

# Deep Learning & Neural Networks

Dr. Shahira Ahmed

DS710

## Lecture 6



# Deep Learning & Neural Networks DS710

**Lecture 6**  
**Main Types of Neural Networks and Their  
Applications**

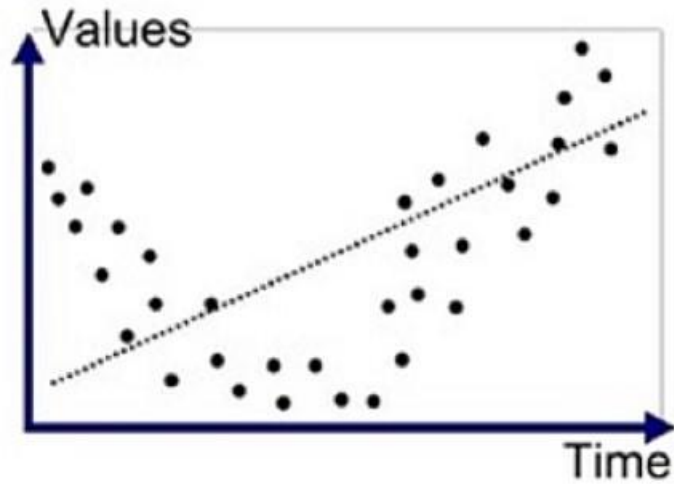
**Dr. Shahira Ahmed**



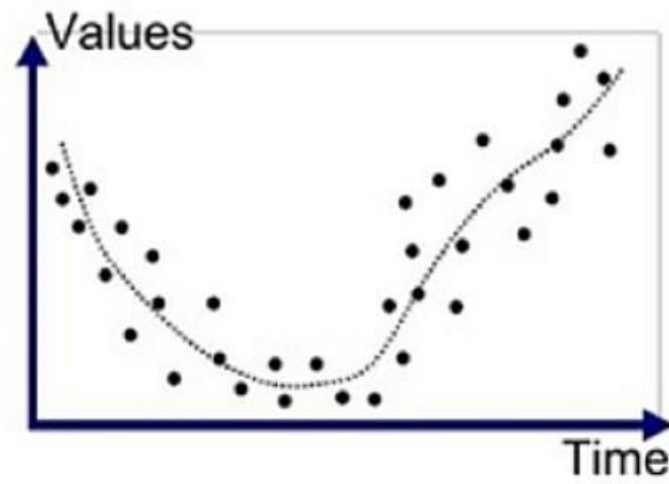
## LECTURE OUTLINE

- ◉ Main Types of Neural Network
- ◉ Neural Network Topology
- ◉ Main type of Layers of Deep Neural Network
- ◉ Lab: Live Demo Your First Neural Network

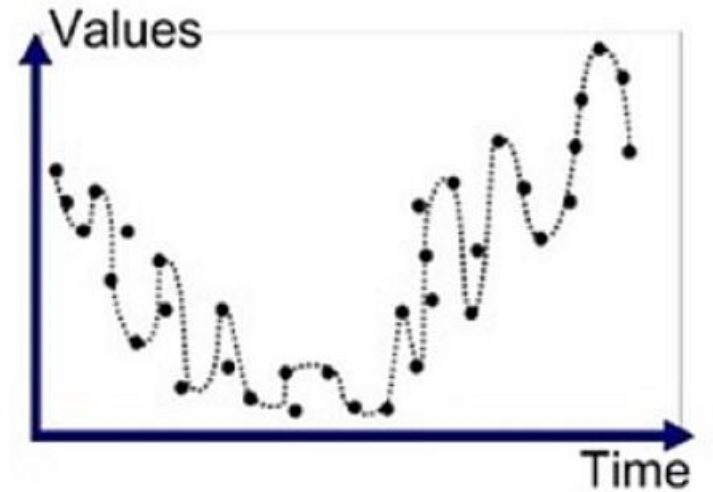
# Overfitting versus underfitting



Underfitted

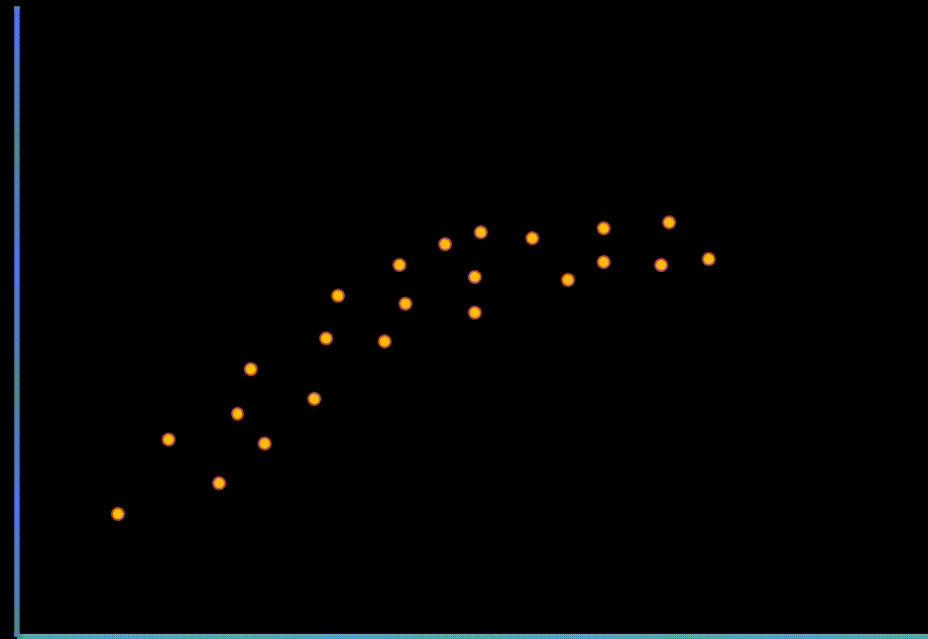


Good Fit/Robust



Overfitted

## Overfitting and Underfitting



© machinelearningknowledge.ai

<https://machinelearningknowledge.ai/overfitting-and-underfitting-in-machine-learning-animated-guide-for-beginners/>

- Overfitting is an **undesirable** machine learning behavior that occurs when the machine learning model gives **accurate predictions** for **training data** but **not for new data**.
- An overfit model can give **inaccurate predictions** and cannot perform well for all types of **new data**.

