Discover how these computational models enable machines to learn, recognize patterns, and make intelligent decisions.

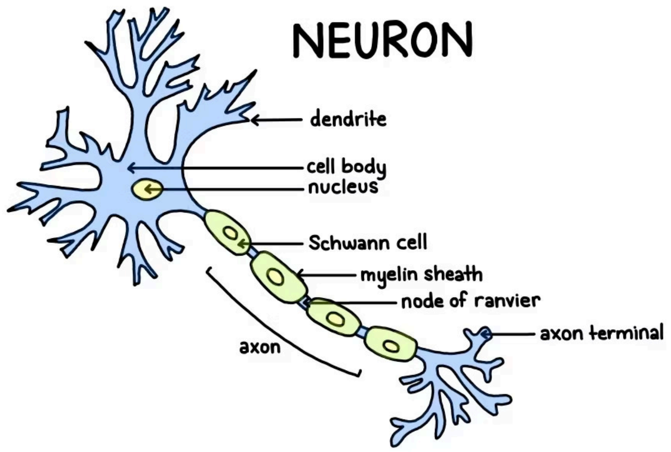
Group Members:

- Hiba Mahdi
- Sara Ali
- Zeenah Shamil
- Noor-AlHuda Ayad

Introduction to ANN

ANN algorithms are based on the structure of the brain neural network, the big picture in neural networks is how we go from having some data, throwing it into some algorithm and hoping for the best. But what happens inside that algorithm? This question is important to answer, for many reasons; one is that you otherwise "might just regard the inner workings of a neural network as a black box".

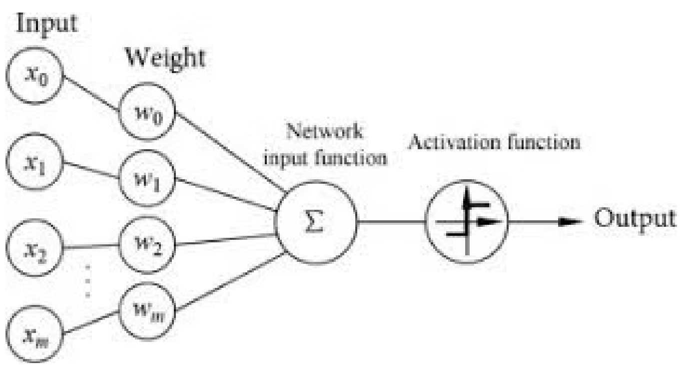
What is neural network?



A neural network is an algorithm inspired by the neurons in our brain. It is **designed to recognize patterns in complex data** and often **performs the best when recognizing patterns in text, audio, images or video**.

- Such algorithms “learn” to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition task, they learn to identify images that contain cats by analyzing example images that have been manually labeled as “cat” or “no cat” and using the results “trained model” to identify cats in other images.

Perceptron Model



Perceptron algorithm is a simplest ANN architecture used for supervised learning of binary classifiers as single-layer neural network. They consist of four main parts including **input values**, **weights** and **bias**, **net sum**, and an **activation function**.

$$w_1a_1 + w_2a_2 + w_3a_3 + w_4a_4 + \cdots + w_na_n$$

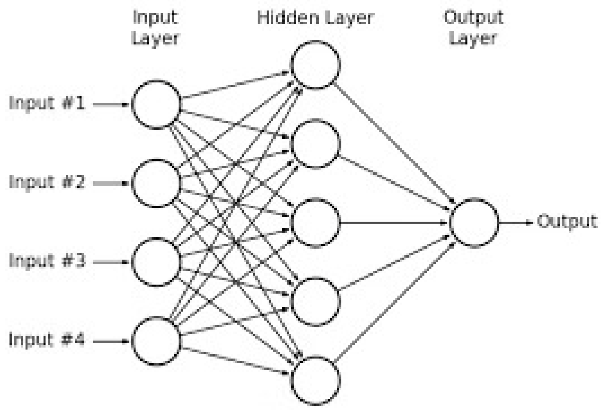
How does a Perceptron work?

The perceptron works by taking a set of input features, along with a constant 'bias' term and assigning weights to these inputs. The linear combination of the weights and inputs are taken. This linear combination is then fed through an activation function that determines which class the set of inputs belongs to.

