An Introduction to Neural Networks and Deep Learning

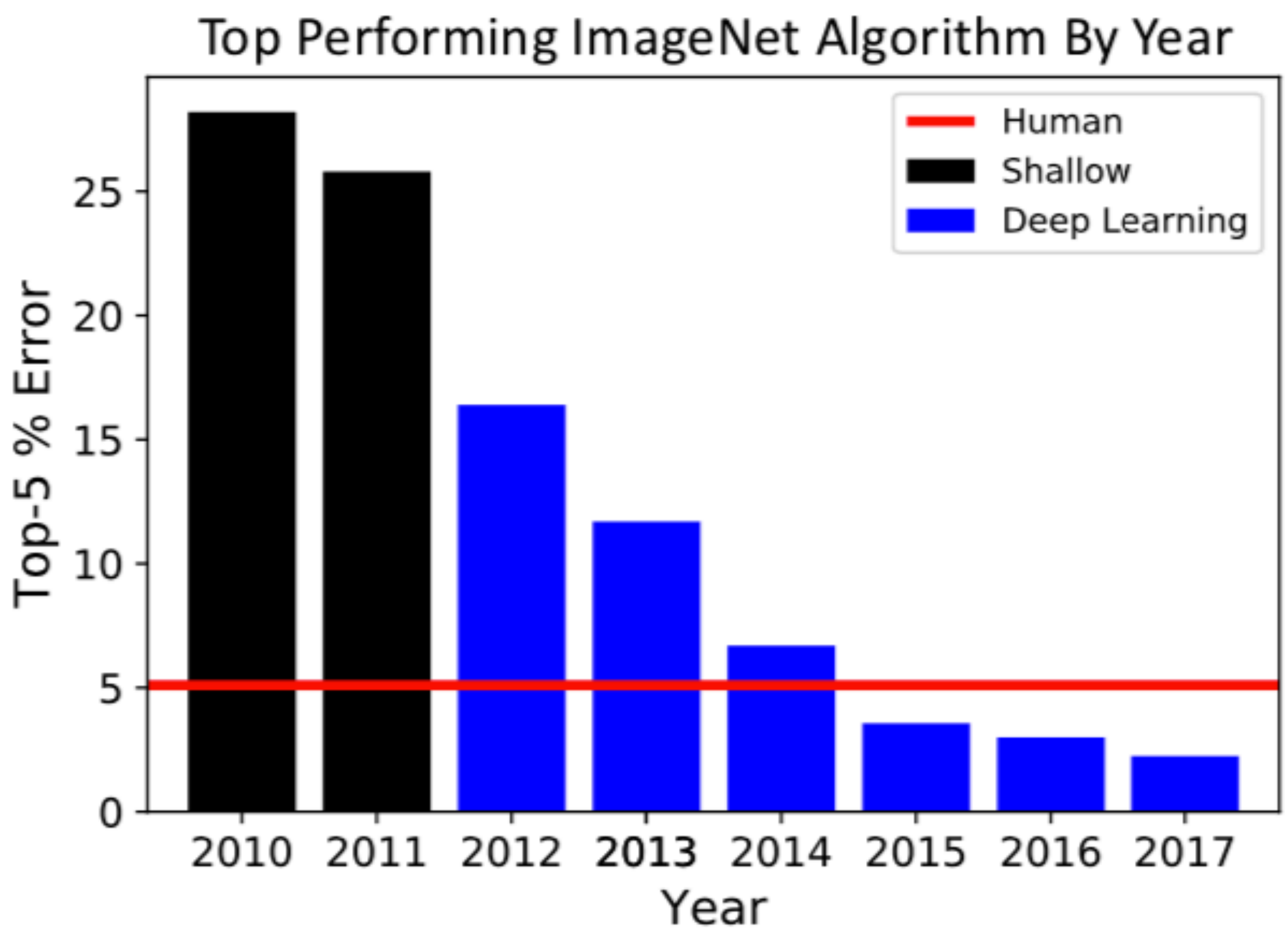
Eric Gossett

Objectives

- Provide an introduction to deep learning with an emphasis on first principles:
  - Intro to machine learning
  - The theory of neural networks
  - How to create a neural network from scratch.
- You will learn algorithms that use data to automate tasks and make predictions
- Can apply these techniques to numerous tasks such as: image classification, voice recognition, object detection, text translation etc.

What is Deep Learning?

- Deep learning is not new! (neural networks [NN] were introduced in the 1950's)
- Resurgence is due to improvements of learning algorithms, more computational power and amazing performance metrics.
- In particular the spark was the ImageNet Challenge where NN out preformed all previous methods!



Machine learning (ML)

- ML focus is developing an algorithm that learns a task from a set of data.

x, a set of **features**

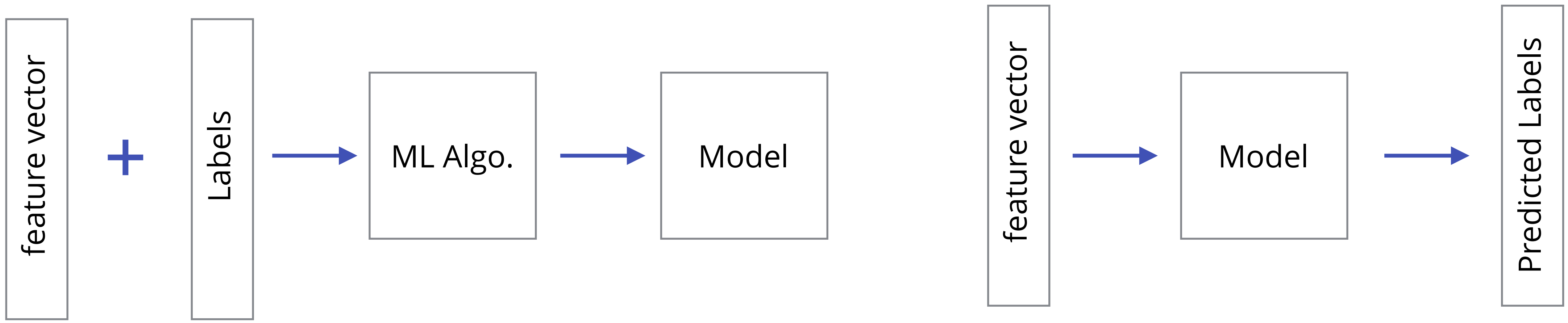
y, associated **labels**

goal: predict **y** from **x**

Machine learning (ML)

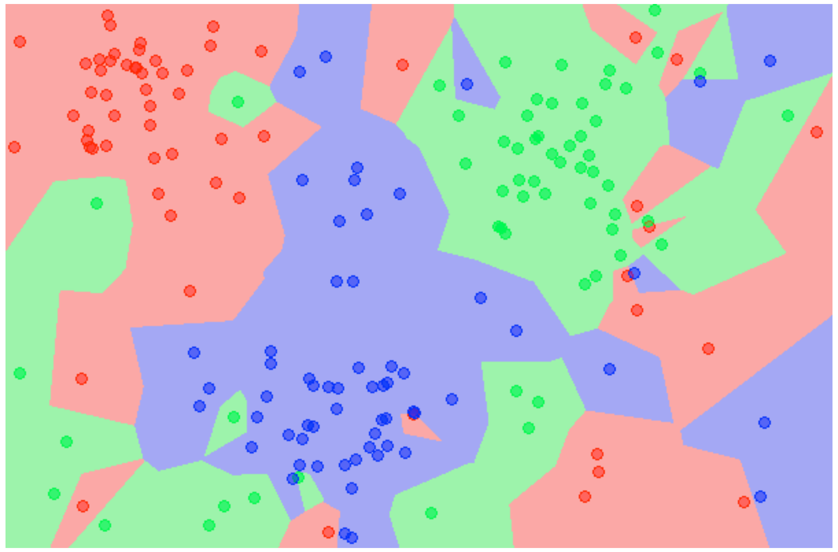
- Two types of tasks regression and classification.
- **unsupervised learning** - Infer a model from unlabeled data.
- **supervised learning** - Infer a models that maps feature to a label by training on a labeled set of data.