# How can you prevent overfitting?

- **Early stopping** **pauses** the training phase before the machine learning model learns the noise in the data. However, getting the timing right is important; else the model will still not give accurate results.
- **Pruning:** You might identify several features or parameters that impact the final prediction when you build a model. **Feature selection—or pruning—**identifies the most important features within the training set and eliminates irrelevant ones. For example, to predict if an image is an animal or human, you can look at various input parameters like face shape, ear position, body structure, etc. You may prioritize face shape and ignore the shape of the eyes.

# How can you prevent overfitting?

- **Regularization**

- a collection of **training/optimization** techniques that seek to **reduce overfitting**
- These methods try to eliminate those factors that do not impact the prediction outcomes by grading features based on importance.
- For example, mathematical calculations apply a penalty value to features with minimal impact. Consider a statistical model attempting to predict the housing prices of a city in 20 years. Regularization would give a lower penalty value to features like population growth and average annual income but a higher penalty value to the average annual temperature of the city.

# How can you prevent overfitting?

- Ensembling
  - Ensembling combines predictions from several separate machine learning algorithms. Some models are called weak learners because their results are often inaccurate.
  - Ensemble methods combine all the weak learners to get more accurate results.
  - Bagging and boosting are examples of ensemble models

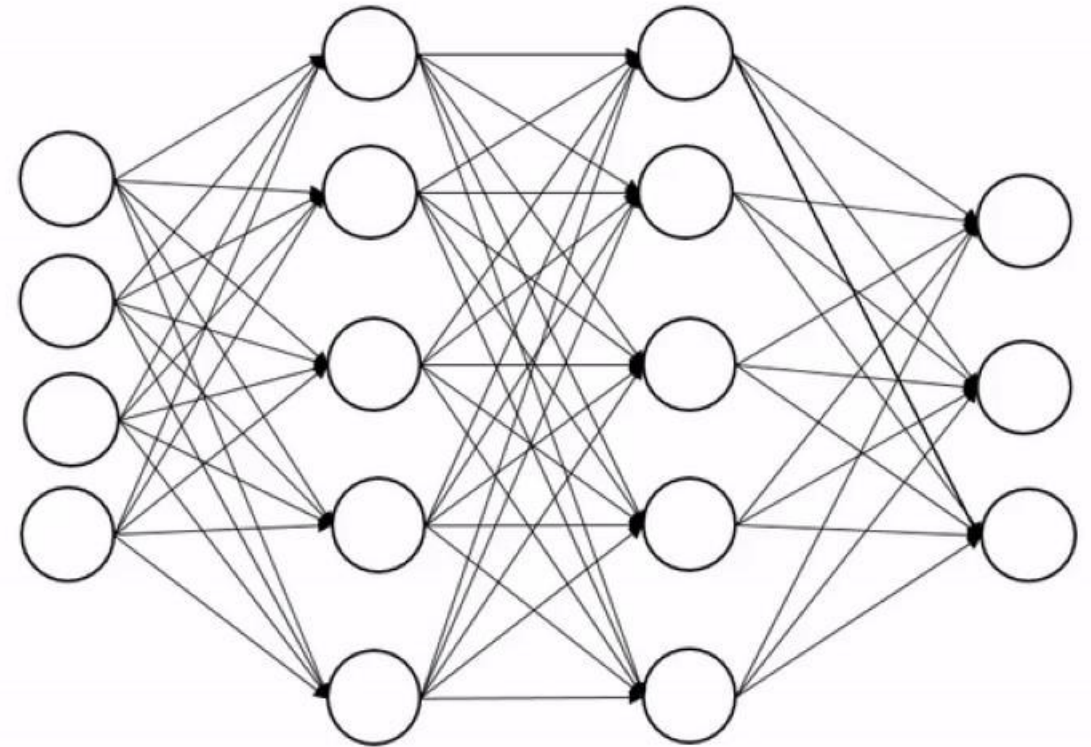
# How can you prevent overfitting?

- **Data augmentation**
  - a machine learning technique that changes the sample data slightly every time the model processes it.
  - You can do this by changing the **input data in small ways**. When done in moderation, data augmentation makes the training sets appear unique to the model and prevents the model from learning their characteristics.
  - For example, applying transformations such as translation, flipping, and rotation to input images.

# How can you prevent overfitting?

- **Dropout** is a regularization technique used in neural networks to prevent overfitting. Overfitting occurs when a neural network becomes too specialized to the training data and performs poorly on unseen data. Dropout helps address this issue by randomly setting a fraction of the input units (neurons) to 0 at each update during training.

<https://www.analyticsvidhya.com/blog/2022/08/dropout-regularization-in-deep-learning/>



# How can you prevent overfitting?

- **Dropout** is a powerful technique for reducing overfitting and improving the generalization performance of neural networks.
- However, it's important to find an appropriate dropout rate, as using too high a dropout rate may result in underfitting, while using too low a dropout rate may not provide sufficient regularization.
- The optimal dropout rate may vary depending on the specific problem and dataset, so it's often necessary to experiment and tune the dropout rate to achieve the best results.

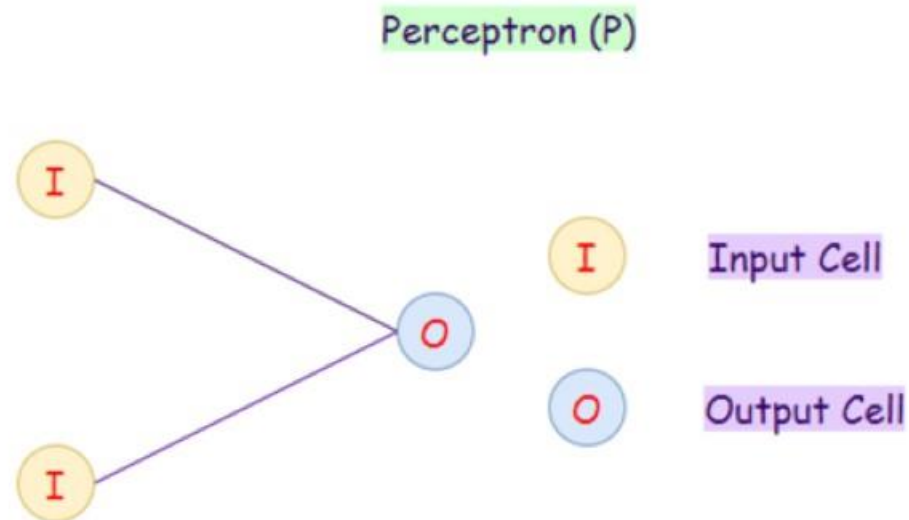
# What is underfitting?

