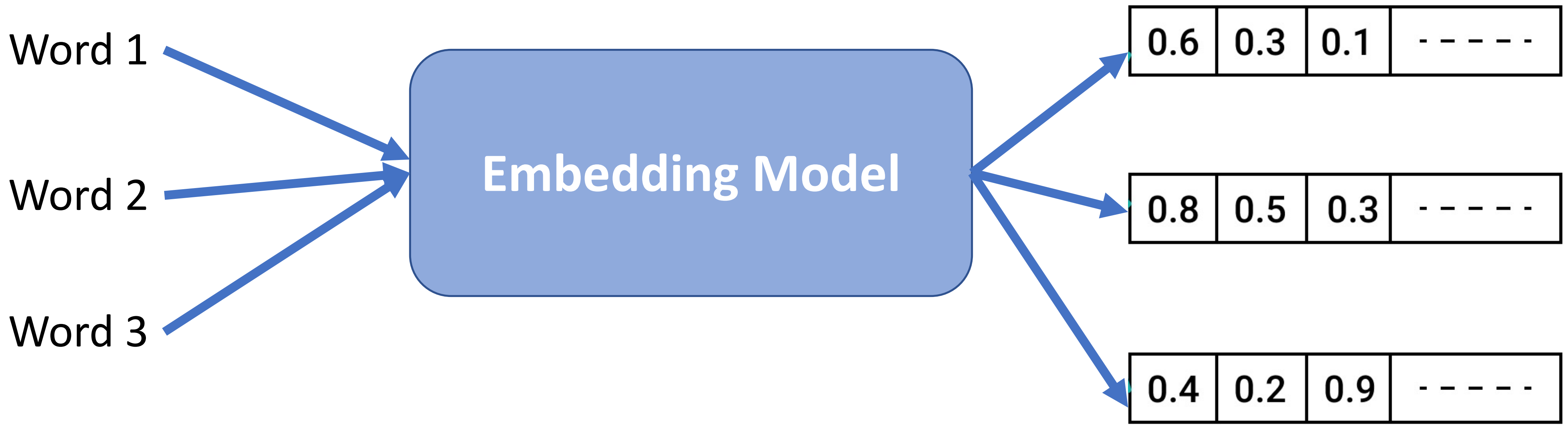
Topic 3.5
Crash course in neural networks

