## Neural Networks

# Introduction to Deep Learning

# Agenda

- Introduction to neural networks
  - Neural Networks
  - Neurons
- Activation functions
  - Sigmoid, Tanh, ReLU
- Feed Forward neural network
  - Layer Details
- Training a neural network

# Agenda

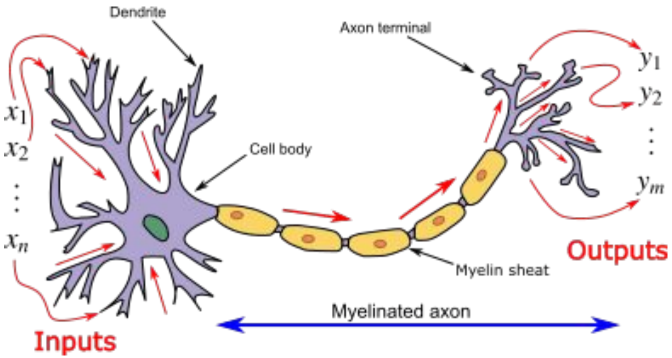
- Error and Loss function
- Optimization
- Gradient descent
  - Gradient
  - Gradient Descent Variations
  - Backpropagation
- Summary

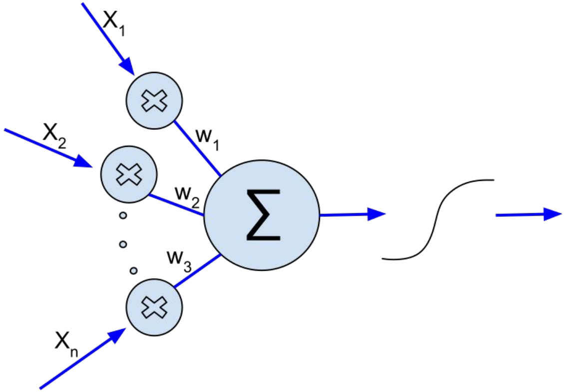
# Neural Networks

*Artificial Neural Networks are computing systems inspired from biological neuron*

# Neuron

Artificial neuron is inspired by biological neuron



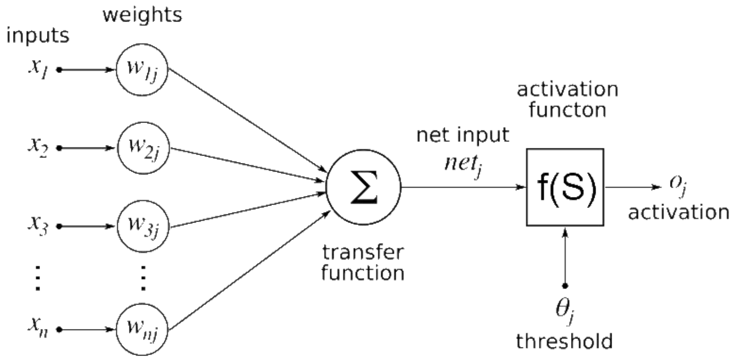



# Activation function

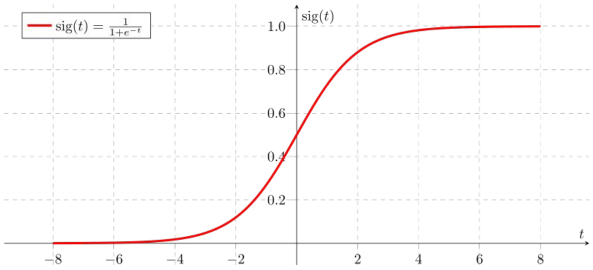
In artificial neural networks, helps in defining the output of a node when a input is given.

