INT246 – Report. CA_02(KOM106)

ML In Finance - Stock Price Predictions Using Regression & LSTM

Shahbaz Ahmad 11809952

Abstract: -

- Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) have been transforming finance and investing.
- "Artificial intelligence is to trading what fire was to the cavemen"!
- Electronic trades account for almost 45% of revenues in cash equities trading" U.K. research firm Coalition Report.
- AI powered robo-advisers can perform real-time analysis on massive datasets and trade securities at an extremely faster rate compared to human traders.
- AI-powered trading could potentially reduce risk and maximize returns.
- Check out the list of companies that leverage AI in trading:
- https://builtin.com/artificial-intelligence/ai-trading-stock-market-tech

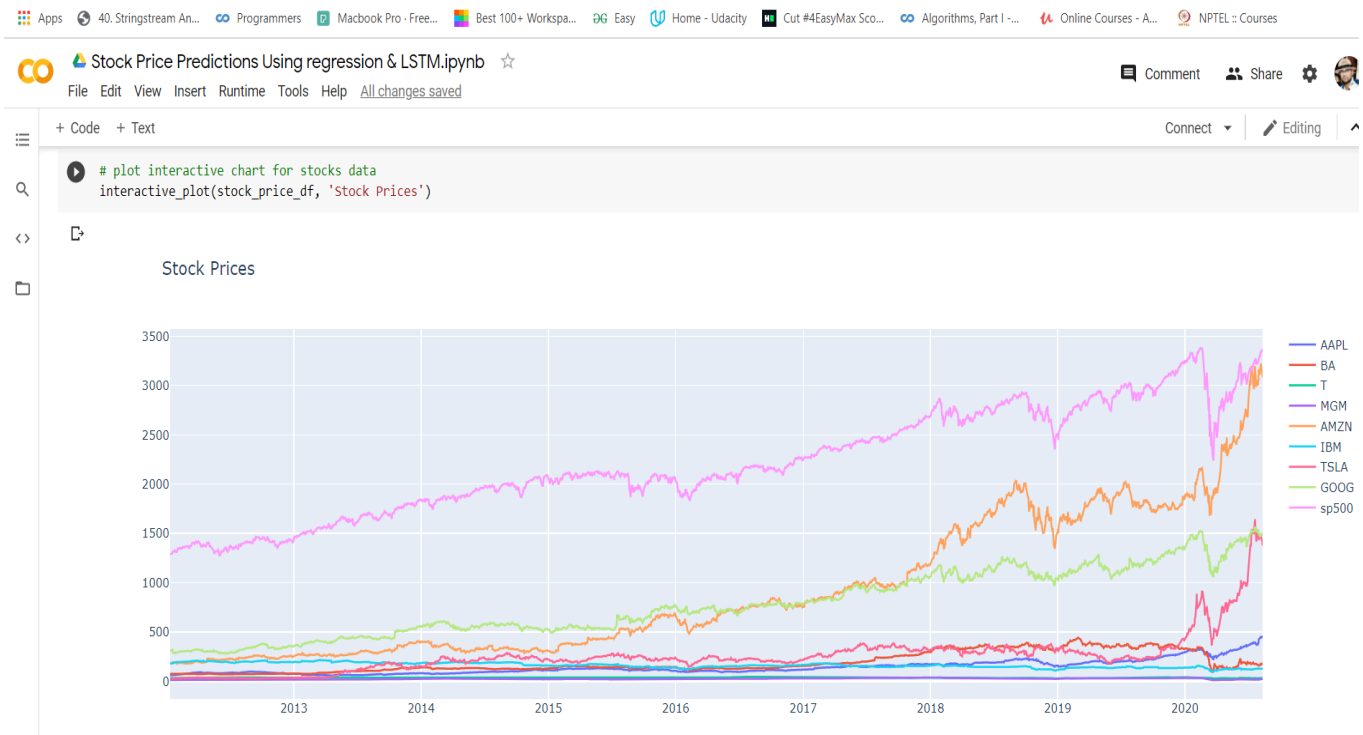
Photo Credit: https://www.pxfuel.com/en/free-photo-qauli

TRAINING & TESTING DATA SPLIT

- Data set is divided into 75% for training and 25% for testing.
  - Training set: used for model training.
  - Testing set: used for testing trained model. Make sure that the testing dataset has never been seen by the trained model before.