Our Goal

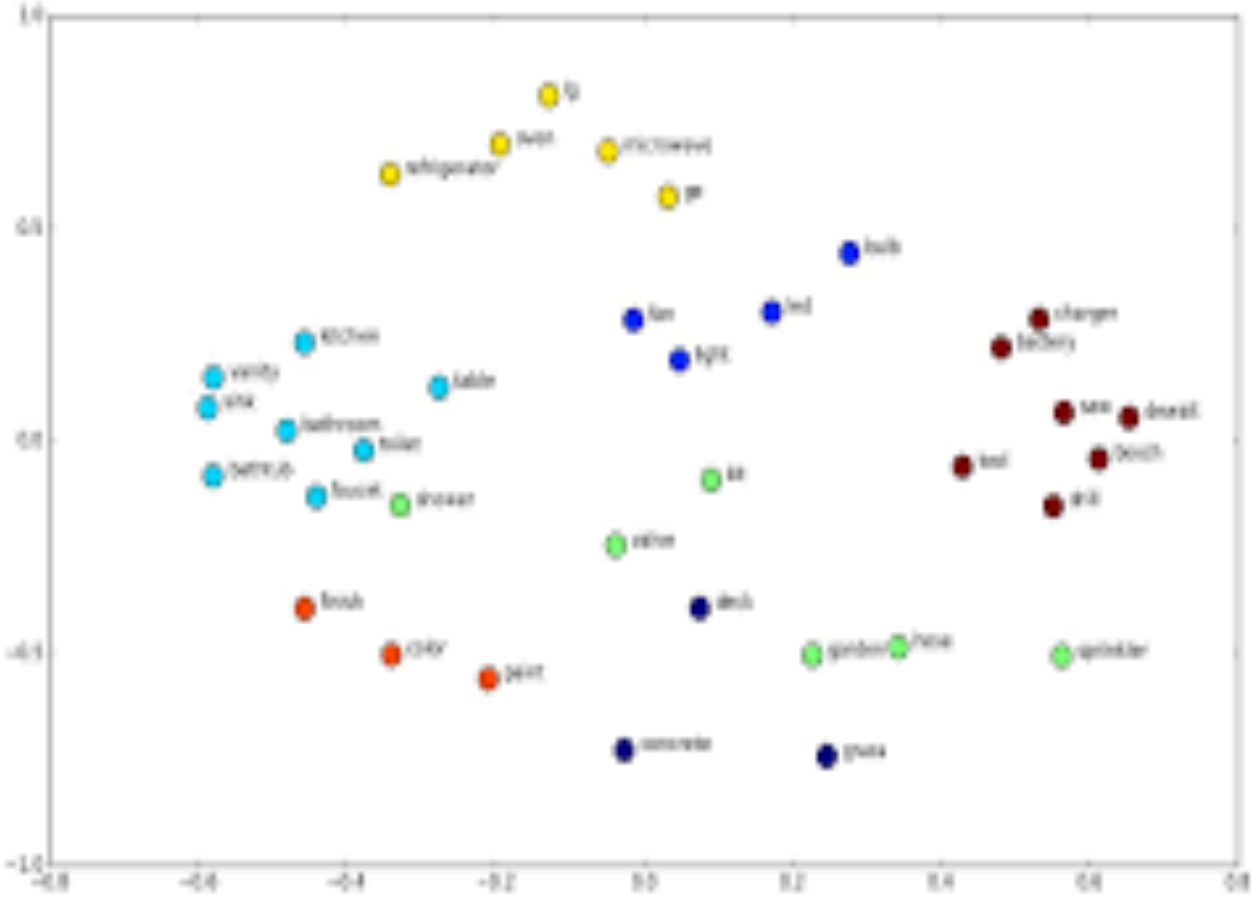


Embedding Space

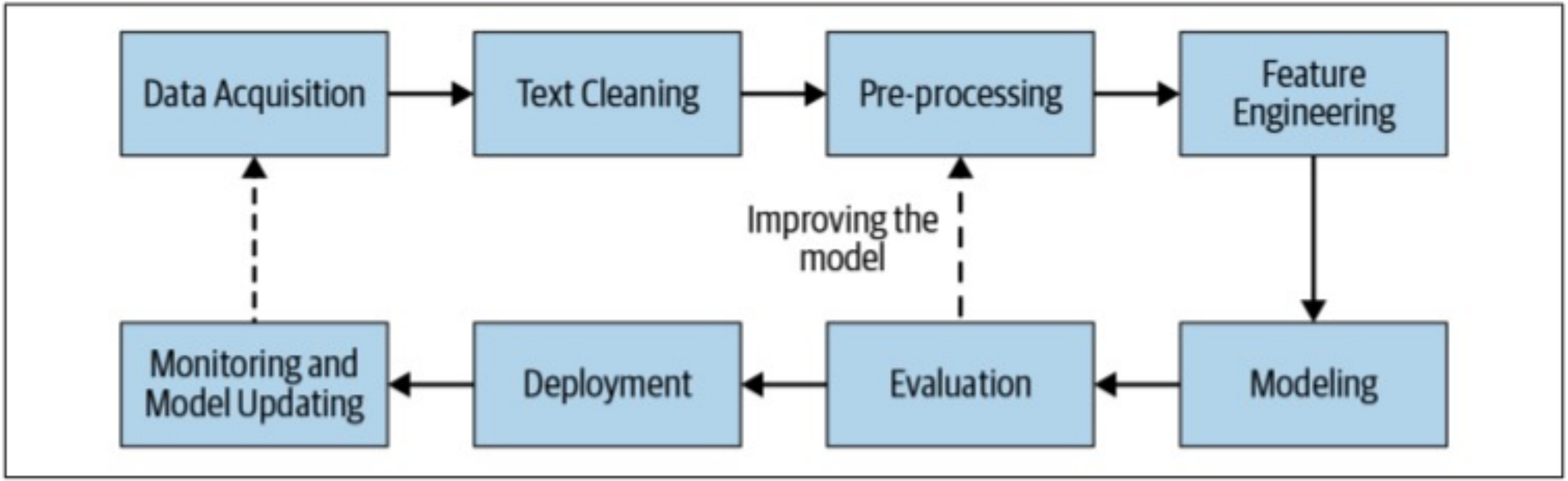
0.6	0.3	0.1	- - - - -

0.8	0.5	0.3	- - - - -

0.4	0.2	0.9	- - - - -

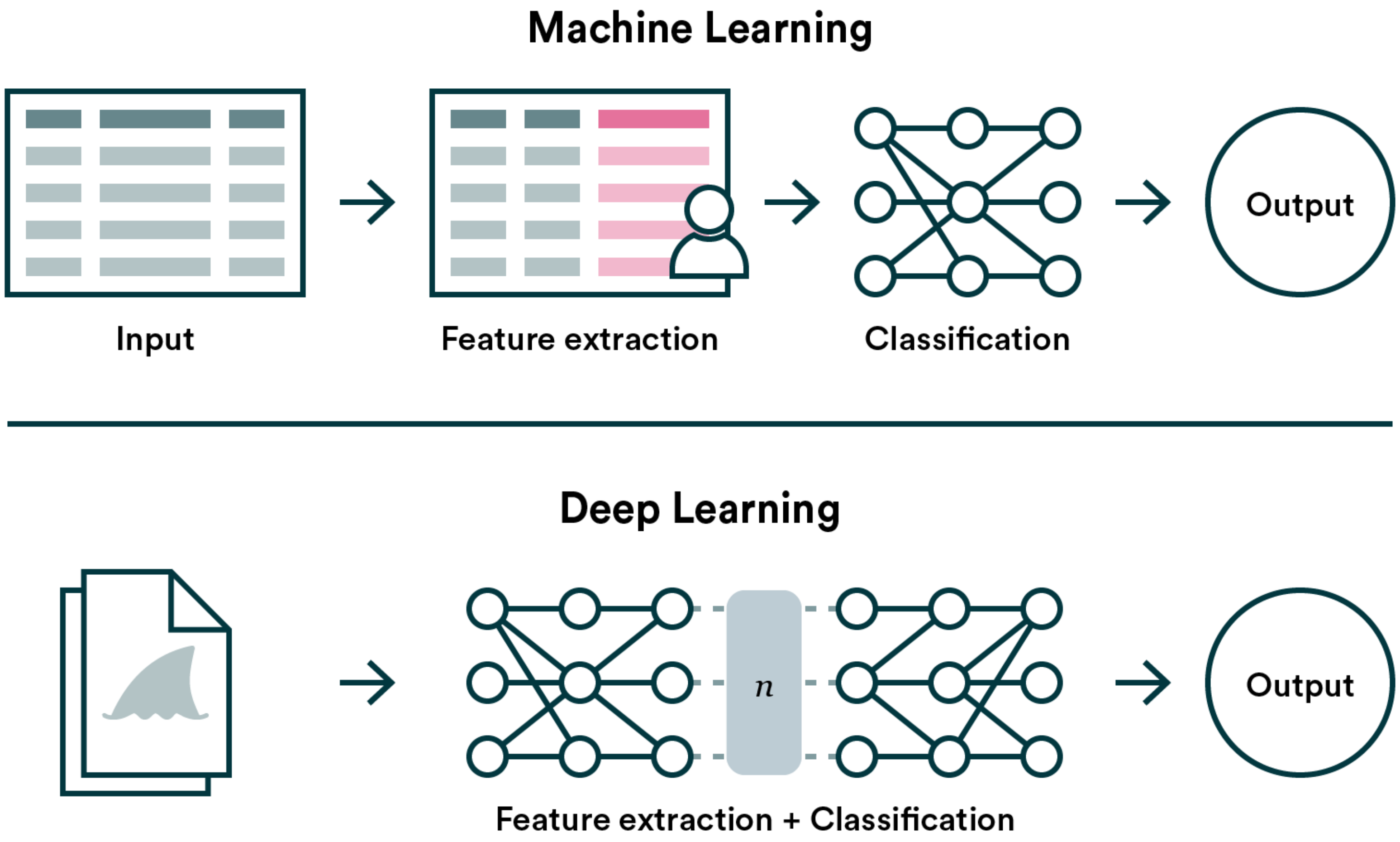


Generic NLP Pipeline



Machine learning vs Deep Learning?

- Handcrafted vs learned features

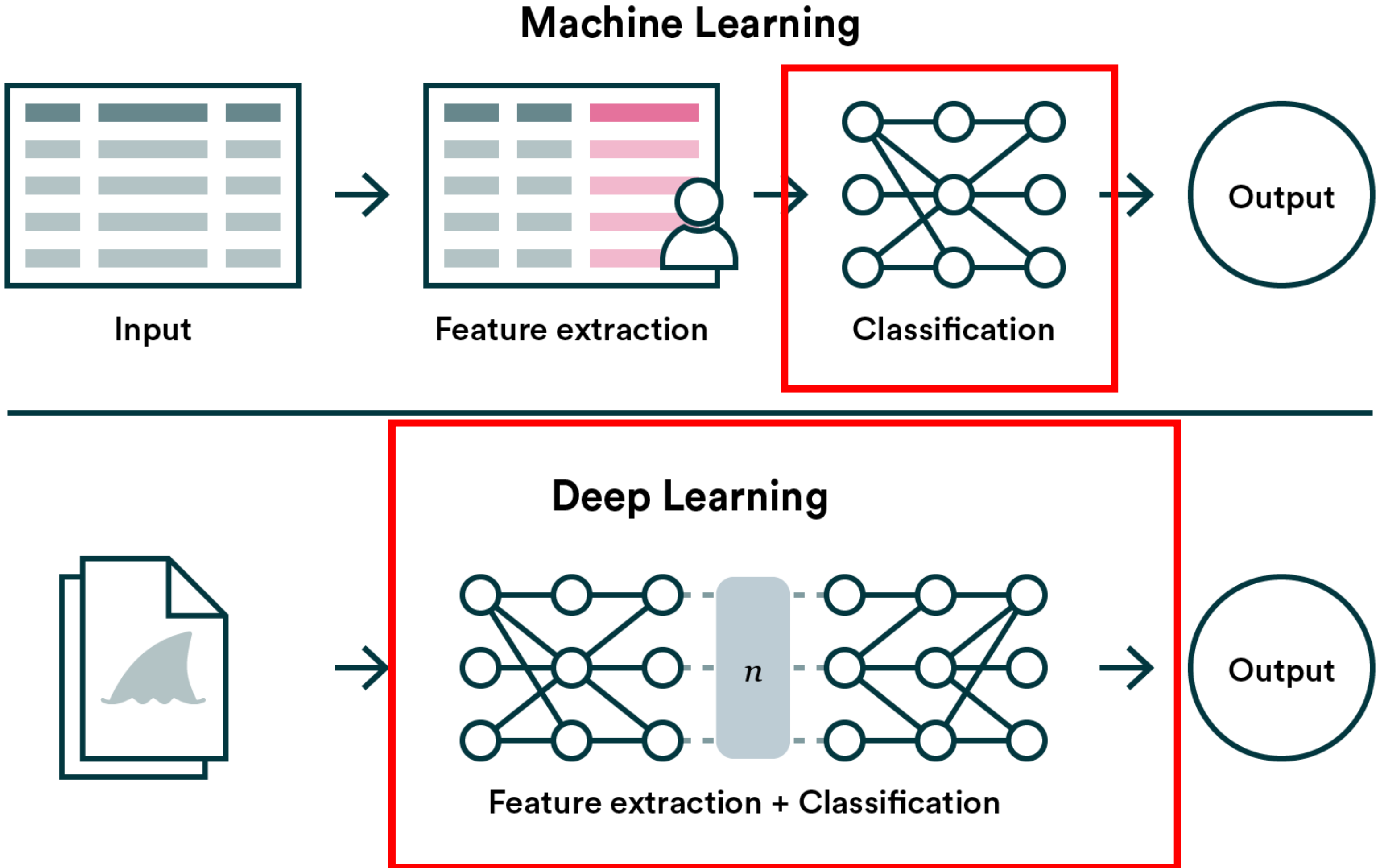


Examples of extracted features of text

- BOW
- TF-IDF
- Handcrafted
  - Cohesion
  - Correctness
  - Syntactic complexity

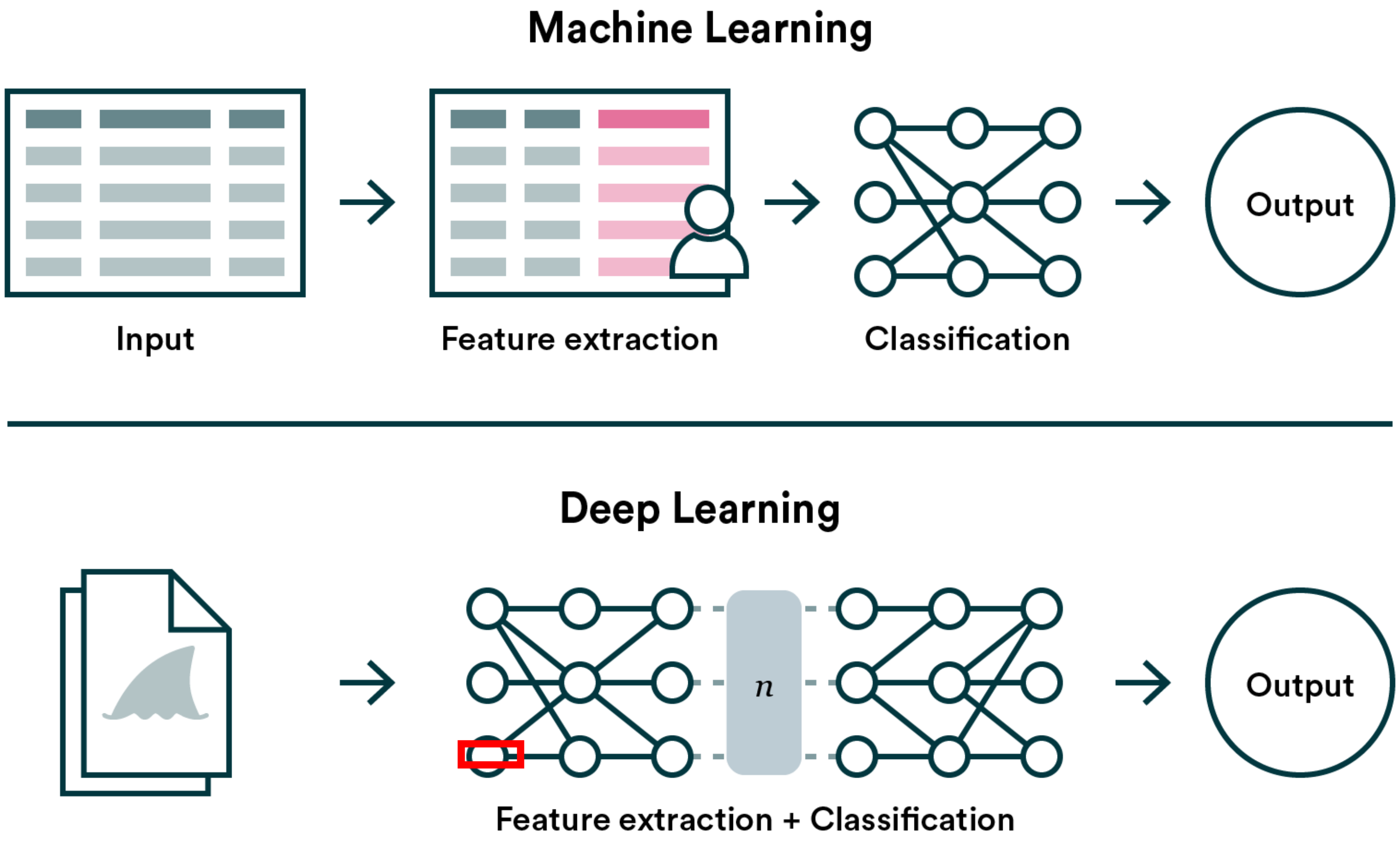
Machine learning vs Deep Learning?

