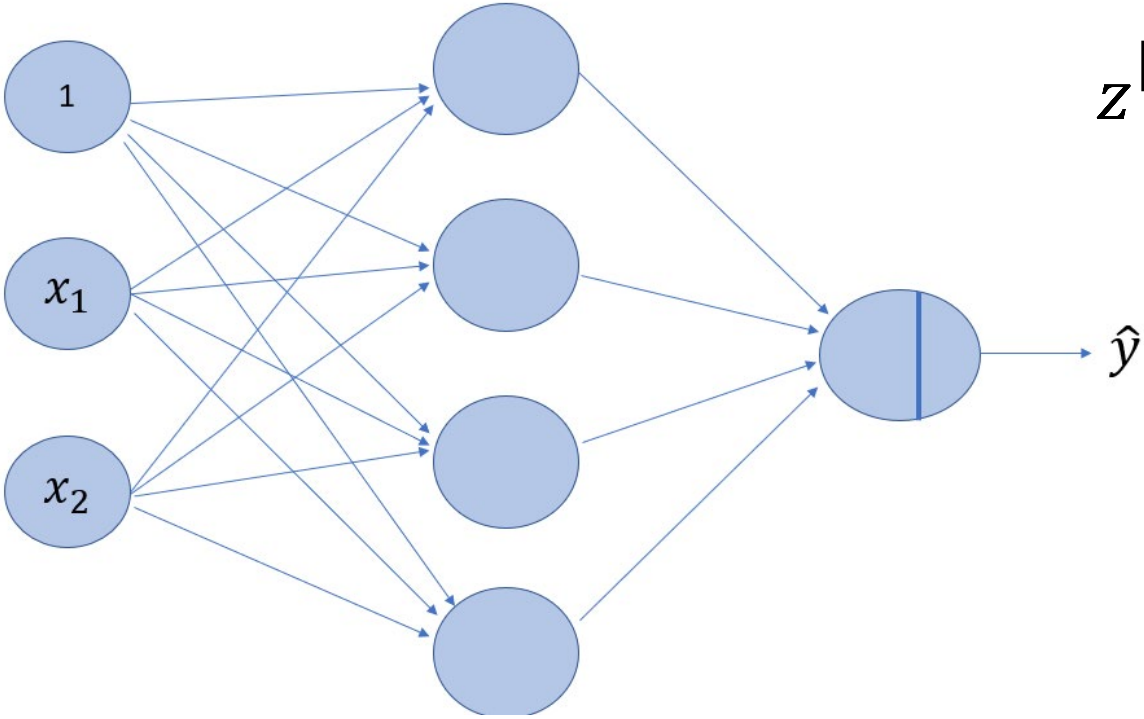
$$z^{[1]} = w^{[1]}x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$



$$L = \frac{1}{2}||a^{[2]} - y||^2$$

A different representation



$$z^{[1]} = f_1(w^{[1]}, b^{[1]}, x)$$
$$a^{[1]} = g_1(z^{[1]})$$

$$z^{[2]} = f_2(w^{[2]}, b^{[2]}, a^{[1]})$$
$$a^{[2]} = g_2(z^{[2]})$$
$$L = l(a^{[2]})$$

Forward Propagation

$$
\left.\begin{array}{l} x \\ w^{[1]} \\ b^{[1]} \end{array}\right\} \rightarrow z^{[1]} = f_1(w^{[1]}, b^{[1]}) \rightarrow a^{[1]} = g_1(z^{[1]}) \rightarrow z^{[2]} = f_2(w^{[2]}, a^{[1]}, b^{[2]}) \rightarrow a^{[2]} = g_2(z^{[2]}) \rightarrow L(a^{[2]})
$$

Back Propagation

$$\frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial g_2}\frac{\partial g_2}{\partial f_2}\frac{\partial f_2}{\partial g_1}\frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial w^{[1]}}$$

x, $w^{[1]}$, $b^{[1]}$

$$z^{[1]} = f_1(w^{[1]}, b^{[1]}) \longrightarrow a^{[1]} = g_1(z^{[1]}) \longrightarrow z^{[2]} = f_2(w^{[2]}, a^{[1]}, b^{[2]}) \longrightarrow a^{[2]} = g_2(z^{[2]}) \longrightarrow L(a^{[2]})$$

Gradient descent (for a 2-layer NN)

- Initialize $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$
- While (not converge):
-     compute $\frac{\partial L}{\partial w^{[1]}}, \frac{\partial L}{\partial b^{[1]}}, \frac{\partial L}{\partial w^{[2]}}, \frac{\partial L}{\partial b^{[2]}}$
-     $w^{[1]} = w^{[1]}\text{-}\alpha \frac{\partial L}{\partial w^{[1]}}$
-     $b^{[1]} = b^{[1]}\text{-}\alpha \frac{\partial L}{\partial b^{[1]}}$
-     $w^{[2]} = w^{[2]}\text{-}\alpha \frac{\partial L}{\partial w^{[2]}}$
-     $b^{[2]} = b^{[2]}\text{-}\alpha \frac{\partial L}{\partial b^{[2]}}$
- end

- Backpropagation works its way from the output layer to the input layer
- Computes the error gradients on the way

Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 275-276

Vanishing gradients

- Backpropagation works its way from the output layer to the input layer
- Computes the error gradients on the way
- Unfortunately, gradients often get smaller and smaller as we move backward through the layers

Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 275-276

Vanishing gradients

- Backpropagation works its way from the output layer to the input layer
- Computes the error gradients on the way
- Unfortunately, gradients often get smaller and smaller as we move backward through the layers
- This leaves the weights associated with shallower layers virtually unchanged, and makes learning at those layers much slower than deeper layers.

Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 275-276

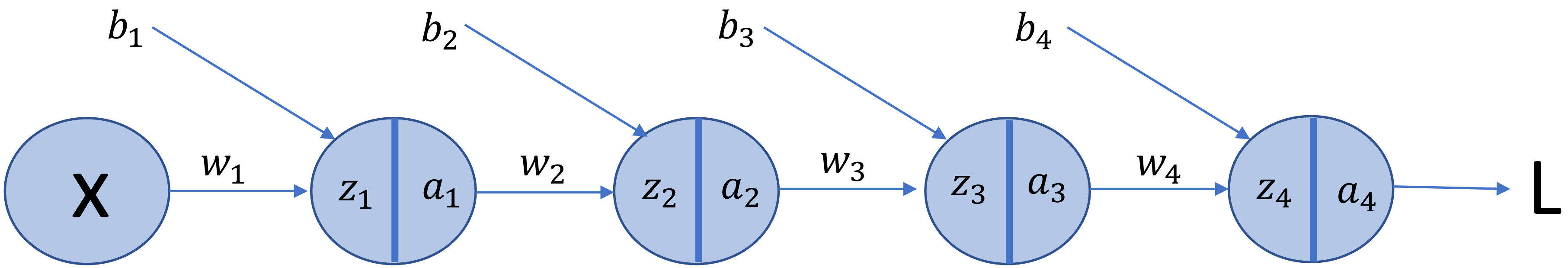
Vanishing gradients

- Backpropagation works its way from the output layer to the input layer
- Computes the error gradients on the way
- Unfortunately, gradients often get smaller and smaller as we move backward through the layers
- This leaves the weights associated with shallower layers virtually unchanged, and makes learning at those layers much slower than deeper layers.
- This was a barrier to training deep NN for a long time.

Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 275-276

Understanding vanishing gradients

- Let us consider the simplest deep neural network

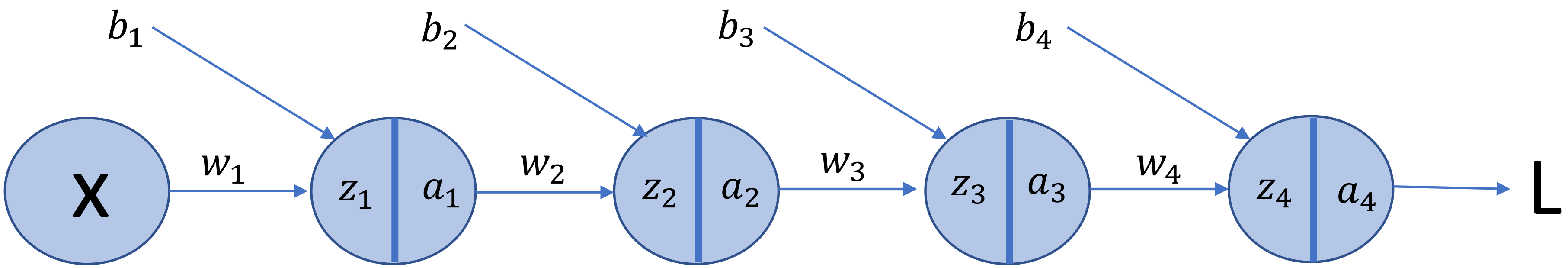


$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j)$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



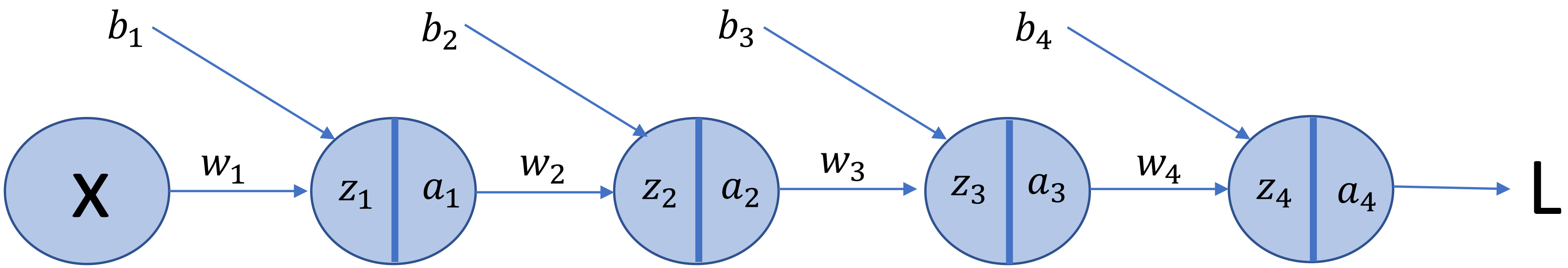
$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j)$$

Let us take a closer look at the gradient $\frac{\partial L}{\partial b_1}$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



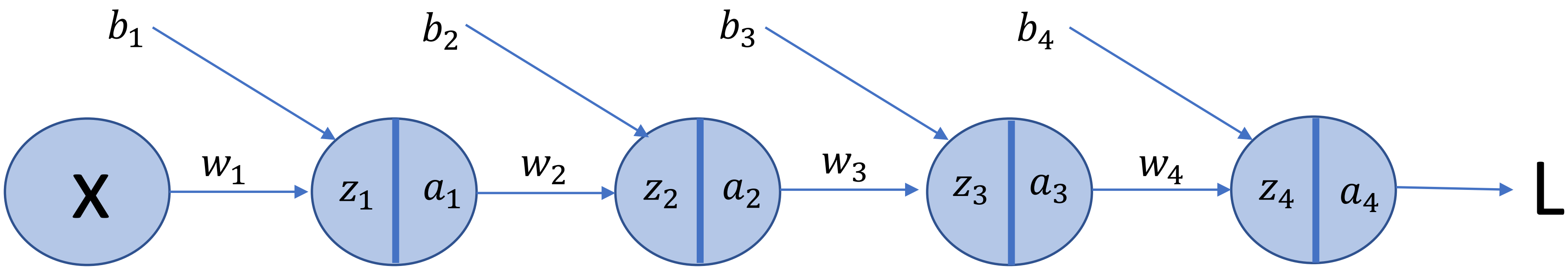
$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



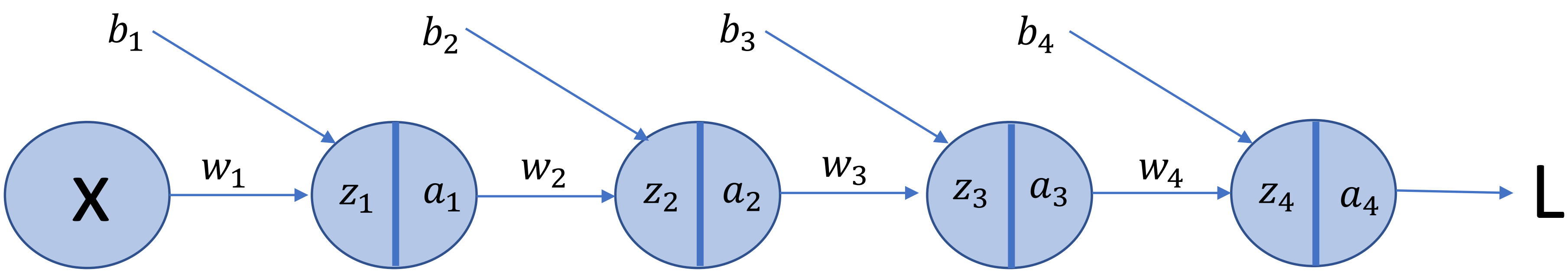
$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



