



Welcome to this session: Task Walkthrough - Task 22 - 26 (Part 2)

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators
answering questions.



Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

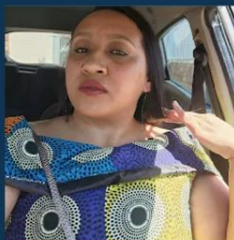
If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles
Designated Safeguarding
Lead



Simone Botes



Nurhaan Snyman



Rafiq Manan



Ronald Munodawafa



Tevin Pitts

Scan to report a
safeguarding concern



or email the Designated
Safeguarding Lead:
Ian Wyles

safeguarding@hyperiondev.com

Skills Bootcamp Data Science

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

Skills Bootcamp Data Science

- For all **non-academic questions**, please submit a query:
www.hyperiondev.com/support
- **Report a safeguarding incident:** **www.hyperiondev.com/safeguardreporting**
- We would love your feedback on lectures: **[Feedback on Lectures](#)**
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

Learning Outcomes

- ❖ **Describe** the structure of a basic neural network and its components.
- ❖ **Explain** how weights, biases, and activation functions influence outputs.
- ❖ **Train a neural network** to simulate logical decision-making (e.g., logic gates).
- ❖ **Adjust network parameters** (weights and biases) to achieve desired behaviour.
- ❖ **Evaluate the effectiveness** of a trained neural network model.

Lecture Overview

- Presentation of the Task
- Neural Networks
- Task Walkthrough



Task Walkthrough

Imagine you are working on an autonomous vehicle system. The car must decide whether to stop or go at a traffic light based on multiple inputs.

Your neural network will take three inputs:

- ❖ Traffic light color (Red = 1, Green = 0)
- ❖ Presence of a pedestrian at the crossing (Yes = 1, No = 0)
- ❖ Car's current speed (Scaled between 0 and 1)

The network will output a decision:



1 (STOP) → If stopping is required

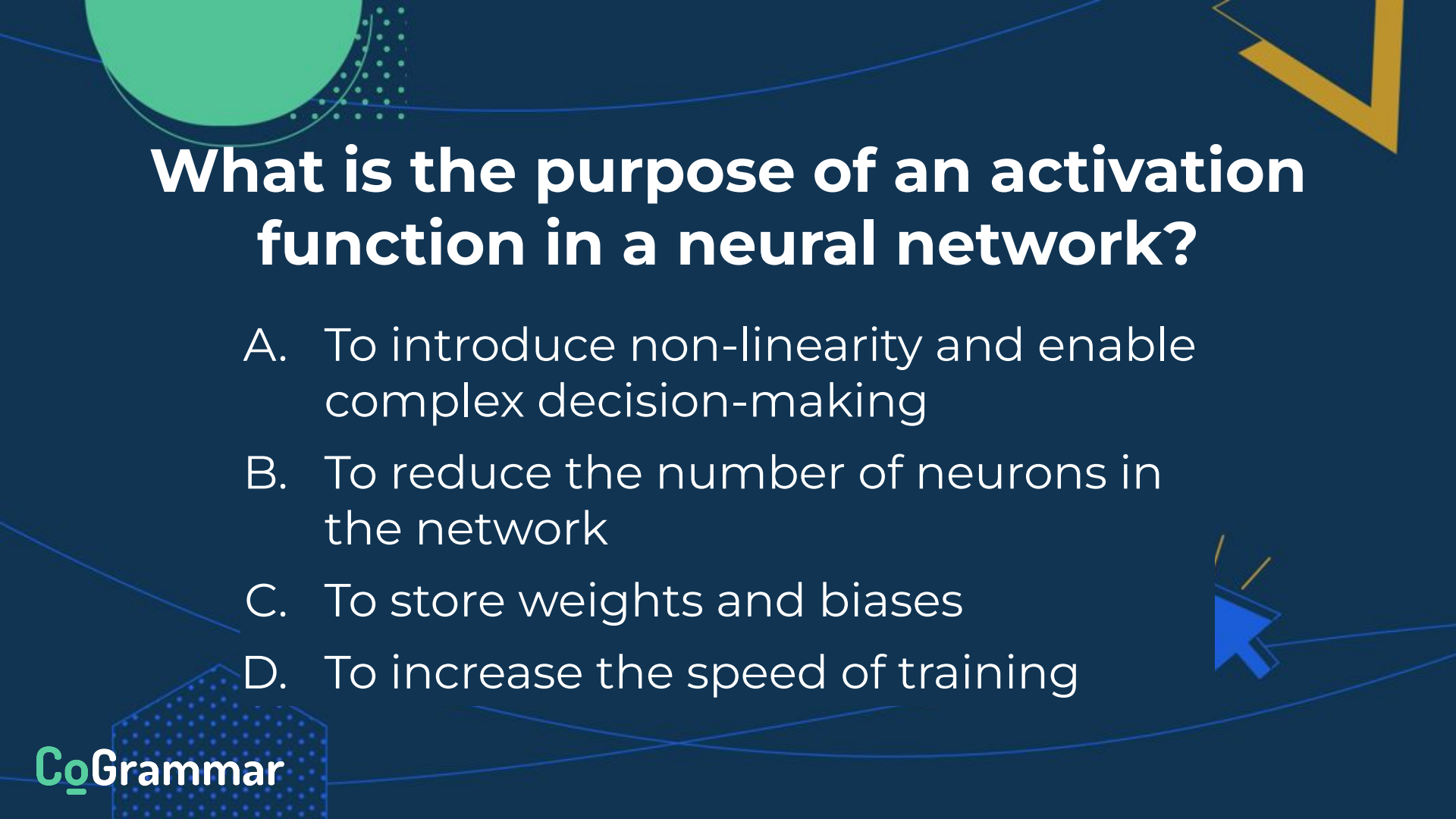


0 (GO) → If the car can continue driving



What is the fundamental building block of an artificial neural network?

- A. If-else statements
- B. Neurons
- C. Loops
- D. Databases



What is the purpose of an activation function in a neural network?