NOTE: Weight values give the strength to the input features and will determine the influence input data has on the output product.

While bias value gives the ability to shift the activation function curve up or down.

Multi-layer Perceptron



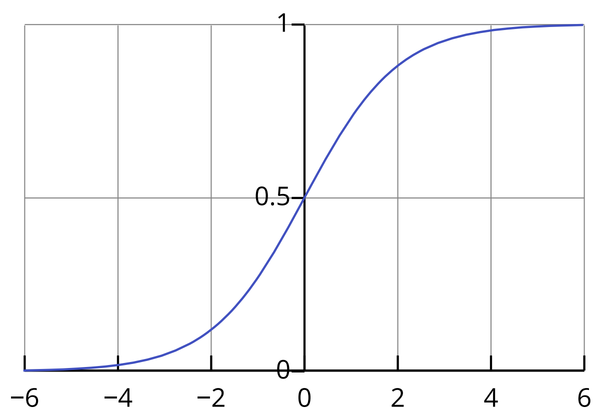
- Input Layer- receives the input as data from the dataset.
- Output Layer - Give the output as a prediction from the model.
- Hidden Layers- Any layer between Input and Output layer.

A single perceptron won't be enough to learn complicated systems. Thus for complicated problems, we need to expand the single perceptron, to create a multi-layer perceptron model by connecting layers of perceptrons.

Activation Function

is a small but crucial component inside each neuron of a neural network. After the neuron adds up all its inputs, it decides what the output should be. It's a "gatekeeper" that determines whether and how strongly the neuron should "fire."

Sigmoid or Logistic Activation Function



Sigmoid functions are mathematical functions that share similar properties and they have S-shaped curves.

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

It maps the feature space into probability functions.

- The main reason we use sigmoid function is because it exists between (0 to 1).
  Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1.

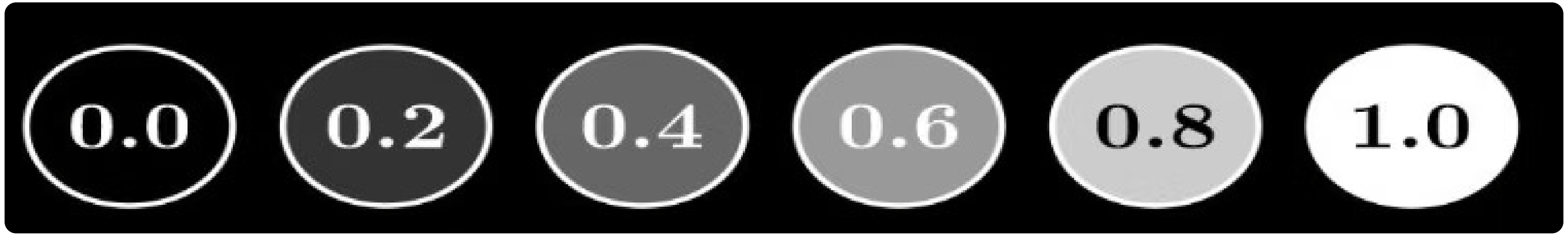
When x is really large (goes to infinity), the output will be close to 1
When x is really small (goes to -infinity), the output will be close to 0
When x is 0, the output will be ½

It uses exponential

That means a nonlinear relationship ensures most points to be either close to 0 or 1.

Neurons Activation in ANN

- When we say neuron, all we want to think about it is “a thing that holds a ***number***. Specifically, a number between ***0.0 and 1.0***. Neural networks are really just a bunch of neurons connected together.
- This number inside the neuron is called the “***activation***” of that neuron, and the image you might have in your mind is that each neuron is lit up when its ***activation is a high number***.



The Structure of a Neural Network

Fully-connected Feedforward Network (FFN)

The Input Layer

Purpose: This is the **entry** point for your data. Its job is to receive the gene expression values for a single patient sample.

- **Structure:** It consists of **100 neurons**.
- **Why 100?** Because in our feature selection step intelligently reduced the initial 7,129 genes down to the 100 most statistically significant ones. Each neuron in this input layer corresponds to one of these top 100 genes.

