Unit 3

Cost Function

- A cost function is a measure of error between what value your model predicts and what the value actually is. For example, say we wish to predict the value yi for data point xi.

$$f_\theta(x_i)$$

- represent the prediction or output of some arbitrary model for the point xi with parameters θ. One of the many cost functions could be

$$\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

this function is known as the L2 loss. Training the hypothetical model we stated above would be the process of finding the θ that minimizes this sum.
fter training your model, you need to see how well your model is performing. While accuracy functions tell you how well the model is performing, they do not provide you with an insight on how to better improve them. Hence, you need a correctional function that can help you compute when the model is the most accurate, as you need to hit that small spot between an undertrained model and an overtrained model.

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting

Consider a robot trained to stack boxes in a factory. The robot might have to consider certain changeable parameters, called Variables, which influence how it performs. Let's say the robot comes across an obstacle, like a rock. The robot might bump into the rock and realize that it is not the correct action.

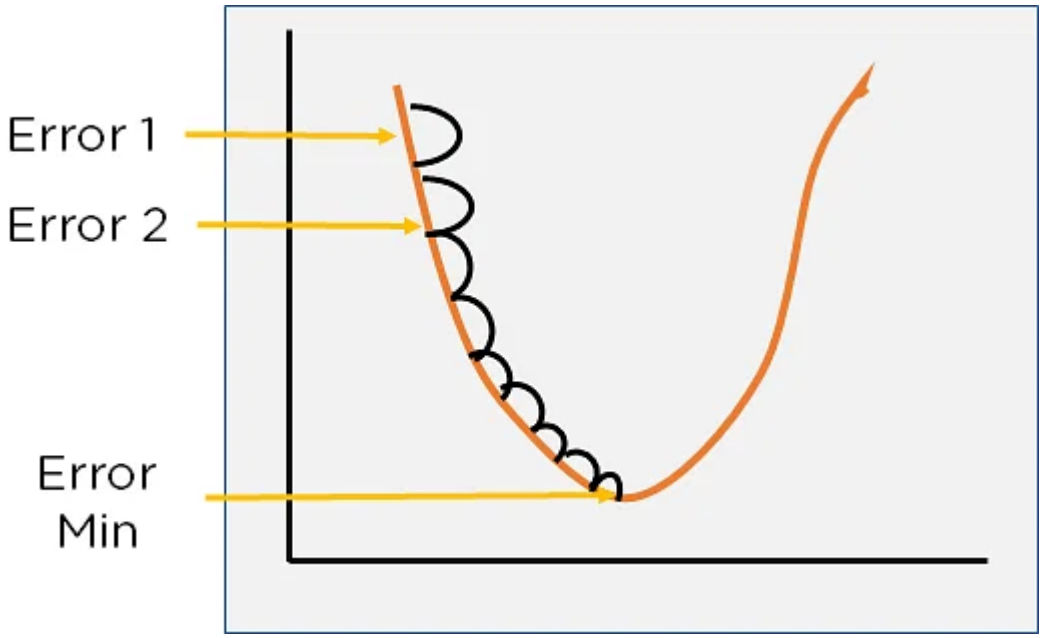
Gradient Descent

Gradient Descent is an algorithm that is used to optimize the cost function or the error of the model. It is used to find the minimum value of error possible in your model.

Gradient Descent can be thought of as the direction you have to take to reach the least possible error. The error in your model can be different at different points, and you have to find the quickest way to minimize it, to prevent resource wastage.

Gradient Descent can be visualized as a ball rolling down a hill. Here, the ball will roll to the lowest point on the hill. It can take this point as the point where the error is least as for any model, the error will be minimum at one point and will increase again after that.

In gradient descent, you find the error in your model for different values of input variables. This is repeated, and soon you see that the error values keep getting smaller and smaller. Soon you'll arrive at the values for variables when the error is the least, and the cost function is optimized.



Stocastic Gradient Descent

The word '*stochastic*' means a system or process linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, the problem arises when our dataset gets big.

Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima are reached. Hence, it becomes computationally very expensive to perform.
This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm used for optimizing machine learning models. In this variant, only one random training example is used to calculate the gradient and update the parameters at each iteration. Here are some of the advantages and disadvantages of using SGD:

Learning Rate

The weights of a neural network cannot be calculated using an analytical method. Instead, the weights must be discovered via an empirical optimization procedure called stochastic gradient descent.