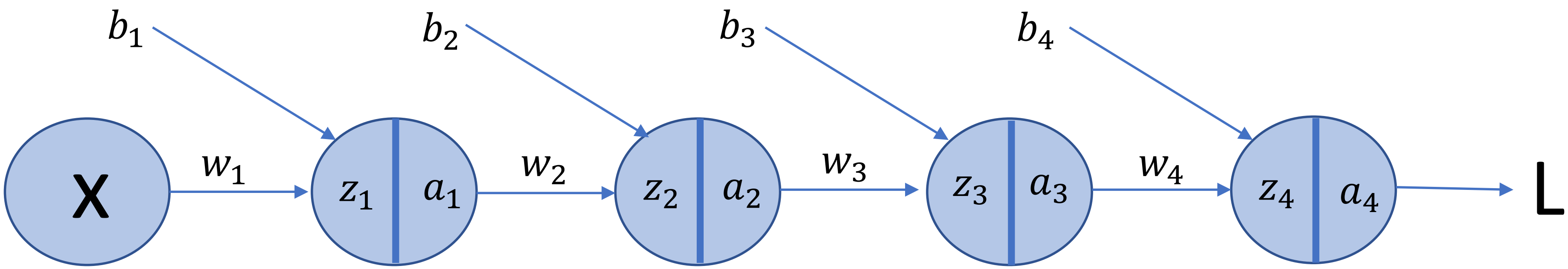
$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j) \longrightarrow \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



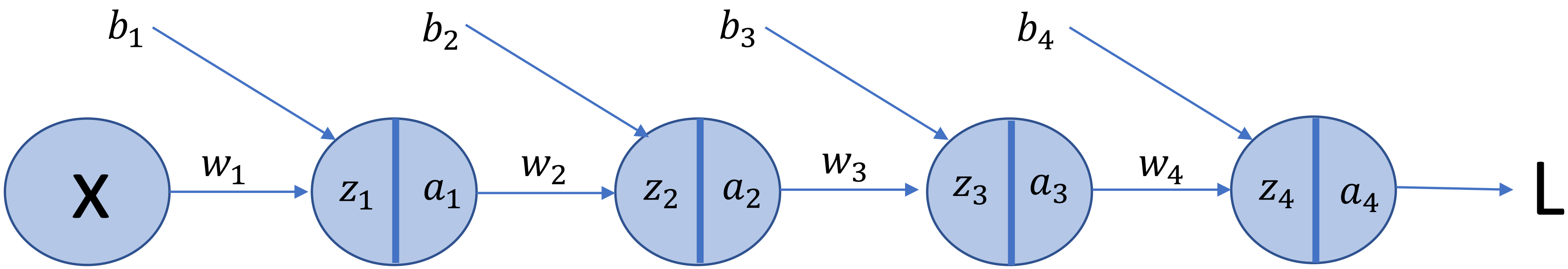
$$z_j = w_j a_{j-1} + b_j$$

$$a_j = g(z_j) \longrightarrow \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



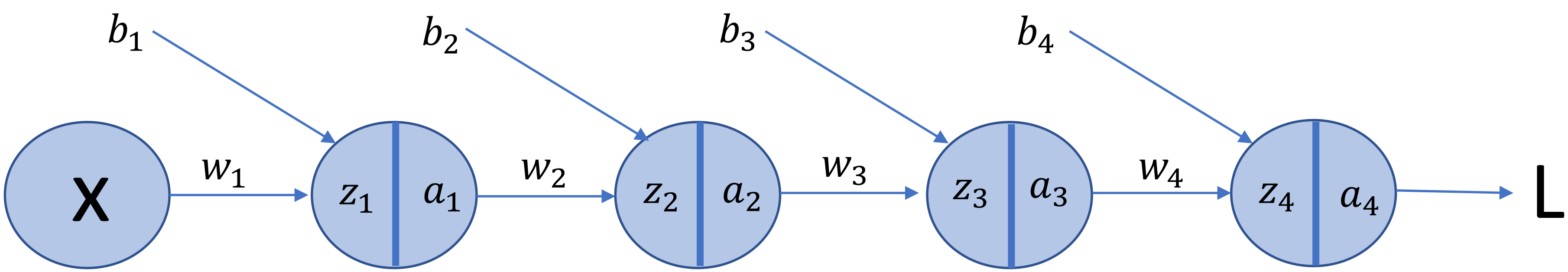
$$z_j = w_j a_{j-1} + b_j \Rightarrow \frac{\partial z_j}{\partial a_{j-1}} = w_j$$

$$a_j = g(z_j) \Rightarrow \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



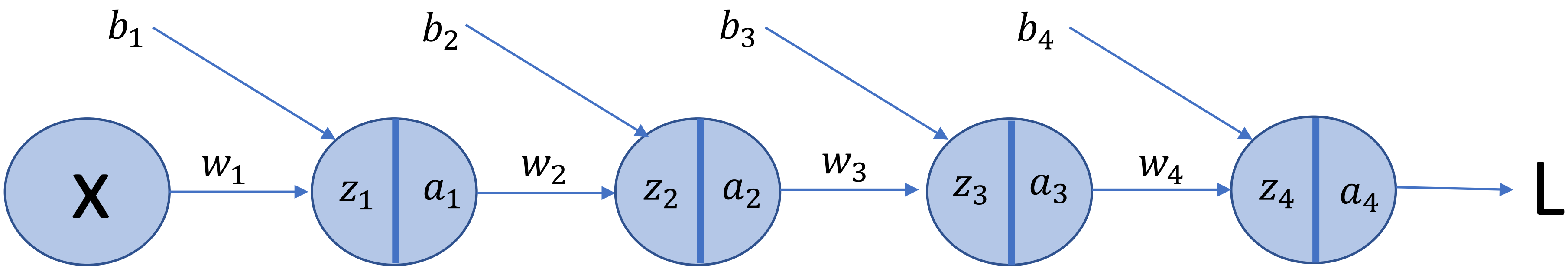
$$z_j = w_j a_{j-1} + b_j \Longrightarrow \frac{\partial z_j}{\partial a_{j-1}} = w_j$$

$$a_j = g(z_j) \Longrightarrow \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



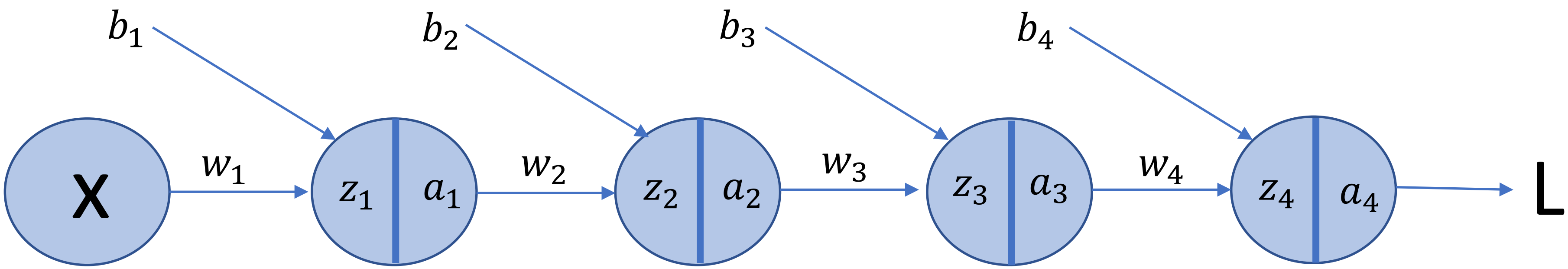
$$z_j = w_j a_{j-1} + b_j \Rightarrow \frac{\partial z_j}{\partial a_{j-1}} = w_j \qquad \frac{\partial z_1}{\partial b_1} = 1$$

$$a_j = g(z_j) \Rightarrow \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times \frac{\partial a_4}{\partial z_4} \times \frac{\partial z_4}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial a_2} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1}$$