- Handcrafted vs learned features

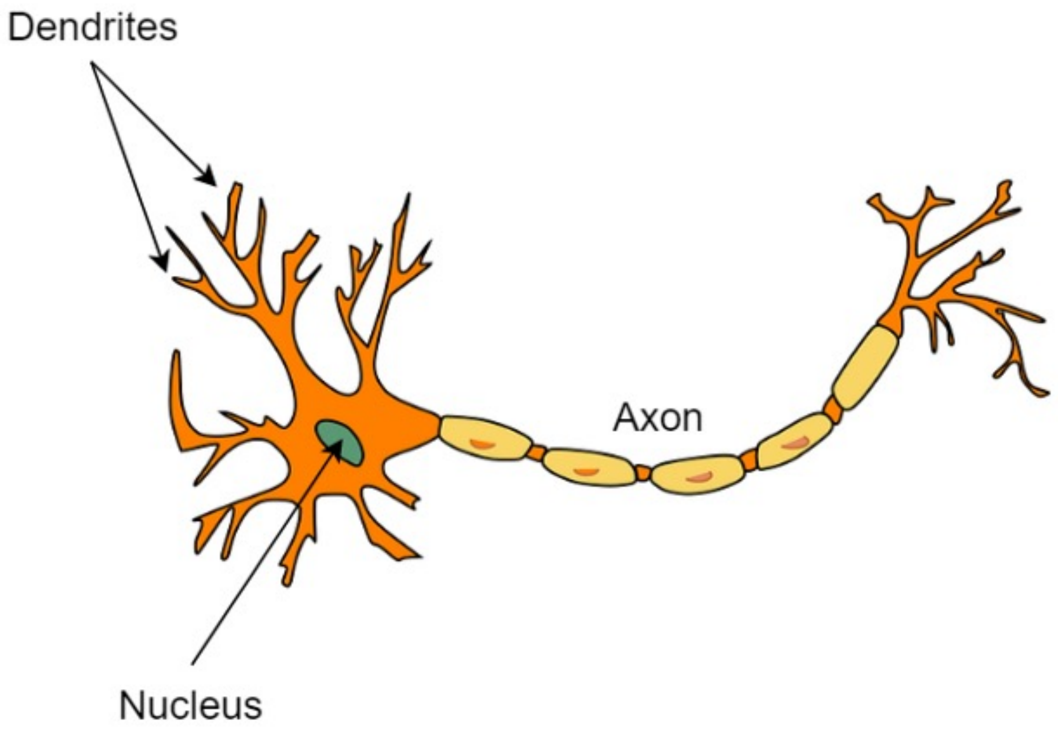


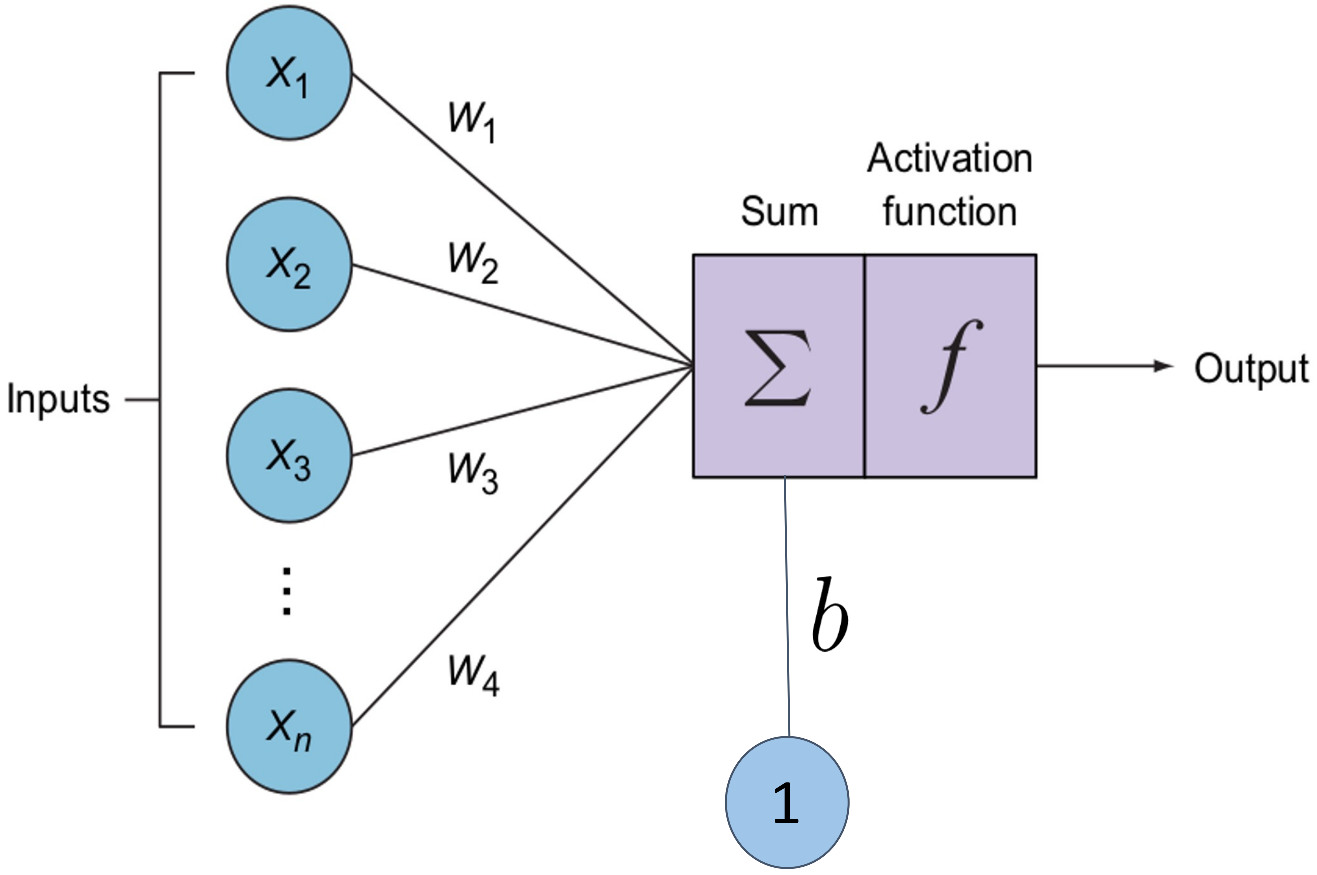
Machine learning vs Deep Learning?

- Handcrafted vs learned features

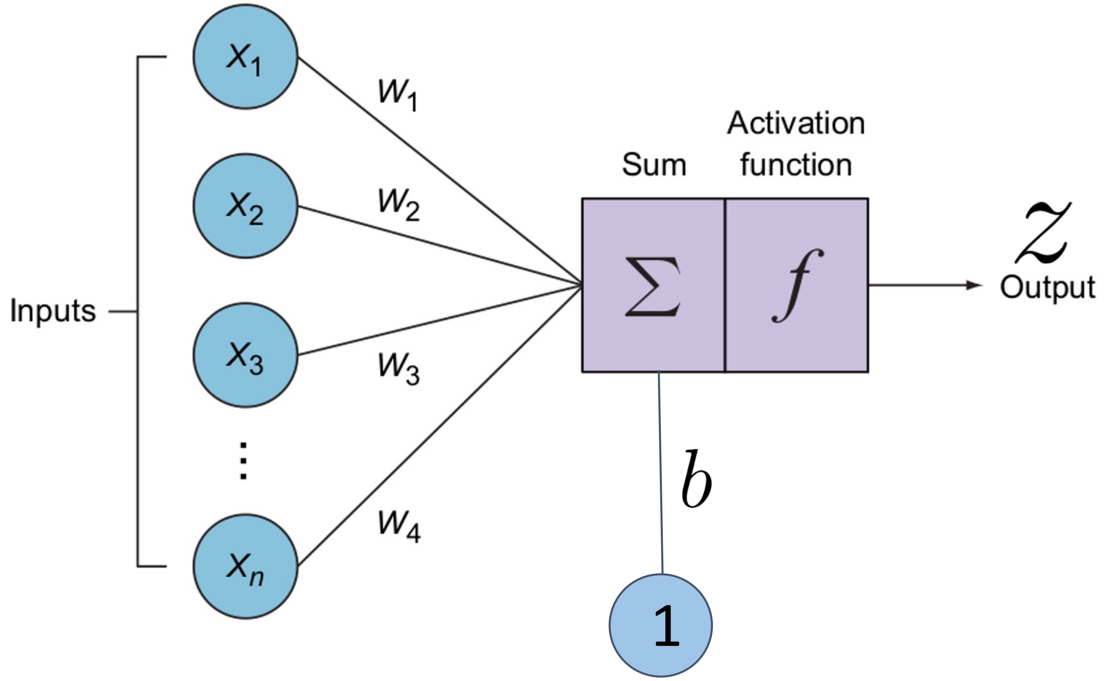


Neurons


Dendrites
Axon
Nucleus


Inputs
x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_4
Sum
Activation function
Σ
f
Output
b
1

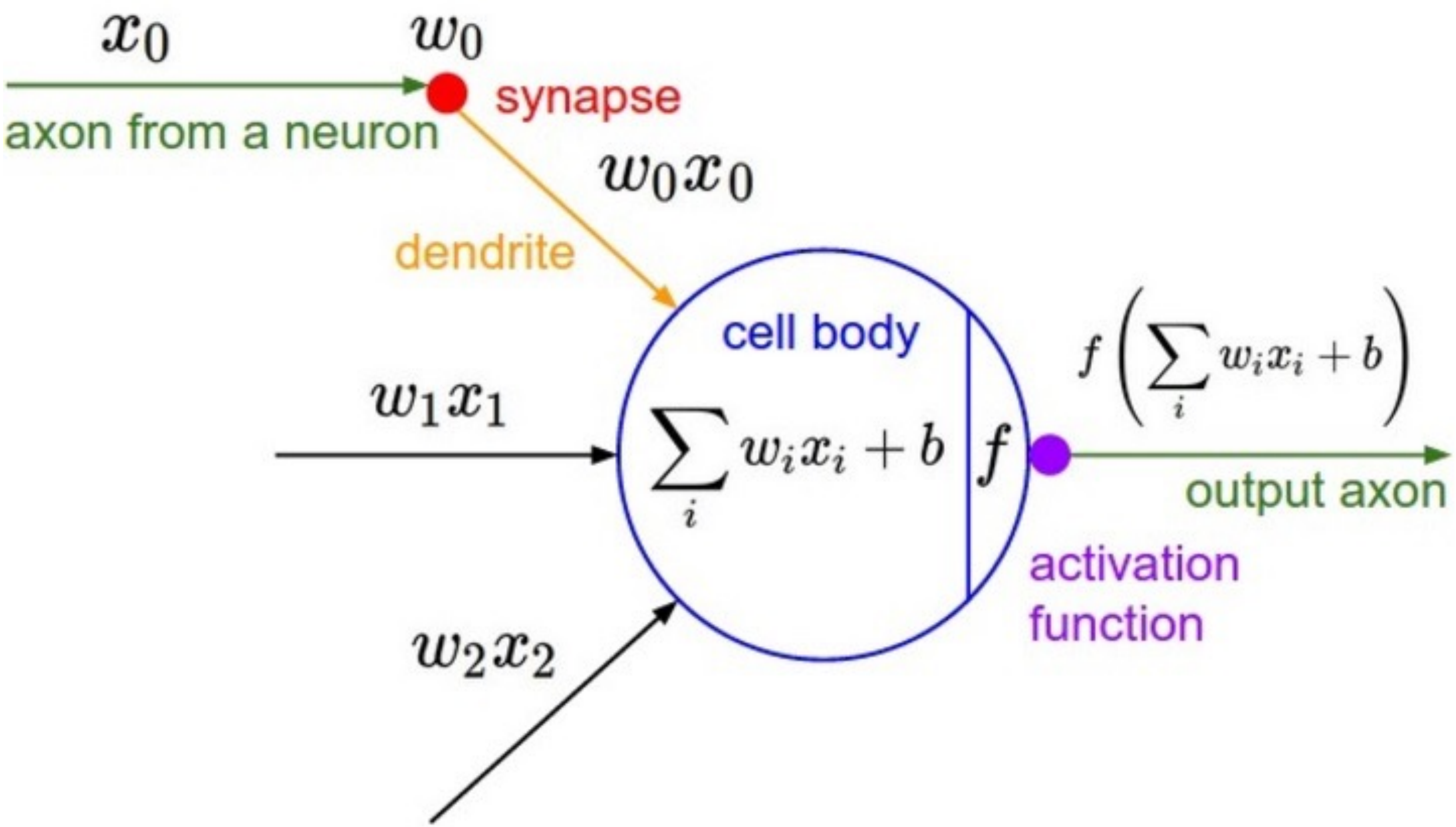
- Input vector $(x_1, x_2, x_3, \ldots, x_n)$
- Weights vector $(w_1, w_2, w_3, \ldots, w_n)$
- Neuron computation $\sum$
- Activation function f
- Output z



$$y = f(w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3 + \cdots + w_n \times x_n + b)$$

$$y = f(\boldsymbol{w} \times \boldsymbol{x} + b) \qquad \boldsymbol{w} = (w_1, w_2, w_3, \ldots, w_n)$$

