Artificial Neural Networks P1

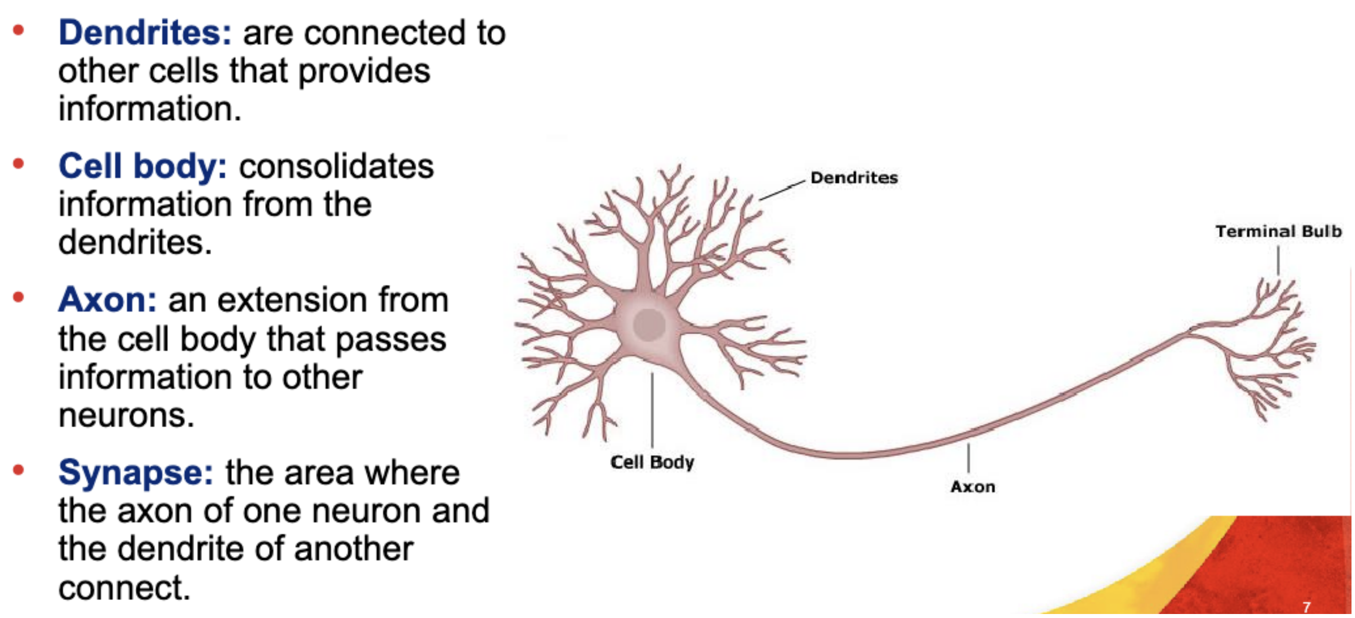
ENSF 444

Introduce biological neurons, used to motivate artificial neurons

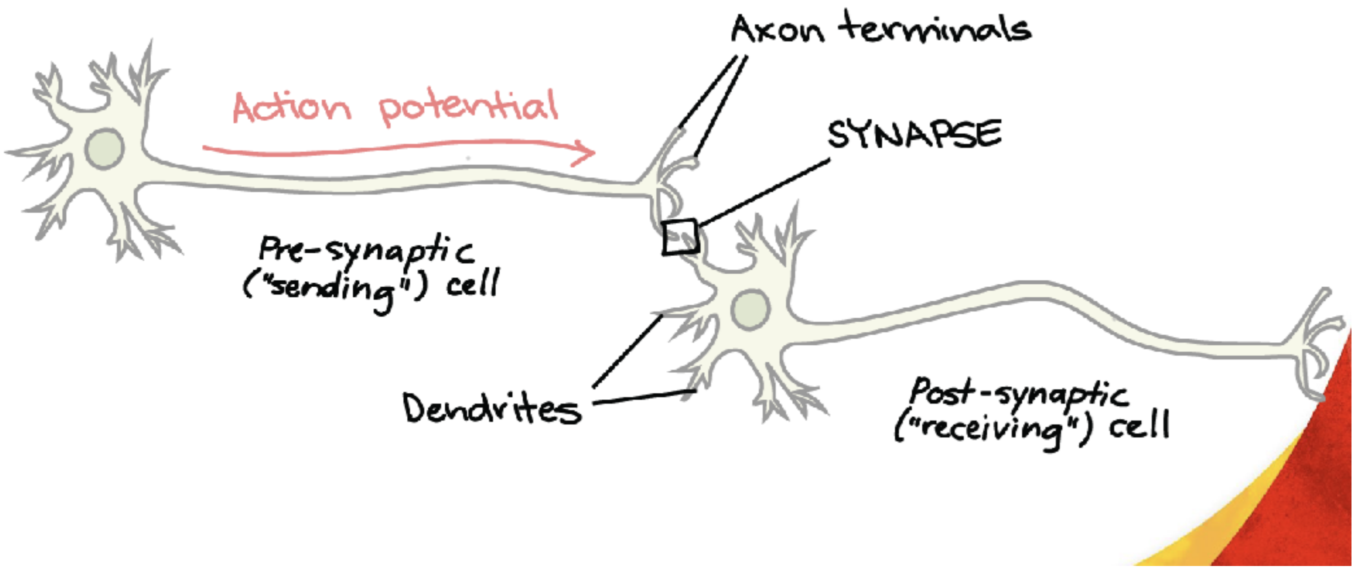
Introduce Artifical Neurons

Using Pigeon to train example

Neuron Anatomy (simplified)



Synapse



Artificial Neural Networks vs Biological

1. Size: Our brains has about 86 billion neurons and 100-1000 trillion synapses
2. Topology biological neurons can fire asynchronously