- Underfitting is another type of error that occurs when the model **cannot** determine a meaningful **relationship between the input and output data**.
- You get underfit models if they have not trained for the appropriate length of time on a large number of data points.

- Topology of a neural network refers to the way the **Neurons are connected**, and it is an important factor in network functioning and learning.

- **Perceptron**
- The perceptron model is also known as a single-layer neural network. This neural net contains only two layers:

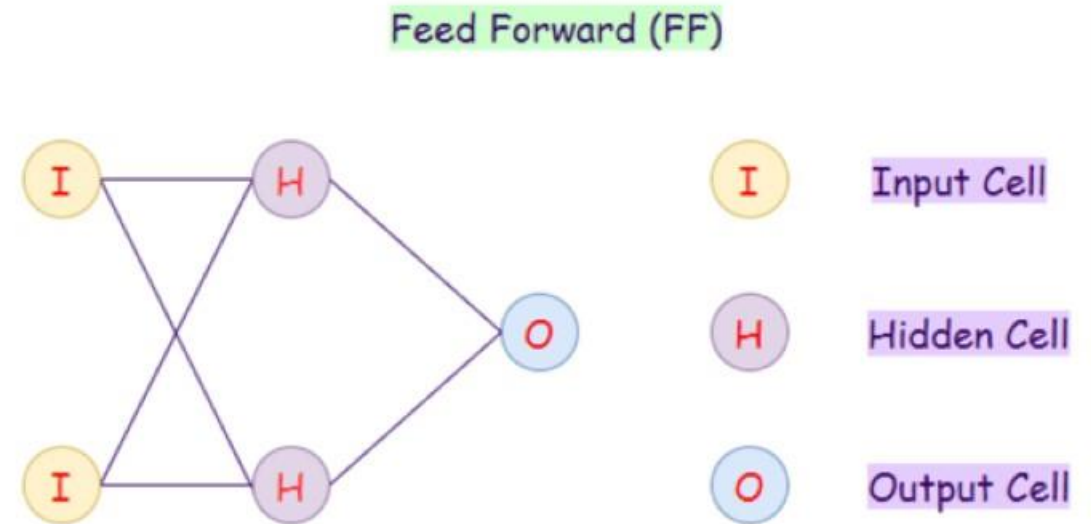
- Input Layer
- Output Layer



Applications:  
Classification

<https://towardsai.net/p/machine-learning/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e>

- **Feed-forward Neural Network**
- There are no back-loops in the feed-forward network. Hence, to minimize the error in prediction, we generally use the backpropagation algorithm to update the weight values.