- A. To introduce non-linearity and enable complex decision-making
- B. To reduce the number of neurons in the network
- C. To store weights and biases
- D. To increase the speed of training

Neural Networks

```
def train(self, data_loader):\n    for epoch in range(1, self.epochs + 1):\n        for batch_idx, (data, target) in enumerate(data_loader):\n            # Zero the parameter gradients\n            self.optimizer.zero_grad()\n            # Forward pass: data into the model\n            output = self.model(data)\n            # Compute the loss and its backpropagation\n            loss = self.criterion(output, target)\n            loss.backward()\n            # Add parameters to the optimizer\n            self.optimizer.step()\n            # Print the loss every 10 batches\n            if batch_idx % 10 == 0:\n                print('Epoch: {} Batch: {} Loss: {}'.format(epoch, batch_idx, loss.item()))\n\n# Create the data loader\ntrain_loader = DataLoader(train_data_loader)\n\n# Create the model\nmodel = SimpleNN()\n\n# Create the optimizer\noptimizer = optim.Adam(model.parameters())\n\n# Train the model\ntrainer = GradientDescentTrainer(model, train_loader, optimizer)\ntrainer.train()\n\n# Test the model\ntest_loader = DataLoader(test_data_loader)\ntest_loss = trainer.test(test_loader)
```

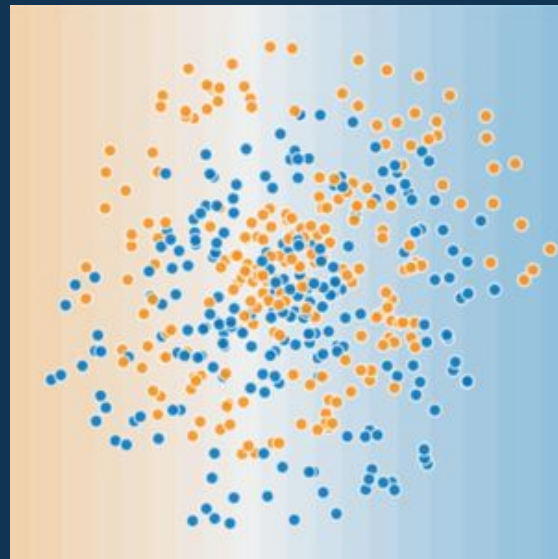


Neural Networks

- ❖ A **neural network (NN)** is a machine learning model inspired by the structure and function of the central nervous system in a human body.
 - ❖ Consist of interconnected **nodes or neurons**, processing units that pass data to each other, like in the brain, biological neurons pass electrical impulses to each other.
 - ❖ Enable tasks such as **pattern recognition** and **decision making** in machine learning.
- 

Importance of Neural Networks

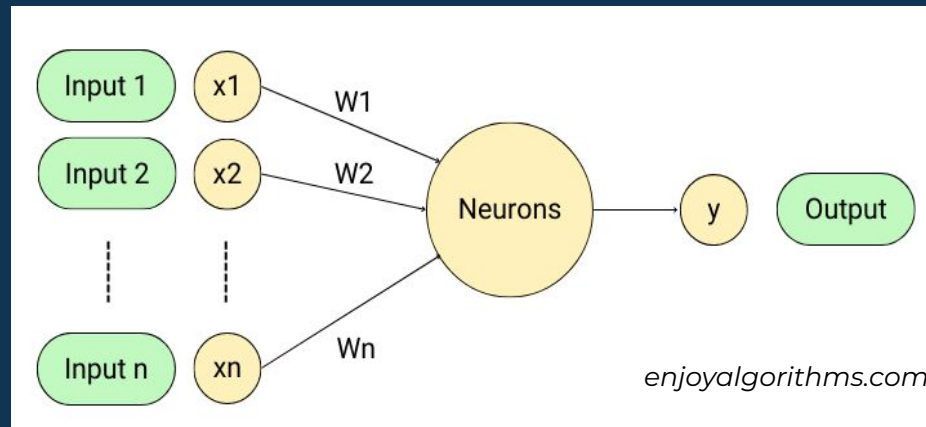
- ❖ Ideally suited to solve **complex and nonlinear** problems
- ❖ Make generalisations and inferences
- ❖ Reveal **hidden relationships, patterns** and **predictions**
- ❖ Model **highly volatile data** (financial time series data) and variances needed to **predict rare events** (fraud detection).



Non-linear classification

Neural Network Components

- ❖ Components include **neurons, connections, weights, biases, activation functions**.
- ❖ **Neurons** receive inputs, governed by **thresholds** and **activation functions**.
- ❖ **Connections** involve **weights** and **biases** regulating information transfer.
- ❖ Learning, adjusting weights and biases, occurs in **three stages: input computation, output generation, and iterative refinement** enhancing the network's proficiency in diverse tasks.



Example

Example: Your siblings have eaten a really great chicken soup at a restaurant and wants the chef in you to make it for them at home.

- ❖ You have 10 chances to recreate the soup, start preparing and every time you try the combination of recipe, you give it to your siblings to taste to get feedback and make changes each time, and finally you come with that perfect soup (that is the neural network model with desired output)
- ❖ Inputs to the NN: Ingredients for soup
- ❖ Weights: right quantity of salt, spice, temperature, duration of sauteing etc.
- ❖ Epochs or iterations = 10 chances to make the soup
- ❖ Feedback of your siblings = losses, make sure their feedback improves over time!

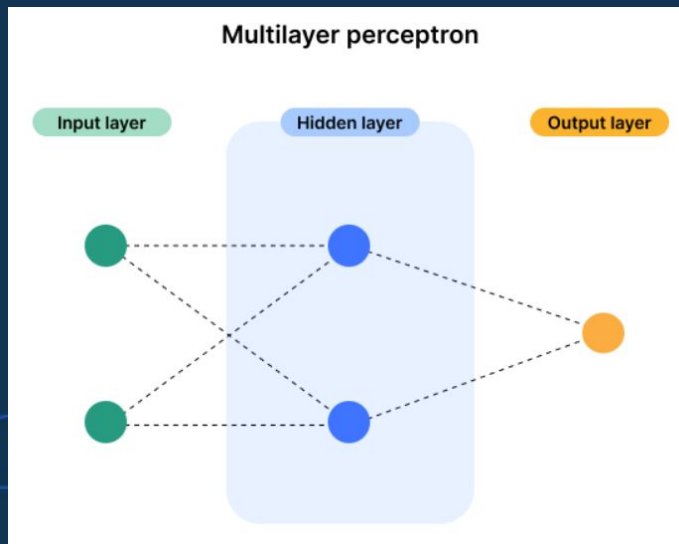
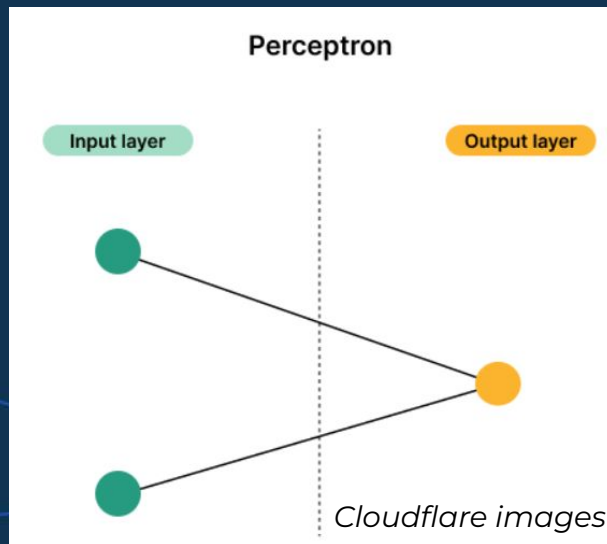
Types of Neural Network



The Perceptron

- ❖ **Perceptron** NNs are simple, shallow networks with an input layer and an output layer.
- ❖ **Multilayer perceptron (MLP)** NNs add complexity to perceptron networks, and include a hidden layer.