INTRODUCTION: -

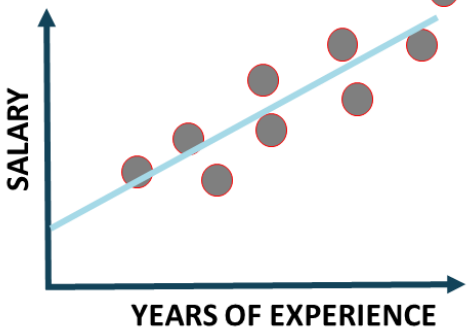
- In this project, we will train a ridge regression model and deep neural network model to predict future stock prices.
- By accurately predicting stock prices, investors can maximize returns and know when to buy/sell securities.
- The AI/ML model will be trained using historical stock price data along with the volume of transactions.
- We will use a type of neural nets known as Long Short-Term Memory Networks (LSTM).
- Disclaimer: Stock prices are volatile and are generally hard to predict. Invest at your own risk.



BACKGROUND: -

SIMPLE LINEAR REGRESSION: INTUITION

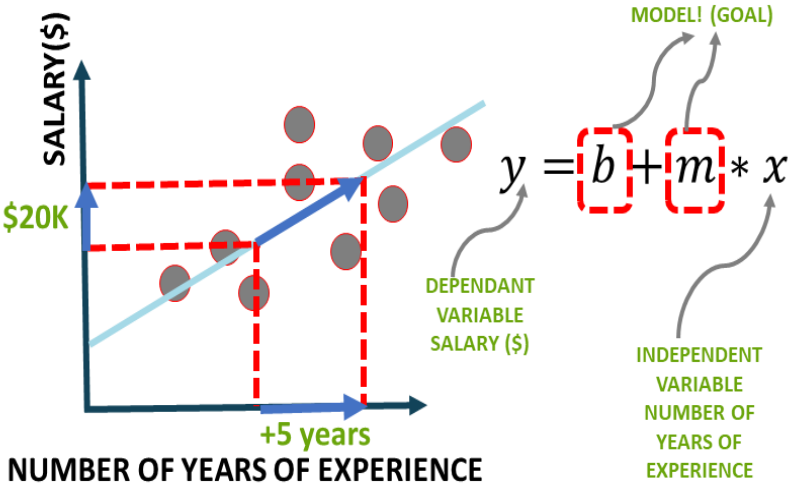
- In simple linear regression, we predict the value of one variable Y based on another variable X.
- X is called the independent variable and Y is called the dependant variable.
- Why simple? Because it examines relationship between two variables only.
- Why linear? when the independent variable increases (or decreases), the dependent variable increases (or decreases) in a linear fashion.



	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111

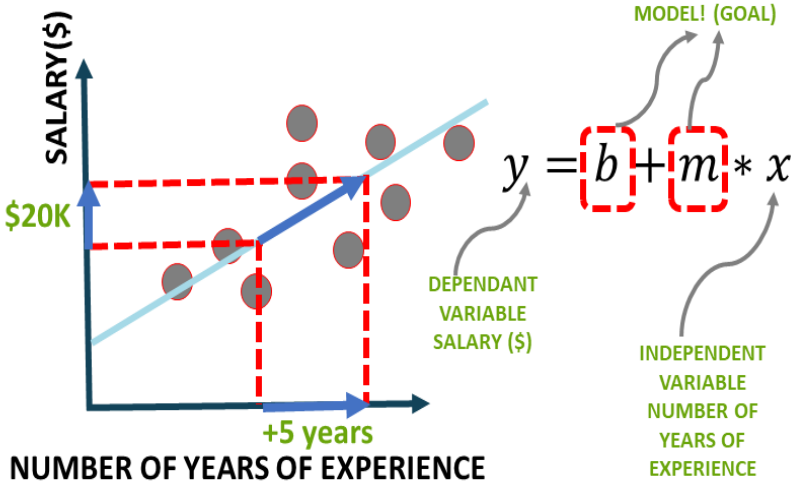
SIMPLE LINEAR REGRESSION: SOME MATH!

