INTRODUCING TENSORFLOW

Deep Learning Framework

TENSORFLOW

An end-to-end Open Source Machine Learning Platform which has a comprehensive, flexible ecosystem of tools, libraries as well as wide community resources that will let Researchers & developers to build state-of-the-art ML Models, and deploy ML Powered applications

ADVANTAGES OF TENSORFLOW

Computational Graph -AutoDiff

Library Management - Backed by Google

Debugging

Scalability

Create Pipelines

Tensorboard for Training Monitoring

INSTALLATION STEPS

```
# Requires the latest pip
!pip install --upgrade pip

# Current stable release for CPU and GPU
!pip install tensorflow

# Or try the preview build (unstable)
!pip install tf-nightly

#Importing Steps
import tensorflow as tf
print(tf.__version__) #Display version
```

BUILDING A SIMPLE NEURAL NETWORK USING TENSORFLOW 2.X

BUILD NEURAL NETWORK USING TENSORFLOW 2.X

- tf.keras is TensorFlow's Implementation of Keras API
- High Level API to build & train models - making tensorflow easier to use

```
import tensorflow as tf
from tensorflow import keras

#print tensorflow version & keras version
print(tf.__version__)
print(keras.__version__)
```

```
2.5.0
2.5.0
```

SPECIFY MODEL LAYERS

- Use Sequential class to create linear stack of layers

```
#Syntax
#Create the Model as Sequential
model = tf.keras.Sequential()

#To add layers - write : model.add()
model.add(...)
model.add(...)
model.add(...)
```

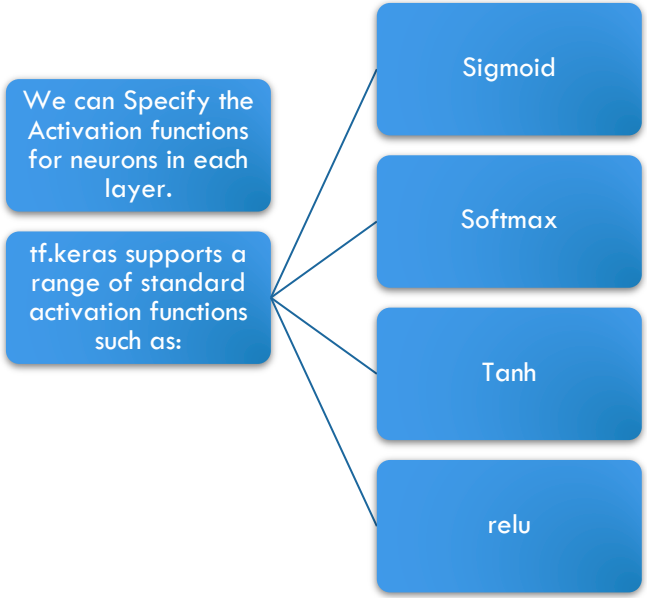
Example:

```
#Add a Hidden layer with 3 Neurons
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(3, input_shape = (784,)))
```

I have a model which takes input of shape – 784 (1d)

This has a Single hidden layer with 3 Neurons

SPECIFYING ACTIVATIONS

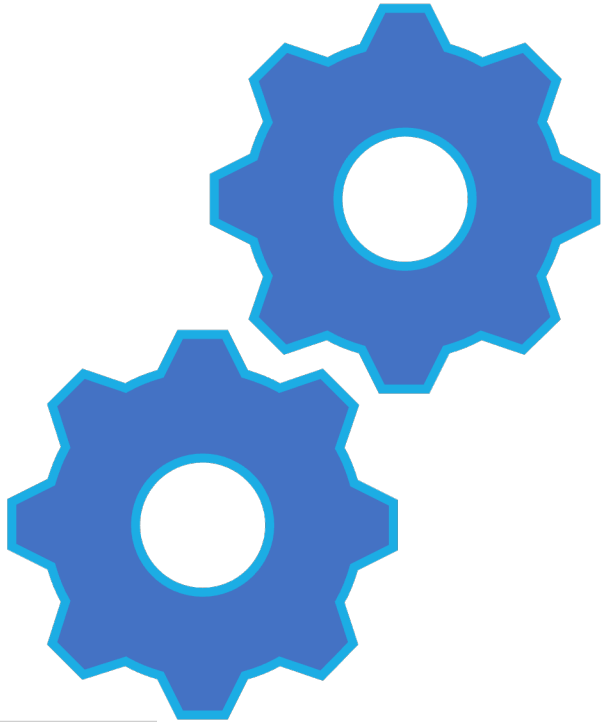



```
#Add a Hidden Layer with 3 Neurons - Sigmoid activation
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(3, input_shape = (784,),
                                activation = "sigmoid"))
```

MODEL COMPILATION

- Once the Structure of the model is defined, we will compile it using compile() function
- Main attributes are:
  - Model Optimizer (Learning Algorithm)
  - Loss Function
  - Metrics (for Evaluation)
- Syntax:

```
model.compile(optimizer=... , loss=..., metrics=...)
```