Training set

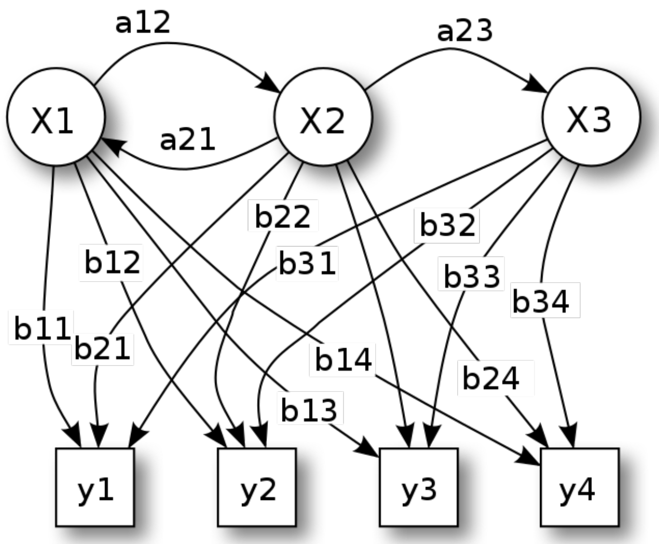


Unsupervised Learning Methods

Clustering



Hidden Markov Model

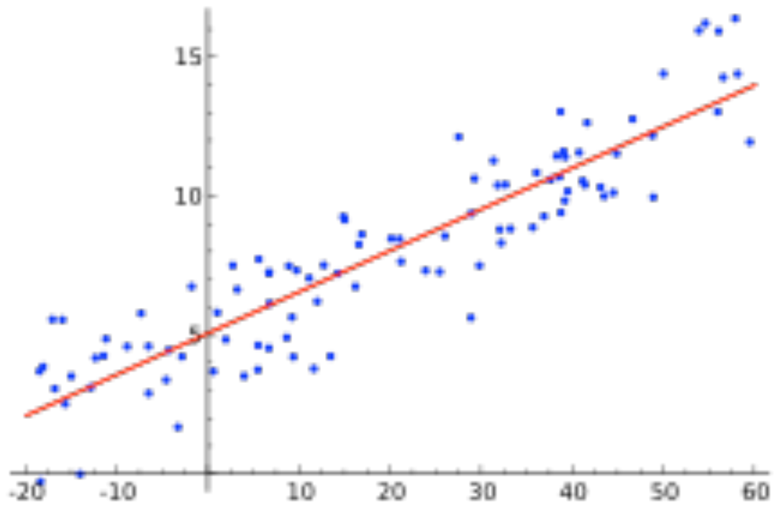


Single Value Decomposition (SVD)

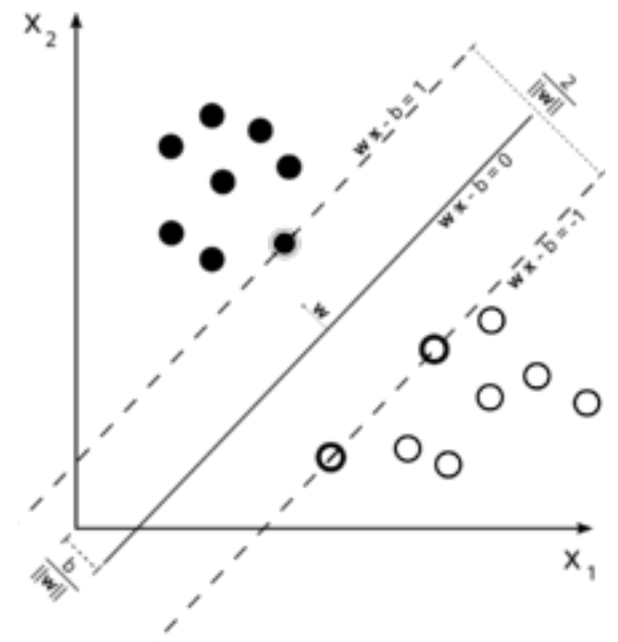
$$A = U\Sigma V^T$$

Supervised Learning methods

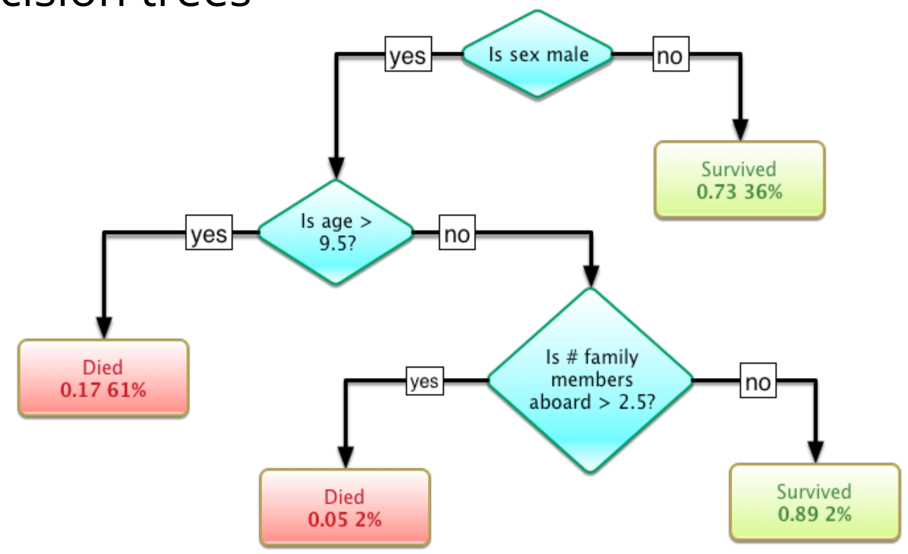
Linear regression



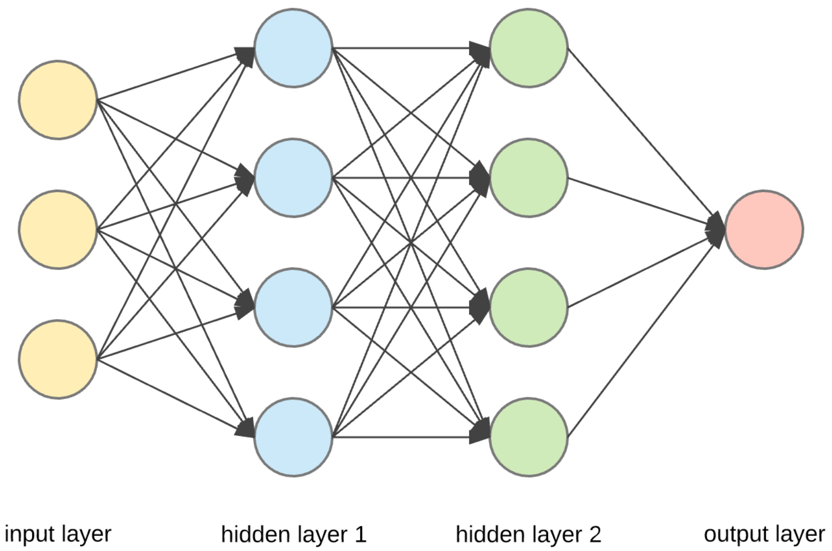
Support vector machines (SVM)



Decision trees



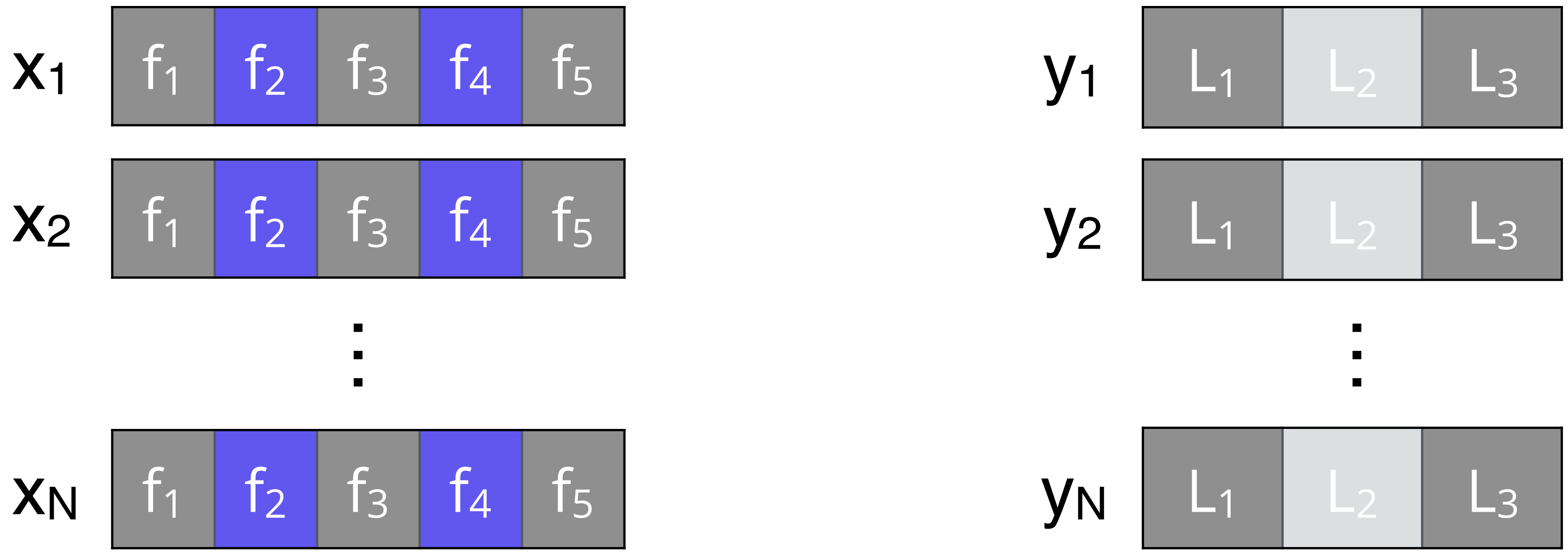
Neural networks (NN)



