Notes by Izam Mohammed

Tensorflow

Overview
TensorFlow uses a computational graph to represent a series of mathematical operations (nodes) and the flow of data between them (edges).

```
import tensorflow as tf

# Create tensors
tensor_1 = tf.constant([[1, 2, 3], [4, 5, 6]])
tensor_2 = tf.constant([[7, 8, 9], [10, 11, 12]])

# Create variables
variable_1 = tf.Variable(tensor_1)
variable_2 = tf.Variable(tensor_2)

# Perform operations on tensors and variables
sum_tensor = tf.add(tensor_1, tensor_2)
product_tensor = tf.multiply(tensor_1, tensor_2)
sum_variable = tf.add(variable_1, variable_2)
```

Advantages

- *OpenSource and Active Community*
- *Flexible and scalable*
- *Extensive documentation and Resources*
- *Tensorboard for visualization*
- *TFX for production*

Disadvantages

- *No windows support*
- *Comparatively slow*
- *Only NVIDIA GPU support*
- *Architectural limitations*
- *Inconsistent*

Extra terms

- *Tensor:*
  - *The fundamental data type in Tensorflow. Represents multi-dimensional arrays*
  - *Can be constants or variables*
  - *Can have different ranks (dimensions)*
  - *Types*
    - *Scaler Tensor → rank-0*
    - *Vector Tessor → rank-1*
    - *Matrix Tensor → rank-2*

        - *3D and higher dimensional tensors → rank > 2*
        - *Sparse Tensor → represents tensors with a large number of zero values*
        - *Ragged tensor → tensors with non-uniform shapes, tf.ragged.constat([..])*
        - *Variable Tensor → mutable tensors, tf.Variable([1,2,3])*
        - *String Tensor → tf.constant("hello world")*
- *tf.constant: create a constant tensor*
- *tf.Variable: typically used for model parameters that need to be updated during the training*
    - *Some methods*
- *tf.function → decorator that converts a Python function to a TensorFlow graph function*
- *tf.optimizers → Contains various optimization functions for updating the model*
- *tf.keras.models → Module for defining and training ml models using high-level APITf.data → Module for building scalable and efficient input pipelines for data preprocessing and augmentation*
- *tf.keras.layers → A module that provides a variety of pre-built layers for nn*
- *tf.image → Module for image processing operations such as resizing, cropping, etc*
- *tf.reduce_mean, tf.reduce_sum → Methods for reducing tensors along specified dimensions.*
- *tf.losses → Module that contains various loss functions*
- *tf.saved_model.load() → load the saved model in tf*

Keras

Keras is a high-level neural networks API written in Python and integrated into TensorFlow.

```python
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Create a simple feedforward neural network
model = Sequential([
    Dense(64, activation='relu', input_shape=(input_dimension,)),
    Dense(32, activation='relu'),
    Dense(output_dimension, activation='softmax')
])

# Display the model architecture
model.summary()
```

Why Keras?

- *Simplicity and user-friendly interface*
- *Compatibility with TensorFlow*
- *Modularity and extensibility: can stack predefined building blocks together (layers)*
- *Gives consistency*
- *Integrated with TensorFlow tools such as TFX*

Pytorch

Advantages

- ➢ Dynamic Computational Graph: also known as eager execution, making easy to debug and experiment models
- ➢ Extensive community and research scope
- ➢ TorchScript for production
- ➢ Native support for Dynamic models → well suited for sequence-to-sequence models and RNN
- ➢ Libraries available for research → torch vision, torch audio

Disadvantages

- ➢ Deployment overhead, compared to tf
- ➢ Less Production-ready tools
- ➢ Smaller community compared to tf
- ➢ Not have enough deployment options for mobile and embedded devices

Extra terms

- ➢ Tensor → The fundamental data type in PyTorch, representing a multi-dimensional array
    - ○ torch.Tensor([1,2,43])

```python
import torch

# Create tensors
tensor_1 = torch.tensor([[1, 2, 3], [4, 5, 6]])
tensor_2 = torch.tensor([[7, 8, 9], [10, 11, 12]])

# Create variables (requires_grad is set to True for tracking grad
variable_1 = torch.tensor(tensor_1, requires_grad=True)
variable_2 = torch.tensor(tensor_2, requires_grad=True)

# Perform operations on tensors and variables
sum_tensor = tensor_1 + tensor_2
product_tensor = torch.mul(tensor_1, tensor_2)
sum_variable = variable_1 + variable_2
```