Forward Propagation



Neural Network Layers

- ❖ **Input layer:** number of neurons in the input layer is equal to the number of features in the data, sometimes one additional for bias.
- ❖ **Hidden layer/s:** intermediate layer/s between input and output layer where all the computation is done. If number of layers is
 - 0: Only capable of representing linearly separable data
 - **1 - 2:** Data is less complex and have fewer dimensions or features
 - More layers for optimum solution in large dimensions/many features
- ❖ **Output layer:** number of neurons depends on whether the model is a regressor (only one neuron) or classifier (one neuron for each class label).

Most common

Neural Network Layers

Number of neurons in hidden layers

- ❖ Few neurons in hidden layers can cause **underfitting**
- ❖ Too many neurons in the hidden layers may result in
 - **Overfitting**, limited training set cannot train all the neurons.
 - **Training time increases** with more neurons in the hidden layers.
- ❖ **Number of neurons in the hidden layer** (generally)
 - should be between input and output layer sizes.
 - should be $\frac{2}{3}$ the input layer size, plus the output layer size.
 - should be $\sqrt{\text{input layer nodes} * \text{output layer nodes}}$.
 - should keep on decreasing in subsequent layers to get more and more close to pattern and feature extraction and to identify the target class.

*Introduction to Neural
Networks with Java
Jeff Heaton*

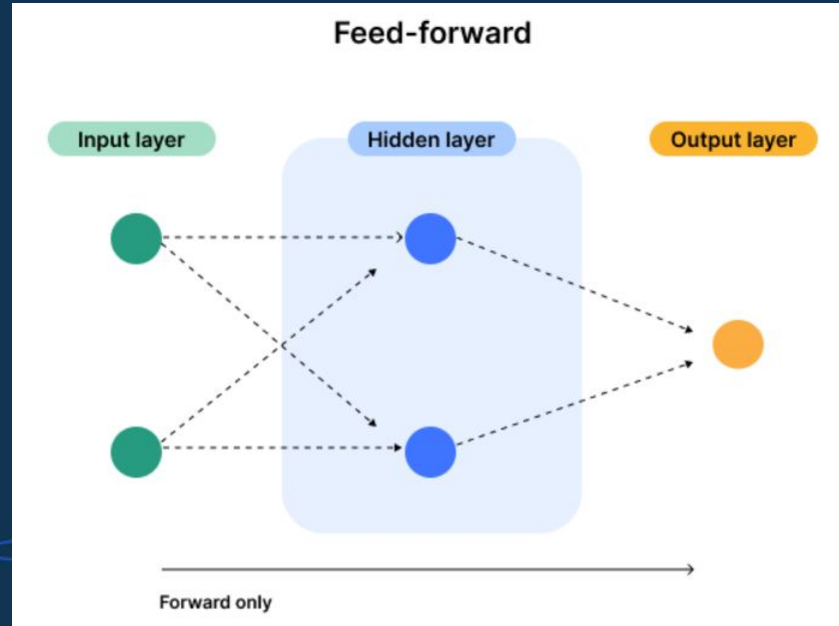
Feedforward Neural Networks

- ❖ **Feedforward Networks / Artificial Neural Networks (ANNs):** data moves from input to output in a single direction, with only input, hidden, and output layers, no feedback loops.

Mainly used for textual data or tabular data.

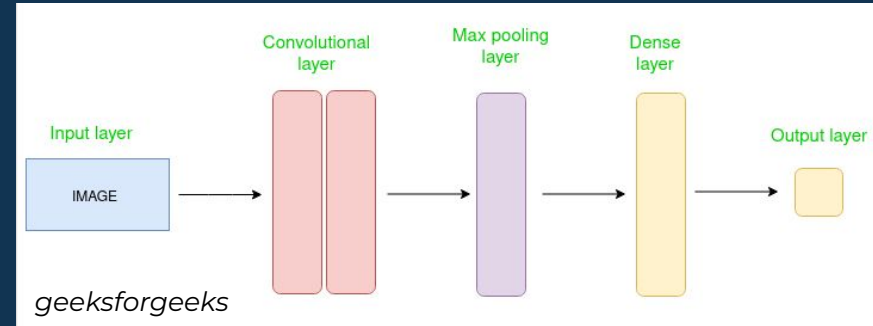
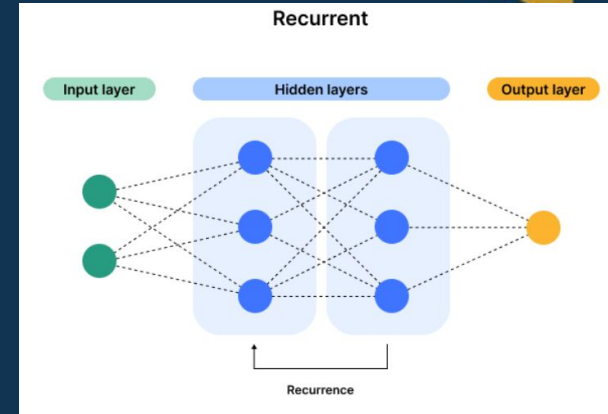
Widely used real-life application is facial recognition.

Comparatively less powerful than other complex neural networks.



Other Neural Networks

- ❖ **Recurrent neural networks (RNNs)** used where contextual dependencies are critical, e.g. **time series prediction**, **sequential data processing** and **natural language processing**; makes use of feedback loops, which enable information to survive within the network.
- ❖ **Long short-term memory (LSTM)** - type of RNN, uses memory cells and gates.
- ❖ **Convolutional Neural Networks (CNNs)** contain a combination of neurons and convolutional layers (automatically learn hierarchical features from input images); more powerful than both ANN and RNN; mainly used for **image data**, **computer vision**, object detection in autonomous vehicles.