Understanding vanishing gradients

- Let us consider the simplest deep neural network



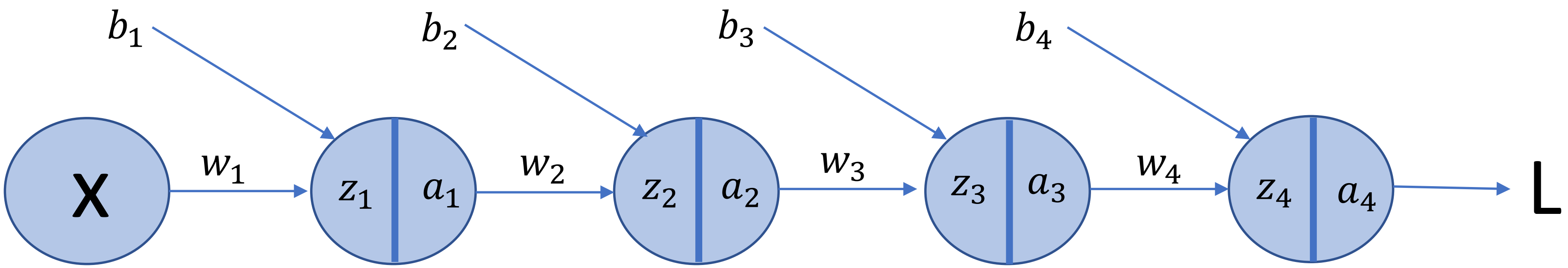
$$z_j = w_j a_{j-1} + b_j \quad \Rightarrow \quad \frac{\partial z_j}{\partial a_{j-1}} = w_j \qquad \frac{\partial z_1}{\partial b_1} = 1$$

$$a_j = g(z_j) \quad \Rightarrow \quad \frac{\partial a_j}{\partial z_j} = g'(z_j)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times g'(z_4) \times w_4 \times g'(z_3) \times w_3 \times g'(z_2) \times w_2 \times g'(z_1)$$

Understanding vanishing gradients

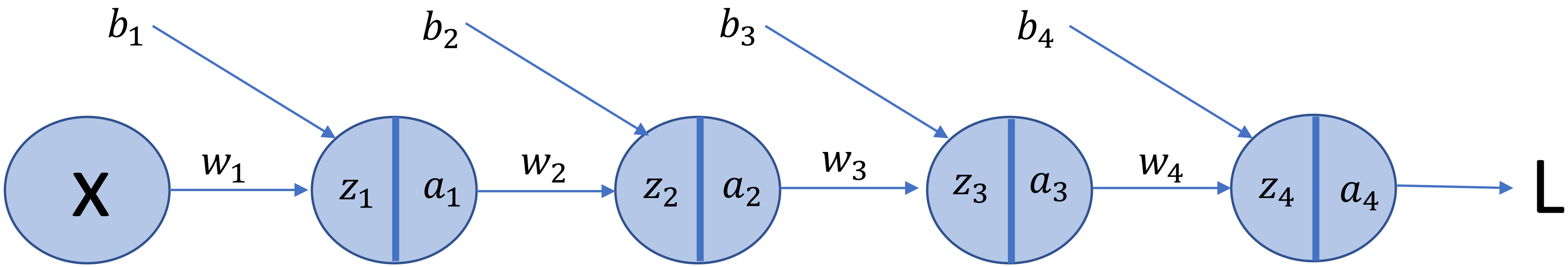
- Let us consider the simplest deep neural network



$\frac{\partial L}{\partial b_1}$ measures how much change would happen to the cost function if b_1 changes.

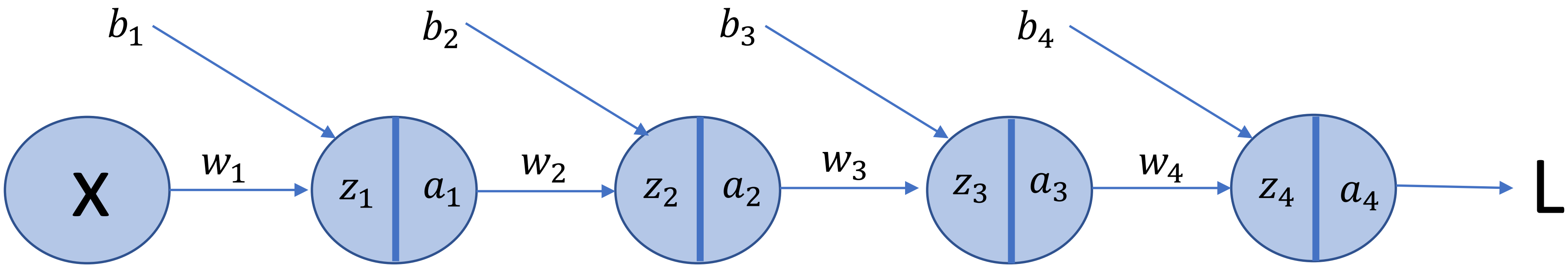
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \times g'(z_4) \times w_4 \times g'(z_3) \times w_3 \times g'(z_2) \times w_2 \times g'(z_1)$$

Understanding vanishing gradients



$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4} \boxed{g'(z_3)\, w_3} \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

Understanding vanishing gradients



$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4} \boxed{g'(z_3)\, w_3} \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

