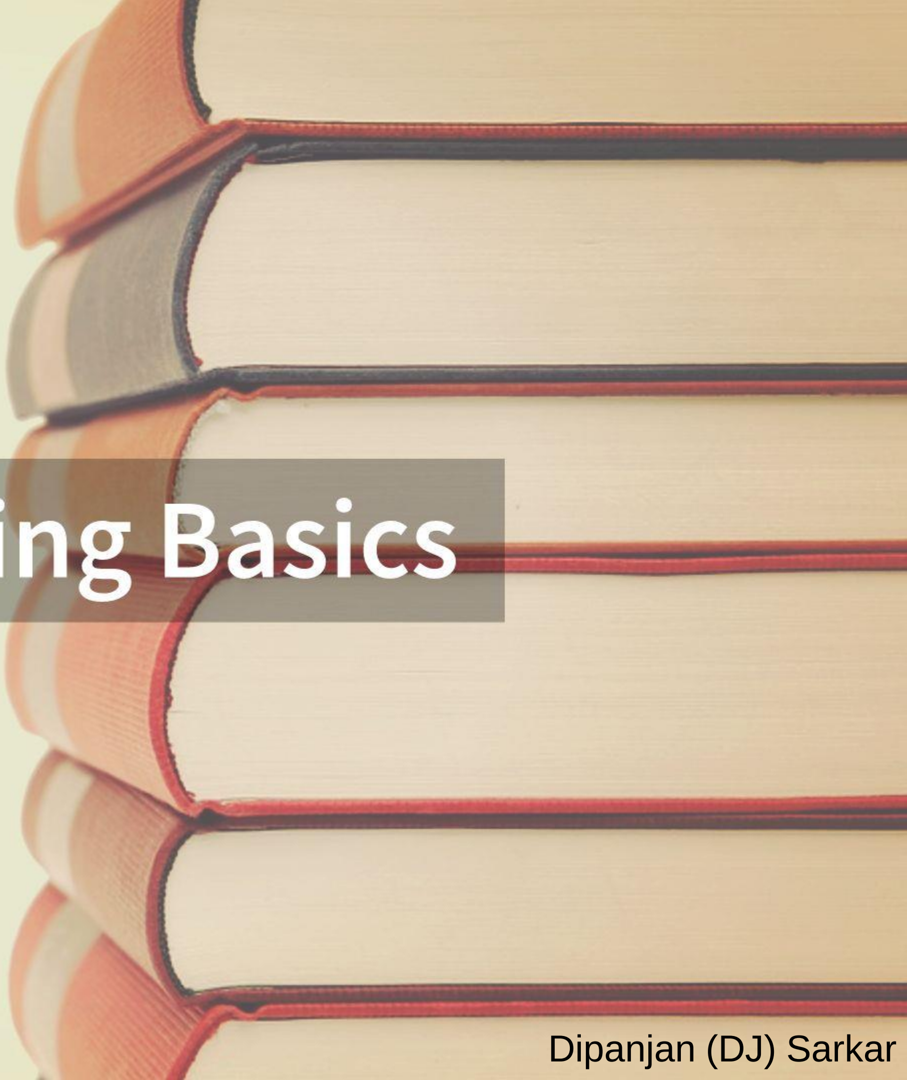
Deep Learning Basics



Dipanjan (DJ) Sarkar

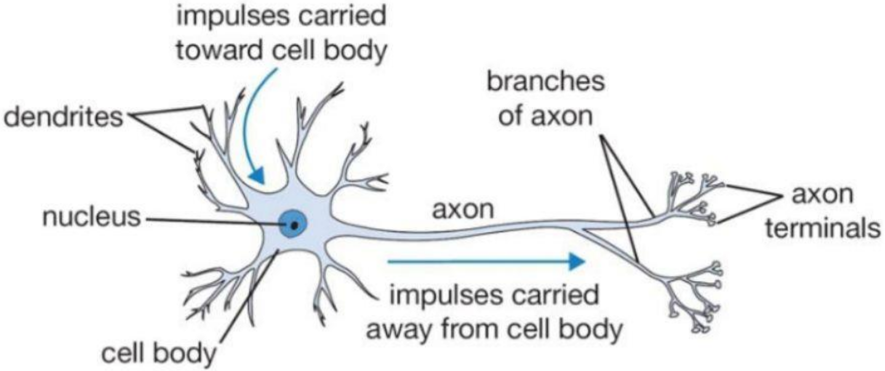
Repo: https://bit.ly/tf2-crash-course