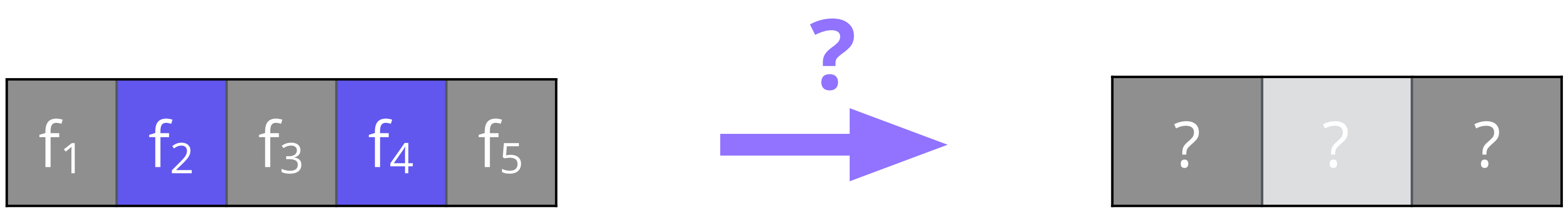
Training set

- ML algorithms require a training set to learn parameters

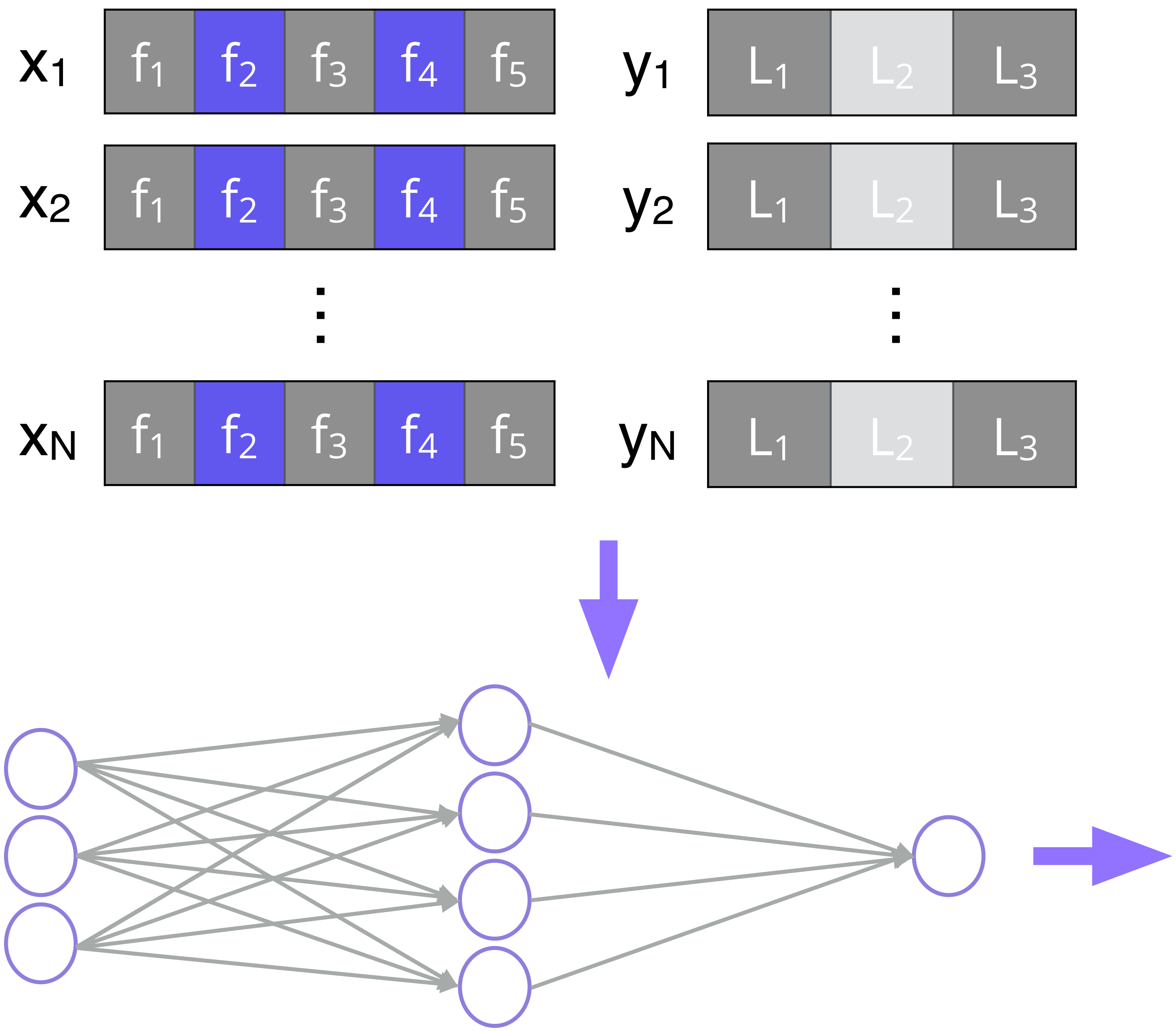
A set of **feature vectors** with their associated **labels**



Given a **new** feature vector predict an **unknown** y



From training data to prediction (learning parameters)



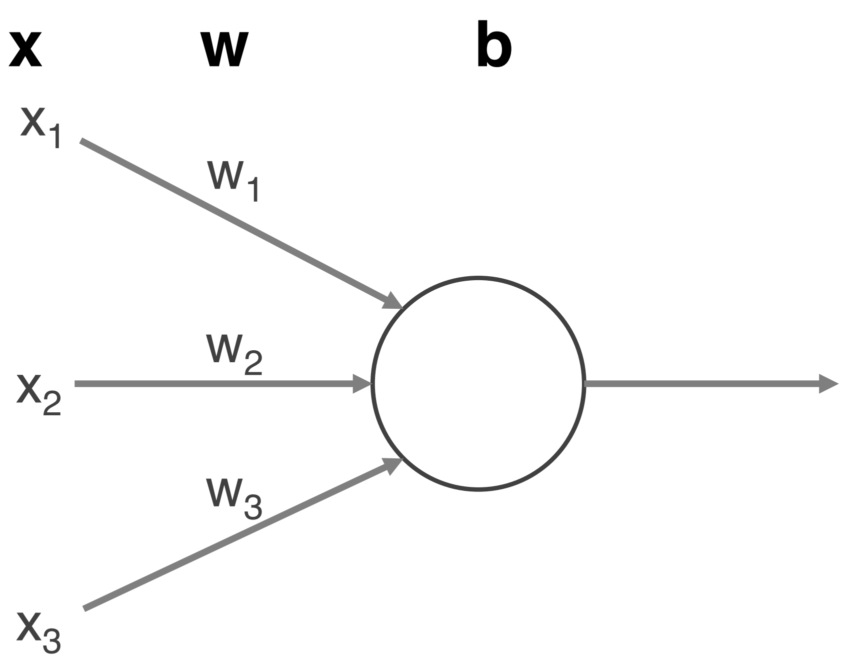
Learn the parameters of the network:

w - the weights
b - bias

Classic Neural Network

- Neural networks are comprised of layers. Each layer contains a number of **neurons.**

- In a traditional NN **neurons** have the following form:

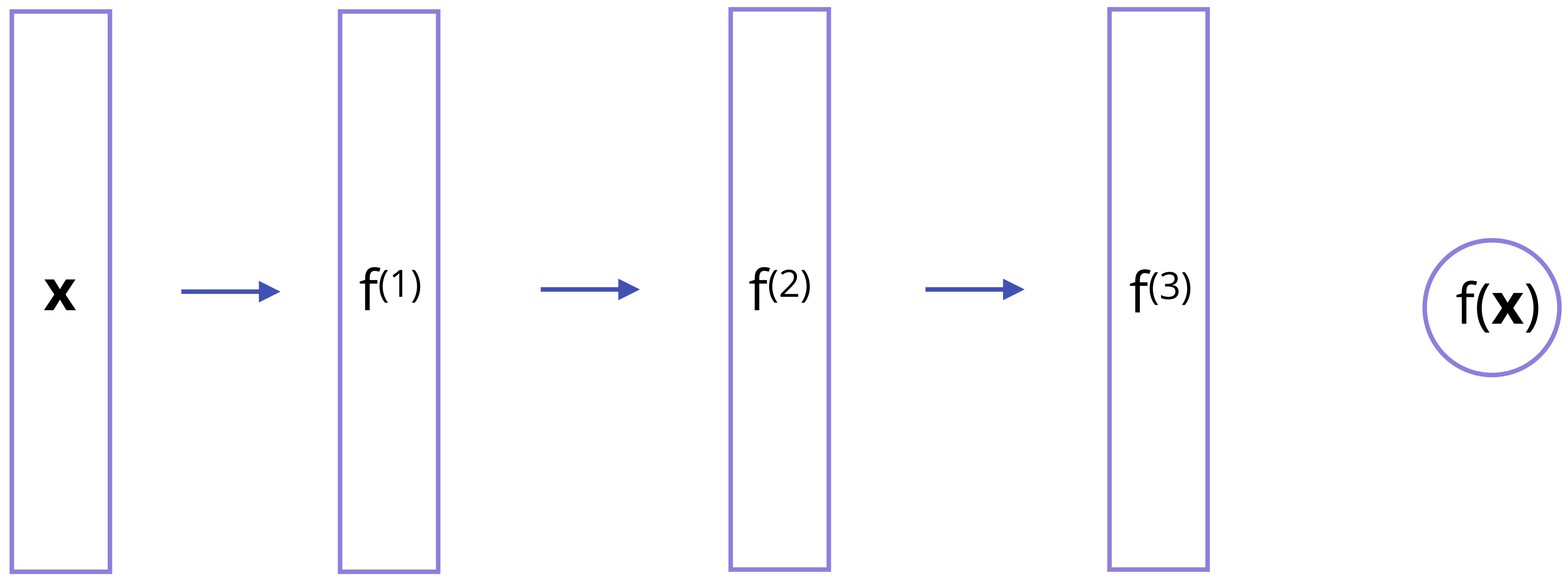


$$\text{output} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} + \mathbf{b} > 0 \\ 0 & \mathbf{w} \cdot \mathbf{x} + \mathbf{b} \leq 0 \end{cases}$$

M. Nielsen, Neural Networks and Deep Learning (2017).

Neural Networks

- Neural networks aim to approximate some a task function f* by composing together many different functions $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$.