MODEL OPTIMIZER

- A Learning Algorithm/Search technique to update the weights & bias terms of Model
- Popular Optimizers are:
  - SGD : Stochastic Gradient Descent
  - RMSprop
  - Adam : Adaptive Moment Estimation

```
#Example
model.compile(optimizer="SGD" , loss=..., metrics=...)
```

MODEL LOSS FUNCTION

- An objective function used by learning algorithm/optimizer to update the weights & bias terms of Neurons during training
- Popular Model Loss Functions are:
  - mse : Mean Square Error
  - binary_crossentropy : for binary Logarithmic loss (logloss)
  - categorical_crossentropy : for multi-class logarithmic loss

```
#Example
model.compile(optimizer="SGD" ,
              loss="mse",
              metrics=["accuracy"])
```

MODEL TRAINING

- We train the Model using fit() function. Important Attributes are:
  - epochs – Number of passes of dataset the Model has to complete
  - batch_size – Number of training instances to be used by model for each update on Model Parameters
  - x,y → Training input data & Training Label data
  - validation_data → Validation data to be used during training (sent as tuple)

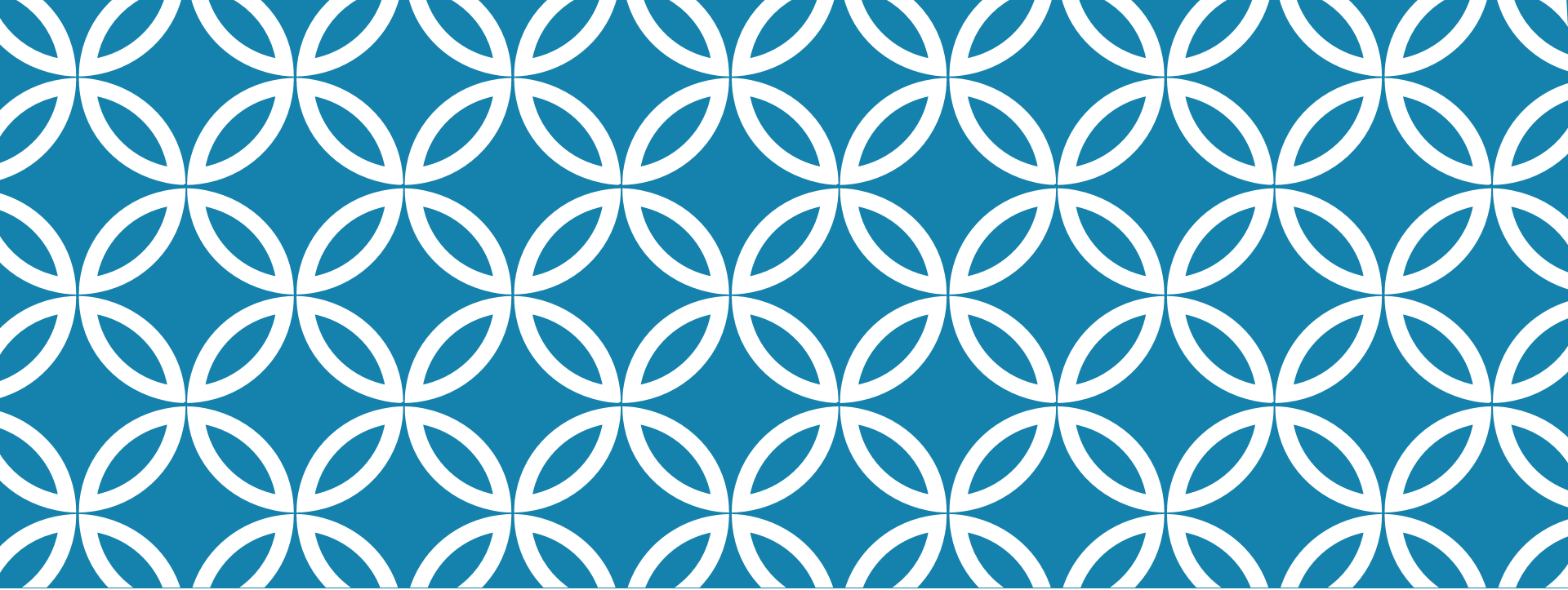
```
#Syntax
model.fit(x = X_train, y = y_train,
          batch_size = 32,
          epochs = 16,
         validation_data = (X_val,y_val))
```

CREATING OUR FIRST NEURAL NETWORK USING TF 2.X

Hands-on

SUMMARY

Data Preperation	Perform Split	Model Creation	Configuration of Model	Training
Prepare the Data	Split the data into two parts	Define the Model • Specify number of Neurons in each layer • Specify the individual layers • Specify the Activation in each layer	Compile the Model • Optimizer • Loss Function • Evaluation	Perform the Fit() • Specify Training & Test Data • Run Evaluation