- Goal is to obtain a relationship (model) between employee salary and number of years of experience.



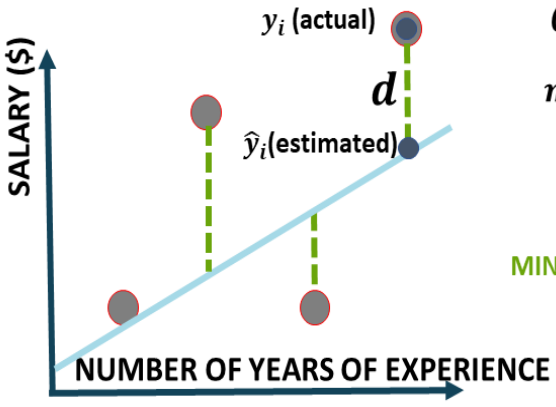
SIMPLE LINEAR REGRESSION: SOME MATH!

- Goal is to obtain a relationship (model) between employee salary and number of years of experience.



SIMPLE LINEAR REGRESSION: HOW TO OBTAIN MODEL PARAMETERS? LEAST SUM OF SQUARES

- Least squares fitting is a way to find the best fit curve or line for a set of points.
- The sum of the squares of the offsets (residuals) are used to estimate the best fit curve or line.
- Least squares method is used to obtain the coefficients m and b.



$$d = \hat{y}_i - y_i$$

$$min \sum (\hat{y}_i - y_i)^2$$

MINIMUM (LEAST) SUM OF SQUARES

TRAINING & TESTING DATA SPLIT

- **Data set is divided into 75% for training and 25% for testing.**
  - **Training set: used for model training.**
  - **Testing set: used for testing trained model. Make sure that the testing dataset has never been seen by the trained model before.**

REGULARIZATION: INTUITION

- **Regularization techniques are used to avoid networks overfitting**
- **Overfitting occurs when the model provide great results on the training data but performs poorly on testing dataset.**
- **Overfitting occurs when the model learns all the patterns of the training dataset but fails to generalize.**
- **Overfitted models generally provide high accuracy on training dataset but low accuracy on testing and validation (evaluation) datasets**

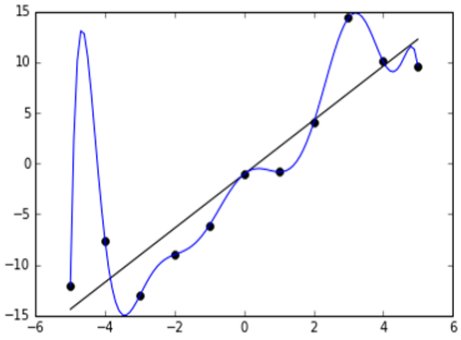
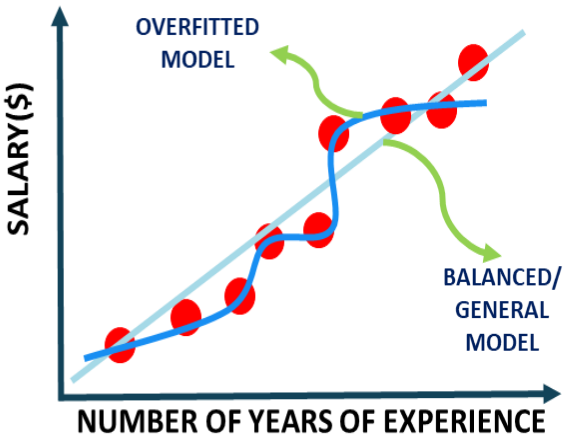


