EC3357:Machine Learning

Lecture 5: Neural Network Representation

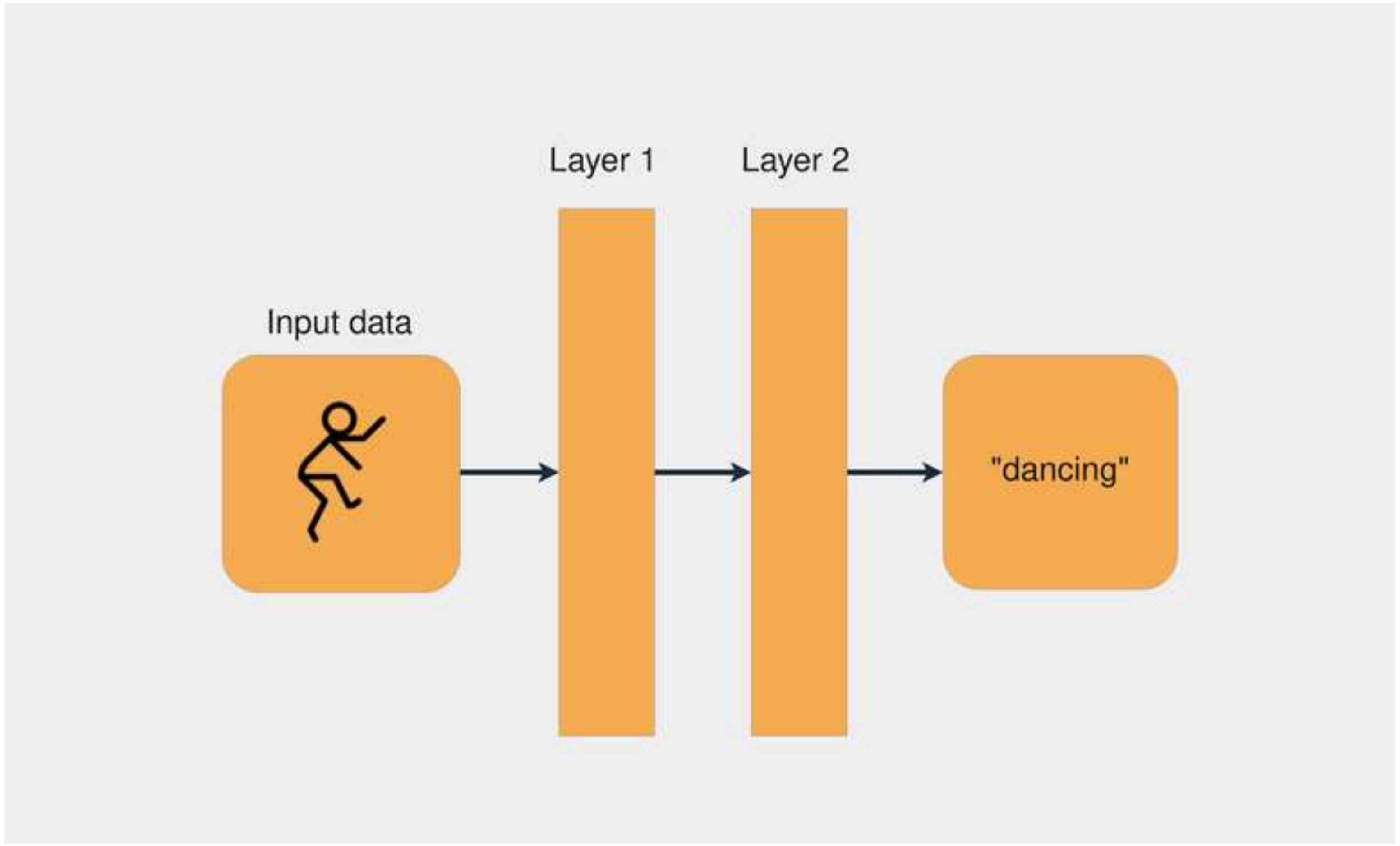
Introduction to Neural Network

- Artificial Neuron is a basic building block of the neural network.
- Computational models inspired by the human brain:
  - Algorithms that try to mimic the brain.
  - Massively parallel, distributed system, made up of simple processing units (neurons)
  - Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process
- Examples:
  - Speech phoneme recognition
  - Image classification
  - Financial prediction
- In simple words, Neural Networks are a set of algorithms that tries to recognize the patterns, relationships, and information from the data through the process which is inspired by and works like the human brain/biology.