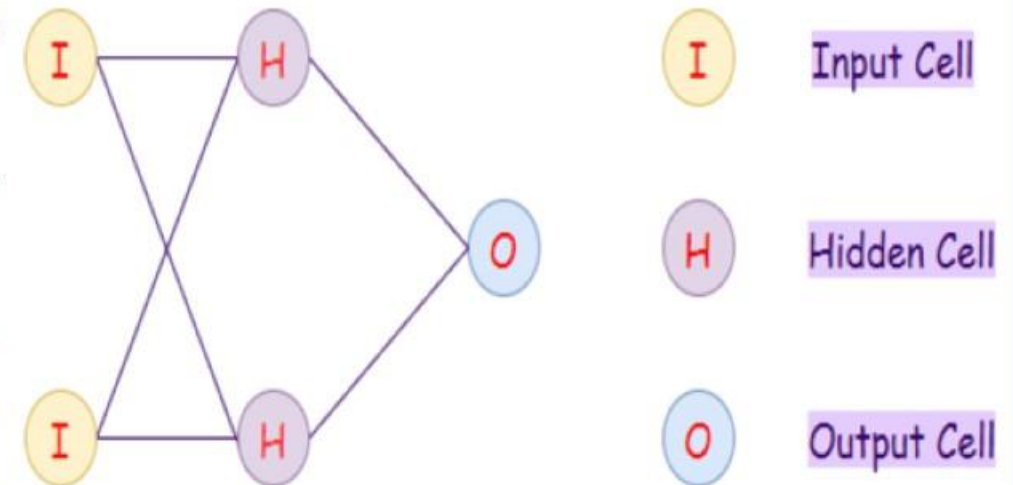


# Neural Network Topologies

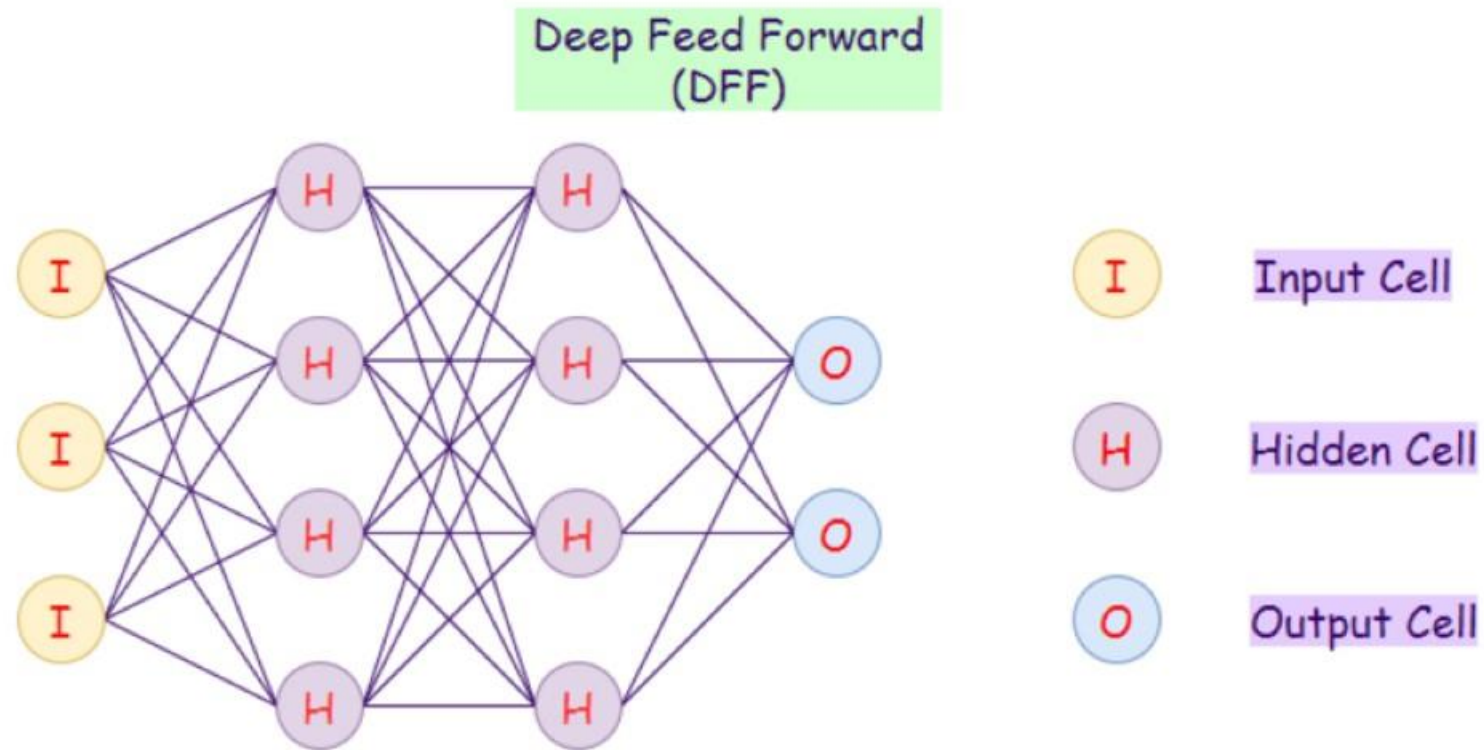
- **Feed-forward Neural Network**

- A feed-forward neural network is an artificial neural network in which the nodes **do not ever form a cycle**.
- In this neural network, all of the perceptrons are arranged in layers where the input layer takes in input, and the output layer generates output.
- The **hidden** layers have no connection with the outer world; that's why they are called hidden layers.
- In a feed-forward neural network, every perceptron in one layer is connected with each node in the next layer. Therefore, all the nodes are **fully connected**. There is no connection between the nodes in the same layer.
- Need as many neurons as coefficients of my feature vector.

Feed Forward (FF)



## Deep Feed-forward (DFF):



- **Deep Feed-forward (DFF):**
- A deep feed-forward network is a feed-forward network that uses **more than one hidden layer**.
- The main problem with using only one hidden layer is the one of overfitting, therefore by adding more hidden layers, we may **reduced overfitting and improved generalization**.

- **Recurrent Neural Network (RNN):**
- Recurrent neural networks (RNNs) are a variation to feed-forward (FF) networks. In this type, each of the neurons in hidden layers receives an input with a specific delay in time(cycle).



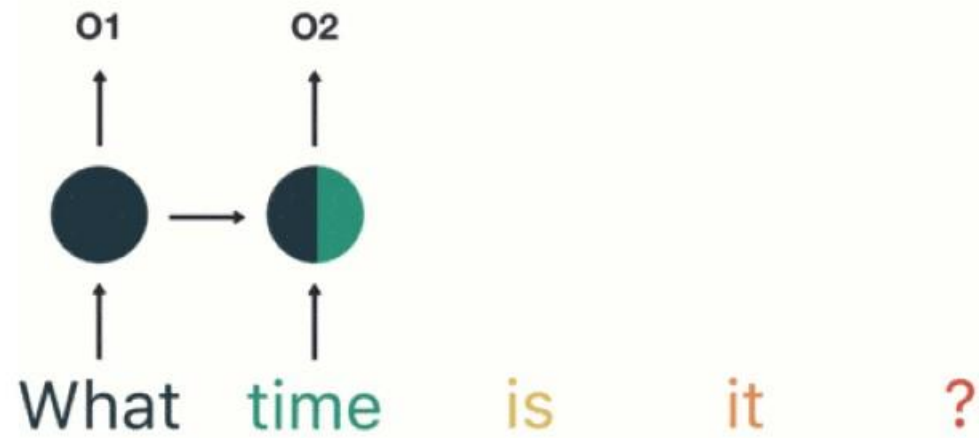
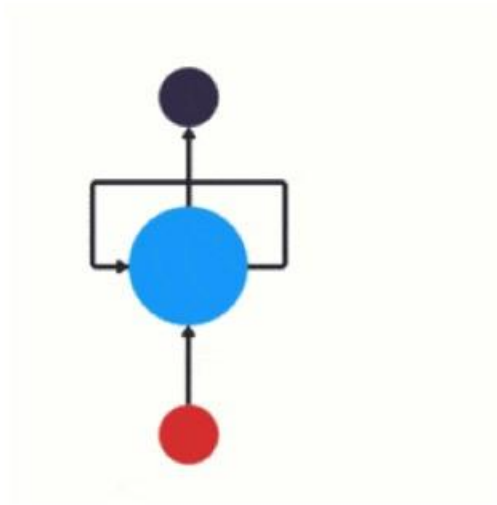
# Recurrent Neural Network (RNN):

- The input layer in architecture for our DNN is fixed. In a feed-forward neural network, the number of neurons in the input layer is determined by the number of features in the input data.
- Different input size. Add Zeros to make the input size equal.

# Recurrent Neural Network (RNN):

- Number of parameters becomes astronomically high. Suppose the maximum length of the sentence is 20 which pretty small for most data sets available. Let the number of words in corpus= 20k (considering a very small corpus)
- Then each input will become 400k dimensional and with just 10 neurons in the hidden layer, our number of parameters becomes 4 million! To have a proper network means having billions of parameters. To overcome this we need to have a network with weight sharing capabilities

# Recurrent Neural Network (RNN):



- **Recurrent Neural Network (RNN):**
- We use this type of neural network where we need to access previous information in current iterations. For example, when we are trying to predict the next word in a sentence, we need to know the previously used words first.
- RNNs can process inputs and share any lengths and weights across time. The model size does not increase with the size of the input, and the computations in this model take into account the historical information.
- However, the problem with this neural network is the slow computational speed. It cannot remember info from a long time ago.