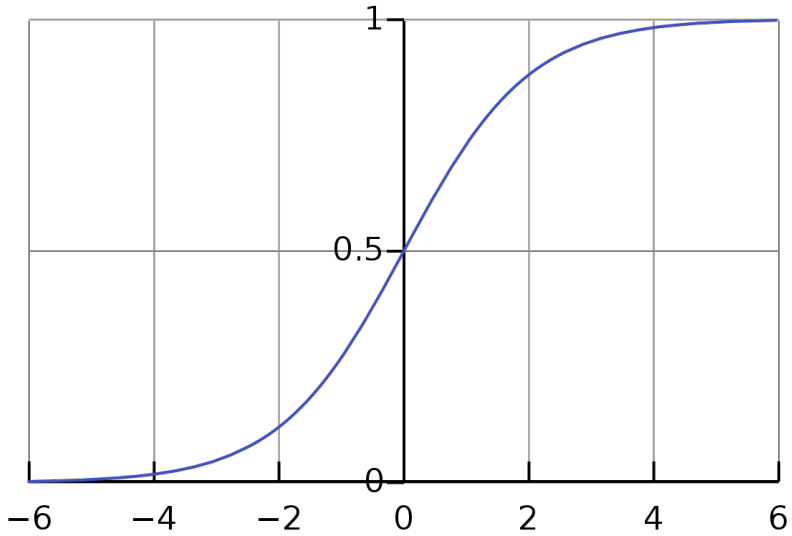
ACTIVATION FUNCTIONS

Deep Learning

ACTIVATION FUNCTION

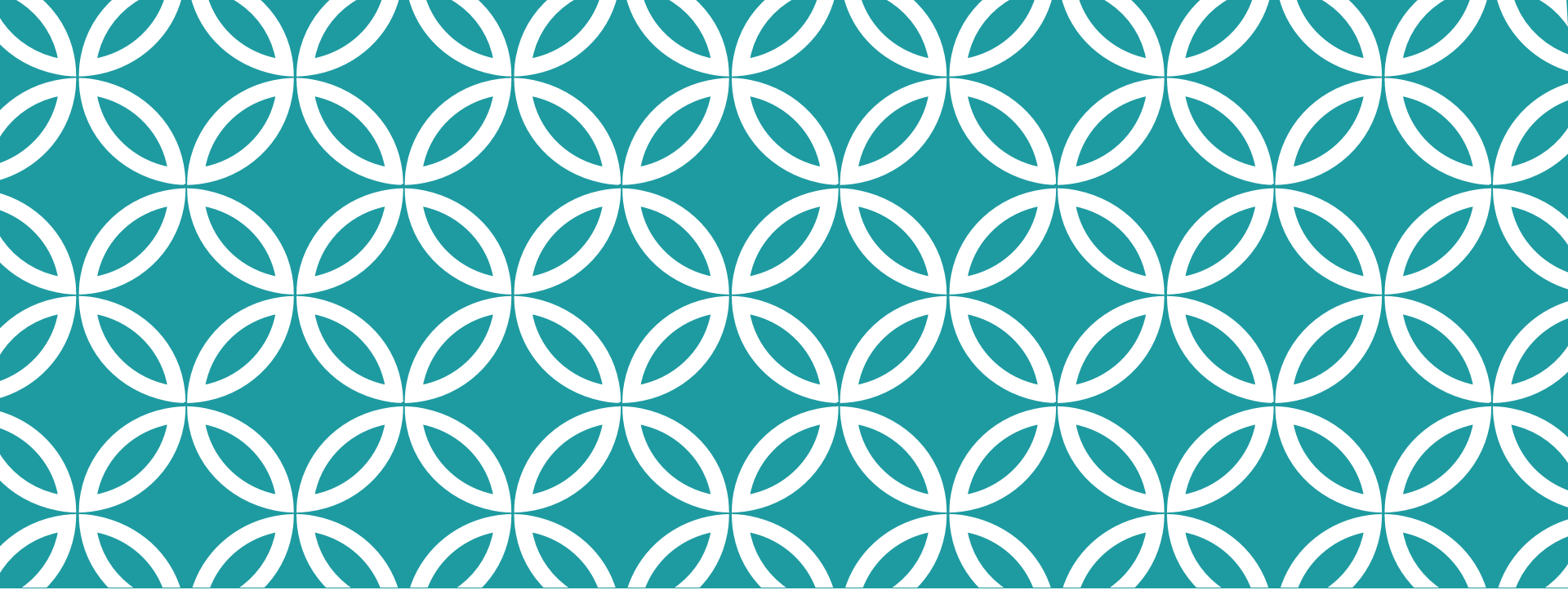
An Activation function in a neural network defines how the Weighted Sum of input is transformed into an output from a node.

Till now, we have been using the Activation function as : **Sigmoid**



OTHER ACTIVATION FUNCTIONS:

1. Tanh
2. Relu
3. Leaky Relu
4. Selu
5. Softmax
6. Others like: Swish, Parameterised ReLU



ACTIVATION FUNCTIONS

Deep Dive

SUMMARY

We have learnt various Activation functions which can be used on any hidden layer.

