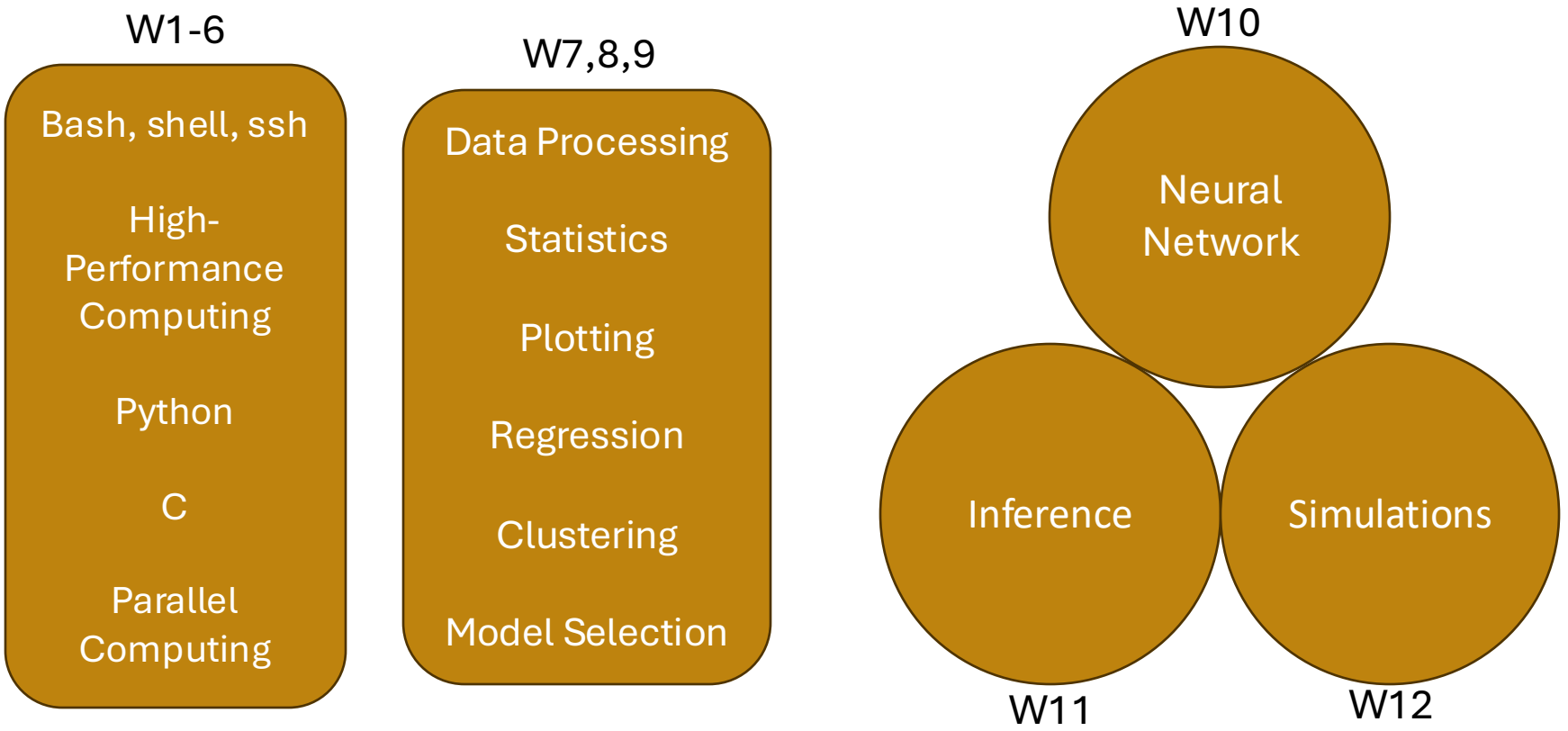
ASTR4004/8004 Computational Astronomy

Neural Networks

Yuxiang.Qin@anu.edu.au
Duffield Building D.115

Australian National University



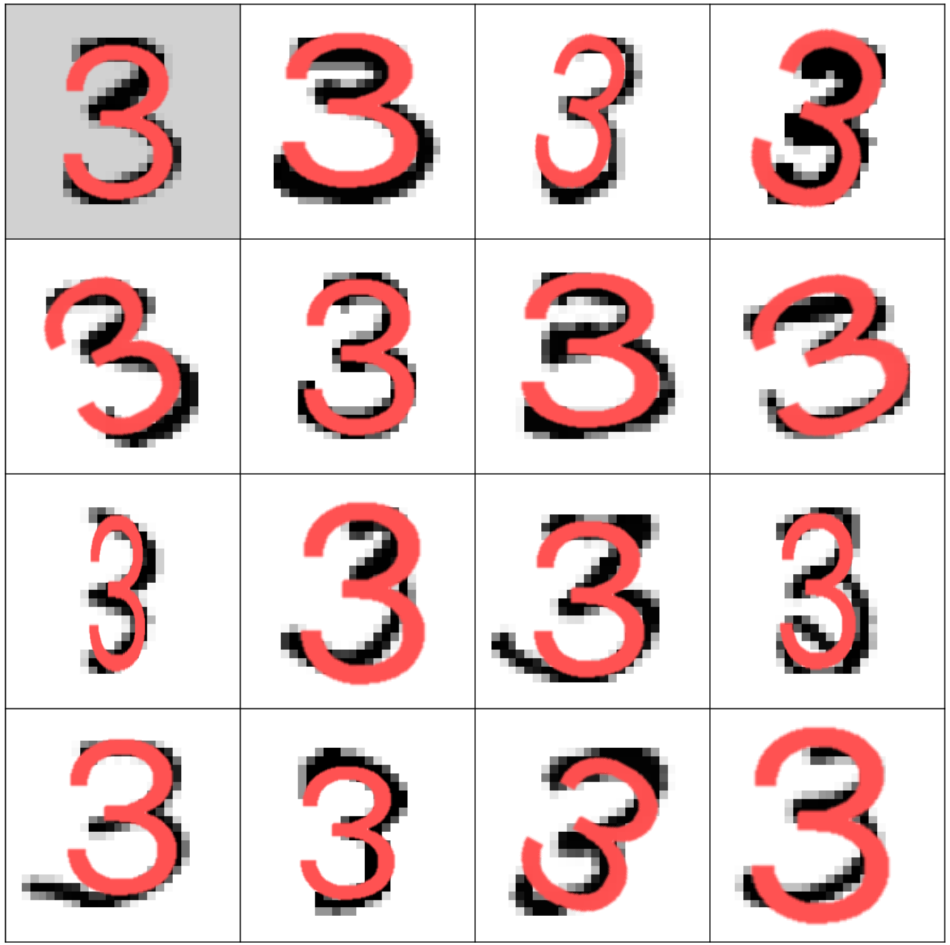
https://www.surveymonkey.com/r/NDKK3GR

ESTIMATED TIME TO COMPLETE 2 Minutes

Quiz time

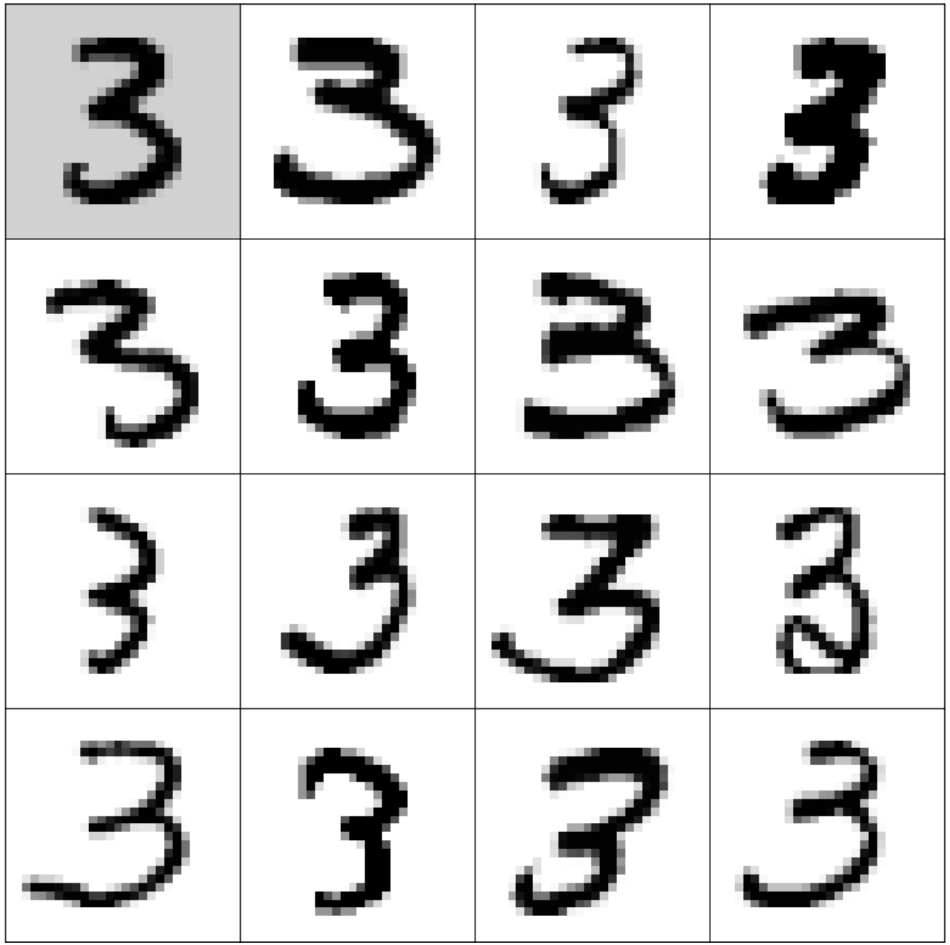
What are these?