The Hidden Layers

Purpose: are considered as the "brain" of our network. They are responsible for finding the complex, non-linear patterns in the gene data that distinguish ALL from AML. The multi-layer structure allows the network to learn hierarchical patterns; the first layer might learn simple associations, and the subsequent layers combine those to learn more complex signatures.

- **Structure:**
  - **Three separate hidden layers** arranged sequentially.
  - Each layer contains **10 neurons**.
  - Each neuron in these layers uses the **Sigmoid activation function**. This function takes the neuron's calculated output and squashes it to a value between 0 and 1 before passing it to the next layer.

The Output Layer

Purpose: This is the final layer that produces the actual prediction.

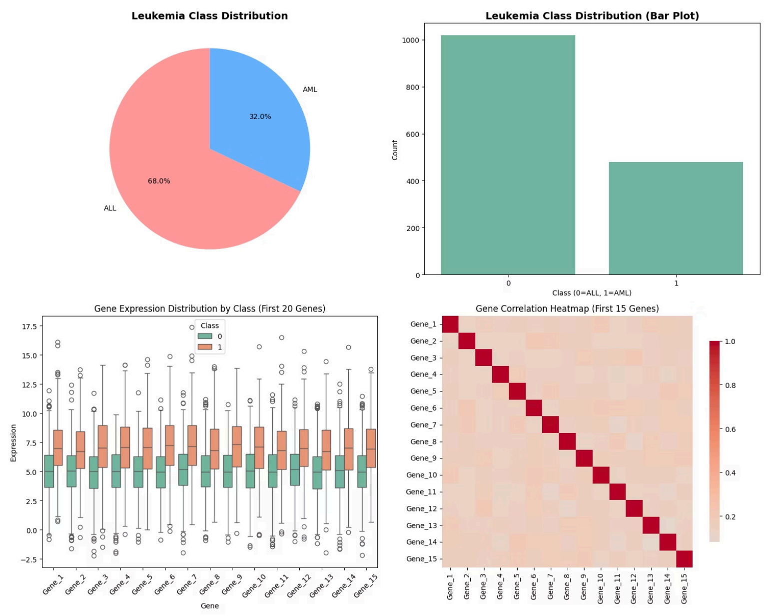
- **Structure:** It contains only **one single neuron**.
- **Why one?** Because our problem is a binary classification (a yes/no question: Is this sample AML or not?).
- **Activation Function:** This neuron also uses the **Sigmoid activation function**. This is a crucial choice because it forces the final output to be a number between 0 and 1.

Layer	No. of Neurons	Activation Function	Purpose
Input	100	(None)	Receives the 100 most important gene values.
Hidden Layer 1	10	Sigmoid	Learns initial patterns from the gene data.
Hidden Layer 2	10	Sigmoid	Combines patterns to learn more complex features.
Hidden Layer 3	10	Sigmoid	Further refines the learned features for prediction.
Output	1	Sigmoid	Produces the final probability of the sample being AML.

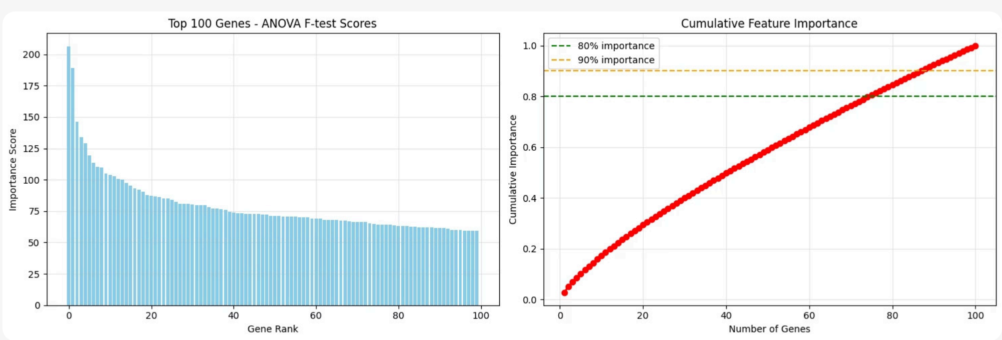
`dropout_rate=0.3`	Parameter	Fraction of neurons to drop randomly during training.	Prevents overfitting by forcing the network to generalize better. (0.3 = 30% of neurons dropped).