Activation Function : Softmax will be used at the Output layer when we want to generate the probability along each axis. - A Common requirement during Multi Class Classification

At the end it's a Mathematical function which will modify the Weighted Sum, and generate the Output

COMMON NETWORK ARCHITECTURE

Deep Learning for various tasks

BINARY CLASSIFICATION

Hyperparameter	Typical Values
# input Neurons	One per input feature
# hidden Layers	Depends on Problem (Typically 1 to 5)
# Neurons per hidden layer	Depends on Problem (Typically 10 to 100)
# output neurons	1
Hidden Activation	ReLU, Sigmoid, etc. (Typically - ReLu)
Output Activation	Logistic (Sigmoid)
Loss Function	Binary Cross Entropy

MULTI CLASS CLASSIFICATION

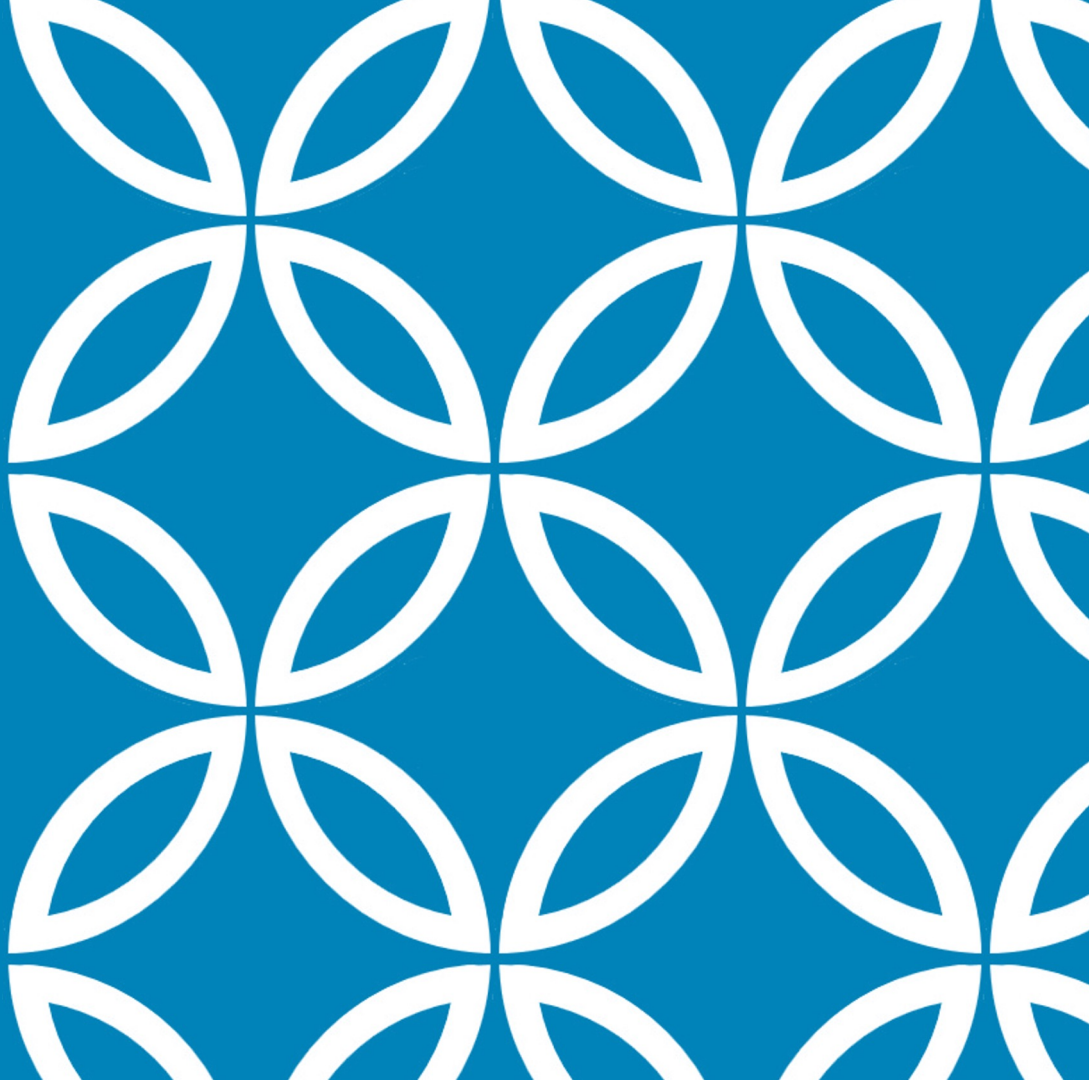
Hyperparameter	Typical Values
# input Neurons	One per input feature
# hidden Layers	Depends on Problem (Typically 1 to 5)
# Neurons per hidden layer	Depends on Problem (Typically 10 to 100)
# output neurons	1 per label
Hidden Activation	ReLU, Sigmoid, etc. (Typically - ReLu)
Output Activation	Softmax
Loss Function	Categorical Cross Entropy