Learning rate is a hyper-parameter that controls the weights of our neural network with respect to the loss gradient. It defines how quickly the neural network updates the concepts it has learned.

A desirable learning rate is low enough that the network converges to something useful, but high enough that it can be trained in a reasonable amount of time.

Smaller learning rates require more training epochs (requires more time to train) due to the smaller changes made to the weights in each update, whereas larger learning rates result in rapid changes and require fewer training epochs. However, larger learning rates often result in a sub-optimal final set of weights.

Batches, Epochs and Iteration

The batch size is a number of samples processed before the model is updated.

The number of epochs is the number of complete passes through the training dataset.

The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

The number of epochs can be set to an integer value between one and infinity. You can run the algorithm for as long as you like and even stop it using other criteria besides a fixed number of epochs, such as a change (or lack of change) in model error over time.

They are both integer values and they are both hyperparameters for the learning algorithm, e.g. parameters for the learning process, not internal model parameters found by the learning process.

You must specify the batch size and number of epochs for a learning algorithm.

There are no magic rules for how to configure these parameters. You must try different values and see what works best for your problem.

Deep Neural Network Tools

- Considering the impact of deep learning in the world, we decided to bring you 7 deep learning tools that you should know in 2021. Let's get started.
  1. **1. H2O.ai**
  2. **2. TensorFlow**
  3. **3. Keras**
  4. **4. Caffe**
  5. **5. DeepLearningKit**

6. Torch

7. Theano

Here were some of the best deep learning tools that you should know in 2021. Though this list is by no means exhaustive! There are more deep learning tools in the market but these are the ones that are used the most.These deep learning tools help solve real world problems and their knowledge can help you become job ready with appropriate AI skills.

Deep Neural Network overview

- ✔ A neural network is a system of hardware or software patterned after the operation of neurons in the human brain. Neural networks, also called artificial neural networks, are a means of achieving deep learning.
- ✔ The lines connected to the hidden layers are called weights, and they add up on the hidden layers. Each dot in the hidden layer processes the inputs, and it puts an output into the next hidden layer and, lastly, into the output layer.
- ✔ This neural network has the potential for high fault tolerance and can debug or diagnose a network on its own. ANN can go through thousands of log files from a company and sort them out. It is currently a tedious task done by administrators, but it will save a significant amount of time, energy, and resources if it can be automated.
- ✔ Neural networks will be a lot faster in the future, and neural network tools can get embedded in every design surface. Neural networks will also find their way into the fields of medicine, agriculture, physics, research, and anything else you can imagine.

Difference between neural network and deep neural network

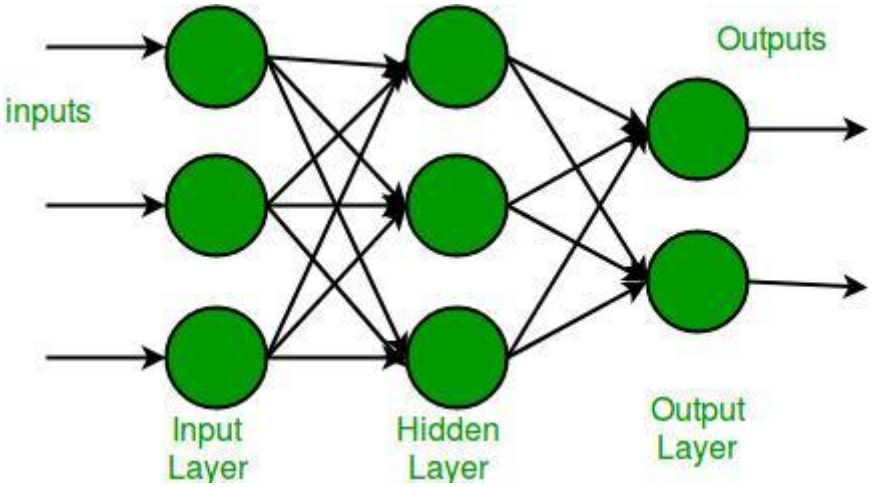
Since their inception in the late 1950s, **Artificial Intelligence** and **Machine Learning** have come a long way. These technologies have gotten quite complex and advanced in recent years. While technological advancements in the **Data Science** domain are commendable, they have resulted in a flood of terminologies that are beyond the understanding of the average person.