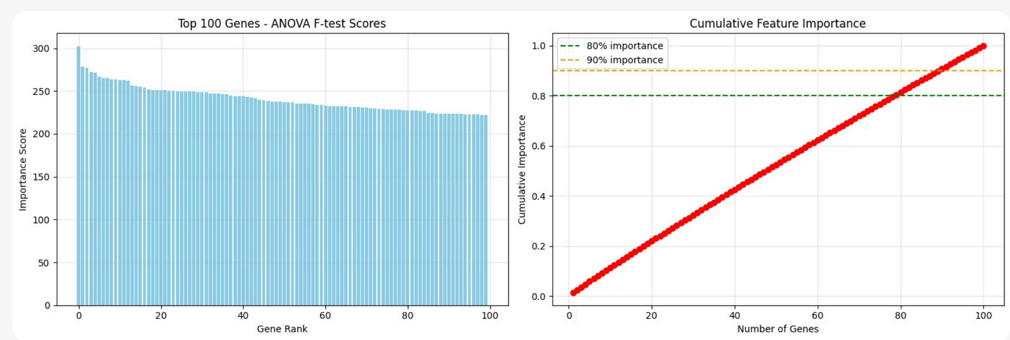
Layer	No. of Neurons	Activation Function	Purpose
Input	100	(None)	Receives the 100 most important gene values.
Hidden Layer 1	10	ReLU	Learns non-linear relationships between input genes and leukemia class.
Dropout	0.3	Randomly deactivates 30% of neurons from the previous layer.	Regularizes the model, reducing overfitting and improving generalization.
Hidden Layer 2	10	ReLU	Adds deeper abstraction and captures more complex feature interactions.
Dropout	0.3	Applies dropout again after second dense layer.	Provides additional regularization to avoid memorizing training data.
Hidden Layer 3	5	ReLU	Gradually reduces network size to help the model compress information before output.
Output	1	Sigmoid	Produces a probability (0–1) for binary classification (ALL vs AML).

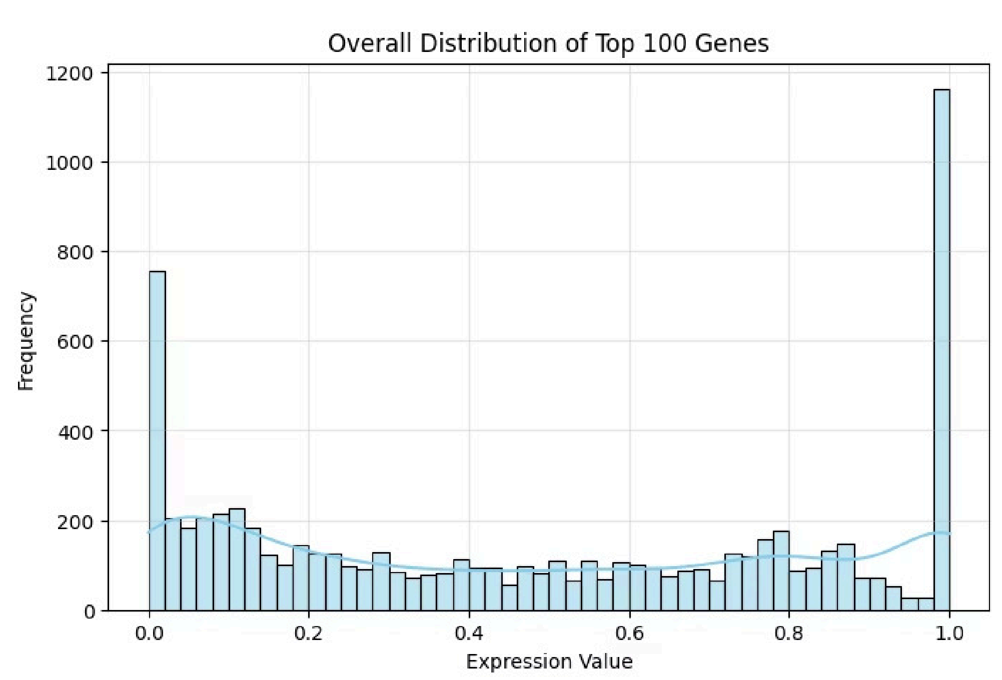
Return	`return model`	Returns the compiled Keras Sequential model.	Makes the function reusable — you can call it with different input sizes or hyperparameters.