- neural network defines a mapping $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$ by learning the value of the parameters $\mathbf{w}$ that best approximate the function f*.

Learning the weights and biases

- Given a set of training data our objective is to learn the best set of **weights (w)** and **biases (b)** that give the best prediction of **y**

- This is an variational problem: Determine the best parameters (**w** and **b**) that **minimize the error** (e.g. find the most accurate prediction)

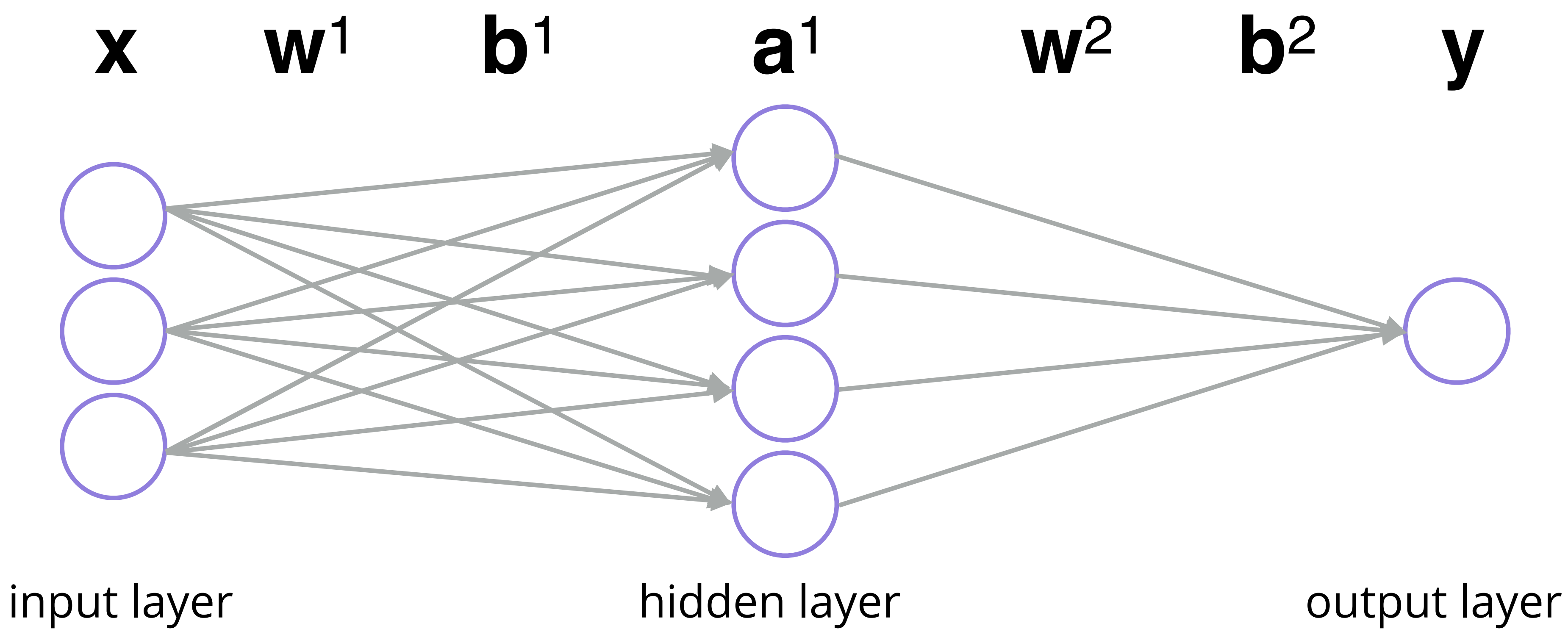
- Learning is done in the following steps:

    - Feed forward

    - Back propagation of error

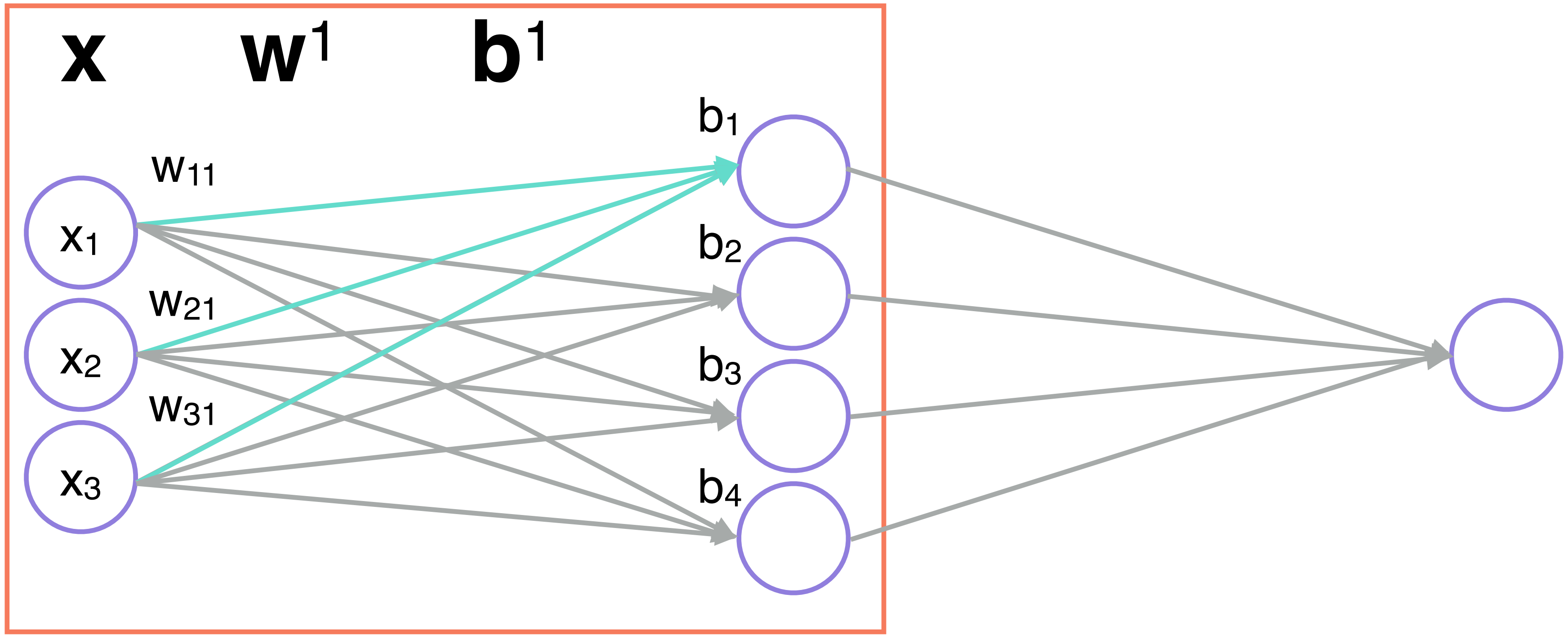
    - Gradient descent

Neural Networks



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{n1} \\ w_{12} & w_{22} & \cdots & w_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{nm} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

NN: feed forward



$$\mathbf{z}^{(1)} = \mathbf{w}^{(1)} \cdot \mathbf{x} + \mathbf{b}^{(1)} = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \\ w_{14} & w_{24} & w_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$= \begin{bmatrix} (w_{11} \cdot x_1 + w_{21} \cdot x_2 + w_{31} \cdot x_3) + b_1 \\ (w_{12} \cdot x_1 + w_{22} \cdot x_2 + w_{32} \cdot x_3) + b_2 \\ (w_{13} \cdot x_1 + w_{23} \cdot x_2 + w_{33} \cdot x_3) + b_3 \\ (w_{14} \cdot x_1 + w_{24} \cdot x_2 + w_{34} \cdot x_3) + b_4 \end{bmatrix}$$