Different types of activations helps in different tasks.  Some examples are -

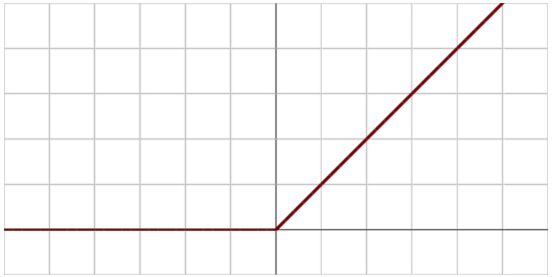
- Sigmoid
- Tanh
- ReLU
- Binary Step Function



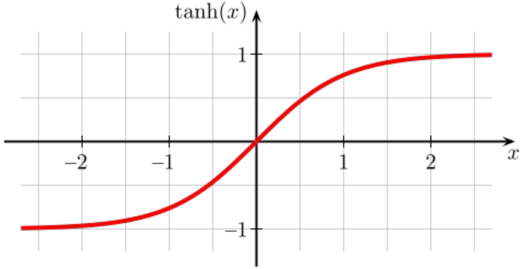
# Types of activation function




Sigmoid



R eLU




Tanh

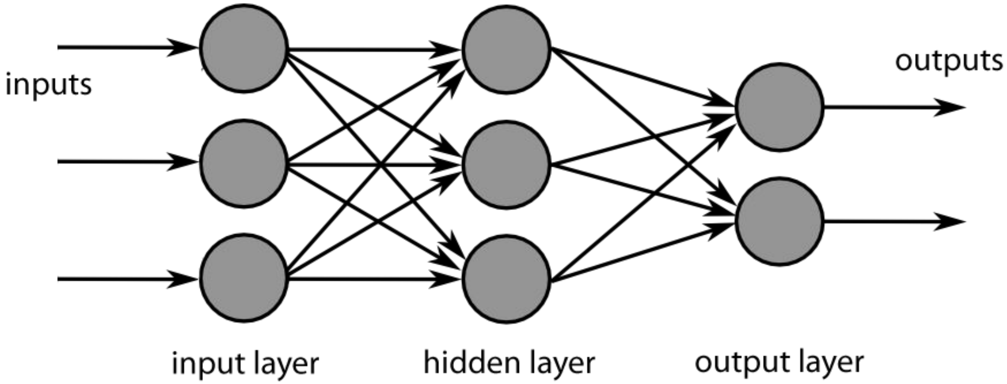
Activation function performs certain mathematical operations on it's input, which is a number

Question:
What is the range of Sigmoid, Tanh and ReLU?

Answer:
Sigmoid (0,1)
Tanh (-1, 1)
ReLU (0, max)

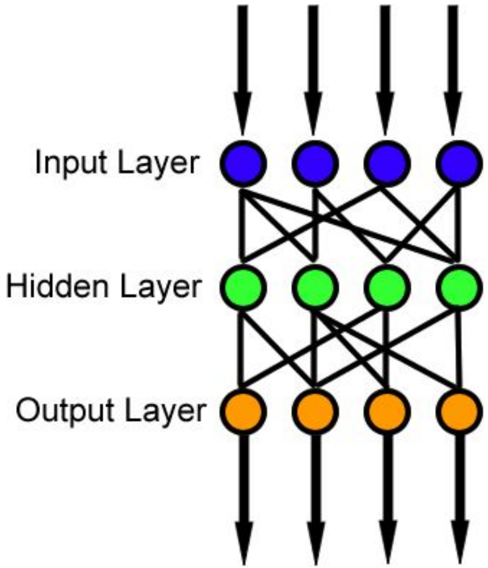

Activation function takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell.

# Feed forward neural network




This is a 2-layer neural network. One is the hidden layer (having 3 neurons) and the other is output layer (having 2 neurons).

# Layer details




Output layer

- Represents the output of the neural network

Hidden layer(s)

- Represents the intermediary nodes.
- It takes in a set of weighted input and produces output through an activation function

Input layer

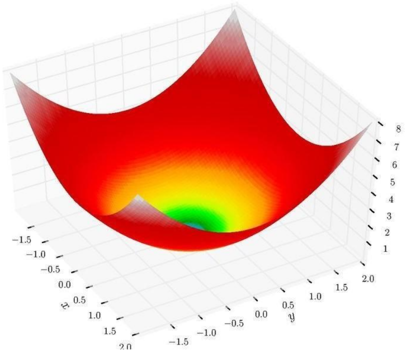
- Represents dimensions of the input vector (one node for each dimension)

