We at The Data Monk hold the vision to make sure everyone in the IT industry has an equal stand to work in an open domain such as analytics. Analytics is one domain where there is no formal under-graduation degree and which is achievable to anyone and everyone in the World.

We are a team of 30+ mentors who have worked in various product-based companies in India and abroad, and we have come up with this idea to provide study materials directed to help you crack any analytics interview.

Every one of us has been interviewing for at least the last 6 to 8 years for different positions like Data Scientist, Data Analysts, Business Analysts, Product Analysts, Data Engineers, and other senior roles. We understand the gap between having good knowledge and converting an interview to a top product-based company.

Rest assured that if you follow our different mediums like our blog cum questions-answer portal www.TheDataMonk.com , our youtube channel - The Data Monk, and our e-books, then you will have a very strong candidature in whichever interview you participate in.

There are many blogs that provide free study materials or questions on different analytical tools and technologies, but we concentrate mostly on the questions which are asked in an interview. We have a set of 100+ books which are available both on Amazon and on The Data Monk e-shop page

We would recommend you to explore our website, youtube channel, and e-books to understand the type of questions covered in our articles. We went for the question-answer approach both on our website as well as our e-books just because we feel that the best way to go from beginner to advance level is by practicing a lot of questions on the topic.

We have launched a series of 50 e-books on our website on all the popular as well as niche topics. Our range of material ranges from SQL, Python, and Machine Learning algorithms to ANN, CNN, PCA, etc.

We are constantly working on our product and will keep on updating it. It is very necessary to go through all the questions present in this book.

Give a rating to the book on Amazon, do provide your feedback and if you want to help us grow then please subscribe to our Youtube channel.

LSTM

In this book we will be looking at the "Long Short Term Memory"(LSTM) cell, how it works and its implementation in python.

But before getting into it, let us have a brief overview of some of the pre-requisite concepts required to fully understand LSTMs:

Q1. What is machine learning?

A1. Machine learning is a domain where in a computer algorithm is formed by the computer itself as it learns on its own any specific task.

Q2. How does machine learning work?

A2. In machine learning, you feed some data to a computer algorithm and specify a certain task, and the algorithm uses the data to generate an algorithm that generalizes well on the given dataset and performing the given task.

Q3. What is Artificial Intelligence?

A3. Artificial Intelligence is simply put intelligence demonstrated by machines. It does not necessarily mean machine learning or deep learning, but any form that mimics human or animal intelligence.

Q4. What is deep learning?

A4. Deep learning is simply machine learning that mimics the human brain. It consists of advanced algorithms and neural networks whose goal is to imitate the thinking of a human brain.

Q5. What are some examples where machine learning is used?

A5. Some applications of machine learning include recommender systems, spam detection, image to text conversion etc.

Q6. What are some examples where deep learning is used?

A6. Some applications of deep learning include machine translation, language models, image recognition etc.

Q7. What are some of the algorithms used in machine learning?

A6. Some common machine learning algorithms are linear regression, logistic regression, support vector machines, decision trees, etc.

Q8. What are some of the algorithms used in deep learning?

A8. Some of the algorithms or network architectures used in deep learning are convolutional neural network, recurrent neural network, generative adversarial networks, etc.

Q9. What is regression?

A9. Regression is a method or a function that is used to find a relationship between two or more variables given from a dataset. Usually, you would have a dependent variable that is to be predicted using some independent variables.

For example, predicting the price of a house given its area, number of rooms, how old it is etc.

Q10. What are different types of regression?

A10. The 5 basic types of regression are:

1. Linear regression
2. Logistic regression
3. Ridge regression
4. Lasso regression
5. Polynomial regression

Q11. What is the relation between machine learning and deep learning?

A11. Deep learning can be thought of as a subset of machine learning. At it's core, the basic equations of a neural network are the same as that of machine learning.

Neural Networks

Q12. What is a neural network?

A12. A neural network is an advanced machine learning algorithm where in you group a bunch of neurons in a layer have them perform tasks like classification, regression etc.

Q13. What is the core logic of a neural network?

A13. A neural network can be thought of as many polynomial regression units working together to solve a task

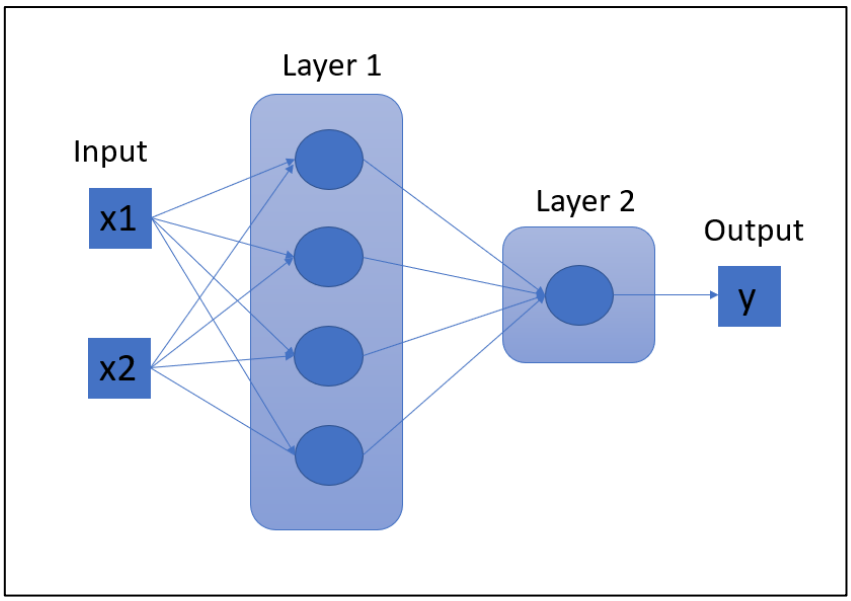
Q14. Why is a neural network called so?

A14. A neural network attempts to replicate the process of the brain cell which is why it's called a neural network.

Q15. How does a neural network work?

A15. Suppose you have a simple classification task given two input variables x1 and x2, and you have the prediction value y that can be 0 or 1.

So, you can have a simple neural network with 2 layers that will take x1 and x2 as input, forward propagate those values through the network, and then output the prediction value y.