3



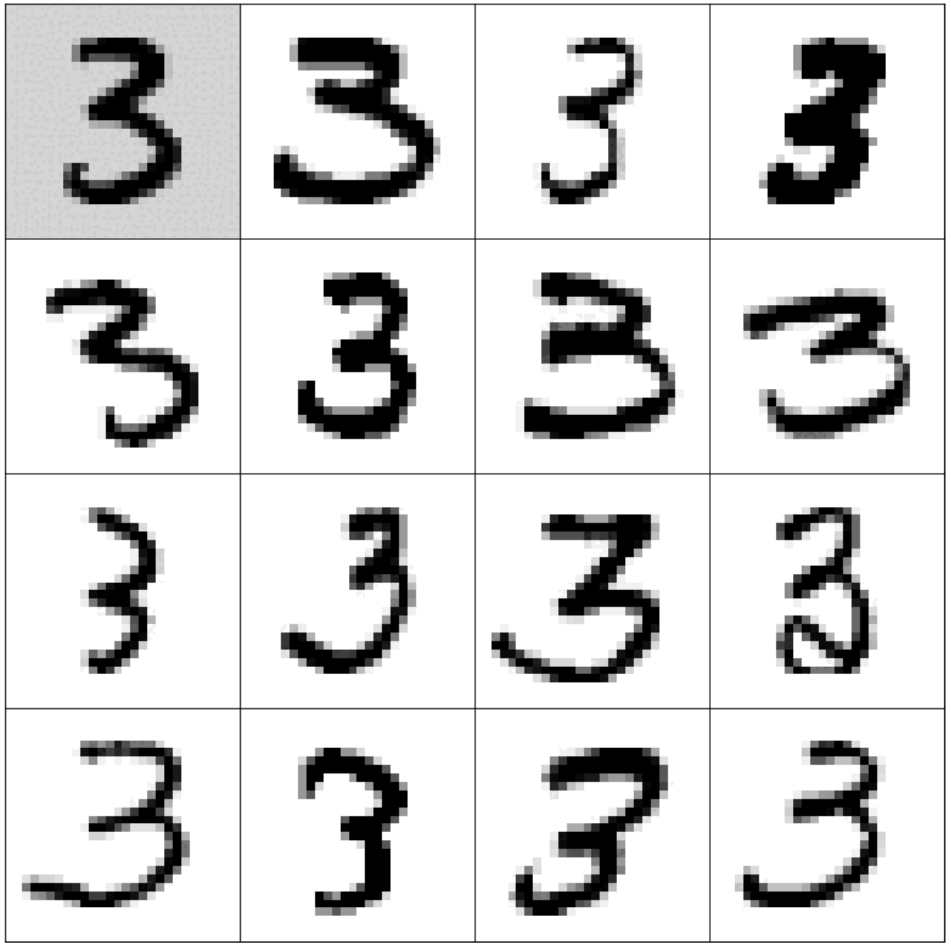
Quiz time

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.5	0.9	1	1	0.7	0	0	0	0	0
0	0	0	0	0.4	0.3	0.4	0.5	1	0.5	0	0	0	0
0	0	0	0	0	0	0	0.6	1	0.5	0	0	0	0
0	0	0	0	0	0.3	0.9	0.9	0.5	0	0	0	0	0
0	0	0	0	0	1	1	0.9	0.4	0	0	0	0	0
0	0	0	0	0	0.1	0.1	0.5	0.9	0.6	0	0	0	0
0	0	0	0	0	0	0	0	0.2	1	0.4	0	0	0
0	0	0	0.1	0.4	0	0	0	0.4	1	0.4	0	0	0
0	0	0	0.4	0.7	0.3	0.5	0.8	1	0.7	0	0	0	0
0	0	0	0	0.6	1	1	0.8	0.4	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

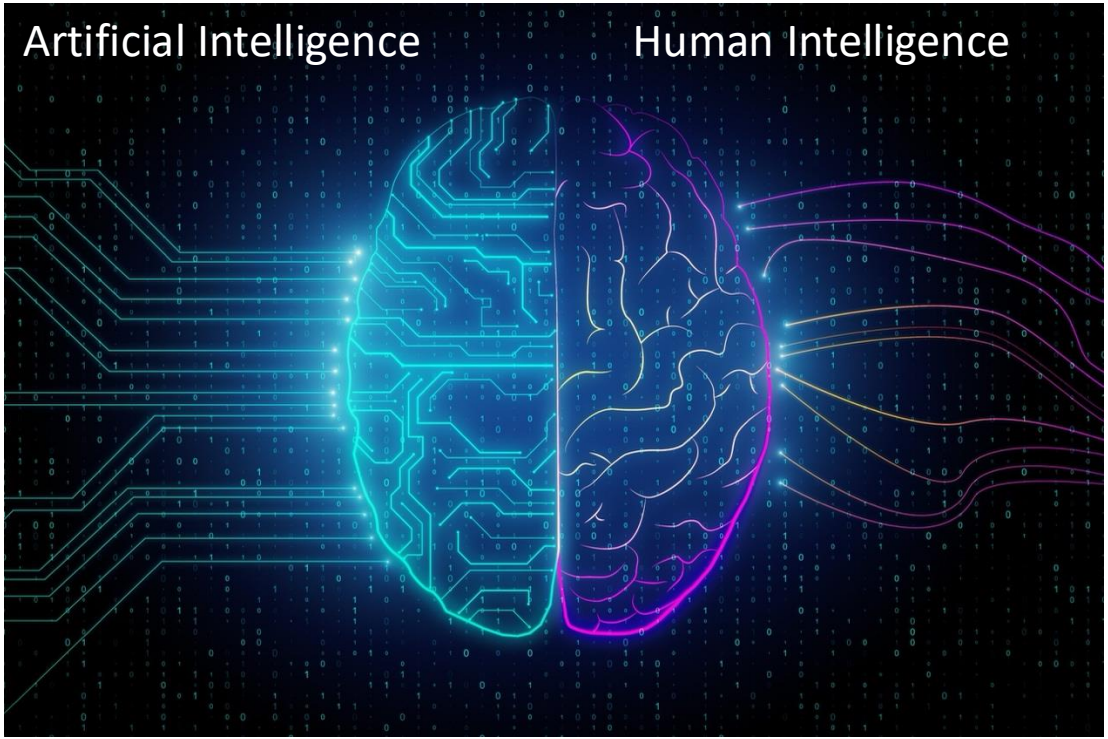


Quiz time

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.5	0.9	1	1	0.7	0	0	0	0	0
0	0	0	0	0.4	0.3	0.4	0.5	1	0.5	0	0	0	0
0	0	0	0	0	0	0	0.6	1	0.5	0	0	0	0
0	0	0	0	0	0.3	0.9	0.9	0.5	0	0	0	0	0
0	0	0	0	0	1	1	0.9	0.4	0	0	0	0	0
0	0	0	0	0	0.1	0.1	0.5	0.9	0.6	0	0	0	0
0	0	0	0	0	0	0	0	0.2	1	0.4	0	0	0
0	0	0	0.1	0.4	0	0	0	0.4	1	0.4	0	0	0
0	0	0	0.4	0.7	0.3	0.5	0.8	1	0.7	0	0	0	0
0	0	0	0	0.6	1	1	0.8	0.4	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