3. Speed: Biological neurons can fire 200 times a second
4. Learning: the brain appearrs to have different learning mechanisms

Fault Tolerance

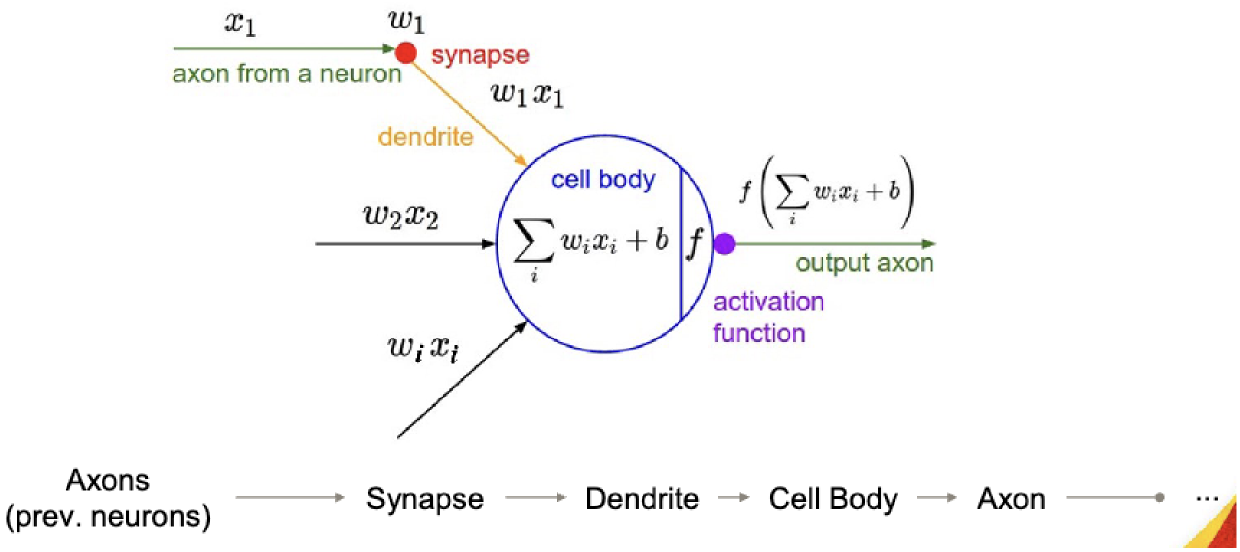
Biological neurons are much more redundant and brain can adapt to severe damage by relearning tasks in completely different areas

Energy Usage

It's been estimated the human brain consumes ~20W ○ Most high-end GPUs use > 250W, and training a model often uses 4-8 GPUs even for standard models, thousands of high-end GPUs for LLMs.