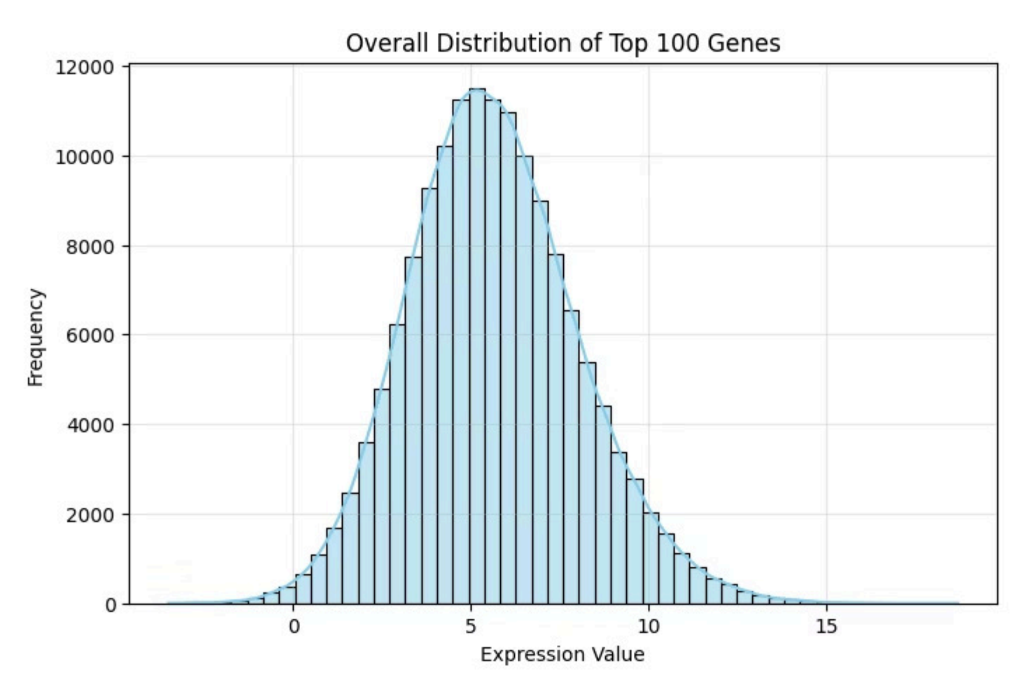


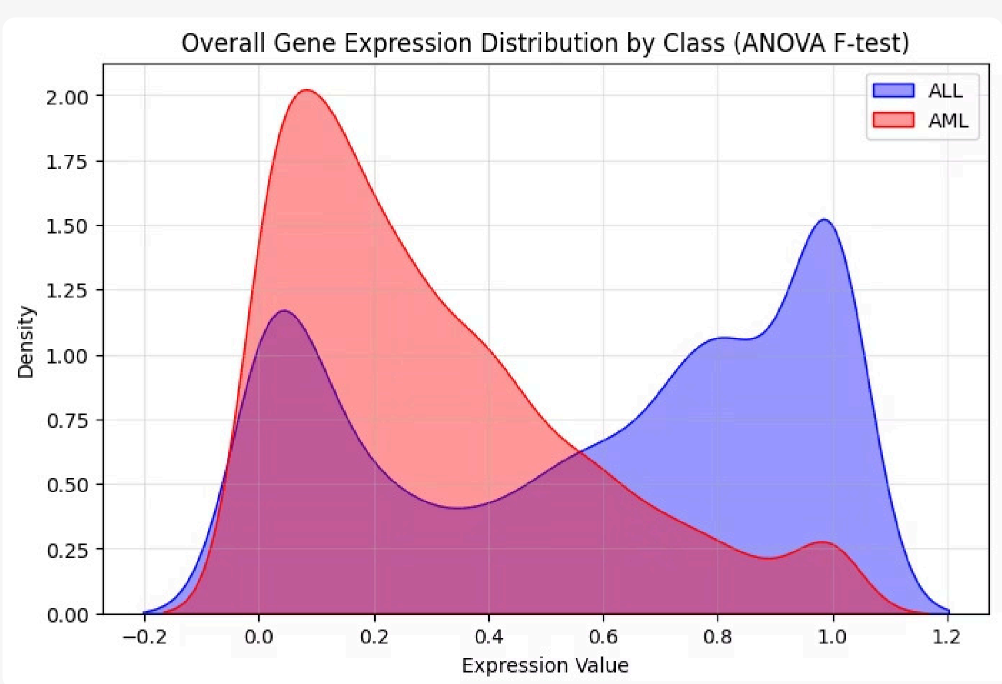
Old Database	New Database
Dataset Shape: (72, 7130)	Dataset Shape: (1500, 501)
Class distribution: Leukemia_class 0 (ALL) 49 1 (AML) 23	Class distribution: Leukemia_class 0 (ALL) 1020 1 (AML) 480