Neural Networks: Main Concepts

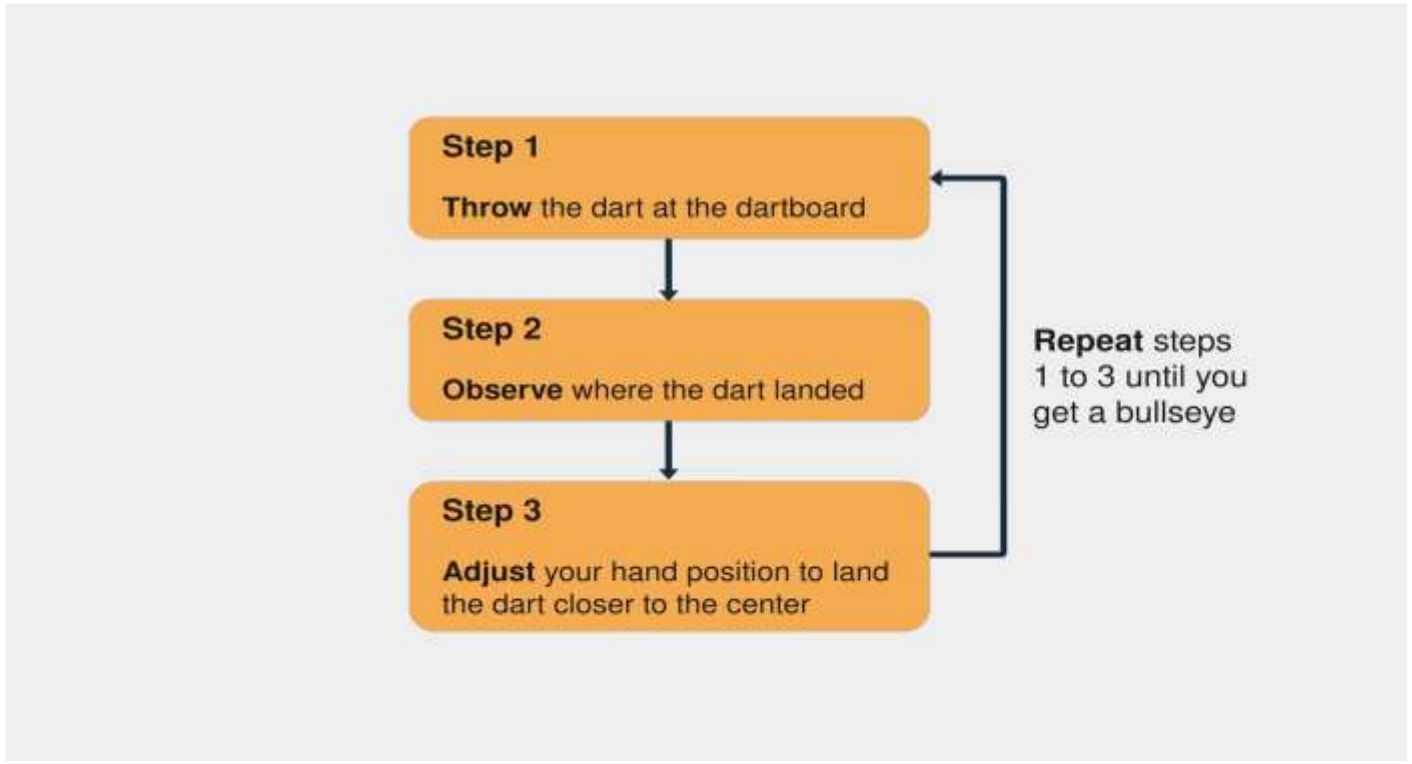
- A neural network is a system that learns how to make predictions by following these steps:
    1. Taking the input data
    2. Making a prediction
    3. Comparing the prediction to the desired output
    4. Adjusting its internal state to predict correctly the next time
- **Vectors**, **layers**, and **linear regression** are some of the building blocks of neural networks.
- The data is stored as vectors, and with Python we store these vectors in arrays.
- Each layer transforms the data that comes from the previous layer.