Methods

- ➢ torch.nn.Module → base class for all neural network model

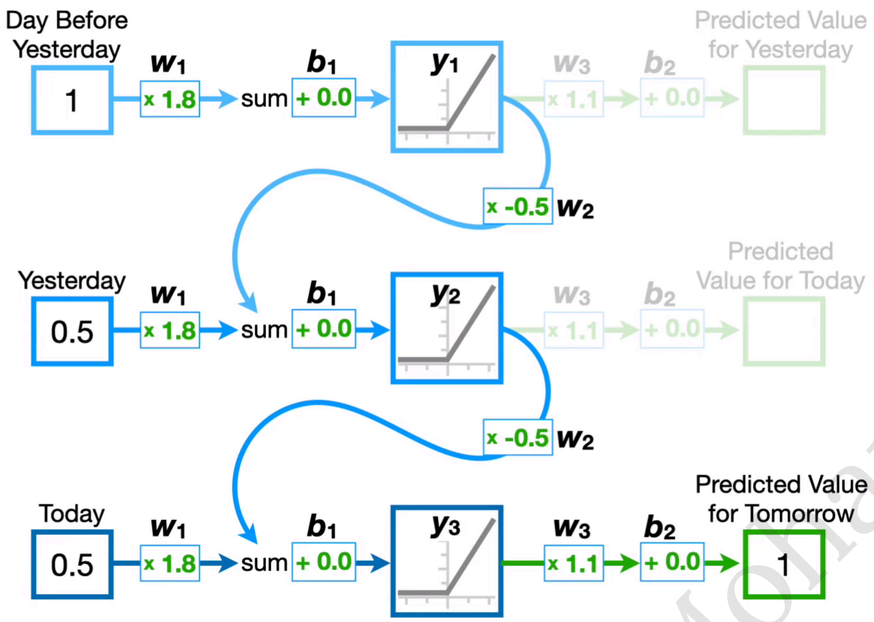
```python
import torch.nn as nn
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc = nn.Linear(10, 5)
```

- ➢ torch.nn.functional → a module containing a variety of functions that can be applied to tensors
- ➢ torch.nn.optim → module that contains various optimization algos
- ➢ torch.utils.data.Dataset → An abstract class representing a dataset in torch

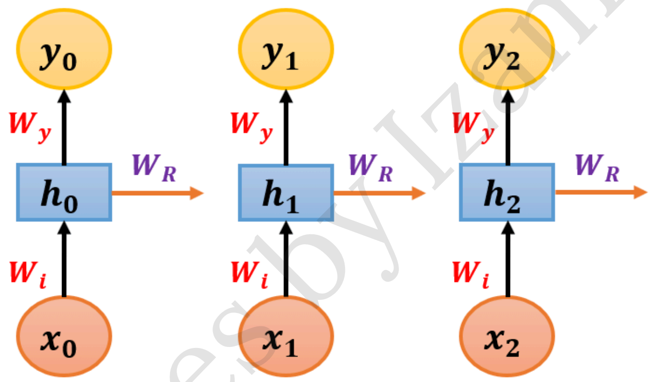
RNN

Is it a type of neural network architecture designed to process sequential data.

eg:-



So each of the RNN units has 3 weights and 2 biases. Bias will be there at the calculation of the output value and at the calculation of the hidden state.



Advantages

- *Sequential information processing*
- *Flexibility in input and output length*
- *maintain a hidden state that captures information from previous inputs*
- *Natural representation of temporal patterns*

Disadvantages

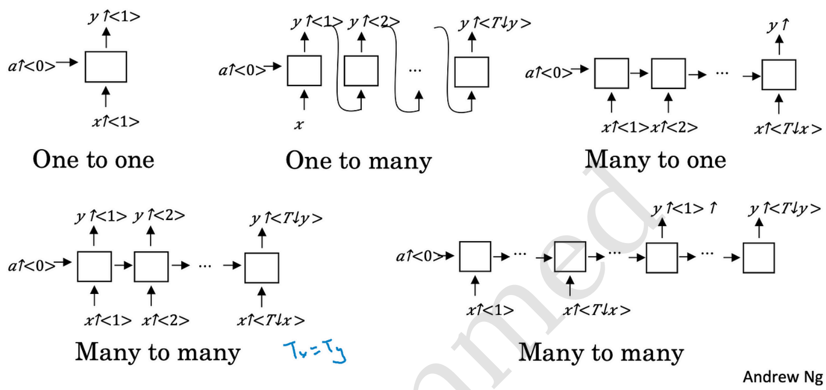
- *Vanishing gradient → gradients become too small in training*
- *Exploding gradient → gradients become extremely high in training*
- *Difficulty in Learning Long-term dependencies*
- *Limited context*
- *Difficulty in irregular time intervals*
- *Difficulty in capturing global context*

Extra terms

- *Hidden state → a vector that represents the memory of the RNN, only in GRU and LSTM*
- *BPTT → Backpropagation through time*

Types

- *One-to-one → normal FNN*
- *One-to-Many → A single input time step is used to generate multiple output timesteps, for ex:-music generation*
- *Many-to-one → sentiment analysis*
- *Many-to-Many → sequence to sequence models*