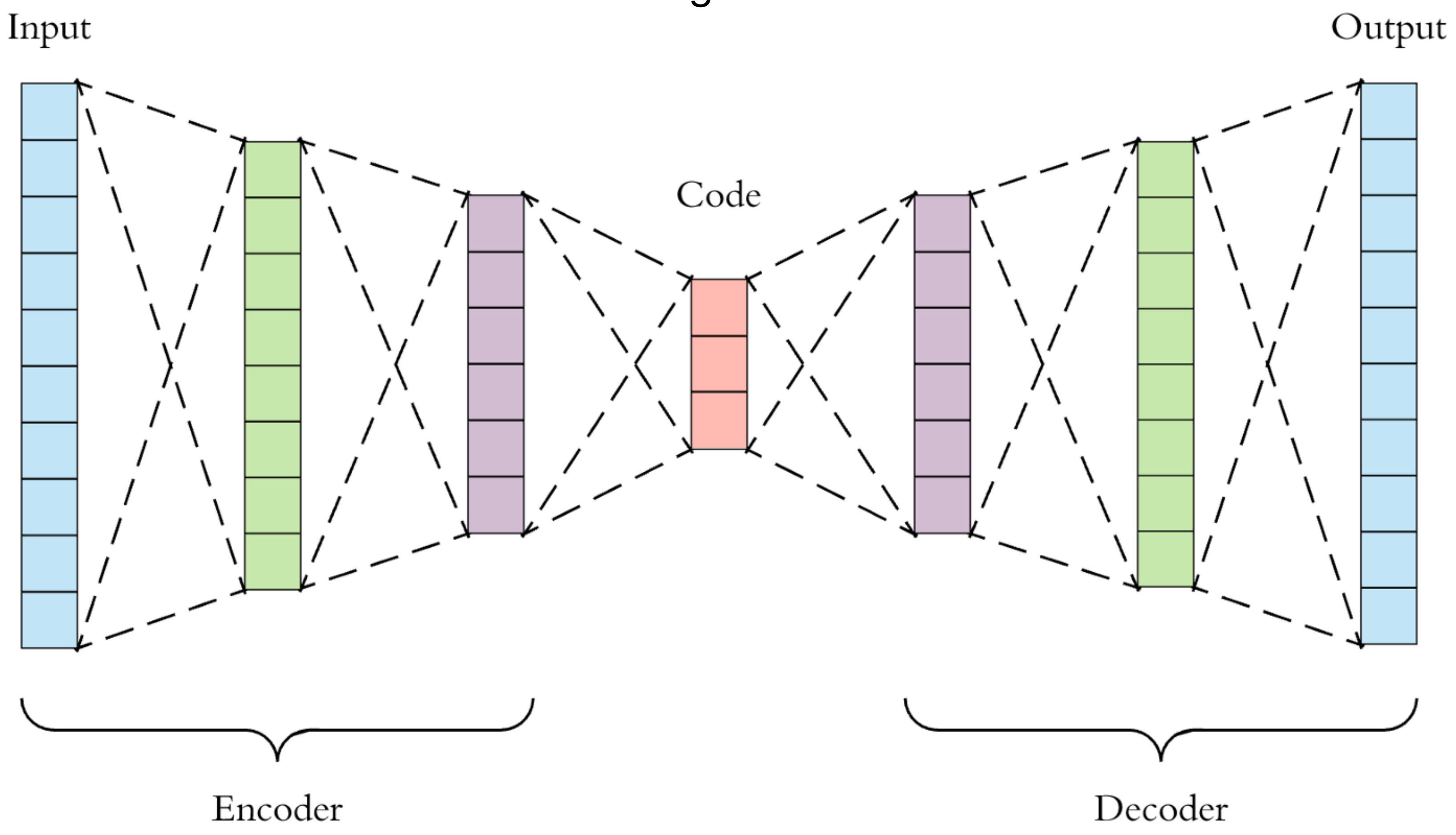
$$\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n)$$



https://www.jeremyjordan.me/intro-to-neural-networks/

Autoencoder

Encoder: word -> embedding
Decoder: embedding -> word

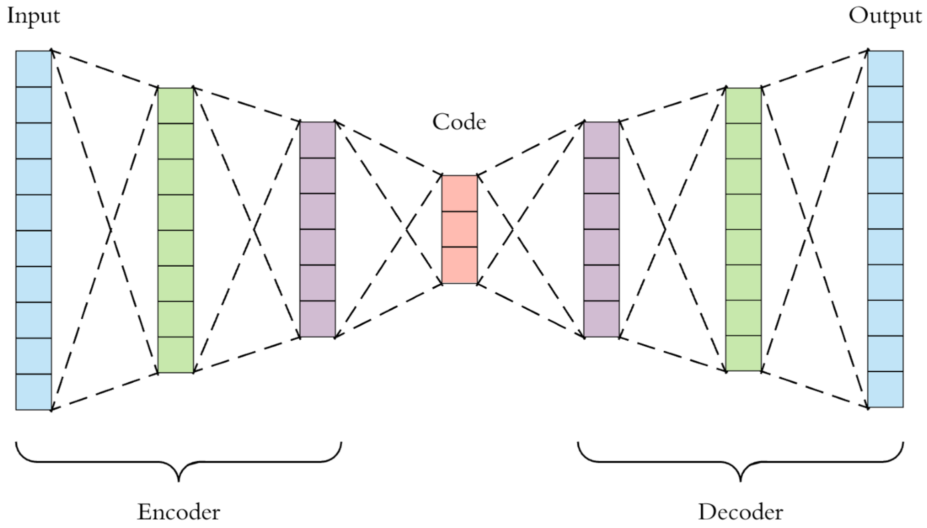


Supervised vs Unsupervised and Self-supervised Models

- What is the difference between an unsupervised, unsupervised, or self-supervised task?

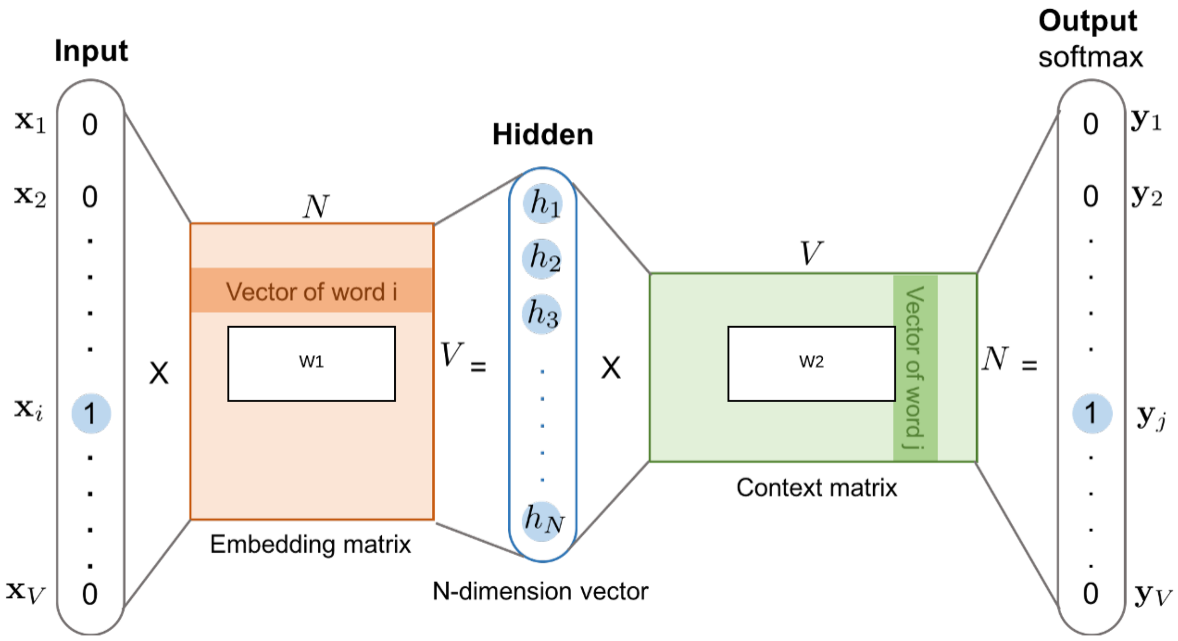
The quick brown fox jumped over the lazy dog.