In the image below, shows an example of a network architecture with two layers:



Each layer transforms the data that came from the previous layer by applying some mathematical operations.

The Process to Train a Neural Network


Step 1
Throw the dart at the dartboard
Step 2
Observe where the dart landed
Step 3
Adjust your hand position to land the dart closer to the center
Repeat steps 1 to 3 until you get a bullseye

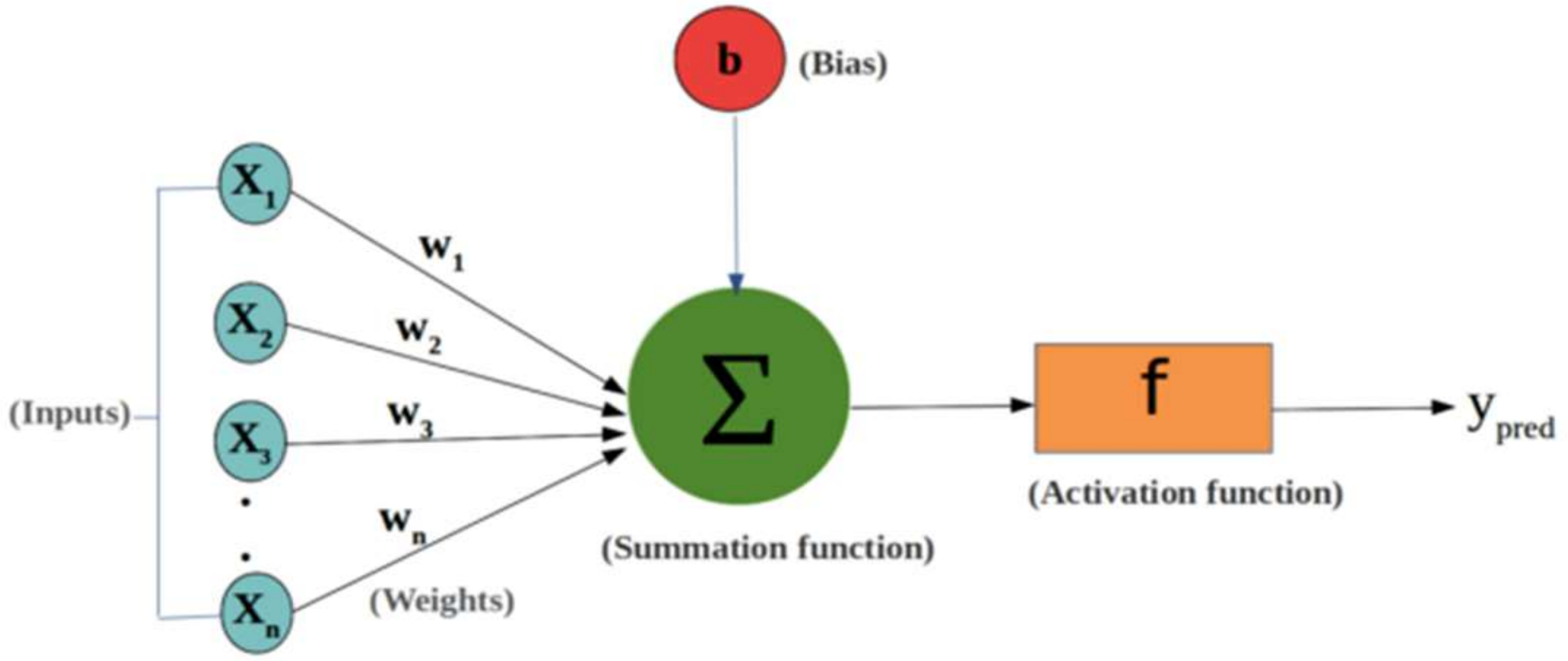
With neural networks, the process is very similar: we start with some random weights and bias vectors, make a prediction, compare it to the desired output, and adjust the vectors to predict more accurately the next time. The process continues until the difference between the prediction and the correct targets is minimal.