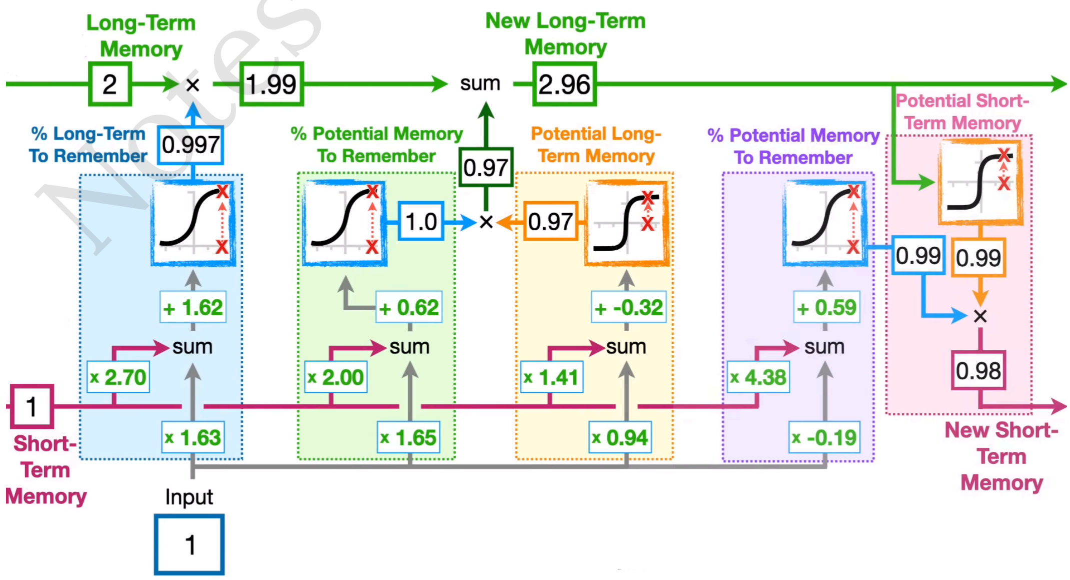


Evaluation metrics

- *Accuracy, precision, recall, f1-score → classification*
- *MSE, RMSE, MAE, R2 → regression*
- *BLEU score (bilingual evaluation understudy) →*
    - *measures the quality of the machine-generated text compared to human-generated reference text.*
    - *Primarily focus on the n-gram precision*
    - *The BLEU score is between 0 and 1*
    - *BLEU = brevity penalty* **x** *exp of cumulative log precision*
- *METEOR, ROUGE → similar to BLEU*

LSTM

Designed to overcome the vanishing gradient problem in RNN



Terms

- *Cell state → runs along the sequence, allowing information to be passed from one time step to the other*
- *Hidden state → responsible for maintaining and updating the context information*
- *Gates*
  - *Forget Gate → determines what information from the cell state should be discarded*
  - *Input Gate → determines what new information to store in the cell state*
  - *Output Gate → chooses what information to output based on the cell state*

Advantages

- *Handle long-term dependencies*
- *Gating mechanism*
- *Robust training*
- *Adaptability to various tasks*

Disadvantages

- *Complexity*
- *Overfitting → mainly in small datasets*
- *Training time will be higher*
- *Difficulty in interpretability*

Why tanh ???

- *Tanh's output range is (-1, 1)*
- *Don’t have the vanishing gradient problem*
- *Underlying for the gating system in lstm*

Regularization

- ***L2 regularization***
  - *Also known as weight decay*
  - *It adds a penalty term to the loss function*
  - *Prevent overfitting*

$$\text{loss} = \text{original loss} + \frac{\lambda}{2}\sum_i \|W_i\|_2^2$$

- ***Batch normalization***
  - *designed to address the internal covariate shift problem during training.*
  - *It normalizes the input of each layer by subtracting the mean and dividing it by the standard deviation.*
  - *Can increase stability*

$$\text{Batch Normalization}(x) = \frac{x - \text{mean}(x)}{\sqrt{\text{variance}(x) + \epsilon}} \times \gamma + \beta$$

- ***Dropout***
  - *It randomly drops (sets to zero) a fraction of the input units during training.*
  - *It helps prevent overfitting by introducing redundancy in the network.*

    - *Prevent overfitting*

Loss fn

- *Binary classify → Binary cross-entropy*
- *Multiclass → categorical cross-entropy*
- *Multiclass (when target values are integers rather than one-hot vectors) → sparse categorical cross-entropy*
- *Multilabel → Binary cross-entropy*
- *Regression → MSE, MAE, Huber loss*

Optimizers

- *Batch Gradient Descent → the traditional one*
- *SGD → divide into batches and do gradient descent*
- *RMSProp → Different learning rates for each*
- *Momentum → have a momentum oscillator*
- *Adam → momentum + RMSProp*