# Training a neural network

- Decide the structure of network
- Create a neural network
- Choose different hyper-parameters
- Calculate loss
- Reduce loss
- Repeat last three steps

# Error and Loss function

- In general, error/loss for a neural network is difference between actual value and predicted value.
- The goal is to minimize the error/loss.
- Loss Function is a function that is used to calculate the error.
- You can choose loss function based on the problem you have at hand.
- Loss functions are different for classification and regression

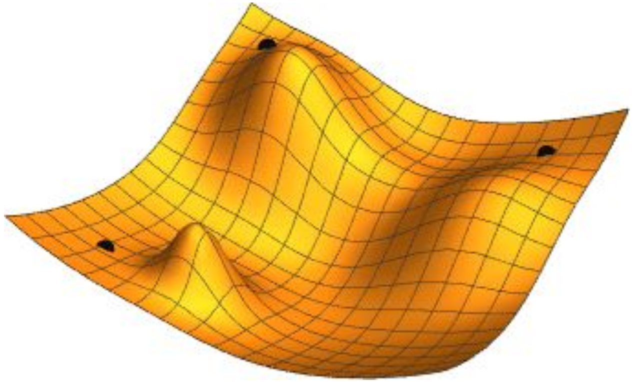


# Optimization

In optimization, the main aim is to find weights that reduce loss

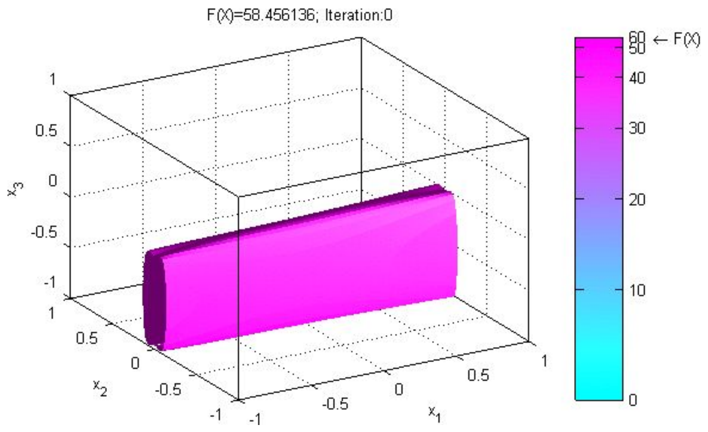
# Gradient

- Gradient is calculated by optimization function
- Gradient is the change in loss with change in weights.
- The weights are modified according to the calculated gradient.
- Same process keep on repeating until the minima is reached



# Gradient descent

Gradient descent is a method that defines a cost function of parameters and uses a systematic approach to optimize the values of parameters to get minimum cost function.



Question:
Do we get multiple local optimum solutions if we solve using gradient descent.

Answer:
False. We get only one local optimum solution after using gradient descent

# Gradient descent variations

Gradient descent has 3 variations, these differ in using data to calculate the gradient of the objective function

1. Batch gradient descent
   - Updates the parameter by calculating gradients of whole dataset
2. Stochastic gradient descent
   - Updates the parameters by calculating gradients for each training example
3. Mini -batch gradient descent
   - Updates the parameters by calculating gradients for every mini batch of "n" training example
   - Combination of batch and stochastic gradient descent

# Backpropagation

- Backpropagation is used while training the feedforward networks
- It helps in efficiently calculating the gradient of the loss function w.r.t weights
- It helps in minimizing loss by updating the weights

# Learning rate and Momentum

- The learning rate is a hyperparameter which determines to what extent newly  acquired weights overrides old weights. In general it lies between 0 and 1.
- You can try different learning rates for a neural networks to improve results.
- Momentum is used to decide the weight on nodes from previous iterations. It  helps in improving training speed and also in avoiding local minimas.