Knowing when to stop the training and what accuracy target to set is an important aspect of training neural networks, mainly because of overfitting and underfitting scenarios.

Vectors and Weights, The Linear Regression Model

- By modeling the relationship between the variables as linear, you can express the dependent variable as a weighted sum of the independent variables.
- So, each independent variable will be multiplied by a vector called weight. Besides the weights and the independent variables, you also add another vector: the bias.
- It sets the result when all the other independent variables are equal to zero. Example:
    - price = (weights_area * area) + (weights_age * age) + bias
- In the above example, there are two weights: weights_area and weights_age. The training process consists of adjusting the weights and the bias so the model can predict the correct price value. To accomplish that, you'll need to compute the prediction error and update the weights accordingly.

Skeleton of this Artificial Neuron.



Components of the basic Artificial Neuron:

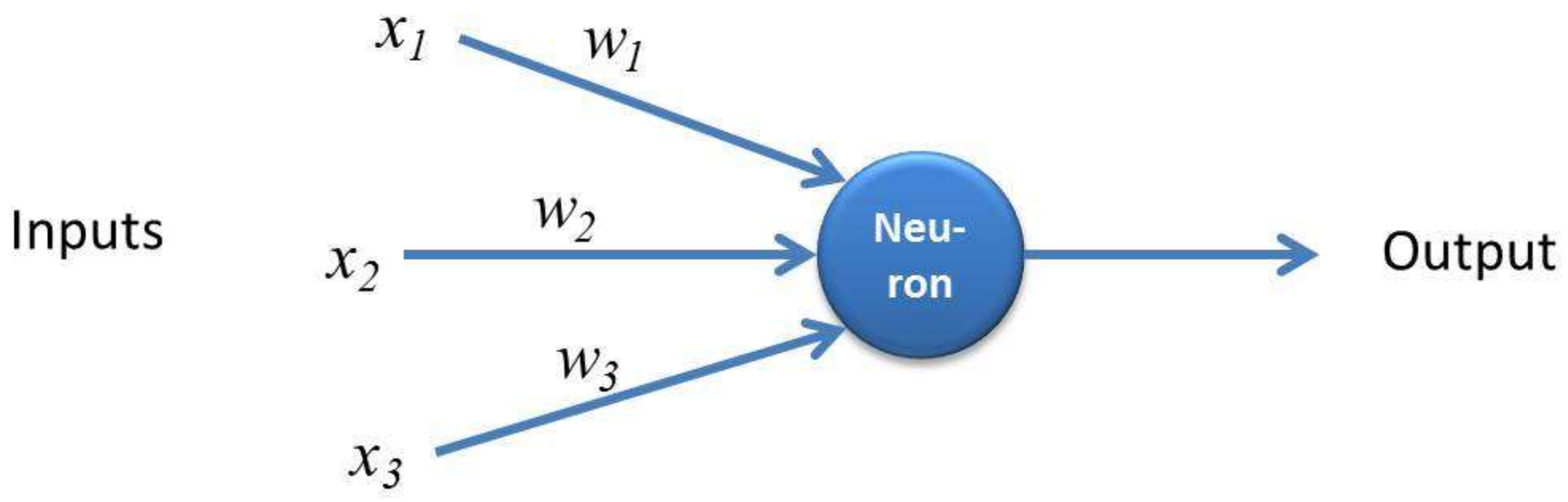
- Inputs: Inputs are the set of values for which we need to predict the output value. They can be viewed as features or attributes in a dataset.
- Weights: weights are the real values that are associated with each feature which tells the importance of that feature in predicting the final value.
- Bias: Bias is used for shifting the activation function towards left or right, it can be referred to as a y-intercept in the line equation.
- Summation Function: The work of the summation function is to bind the weights and inputs together and find their sum.
- Activation Function: It is used to introduce non-linearity in the model. This non-linearity allows the neural network to learn complex patterns and relationships within the data.