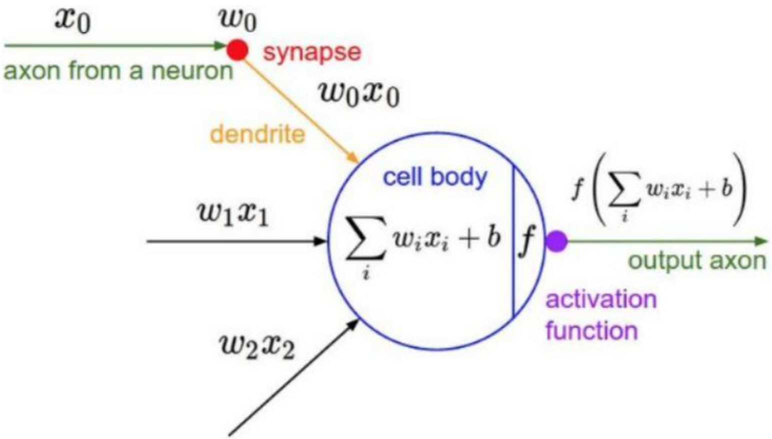
Why Learning?

Sheepdog or mop?

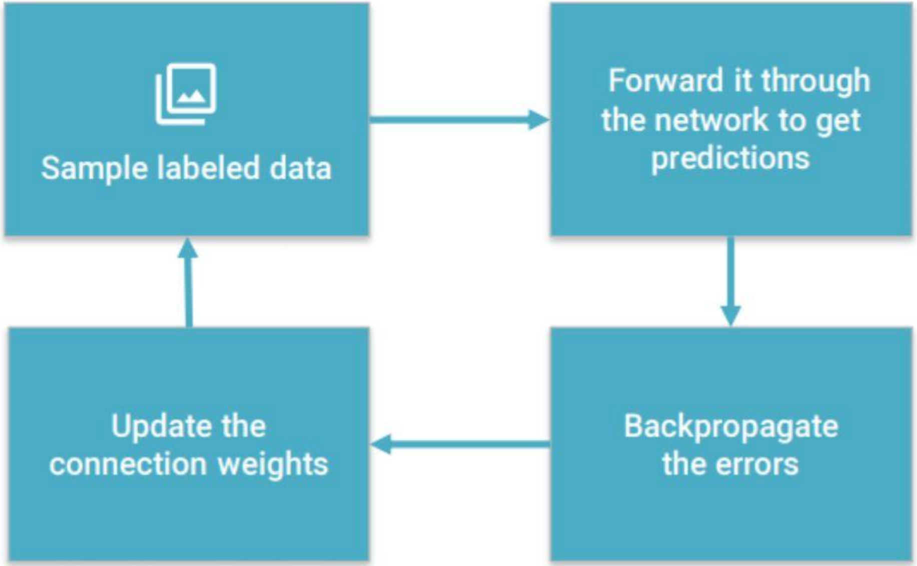
Chihuahua or muffin?