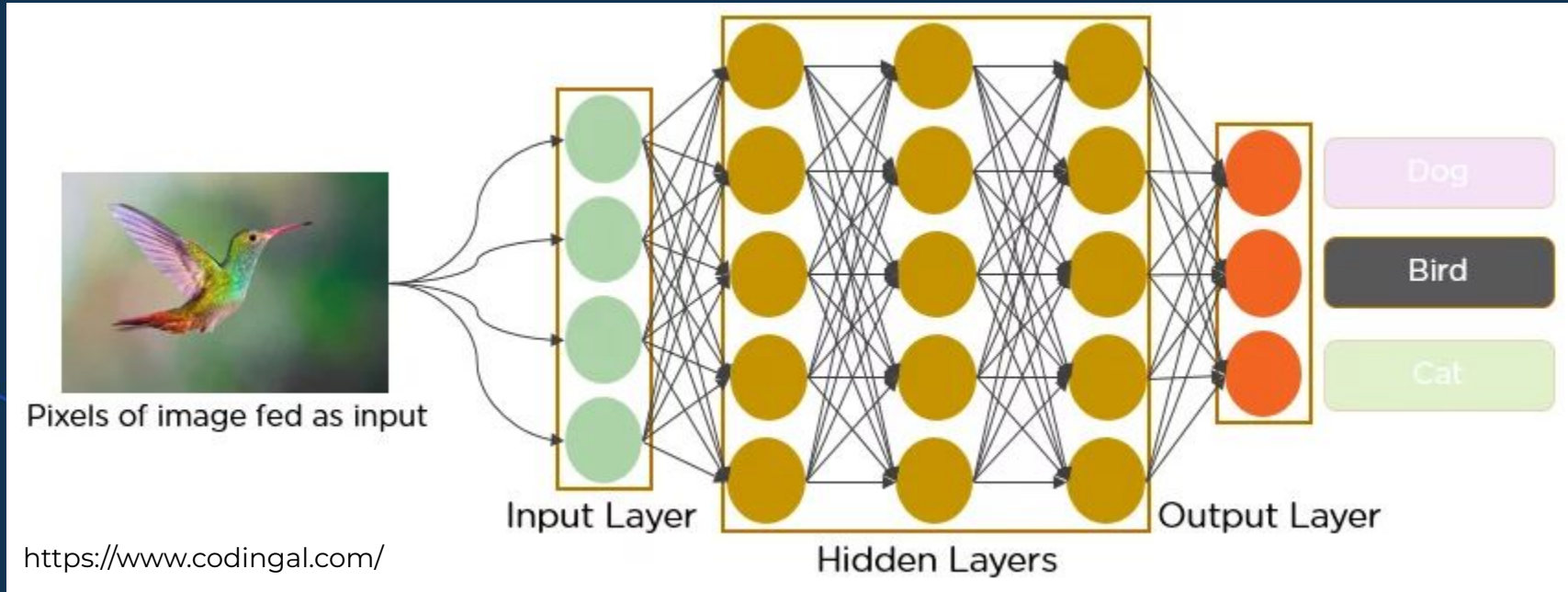


applies filters to extract features downsample image makes final prediction

How Neural Networks work



How Neural Networks work

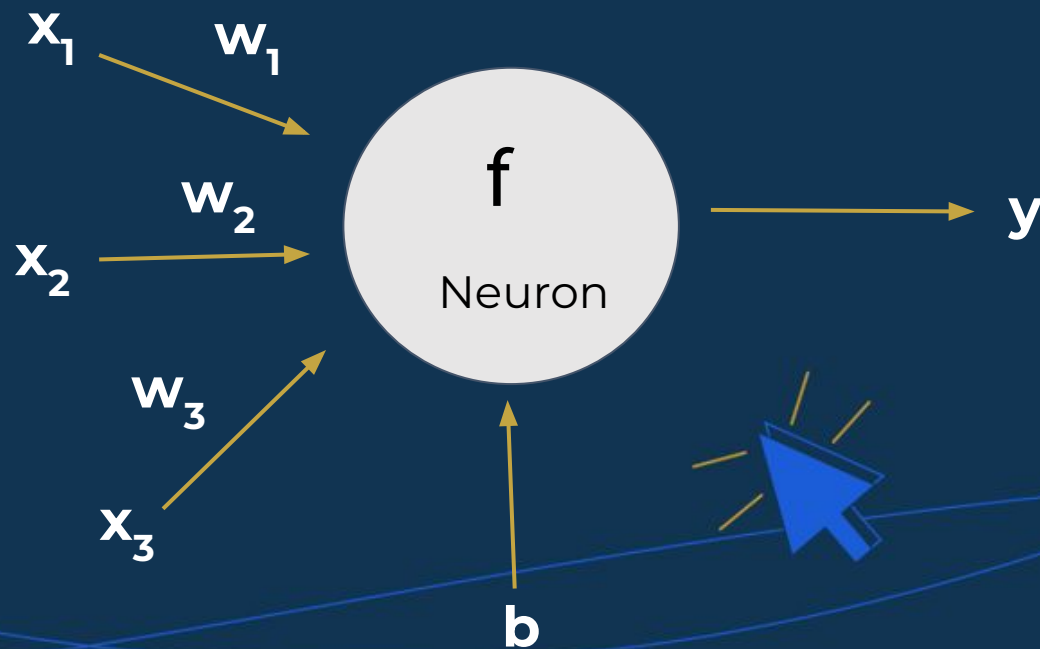


Key Concepts

Example

Output $y = f(x_1w_1 + x_2w_2 + x_3w_3 + b)$

- ❖ 1 neuron
- ❖ 3 inputs x_1, x_2, x_3
- ❖ 3 weights w_1, w_2, w_3
- ❖ Bias b
- ❖ Activation function f
- ❖ Output y