Photo Credit: https://commons.wikimedia.org/wiki/File:Overfitted_Data.png

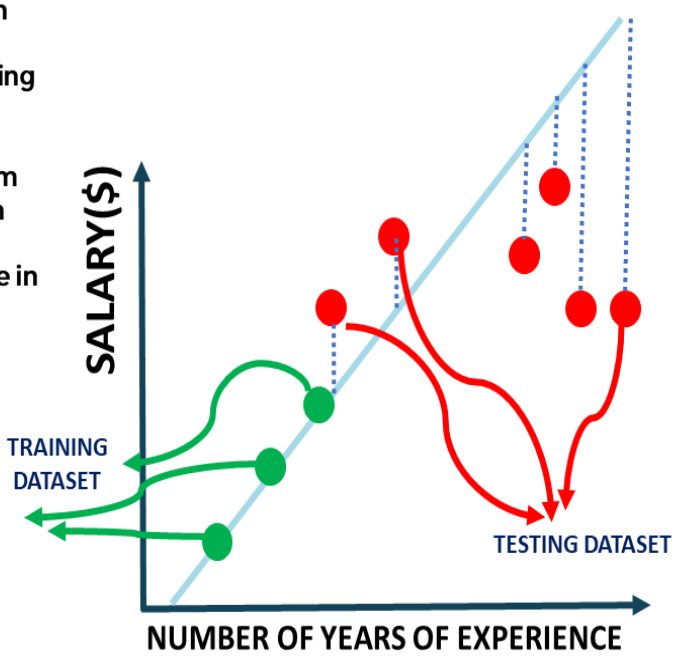
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

- Ridge regression advantage is to avoid overfitting.
- Our ultimate model is the one that could generalize patterns; i.e.: works best on the training and testing dataset
- Overfitting occurs when the trained model performs well on the training data and performs poorly on the testing datasets
- Ridge regression works by applying a penalizing term (reducing the weights and biases) to overcome overfitting.



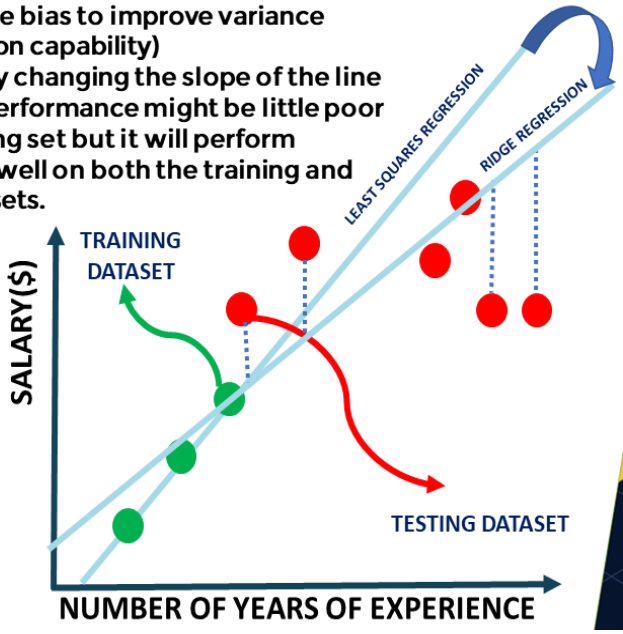
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

- Least sum of squares is applied to obtain the best fit line
- Since the line passes through the 3 training dataset points, the sum of squared residuals = 0
- However, for the testing dataset, the sum of residuals is large so the line has a high variance.
- Variance means that there is a difference in fit (or variability) between the training dataset and the testing dataset.
- This regression model is overfitting the training dataset