REGRESSION

Hyperparameter	Typical Values
# input Neurons	One per input feature
# hidden Layers	Depends on Problem (Typically 1 to 5)
# Neurons per hidden layer	Depends on Problem (Typically 10 to 100)
# output neurons	1
Hidden Activation	ReLU
Output Activation	None, or Relu (Positive inputs) or Logistic/tanh(bounded outputs)
Loss Function	MSE or MAE

CROSS ENTROPY LOSS

Multi Class Classification

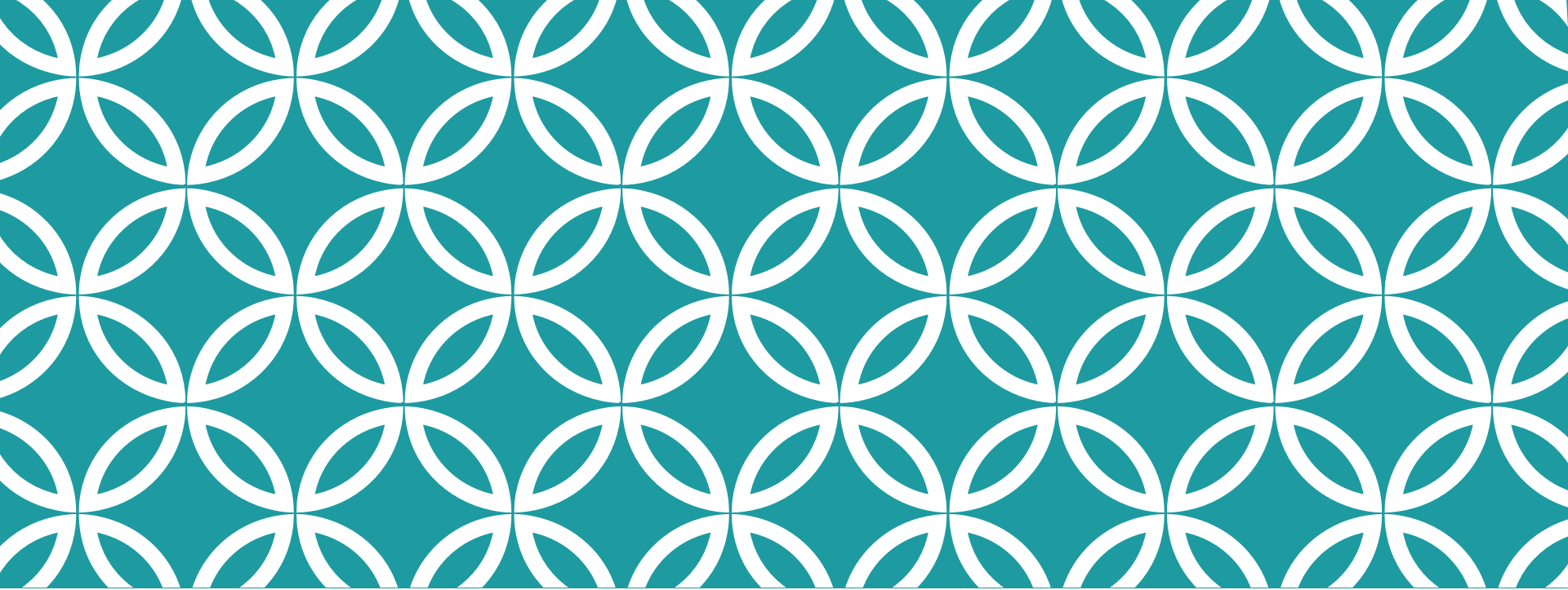


MOTIVATION

Why do we require a Cross Entropy Loss ?