1. In the first step, Input units are passed i.e data is passed with some weights attached to it to the hidden layer. We can have any number of hidden layers. In the previous image inputs $x_1,x_2,x_3,....x_n$ is passed.
2. Each hidden layer consists of neurons. All the inputs are connected to each neuron.
3. After passing on the inputs, all the computation is performed in the hidden layer.
4. The whole process described in point 3 is performed in each hidden layer. After passing through every hidden layer, we move to the last layer i.e our output layer which gives us the final output.
5. After getting the predictions from the output layer, the error is calculated i.e the difference between the actual and the predicted output.

If the error is large, then the steps are taken to minimize the error and for the same purpose, Back Propagation is performed.

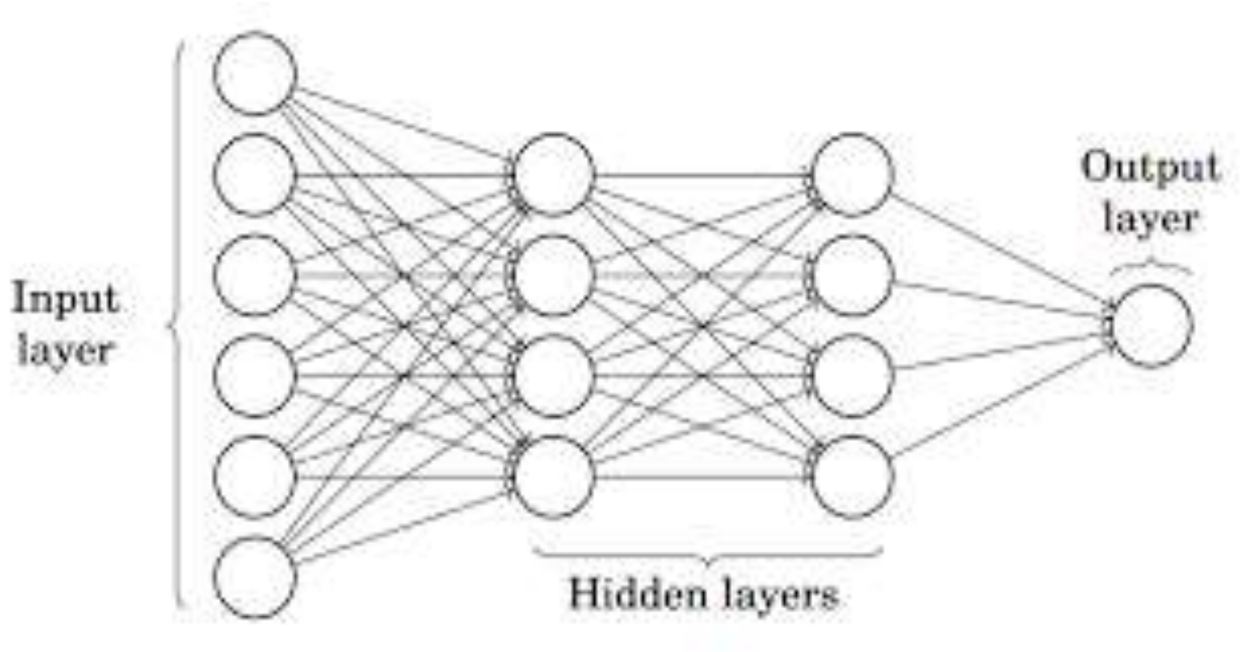
A Neuron (= a perceptron)

Perceptron is a simple form of Neural Network and consists of a single layer where all the mathematical computations are performed.