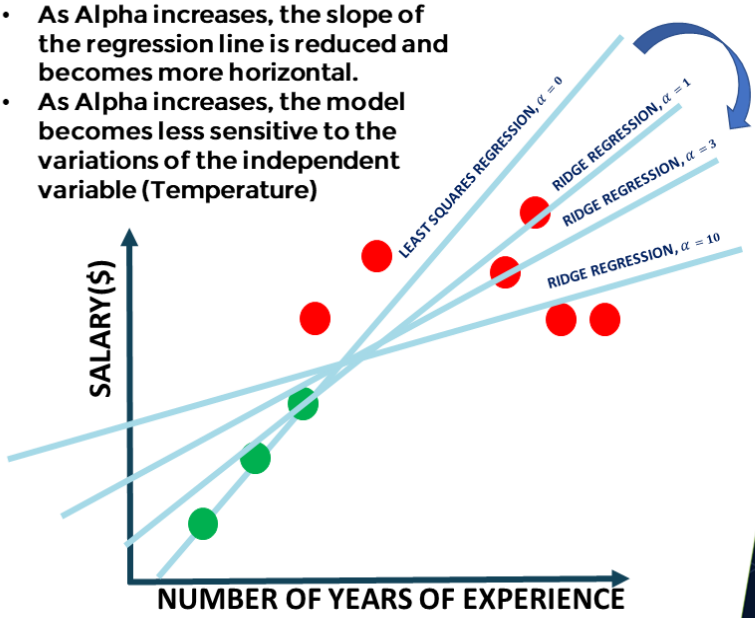
RIDGE REGRESSION (L2 REGULARIZATION): INTUITION

- Ridge regression works by attempting at increasing the bias to improve variance (generalization capability)
- This works by changing the slope of the line
- The model performance might be little poor on the training set but it will perform consistently well on both the training and testing datasets.



RIDGE REGRESSION (L2 REGULARIZATION): ALPHA EFFECT

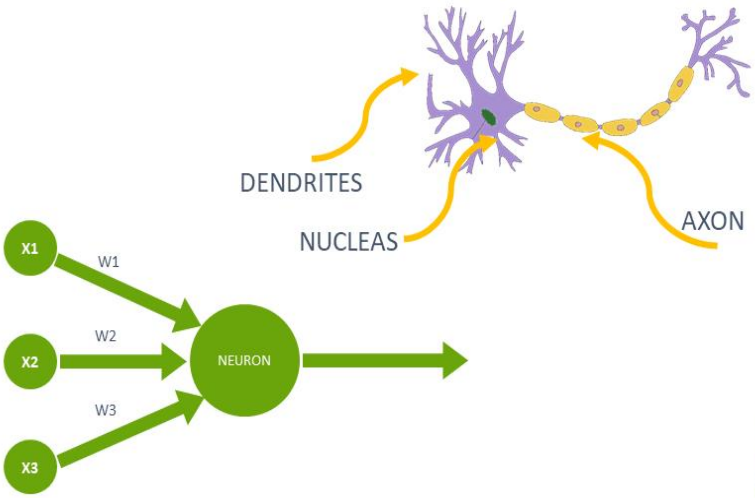
- As Alpha increases, the slope of the regression line is reduced and becomes more horizontal.
- As Alpha increases, the model becomes less sensitive to the variations of the independent variable (Temperature)



PROPOSED SYSTEM: -

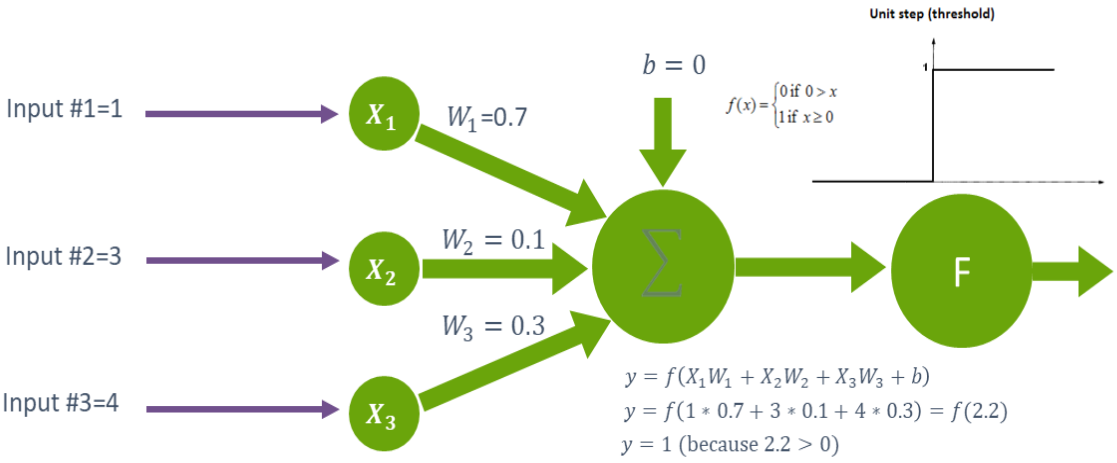
NEURON MATHEMATICAL MODEL

- Artificial Neural Networks are information processing models that are inspired by the human brain.
- The neuron collects signals from input channels named dendrites, processes information in its nucleus, and then generates an output in a long thin branch called axon.