Getting Familiar to Terminology

DEEP LEARNING NUTS & BOLTS



HANDS ON IMPLEMENTATION

Deep Learning Tasks

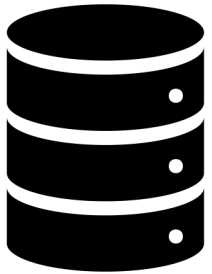
BATCH SIZE & EPOCHS

Intution

EPOCH

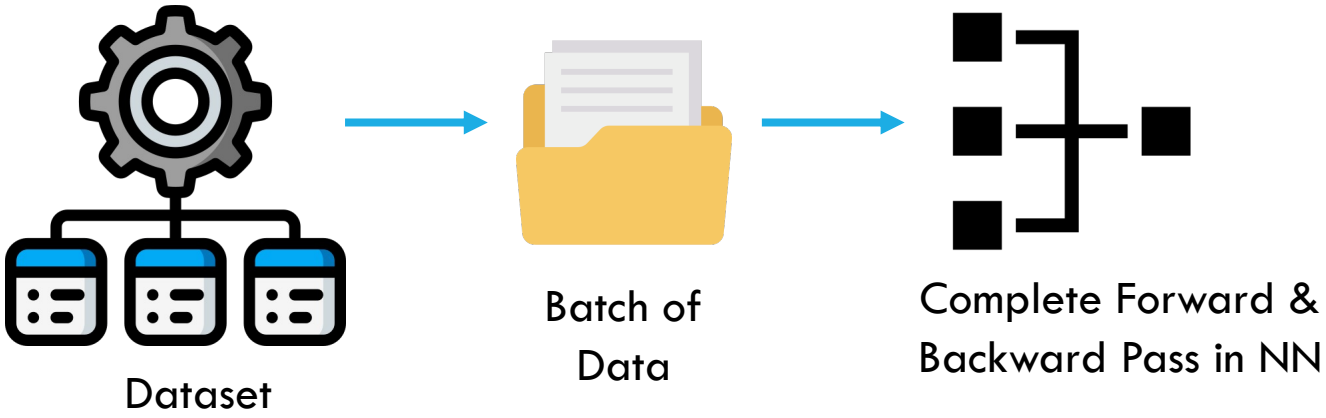
One Epoch is when the Entire Dataset has completed forward pass and backward pass through the neural network once.

Note:
By Increasing the number of epochs, we will enable our Neural Network to change the parameters a greater number of times, thus helping it to perform better



BATCH SIZE

It's the number of training instances used by the model to complete one forward and one backward pass.



ITERATIONS

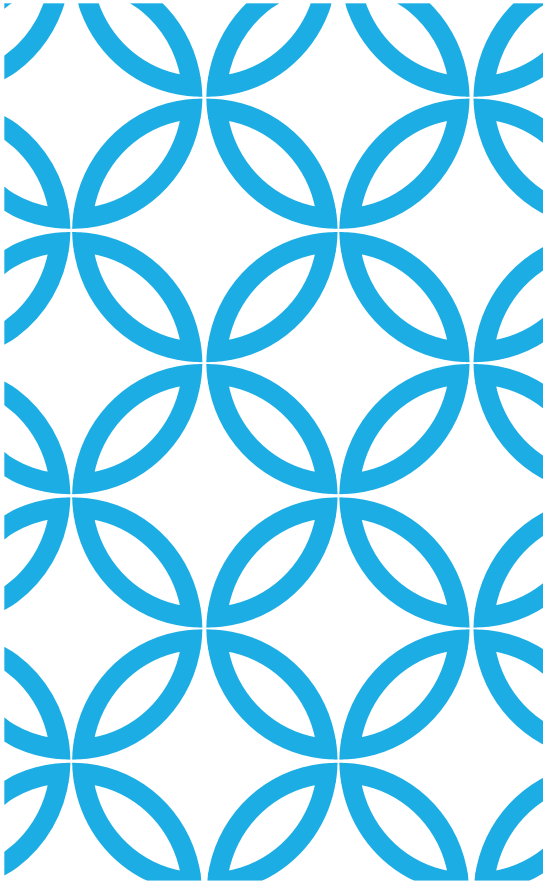
Iteration represents the number of passes that has been performed on a network

One Pass → One Forward + One Backward Pass

EXAMPLE

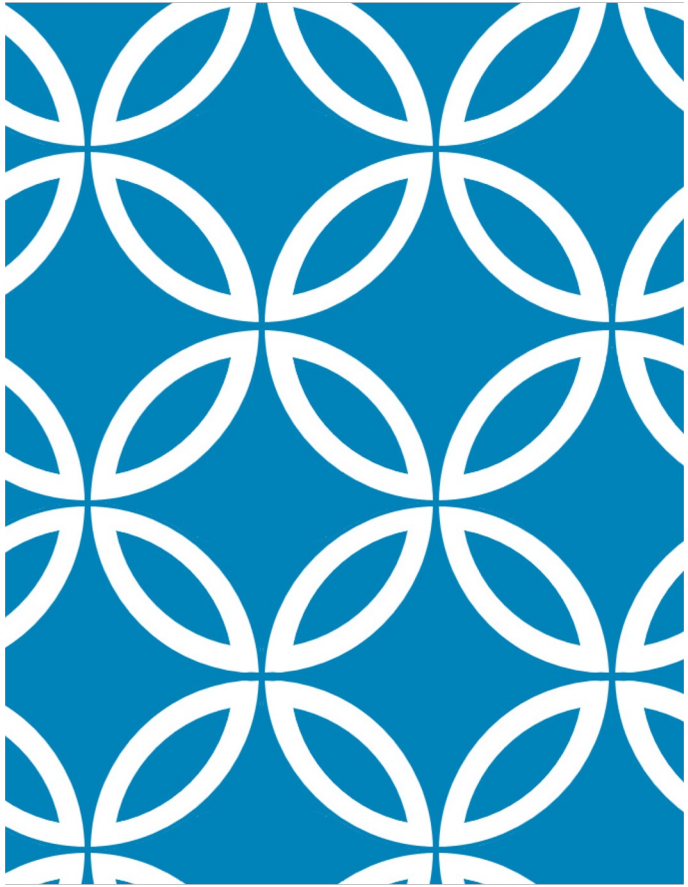
Let's say we have a dataset with 100 instances, and we have the batch size of 5, it is running for 3 epochs :