quick

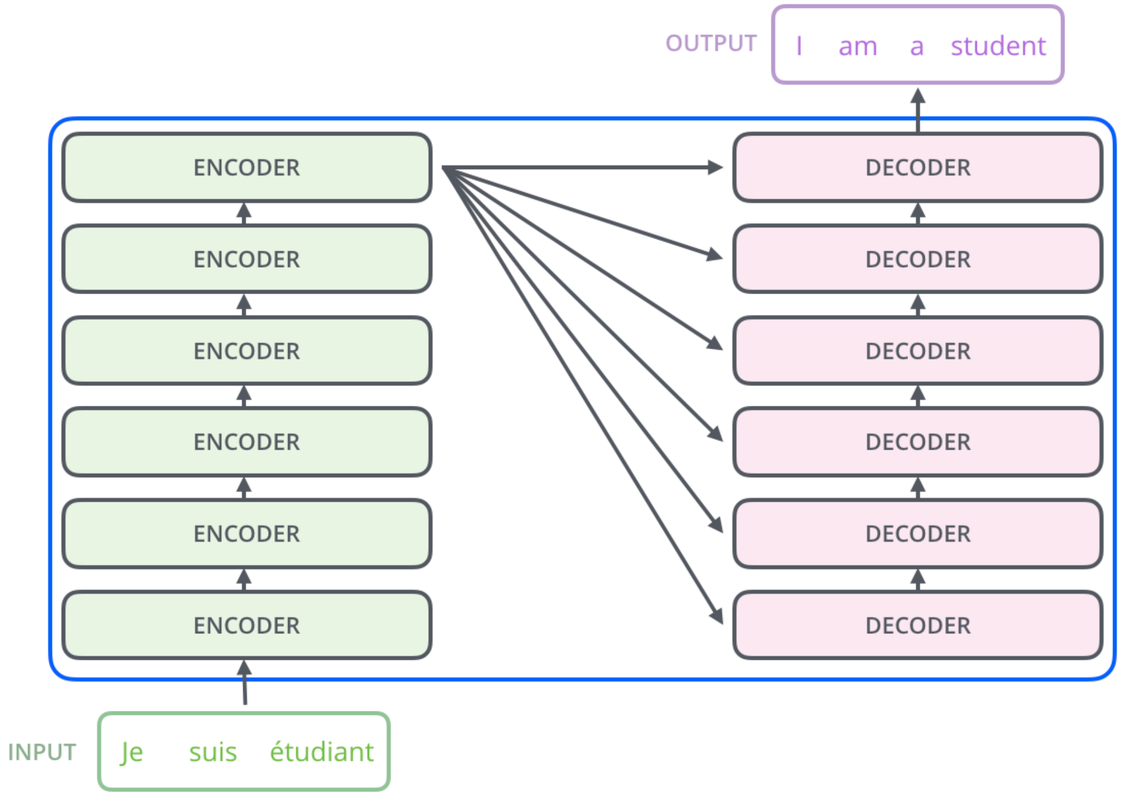


quick

Why focus on autoencoders?



This is what word2vec looks like

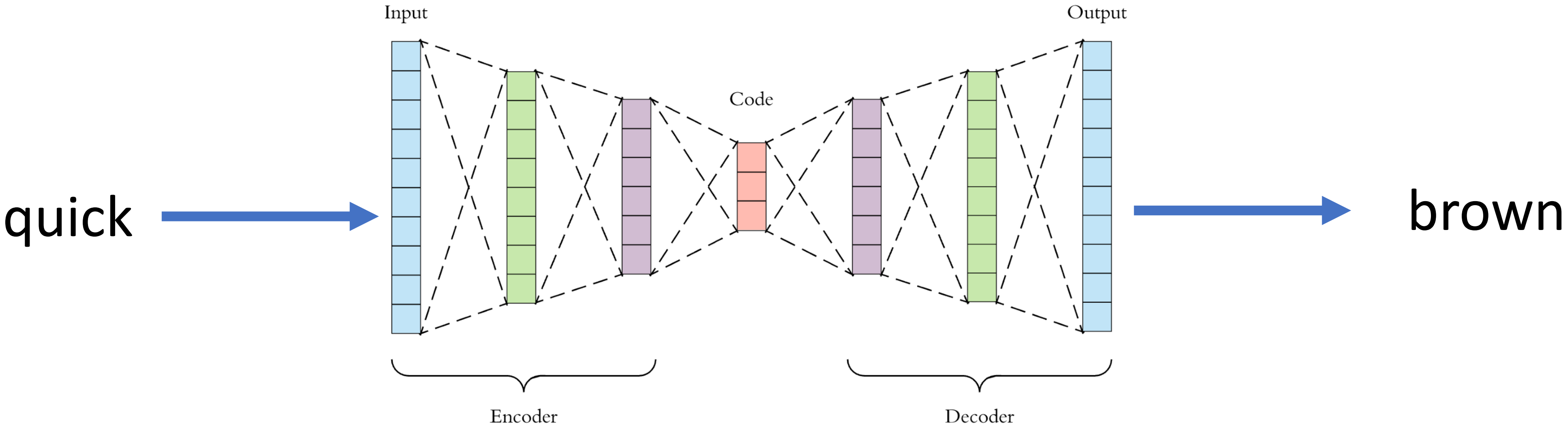


This is an example of a transformer architecture

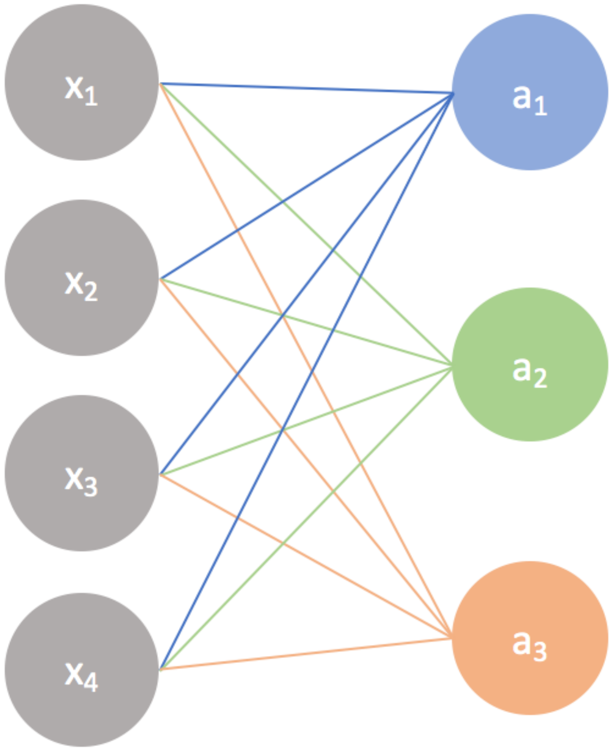
Supervised vs Unsupervised and Self-supervised Models

- What is the difference between an unsupervised, unsupervised, or self-supervised task?

The quick brown fox jumped over the lazy dog.

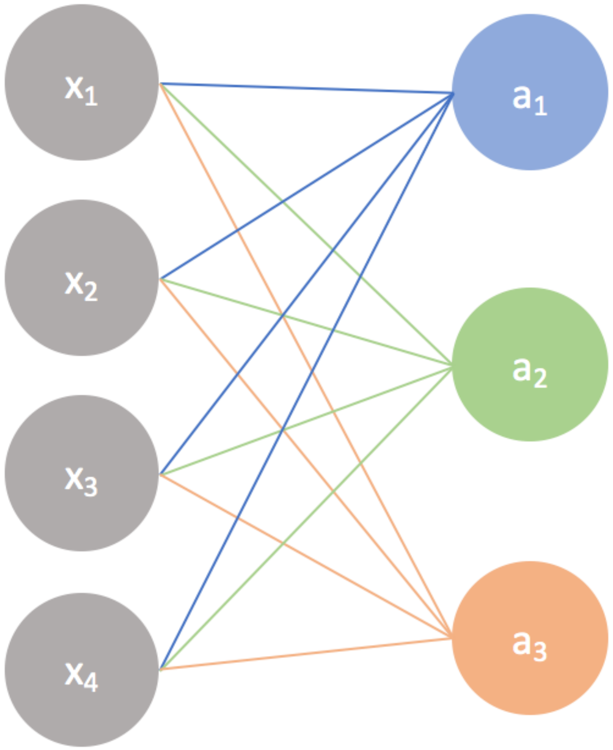


Building a network of neurons



https://www.jeremyjordan.me/intro-to-neural-networks/

Building a network of neurons

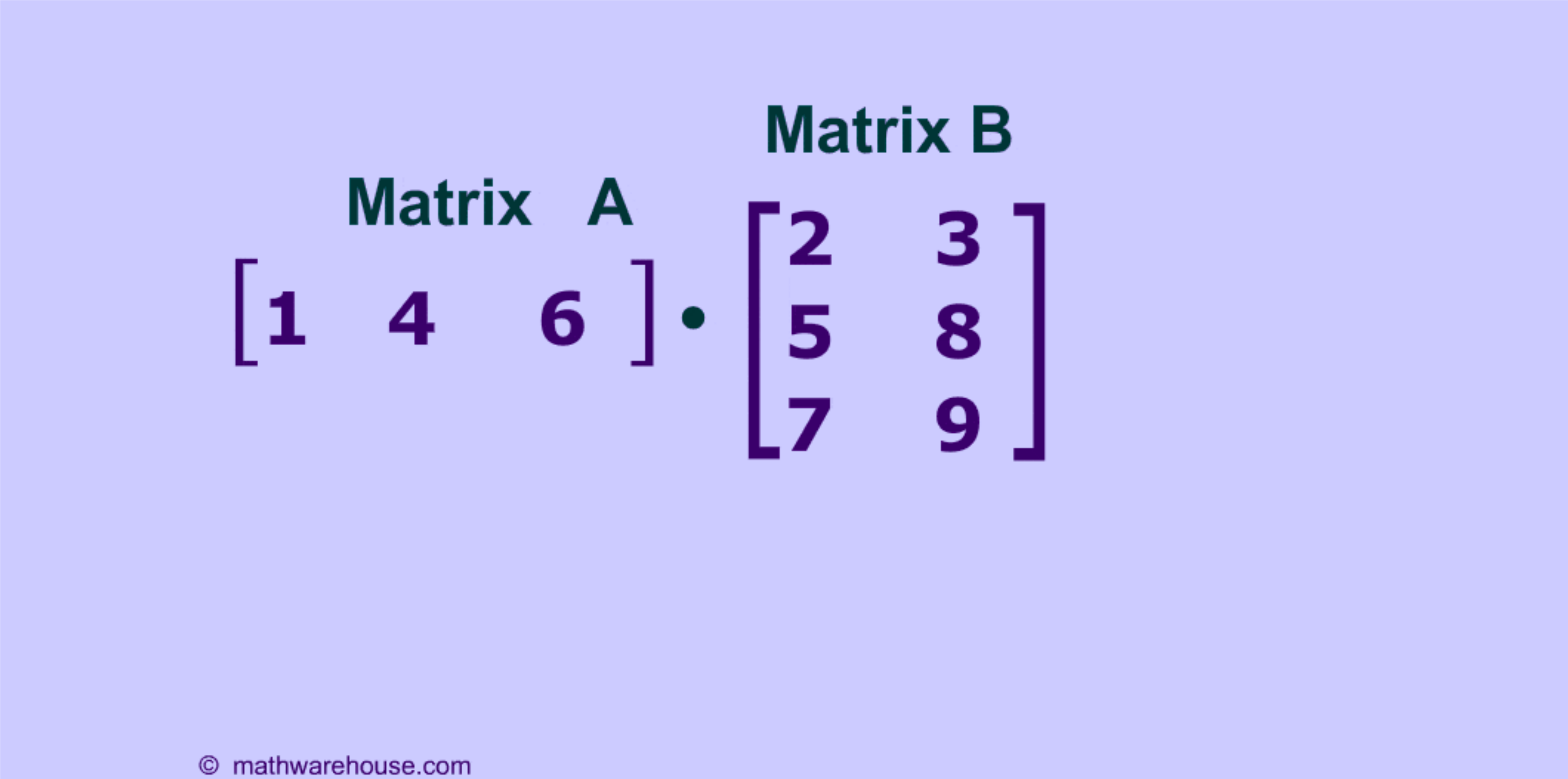


$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[activation]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