Neural Networks



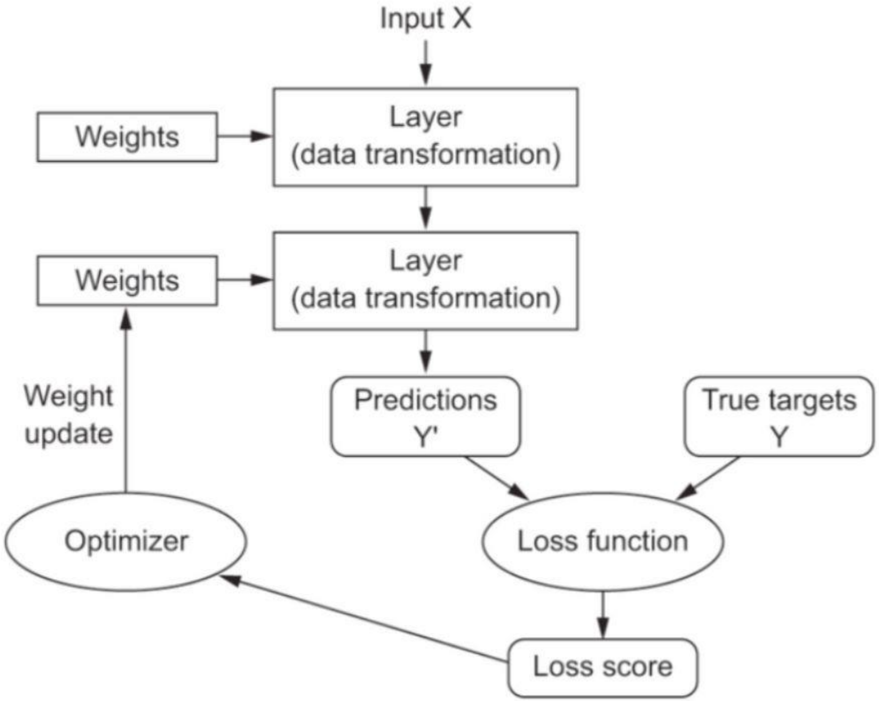


Neural Networks - Training


Sample labeled data
Forward it through the network to get predictions
Backpropagate the errors
Update the connection weights

Learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then using this error signal to change the weights (or parameters) so that predictions get more accurate.

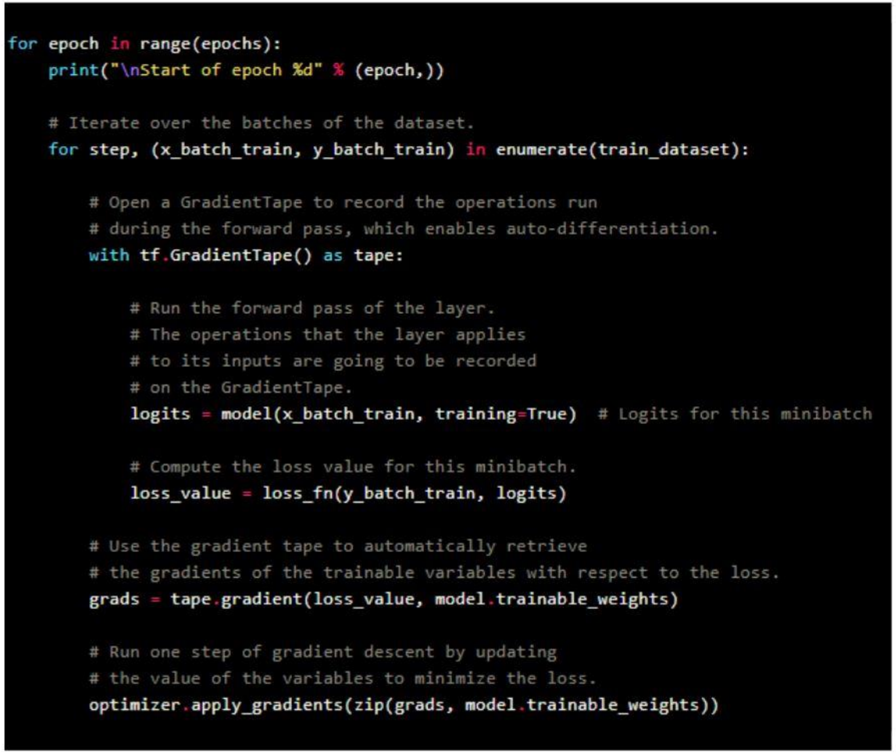
Training a Neural Network



Deep Learning with Python, Francois Chollet

- **Perform parameter updates to minimize the loss (training objective)**
- **Typical flow involves:**
  - Forward pass with the input going through various transformations
  - Compute the loss based on predictions and actuals
  - Compute gradients
  - Backpropagate gradients to update layer weights
- **TensorFlow / PyTorch enables easy Automatic Differentiation**