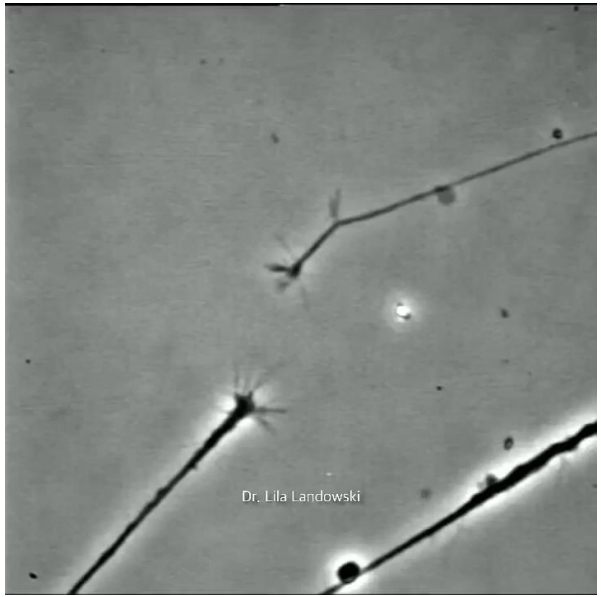


Neural Network



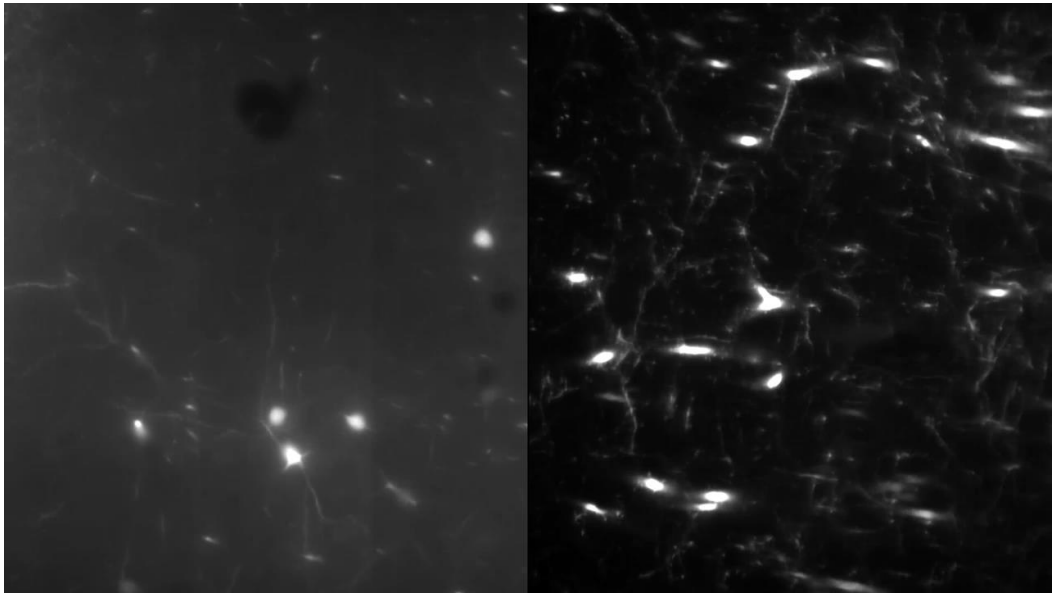
Biological Neural Network

Two neurons connecting to each other



Credit: Lila Landowski

Large-scale 3D tissue structure

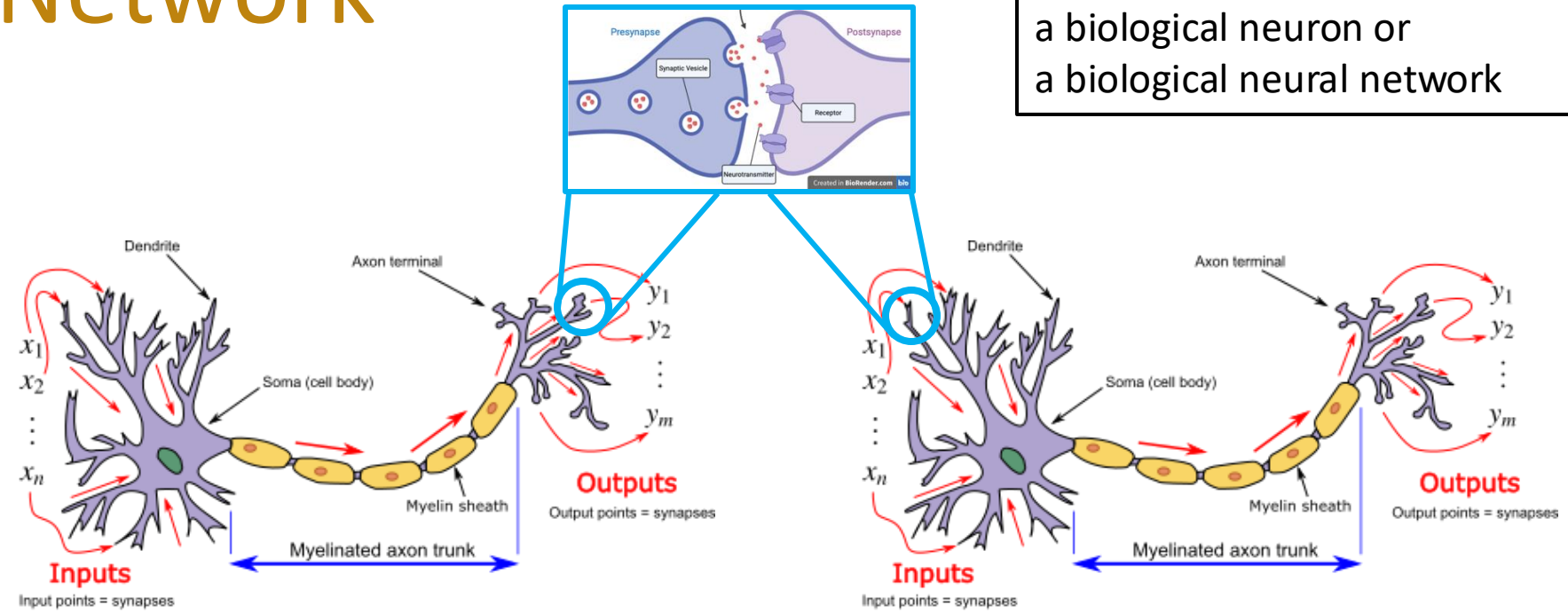


Credit: Yanlu Chen

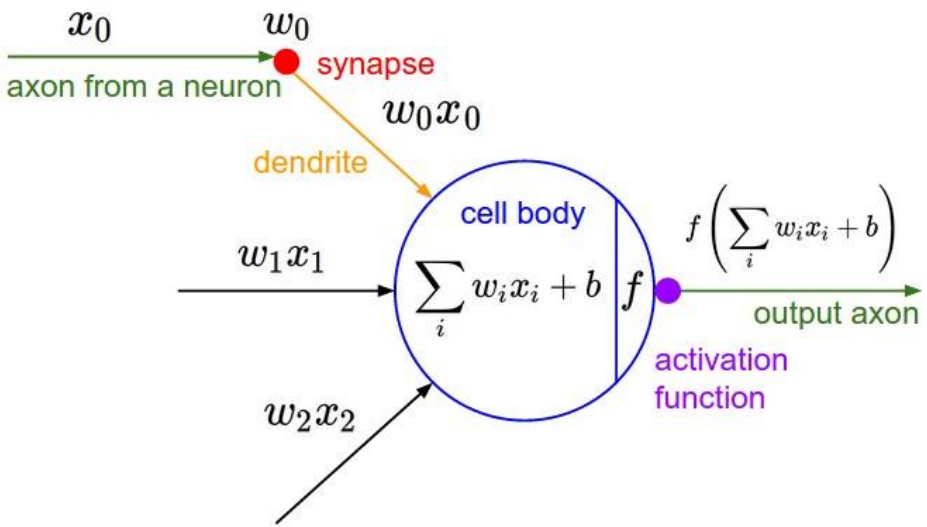
Biological Neural Network

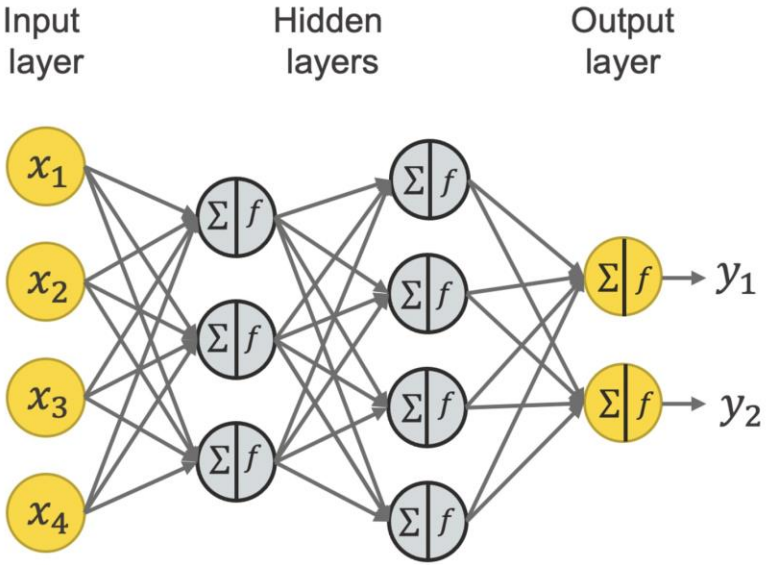
???

What is the simplest mathematical model for a biological neuron or a biological neural network



Artificial Neural Network

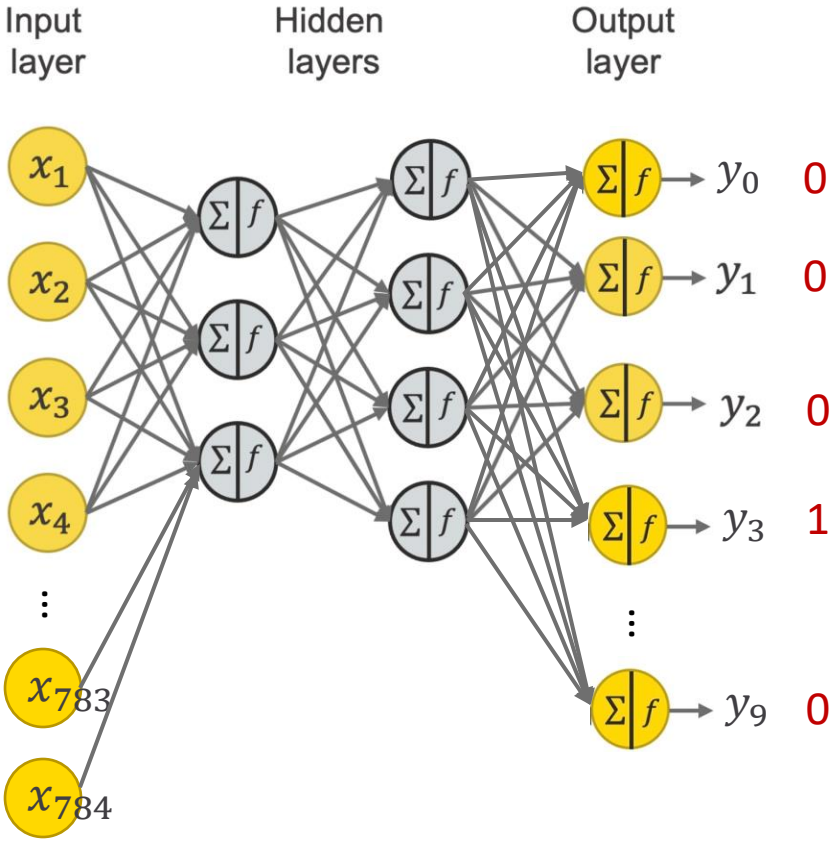




Artificial Neural Network

INPUT
(28x28)

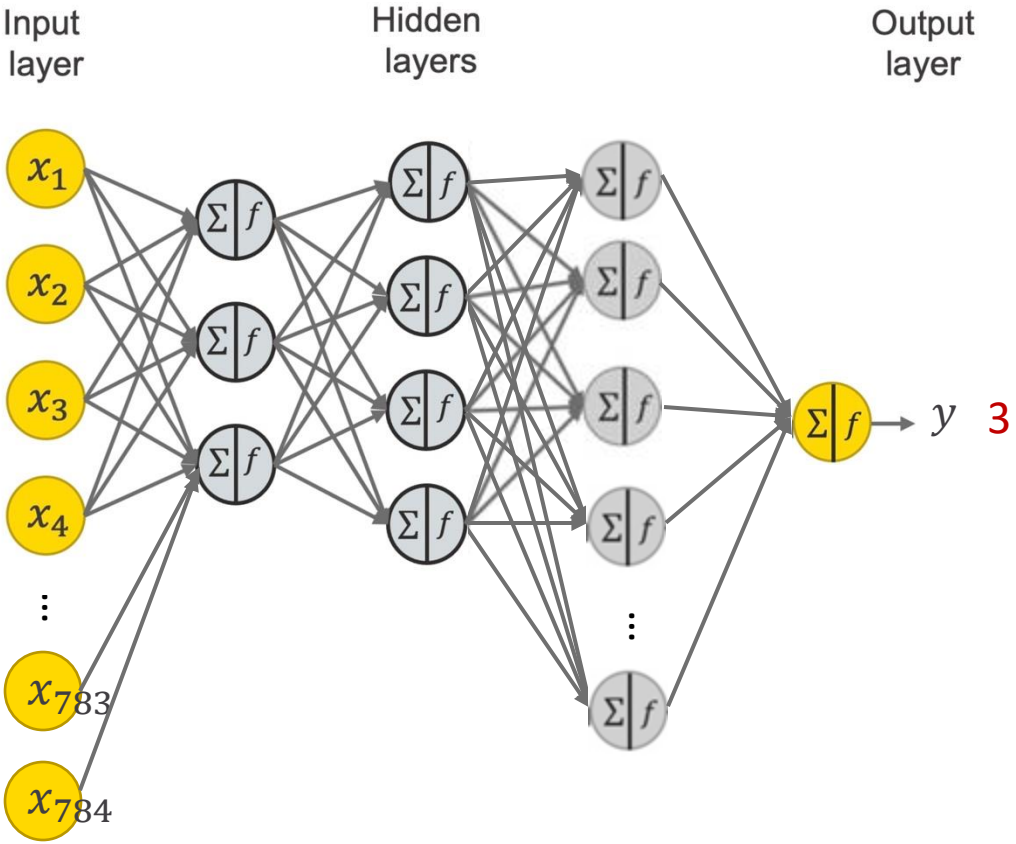
INPUT
(784)



Artificial Neural Network

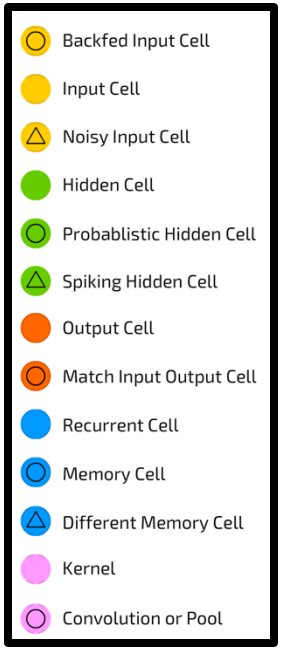
INPUT
(28x28)

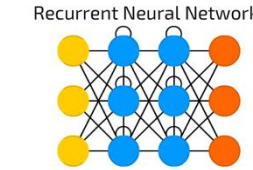
INPUT
(784)