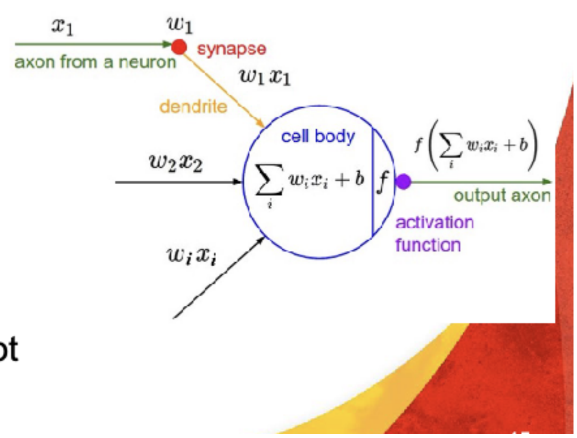
Signals

Action potentials in neurons are binary, triggered or not triggered, although neurons may use firing frequency instead to encode information



An over view of an Artificial Neuron

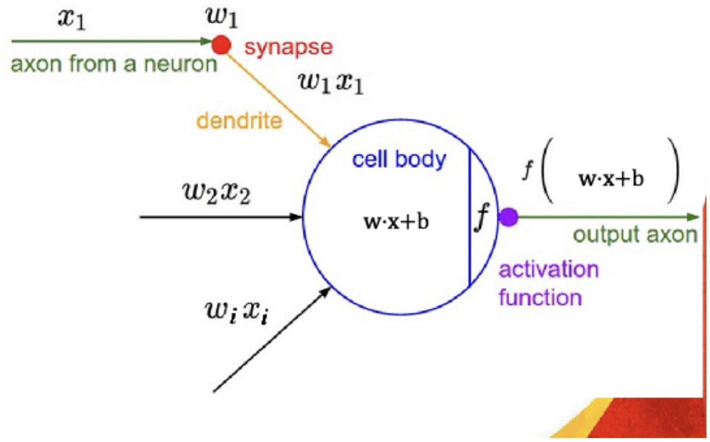
- x_i is an **input**:
  - e.g. a pixel in biopsy image
- w_i is the **weight** for input x_i:
  - weighting we learn for this particular input
- b is the **bias**:
  - a weight we learn with no input
- f is the **activation function**:
  - function that determines how our output changes with the "cell body" potential (i.e. sum of all weight-input products)
- y is the **output**:
  - e.g. binary classification problem, number from which we infer cancer or not cancer



Now putting into vector notation

In vector notation this is even easier:

- **x** = (x_1, …, x_n) is an **input vector**:
  - e.g. a biopsy image
- **w** = (w_1, …, w_n) is the **weight vector**:
- b is the bias (a scalar)
- f is the **activation function**
- y is the **output** (a scalar)



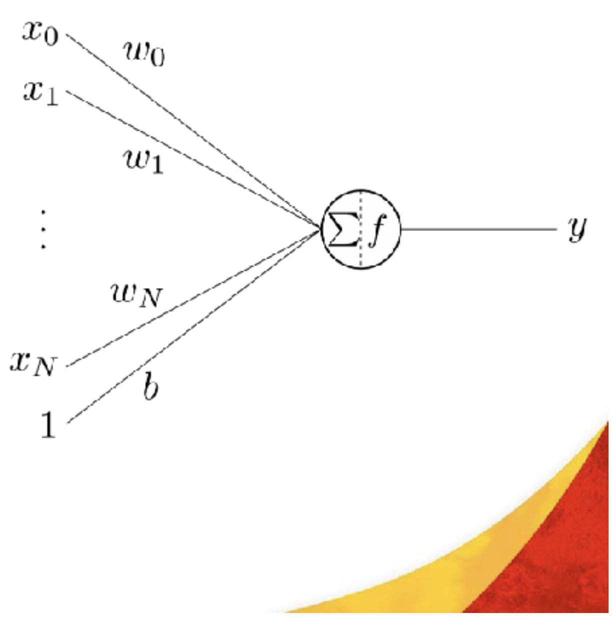
Making it more easier

In vector notation this is even easier:

- **x** = (x_0, …, x_n) is an **input vector**:
  - e.g. a biopsy image
- **w** = (w_0, …, w_n) is the **weight vector**:
- b is the bias (a scalar)
- f is the **activation function**
- y is the **output** (a scalar)

$$y = f(\mathbf{w} \cdot \mathbf{x} + b)$$

This equation looks vaguely familiar...