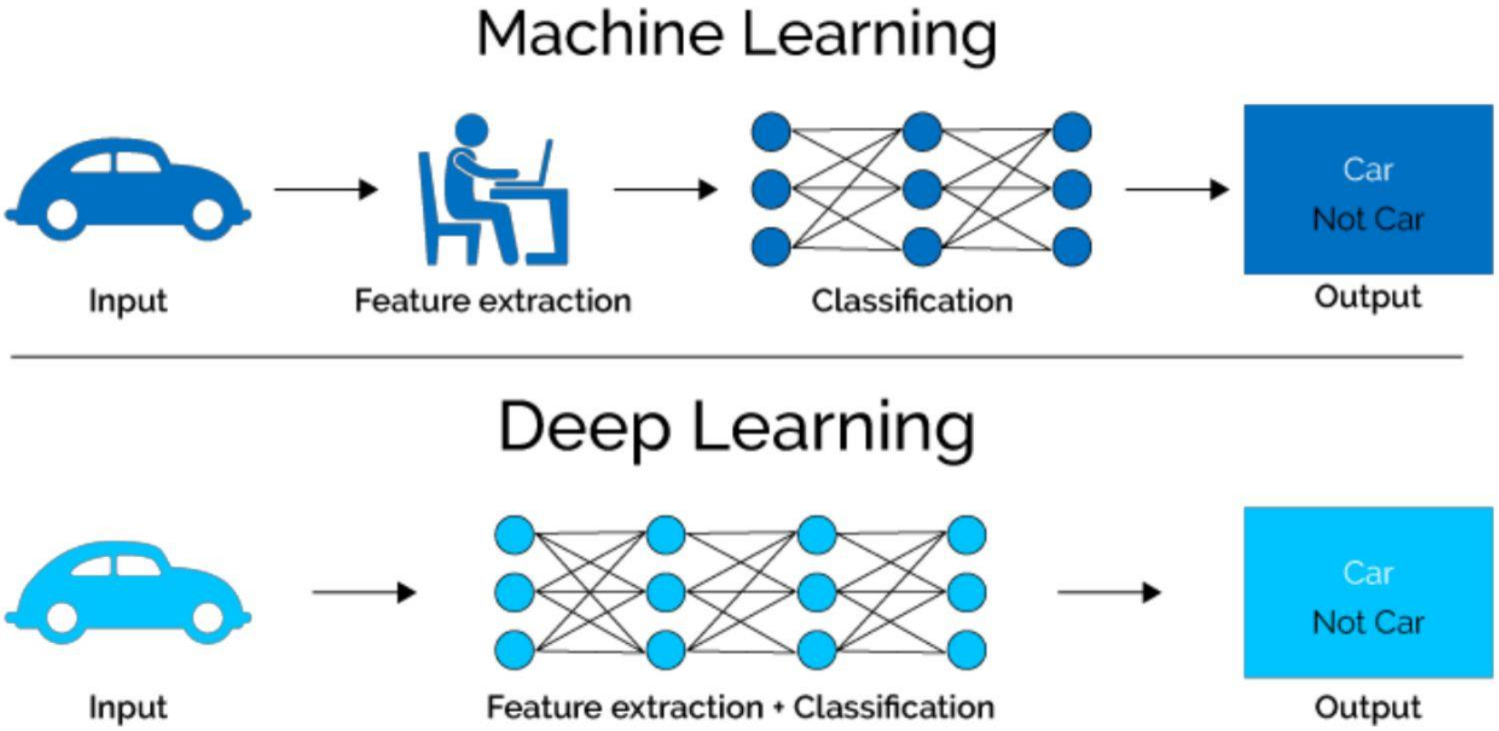
Custom Training Loops in TensorFlow



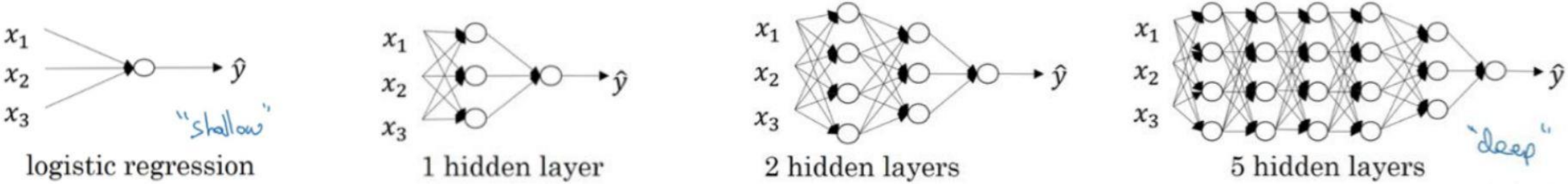
```
for epoch in range(epochs):
    print("\nStart of epoch %d" % (epoch,))

    # Iterate over the batches of the dataset.
    for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):

        # Open a GradientTape to record the operations run
        # during the forward pass, which enables auto-differentiation.
        with tf.GradientTape() as tape:

            # Run the forward pass of the layer.
            # The operations that the layer applies
            # to its inputs are going to be recorded
            # on the GradientTape.
            logits = model(x_batch_train, training=True)  # Logits for this minibatch

            # Compute the loss value for this minibatch.
            loss_value = loss_fn(y_batch_train, logits)

        # Use the gradient tape to automatically retrieve
        # the gradients of the trainable variables with respect to the loss.
        grads = tape.gradient(loss_value, model.trainable_weights)

        # Run one step of gradient descent by updating
        # the value of the variables to minimize the loss.
        optimizer.apply_gradients(zip(grads, model.trainable_weights))
```

- GradientTape records all relevant NN operations in the forward pass