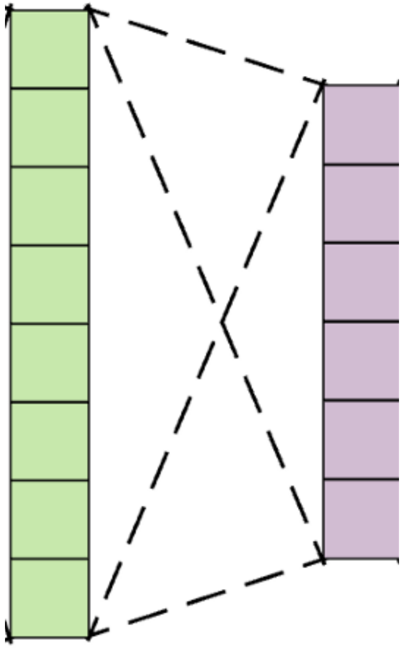
Just a series of matrix multiplications!

https://www.jeremyjordan.me/intro-to-neural-networks/

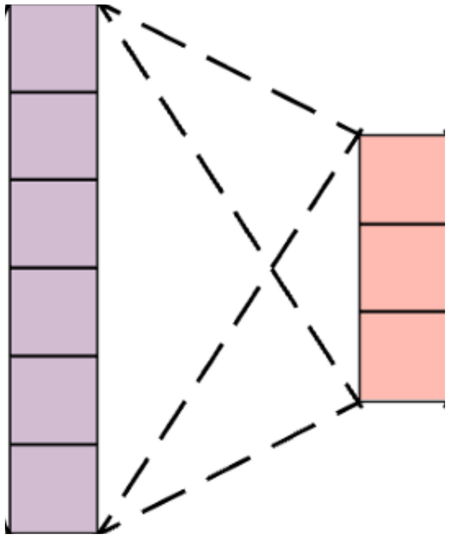
Matrix Multiplication



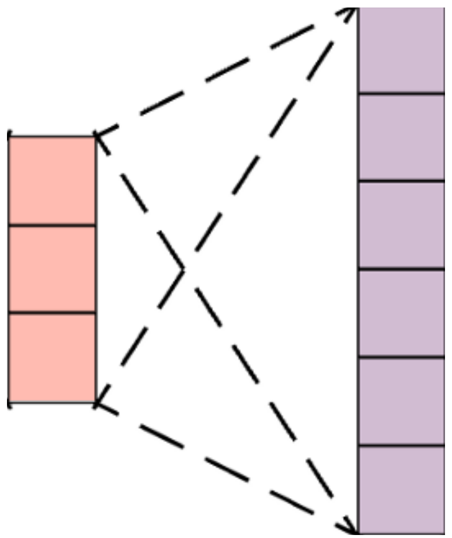
Let's expand on our simple example of 1 hidden layer



Let’s expand on our simple example of 1 hidden layer

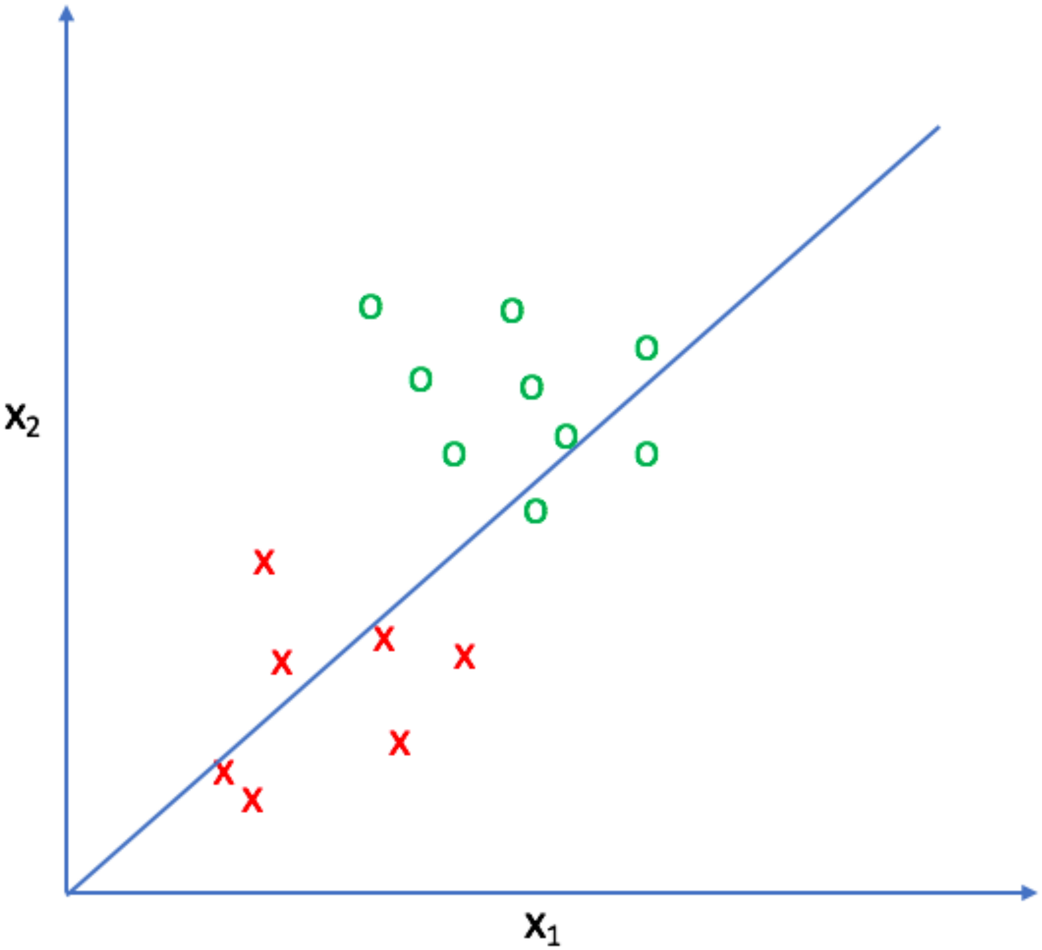


Let's expand on our simple example of 1 hidden layer

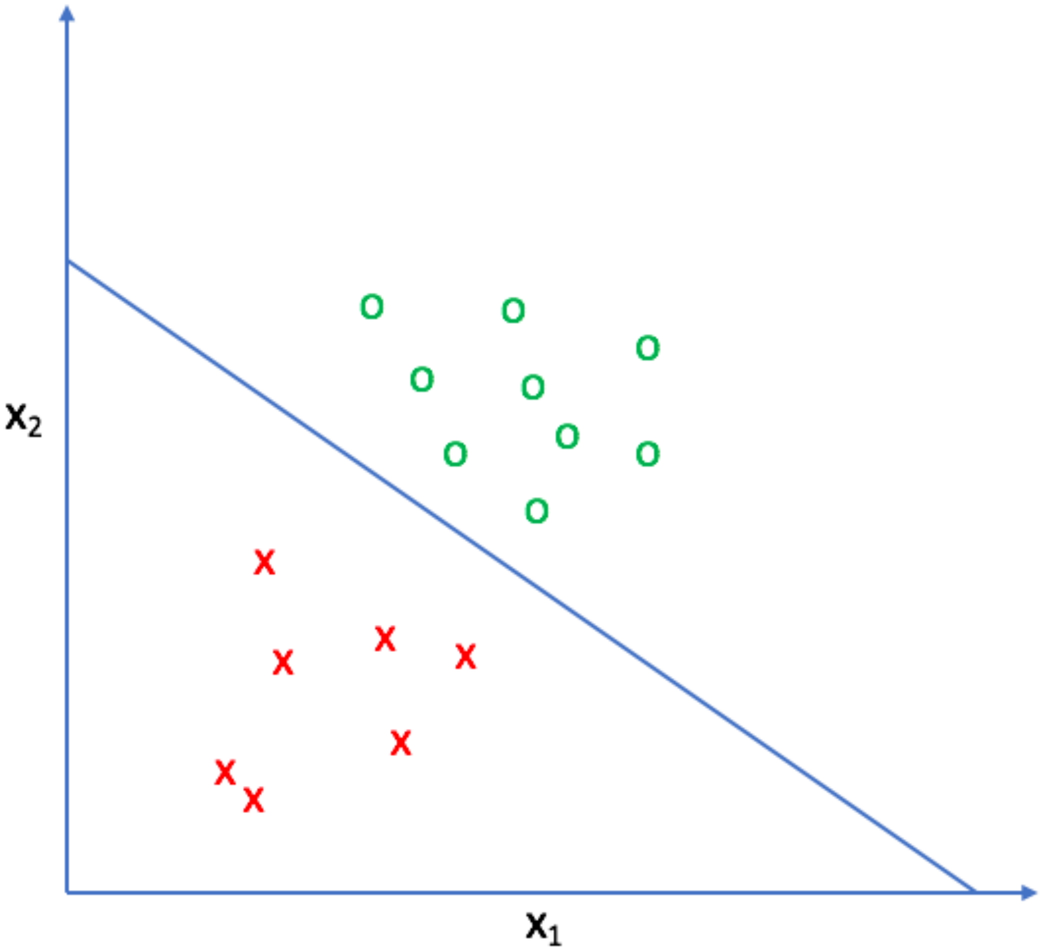


Do we need a bias term (b)?