There are so many companies of all sizes out there that use these technologies viz. AI and ML in their day-to-day applications. Yet many have trouble distinguishing between their vast terminologies. Most people even use the terms **"Machine Learning", "Deep Learning"** and **"Artificial Intelligence"** interchangeably.

The reason behind this confusion is that although they have so many different names for different concepts – most of them are deeply entwined with one another and share similarities. Even so, each of these terminologies in itself is unique and useful in its way.

Now, let's talk about **Neural Networks** and **Deep Learning systems** individually before we can see their differences!

What is a Neural Network?



- Neural Networks are inspired by the most complex object in the universe – the human brain. Let us understand how the brain works first. The human brain is made up of something called Neurons. A neuron is the most basic computational unit of any neural network, including the brain.
- Neurons take input, process it, and pass it on to other neurons present in the multiple hidden layers of the network, till the processed output reaches the Output Layer.
- Neural networks are algorithms that can interpret sensory data via machine perception and label or group the raw data. They are designed to recognize numerical patterns contained in vectors that need to transform all real-world data (images, sounds, text, time series, etc.)
- In its simplest form, an Artificial Neural Network (ANN) has only three layers – the input layer, the output layer, and a hidden layer.

To know more about Neural Networks – Click here!

What is Deep Learning?

Now that we have talked about Neural Networks, let's talk about Deep Learning.

Deep learning, also known as **hierarchical learning**, is a subset of machine learning in artificial intelligence that can mimic the computing capabilities of the human brain and create patterns similar to those used by the brain for making decisions. In contrast to task-based algorithms, deep learning systems learn from data representations. It can learn from unstructured or unlabeled data.

Lab 7: Recurrent Neural Networks using Deep Learning

Source Code:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM#, CuDNNLSTM
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data() x_train = x_train/255.0
x_test = x_test/255.0
print(x_train.shape)
print(x_train[0].shape)
model = Sequential()
model.add(LSTM(128, input_shape=(x_train.shape[1:]), activation='relu', return_sequences=True))
model.add(Dropout(0.2))
```

```
model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)
model.compile(loss='sparse_categorical_crossentropy',optimizer=opt,metrics=['accuracy'])
model.fit(x_train,y_train,epochs=3,validation_data=(x_test, y_test))
```

Output:

```
(60000, 28, 28)
(28, 28)
Epoch 1/3
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_
  super(Adam, self).__init__(name, **kwargs)
1875/1875 [==============================] - 181s 96ms/step - loss: 0.5375 - accuracy: 0.8252 - val_loss: 0.1533 - val_accuracy: 0.9534
Epoch 2/3
1875/1875 [==============================] - 166s 88ms/step - loss: 0.1509 - accuracy: 0.9596 - val_loss: 0.0702 - val_accuracy: 0.9788
Epoch 3/3
1875/1875 [==============================] - 170s 91ms/step - loss: 0.1000 - accuracy: 0.9734 - val_loss: 0.0753 - val_accuracy: 0.9799
<keras.callbacks.History at 0x7f481e40c8e0>
```

Deep Learning Neural Network overview

Deep learning neural networks are a class of machine learning models that are inspired by the structure and function of the human brain. These models consist of multiple layers of interconnected nodes or neurons that process and transform input data to generate an output. ▪ Deep neural networks are called "deep" because they typically have many layers, which allows them to learn complex patterns and relationships in data. The input data is processed by the first layer, which passes the output to the second layer, and so on, until the final output is generated. ▪ The neurons in each layer of the network are connected to the neurons in the next layer by weights, which are adjusted during training to minimize the difference between the predicted output and the true output. This process is known as backpropagation, and it allows the network to learn from examples and improve its accuracy over time. ▪ The most widely used architectures in deep learning are feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs)

Deep Convolutional Neural Network

- ✔ Deep convolutional neural network has recently been applied to image classification with large image datasets. A deep CNN is able to learn basic filters automatically and combine them hierarchically to enable the description of latent concepts for pattern recognition.

- ✔ CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data.

- ✔ CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

- ✔ CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters through automated learning, whereas in traditional algorithms these filters are hand-engineered.

Improving accuracy of the neural networks

- ✔ The first thing that we can do to enhance a model **accuracy** is to add more data to train your model. Having more data is always a good idea. For instance, we do not get a choice to increase the size of training data because we haven't more data and we can't find more data from outside.

- ✔ Tuning Algorithm is about tune the parameters of machine learning algorithm to get the optimum value of each parameter. it will improve the accuracy of model to predict data For instance, in neural network we can tune parameter like hidden layer, activation function, epoch, optimizer, batch size, learning rate, Verbose, dropout, Cross Validation.

- ✔ To improve generalization on small noisy data, you can train multiple neural networks and average their output or you can also take a weighted average .If data is less complex and is having fewer dimensions or features then neural networks with 1 to 2 hidden layers would work.

- ✔ we can choose different neural network architectures and train them on different parts of the data and ensemble them and use their collective predictive power to get high accuracy on test data. Suppose, we are building a cats vs dogs classifier, 0-cat and 1-dog. When combining different cats vs dogs classifiers, the accuracy of the ensemble algorithm increases based on the Pearson Correlation between the individual classifiers.

The problem of Explainability

- Layerwise Relevance Interpretation and Error Detection
- Singular Vector Canonical Correlation Analysis
- Local interpretable Model-Agnostic Explanations

Layerwise Relevance Interpretation and Error Detection

- In applications involving object detection, the margin of error is low and it's important to analyze what the model is learning.
- For tasks such as brain tumor detection, urine sediment classification, and disease prediction, which require learning patterns in the images.
- Analyzing the feature maps of hidden layers can improve model performance and detect erroneous learning.
- The relevance scores and activations of previous layers can be back-propagated to identify important neurons with high relevance scores.

Singular Vector Canonical Correlation Analysis

- Singular Vector Canonical Correlation Analysis (SVCCA) involves creating an activation matrix from consecutive layers' output.

- Analyzing the SVCC score of a ConvNet trained on the CIFAR-16 dataset reveals that initial layers are trained first, followed by subsequent layers.
- The SVCC score between initial layers is high after training, and the SVCC between the last layers is also high after training completes.

Local interpretable Model-Agnostic Explanations

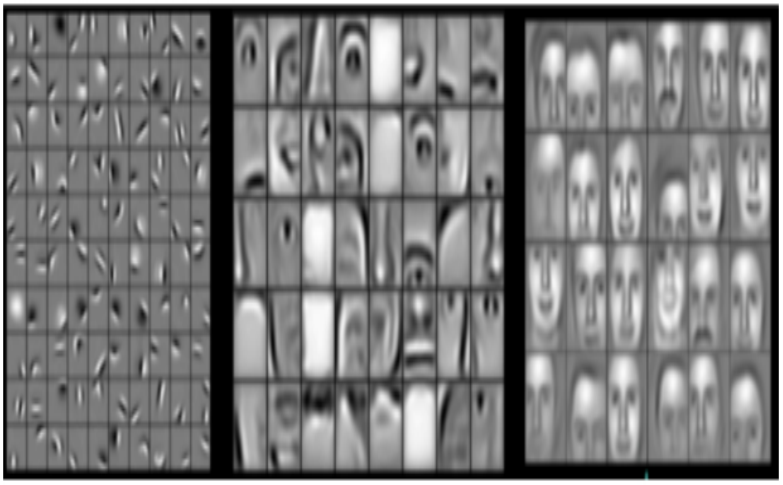
- Local Interpretable Model-Agnostic Explanations (LIME) is a library that includes an explainer for interpreting machine learning models.
- To use LIME, you input the training data, column names, and problem type (classification or regression).
- The interpreter uses the test data and trained model as parameters and provides output probability and individual feature explanations.
- LIME can also be used for images to interpret which parts of an image are responsible for predicting a particular class.

Interpretability of Neural Networks

- Neural networks are complex models that can be difficult to understand and interpret, particularly for deep networks.
- Interpretability is important in machine learning for various reasons, including ensuring model safety, detecting biases, and gaining insights into how models make predictions.
- Different techniques have been developed for interpreting neural networks, including saliency maps, feature visualization, and activation maximization.
- Local interpretability methods, such as LIME and SHAP, can provide explanations for individual predictions, while global interpretability methods, such as Principal Component Analysis (PCA) and t-SNE, can provide insights into the overall behavior of the model.
- Interpreting neural networks can help improve their performance and identify areas for improvement.
- Interpretability is particularly important in sensitive areas such as healthcare, finance, and law, where the consequences of incorrect predictions can be significant.

Learned Features

- CNNs learn abstract features from raw input images without requiring feature engineering.
- CNNs have multiple hidden layers, allowing them to learn many high-level features.
- The convolutional layers in a CNN transform the input images by learning complicated features.
- Feature visualization is a technique that helps visualize the learned features in a CNN.
- By visualizing the learned features, we can gain insights into what the CNN is learning and how it's making its predictions.
- The fully connected layers in a CNN provide the final outcome, based on the features learned by the convolutional layers.



Learned Features

Feature visualization

- Feature visualization is a technique used in machine learning and deep learning to understand how a neural network processes input data. It involves generating images or other visualizations that maximize the activation of a particular neuron or set of neurons in a neural network. By visualizing the features that activate certain neurons, researchers can gain insight into what patterns the neural network is recognizing in the input data.

- Feature visualization is often used to investigate the internal representations of a neural network, as well as to debug and improve the performance of the network. It can also be used to generate new examples of data that are similar to the training data but have unique features that the network has learned to recognize.

- There are many different techniques for feature visualization, including gradient ascent, which involves iteratively adjusting the input data to maximize the activation of a particular neuron or set of neurons, and activation maximization, which involves optimizing the input data to produce a particular output from the neural network.

Feature Visualization through Optimization

- Feature visualization through optimization is a popular technique used in machine learning and deep learning to generate visualizations that help researchers better understand how neural networks process input data. This technique involves iteratively optimizing an input image to maximize the activation of a particular neuron or set of neurons in a neural network.

- The basic idea behind optimization-based feature visualization is to start with a random input image and then iteratively adjust it to maximize the activation of a target neuron or set of neurons in the neural network.

- The optimization process typically involves computing the gradient of the target neuron(s) with respect to the input image and then updating the input image in the direction of this gradient.

- The optimization process can be performed using a variety of optimization algorithms, such as gradient descent, Adam, or L-BFGS. One common approach is to add a regularization term to the optimization objective function to prevent the generated image from being too noisy or unrealistic.

Lab 8: Normalizing & Creating Sequences using Deep Learning

Source Code:

```
import pandas as pd
import pathlib
DATADIR =  pathlib.Path('/content/BTC-2021min.csv')
df = pd.read_csv(DATADIR, names=['time', 'low', 'high', 'open', 'close', 'volume'])
print(df.head())
main_df = pd.DataFrame() # begin empty
ratios = ["BTC-USD", "LTC-USD", "BCH-USD", "ETH-USD"]  # the 4 ratios we want to consider
for ratio in ratios:  # begin iteration
    print(ratio)
    dataset = "/content/BTC-2021min.csv"  # get the full path to the file.
    df = pd.read_csv(dataset, names=['time', 'low', 'high', 'open', 'close', 'volume'])  # read in specific file
    # rename volume and close to include the ticker so we can still which close/volume is which:
    df.rename(columns={"close": f"{ratio}_close", "volume": f"{ratio}_volume"}, inplace=True)
    df.set_index("time", inplace=True)  # set time as index so we can join them on this shared time
    df = df[[f"{ratio}_close", f"{ratio}_volume"]]  # ignore the other columns besides price and volume
    if len(main_df)==0:  # if the dataframe is empty
        main_df = df  # then it's just the current df
    else:  # otherwise, join this data to the main one
        main_df = main_df.join(df)
main_df.fillna(method="ffill", inplace=True)  # if there are gaps in data, use previously known values
main_df.dropna(inplace=True)
print(main_df.head())  # how did we do??
SEQ_LEN = 60  # how long of a preceeding sequence to collect for RNN
FUTURE_PERIOD_PREDICT = 3  # how far into the future are we trying to predict?
RATIO_TO_PREDICT = "LTC-USD"
def classify(current, future):
    if float(future) > float(current):
        return 1
    else:
        return 0
main_df['future'] = main_df[f'{RATIO_TO_PREDICT}_close'].shift(-FUTURE_PERIOD_PREDICT)
```

```
main_df['target'] = list(map(classify, main_df[f'{RATIO_TO_PREDICT}_close'], main_df['future']))
print(main_df.head())
```

Output:

```
                                      open       close          volume
unix       date                symbol      close  Volume BTC      Volume USD
1646106180 2022-03-01 03:43:00 BTC/USD  43046.58  0.00000000             0.0
1646106060 2022-03-01 03:41:00 BTC/USD  43046.58  0.14297705  6154.673020989
1646106000 2022-03-01 03:40:00 BTC/USD  43016.03  0.00923000     397.0379569
1646105940 2022-03-01 03:39:00 BTC/USD  42999.44  0.82095000    35300.390268
BTC-USD
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: DtypeWarning: Columns (0
  exec(code_obj, self.user_global_ns, self.user_ns)
LTC-USD
BCH-USD
ETH-USD
```

Connection to Adversarial Example

- An adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction. I recommend reading the chapter about Counterfactual Explanations first, as the concepts are very similar. Adversarial examples are counterfactual examples with the aim to deceive the model, not interpret it.
- Why are we interested in adversarial examples? Are they not just curious by-products of machine learning models without practical relevance? The answer is a clear "no". Adversarial examples make machine learning models vulnerable to attacks, as in the following scenarios.
- A self-driving car crashes into another car because it ignores a stop sign. Someone had placed a picture over the sign, which looks like a stop sign with a little dirt for humans, but was designed to look like a parking prohibition sign for the sign recognition software of the car.
- A spam detector fails to classify an email as spam. The spam mail has been designed to resemble a normal email, but with the intention of cheating the recipient.
- A machine-learning powered scanner scans suitcases for weapons at the airport. A knife was developed to avoid detection by making the system think it is an umbrella.

Text and Tabular Data

- Text and tabular data are two types of data commonly used in machine learning and data analysis. While they have some similarities, they also have some key differences that require different approaches to handling and analyzing them.
- Text data consists of unstructured text, such as articles, reviews, tweets, and other forms of natural language. It can be challenging to analyze text data because it does not have a fixed structure or format. Common techniques used to analyze text data include natural language processing (NLP) and deep learning models such as recurrent neural networks (RNNs) and transformers.
- Tabular data, on the other hand, is structured data that is organized into rows and columns, such as spreadsheets or databases. Each column represents a variable, while each row represents an

observation or instance. Tabular data can be analyzed using traditional statistical and machine learning techniques, such as linear regression, decision trees, and random forests.

- When dealing with text data, it is common to perform text preprocessing tasks such as tokenization, stopword removal, stemming, and lemmatization to convert the text into a more structured format that can be analyzed using machine learning techniques. NLP techniques are also used to extract meaningful features from text data, such as sentiment, topic, and named entities.

- When working with tabular data, it is important to perform data cleaning and preprocessing tasks to ensure the data is accurate and ready for analysis. This may involve handling missing data, dealing with outliers, and transforming the data to a more appropriate format for analysis.

- In summary, text and tabular data are two common types of data used in machine learning and data analysis. While they have some similarities, they also require different approaches to handling and analyzing them due to their different structures and formats.

Network Dissection

- Network Dissection is a technique used in computer vision research to understand the inner workings of deep neural networks. It involves analyzing the activation patterns of individual neurons in a network to identify their semantic meaning, i.e., the concepts or objects they respond to. This can provide insights into how the network is processing visual information and making predictions.
- The process of network dissection typically involves presenting a set of images to the network and measuring the activation of each neuron in response to each image. The activations are then analyzed using various techniques to identify the objects, scenes, textures, or other visual concepts that each neuron responds to. This can be done by comparing the neuron activations to the activations of a set of known visual concepts, or by clustering the activations into groups that correspond to different concepts.
- One of the key benefits of network dissection is that it can help identify the specific features of an image that a network is using to make a prediction. For example, if a network is classifying images of cars, network dissection can help identify which parts of the car (e.g., wheels, windows, headlights) are most important for the network's decision.

Network Disection Algorithm

The Network Dissection algorithm involves several steps to analyze the semantic meaning of individual neurons in a deep neural network. The following is a general overview of the algorithm:

1. Prepare the dataset: Select a dataset of images that are representative of the kinds of images the network is designed to classify. These images should be labeled with the objects or scenes they contain.
2. Feed the images through the network: Pass each image in the dataset through the network and record the activation of each neuron in the network for each image.
3. Identify semantic concepts: Use a dataset of pre-defined semantic concepts to identify which neurons in the network respond to each concept. This is done by calculating the correlation between the activations of each neuron and the activations of each concept in the dataset.

4. Cluster neurons: Cluster the neurons that respond to the same concept together, to group them into "concept modules". The clustering can be done using various techniques, such as k-means clustering or hierarchical clustering.
5. Evaluate concept modules: Evaluate the concept modules to determine how well they correspond to the semantic concepts in the dataset. This can be done by examining the average activation of each module across the dataset and comparing it to the activation patterns for the corresponding concept.