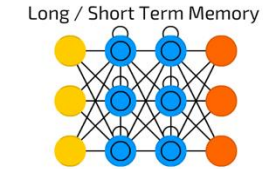


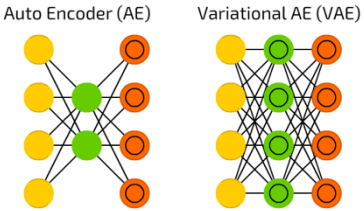
Artificial Neural Network

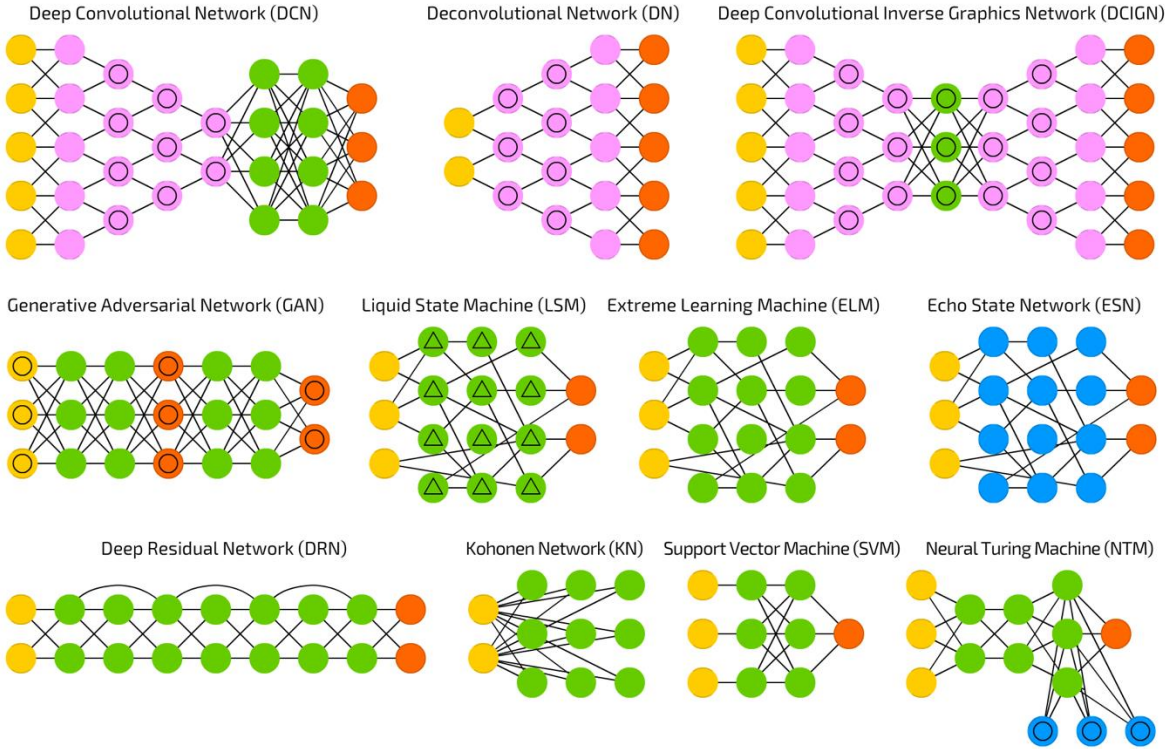
Things can get (very) interesting from here









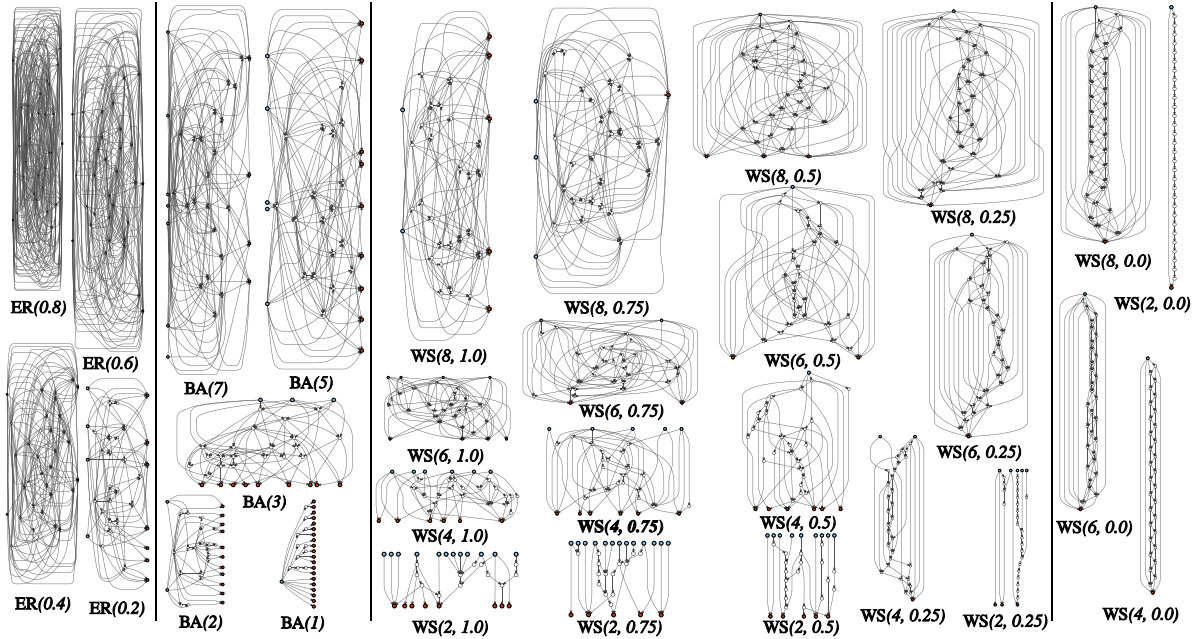


Artificial Neural Network

Things can get (very) interesting from here

"the physical connections of the nervous system … are not identical from one organism to another", and "at birth, the construction of the most important networks is largely random."

-- Frank Rosenblatt (1958)



Xie+19

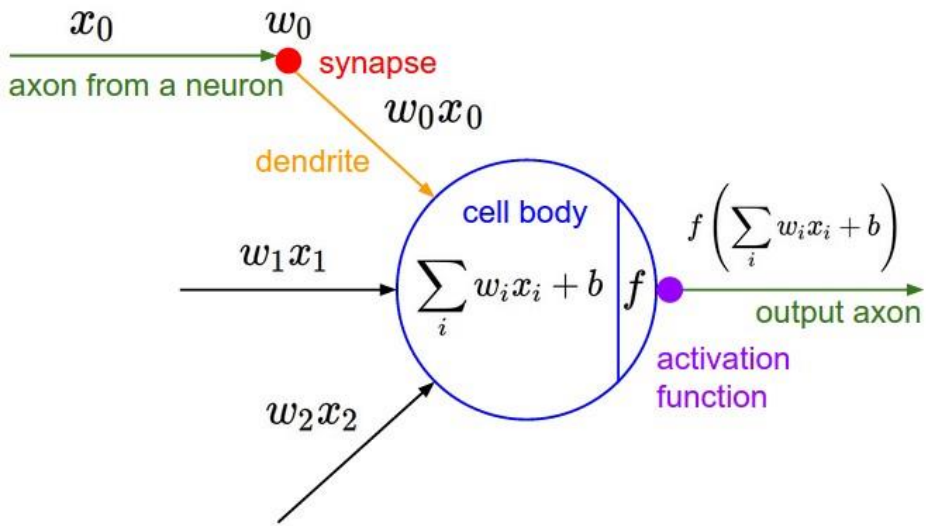
ER(P) [Erdos-Renyi] two nodes are connected with a probability of P

BA(M) [Barabasi-Albert], initial state has M nodes with no connection, new nodes are added and connected to existing node with a probability proportional to its degree.

WS(K,P) [Watts-Strogatz], initial state has N nodes regularly placed in a circle, and each connected to its K/2 neighbours on both side, every node is then connected to the ith next node with a probability of P

Neurons

- Receives inputs (an N-D array)
- Multiples each input by its weight (w; coefficient)
- Applies activation function (f) to the sum and bias (b)
- Output result

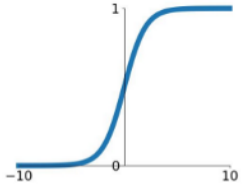


Activation functions

- Controls when the unit is active or inactive.
- Adds non-linearity to the neural network.

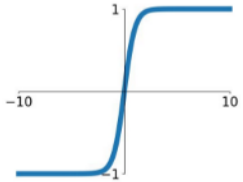
Sigmoid

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



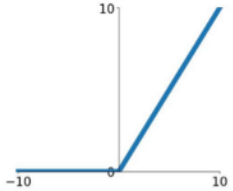
tanh

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



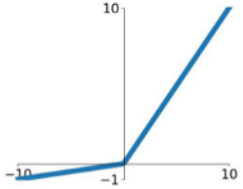
ReLU

$$\max(0, x)$$



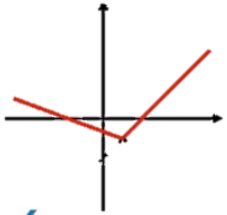
Leaky ReLU

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



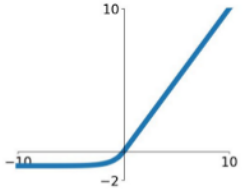
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$



ELU

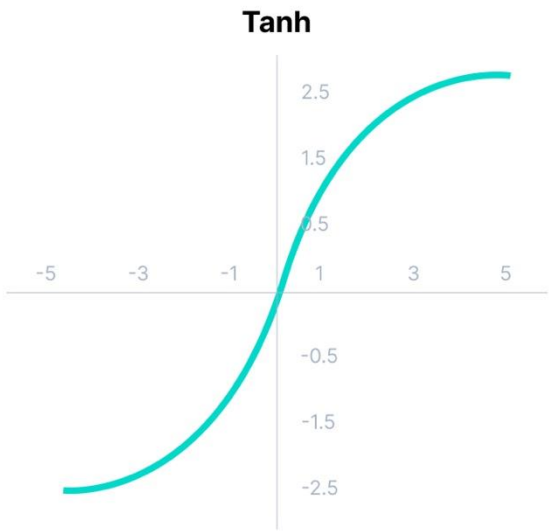
$$\begin{cases} x & x \geq 0 \\ \alpha(e^{x}-1) & x < 0 \end{cases}$$



Activation functions

There is no good or bad AF. Things to consider:

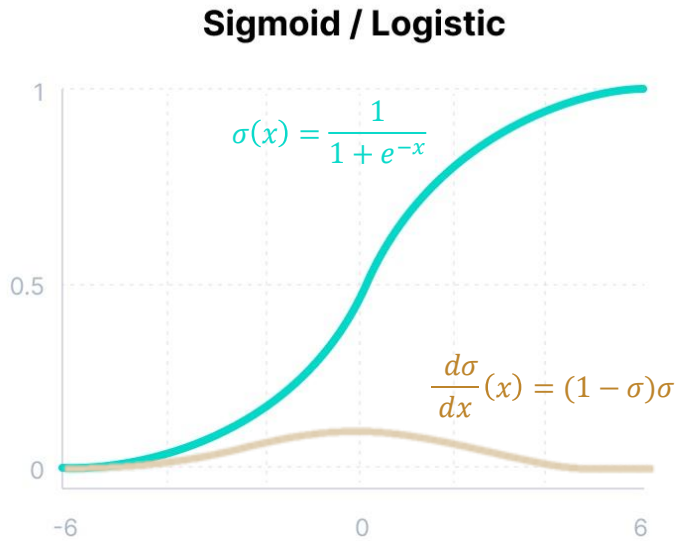
- Zero center or not, e.g., the output of the tanh activation function is zero centred; hence we can easily map the output values as strongly negative, neutral, or strongly positive.