Remember $g(z) = \frac{1}{1+e^{-z}}$

$$|g'(z_j)| \le \frac{1}{4}$$

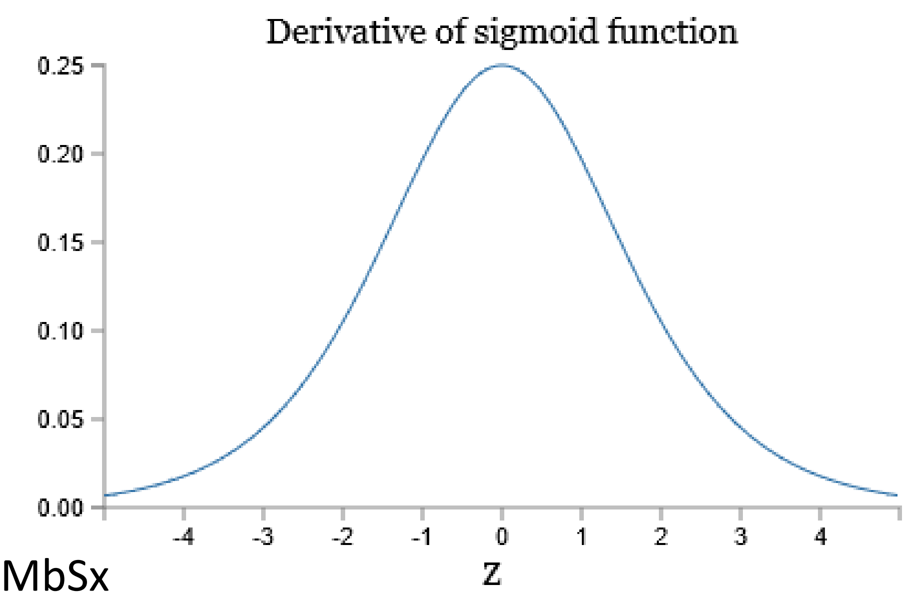
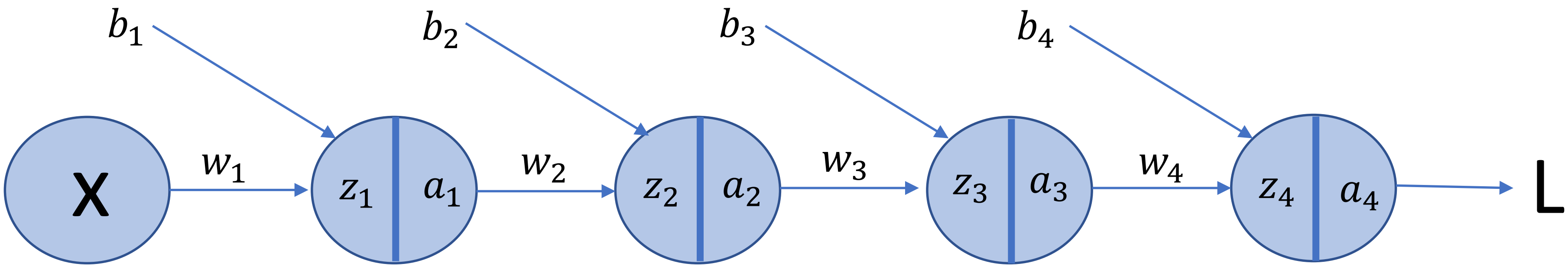


Image source: https://goo.gl/s3MbSx

Understanding vanishing gradients



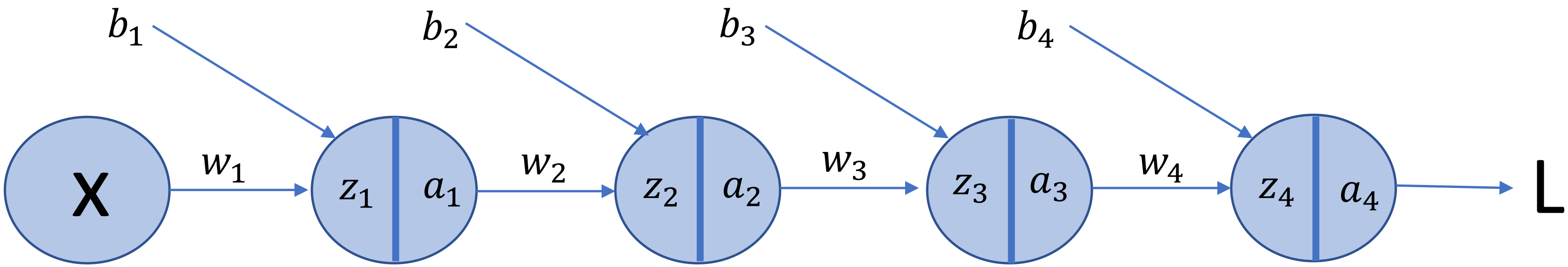
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4} \boxed{g'(z_3)\, w_3} \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

Let us assume that the weights are randomly initialized using a Gaussian with mean 0 and standard deviation of 1. (very popular)

So, weights will usually satisfy $|w_j| < 1$

Therefore, $|g'(z_j)w_j| < \frac{1}{4}$

Understanding vanishing gradients



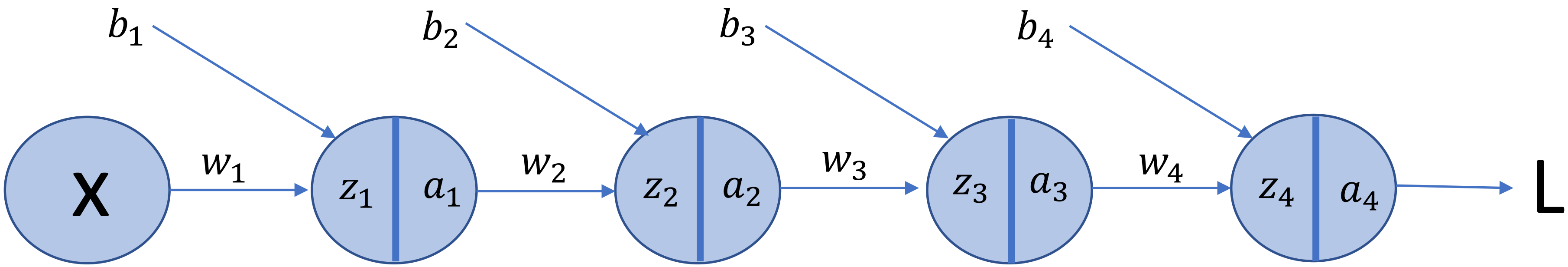
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4} \boxed{g'(z_3)\, w_3} \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

When we take a product of many such terms, the product will exponentially decrease: the more terms, the smaller the product.

$\frac{\partial L}{\partial b_1}$ becomes very very small for deep neural networks.

Therefore, $\left|g'(z_j)w_j\right| < \frac{1}{4}$

Understanding vanishing gradients



$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4}\, \boxed{g'(z_3)\, w_3}\, \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

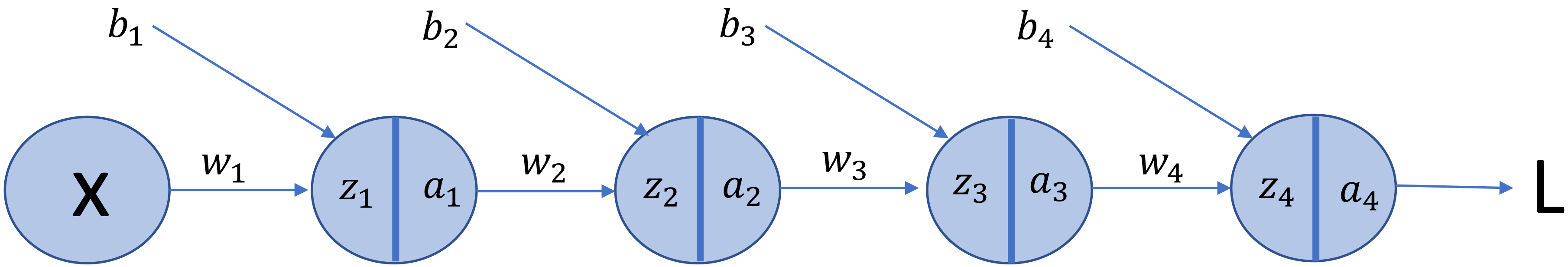
When we take a product of many such terms, the product will exponentially decrease: the more terms, the smaller the product.