Feed forward: Activation function

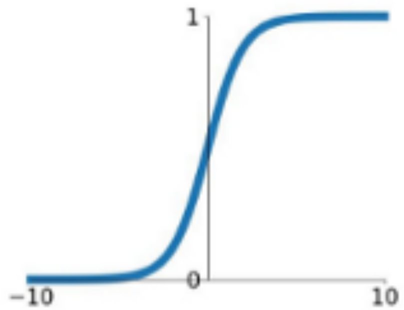
- During learning want small changes in **w** or **b** to result in small changes to **z** (the output).
- For a traditional neuron this is not the case, since a small change in either can flip the neuron.
- Therefore, must pass **z** to a special function known as the activation function. It has the following properties:
  - Has a derivative that can be computed
  - Is non-decreasing
  - Has horizontal asymptotes at 0 and 1 (or -1 and 1)

M. Nielsen, Neural Networks and Deep Learning (2017).

Activation functions

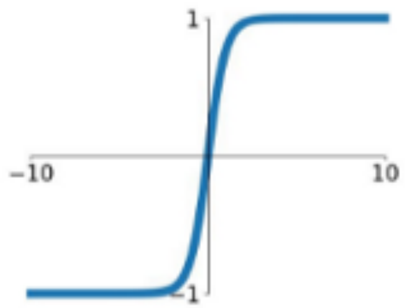
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

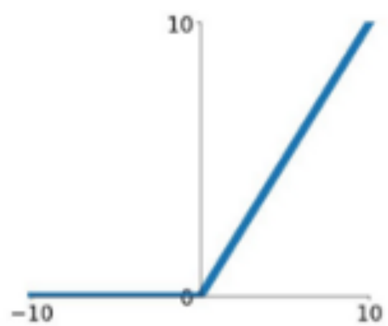
$$\tanh(x)$$



Rectified Linear Unit

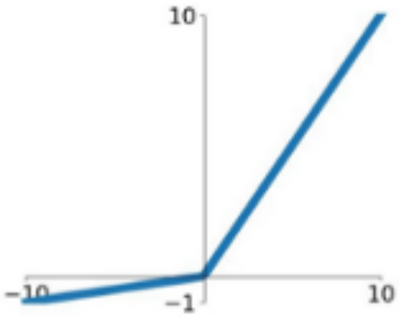
ReLU

$$\max(0, x)$$



Leaky ReLU

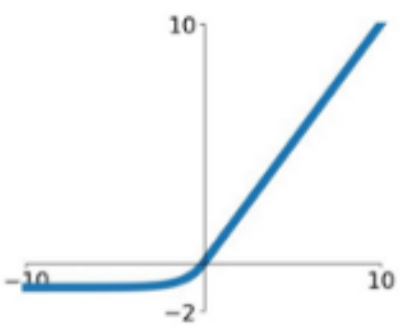
$$\max(0.1x, x)$$



Exponential Linear Unit

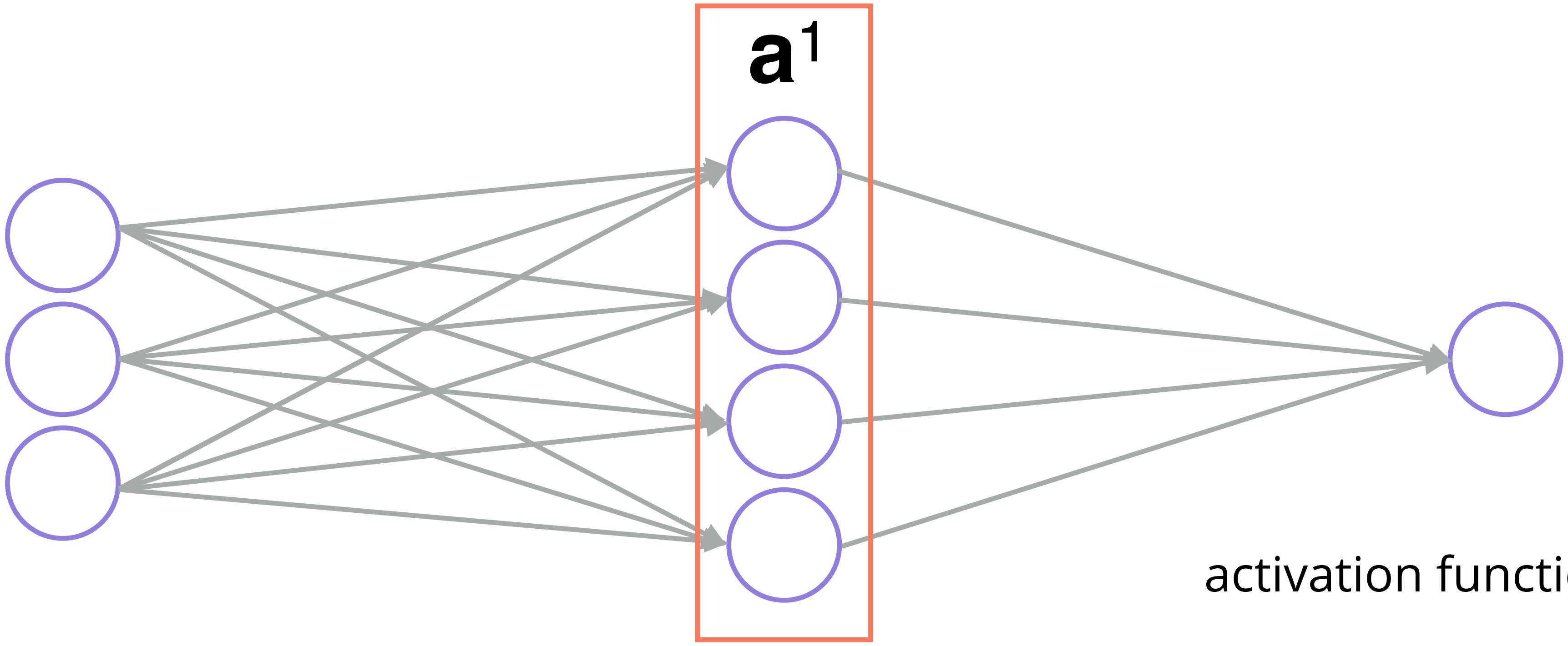
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