# Summary of the Neural Network Process

- A neural network is made of neurons
- These neurons are connected to each other
- Every neuron has an activation function that defines it's output
- Then we train our neural network to learn the parameter values i.e. weights and biases
- This process consists of forwardprop and backprop
- After forward prop we calculate the loss using a loss function and propagate the information backwards, that's backprop
- This process is repeated layer by layer, until all the neurons receive a loss signal which describes their contribution to the total loss

# TensorFlow

## Introduction

# Agenda

- What is tensorflow?
- Why we are using tensorflow?
- TensorFlow 2.x
- TensorFlow 2.x - features and changes
- Changes with respect to TensorFlow 1.x
- Keras and it's advantages
- Keras vs tf.keras
- Tutorials and guides
- Notes

# TensorFlow

- TensorFlow is a machine learning library by Google
- It is open sourced
- It is mainly used for implementing neural networks
- It is an end-to-end platform, which means you can use it for building your models from scratch to deploying them into a production environment

# Why TensorFlow for this course?

- Most used library for deep learning
- Easy transition from research to production
- Extensive industry support
- Easy deployment around servers, mobile devices, web platforms etc
- You can easily sort out the issues using GitHub, StackOverflow etc
- Used by world's top AI companies
- Consistently updated with cutting edge changes

# Some technical reasons

- Easy and readable syntax
- Being a low-level library it provide more flexibility to developers to implement  their own functionalities and services
- Provides high level API for implementing advanced neural net architectures
- Distributed training

# OS Support

TensorFlow is supported on following 64-bit systems:

- Ubuntu 16.04 or later
- macOS 10.12.6 (Sierra) or later (no GPU support)
- Windows 7 or later
- R aspbian 9.0 or later

# Language support

- Python
- C++
- JavaScript
- Java
- Go
- Swift

# What is a Tensor?

- A tensor is an n-dimensional array which can be a scalar, vector or matrix etc
- Tensorflow uses data flow graphs for parallel computing
- Data flow graphs provide parallelism, distributed execution, faster compilation and portability
- Tensorflow can be used for developing cool projects like Image classification, speech recognition, object detection, transfer learning etc.

# Tensors

- A Tensor is a multi-dimensional array.
- Similar to NumPy ndarray objects, tf.Tensor objects have a data type and a shape. Additionally, tf.Tensors can reside in accelerator memory (like a GPU).
- TensorFlow offers a rich library of operations (tf.add, tf.matmul, tf.linalg.inv etc.) that consume and produce tf.Tensors.
- These operations automatically convert native Python types, for example:

```python
print(tf.add(1, 2))
print(tf.add([1, 2], [3, 4]))
print(tf.square(5))
print(tf.reduce_sum([1, 2, 3]))

# Operator overloading is also supported
print(tf.square(2) + tf.square(3))
```

# Tensor vs Numpy

The most obvious differences between NumPy arrays and tf.Tensors are:

1. Tensors can be backed by accelerator memory (like GPU, TPU).
2. Tensors are immutable.

## NumPy Compatibility

Converting between a TensorFlow tf.Tensors and a NumPy ndarray is easy:

- TensorFlow operations automatically convert NumPy ndarrays to Tensors.
- NumPy operations automatically convert Tensors to NumPy ndarrays.

Tensors are explicitly converted to NumPy ndarrays using their .numpy() method.