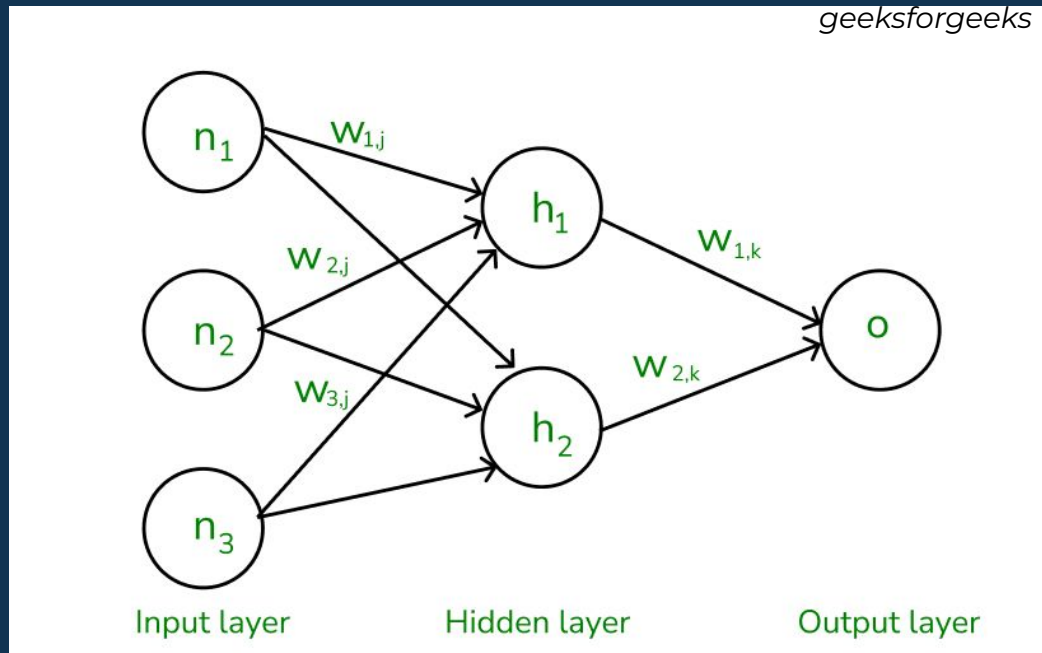
Key Concepts

- ❖ Weights w_{1j} , w_{2j} , and w_{3j} for connections between the three input nodes to the two hidden layers h_j ($j = 1$ and 2)

$$h_j = n_1 w_{1j} + n_2 w_{2j} + n_3 w_{3j}$$

- ❖ For the output layer

$$O = h_1 w_{1k} + h_2 w_{2k}$$



Weights and Bias



Weights and Bias

- ❖ **Weights (w)** and **biases (b)**: learnable parameters of NNs.
- ❖ Each neuron in a layer is connected to some or all of the neurons in the next layer.
- ❖ When the inputs are transmitted between neurons, the weights are applied to the inputs along with the bias.

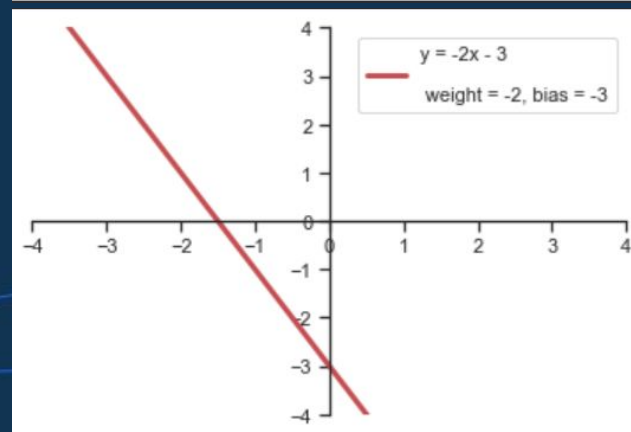
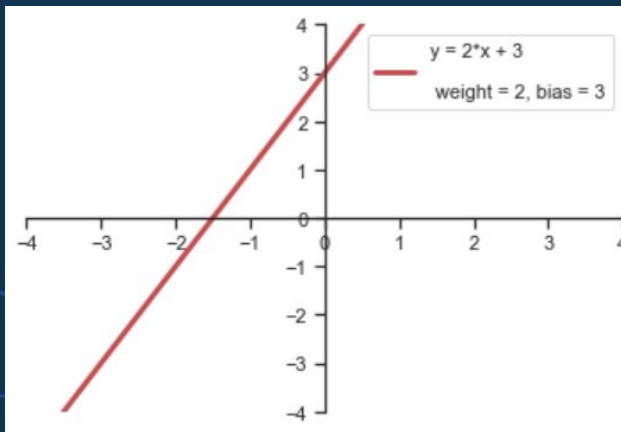
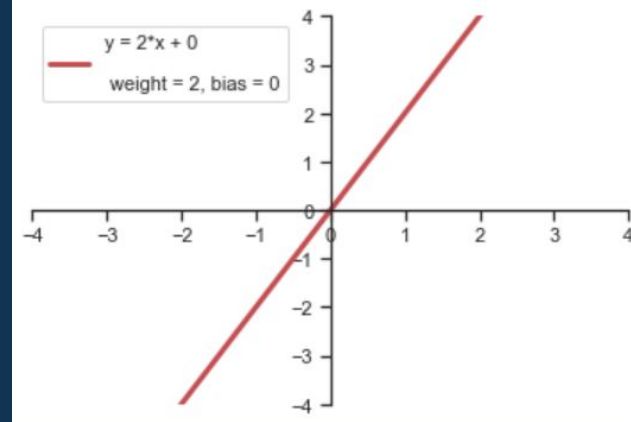
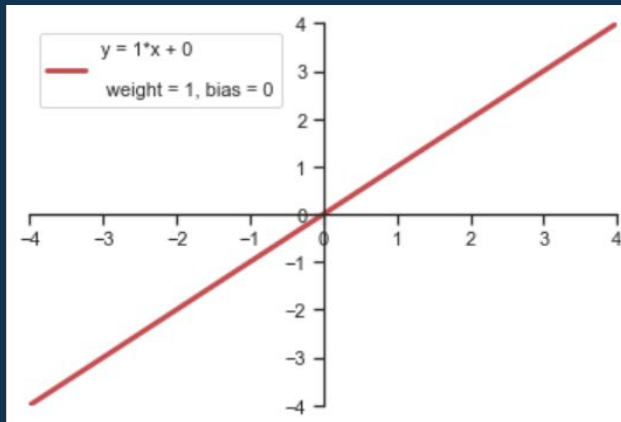
$$Y = \sum (\text{weight} * \text{input}) + \text{bias}$$

Weights and Bias

$$y = w \cdot x + b$$

Weights are multiplied with the input, they affect the magnitude of the latter.

Bias is added to the whole expression, moves the function in the dimensional plane.



Weights

- ❖ **Weights:** numerical values associated with strength of connections between neurons, impact of one neuron's output on another neuron's input, increase or decrease the importance of specific information.
- ❖ **Random initialisation,** assigned small random values, considers size of the previous layer of neurons to maintain a specific variance in outputs.
- ❖ **Adjusted iteratively** during training, **fine-tuning** the network's ability to make accurate predictions.
- ❖ **Examples**
 - Natural Language Processing: "I absolutely loved this book!" The word 'loved' should have more weight than 'absolutely' for predicting positive sentiment.
 - Image processing: Identify cat images, more weights to features like whiskers, ears, tail.