Q16. How is forward propagation carried out in a neural network?

A16. Let us take the problem statement considered in the previous question. Let us say that our 2-layer network the first layer has 4 neurons and the second layer has 1 neuron.

Note that the first layer can have any number of nodes regardless of the input size but your last layer needs to be configured to have the same shape as your output shape. Here since our output is a single number, our last layer will have just one neuron.

Now first, the inputs will be passed to the first layer. Since there are 2 variables the input size is 2. And our first layer has 4 neurons.

A simple formula to calculate the weight matrix shape here is:

$$\boldsymbol{Wi = [Ni \times Ni - 1]}$$

where W is the weight matrix and N are the number of nodes in the i^{th} layer.

So here, the first layer's weight matrix will have the shape [4 × 2], and the final layer's weight matrix will have the shape [1 × 4].

So first, the input is passed into the first layer where the matrix multiplication takes place as follows:

$$W_1 \text{ (shape } [4 \times 2]) * X \text{ (shape } [2 \times 1]) + \text{bias} = Z_1 \text{ (shape } [4 \times 1])$$

$$A_1 = g(Z_1)$$

Where g is some activation function.

Then this A_1 value is passed as input into the second layer:

$$W_2 \text{ (shape}[1 \times 4]) * A_1 \text{ (shape } [4 \times 1]) + \text{bias} = Z_2 \text{ (shape}[1 \times 1])$$

$$A_2 = g(Z_2)$$

Where g is some activation, here it can be the sigmoid function as we want the output value to be between 0 and 1.

Q17. What are linear functions?

A17. A linear function is a polynomial function of degree one. For a function with one independent variable, that would be a straight line.

Q18. What are non linear functions?

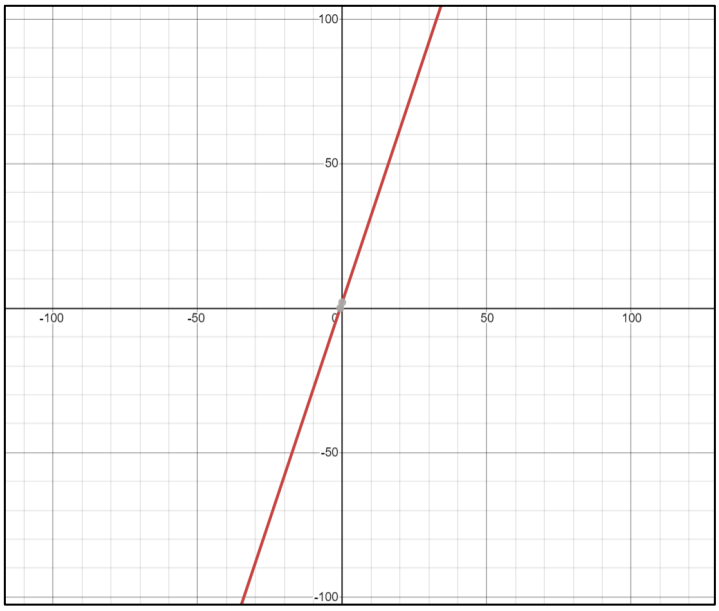
A18. A non linear function is simply any function that is not a linear function, or any function that is not a straight line. Example: Quadratic functions.

Q19. What is the primary difference between a linear function and non linear function?

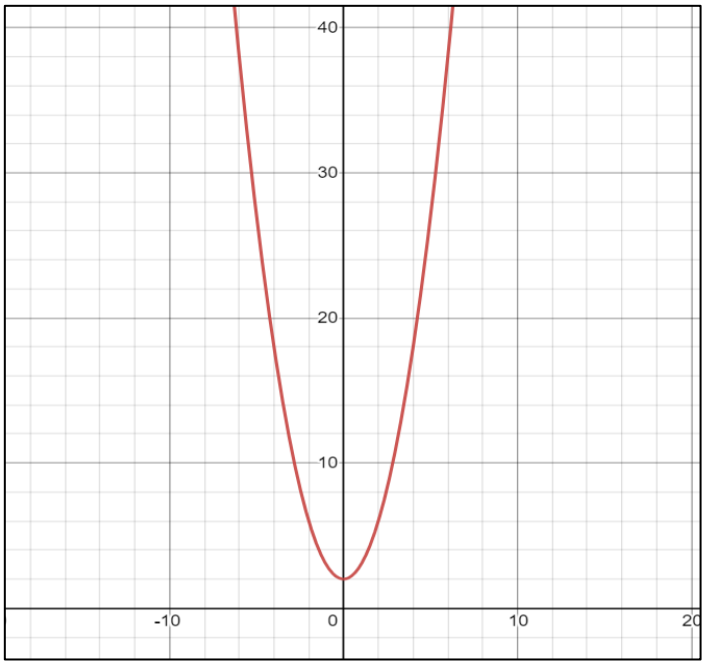
A19. The main difference between a linear and non linear function is that a linear function has a constant gradient or slope while a non linear function does not have a constant gradient or slope. What that means basically is that the rate of change of a graph for a linear function is same everywhere, while for a non linear function it keeps changing.

Example:

A linear function: y = 3*x + 2



A non linear function: $y = x^2 + 2$



Q20. What are activation functions and why are they used in neural networks?

A20. Activation functions are differentiable non-linear functions to bring the output values of the layers in the network in a specific range, but more importantly to break the linearity of the network.