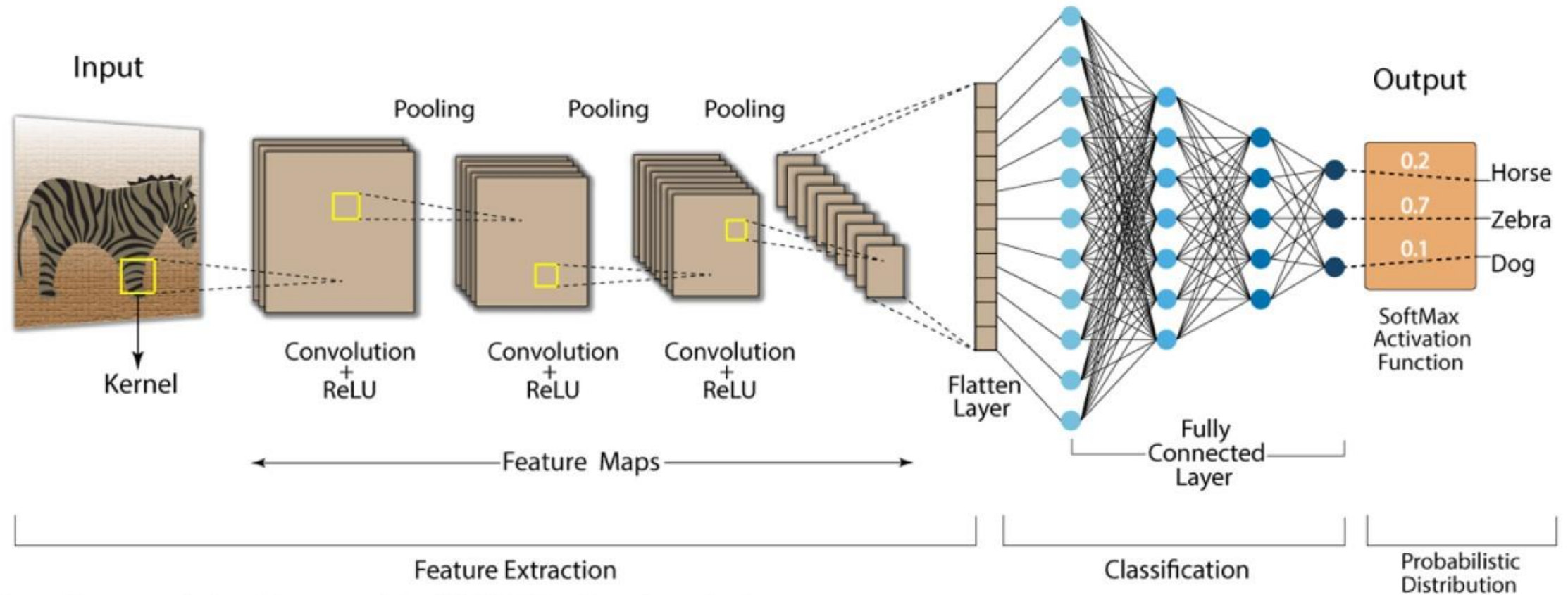
- RNNs are a type of neural network that are designed to process **sequential data, such as text, audio, or time series data**. They can “remember” or store information from previous inputs, which allows them to use context and dependencies between time steps.
- It can use **variable-length** input

# Recurrent Neural Network (RNN):

- LSTMs are a type of RNN that use special type of memory cell and gates to store and output information. LSTMs are effective at storing and accessing long-term dependencies. They are slower to train and run than other types of RNNs.
- GRUs are simplified version of LSTMs that use single “update gate” to control the flow of information into the memory cell. GRUs are easier to train and faster to run than LSTMs, but they may not be as effective at storing and accessing long-term dependencies.

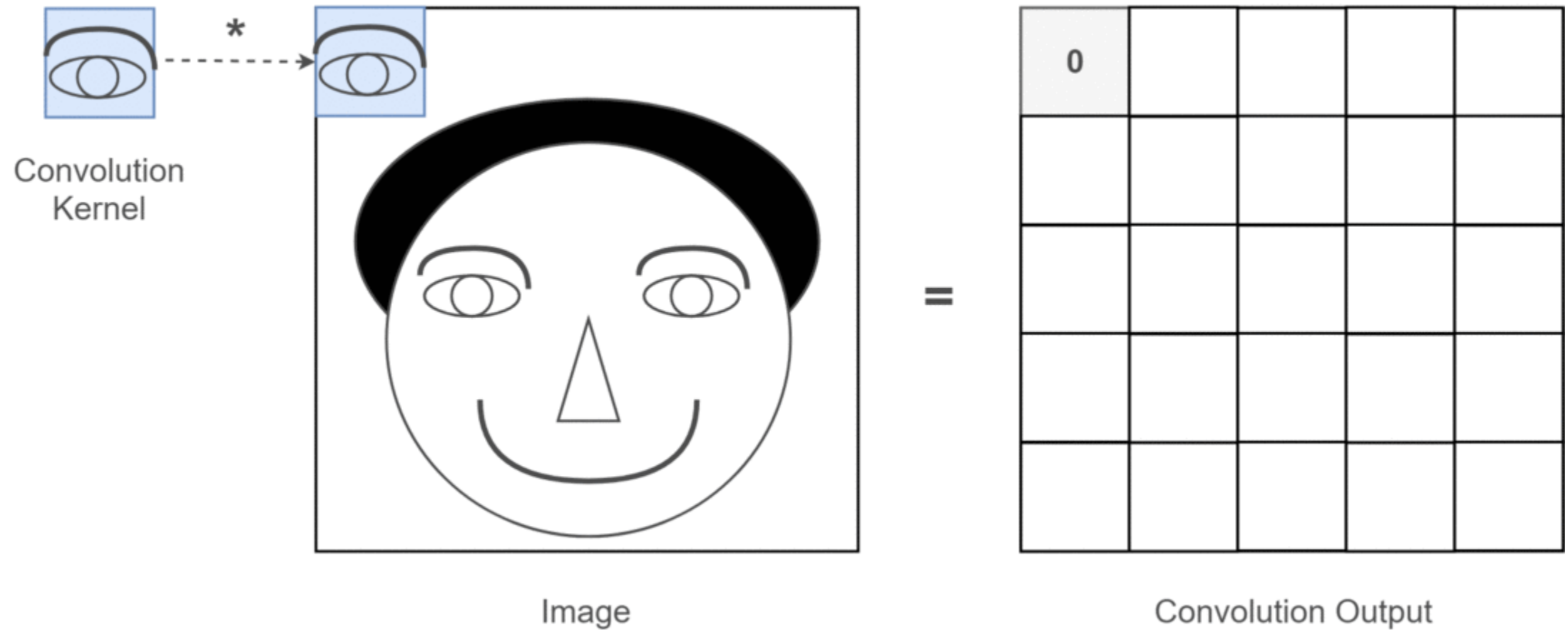
# Convolutional Neural Network (CNN)