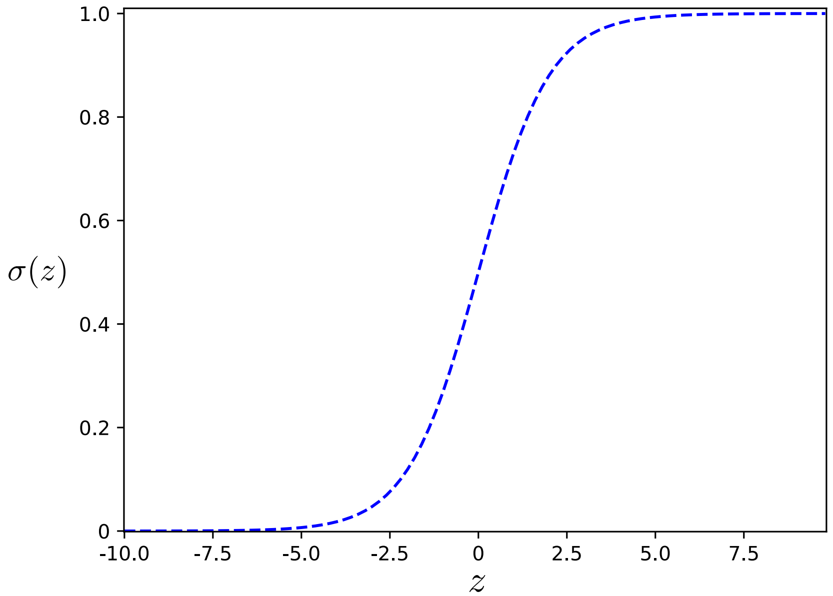
NN: feed forward



$$\mathbf{z}^{(1)} = \mathbf{w}^{(1)} \cdot \mathbf{x} + \mathbf{b}^{(1)}$$

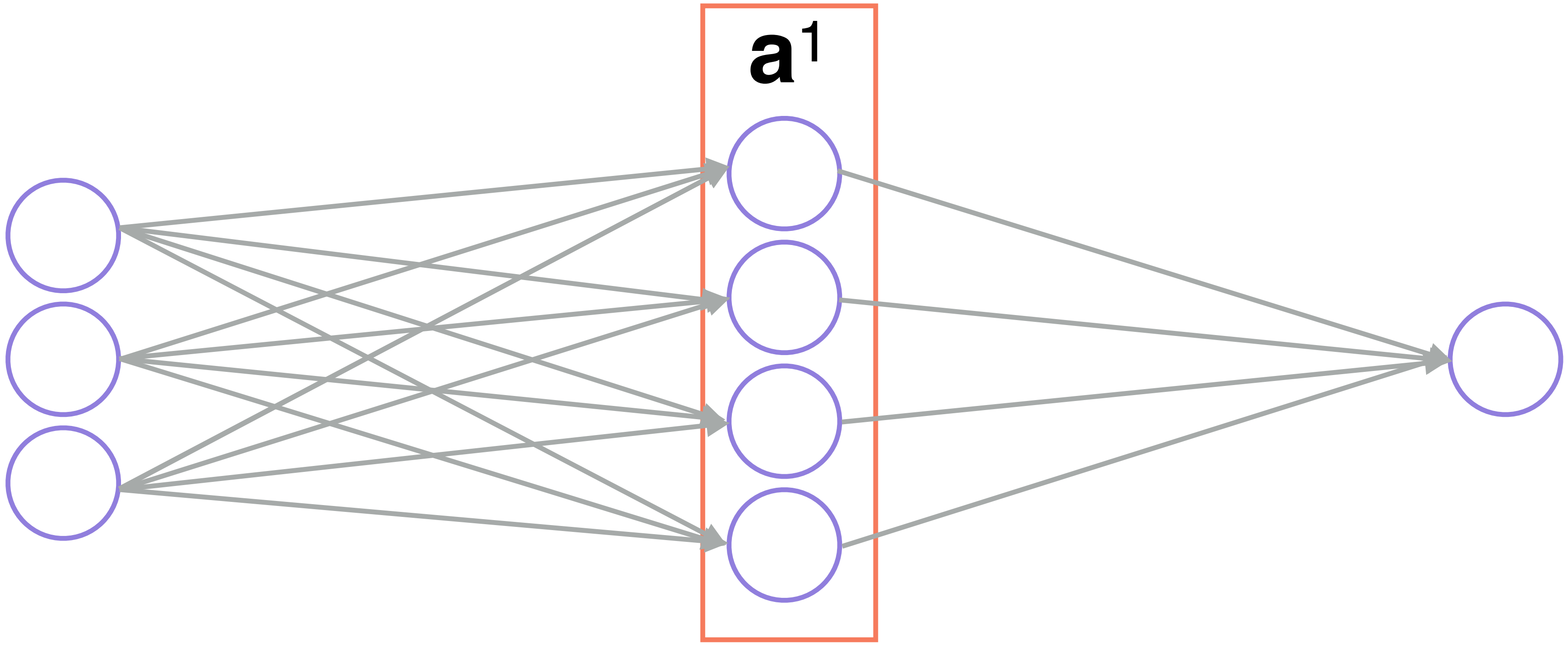
activation function **Sigmoid**

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



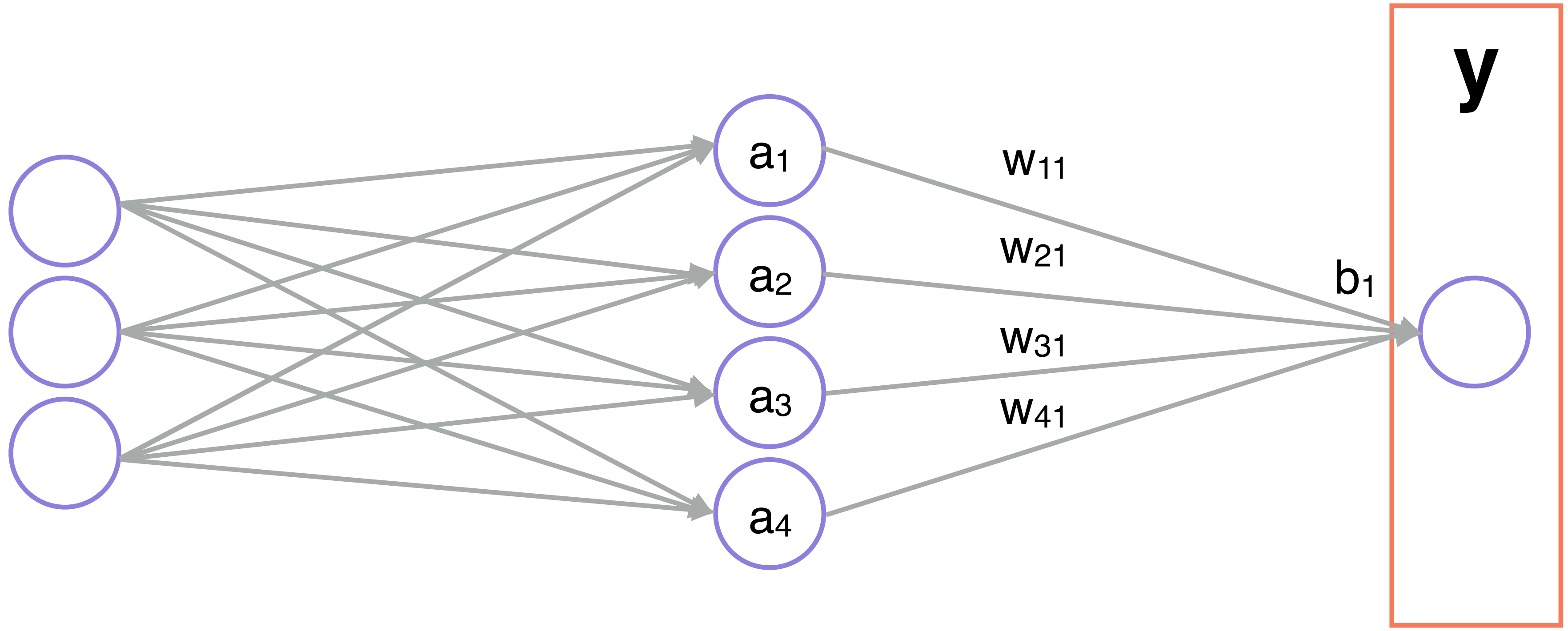
$$w + \Delta w \rightarrow y + \Delta y$$

NN: feed forward



$$\mathbf{a} = \sigma(\mathbf{w}^{(1)} \cdot \mathbf{x} + \mathbf{b}^{(1)}) = \sigma(\mathbf{z}^{(1)}) = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

NN: feed forward



$$\mathbf{y} = \sigma(\mathbf{w}^{(2)} \cdot \mathbf{a} + \mathbf{b}^{(2)})$$

$$= \sigma\left(\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + b_1\right)$$

NN: Determining prediction accuracy

- Now we have a predicted value, how do we determine how good it is?
- This is known as the cost function. Many forms exist however the simplest is the mean squared error (MSE)

$$C(w,b) = \frac{1}{2n}\sum_i ||y_i - \tilde{y}_i||_2^2 = \frac{1}{2n}\sum_x ||\mathbf{y}(x) - \mathbf{a}^L(x)||^2 = \frac{1}{n}\sum_x C_x$$

Minimization of error

- Want to update the weights and biases in order to drive the network towards the f(x) that best approximates f*

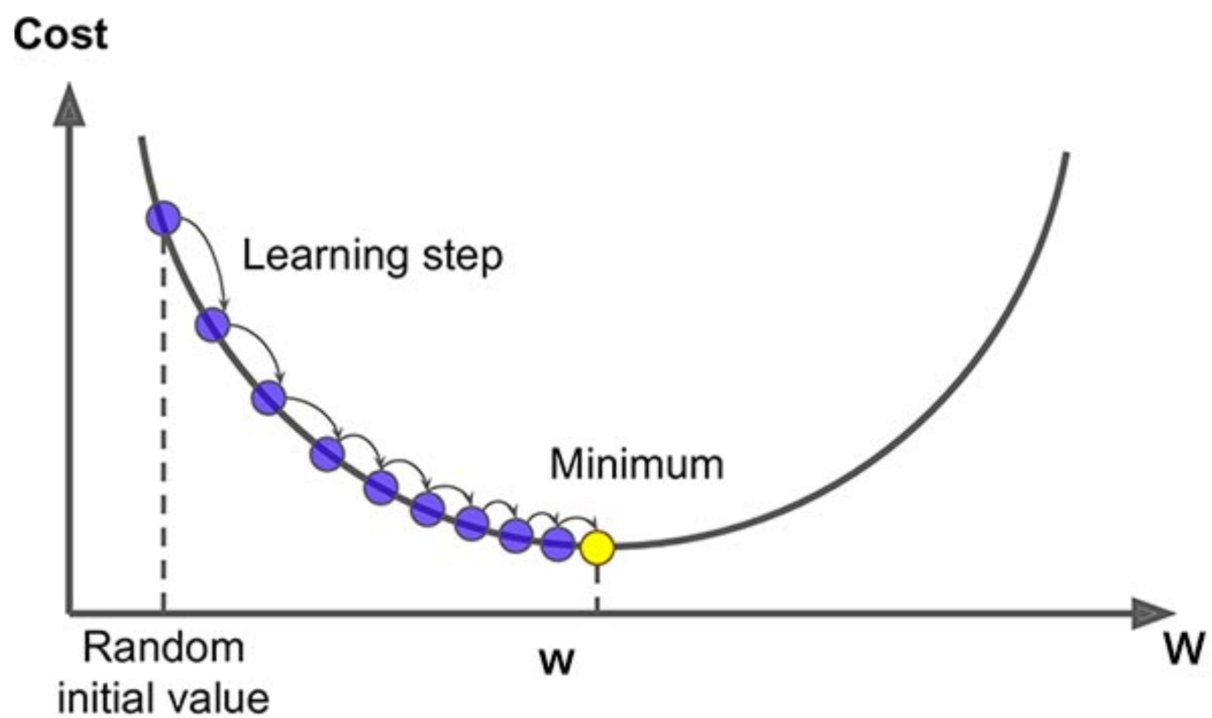
- The metric to calculate the error in our network is the cost function

$$C(w,b) = \frac{1}{2n}\sum_i ||y_i - \tilde{y}_i||_2^2 = \frac{1}{2n}\sum_x ||\mathbf{y}(x) - \mathbf{a}^L(x)||^2 = \frac{1}{n}\sum_x C_x$$

- Want to minimize the cost (error) by changing the weights and biases.