# GPU Acceleration

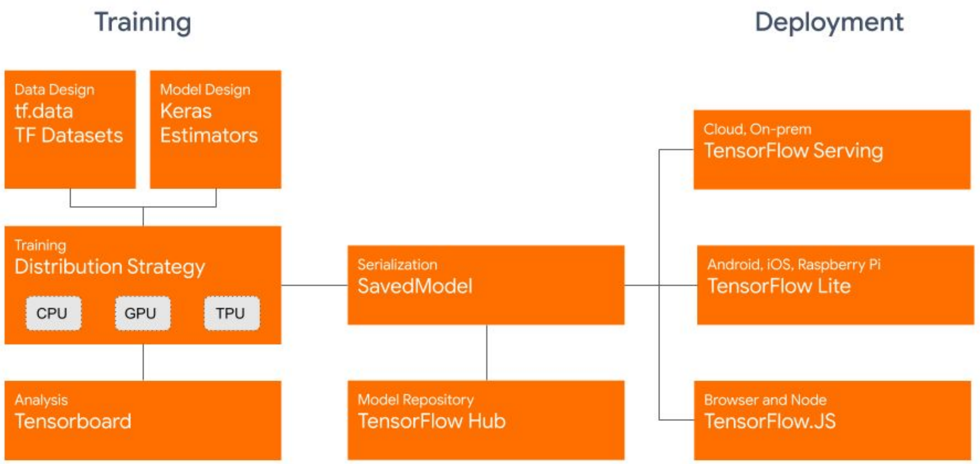
- Many TensorFlow operations are accelerated using the GPU for computation.
- Without any annotations, TensorFlow automatically decides whether to use the GPU or CPU for an operation—copying the tensor between CPU and GPU memory, if necessary.
- Tensors produced by an operation are typically backed by the memory of the device on which the operation executed, for example:

```
x = tf.random.uniform([3, 3])

print("Is there a GPU available: "),
print(tf.config.experimental.list_physical_devices("GPU"))

print("Is the Tensor on GPU #0:  "),
print(x.device.endswith('GPU:0'))
```

```
Is there a GPU available:
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
Is the Tensor on GPU #0:
True
```

# Coding with Tensorflow

- TensorFlow 2.0 makes development of DL applications much easier.
- With tight integration of Keras into TensorFlow, eager execution by default, and Pythonic function execution, TensorFlow 2.0 makes the experience of developing applications as familiar as possible for Python developers.




Source

# Basics with TensorFlow

1. Import tensorflow library

```
import tensorflow as tf
```

2. Load/Read the data

```
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
```

3. Build your model

```
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)
])
```

4. Compile the model with optimizer, loss function and error metric for back propagation

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

5. Fit your model on your train data

```
model.fit(x_train, y_train, epochs=5)
```

# New version - TensorFlow 2.x

- Much easier development
- Tight integration with Keras
- More familiar for Python developers
- Deploy anywhere across servers, mobile and edge devices, browser and  Node.js with TensorFlow Extended, TensorFlow Lite and TensorFlow .J S
- Multi GPU support

# What’s new in TF2?

- tf.function(): Creates a callable TensorFlow graph from a Python function
- tf.GradientTape(): Records operations for automatic differentiation
- tf.data(): helps in building complex input pipelines from simple, and reusable pieces

# Changes with respect to TensorFlow 1.x

- Cleaning up of APIs: many APIs are either gone or moved
- Eager execution (default): no need manually compile the abstract syntax tree
- No more globals: no need to track variables
- Introduction of function and elimination of session

# Model building: from simple to flexible

1. Sequential API + built in layers -----> New users
2. Functional API + built in layers -----> Engineers with standard use case
3. Functional API + custom layers + custom metrics + custom losses -----> Engineers requiring control
4. Subclassing: write everything yourself from scratch -----> Researchers

# Keras

# Keras

- Keras is a high-level neural network API
- Written and implemented in Python
- Can run on top of TensorFlow
- It was designed keeping fast experimentation in mind.

# Keras advantages

- User-friendly
  - Simple and user friendly interface. Actionable feedbacks are also provided.
- Modular and composable
  - Modules are there for every step, you can combine them to build solutions.
- Easy to extend
  - Gives freedom to add custom blocks to build on new ideas.
  - Cusrom layers, metrics, loss functions etc. can be defined easily.

# Keras vs tf.keras

- In TF2 instead of writing "import keras" you will write "from tensorflow import keras"
- In colab, if you see the message "Using TensorFlow Backend", you are not using tensorflow 2.x implementation of keras
- tf.keras is the TensorFlow's implementation of keras so it supports all the latest changes in the TensorFlow version
- tf.keras is also better in support and maintenance

# Latest tutorials and guides

- https://www.tensorflow.org/tutorials
- https://www.tensorflow.org/guide

# Notes

- TensorFlow 2.0 announcement video: https://www.youtube.com/watch?v=EqWsPO8DVXk
- Guide to convert your code from TF 1 to TF 2: https://www.tensorflow.org/guide/migrate
- Detailed introduction video: https://www.youtube.com/watch? v=5ECD8J3dvDQ

# Thank you!

Happy Learning :)