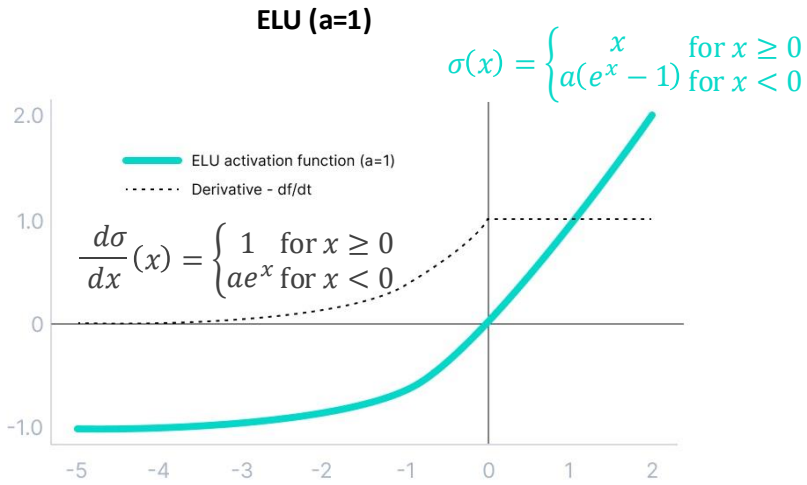


Activation functions

There is no good or bad AF. Things to consider:

- Gradient! You would like it to vanish (in most cases)
  - as the gradient value approaches zero, the network ceases to learn.
  - gradient can also accumulate and result in very large updates to the weight, causing overflow and hence unstable network.

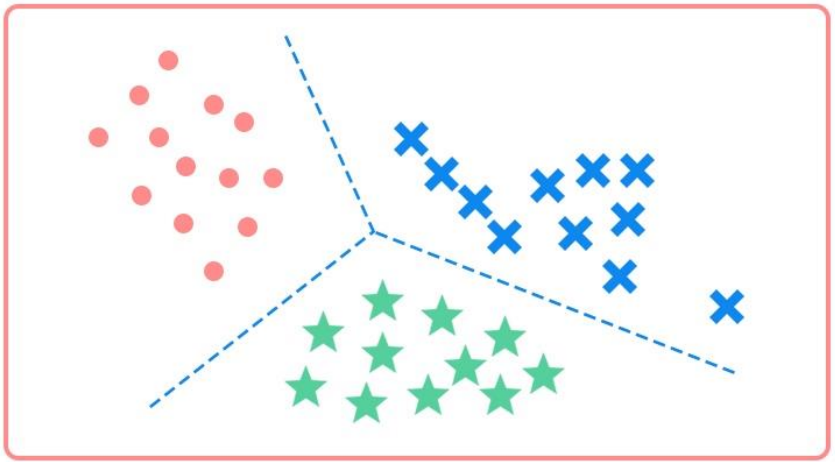


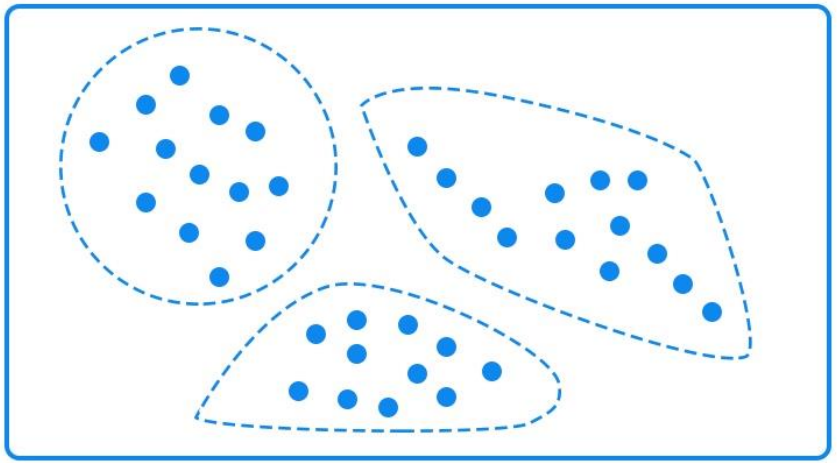


Supervised vs unsupervised

It's about the task you are interested

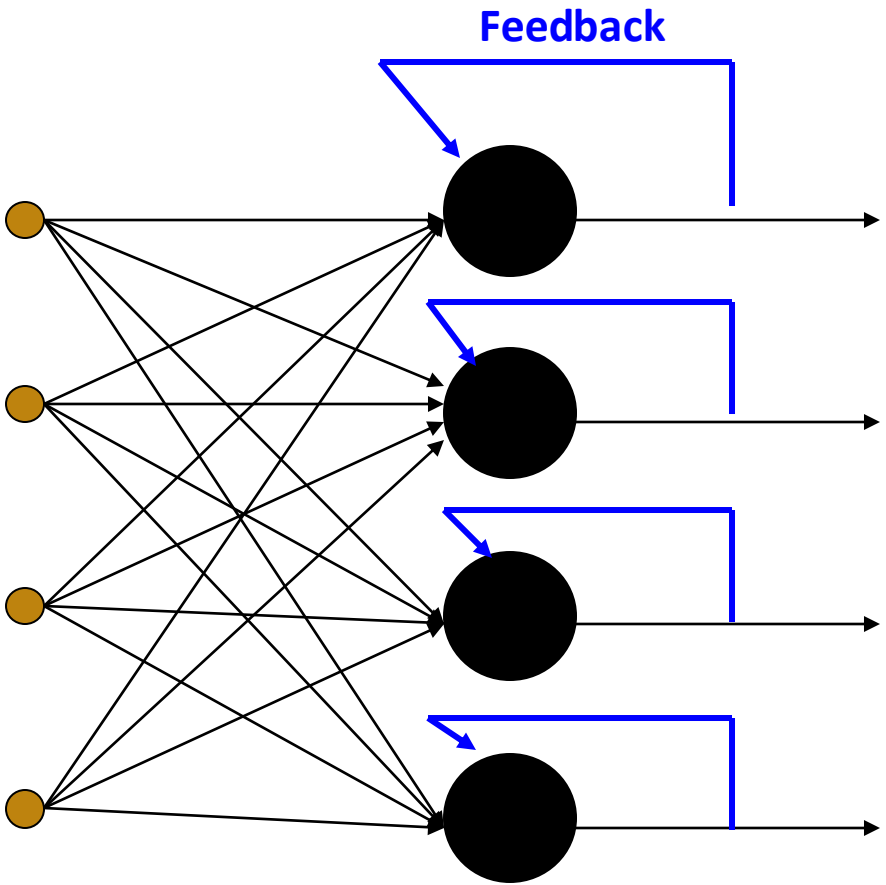
- Pre-categorized data or unlabeled data
- Interested in predictions or patterns
- Classification (divide the socks by color) or clustering (divide by similarity)
- Regression (divide the ties by length) or association (identify sequences)





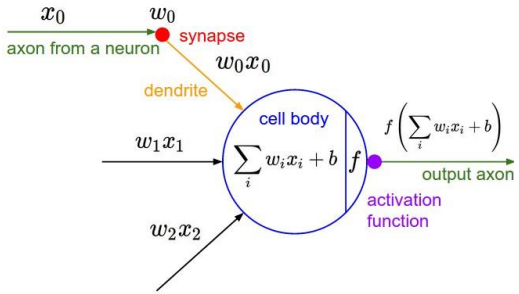
Feedforward vs Recurrent

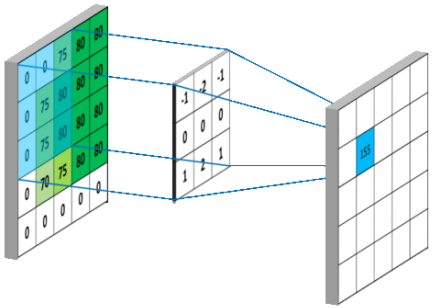
- FFN:
  - Information flows in one direction
  - Simple and efficient
  - Good for predicting a single output
- RNN:
  - Information flows back and forth
  - Learn temporal relationship in the input
  - Good for sequential data, e.g. LLM

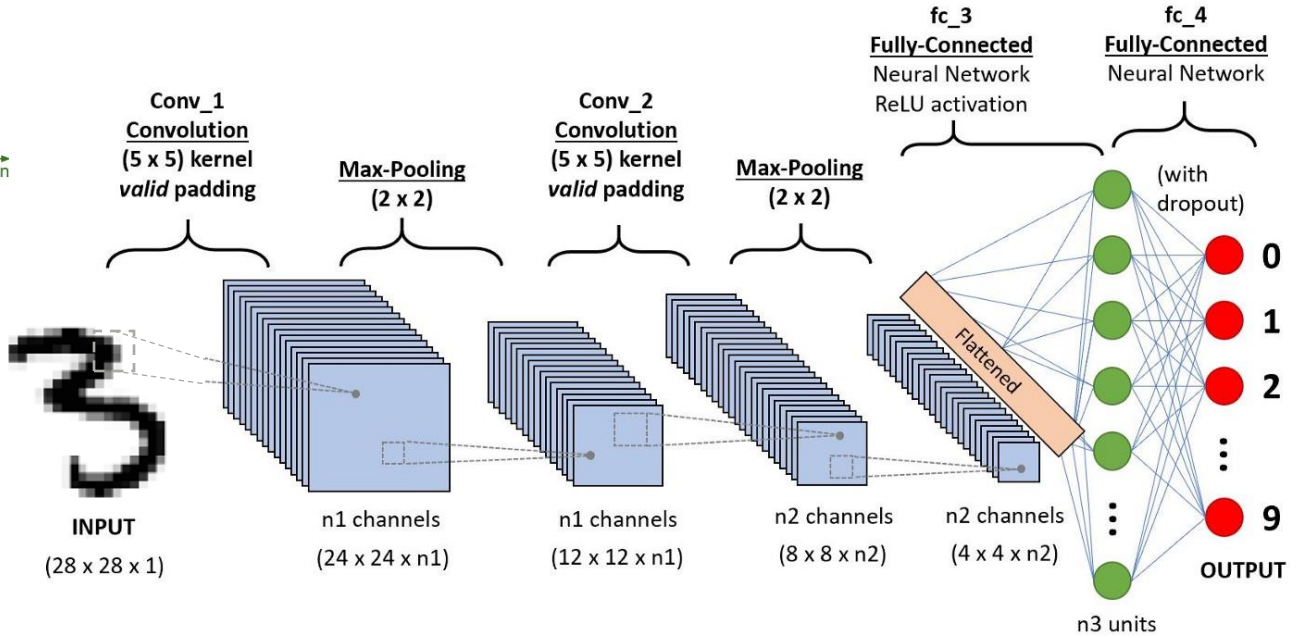


Dimension – Convolutional Neural Network

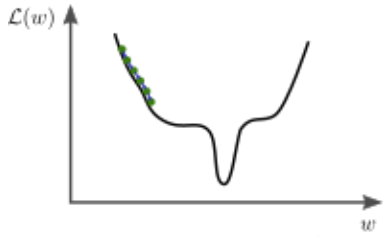
Essentially, f is a convolution kernerl







Neural Network Training



- Loss (L):