## Convolution Neural Network (CNN)



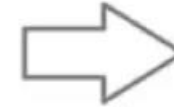
# Convolutional Neural Network (CNN)





- Why ConvNets over Feed-Forward Neural Nets?\_

|   |   |   |
|---|---|---|
| 1 | 1 | 0 |
| 4 | 2 | 1 |
| 0 | 2 | 1 |



|   |
|---|
| 1 |
| 1 |
| 0 |
| 4 |
| 2 |
| 1 |
| 0 |
| 2 |
| 1 |

*Flattening of a 3×3 image matrix into a 9×1 vector*

<https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>

- An image is nothing but a matrix of pixel values, right? So why not just **flatten** the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? Uh.. not really.
- In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to **no accuracy** when it comes to complex images having pixel dependencies throughout.
- A ConvNet is able to successfully capture the **Spatial and Temporal** dependencies in an image

- eNet-5
- AlexNet
- VGGNet
- GoogLeNet
- ResNet
- DenseNet
- MobileNet
- SqueezeNet
- NASNet
- Xception
- ShuffleNet
- EfficientNet
- Inception-ResNet
- MobileNetV2
- MobileNetV3
- ResNeXt
- DarkNet
- YOLO  
(You Only Look Once)
- SSD (Single Shot MultiBox Detector)
- CapsuleNet
- SENet (Squeeze-and-Excitation Network)
- PNASNet (Progressive Neural Architecture Search Network)
- HRNet (High-Resolution Network)
- EfficientDet
- RegNet (Designing Network Architectures for Resource-Constrained Training)

# Other Types of DNN

- Feed-forward Neural Networks (FNN)
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM) Networks
- Gated Recurrent Unit (GRU) Networks
- Autoencoders
- Generative Adversarial Networks (GAN)
- Transformers
- Deep Belief Networks (DBN)
- Radial Basis Function Networks (RBFN)
- Deep Q-Networks (DQN)
- Restricted Boltzmann Machines (RBM)
- Hopfield Networks
- Boltzmann Machines
- Self-Organizing Maps (SOM)
- Deep Residual Networks (ResNet)
- Capsule Networks
- Echo State Networks (ESN)
- Deep Reinforcement Learning Networks
- Modular Neural Networks
- Attention-based Neural Networks
- Hybrid Neural Networks
- Spiking Neural Networks (SNN)
- Neural Turing Machines (NTM)
- Deep Variational Autoencoders (VAE)

- TensorFlow is an free open source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system
- TensorFlow was developed by the Google Brain team for internal Google use in research and production.
- TensorFlow is not limited to Python only. While Python is the most commonly used language for TensorFlow, TensorFlow also provides support for other programming languages such as C++, Java, JavaScript, Swift, and Go.
- tensor : A multidimensional array

- Keras is an open source framework a high-level neural networks API written in Python. It is capable of running on top of popular deep learning frameworks such as TensorFlow, Microsoft Cognitive Toolkit (CNTK), or Theano. Keras provides a user-friendly interface for designing, training, and evaluating deep learning models.

- Load

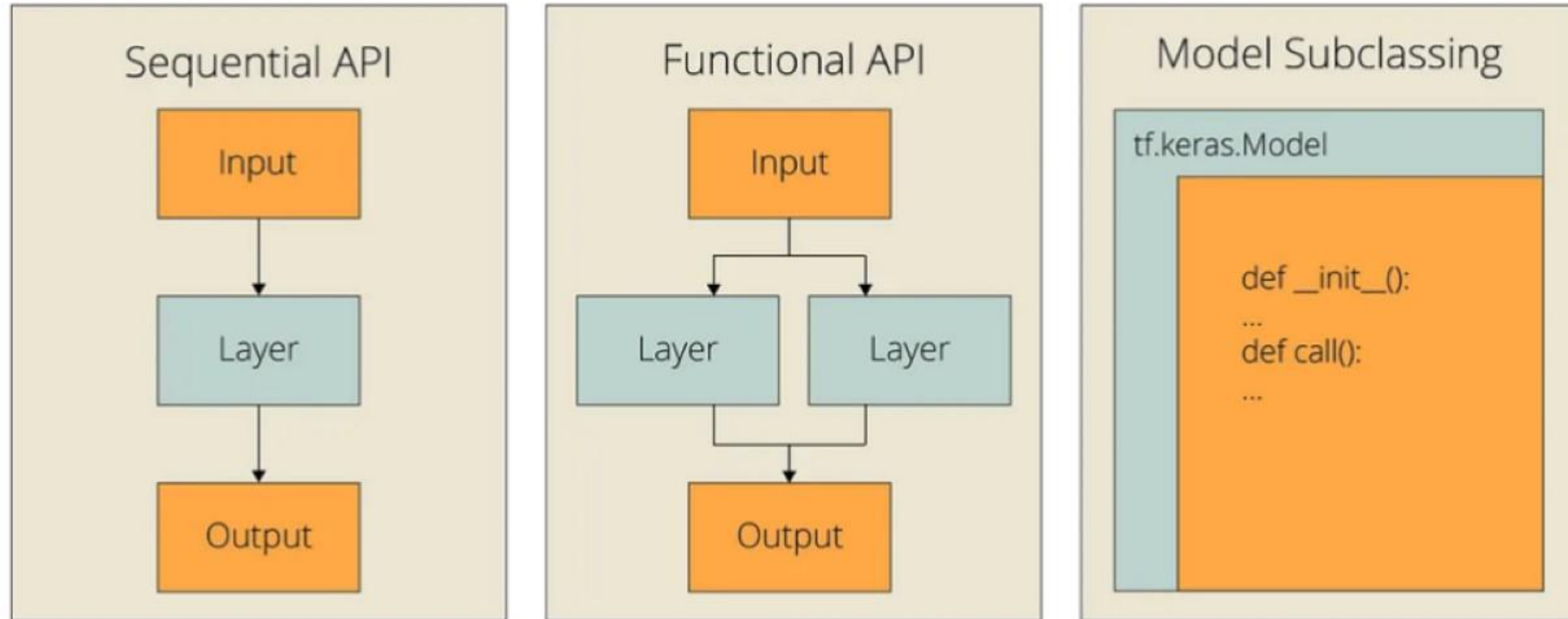


```
mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

# Build Your First Neural Network-Keras



- The Sequential API allows you to build models by stacking layers on top of each other.
- The Functional API in Keras allows you to build more complex models with multiple inputs or outputs or models with shared layers

- There are many types of layers available in the Keras Sequential API.