$\frac{\partial L}{\partial b_1}$ becomes very very small for deep neural networks.

Therefore, $|g'(z_j)w_j| < \frac{1}{4}$

Vanishing gradients!!!

Understanding vanishing gradients



$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial a_4} \boxed{g'(z_4)\, w_4} \boxed{g'(z_3)\, w_3} \boxed{g'(z_2)\, w_2}\, g'(z_1)$$

<u>Vanishing gradients!!!</u>

Different layers in the network will learn at vastly different speeds.

Note

- The problem has been empirically observed for quite a while
- It was one of the reasons why deep NNs were mostly abandoned for a long time
- In 2010, a paper titled 'Understanding the difficulty of training deep feedforward neural networks' by Xavier Glorot and Yoshua Bengio shed some lights on it.

Observations

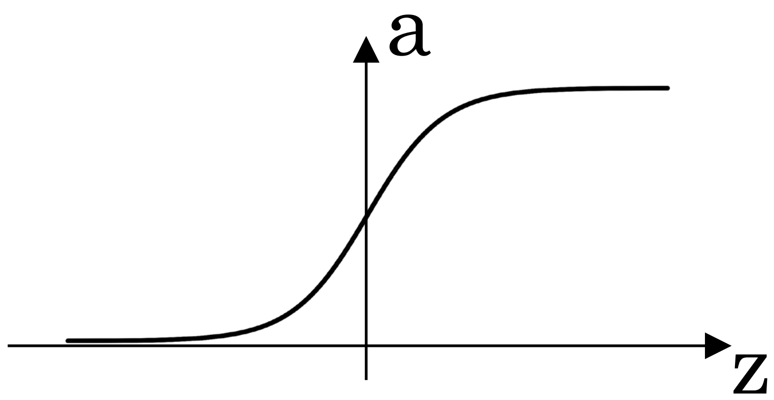
- It is the combination of logistic activation function and the weight initialization strategy that causes the vanishing gradient problem. (Glorot and Bengio, 2010).