Bias

- ❖ **Biases**: constants associated with each neuron, unlike weights, biases are not connected to specific inputs but are added to the neuron's output. Provide **flexibility** to neural networks.
- ❖ Not influenced by the previous layer, can be **initialised** to zero.
- ❖ Offset or threshold, guarantees that even when all the weighted sum of inputs is zero, there will still be an activation in the neuron.
- ❖ Example
 - NLP: "I enjoyed the book" offering variations of positive sentiments.
 - Image recognition: Flexibility to identify cat features with different poses, lighting, backgrounds.

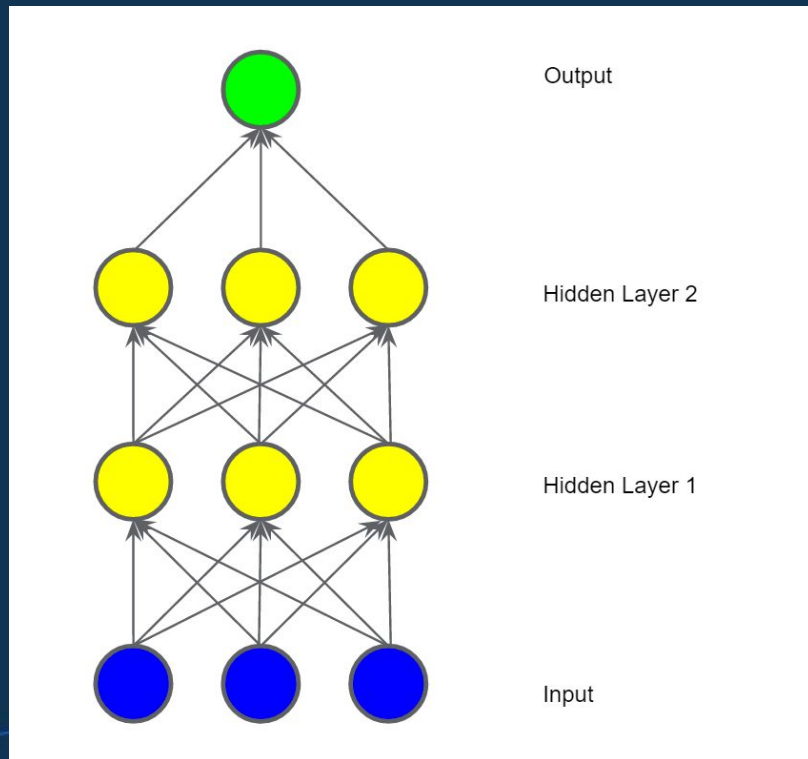
Activation functions



Why an activation function?

Linear model as a graph

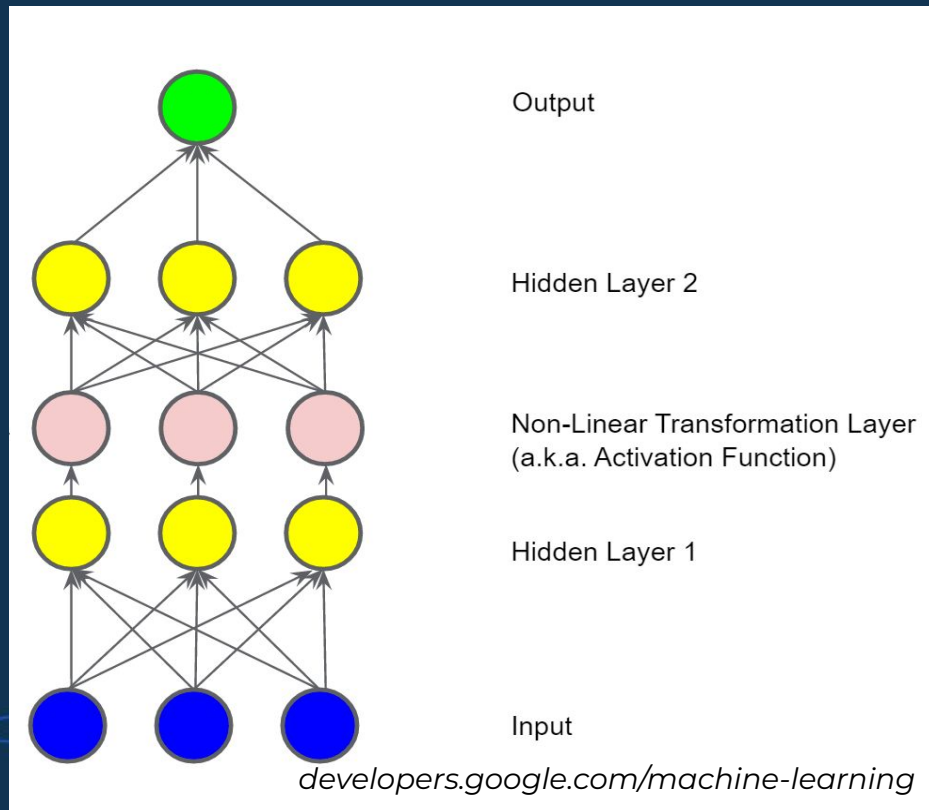
- ❖ **Blue circles:** represent inputs.
- ❖ **Yellow circles:** nodes in hidden layers, weighted sum of blue input node values.
- ❖ **Green circles:** output, weighted sum of the yellow hidden nodes.
- ❖ On simplification, output is **linear combination/weighted** sum of **inputs**. Cannot be used for **non-linear** problems.



Why an activation function?

To introduce **non-linearity** to model **non-linear problems**

- ❖ Pipe each hidden layer node through a nonlinear function.
- ❖ The value of each node in Hidden Layer 1 is transformed by a nonlinear function before being passed on to the weighted sums of the next layer.
- ❖ This nonlinear function is called the **activation function**.

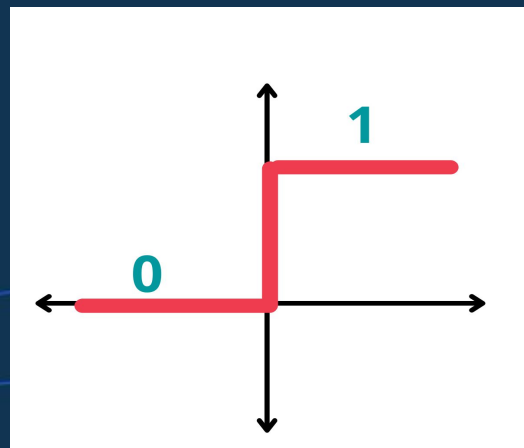


Activation Functions

- ❖ An artificial neuron calculates a “weighted sum” of its input, adds a bias and then decides whether it should be “fired” (activated) or not.