- Hence easy to compute gradients in reverse order during the backward pass
- Useful to extract relevant gradients w.r.t the loss (used in adversarial attacks)

Deep Learning - ML on Steroids



Deep Learning Models - Multi-layered Neural Networks



logistic regression

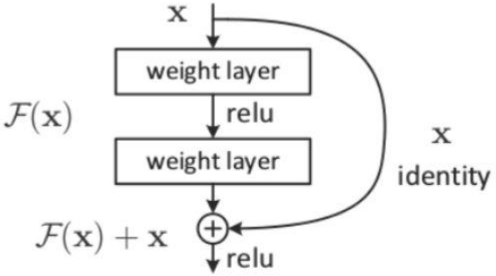
1 hidden layer

2 hidden layers

5 hidden layers

Why is Deep Learning effective now?

New algorithms



More data

Software

Caffe2 Microsoft CNTK TensorFlow theano

Chainer PYTORCH dmlc mxnet K

Faster compute engines

Deep Learning Model Architectures

1. **Convolutional Neural Networks (CNNs)**

   Used extensively in computer vision problems with image, video. Can also be used for audio and text

2. **Recurrent Neural Networks**

   Good for sequential data, used for time series forecasting and NLP problems

3. **Long Short Term Memory Networks (LSTMs)**

   Can remember longer sequences of data and better than RNNs

4. **Gated Recurrent Units (GRUs)**

   Can remember longer sequences of data and faster than LSTMs

5. **Bi-directional Models**

   Processes sequences of data in both directions for capturing better contextual information

6. **Auto-encoders**

   Learns efficient latent data representations in lower dimensions using unsupervised learning