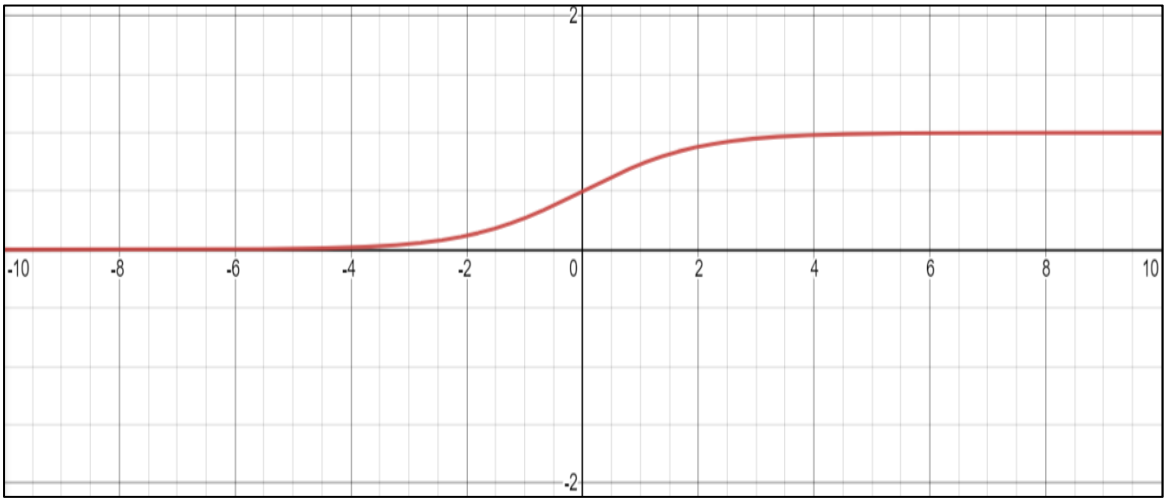
Q21. What would happen to a network without any activation function?

A21. Without activation functions, even a huge neural network will end up being the same as a single node neural network. Also, there might be a very good chance of the values blowing up exponentially due to the constant scaling up by the multiplication of the values with the weights.

Q22. What are some of the activation functions that are used in neural networks?

A22. Here are some of the activation functions used in neural networks:

1. Sigmoid function –



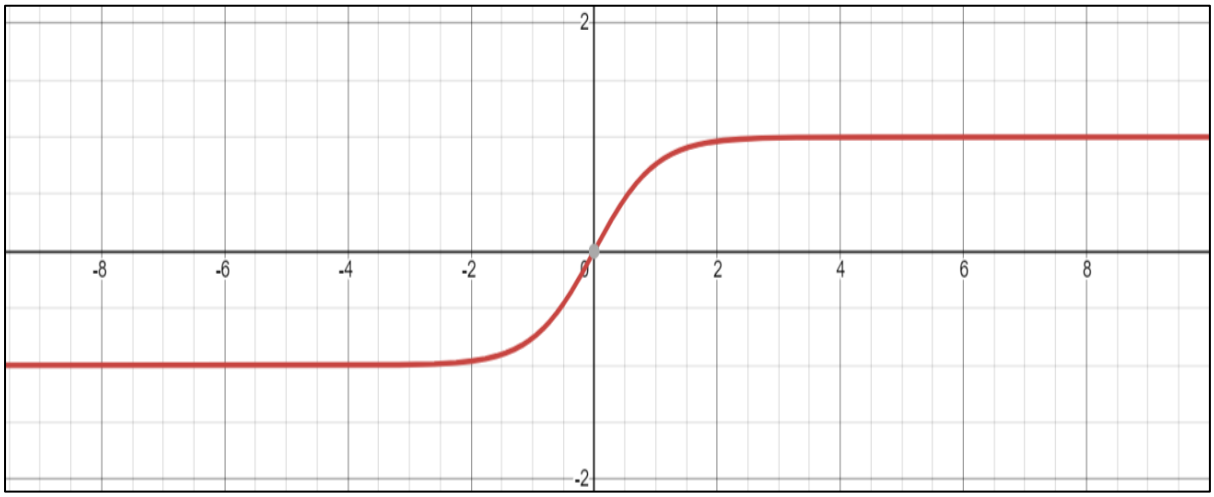
$$S(x) = \frac{1}{1 + e^{-x}}$$

This function outputs values in the range 0 to 1.

2. Tanh function –

$$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

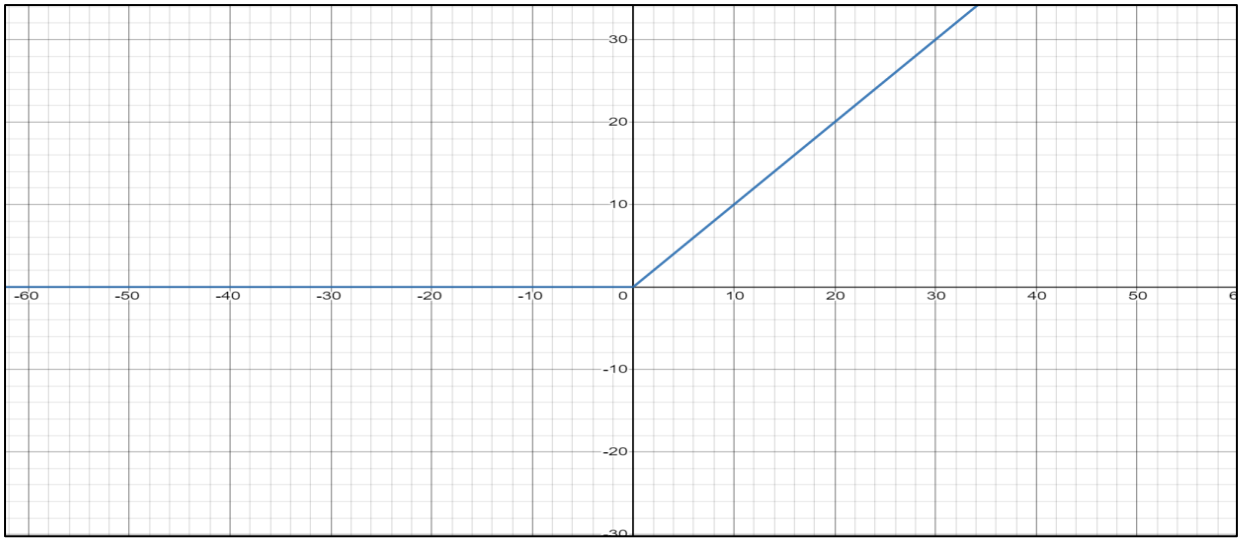
This function outputs values in the range -1 to 1.



3. ReLU function –

$$max(0,x)$$

This function outputs values in the range 0 to x where x is the input value if positive.



4. Softmax function:

$$S(x) = \frac{e^x}{\sum_0^n e^{x_i}}$$

This function is usually used in the output layer of multi-class classification problems. This function outputs a vector in the form of relative probabilities. It basically takes an output vector and scales it to the probability of each being true. The summation of all the values in the output vector formed a softmax layer will always be 1.