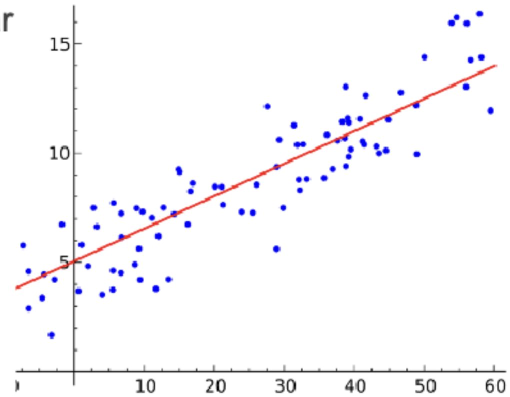
Neuron with Linear Activation Function

Neuron with a Linear Activation Function

$$y = f(\mathbf{w} \cdot \mathbf{x} + b)$$

Assume that our activation function is a simple linear function: $f(x) = x$, then:

$$y = \mathbf{w} \cdot \mathbf{x} + b$$



- This is a special equation, if we write it out for 2D:

$$y = w_0x_0 + w_1x_1 + b$$

- Recall general equation of line: $Ax + By - C = 0$
- The bias is related to the offset of the line from the origin

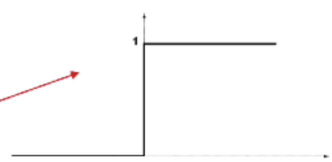
Early Activation Functions: Perceptrons

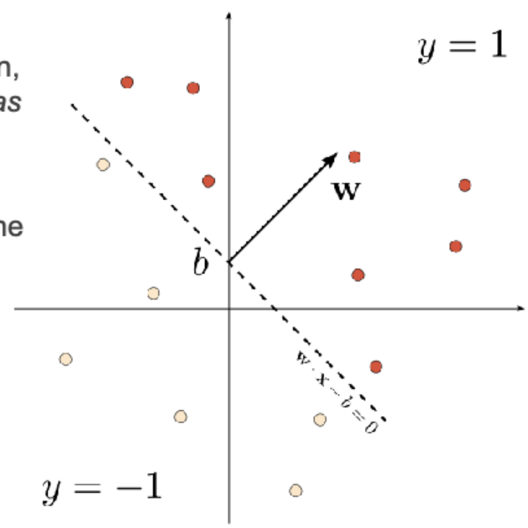
$$y = \mathbf{w} \cdot \mathbf{x} + b$$

- In fact, $y = \mathbf{w} \cdot \mathbf{x} + b$ is a generalized line for any dimension, known as a *hyperplane, e.g. for 2D plane on right, just as we've seen already with linear classification models*
- First artificial neurons (1943-70s) used a simple binary activation function based on which side of the hyperplane the input is, including:

$$f(x) = \text{sign}(x)$$

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$$





- This is called the ***decision boundary***
- *Note the decision boundary is perpendicular to the vector/line in 2D defined by the neuron, not the line itself*

Sigmoid Activation Function

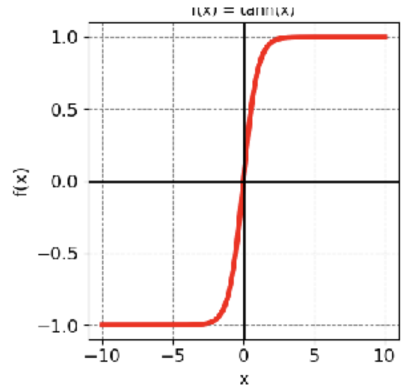
Sigmoid activation functions were the most common before 2012 because:

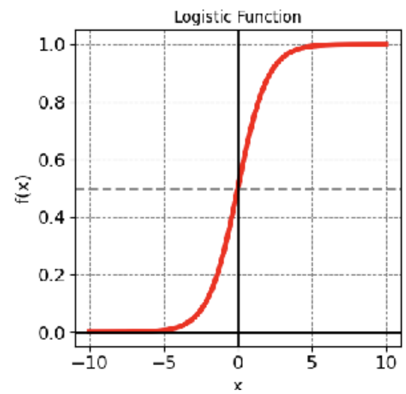
- easily differentiable, smooth, continuous
- range between [-1, 1] or [0, 1]

There are *many* sigmoid functions, the most common are:

$$f(x) = \tanh(x) \qquad \text{(hyperbolic tangent)}$$

$$f(x) = \frac{1}{1 + e^{-x}} \qquad \text{(logistic function)}$$





How do we **learn** the **weights** (and bias) of a neural network?