- Therefore, we need two things: 1) way to update our weights and biases based on the error. 2) way to minimize the cost (error).

Gradient Descent



https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/

- Recall this is a variational problem in which we want to pick the best **w** and **b** such that we minimize the error.
- In other words we want to find the minimum which is done using gradient descent

Gradient Descent cavets

1. Before we calculate the gradient we need some way to relate the error to the **w** and **b** such that we can update them based off the gradient

2. Calculating the total gradient for the entire feature space is expensive! This will require every entry of the training set (yikes)

Solutions

1. Use something known as back propagation to update the **w** and **b**

2. Use stochastic gradient descent to approximate the gradient from a smaller random batch

M. Nielsen, Neural Networks and Deep Learning (2017). http:// neuralnetworksanddeeplearning.com/.

NN: Back propagation

- Want to minimize the cost (error) by changing the weights and biases.

$$C = \frac{1}{2n}\sum_i ||y_i - \tilde{y}_i||_2^2 = \frac{1}{2n}\sum_x ||\mathbf{y}(x) - \mathbf{a}^L(x)||^2 = \frac{1}{n}\sum_x C_x$$

- Want to back propagate error to update the weight and bias.
- Minimize **C** using gradient descent.
- How do we calculate the gradient? Also how do we update the weights?
- Chain rule to the rescue!
- **Starting point:** the error for some input in the network is defined by:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}.$$

I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (MIT Press, 2016).
M. Nielsen, Neural Networks and Deep Learning (2017).

Back propagation

- Derive an expression for the error at the final layer:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \xrightarrow{\text{When j=k else 0}} \delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$$

$$\frac{\partial C}{\partial a_j^L} = a_j^L - y_j$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L).$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z)).$$

In Matrix notation:

$$\delta^L = (\mathbf{a}^L - \mathbf{y}) \odot \sigma(\mathbf{z}^L)$$

I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (MIT Press, 2016).
M. Nielsen, Neural Networks and Deep Learning (2017).

Back propagation

- Derive an expression for the error at all previous layers:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}. \quad \longrightarrow \quad \delta_j^l = \sum_k \delta_j^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l}.$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = \frac{\partial}{\partial z_j^l} \left(\sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \right)$$

$$= w_{kj}^{l+1} \sigma'(z_j^l).$$

In Matrix notation:

$$\delta^l = \left[(\mathbf{w}^{l+1})^{\mathrm{T}} \delta^{l+1} \right] \odot \sigma(\mathbf{z}^l)$$

I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (MIT Press, 2016).
M. Nielsen, Neural Networks and Deep Learning (2017).

Back propagation computing gradients

- Finally we can compute the gradients!

$$\frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial b^l_j}.$$

Term equals to 1
Also recall:

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}.$$

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_k}\frac{\partial z^l_k}{\partial w^l_{jk}}.$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

$$\frac{\partial C}{\partial w^l_{jk}} = \delta^l_j a^{l-1}_k$$

In Matrix notation:

$$\frac{\partial C}{\partial b} = \delta.$$

$$\frac{\partial C}{\partial w} = \delta^l \mathbf{a}^{l-1}$$

I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning (MIT Press, 2016).
M. Nielsen, Neural Networks and Deep Learning (2017).

NN: Learning via Gradient Descent.

- Calculating the total gradient is expensive, so must approximate.
- Use stochastic gradient descent to break training set into mini-batches $\{ x_t \}$ of size m, such that the gradient is:

$$\nabla C = \frac{1}{n} \sum_{x} \nabla C_x \approx \frac{1}{m} \sum_{x_t} \nabla C_{x_t}$$

- For each mini-batch, the components of the gradient are calculated using back propagation.
- Weights/biases updated via:

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_{x_t} \frac{\partial C_{x_t}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_{x_t} \frac{\partial C_{x_t}}{\partial b_l}$$

M. Nielsen, Neural Networks and Deep Learning (2017).

NN: Learning via Gradient Descent.

split training set into mini-batches $\{ x_t \}$

feedforward y

back propagation

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}.$$

update **w**, **b**

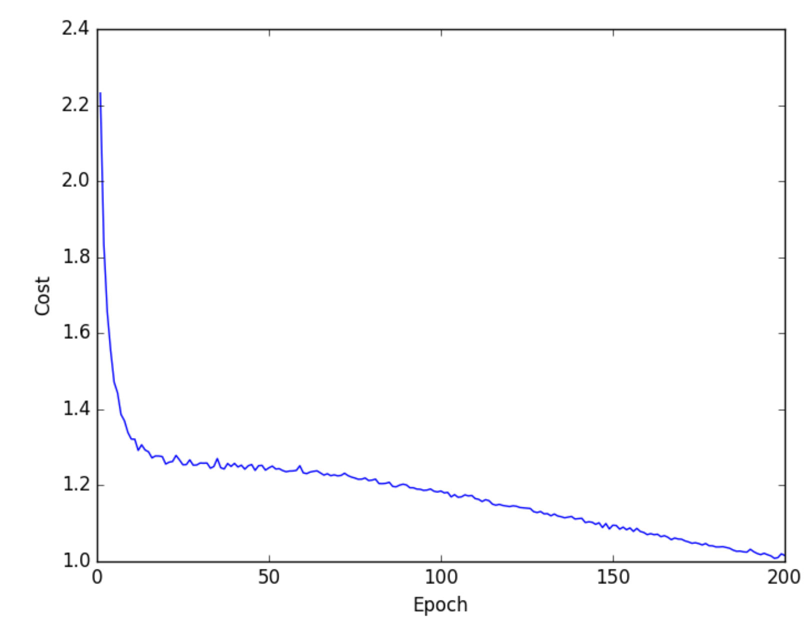
$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_{x_t} \frac{\partial C_{x_t}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_{x_t} \frac{\partial C_{x_t}}{\partial b_l}$$

epochs++

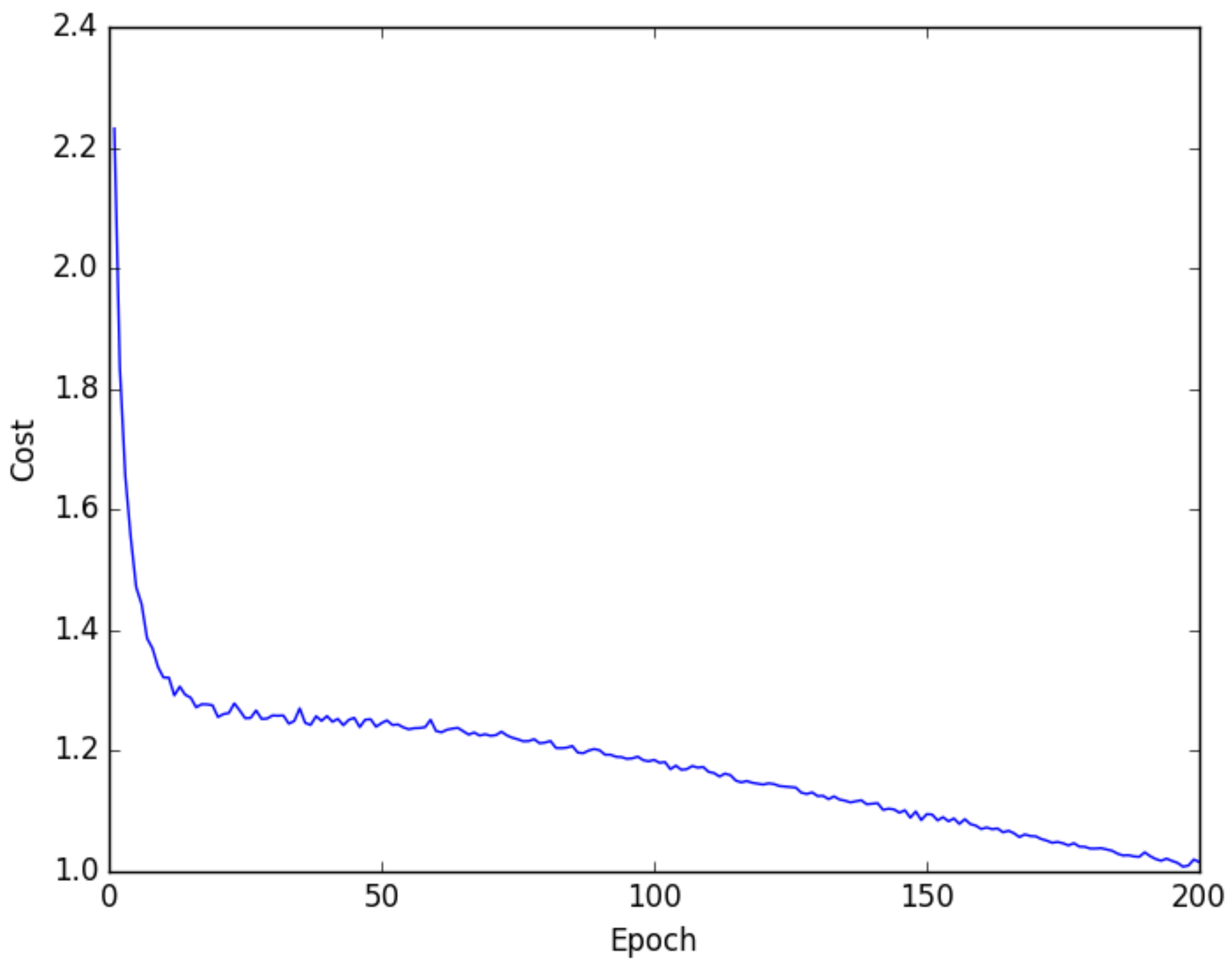
if # epochs < max epoch

trained model



NN: Learning via Gradient Descent.