No bias (intercept) term

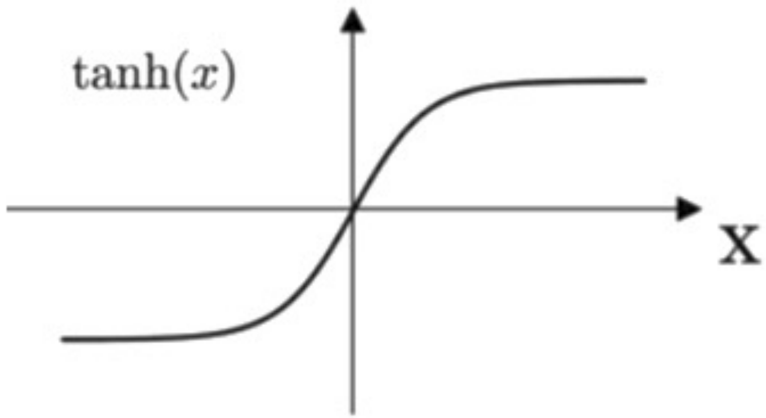


With bias (intercept) term

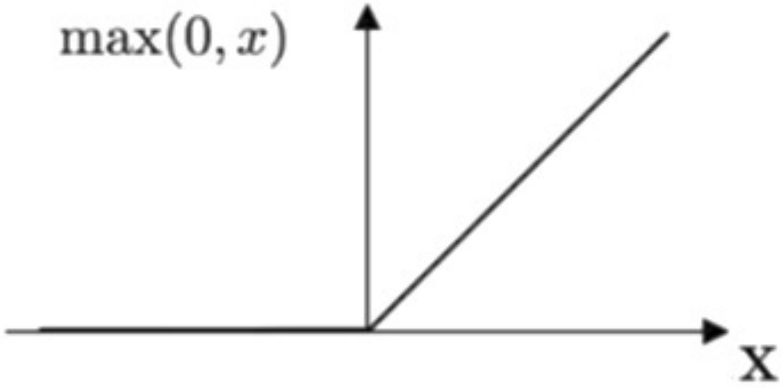


Activation functions

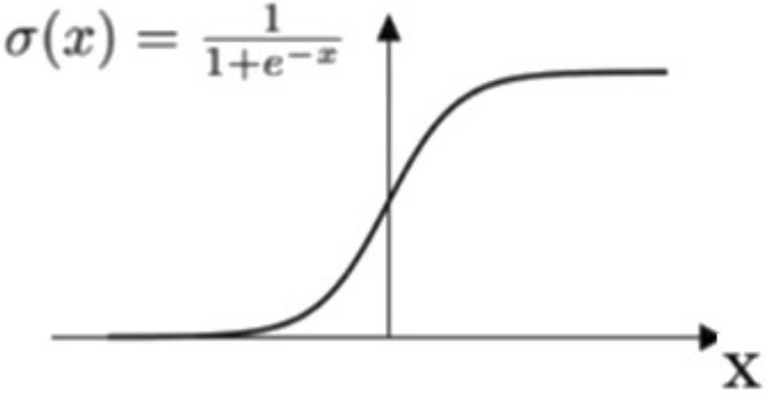
Tanh



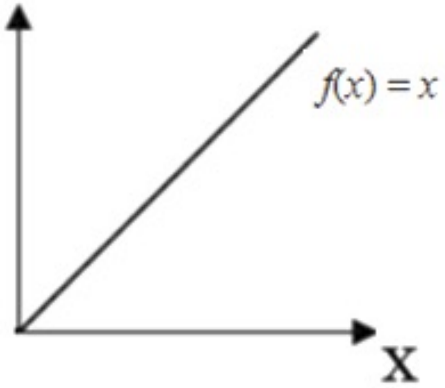
ReLU

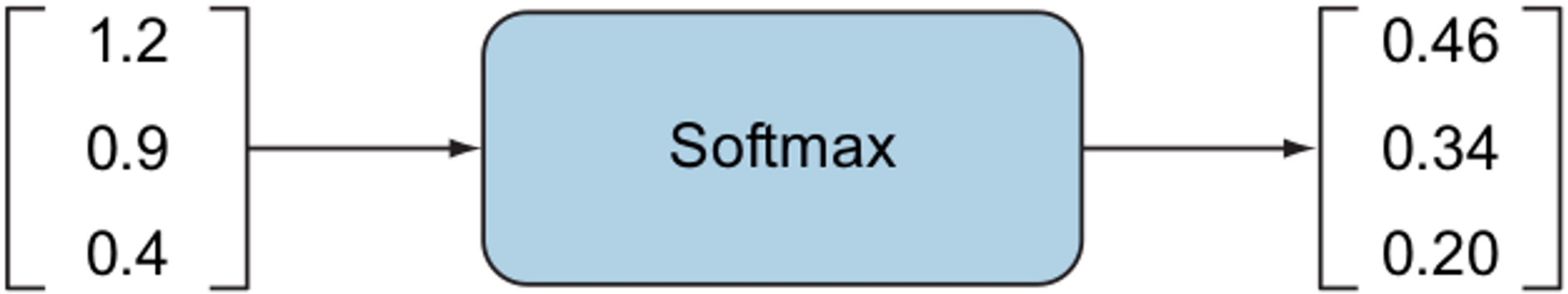


Sigmoid



Linear





$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

```
def softmax(x):
 elements = np.exp(x)
 return elements/np.sum(elements)
```

Loss functions

We will focus on **Cross Entropy Loss**:

CE $Loss = -\sum y_i \ * \ln(\widehat{y_i})$

This implements **softmax** for us

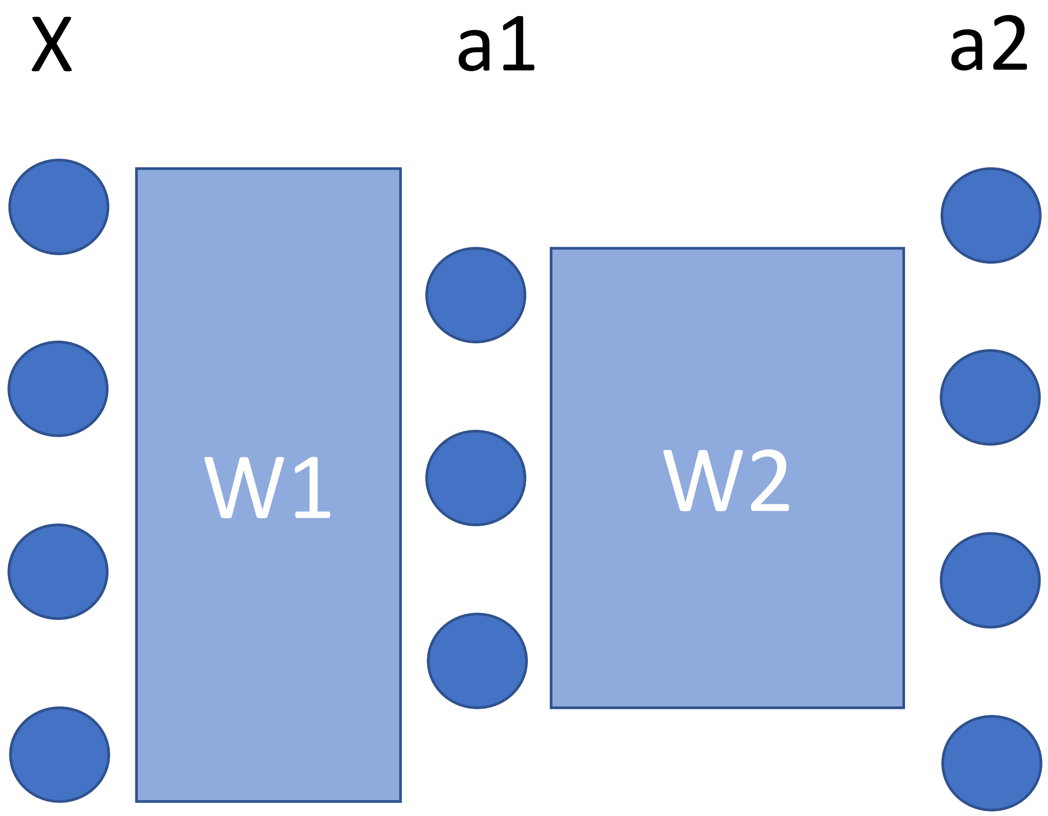
$$\widehat{y_i} \ = \frac{e^{\hat{y}_i}}{(\sum_k \mathrm{e}^{\hat{y}_k})}$$

Extra: Back-propagation

- Backpropagation with **cross entropy loss** is used to update the vector representations of words based on the error between the predicted and actual target words
- Computing gradients of expressions efficiently through recursive application of chain rule

$$\partial f/\partial x = \partial f/\partial g * \partial g/dx$$

Forward Pass



- z1 = W1*x + b1
- a1 = f(z1)
- z2 = W2*a1 + b2
- a2 = softmax(z2)

Calculate the loss:

loss = loss_function(a2, y)

Backpropagation: Step 1

- a1 = W1*x + b1
- z2 = W2*a1 + b2
- a2 = softmax(z2)

Calculate the loss:

loss = loss_function(a2, y)

Calculate the gradient of the loss with respect to the output

$$\frac{\partial Loss}{\partial z2} = a2 - y$$

$$\frac{\partial Loss}{\partial W2} = \frac{\partial Loss}{\partial z2} * \frac{\partial z2}{\partial W2}$$

$$\frac{\partial Loss}{\partial W2} = (a2 - y) * a1$$

Backpropagation: Step 1

- a1 = W1*x + b1
- z2 = W2*a1 + b2
- a2 = softmax(z2)

Calculate the loss:

loss = loss_function(a2, y)

Calculate the gradient of the loss with respect to the output

$$\boxed{\frac{\partial Loss}{\partial z2} = a2 - y}$$

$$\frac{\partial Loss}{\partial W2} = \frac{\partial Loss}{\partial z2} * \frac{\partial z2}{\partial W2}$$

$$\frac{\partial Loss}{\partial W2} = (a2 - y) * a1$$

https://www.youtube.com/watch?v=5-rVLSc2XdE

Backpropagation: Step 2

- a1 = W1*x + b1
- z2 = W2*a1 + b2
- a2 = softmax(z2)

Calculate the loss:

loss = loss_function(a2, y)

$$\frac{\partial Loss}{\partial W2} = (a2 - y) * a1$$

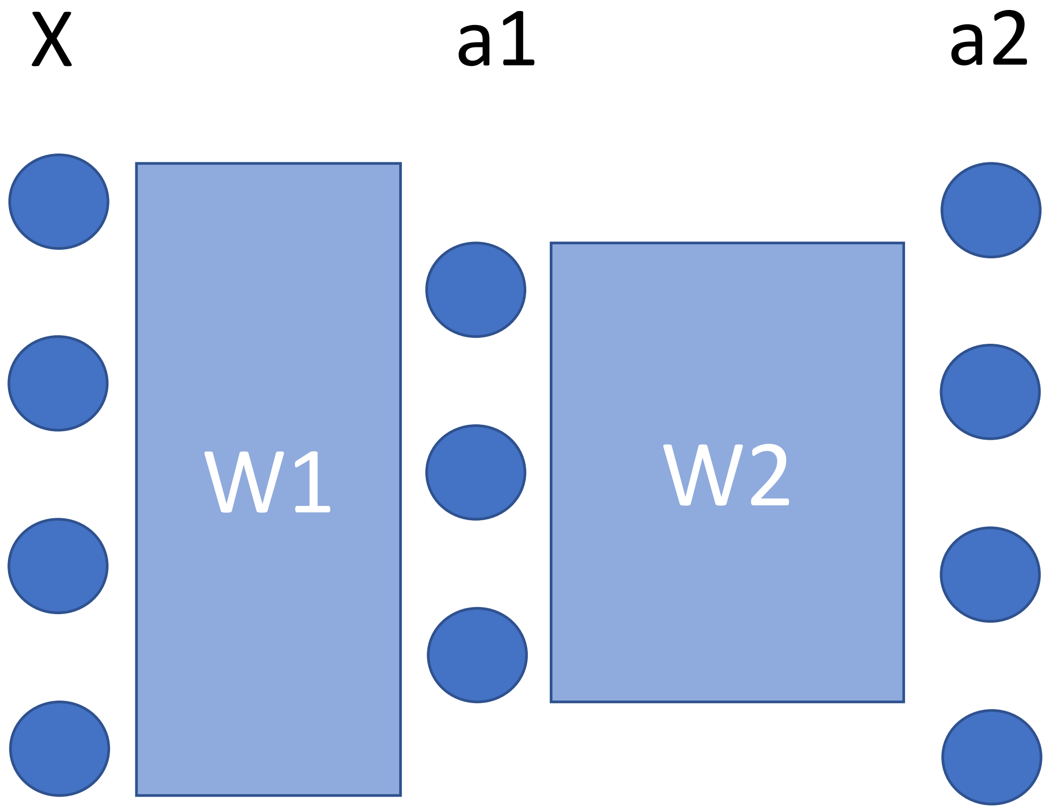
$$\frac{\partial Loss}{\partial a1} = \frac{\partial Loss}{\partial z2} * \frac{\partial z2}{\partial a1}$$