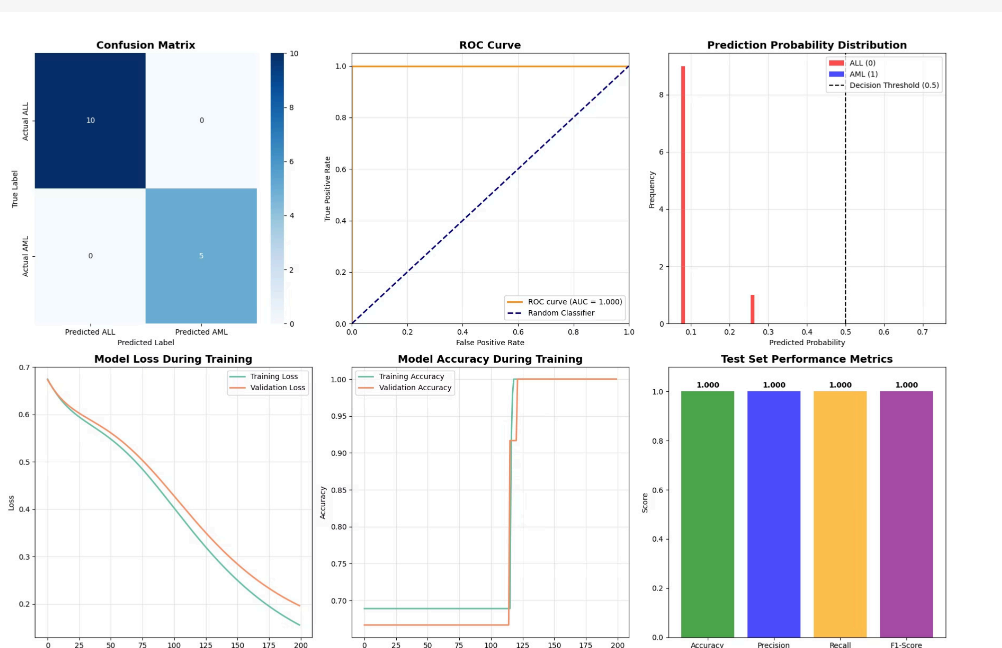


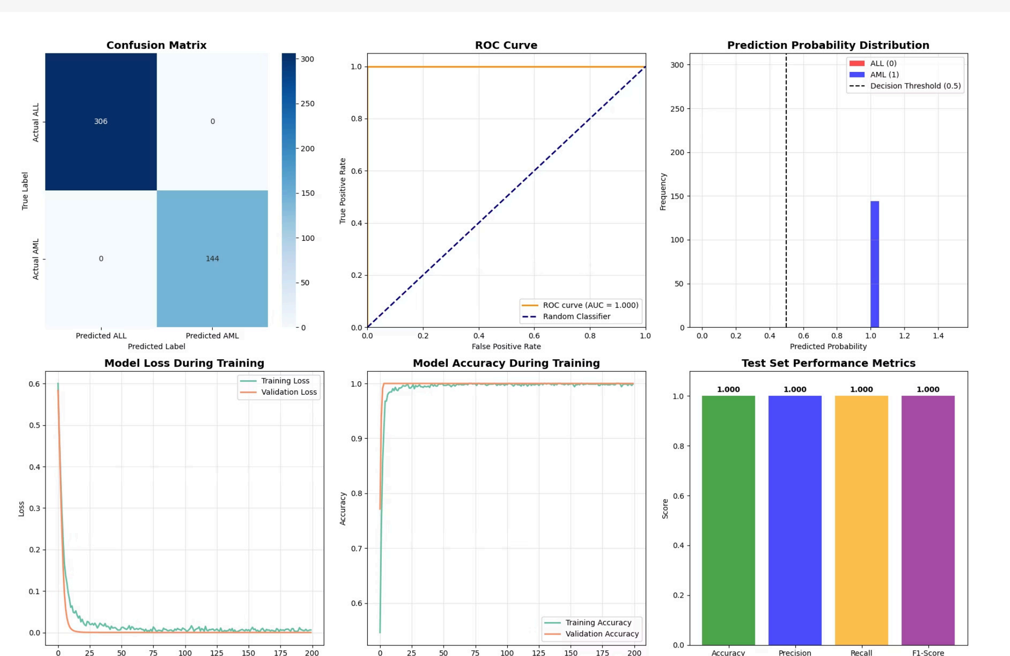






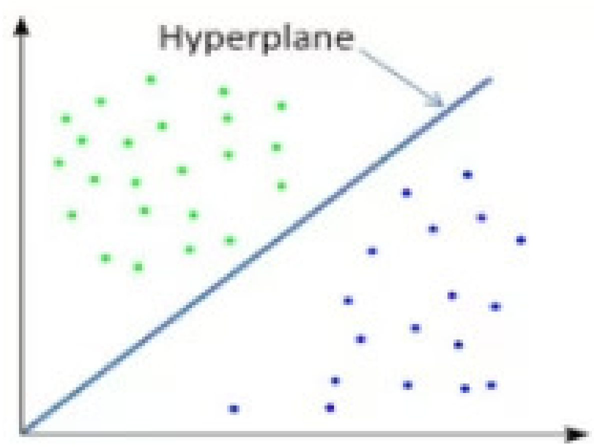
Error uploading image.





Choosing the Hidden Layers

How to choose the number of hidden layer and their nodes?



- If the ***data*** is linearly separable, then you don't need any hidden layers because ***the output layer already calculates a linear combination of features and outputs a number that has discriminative power***.

- If the ***data*** is less complex and is having ***fewer dimensions*** or features then neural networks with ***1 to 2 hidden layers*** would work.
- If the ***data*** is complex and having ***large dimensions*** or features then to get an optimum solution, ***3 to 5 hidden layers*** can be used.

Note: It should be kept in mind that increasing hidden layers would also increase the complexity of the model and choosing hidden layers such as 8, 9, or more may sometimes lead to over-fitting (good performance on the training data, poor generalization to other data "validating or testing").

How to choose the number of hidden nodes

- The number of hidden neurons should be ***between the size of the input layer and the output layer.***
- The ***most appropriate number of hidden neurons*** is sqrt (input layer nodes * output layer nodes)