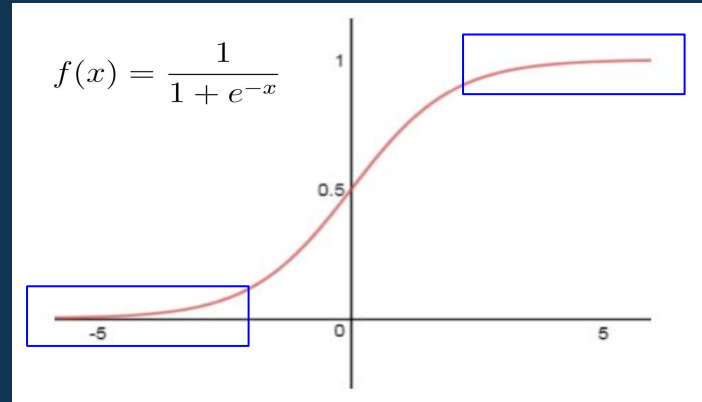
$$Y = \sum (weight * input) + bias$$

- ❖ Threshold based activation function: If Y value is above a certain threshold value, neuron is activated; if less, then it is not.
- ❖ Kind of like **a step function**, $y = f(x) = 1$ if $x > 0$ and $y = f(x) = 0$ if $x < 0$.
- ❖ However, too **simplistic**, and **discontinuous**. Not suitable for real-world examples.



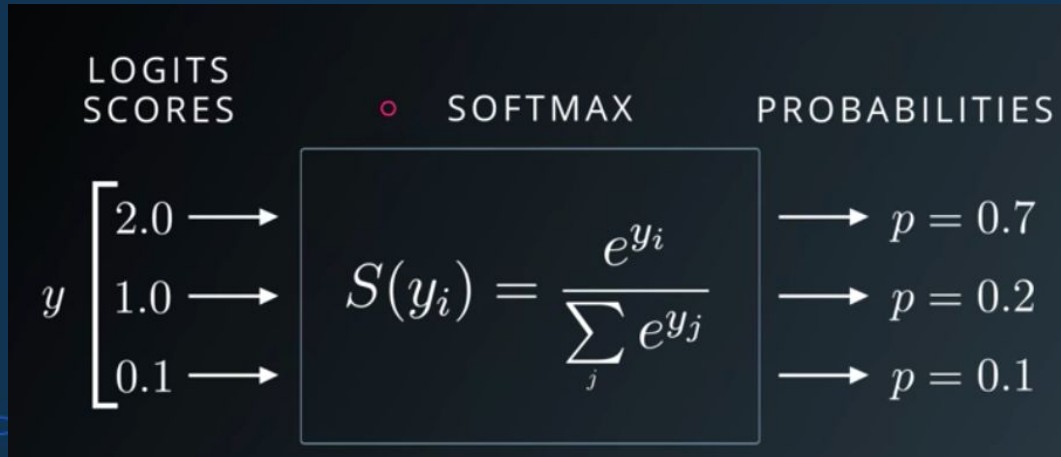
Sigmoid Activation Function

- ❖ **Sigmoid** is a smooth function and is continuously differentiable.
- ❖ Non-linear, ranges from 0 - 1
- ❖ Mostly used in **binary classification problems**
- ❖ However, activation of **neurons saturates** either near **0 or 1** (blue areas), **derivative** of the sigmoid function becomes very **small**
- ❖ Function outputs are **not zero-centered**. Training the neural network is more difficult and unstable.



Softmax Activation Function

- ❖ **Softmax** comes from the family of **Sigmoid** functions
- ❖ Used for **multi-class classification** problems
- ❖ Turns numbers (logits) into **probabilities** that **sum to one**.
- ❖ Outputs vector representing **probability distributions** of a list of potential outcomes



Tanh Activation Function

❖ E.g. Inputs get **multiplied by 3** at each node

❖ Some values can explode and become astronomical, causing others to seem insignificant.

5	15	45	135	405
0,01	0,03	0,09	0,27	0,81
-0,5	-1,5	-4,5	-13,5	-40,5

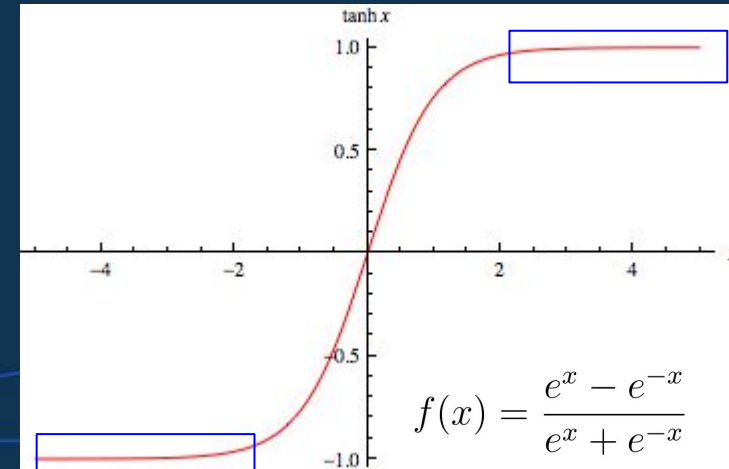
$x \times 3$

❖ **Tanh activation function** regulates values in **between -1 and +1**

$\tanh(x \times 3)$

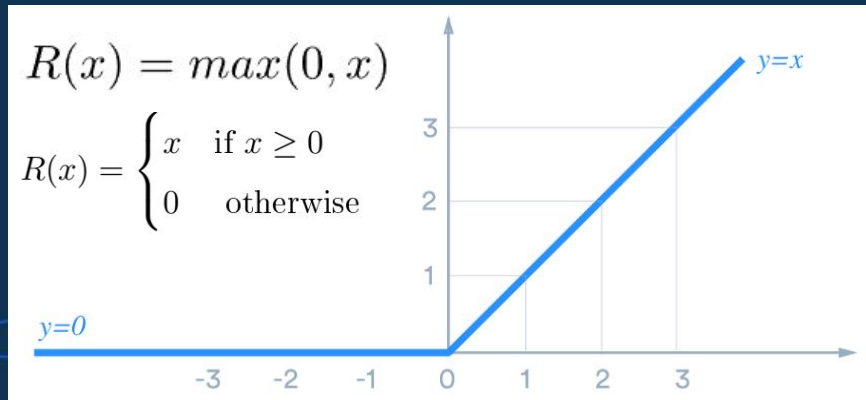
5	1	0,995	0,995	0,995
0,01	0,030	0,090	0,263	0,658
-0,5	-0,905	-0,991	-0,995	-0,995