Q23. How is a neural network trained?

A23. At the end of forward propagation, we compare the predicted value by the model to the actual real value, and calculate the error using a loss function. We use this error or loss value to calculate gradients and perform backpropagation to minimize the loss function and maximize accuracy of predictions.

Q24. What is a loss function?

A24. A loss function is a differentiable function that compares the value of the predicted value and the real value and calculates the loss value. The objective is to minimize the value of the loss function so that the predictions performed are closer to the actual values.

Q25. What are the different kinds of loss functions?

A25. There are a different loss functions that can be used for different kinds of problems:

1. Mean Squared Error (MSE):
   This is one of the simplest and most common loss functions used. This function simply takes the distance of the target points from the actual

points and takes the mean of the squares of their summation. The squaring is to remove the negative distances.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y'_i)^2$$

Where n is the number of points, Y is the actual value and Y' is the predicted value.

2. Cross Entropy Loss (Log loss):
   This is another one of the most commonly used loss functions. This function is generally used in multi-class classification problems. This function heavily penalizes predictions that are very wrong.

$$CE = \sum_{c=1}^{N}(-Y\log(Y') + (1-Y)\log(1-Y'))$$

Where N is the number of classes, Y is the actual value and Y' is the predicted value.

CODE

In this article, we'll be developing 3 models in total, a vanilla neural network, a recurrent neural network, and a LSTM model. We'll be testing all 3 on the same problem statement: An NLP task where in the input is a string of length 5 for the addition of two numbers, and the output being a string of length 5 that is the sum.

Example: Input: "67+60"

Output: "00127"

Let's get to it!

Dataset Creation Code

First, let's import the required libraries:

```
import numpy as np
```

Next, let's create dictionaries of our vocabulary that will be needed for the creation of the dataset:

```
all_chars = '0123456789+'
char_to_index = dict((c, i) for i, c in enumerate(all_chars))
index_to_char = dict((i, c) for i, c in enumerate(all_chars))
```

Let's create a function now to generate examples:

```
def generate_data():
    first_num = np.random.randint(low=0,high=100)
    second_num = np.random.randint(low=0,high=100)
    example = str(first_num) + '+' + str(second_num)
    label = str(first_num+second_num)
```

```
    return example, label
```

Now, when we have to give this text as an input to the network, we can't simply feed in the text. We have to perform what is known as "One-hot encoding" here where we convert the input into one-hot vectors as a better form of input:

```
def vectorize_example(example, label):
    x = np.zeros((5, 11))
    y = np.zeros((5, 11))

    diff_x = 5 - len(example)
    diff_y = 5 - len(label)

    for i, c in enumerate(example):
        x[diff_x+i, char_to_index[c]] = 1
    for i in range(diff_x):
        x[i, char_to_index['0']] = 1
    for i, c in enumerate(label):
        y[diff_y+i, char_to_index[c]] = 1
    for i in range(diff_y):
        y[i, char_to_index['0']] = 1

    return x, y
```

Let's also create a utility function to convert the one-hot vectors back into normal form for later:

```
def devectorize_example(example):
    result = [index_to_char[np.argmax(vec)] for i, vec in enumerate(example)]
    return ''.join(result)
```

Let’s create our dataset now!

```
def create_dataset(num_examples=20000):

    x_train = np.zeros((num_examples, 5, 11))
    y_train = np.zeros((num_examples, 5, 11))

    for i in range(num_examples):
        e, l = generate_data()
        x, y = vectorize_example(e, l)
        x_train[i] = x
        y_train[i] = y

    return x_train, y_train

x_train, y_train = create_dataset(20000)
```

Our dataset is ready!

Now let’s create our first model

Vanilla neural network code

Let's import the required libraries:

```
import tensorflow as tf


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Let's create a simple model:

```
model_ann = Sequential([
    Dense(50, activation='relu', input_shape=(None,11)),
    Dense(100, activation='relu'),
    Dense(11)
])


model_ann.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

This is what the model summary looks like:

```
model_ann.summary()
```

Output:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, None, 50)          600
_________________________________________________________________
dense_10 (Dense)             (None, None, 100)         5100
_________________________________________________________________
dense_11 (Dense)             (None, None, 11)          1111
=================================================================
Total params: 6,811
Trainable params: 6,811
Non-trainable params: 0
```

Let’s train the model for a 100 epochs:

```
model_ann.fit(x_train, y_train, epochs=100, validation_split=0.2,
batch_size=1024)
```

Let’s test the model accuracy:

```
x_test, y_test = create_dataset(num_examples=20)

preds = model_ann.predict(x_test)

full_seq_acc = 0


for i, pred in enumerate(preds):

    pred_str = devectorize_example(pred)

    y_test_str = devectorize_example(y_test[i])

    x_test_str = devectorize_example(x_test[i])

    full_seq_acc += 1/len(preds) * int(pred_str == y_test_str)

    outstring = 'Input: {}, Out: {}, Pred: {}'.format(x_test_str, y_test_str,
pred_str)

    print(outstring)

print('\nFull sequence accuracy: {:.3f} %'.format(100 * full_seq_acc))
```

Output:

```
Input: 39+55, Out: 00094, Pred: 60000
Input: 09+10, Out: 00019, Pred: 00050
Input: 054+4, Out: 00058, Pred: 00000
Input: 85+24, Out: 00109, Pred: 00010
Input: 97+25, Out: 00122, Pred: 00010
Input: 37+63, Out: 00100, Pred: 60056
Input: 25+78, Out: 00103, Pred: 10000
Input: 47+86, Out: 00133, Pred: 00005
Input: 15+46, Out: 00061, Pred: 50005
Input: 43+39, Out: 00082, Pred: 06060
Input: 69+90, Out: 00159, Pred: 50000
Input: 18+33, Out: 00051, Pred: 50066
Input: 80+14, Out: 00094, Pred: 00050
Input: 22+82, Out: 00104, Pred: 11001
Input: 07+72, Out: 00079, Pred: 00001
Input: 12+63, Out: 00075, Pred: 51056
Input: 25+60, Out: 00085, Pred: 10050
Input: 36+88, Out: 00124, Pred: 65000
Input: 86+50, Out: 00136, Pred: 05000
Input: 81+23, Out: 00104, Pred: 05016