Network Disection Experiments

- Object recognition in CNNs: This experiment analyzed the semantic meaning of neurons in a CNN designed for object recognition. The network was trained on the ImageNet dataset, which consists of images of objects such as animals, vehicles, and household items. The Network Dissection technique was used to identify which neurons in the network responded to different object categories in the dataset. The experiment found that the neurons in the network were able to learn to recognize and classify a wide variety of objects, and that the activation patterns of the neurons could be used to generate accurate object recognition predictions.
- Overall, these experiments demonstrate the utility of the Network Dissection technique for analyzing the semantic meaning of neurons in deep neural networks and understanding how they process visual information.
- Object recognition in CNNs: This experiment analyzed the semantic meaning of neurons in a CNN designed for object recognition. The network was trained on the ImageNet dataset, which consists of images of objects such as animals, vehicles, and household items. The Network Dissection technique was used to identify which neurons in the network responded to different object categories in the dataset. The experiment found that the neurons in the network were able to learn to recognize and classify a wide variety of objects, and that the activation patterns of the neurons could be used to generate accurate object recognition predictions.
- Overall, these experiments demonstrate the utility of the Network Dissection technique for analyzing the semantic meaning of neurons in deep neural networks and understanding how they process visual information.

Advantages of Feature Visualization

Data is a critical point for understanding and guiding decision-making processes. From personal to professional to financial data, it gives us the basic information to calculate our choices' potential risks and benefits.

- More Understandable Information
- Better Communication
- Using Data to Tell Stories
- Transparency and Performance Control
- Best Decisions-Making
- Process Automation and Time Savings
- Deep Connection with Customers
- Access to Real-Time Data
- Data Visualization is Interactive

Disadvantages of Feature visualization

1. **It gives assessment not exactness –**

While the information is exact in foreseeing the circumstances, the perception of similar just gives the assessment. It without a doubt is anything but difficult to change over the robust and protracted information into simple pictorial configuration yet such a portrayal of data may prompt theoretical ends now and then.

2. **One-sided –**

   The essential arrangement of information representation occurs with the human interface, which means the information that turns out to be the base of perception can be one-sided. The individual bringing the information for the equivalent may just think about the significant part of the information or the information that requirements center and may reject the remainder of the information which may prompt one-sided results.

3. **Absence of help –**

   One of the downsides of information perception is that it can't help, which means an alternate gathering of the crowd may decipher it in an unexpected way.

4. **Inappropriate plan issue –**

   On the off chance that information perception is viewed as such a correspondence. At that point, it must be certifiable in clarifying the reason. In the event that the plan isn't legitimate, at that point, this can prompt disarray in correspondence.

5. **Wrong engaged individuals can skip center messages –**

   One of the issues with information perception is however it could be logical its clearness in clarification is totally subject to the focal point of its crowd.

Activation Maximization

Activation Maximization is a method to visualize neural networks, and aims to maximize the activation of certain neurons. During normal training, one would iteratively tune the weights and biases of the network such that the error, or loss, of the neural network is minimized across training examples in the data set. On the other hand, activation maximization flips this around: after the classifier has been trained, we want to iteratively find the parts of the data that the model thinks belongs to a class.

For instance, consider a visualization of a neural network identifying handwritten digits from 0 to 9 (MNIST dataset): we want to see which parts of the image the neural network believes is important towards its decision in which digit it is; perhaps it is the bottom loop of the 8 or the hole of the 0.

A network's activation function output represents how confident it is that a training example belongs to one specific class, and so activation maximization constructs an image that checks off every single box the neural network is looking for and hence yields the largest activation function. This is done with gradient *ascent*, which tries to maximize the output neuron. The idea of activation maximization is really just finding the inputs that return an output with the highest confidence.

The results are really a quite enlightening vision into how the model makes decisions; dark regions represent 'penalizations' in that high values in that region make the model less sure the input is that digit, and bright values represent 'bonuses' in that high values in those regions increase the confidence of the output neuron. Activation maximization can also be visualized in the form of a distribution or another distribution representation for one-dimensional, non-image data.

Lab 9: MNIST Number Classification using Keras Library

Source Code:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
batch_size = 128
num_classes = 10
epochs = 1
img_rows, img_cols = 28, 28
(x_train, y_train), (x_test, y_test) = mnist.load_data()
if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
```

```
            metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output:

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
366/469 [======================>.......] - ETA: 37s - loss: 2.2793 - accuracy: 0.1243
```