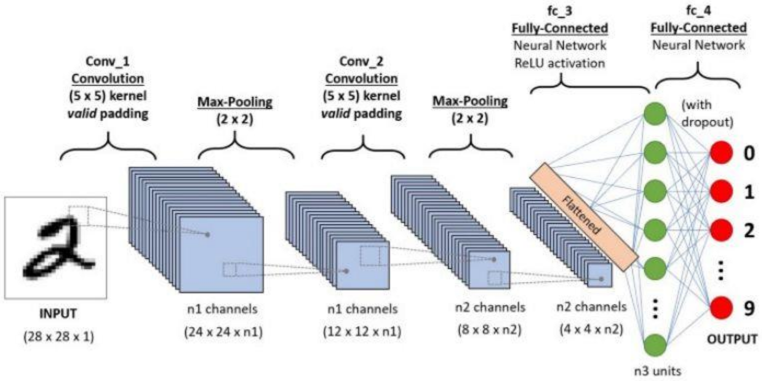
7. **Encoder-Decoder Models**

   Takes in a sequence of data and generates a sequence of data as output

8. **Transformer Models**

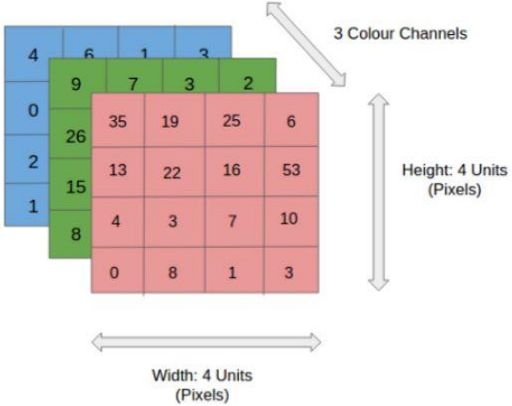
   Stack of encoder-decoder models used for language modeling and can be tuned for different NLP tasks

Convolutional Neural Networks (CNNs)



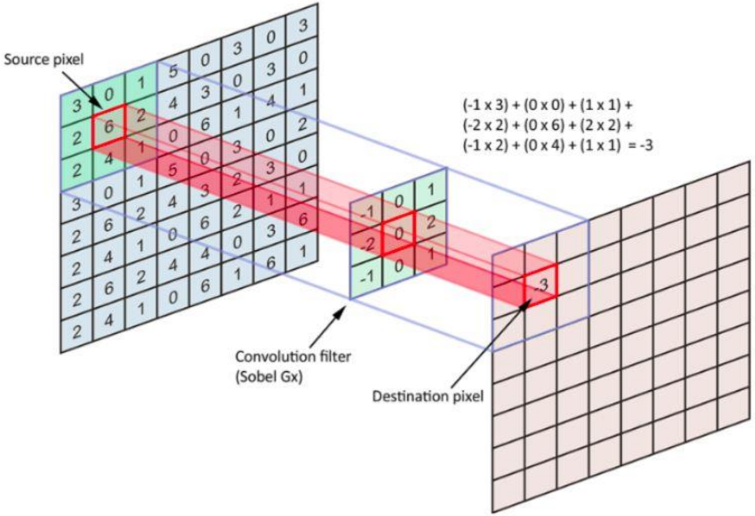
- CNNs have a layered architecture of several layers to learn hierarchical spatio-temporal features
- Convolution Layers use convolution filters to build feature maps (feature extraction)
- Pooling Layers help in reducing dimensionality after convolutions (compression)
- Non-linear activation functions are applied in the network as usual
- Dropout or BatchNormalization Layers may be used to prevent model overfitting
- FC Layers in final stages help with flattening and prediction

Convolutional Neural Networks (CNNs)



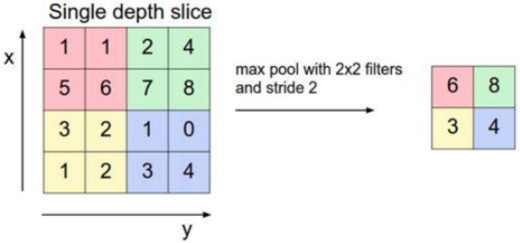
35	19	25	6
13	22	16	53
4	3	7	10
0	8	1	3

Source Image



(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

Convolution Layer



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Pooling Layer