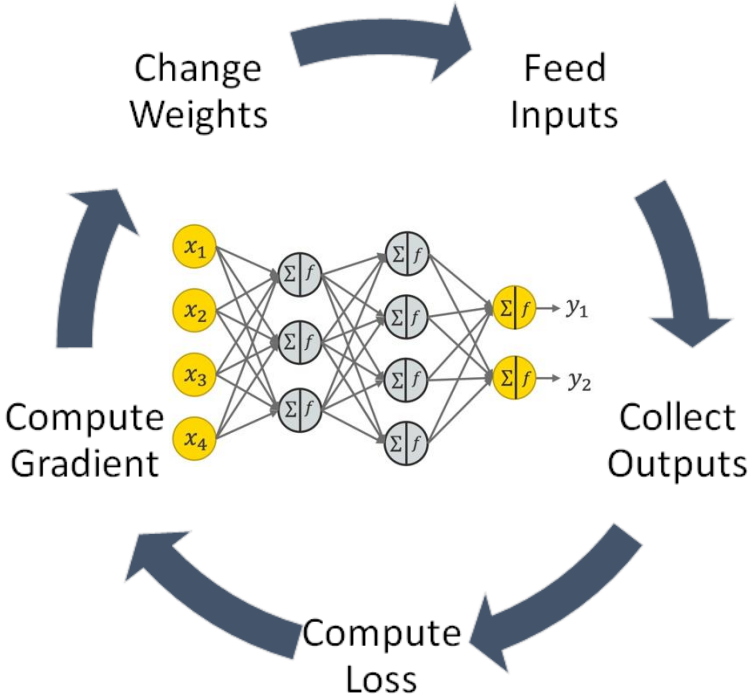
Cost of error, a measure of how well or poorly a model's predictions match the actual target values. The goal of training is to minimize this loss to make the predictions as accurate as possible.

- Backpropagation:

Backward propagation of errors, a fundamental algorithm used to train artificial neural network. It works by calculating the error between the network's predictions and the actual output. The error is then propagated back through the network, and the weights of the network are adjusted to reduce the error.

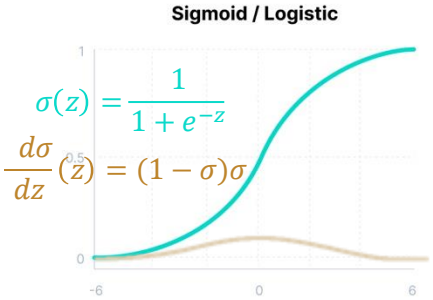
- Gradient descent (∇):

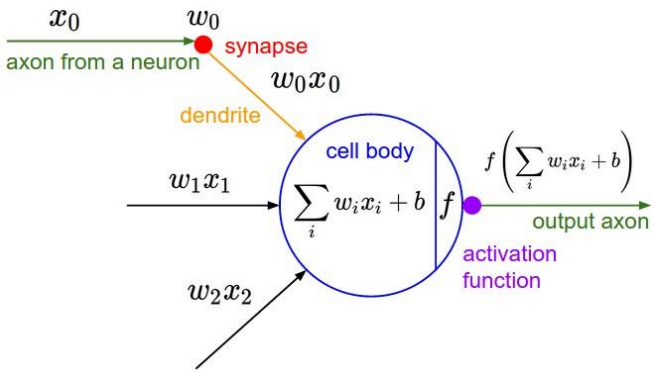
$w = w - \eta \nabla L(w)$, an optimization algorithm that is used to find the minimum of a function. It works by iteratively updating the parameters of a function until the function reaches a minimum. The gradient of a function is a vector that points in the direction of the steepest ascent of the function, updating the parameters of a function in the direction of the negative gradient.



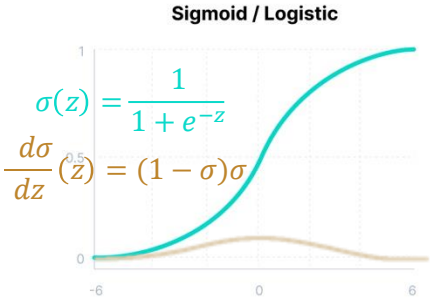
Example

- Loss function is Mean Squared Error: $L =$
- Sigmoid activation function:
- A simple feedforward network with 2 layers and each layer has only 1 node





Example



- Loss function is Mean Squared Error: $L = 0.5(\hat{y} - y)^2$
- Sigmoid activation function: $\sigma(z) = \frac{1}{1+e^{-z}}, \frac{d\sigma}{dz}(z) = (1-\sigma)\sigma$
- A simple feedforward network with 2 layers and each layer has only 1 node

$$x \rightarrow \underset{y_1 = \sigma(z_1) = (1+e^{-z_1})^{-1}}{z_1 = w_1 x + b_1} \rightarrow \underset{y_2 = \sigma(z_2)}{z_2 = w_2 y_1 + b_2} \rightarrow L = 0.5(y_2 - y)^2$$

- Step 1 - Gradient of the output neuron: $\frac{\partial L}{\partial y_2} = y_2 - y$
- Step 2 - Gradient of the neuron in the output layer (chain rule!):

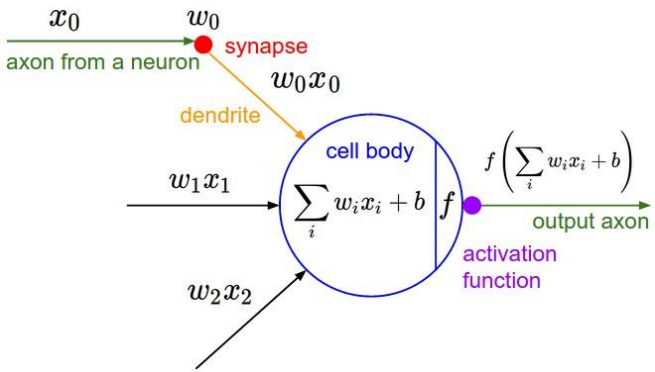
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = (y_2 - y) \cdot (1 - y_2) y_2 \cdot y_1$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = (y_2 - y) \cdot (1 - y_2) y_2$$

- Step 3 – Gradient of the neuron in the input layer (more chain rule!!!):

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = (y_2 - y) \cdot (1 - y_2) y_2 \cdot w_2 \cdot (1 - y_1) y_1 \cdot x$$

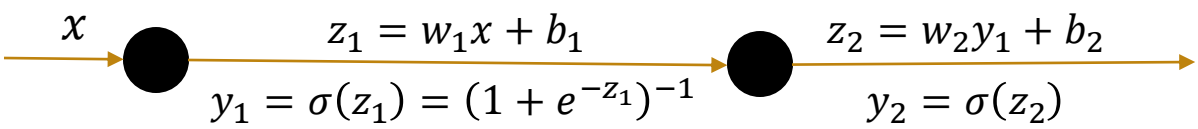
$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = (y_2 - y) \cdot (1 - y_2) y_2 \cdot w_2 \cdot (1 - y_1) y_1$$



Training in action

$$\nabla L(w_2) = (y_2 - y) \cdot (1 - y_2) y_2 \cdot y_1$$
$$\nabla L(b_2) = (y_2 - y) \cdot (1 - y_2) y_2$$
$$\nabla L(w_1) = (y_2 - y) \cdot (1 - y_2) y_2 \cdot w_2 \cdot (1 - y_1) y_1 \cdot x$$
$$\nabla L(b_1) = (y_2 - y) \cdot (1 - y_2) y_2 \cdot w_2 \cdot (1 - y_1) y_1$$
$$w = w - \eta \nabla L(w); \quad b = b - \eta \nabla L(b)$$

- Let's assume some numbers
- $x = 1, y = 1, w_1 = 0.5, w_2 = 0.8, b_1 = 0, b_2 = 0, \eta = 0.1$



step	w_1	b_1	z_1	y_1	$\nabla L(w/\mathbf{b}_1)$	w_2	b_2	z_2	y_2	$\nabla L(w_2)$	$\nabla L(b_2)$	L
1	0.5000	0.0000				0.8000	0.0000					
2												
3												
4												
5												

Learning rate

$$w = w - \eta \nabla L(w)$$

The learning rate determines the size of the steps taken towards minimizing the loss function, influencing how quickly or slowly a model adjusts its weights and biases during training.

Learning rate

$$w = w - \eta \nabla L(w)$$

The learning rate determines the size of the steps taken towards minimizing the loss function, influencing how quickly or slowly a model adjusts its weights and biases during training.

small learning rate leads to slow convergence

$\eta = 0.1$

step	w_1	b_1	z_1	y_1	$\nabla L(w/\mathbf{b}_1)$	w_2	b_2	z_2	y_2	$\nabla L(w_2)$	$\nabla L(b_2)$	L
1	0.5000	0.0000	0.5000	0.6225	-0.0168	0.8000	0.0000	0.4980	0.6220	-0.0554	-0.0889	0.0715
2	0.5017	0.0017	0.5034	0.6232	-0.0167	0.8055	0.0089	0.5151	0.6260	-0.0540	-0.0867	0.0696
3	0.5033	0.0034	0.5067	0.6240	-0.0165	0.8110	0.0176	0.5231	0.6278	-0.0542	-0.0869	0.0691
4	0.5049	0.0051	0.5100	0.6248	-0.0165	0.8164	0.0264	0.5331	0.6302	-0.0538	-0.0862	0.0682
5	0.5066	0.0067	0.5132	0.6256	-0.0164	0.8218	0.0350	0.5388	0.6318	-0.0535	-0.0859	0.0677

Learning rate

$$w = w - \eta \nabla L(w)$$

The learning rate determines the size of the steps taken towards minimizing the loss function, influencing how quickly or slowly a model adjusts its weights and biases during training.

increasing learning rate can speed up training

$\eta = 1$

step	w_1	b_1	z_1	y_1	$\nabla L(w/\mathbf{b}_1)$	w_2	b_2	z_2	y_2	$\nabla L(w_2)$	$\nabla L(b_2)$	L
1	0.5000	0.0000	0.5000	0.6225	-0.0168	0.8000	0.0000	0.4980	0.6220	-0.0554	-0.0889	0.0715
2	0.5168	0.0168	0.5336	0.6303	-0.0165	0.8555	0.0889	0.5566	0.6357	-0.0531	-0.0862	0.0648
3	0.5333	0.0333	0.5666	0.6379	-0.0163	0.9086	0.1751	0.6114	0.6491	-0.0508	-0.0830	0.0590
4	0.5497	0.0497	0.5993	0.6452	-0.0160	0.9595	0.2580	0.6622	0.6617	-0.0483	-0.0793	0.0539
5	0.5657	0.0657	0.6314	0.6522	-0.0157	1.0080	0.3373	0.7095	0.6732	-0.0457	-0.0751	0.0493