Two strategies

- Use a different activation function
- Use a different initialization scheme

Activation function

- Sigmoid function

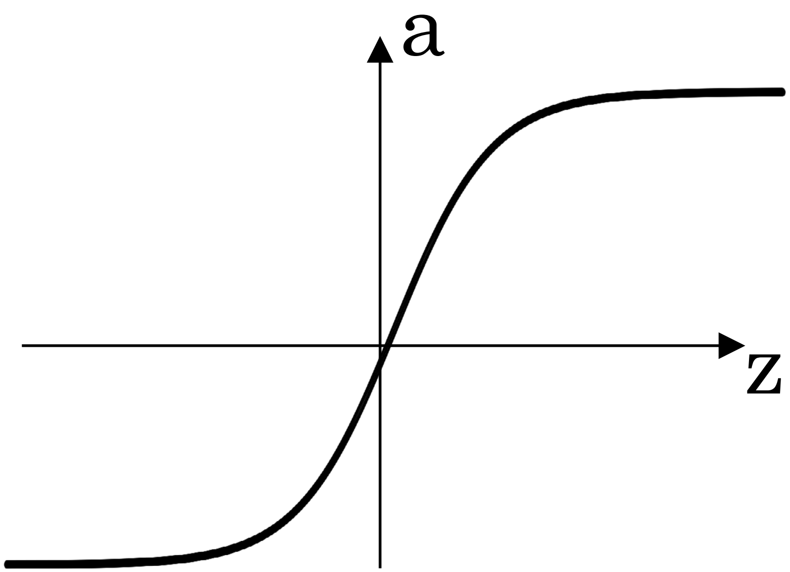


$$g(z) = \frac{1}{1 + e^{-z}}$$

Image credit: Andrew Ng

Activation function

- Tanh (hyperbolic tangent) function

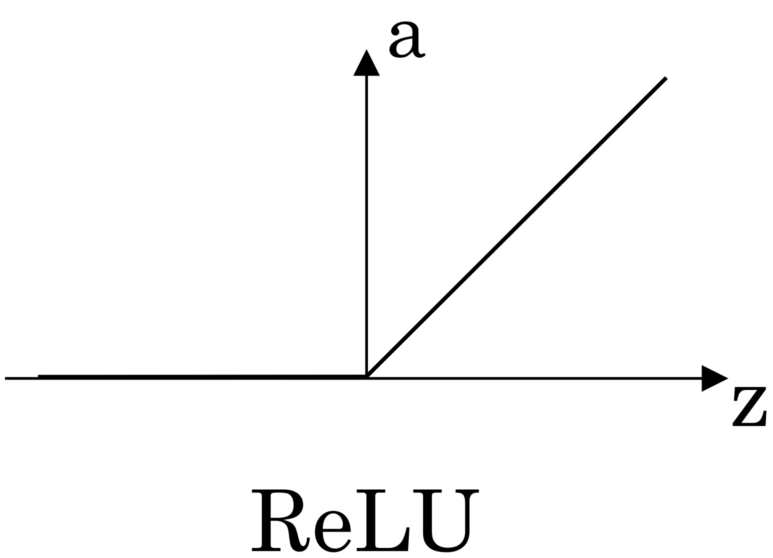


$$g(z) = \tanh(z)$$

Image credit: Andrew Ng

Activation function

- ReLU (rectified linear units) function



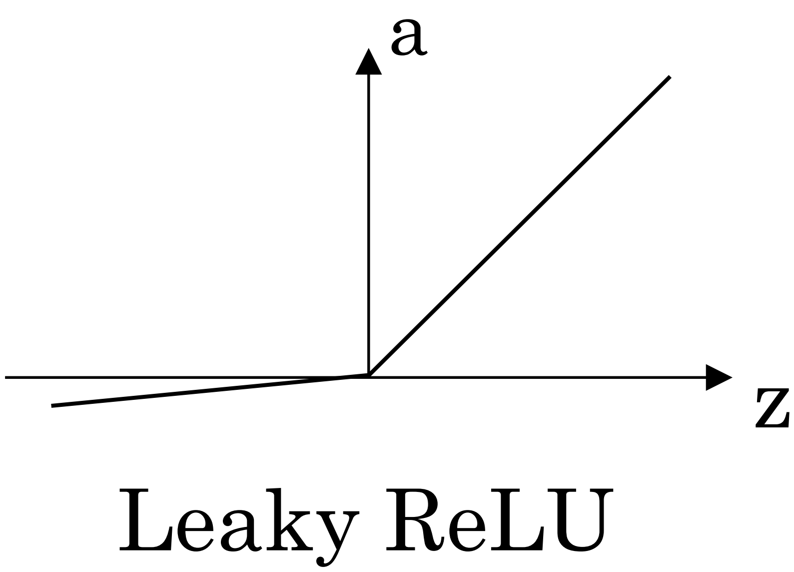
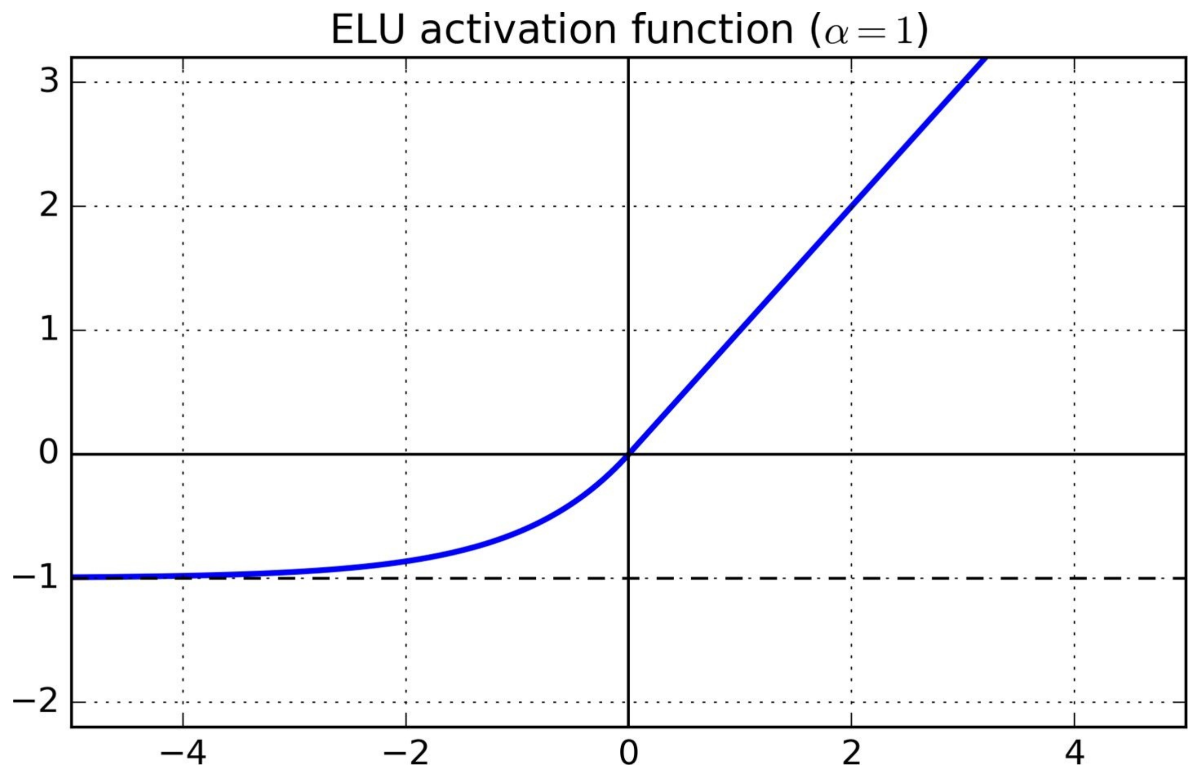


Image credit: Andrew Ng

Activation function

- ELU (exponential linear unit) function



Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 280

Xavier and He initialization

Table 11-1. Initialization parameters for each type of activation function

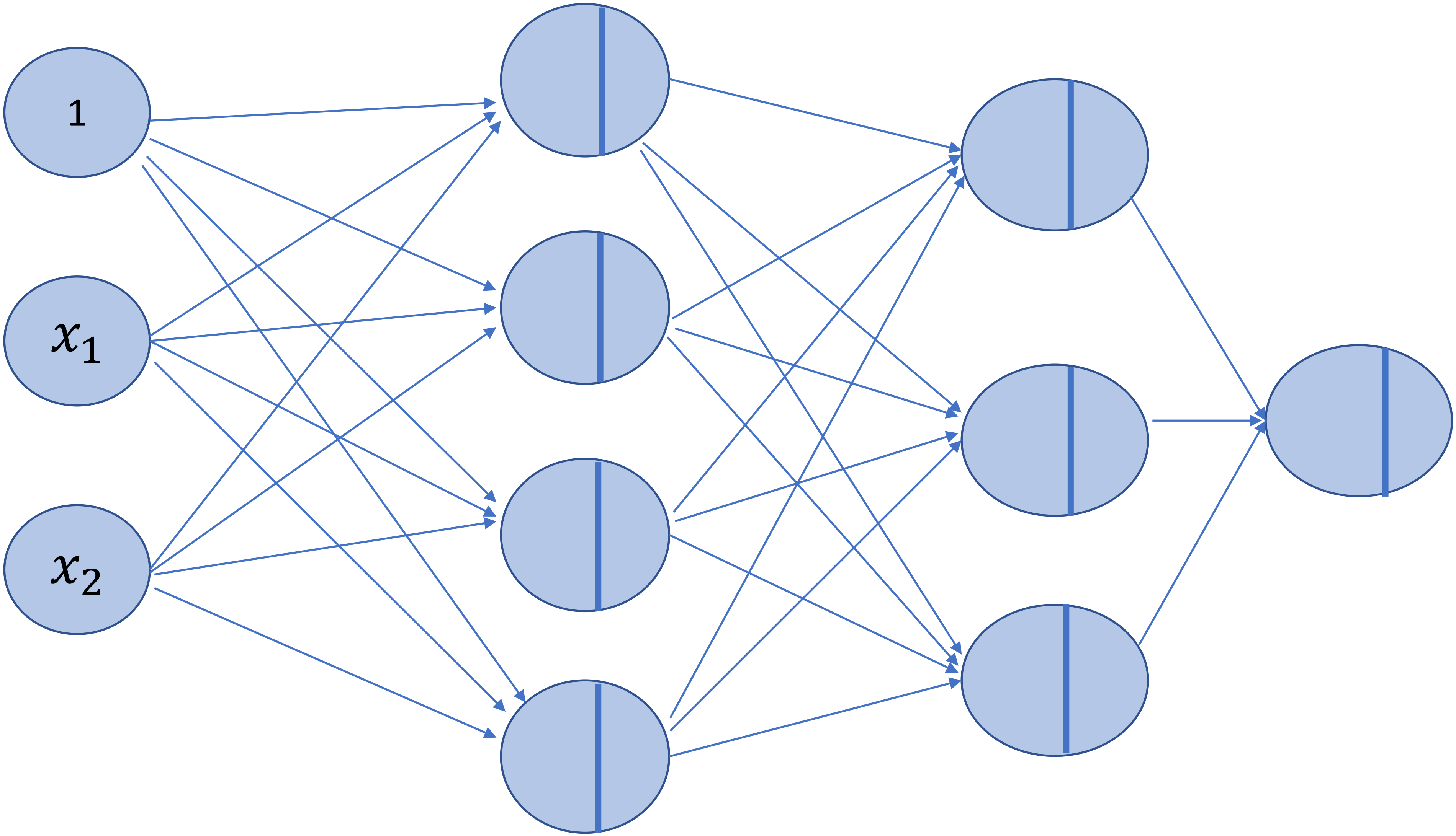
Activation function	Uniform distribution [–r, r]	Normal distribution
Xavier initialization (2010) Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
He initialization (2015) ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Aurelien Geron, 2017, Hands-on Machine Learning with Scikit-Learn & TensorFlow, pp 278