```
[ ] model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10)  
])
```

There are many types of layers available in the Keras Sequential API.

- One of the most common layer types is the **Dense layer**, a fully connected layer.
- Flatten layer: a layer that flattens the multi-dimensional input tensors into a single dimension — used as a transition layer between convolutional or recurrent layers and fully connected layers in a neural network.
- Dropout layer: a layer that randomly sets input units to 0 (using a defined frequency) during training time — used as a regularization technique to prevent overfitting in neural networks.

- **Compiling a Model with Keras**
- Once you've built a model using the Keras Sequential API, you'll need to compile it before it can be used for training. When compiling the model, you'll need to specify both a loss function and an optimizer.

```
[ ] model.compile(optimizer='adam',  
                  loss=loss_fn,  
                  metrics=['accuracy'])
```

- Fitting a Model with Keras
  - Finally, to use Keras for deep learning, the compiled model must be fit to a training dataset.
    - `model.fit(X_train, y_train)`
  - To evaluate the performance of the model after training, you can use evaluate:
    - `test_loss, test_accuracy = model.evaluate(x_test, y_test)`

# LAB

**Dr. Shahira Ahmed**

# A Short History of Programming languages

- 1) Nothing
- 2) machine code
- 3) assembly languages

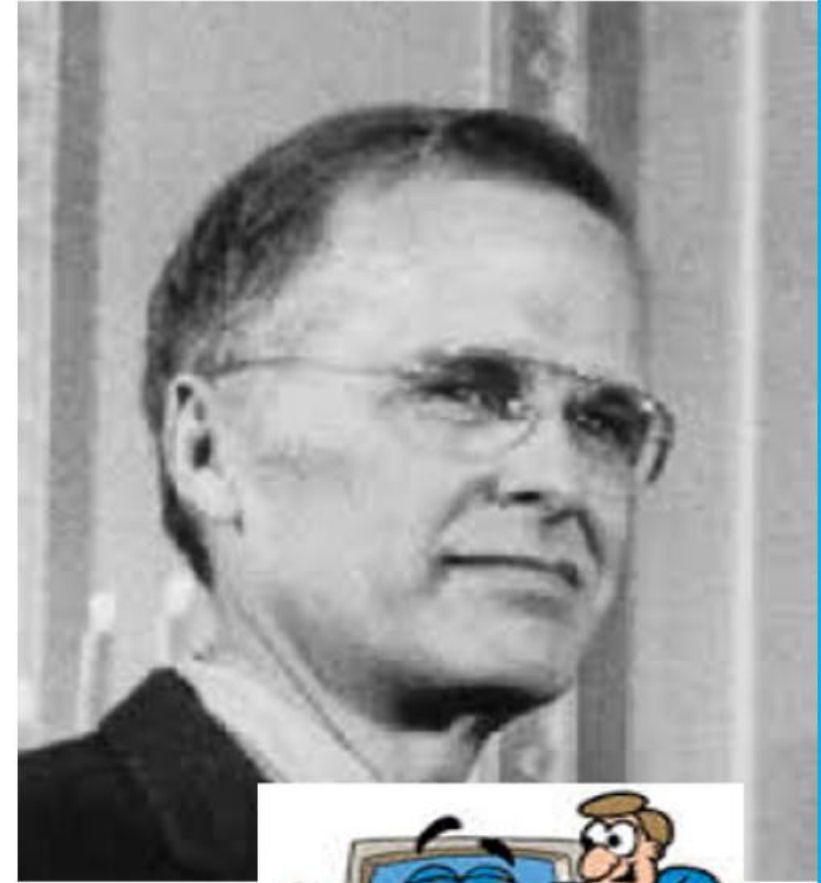
# A Short History of Programming languages

- 1954: IBM develops the 704 – Successor to the 701
- All programming done in assembly
- Problem – Software costs exceeded hardware costs!