Full sequence accuracy: 0.000 %
```

Perhaps with some tweaking to the model architecture, there could have been one or two correct predictions, but as expected, this model performs horribly when it comes to sequence models.

Recurrent Neural Networks

Q26. What are sequence models?

A26. Sequence models are models that work on sequences of data instead of discrete data. That is, the input for these models will be in the form of a time sequence. Example: Language models, stock market prices, speech recognition systems etc.

Q27. Why can't we use normal neural networks for sequence modelling?

A27. Normal neural networks cannot be used for sequence modelling as they cannot capture sequence dependencies. A normal neural network will treat each time step in an input sequence as a different input rather than consider the whole as one.

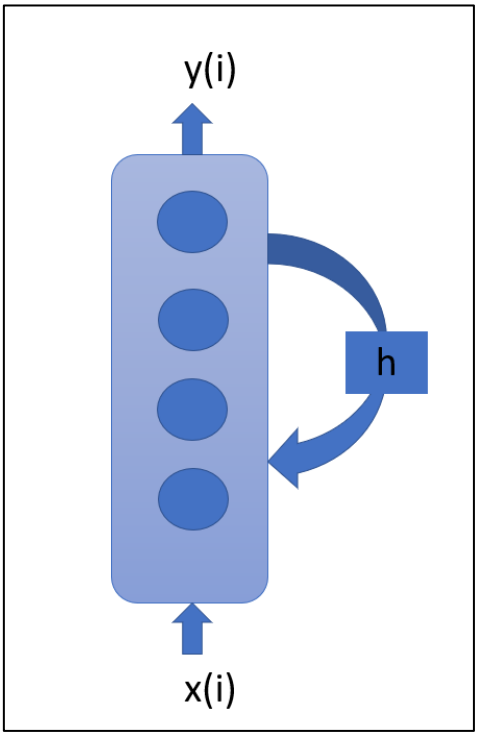
Q28. What are the problems with using normal neural networks for sequence modelling?

A28. There are mainly 3 problems with using normal neural networks for sequence modelling:

1. Parameter sharing – A feedforward neural will use different trained weights in every layer. Suppose you have a sequence of text as "The cat ate the mouse" and another one as "The mouse was eaten by the cat".
   As you can see, both of these texts say the same thing both have the same meaning. But a normal neural network will use different weights for "the cat" in the first and second equation, which will result in it giving different outputs.
2. Variable sequence length – A normal neural network has a fixed output shape. For tasks like machine translation where we have to translate a sequence of words from one language to another, there can be different lengths of output for different inputs. Normal neural nets cannot solve this problem.
3. Huge parameters – For NLP tasks where we use word embeddings, the input dimensions are extremely big in size, often in the order of 10^4-10^5. From the formula in the previous section, you can imagine that the weight matrices will be of even greater size with such huge inputs.

Q29. What is a Recurrent Neural Network?

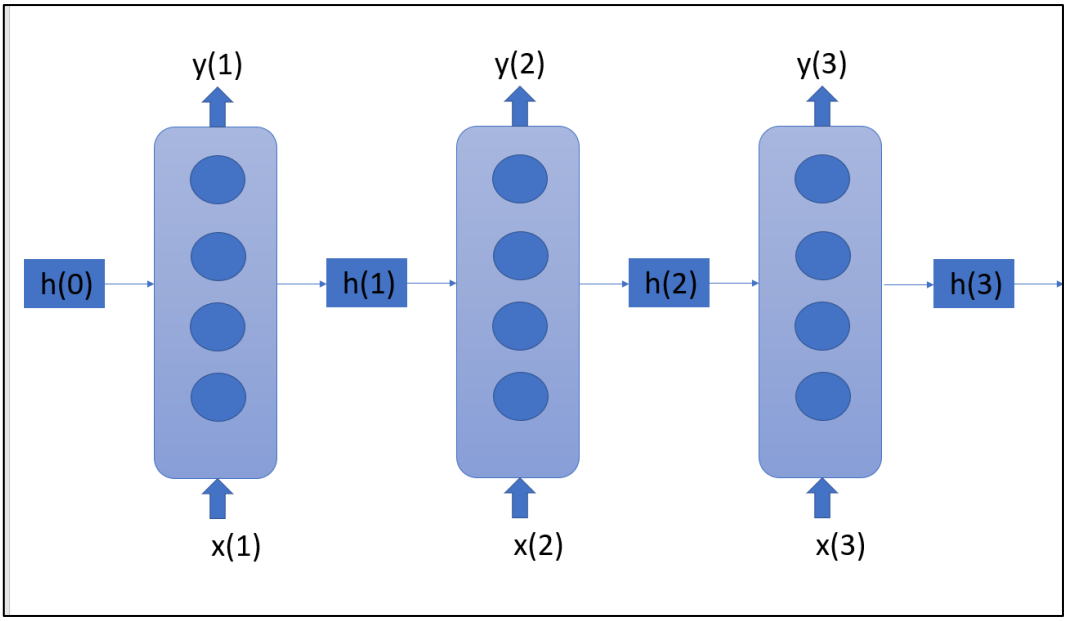
A29. A recurrent neural network is a type of neural network that is used for sequence models. It uses a type of architecture where in it considers not just the input at the current time step but also the hidden state of the previous time step to calculate the output at the current time step. As the name suggests, it does so by cycling through the same weight matrix. Let's look at this in more detail in the next question.



Q30. How does a Recurrent Neural Network work?

A30. Let us define some notation for this example to help you better understand the working of a RNN:

1. Let x be the input where x_i denotes the i^{th} time step of the input sequence.
2. Let y be the output sequence where y_i denotes the i^{th} time step of the output sequence.
3. Let h be the hidden state where h_i denotes the hidden state at the i^{th} time step.



You can keep referring to the diagram given above during the explanation to have a better understanding of the notations and the working.

So, to start with, the first time step of the input sequence x_1 is fed in to the RNN, along with the hidden state h_0 that can be initialized randomly or as an array of zero.

Here, like a normal neural net the input x_1 is multiplied with the weight matrix W_x, and the hidden state h_0 with weight matrix W_h to output the hidden state h_1:

$$h_1 = g(W_x * x_1 + W_h * h_0 + bias)$$

where g is any activation function.

Then the output at that time step y_1 can be calculated by using another set of weights W_y as follows:

$$y_1 = g(W_y * h_1 + bias)$$

where g is any activation function.

Then for the next time step, we will repeat the same process but by considering the hidden state and input of the next time steps. This is what the equations will look like:

$$h_2 = g(W_x * x_2 + W_h * h_1 + bias)$$

$$y_2 = g(W_y * h_2 + bias)$$

where g is any activation function.

Note: The weight matrices W_x, W_h and W_y remain the same during all the recurrences. It's the same weights that are shared at each iteration.

And this process keeps on recurring until the end of the sequence. So we can derive a generalized formula for the i^{th} time step as:

$$h_i = g(W_x * x_i + W_h * h_{i-1} + bias)$$

$$y_i = g(W_y * h_i + bias)$$

where g is any activation function.

Note: Most of the times, for simplicity, in the above equation, instead of writing the multiplication of 'x' and 'h' with their weights separately, it's written in this way:

$$h_i = g(W_h * [x_i, h_{i-1}] + bias)$$

where W_h is simply a new matrix formed by stacking the matrices W_h and W_x horizontally.

Q31. What are the advantages of RNN over normal Neural Nets?

A31. As discussed above, normal neural nets had 3 primary issues when it came to sequence modelling, all 3 of which are solved by recurrent neural networks:

1. Parameter Sharing: As you saw, the same weight matrices are used throughout the network. So, the same weights will be applied to the input throughout the sequence. This way, any input will have a similar effect on the output as long as it's in the sequence. The input is treated as a sequence rather than discrete points.
2. Variable length: We can choose any length for the output sequence in recurrent neural networks.
3. Huge parameters: Since we are using the same shared weight matrix, there are no other parameters needed in recurrent neural networks.

Q32. Give examples of variable lengths RNNs.

A32. There are a number of applications for different types of RNNs:

1. Many to one: For tasks like sentiment analysis, the input sequence is a sequence of text, and the output is a single number classifying the sentiment of the text.
2. One to many: For tasks like text generation or music generation, a single input is given as a starting element or the class to generate and using that single value a sequence of output is produced.
3. Many to many: For tasks like Machine Translation or Named Entity Recognition(NER), the many to many architecture is used as the input and output both are sequences of variable length.

Q33. What are the disadvantages of RNNs?

A33. RNNs sometimes suffer from the problem of vanishing gradients when it comes to very long sequences. This means that the network is unable to create long range dependencies in such cases.

Q34. What are vanishing gradients?

A34. Vanishing gradients simply mean that during back propagation the gradients begin diminishing to an extremely small scale. This usually happens in very deep networks, where the gradients from the final layers gradually vanish till they reach the early layers. Because of this, the values in the final layers don't really influence the weights of the early layers. So the network is unable to create long range dependencies, that is the network cannot remember values far back in a long sequence of input.

Q35. What are exploding gradients?

A35. Exploding gradients are similar to vanishing gradients, but instead of the gradients becoming extremely small here the gradients keep on growing throughout the network, blowing up the weights to extreme proportions. This is a relatively easier problem to tackle than vanishing gradients, by simply clipping the gradients.

RECURRENT NEURAL NETWORK CODE

Let's create a RNN model now!

First, let's import the required libraries:

```
import tensorflow as tf


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, RepeatVector
```

Let's create a simple RNN model for our previous dataset now:

```
model_rnn=Sequential([
    SimpleRNN(128, input_shape=(None, 11)),
    RepeatVector(5),
    SimpleRNN(128, return_sequences=True),
    Dense(11, activation='softmax')
])


model_rnn.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Note: The RepeatVector layer here is used to adjust the output shape. As you can see below in the model summary, the first rnn layer does not output the sequence length, but by using the RepeatVector layer you can make it output a sequence. It simply repeats the input 'n' times, which is 5 times in this case to output a sequence of the same length

This is what the model summary looks like:

```
model_rnn.summary()
```

Output:

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_10 (SimpleRNN)    (None, 128)               17920
_________________________________________________________________
repeat_vector_7 (RepeatVecto (None, 5, 128)            0
_________________________________________________________________
simple_rnn_11 (SimpleRNN)    (None, 5, 128)            32896
_________________________________________________________________
dense_20 (Dense)             (None, 5, 11)             1419
=================================================================
Total params: 52,235
Trainable params: 52,235
Non-trainable params: 0
_________________________________________________________________
```

Let's train this model for 50 epochs now:

```
model_rnn.fit(x_train, y_train, epochs=50, validation_split=0.2, batch_size=1024)
```

Let's test the model accuracy on some test examples!

```
x_test, y_test = create_dataset(num_examples=20)
preds = model_rnn.predict(x_test)
full_seq_acc = 0

for i, pred in enumerate(preds):
 pred_str = devectorize_example(pred)
  y_test_str = devectorize_example(y_test[i])
 x_test_str = devectorize_example(x_test[i])
   full_seq_acc += 1/len(preds) * int(pred_str == y_test_str)
outstring = 'Input: {}, Out: {}, Pred: {}'.format(x_test_str, y_test_str, pred_str)
print(outstring)
print('\nFull sequence accuracy: {:.3f} %'.format(100 * full_seq_acc))
```

Output:

```
Input: 028+0, Out: 00028, Pred: 00028
Input: 52+17, Out: 00069, Pred: 00069
Input: 91+80, Out: 00171, Pred: 00171
Input: 56+38, Out: 00094, Pred: 00094
Input: 78+35, Out: 00113, Pred: 00113
Input: 059+2, Out: 00061, Pred: 00061
Input: 68+53, Out: 00121, Pred: 00121
Input: 24+19, Out: 00043, Pred: 00043
Input: 86+94, Out: 00180, Pred: 00180
Input: 086+7, Out: 00093, Pred: 00093
Input: 27+58, Out: 00085, Pred: 00085
Input: 29+15, Out: 00044, Pred: 00044
Input: 10+71, Out: 00081, Pred: 00081
Input: 09+85, Out: 00094, Pred: 00094
Input: 58+78, Out: 00136, Pred: 00136
Input: 067+1, Out: 00068, Pred: 00068
Input: 06+13, Out: 00019, Pred: 00029
Input: 70+33, Out: 00103, Pred: 00103
Input: 66+60, Out: 00126, Pred: 00126
Input: 60+46, Out: 00106, Pred: 00106