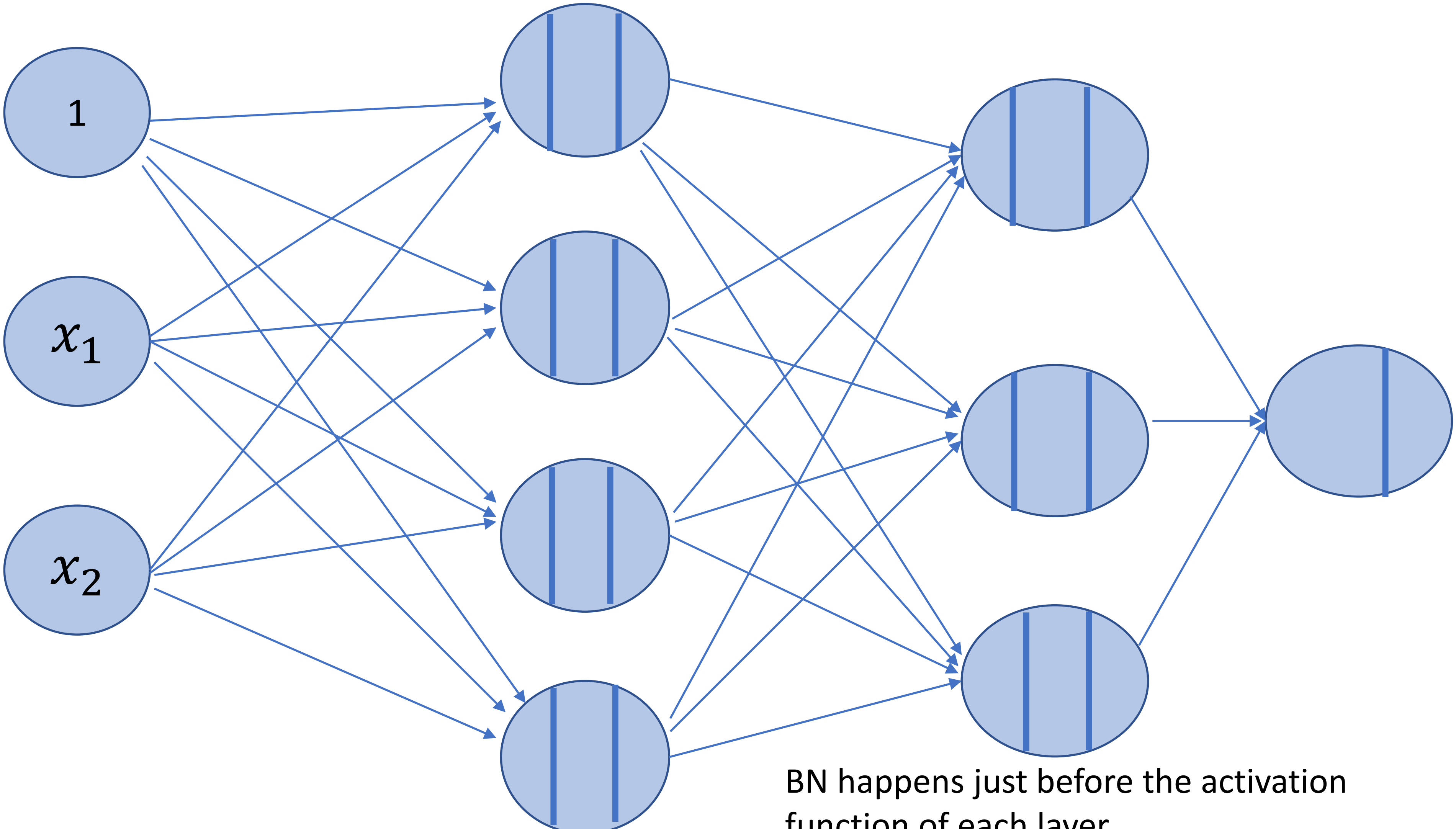
Batch normalization

- Another way of dealing with the vanishing gradient problem
- Proposed by Sergey Ioffe and Christian Szegedy in 2015.

A deep neural network w/o BN



A deep neural network w/ BN



BN happens just before the activation function of each layer

Batch normalization (Ioffe and Szegedy, 2015)

- Let us consider jth layer, suppose there are m neurons in this layers
- $\mu = \frac{1}{m}\sum_{i=1}^{m} z_i^{[j]}$
- $\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(z_i^{[j]} - \mu)^2}$
- $\hat{z}_i^{[j]} = \frac{z_i^{[j]}-\mu}{\sqrt{\sigma^2+\varepsilon}}$
- $\tilde{z}_i^{[j]} = \gamma\hat{z}_i^{[j]} + \beta$

γ and β are learned during training.

Optimization algorithms

- Mini-batch gradient descent
  - tf.train.GradientDescentOptimizer()
- Momentum Optimization
  - tf.train.MomentumOptimizer()
- RMSProp
  - tf.train.RMSPropOptimizer()
- Adam Optimization
  - tf.train.AdamOptimizer()

https://www.tensorflow.org/api_guides/python/train

https://www.tensorflow.org/api_guides/python/train

TensorFlow ™ Install Develop API r1.7 Deploy Extend Community Versions TFRC

Regression Examples
Tensor Handle Operations
Sparse Tensors
Spectral Functions
Variables
Strings
Summary Operations
Testing
TensorFlow Debugger
Threading and Queues
Training
Overview r1.7
tf
tf.app
tf.bitwise
tf.compat
tf.contrib
tf.data
tf.distributions
tf.errors
tf.estimator
tf.feature_column
tf.gfile
tf.graph_util
tf.image
tf.initializers
tf.keras
tf.layers
tf.linalg

Training

`tf.train` provides a set of classes and functions that help train models.

Optimizers

The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.

You never instantiate the Optimizer class itself, but instead instantiate one of the subclasses.

- `tf.train.Optimizer`
- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`

See `tf.contrib.opt` for more optimizers.

Implementing DNN in TensorFlow