❖ **Saturation issue** present, but **centered around zero**, preferred over Sigmoid



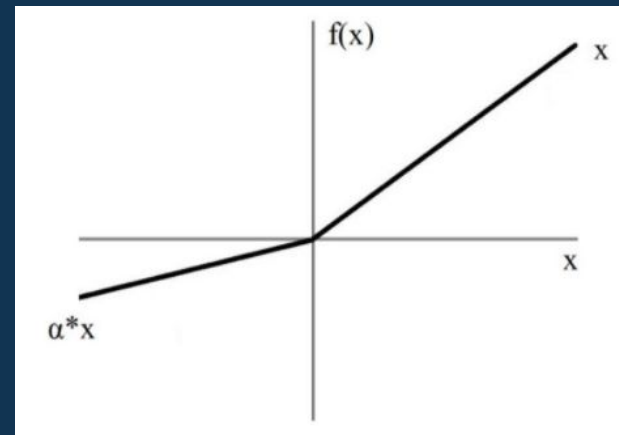
ReLU Activation Function

- ❖ **Rectified linear unit (ReLU):** most widely used activation function.
- ❖ Computationally cheap, less time to train, converges faster.
- ❖ Linearity ensures the slope does not plateau, or “saturate,” for large x
- ❖ No vanishing gradient problem suffered by other activation functions
- ❖ Downside: zero for all negative values - **“dying ReLU”**, a neuron is “dead” if stuck in the negative side and always outputs 0. Learning rate is too high or there is a large negative bias.



Leaky ReLU Activation Function

- ❖ **Leaky ReLU:** Improved version of ReLU activation function, to prevent **dying ReLU** problem
- ❖ **Small linear component** for **negative** values, avoid zero gradients, no “dead” neurons contributing.
- ❖ Others: **Parametric ReLU** (fine-tunes the activation function based on its learning rate)
- ❖ **Exponential Linear Unit (ELU)**
 - Smoother convergence for any negative values.
 - For positive output, behaves like a step function and gives a constant output.



$$R(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases}$$

Backpropagation



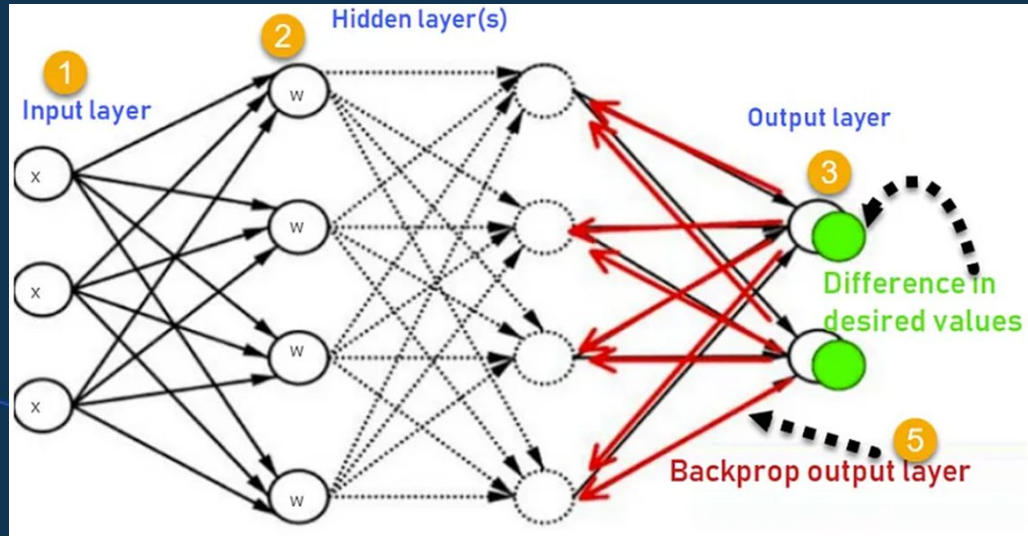
Backpropagation

- ❖ Backpropagation algorithm widely used to train feedforward NNs.
- ❖ Computes the **gradient** of the **loss function** with respect to the network weights layer by layer, and iterating backward from the last layer.
- ❖ Efficiently compute the gradient concerning each weight, to train multi-layer networks and **update weights** to **minimise loss**.
- ❖ **Gradient descent** or **stochastic gradient descent** estimation method used by the optimisation algorithm to compute the network parameter updates and train neural network models.

Lecture 2

Backpropagation

- ❖ Neuron N_i at layer i is connected to another neuron N_{i+1} at layer $i + 1$
- ❖ Value of N_{i+1} is calculated according to the linear equation $N_{i+1} = w_i N_i + b$
- ❖ w_i - associated weight, b - bias



Backpropagation Steps

1. Inputs x , arrive through the preconnected connections.
2. Inputs are modeled using true weights w , usually chosen randomly.
3. Calculate each neuron's output from input to hidden layer/s to output layer.
4. Calculate error in outputs.
Backpropagation Error = Actual Output – Desired Output
5. From the output layer, go back to the hidden layer to **adjust the weights to reduce the error.**
6. Repeat the process until the desired output is achieved.

Lecture 2

Task Walkthrough

Imagine you are working on an autonomous vehicle system. The car must decide whether to stop or go at a traffic light based on multiple inputs.

Your neural network will take three inputs:

- ❖ Traffic light color (Red = 1, Green = 0)
- ❖ Presence of a pedestrian at the crossing (Yes = 1, No = 0)
- ❖ Car's current speed (Scaled between 0 and 1)

The network will output a decision:



1 (STOP) → If stopping is required

0 (GO) → If the car can continue driving



What happens if all activation functions in a neural network are removed?

- A. The network will still work normally
- B. The network will only learn linear relationships
- C. The network will train faster
- D. The network will become more complex



What is the primary role of backpropagation in training a neural network?

- A. It helps add more layers to the network
- B. It updates weights and biases to minimize error
- C. It increases the number of neurons
- D. It eliminates the need for training data

Summary

- ★ **Neural networks** are made up of **neurons, weights, biases, and activation functions.**
- ★ **Forward propagation** computes outputs based on inputs and weights.
- ★ **Logic gates** can be modelled **using simple neural networks.**
- ★ **Neural networks must be trained** using **backpropagation** to **adjust weights and improve accuracy.**
- ★ Real-world applications include **self-driving cars, speech recognition, and medical diagnosis.**

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**

Thank you for attending



CoGrammar



Department
for Education