Multilayer Perceptron

- Multilayer Perceptron also known as Artificial Neural Networks consists of more than one perception which is grouped together to form a multiple layer neural network.



- In the above image, The Artificial Neural Network consists of four layers interconnected with each other:
  - An input layer, with 6 input nodes
  - Hidden Layer 1, with 4 hidden nodes/4 perceptions
  - Hidden layer 2, with 4 hidden nodes
  - Output layer with 1 output node

Artificial Neural Networks

- An ANN can:
  1. compute *any computable* function, by the appropriate selection of the network topology and weights values.
  2. learn from experience!
  - Specifically, by trial-and-error

Learning by trial-and-error

Continuous process of:

- Trial:

Processing an input to produce an output (In terms of ANN: Compute the output function of a given input)

- Evaluate:

Evaluating this output by comparing the actual output with the expected output.

- Adjust:

Adjust the *weights.*

How it works?

This is a description of the steps involved in training a neural network to learn the XOR function using a basic algorithm like the perceptron learning algorithm or more commonly, a simple form of gradient descent in a multi-layer perceptron (MLP). Here’s a detailed explanation of each step:

- Set initial values of the weights randomly.
- Input: truth table of the XOR
    - Do
        - Read input (e.g. 0, and 0)
        - Compute an output (e.g. 0.60543)
        - Compare it to the expected output. (Diff= 0.60543)
        - Modify the weights *accordingly*.
- Loop until a condition is met
- Condition: certain number of iterations
- Condition: error threshold

Set Initial Values of the Weights Randomly:

- Initialize the weights of the neural network with small random values. This randomness helps to break symmetry and allows the network to learn different patterns.
- Example: If we have a neural network with weights $w1$,$w2$,$w3$, they might be initialized to random values like 0.1, -0.2, and 0.05. Initial weights are set to small random values within the range [-1, 1]

Input: truth table of the XOR:

- The XOR truth table consists of all possible input combinations of two binary variables and their corresponding outputs:

Input	Output
0 0	0
0 1	1
1 0	1
1 1	0

Do:

- This starts the training loop, which will run until one of the stopping conditions is met.
  - Read input (e.g., 0 and 0):
    - For each iteration, pick a sample from the truth table. Let's start with (0, 0).
  - Compute an output (e.g., 0.60543):
    - The neural network processes the input through its layers to produce an output. This involves computing the weighted sum of the inputs and passing it through an activation function.
    - For example, if the input is (0, 0) and the network produces an output of 0.60543, this is an initial guess based on the random weights.
  - Compare it to the expected output. (Diff = 0.60543):
    - Calculate the error by comparing the network's output to the expected output from the truth table.
    - For input (0, 0), the expected output is 0. The error (Diff) 0.60543−0 = 0.60543
  - Modify the weights accordingly:
    - Adjust the weights to minimize the error. This is typically done using the backpropagation algorithm in a multi-layer perceptron (MLP).

Loop until a condition is met:

- Repeat the process of reading inputs, computing outputs, comparing with expected outputs, and modifying weights.
- Condition: certain number of iterations:
  - Stop the training after a predefined number of iterations (epochs). For example, we might train for 10,000 iterations.
- Condition: error threshold:
  - Alternatively, stop the training when the error across all training examples is below a certain threshold. For instance, if the mean squared error falls below 0.01, we can consider the network sufficiently trained.

Design Issues

- Initial weights (small random values ∈[-1,1])
- Transfer function/Activation Function (How the inputs and the weights are combined to produce output?)
- Error estimation
- Weights adjusting
- Number of neurons
- Data representation
- Size of training set

Transfer Functions

- **Linear:** The output is proportional to the total weighted input.
- **Threshold:** The output is set at one of two values, depending on whether the total weighted input is greater than or less than some threshold value.
- **Non-linear:** The output varies continuously but not linearly as the input changes.