Learning rate

$$w = w - \eta \nabla L(w)$$

The learning rate determines the size of the steps taken towards minimizing the loss function, influencing how quickly or slowly a model adjusts its weights and biases during training.

increasing learning rate can speed up training **BUT!!!** $\eta = 10$

step	w_1	b_1	z_1	y_1	$\nabla L(w/\mathbf{b}_1)$	w_2	b_2	z_2	y_2	$\nabla L(w_2)$	$\nabla L(b_2)$	L
1	0.5000	0.0000	0.5000	0.6225	-0.0168	0.8000	0.0000	0.4980	0.6220	-0.0554	-0.0889	0.0715
2	0.6680	0.1680	0.8359	0.6976	-0.0126	1.3541	0.8890	1.0191	0.7348	-0.0407	-0.0661	0.0351
3	0.7941	0.2939	1.0880	0.7479	-0.0092	1.7612	1.5500	1.4579	0.8112	-0.0298	-0.0484	0.0177
4	0.8864	0.3862	1.2726	0.7811	-0.0067	2.0596	2.0344	1.8551	0.8647	-0.0219	-0.0356	0.0089
5	0.9533	0.4532	1.4065	0.8012	-0.0049	2.2777	2.3903	2.2228	0.9023	-0.0160	-0.0260	0.0044

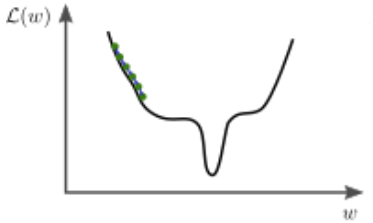
Learning rate

$$w = w - \eta \nabla L(w)$$

The learning rate determines the size of the steps taken towards minimizing the loss function, influencing how quickly or slowly a model adjusts its weights and biases during training.

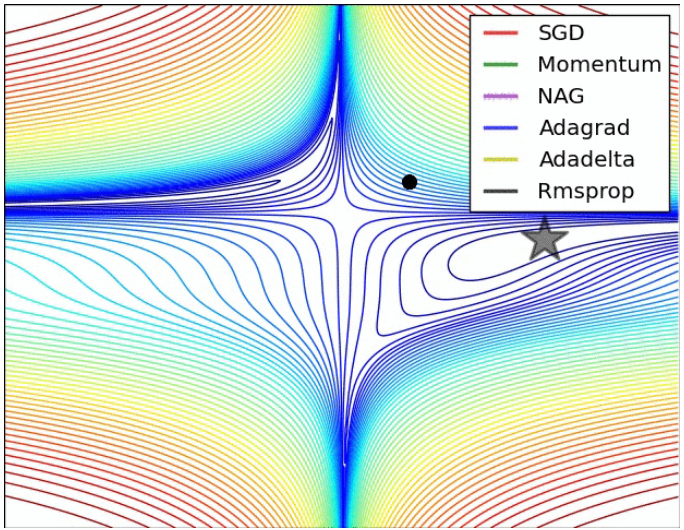
increasing learning rate can speed up training

BUT!!! large η is not always good



Optimizer

Mathematical tools or algorithms used to adjust parameters



- **Stochastic Gradient Descent (SGD)**

updates model parameters using a randomly selected subset of training data (a mini-batch) at each iteration. This helps speed up training and allows for updates more frequently compared to traditional gradient descent, which uses the entire dataset.

- **Adaptive Gradient Algorithm (Adagrad)**

adapts the learning rate for each parameter based on the historical gradients. It gives more weight to infrequently updated parameters and less to frequently updated ones, making it suitable for sparse data.

- **Root Mean Square Propagation (RMSprop)**

modifies Adagrad by using a moving average of the squared gradients to normalize the learning rate. This helps to avoid the rapid decay of learning rates that can occur with Adagrad, making it more effective for non-stationary objectives.

- **Adaptive Moment Estimation (Adam)**

improves upon SGD and RMSprop by adding a fraction of the previous update to the current update. It maintains a moving average of both the gradients and their squared values, allowing it to adaptively adjust the learning rate for each parameter.

Loss function

Cost of error, a measure of how well or poorly a model's predictions match the actual target values. The goal of training is to minimize this loss to make the predictions as accurate as possible.

Training (70%)	+	Validation (20%)	+	Test set (10%)
Things the network tries to fit		Things the network uses to exam itself		Things the network has never seen and you use to exam it

- Regression:
  - Mean Squared Error Loss (MSE):

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  - Mean Absolute Error Loss (MAE):

$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

  - Mean Squared Logarithmic Error Loss (MSLE):

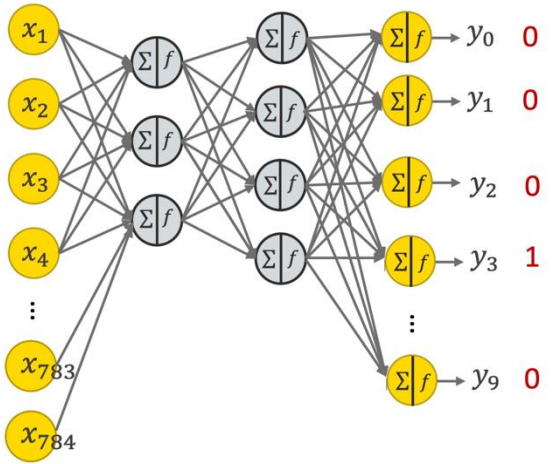
$$L(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}[\log(y_i + 1) - \log(\hat{y}_i + 1)] = \frac{1}{N}\sum_{i=1}^{N}\left(\log\frac{y_i+1}{\hat{y}_i+1}\right)$$

- Classification:
  - Cross-entropy Loss:

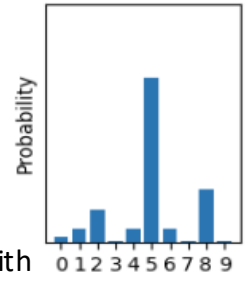
$$L(y,\hat{y}) = -\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i)]$$

  - Binary Cross Entropy Loss:

$$L(\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i \cdot \log[p(y_i)] + (1 - y_i) \cdot \log[1 - p(y_i)]\}$$



Cross-entropy and softmax



For tasks like recognizing handwritten digit, we can use softmax activation function in the output layer to predicts the probability ($\hat{y}$) of the image being 0, 1, …, 9, i.e.,$\hat{y}_i = \sigma(z_i) = \frac{e^{z_i}}{\sum_{i=0}^{9} e^{z_i}}$. The softmax function naturally gives $\Sigma\sigma(z_i) = 1$ with each elements representing the probability of the output. In other word, the probability of the input being 0, 1, …, 9.

Cross entropy, $L(y, \hat{y}) = -\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i)]$, measures the difference between two probability distributions (i.e. the true vs predicted). We need $\hat{y}$ to be as close to 1 as possible for only the label where y=1, and as close to 0 as possible for all other labels.

Expand the cross-entropy loss function

$$L(y, \hat{y}) = -\sum_{k=0}^{9}[y_k \cdot \log(\hat{y}_k)] = -\sum_{k=0}^{9}\left[y_k \cdot \log\left(\frac{e^{z_k}}{\sum_{j=0}^{9} e^{z_j}}\right)\right] = -\sum_{k=0}^{9} y_k \left[z_k - \log\left(\sum_{j=0}^{9} e^{z_j}\right)\right]$$

Calculate the derivative over the z_j

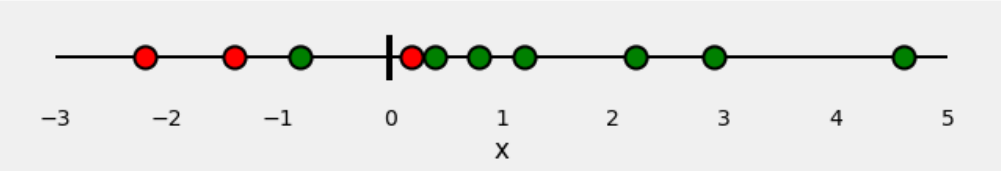
$$\frac{\partial L}{\partial z_i} = -\sum_{k=0}^{9} y_k \left[\frac{\partial z_k}{\partial z_i} - \frac{\partial}{\partial z_i}\log\left(\sum_{j=0}^{9} e^{z_j}\right)\right] = -\sum_{k=0}^{9} y_k \delta_{ki} + \sum_{k=0}^{9} y_k \frac{e^{z_i}}{\sum_{j=0}^{9} e^{z_j}} = -y_i + \frac{e^{z_i}}{\sum_{j=0}^{9} e^{z_j}} = -y_i + \hat{y}_i$$

For the correct label (y_i=1), if the network predicts a higher probability ($\hat{y}_i > y_i$), the gradient is positive, meaning that z_i and hence $\hat{y}_i$ should be decreased to reduce the loss. When $\hat{y}_i < y_i$, the gradient is negative, z_i and $\hat{y}_i$ should be increased to reduce the loss. For incorrect labels (y_i=0), the gradient is simply $\hat{y}_i$, encouraging the network to decrease the probability of incorrect labels.

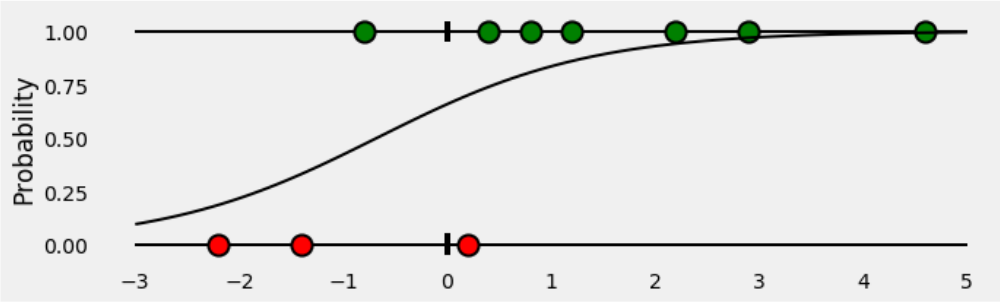
Binary Cross Entropy

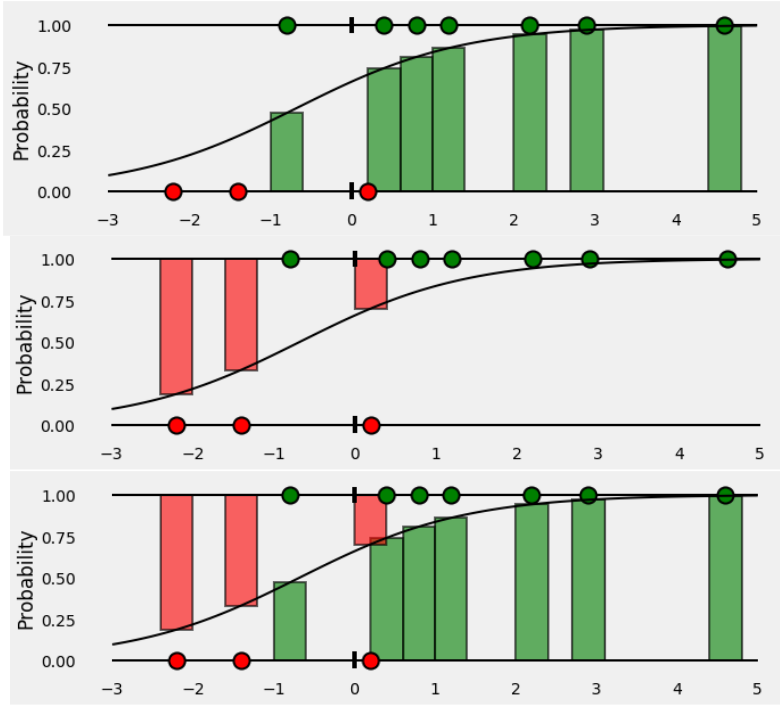
$$L(\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i \cdot \log[p(\hat{y}_i)] + (1 - y_i) \cdot \log[1 - p(\hat{y}_i)]\}$$

Task: classify the red and green points given x



ML language -> what is the probability of the point being green?
i.e., P(green is green) = 1 and P(green is red)=0
Labelling green as y=1 and red as y=0, then p(y) is our model, representing the probability of the point being green





Credit: Daniel Godoy

Australia's Artificial Intelligence Ethics Principles

Australia's 8 Artificial Intelligence (AI) Ethics Principles are designed to ensure AI is safe, secure and reliable.