- The entire process is known as an epoch.
- This is then repeated over multiple epochs until the cost reaches a minimum.



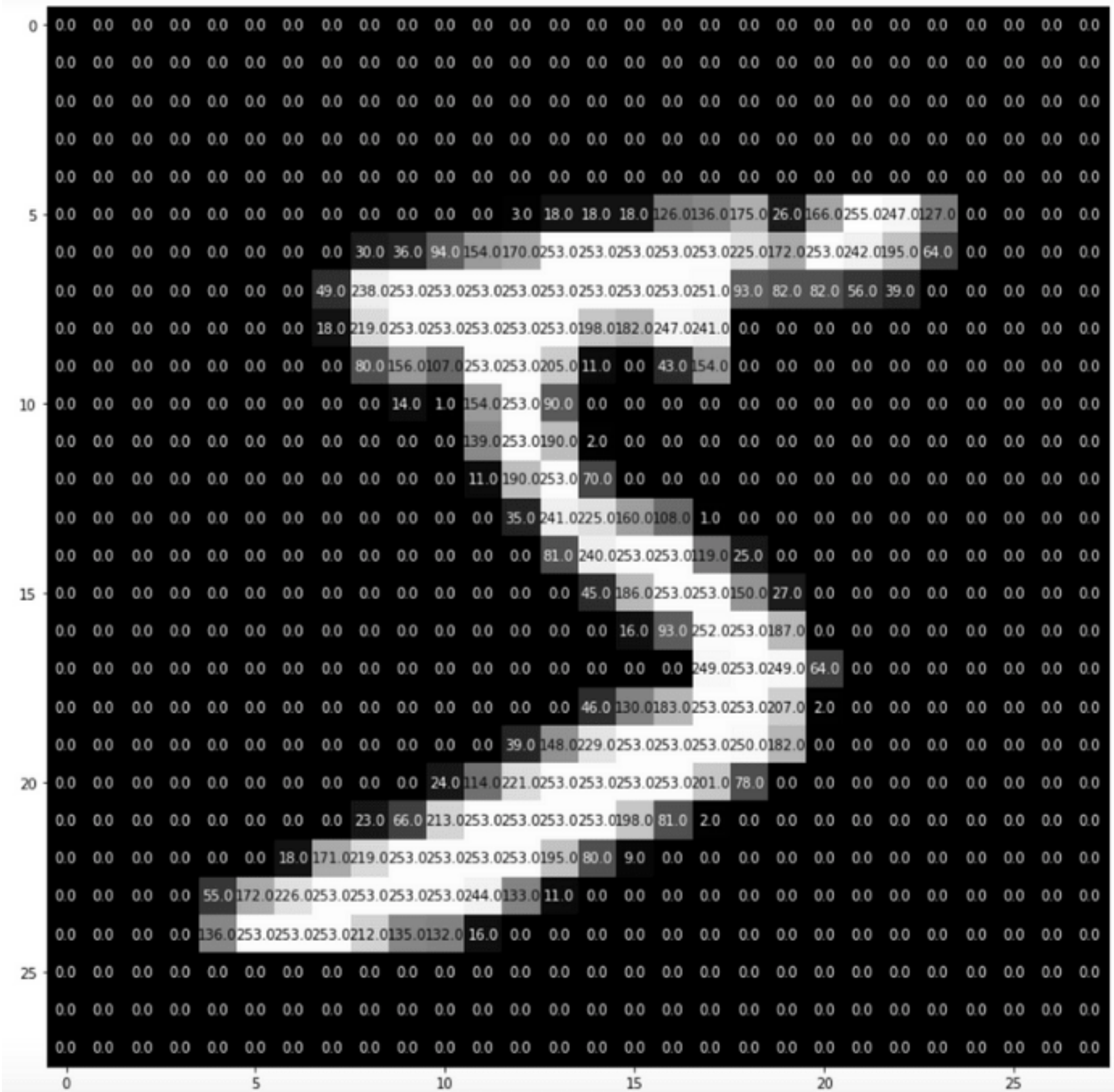
Building a NN: Training set

- **MNIST**: Modified National Institute of Standards and Technology is a collection of hand written digits (0 - 9)
- **Task:** Create a model that can classify the digit (e.g. optical character recondition OCR)

dataset: http://yann.lecun.com/exdb/mnist/

Building a NN: Features

- Use the pixels of the image as features
- MNIST images are 28 x 28 so we will flatten to create a **784** dimensional feature vector.

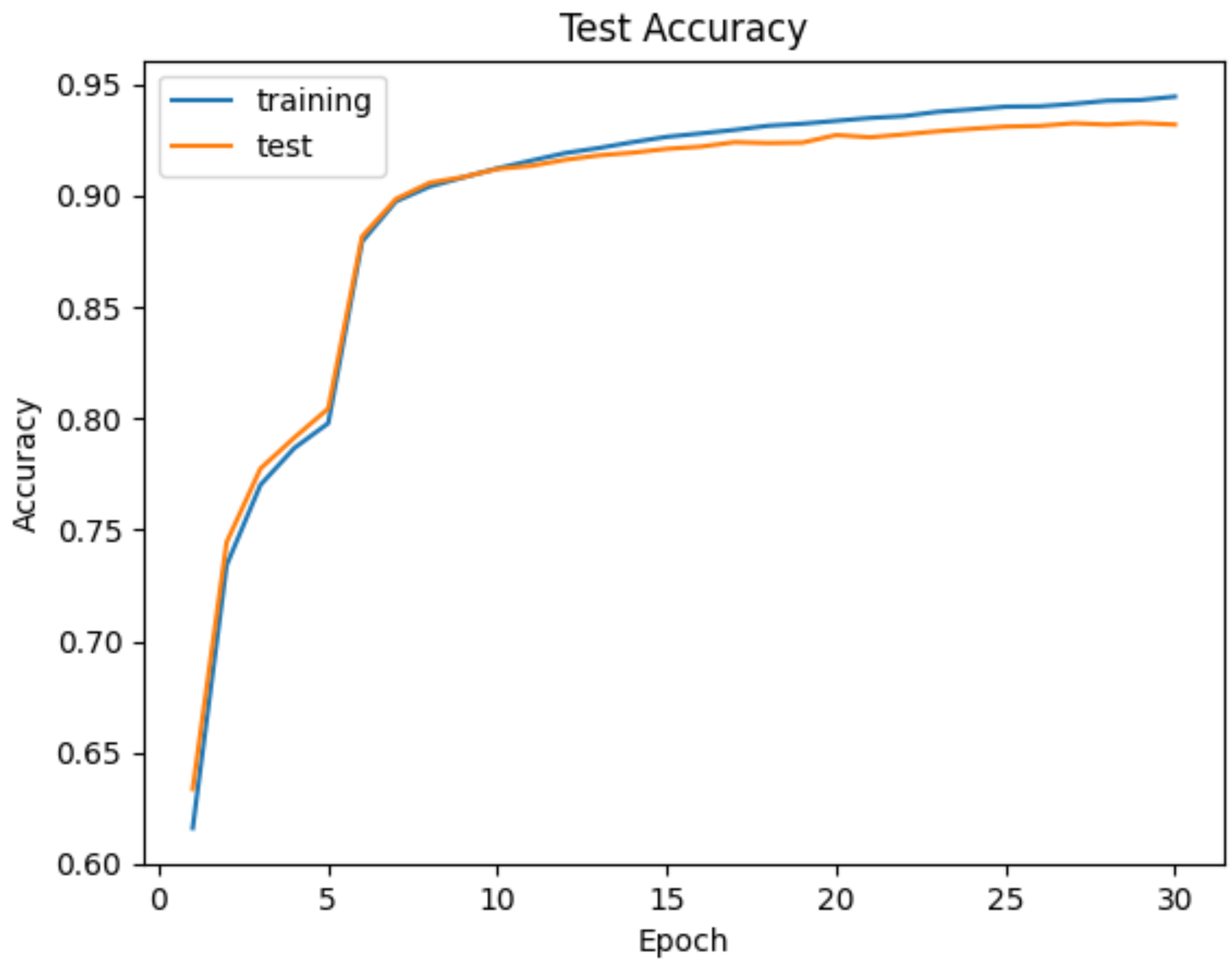


https://medium.com/comet-ml/real-time-numbers-recognition-mnist-on-an-iphone-with-coreml-from-a-to-z-283161441f90

Git Repo

https://github.com/ericgossett/Intro-to-Neural-Networks-Tech-Talk

Results



Test Accuracy
training
test
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25
30
Epoch