Full sequence accuracy: 95.000 %
```

As you can see, in just 50 epochs the model is performing really well with an amazing 95% accuracy!

Long Short Term Memory Cell

Q36. What is LSTM short for?

A36. LSTM stands for "Long Short Term Memory". It is an advanced architecture of the RNN model.

Q37. How are LSTMs different than RNNs?

A37. LSTMs have a similar root concept like that on RNNs, with some advanced modifications.

1. Cell variables: In LSTM, along with the hidden state 'h', there is a new cell variable 'c' that we'll call as the cell state.
2. Instead of a single dense layer like in RNN, a LSTM has 3 different gates that are used for calculating and updating the cell values.
3. In RNNs each cell had 2 inputs: the hidden state of the previous cell 'h' and the input of the current time step 'x'.
   In LSTMs, each cell has 3 inputs: the hidden state of the previous cell 'h', the input of the current time step 'x', and an additional cell state of the previous cell 'c'.

Q38. What is the significance of the cell state variable 'c'?

A38. The cell state variable 'c' is used in LSTMs to store long term dependencies. In a way, it is used to decide what prior part of the sequence was relevant and what was not. It serves like a memory variable for the network.

Q39. What are gates in LSTM?

A39. Gates are used in LSTMs to decide what information is to be forgotten and what information is to be forgotten. They are an essential part of the LSTM architecture as they are the ones who update the value of the cell state and the hidden state.

Q40. What activation function is used by the gates in LSTM?

A40. The gates in LSTM use the sigmoid function.

Q41. Why is sigmoid function used in gates?

A41. Sigmoid function outputs value in the range of 0 to 1. The gates in LSTM are used to determine what information is relevant and what is not. It can be thought of as real gates, they decide what goes through and what doesn't. Similarly, here a value of 0 means the gate will not let that information affect the current state at all, while a value of 1 means that information is very relevant to the current cell state.

Q42. How many types of gates does a LSTM have?

A42. The LSTM cell has 3 gates:

1. Forget Gate:
   This gate can be thought of as the gate to eliminate irrelevant information from the cell state.

2. Input Gate:
   This gate can be thought of as the gate that is used to update the new relevant information to the cell state from the given current input.

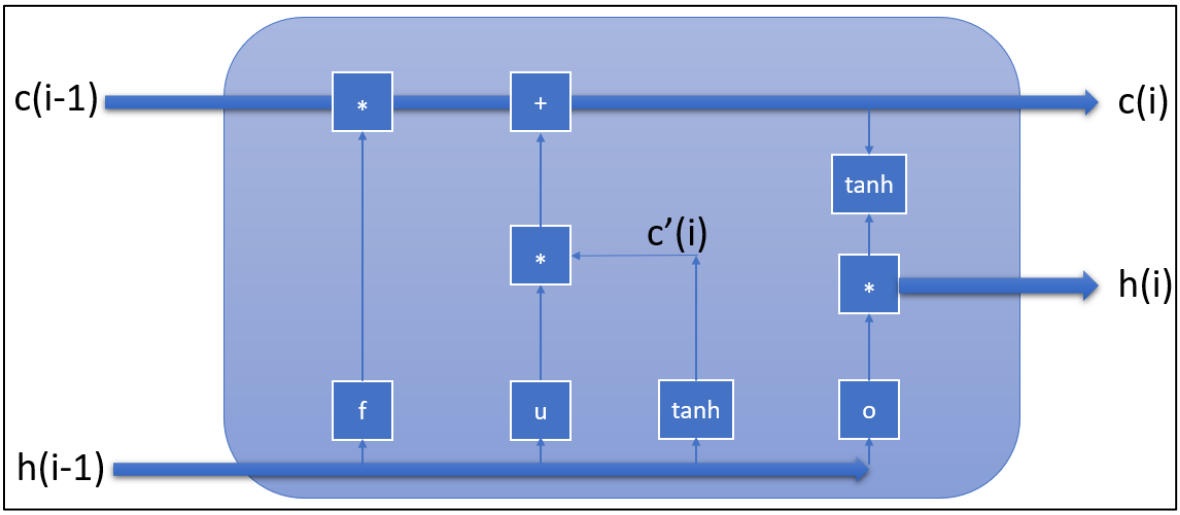
3. Output Gate:
   This gate is used to output the hidden state of the current cell.

Q43. How does a LSTM cell work?

A43. Let's define some notation:

1. Let x be the input sequence where x_i denotes the i^{th} time step of the input sequence.
2. Let y be the output sequence where y_i denotes the i^{th} time step of the output sequence.
3. Let h be the hidden state where h_i denotes the hidden state at the i^{th} time step.
4. Let c be the hidden state where c_i denotes the hidden state at the i^{th} time step.

5. Let c' be the temporary hidden state where c'_i denotes the temporary hidden state at the i^{th} time step.
6. Let f be the value of the forget gate where f_i denotes the value of the forget gate at the i^{th} time step.
7. Let u be the value of the update gate where u_i denotes the value of the update gate at the i^{th} time step.
8. Let o be the value of the output gate where o_i denotes the value of the output gate at the i^{th} time step.



Working:

1. First, a temporary cell state is calculated for the current cell using the hidden state of the previous time step and the input of the current time step:

$$c'_i = tanh(W_c * [h_{i-1}, x_i] + bias)$$

2. Then the value of the 3 gates are calculated for the current cell using the hidden state of the previous time step and the input of the current time step:

$$f_i = sigmoid(W_f * [h_{i-1}, x_i] + bias)$$

$$u_i = sigmoid(W_u * [h_{i-1}, x_i] + bias)$$

$$o_i = sigmoid(W_o * [h_{i-1}, x_i] + bias)$$

3. The forget gate and update gate are used to then update the value of the current cell state ‘c’ from the previous time step’s cell state and the current cell’s temporary cell state:

$$c_i = u_i * c'_i + f_i * c_{i-1}$$

   As you can see, here the update gate determines what relevant information to update from the current cell, and the forget gate determines what information is to be kept from the previous cell state.

4. Finally, the output gate is used to calculate the final hidden state output of the current cell:

$$a_i = o_i * tanh(c_i)$$

5. These values are then passed along with the next time step input to the next cell where the process reiterates again like a RNN.

6. Note: Similar to the RNN here also the weight matrices used are shared in all recursions and the same weights are used and applied for the inputs of all the time steps.

Q44. What is a peephole connection in LSTM?

A44. Peephole connections in LSTMs are slight variations where the gates are given 3 inputs instead of the usual 2, along with the previous hidden state and current time step input, they are also given the previous cell state as an input:

$$f_i = sigmoid(W_f * [c_{i-1}, h_{i-1}, x_i] + bias)$$

$$u_i = sigmoid(W_u *[c_{i-1}, h_{i-1}, x_i] + bias)$$

$$o_i = sigmoid(W_o *[c_{i-1}, h_{i-1}, x_i] + bias)$$

Q45. What are GRUs?
A45. GRUs are a simplified version of the LSTM where we use only one gate instead of the forget and update gate.

Q46. What does GRU stand for?
A46. GRU is short for "Gated Recurrent Units".

Q47. How does a GRU work?
A47. A GRU works exactly like a LSTM, just instead of the update and forget gates, we have a single update gate:

$$u_i = sigmoid(W_u *[h_{i-1}, x_i] + bias)$$

And this is how we update the cell state value in GRUs:

$$c_i = u_i *c'_i + (1-u_i) * c_{i-1}$$

Also in GRUs, the hidden state is equal to the cell state:

$$c_i = h_i$$

Q48. What are the advantages and disadvantages of GRU?
A48.
Advantages of GRU are:
1. GRUs are less complex than LSTMs.
2. GRUs use less space than LSTMs.

Disadvantages of GRUs are:

1. Less control over the network with just one gate.
2. Hard to capture complex and long-term dependencies.
3. LSTMs generally perform better due to the advanced architecture.

Q49. Are LSTMs better or GRUs?

A49. There is no clear answer to the question of whether LSTMs are better or GRUs are better. Depending on the problem statement or the application used, the answer will differ. Although generally LSTM should have a better accuracy, sometimes it's better to use GRUs because they take up a smaller number of resources and are efficient. Usually in cases where your data is small and not too complex, both GRU and LSTM will give a similar accuracy, so in this case it would be profitable to use GRUs.

Q50. Are LSTMs the most advanced sequence models?

A50. No, LSTMs are not the most advanced sequence models, New concepts and new models are always being created by deep learning researches all around the world. Transformers and the attention model architecture are arguably the most advanced sequence model architectures today with state-of-the-art NLP models like GPT-3 and BERT implementing those architectures.

LSTM MODEL CODE

First, let's import the required libraries:

```
import tensorflow as tf


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, RepeatVector
```

Let's create a LSTM model for our dataset now:

```
model_lstm=Sequential([
    LSTM(128, input_shape=(None,11)),
    RepeatVector(5),
    LSTM(128, return_sequences=True),
    Dense(11,activation='softmax')
])


model_lstm.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

This is what the model summary looks like:

```
model_lstm.summary()
```

Output:

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_6 (LSTM)                (None, 128)               71680
_________________________________________________________________
repeat_vector_9 (RepeatVecto (None, 5, 128)            0
_________________________________________________________________
lstm_7 (LSTM)                (None, 5, 128)            131584
_________________________________________________________________
dense_22 (Dense)             (None, 5, 11)             1419
=================================================================
Total params: 204,683
Trainable params: 204,683
Non-trainable params: 0
_________________________________________________________________
```

Let's train the model for 100 epochs now:

```
model_lstm.fit(x_train, y_train, epochs=100, validation_split=0.2,
        batch_size=1024)
```

Let's test the model accuracy on some test examples!

```
x_test, y_test = create_dataset(num_examples=20)
preds = model_lstm.predict(x_test)
full_seq_acc = 0

for i, pred in enumerate(preds):
    pred_str = devectorize_example(pred)
    y_test_str = devectorize_example(y_test[i])
    x_test_str = devectorize_example(x_test[i])
    full_seq_acc += 1/len(preds) * int(pred_str == y_test_str)
    outstring = 'Input: {}, Out: {}, Pred: {}'.format(x_test_str,
y_test_str, pred_str)
    print(outstring)
print('\nFull sequence accuracy: {:.3f} %'.format(100 * full_seq_acc))
```

Output:

```
Input: 12+22, Out: 00034, Pred: 00034
Input: 035+7, Out: 00042, Pred: 00042
Input: 15+13, Out: 00028, Pred: 00028
Input: 94+48, Out: 00142, Pred: 00142
Input: 55+37, Out: 00092, Pred: 00092
Input: 16+26, Out: 00042, Pred: 00042
Input: 007+8, Out: 00015, Pred: 00015
Input: 044+8, Out: 00052, Pred: 00052
Input: 13+84, Out: 00097, Pred: 00097
Input: 69+42, Out: 00111, Pred: 00111
Input: 012+5, Out: 00017, Pred: 00017
Input: 99+16, Out: 00115, Pred: 00115
Input: 98+50, Out: 00148, Pred: 00148
Input: 79+22, Out: 00101, Pred: 00101
Input: 50+10, Out: 00060, Pred: 00060
Input: 95+34, Out: 00129, Pred: 00129
Input: 92+39, Out: 00131, Pred: 00131
Input: 99+32, Out: 00131, Pred: 00131
Input: 90+72, Out: 00162, Pred: 00162
Input: 68+99, Out: 00167, Pred: 00167

Full sequence accuracy: 100.000 %
```

As you can see, the LSTM model has predicted each and every output in the test examples correctly. This LSTM model has achieved a perfect accuracy of 100%!

Although the training was a lot slower. It took almost twice the time to train as the RNN. As mentioned in the question above at the end of the LSTM section, in this case, it might be better to use the normal recurrent neural network model as the training time was much shorter and faster for the RNN. Also, as you can see in the model summary, the recurrent neural network has 52,235 parameters, and the LSTM model has 204,683 parameters, which is almost 4 times the size of the RNN model.