Input Layer

Receive raw features and signals

Weighted Sum

Compute weighted inputs per neuron

Activation Function

Apply nonlinearity to produce output

Hidden Layers

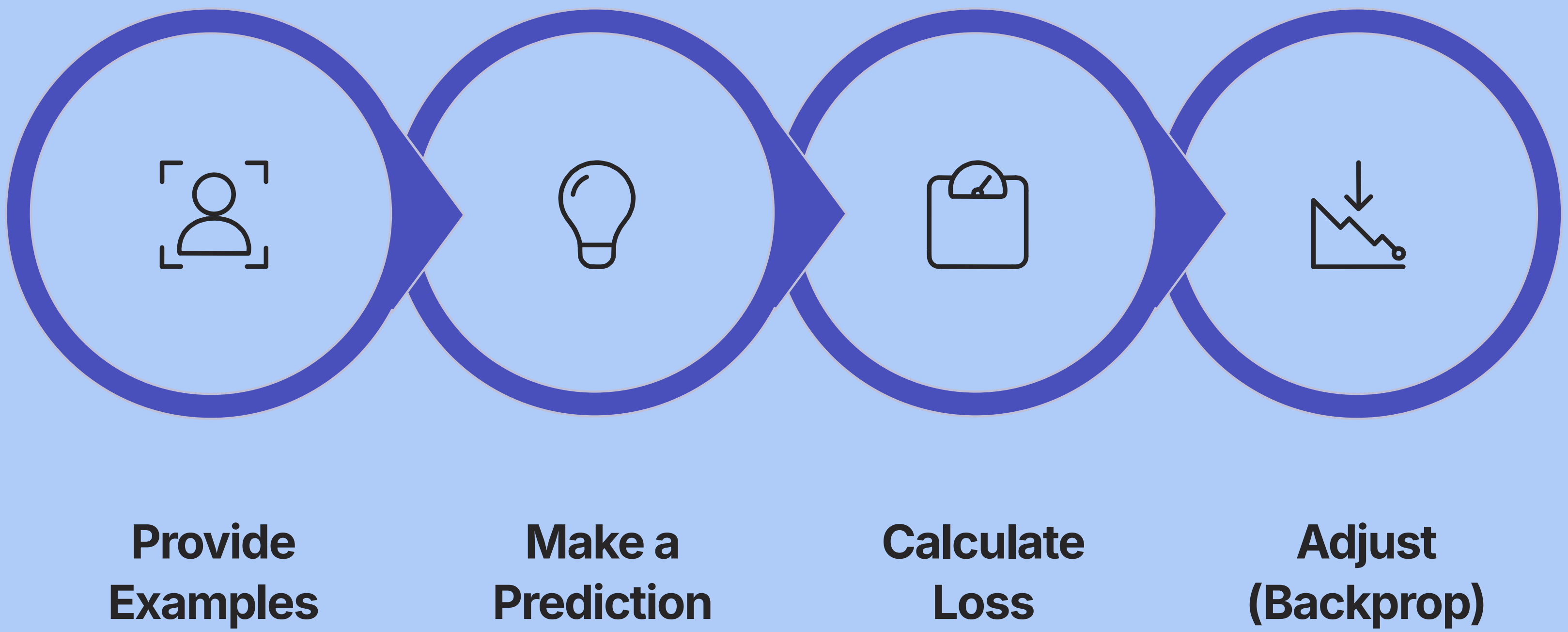
Propagate activations through network

Output Layer

Generate final predictions or decisions

How Does the *Network* Learn? (Training)

- ***The* Goal of Training:** To find the perfect set of "weights" for all the connections so that the network's predictions are as accurate as possible.
- **The Process:**
  a. **Provide Examples:** We show the network thousands of examples from the training data (e.g., a patient's gene data and their correct diagnosis).
  b. **Make a Prediction:** The network makes a guess, or a prediction.
  c. **Calculate the Error (Loss):** We compare the network's guess to the correct answer and measure how wrong it was. This is called the "loss."
  d. **Adjust and Repeat:** The network then works backward (a process called **backpropagation**) to slightly adjust all of its weights to reduce the error.



- This process is repeated thousands of times, and with each cycle, the network gets better and better at its task.

Training Strategy:

- **Optimizer:** We used the **Adam optimizer**, an efficient and popular algorithm for adjusting the network's weights during training.
- **Loss Function:** We used **Binary Cross-Entropy**, the standard mathematical formula for measuring error in a two-class classification problem.
- **Smart Training:** We used **Early Stopping**. This technique monitors the model's performance and stops the training process automatically when it's no longer improving, which prevents the model from "memorizing" the training data (overfitting).

Why Was an ANN the Right Tool for This Project?

- **Handles Complexity:** Gene expression data isn't simple. The way genes interact is highly complex and non-linear. ANNs are specifically designed to learn these kinds of intricate relationships that simpler models might miss.
- **Automatic Feature Learning:** The hidden layers automatically learn the most relevant "features" or combinations of genes. We don't need to tell the model to look for "high CD33 and low Zyxin"; it discovers these predictive patterns on its own.
- **Proven Power:** Deep learning has become the state-of-the-art in many areas of bioinformatics and medical image analysis for its ability to extract meaningful signals from high-dimensional data.

Thank You!

For your attention and engagement during our presentation.

We hope you gained valuable insights into the fascinating world of ANNs and their potential in medical diagnostics.