- For each epoch, we will have 20 batches (100/5 = 20)
- Each Batch will pass through the algorithm, so there will be 20 iterations per epoch
- Epochs = 3, meaning total number of iterations for training is 60 (20*3)



HANDWRITTEN DIGIT CLASSIFICATION USING TF 2.X

Hands-on



HYPERPARAMETER TUNING – DEEP LEARNING – PART 1

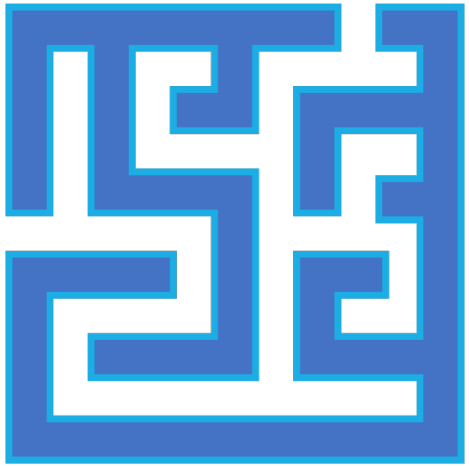
Improving the Performance of ANN

WHAT ARE HYPERPARAMETERS?

Hyperparameters are variables that we need to set before applying a learning algorithm to a dataset.

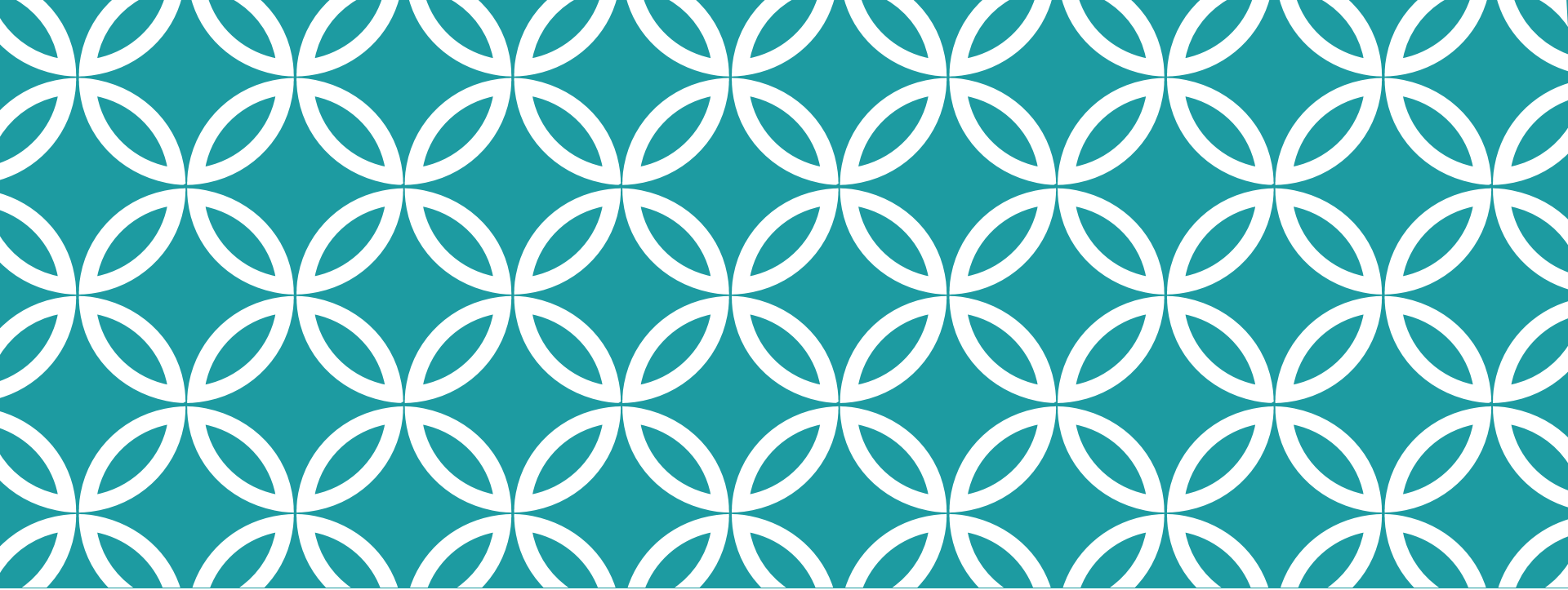
Challenge:

No Specific Magic number for every task. The best numbers depend on each task and each dataset.



HYPERPARAMETERS IN DEEP LEARNING NEURAL NETWORKS

1. Optimizer Hyper parameters
   a. Learning Rate
   b. Batch Size
   c. Number of Epochs
2. Model Specific Hyper parameters
   a. Number of hidden units
   b. Number of Hidden layers
   c. Activation Function



HYPER PARAMETERS

Optimizer

LEARNING RATE

```
#Example
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01) ,
              loss="mse",
              metrics=["accuracy"])
```

Learning rate refers to rate at which updates will take place on model parameters during training.