$$\frac{\partial Loss}{\partial a1} = (a2 - y) * W2$$

$$\frac{\partial Loss}{\partial W1} = \frac{\partial Loss}{\partial a1} * \frac{\partial a1}{\partial W1}$$

$$\frac{\partial Loss}{\partial W1} = ((a2 - y) * W2) * x$$

Backpropagation: Update weights and bias terms

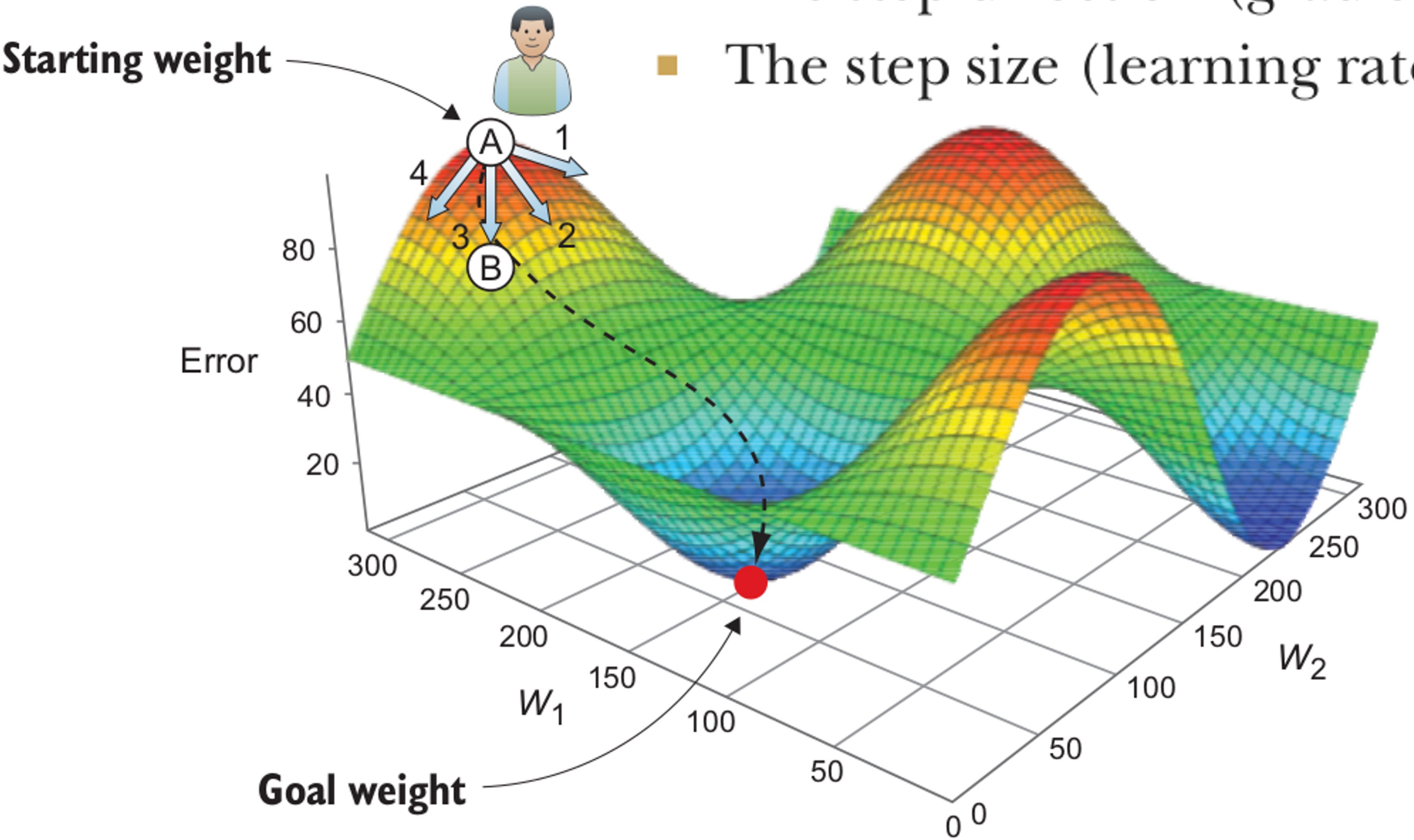


$$W2 = W2 - \eta \frac{\partial Loss}{\partial W2}$$

$$W1 = W1 - \eta \frac{\partial Loss}{\partial W1}$$

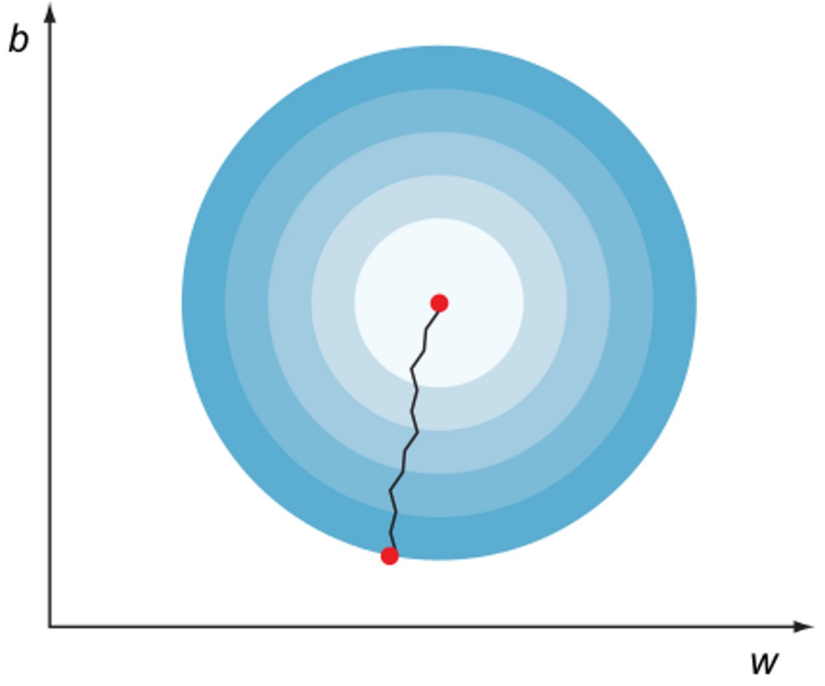
$$\eta = learning\ rate$$

- The step direction (gradient)
- The step size (learning rate)



GD

1. Take all the data.
2. Compute the gradient.
3. Update the weights and take a step down.

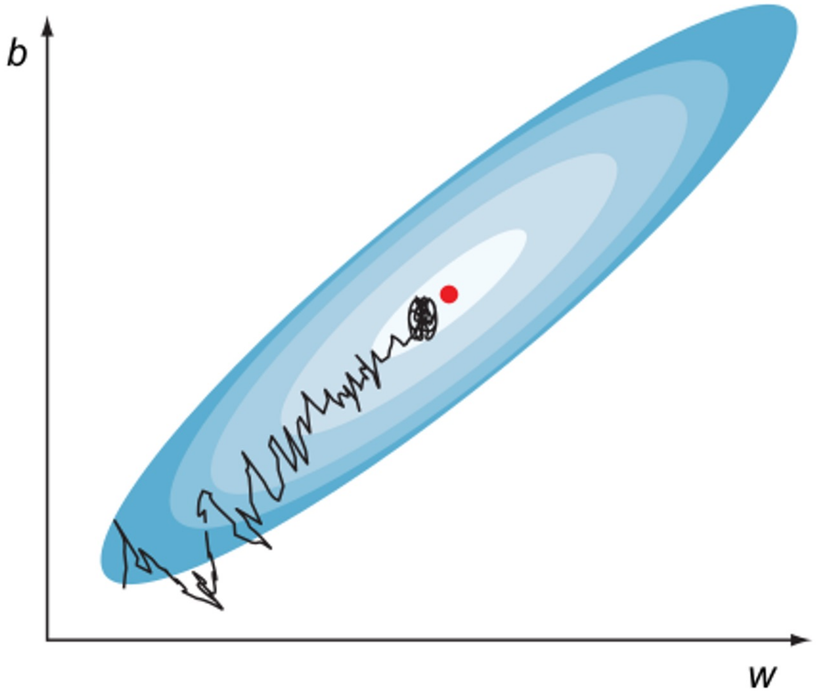


4. Repeat for *n* number of epochs (iterations).

A smooth path for the GD down the error curve

Stochastic GD

1. Randomly shuffle samples in the training set.
2. Pick one data instance.
3. Compute the gradient.
4. Update the weights and take a step down.
5. Pick another one data instance.
6. Repeat for *n* number of epochs (training iterations).



An oscillated path for SGD down the error curve

Common terminology you may come across when training neural networks

1. **Epoch** — One epoch is when the entire dataset is passed through the network once. This comprises of one instance of a forward pass and back-propagation.
2. **Batch size** — The number of training examples passed through the network simultaneously.
3. **The number of iterations** — One iteration equals one pass using training examples set as batch size. One pass is a forward pass and a back-propagation.

Common terminology you may come across when training neural networks

4. Loss Function — Calculates the error in a model’s guess/prediction

5. Optimizer — Optimization is the process of adjusting model parameters to reduce model error in each training step. Optimization algorithms define how this process is performed

6. Learning Rate — Parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function

Common terminology you may come across when training neural networks

4. Loss Function — calculates the error in a model's guess/prediction

5. Optimizer — optimization is the process of adjusting model parameters to reduce model error in each training step. Optimization algorithms define how this process is performed

6. Learning Rate — parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function

We can change these to get better performing models
= Hyperparameter tuning

Next time: the Word2Vec architecture