Error Estimation

- Error estimation in Artificial Neural Networks (ANNs) is crucial for understanding how well the model is performing and for guiding improvements.
- The **Mean square error (MSE) and Root Mean Square Error** are frequently-used measures of the differences between values predicted by a model or an estimator and the values actually observed from the thing being modeled or estimated.
- These error estimation techniques help in evaluating and comparing the performance of different neural network models, guiding model selection, tuning, and improvement.

Weights Adjusting

- After each iteration, weights should be adjusted to minimize the error. The goal is to find the optimal set of weights that minimizes the error between the predicted outputs and the actual outputs.
- All possible weights:

  This implies considering the entire weight space, which consists of all potential values that the weights can take. In practice, it's not feasible to examine every possible combination of weights due to the immense size of the weight space, especially in large networks. Instead, optimization algorithms are used to iteratively adjust the weights in a way that reduces the error.
- Back propagation:

  This is a common and efficient method for training ANNs. Backpropagation stands for "backward propagation of errors"

Back Propagation

- Back Propagation is the process of updating and finding the optimal values of weights or coefficients which helps the model to minimize the error i.e difference between the actual and predicted values.

- But here are the question is: How the weights are updated and new weights are calculated?

- The weights are updated with the help of optimizers. Optimizers are the methods/ mathematical formulations to change the attributes of neural networks i.e weights to minimizer the error.

Backpropagation

- Involves the following steps:
  - Forward Pass: Input data is passed through the network, and the output is computed using the current weights.
  - Error Calculation: The error (difference between the predicted output and the actual output) is calculated using a loss function such as Mean Squared Error (MSE).
  - Backward Pass: The error is propagated back through the network. During this phase, the partial derivatives of the error with respect to each weight are calculated. This is done using the chain rule of calculus, which allows the computation of gradients for each weight.
  - Weight Update: Weights are adjusted using an optimization algorithm such as Gradient Descent. The weights are updated in the direction that reduces the error, which is determined by the gradients computed during the backward pass.
- By iteratively adjusting the weights in this manner, the neural network learns to produce outputs that are closer to the actual values, thereby minimizing the error.

Number of neurons

- Many neurons:
  - Higher accuracy
  - Slower
  - Risk of over-fitting
    - Memorizing, rather than understanding
    - The network will be useless with new problems.
- Few neurons:
    - Lower accuracy
    - Inability to learn at all
- Optimal number.
  - This typically involves experimenting with different network architectures and using techniques such as cross-validation to evaluate model performance. The optimal number of neurons allows the model to achieve good accuracy on both training and validation datasets, indicating good generalization.

Data representation

- Usually input/output data needs pre-processing-Data representation in Artificial Neural Networks (ANNs) refers to how input data is prepared, transformed, and presented to the network for training and inference. Proper data representation is crucial as it can significantly impact the performance and effectiveness of the model.
- Pictures
  - Pixel intensity
- Text:
  - A pattern

Size of training set

- No one-fits-all formula
- Over fitting can occur if a “good” training set is not chosen
- What constitutes a “good” training set?
  - Samples must represent the general population.
  - Samples must contain members of each class.
  - Samples in each class must contain a wide range of variations or noise effect.
- The size of the training set is related to the number of hidden neurons.
  - When deciding on the number of hidden neurons, it's important to consider the size of the training set. A balance needs to be struck between having enough neurons to learn the data's complexity and avoiding overfitting.

Ccomparing the design issues with the sample program

Concept	Explanation (From the sample code)
Initial Weights	Small random values between [−1,1][-1,1][−1,1], adjusted during training.
Activation Function	Uses **ReLU** in hidden layers to introduce non-linearity.
Error Estimation	Uses **Cross-Entropy Loss** to measure incorrect predictions.
Weights Adjusting	Uses **Gradient Descent & Backpropagation** to reduce errors.
Number of Neurons	**10 neurons** in the hidden layer help the model learn relationships.
Data Representation	NumPy array **(Experience, Salary, Label)**, standardized for better training.
Size of Training Set	**8 training samples, 2 testing samples**. Larger datasets improve accuracy.