# A Short History of Programming languages

- Fortran I
  - 1958 – >50% of all software is in FORTRAN
  - Performance is close to hand-written assembly!
  - Faster Development time.



<http://www.columbia.edu/cu/computinghistory/backus.html>

# A Short History of Programming languages

- Fortran I
  - 1958 – >50% of all software is in FORTRAN
  - Performance is close to hand-written assembly!
  - Faster Development time.
- C
- JAVA
- Python



- Python is a popular language for AI because
  - it is **easy** to learn
  - has a large and active **community**
  - offers powerful **libraries** for data analysis and visualization and has excellent machine-learning libraries.

# Why Python?

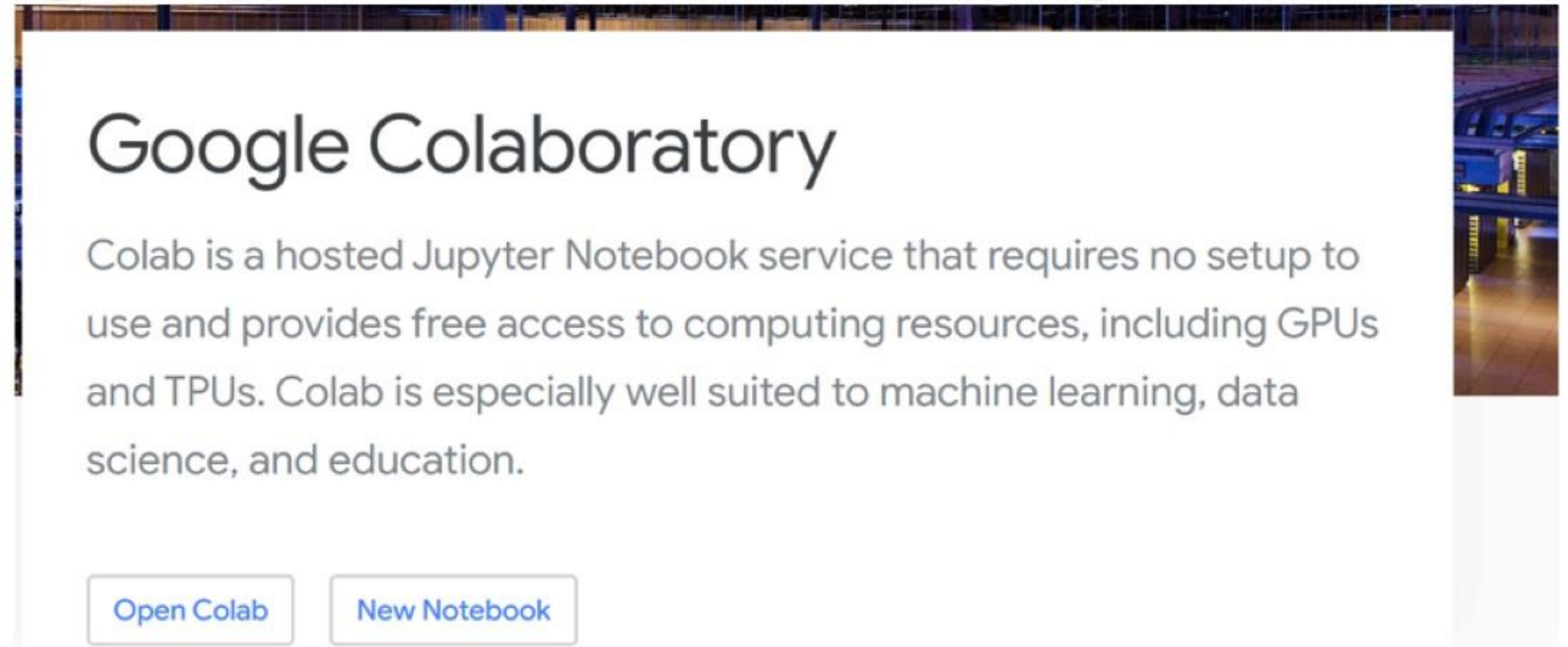
- Anaconda
- Colab

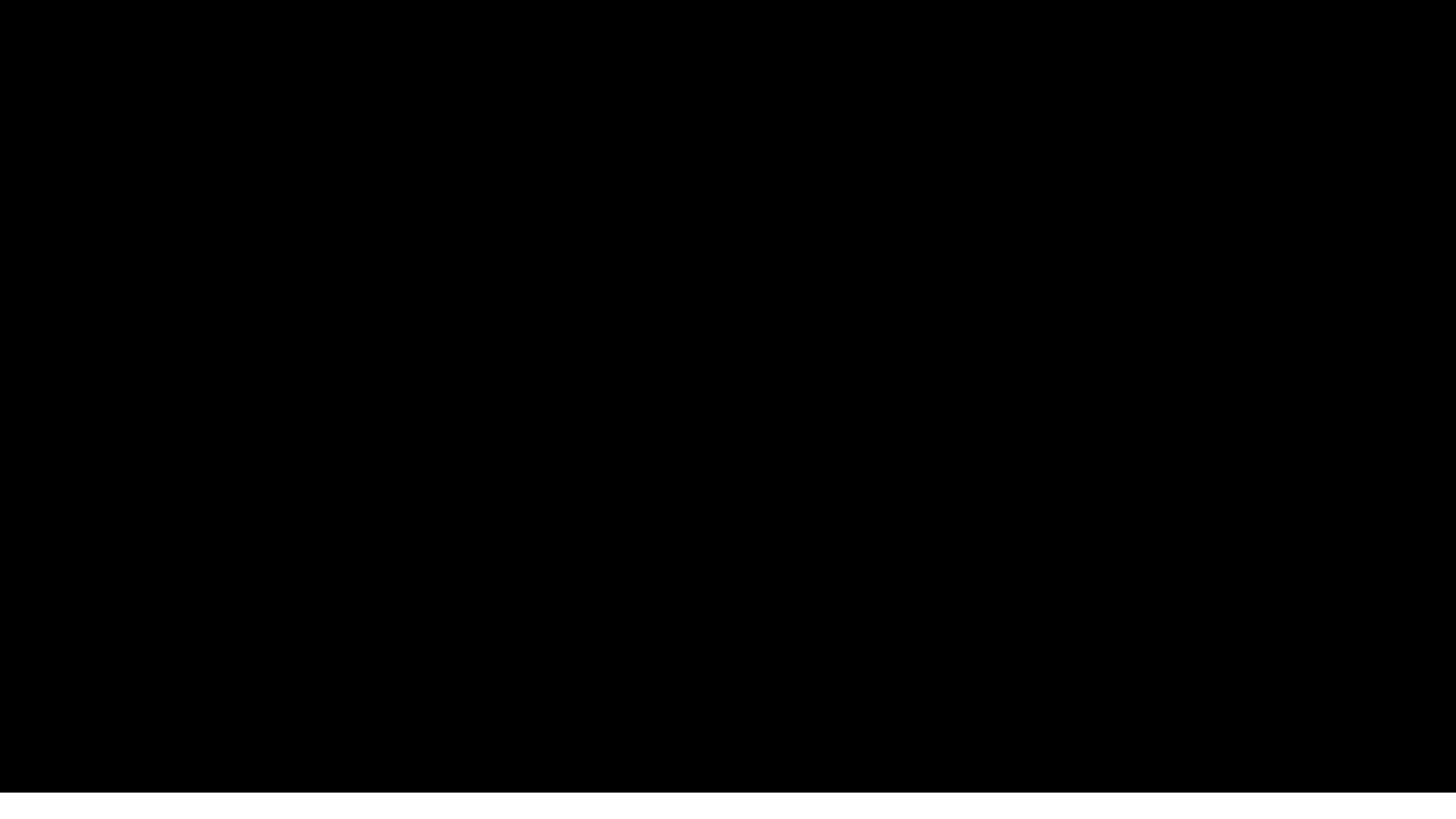
# What is Anaconda?

- Anaconda is actually a distribution of software that comes with conda, Python, and over 150 scientific packages and their dependencies.
- The application conda is a package and environment manager. Anaconda is a fairly large download (~500 MB) because it comes with the most common data science packages in Python.
- If you don't need all the packages or need to conserve bandwidth or storage space, there is also **Miniconda**, a smaller distribution that includes only conda and Python. You can still install any of the available packages with conda, it just doesn't come with them.

- Anaconda is available for Windows, Mac OS X, and Linux. You can find the installers and installation instructions at <https://www.anaconda.com/download/>

- Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.







**Thank You**