If the learning rate is too small than the optimal value, then it would take a much longer time (hundreds or thousands) of epochs to reach the ideal state

If our learning rate is too large than optimal value, then it would overshoot the ideal state and our algorithm might not converge

BATCH SIZE

Batch Size is the number of training instances used by the model to complete one forward and one backward pass.

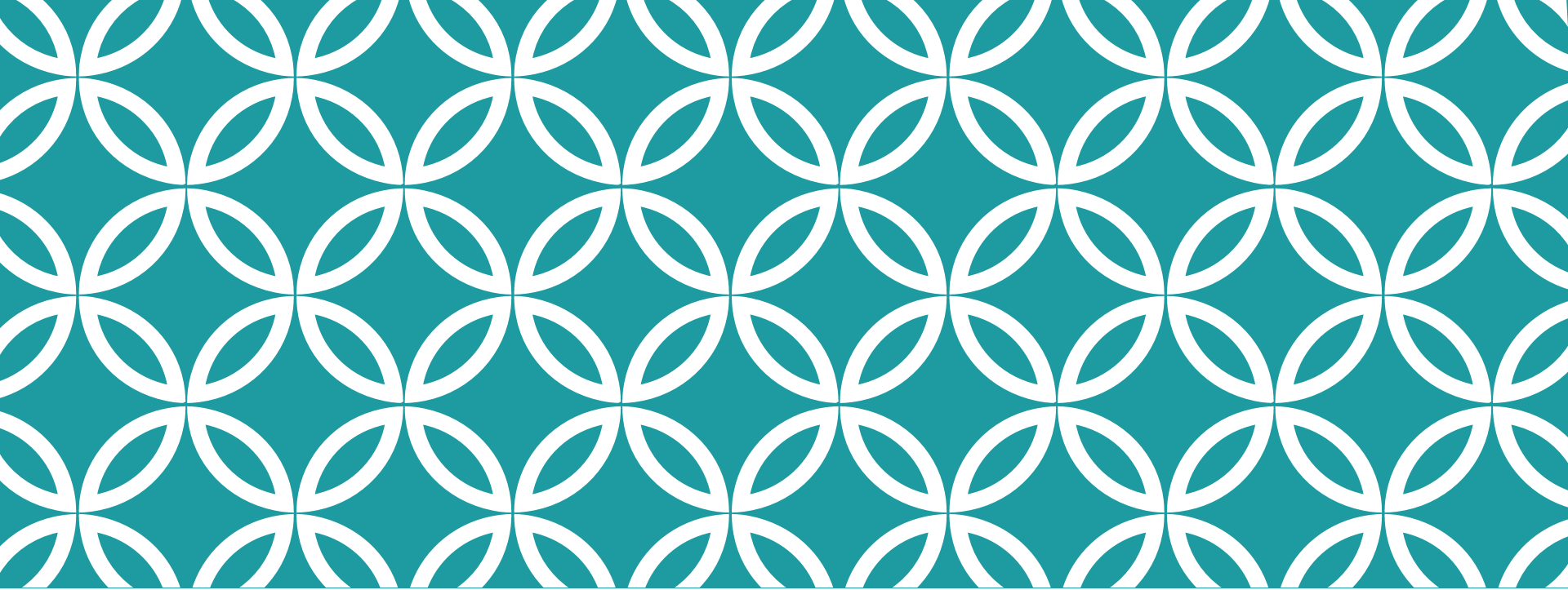
BATCH SIZE

- A small minibatch size induces more noise during error calculations/Loss and this is preferred in preventing the training process from stopping at local minima.
- A larger minibatch size gives us the computational boosts but comes at the expense of needing more memory for the training process.
- Always choose the maximum size of batch size that can be fit in the memory for optimal performance of Deep Learning Models

EPOCHS

In most cases, the number of epochs are not required to be tweaked, we would just employ Early Stopping.

Early Stopping:
A technique used in Deep Learning training, where we monitor the model training, and stop model training when the model performance stops improving on Validation dataset



HYPER PARAMETERS

Model

NUMBER OF HIDDEN LAYERS & UNITS

As the complexity of the data representation increases, this require the more number of neurons & layers. Since there is no magic number for this, below strategies can be employed during model training.

1. Experimentation
2. Intuition
3. Prefer to increase Depth
4. Borrow Ideas from Research papers

ACTIVATION FUNCTION

Sigmoid functions and their combinations generally work better in the case of classifiers

Sigmoids and tanh functions are avoided in some scenarios due to the vanishing gradient problem (More on this in later section)

ReLU function is a general activation function – most used activation function

If we observe a case of dead neurons in our Neural networks, at that time leaky ReLU function is the best choice

Relu is used only on Hidden layers

Start with Relu, then proceed to other activations based on the result.