We can use our prediction error to decide how to change weights:

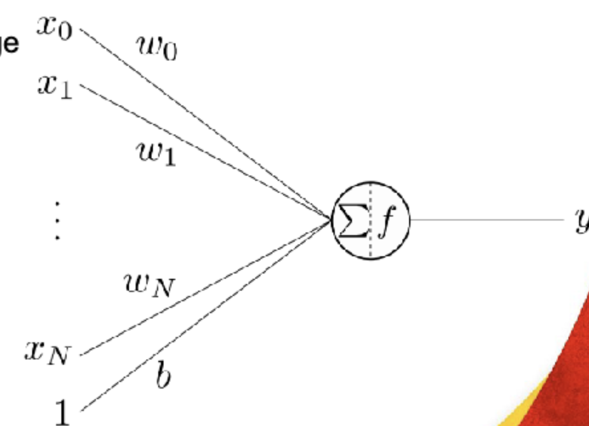
1. Make a **prediction** for some input data **x**, with a known correct output, i.e. ground truth or label: t

   input: **x**, output: y, label: t

1. Compare the correct output with our predicted output (e.g. squared error):

   E = loss(y, t) = $(y - t)^2$

1. **Adjust the weights/bias** to make the prediction closer to the ground truth, i.e. minimize error
2. **Repeat** until we have an acceptable level of error



21

Training a Neuron - Forward Pass

During the forward pass, input data is fed into the network, and the model generates predictions.

Neural Network Layer (Vector, Matrices, and Tensors)

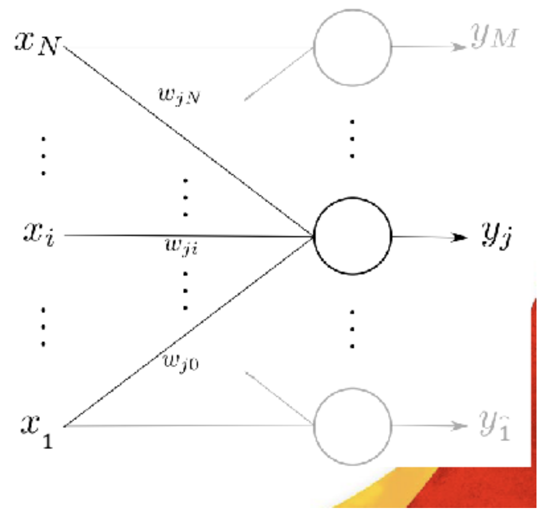
e.g., a neural network *layer* with two neurons:

$$y_1 = f(\mathbf{w}_1 \cdot \mathbf{x} + b_1),$$
$$y_2 = f(\mathbf{w}_2 \cdot \mathbf{x} + b_2)$$

Can represent NN layer easier with a *weight matrix*, e.g. where each neuron's weight vector is a **row** of the weight matrix **W** and the input is a **column** vector **x**:

$$\mathbf{y} = f(\mathbf{Wx} + \mathbf{b})$$

- This is important to know about because you will spend a lot of time debugging the dimensions of your tensors (where we add other dimensions such as batch size also)!
- Calculating **y** given the input **x** is known as ***inference***



Loss / Error Function

Neural Network Single Layer: Inference

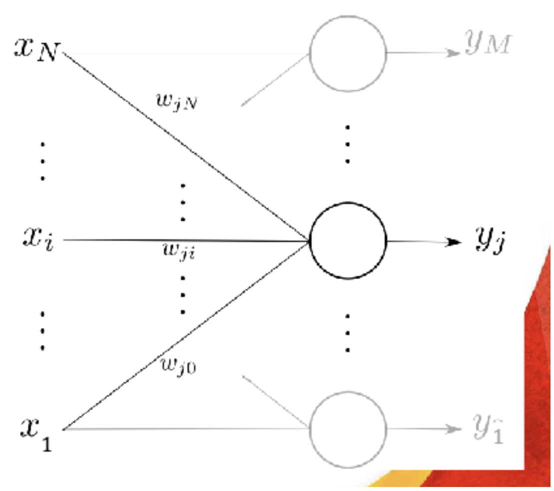
Neural Network Single Layer: Inference

Calculating the error for a neural network output layer is easy:

$$E = \text{Loss}(y,\ t) = ||\, y - t \,||^2$$

However, we want to calculate this error over **all training samples**

- We must use an error of the form of the average error over all N training samples (for backpropagation)
- For example Mean Squared Error (MSE):



Error function depends on the problem

For regression

For **regression problems**, we use mean squared error (MSE) defined as:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - t_n)^2$$

Where ***N*** is the number of training samples, ***y*** is the model predicted output and ***t*** is the target label

For Classification

For **classification problems**, we use cross entropy (CE) defined as:

$$CE = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\log(y_{n,k})$$

Where ***N*** is the number of training samples, **K** is the number of classes, ***y*** is the model predicted output and ***t*** is the target label