- Department of Industry, Science and Resources (7 November 2019)

- **Human, societal and environmental wellbeing**: AI systems should benefit individuals, society and the environment
- **Human-centred values**: AI systems should respect human rights, diversity, and the autonomy of individuals
- **Fairness**: AI systems should be inclusive and accessible, and should not involve or result in unfair discrimination against individuals, communities or groups
- **Privacy protection and security**: AI systems should respect and uphold privacy rights and data protection, and ensure the security of data
- **Reliability and safety**: AI systems should reliably operate in accordance with their intended purpose
- **Transparency and explainability**: There should be transparency and responsible disclosure so people can understand when they are being significantly impacted by AI, and can find out when an AI system is engaging with them
- **Contestability**: When an AI system significantly impacts a person, community, group or environment, there should be a timely process to allow people to challenge the use or outcomes of the AI system.
- **Accountability**: People responsible for the different phases of the AI system lifecycle should be identifiable and accountable for the outcomes of the AI systems, and human oversight of AI systems should be enabled

Australian Government Use & Regulatory Direction

National framework for the assurance of artificial intelligence in government
- Department of Finance (21 June 2024)

Policy for the responsible use of AI in government
- Australian Public Service (1 September 2024)

Safe and responsible AI in Australia
- Therapeutic Goods Administration (September 2024)

Government's interim response to the “Safe and Responsible AI in Australia” consultation
- Therapeutic Goods Administration (17 January 2024)

If you’re building, deploying, or using AI in Australia (or with Australian users)
- Be proactive.
- Maintain auditability, logging, test suites, monitoring.
- Think about transparency & explainability.
- Pay attention to copyright, privacy & licensing.
- Follow the advice of the AI Safety Standard.

Internationally

The AI Bill of Rights (2022) from the White House outlines key principles for the responsible use of artificial intelligence. It emphasizes the need for AI systems to be designed and used in ways that respect individual rights and promote fairness. The document focuses on five main areas, including **Safety and Security, Non-Discrimination**, **Privacy, Notice and Explanation, Human Alternatives.** Also, Executive Order 14110 (Oct 2023) and State laws (e.g., California's SB 53; Transparency in Frontier Artificial Intelligence Act)

The primary legislative framework for regulating AI in the EU is the EU AI Act. It was published in the EU Official Journal on July 12, 2024, and is the first comprehensive horizontal legal framework for the regulation of AI across the EU. The EU AI Act enters into force on August 1, 2024, and will be effective from August 2, 2026, except for the specific provisions listed in Article 113. The key features of the EU AI Act include **Risk-Based Classification, Safety and Accountability, Prohibitions, Compliance and Enforcement.**

Canada: The Directive on Automated Decision-Making (2019, updated 2023) governs federal public-sector. Working on Artificial Intelligence and Data Act (AIDA).

China: Multiple binding regulations: Algorithmic Recommendation Provisions (2021); Deep Synthesis (Deepfake) Provisions (2022); Interim Measures for Generative AI (2023)

India: No binding AI law yet. Current stance is to promote innovation; voluntary principles on transparency, consent, deepfake labelling, and election integrity.

Japan: AI Guidelines for Business (2024) consolidate prior ethical & data-use guidance. Emphasis on trustworthy AI, human rights, and international harmonization via the Hiroshima AI Process (G7).

Singapore: Soft-law: Model AI Governance Framework (2019/2020); Model Framework for Generative AI (draft 2024); AI Verify Foundation (2023) for open-source governance tools.

UK: Initiatives like the AI White Paper (2023) outlining guidelines for AI use. Regulators issue domain-specific guidance and coordinate via Central AI Unit. Bletchley Declaration on frontier-model safety.

(incomplete list…)

Neural network example – prepare dataset

```
import tensorflow.compat.v1 as tf
from tensorflow.keras import layers
from tensorflow.keras.optimizers import Adam

# load training data (X_train, Y_train)
# load validation data (X_valid, Y_valid)
# covert them into tensorflow format
x_train = tf.data.Dataset.from_tensor_slices(X_train); x_val = tf.data.Dataset.from_tensor_slices(X_valid)
y_train = tf.data.Dataset.from_tensor_slices(Y_train); y_val = tf.data.Dataset.from_tensor_slices(Y_valid)

# This is the size of a single training batch (training is not done on the entire test set at once, but on batches of data)
batch_size = 64

# package the data
training_data = tf.data.Dataset.zip((x_train, y_train)).shuffle(X_train.shape[0]).batch(batch_size)
validation_data = tf.data.Dataset.zip((x_val, y_val)).shuffle(X_valid.shape[0]).batch(batch_size)

# fancy features
callbacks = [tf.keras.callbacks.ReduceLROnPlateau(patience = 20), # If loss does not improve for 20 epochs, reduce learning rate
   tf.keras.callbacks.EarlyStopping(patience = 50)] # If loss does not improve for 50 epochs, stop the learning
```

Neural network example – define network structure

```
# Input size = number of params
input_layer = layers.Input(shape = X_train.shape[-1])

# Hidden fully-connected layers
output = layers.Dense(64)(input_layer)
# Batch normalization = normalize the training weights at every batch.
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
output = layers.Dense(128)(output)
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
output = layers.Dense(64)(output)
output = layers.BatchNormalization()(output)
output = layers.ReLU()(output)
# Last layer output shape
output = layers.Dense(Y_train.shape[-1])(output)

model = tf.keras.Model(inputs = [input_layer], outputs = [output])
model.summary()
model.compile(loss = 'mse', optimizer = Adam(learning_rate = 1e-3))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 9)]               0
_________________________________________________________________
dense (Dense)                (None, 64)                640
_________________________________________________________________
batch_normalization (BatchNo (None, 64)                256
_________________________________________________________________
re_lu (ReLU)                 (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
re_lu_1 (ReLU)               (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256
_________________________________________________________________
re_lu_2 (ReLU)               (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 54)                3510
=================================================================
Total params: 21,750
Trainable params: 21,238
Non-trainable params: 512
_________________________________________________________________
```

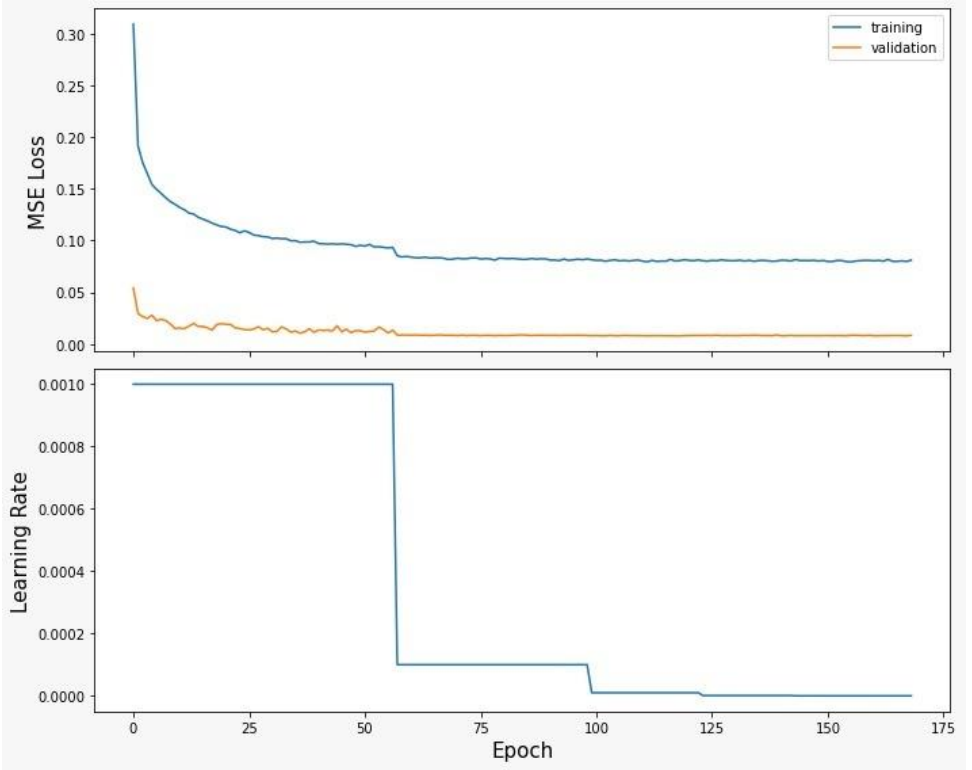
Neural network example – train network

```
history = model.fit(training_data,
            epochs = 300,
            batch_size = batch_size,
            validation_data = validation_data,
            callbacks = callbacks,
            shuffle = True # shuffle the data in each batch
            )

# get histories of loss for training set
(history.history['loss']) and validation set
(history.history['val_loss']) as well as the learning rate
(history.history['lr'])

# make prediction by calling model.predict(X_test)

# save the network by calling model.save('emulator')
```



prerequisite

We will start at 1:05pm, before that, please

git clone https://github.com/qyx268/astr4004-8004-2025.git

To open a gz file, do gzip -d file.gz

Install tensorflow, matplotlib, numpy

Assignment 4 (due on the 19th)

Design a neural network for regression *(bonus point if the model is astrophysical)*

- with at least 5 neurons;
- involving either relu (max(0,x)) or sigmoid (1/(exp(-x)+1)) activation functions;
- having a loss that within 10 steps
  - monotonically decreases when the learning rate is 0.1,
  - but can increase within 10 steps when the learning rate is >0.1.

The report should include

- a description of the model or regression problem you are solving; (10%)
- a graph illustrating the network structure; (10%)
- equations of gradient for all neurons; (30%)
- two tables listing weights, gradients, outputs and loss for 10 steps
  - learning rate = 0.1, (25%)
  - learning rate > 0.1 (25%)

Bonus point might maximize the score for this assignment and this assignment only