SINGLE NEURON MODEL IN ACTION!

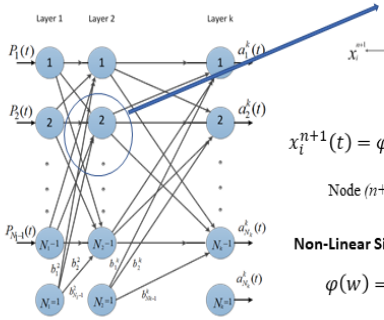
- Let's assume an activation function of Unit Step.
- The activation functions is used to map the input between (0, 1).

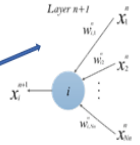


MULTI-LAYER PERCEPTRON NETWORK

- Let's connect multiple of these neurons in a multi-layer fashion.
- The more hidden layers, the more "deep" the network will get.

$$P = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{N_1} \end{bmatrix}$$





$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n \, x_j^n(t)\right)$$

Node $(n+1, i)$ representation

Non-Linear Sigmoid Activation function

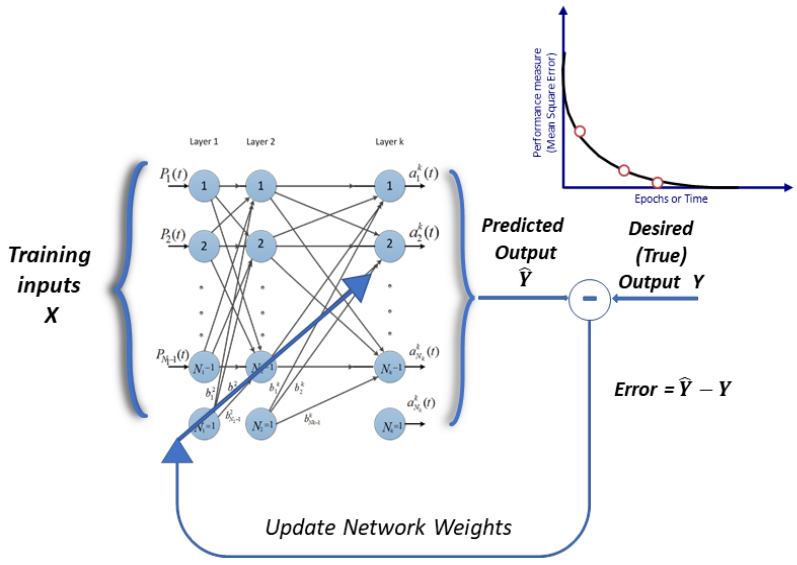
$$\varphi(w) = \frac{1}{1+e^{-w}}$$

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,N_1} \\ W_{21} & W_{22} & & W_{2,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \cdots & W_{m-1,N_1} \\ W_{m,1} & W_{m,2} & \cdots & W_{m,N_1} \end{bmatrix}$$

m: number of neurons in the hidden layer

N_1: number of inputs

HOW DO ANN TRAIN? PROCESS OVERVIEW



HOW DO ANN TRAIN? GRADIENT DESCENT

- Gradient descent is an optimization algorithm used to obtain the optimized network weight and bias values.
- It works by iteratively trying to minimize the cost function.
- It works by calculating the gradient of the cost function and moving in the negative direction until the local/global minimum is achieved.
- The size of the steps taken are called the learning rate
- If learning rate increases, the area covered in the search space will increase so we might reach global minimum faster
- However, we can overshoot the target
- For small learning rates, training will take much longer to reach optimized weight values

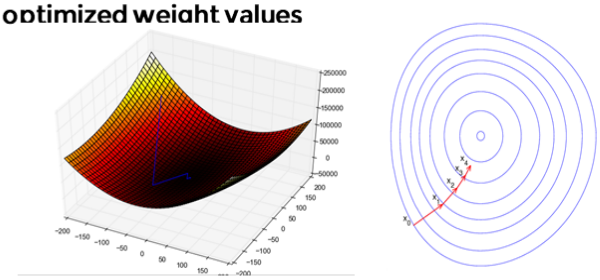


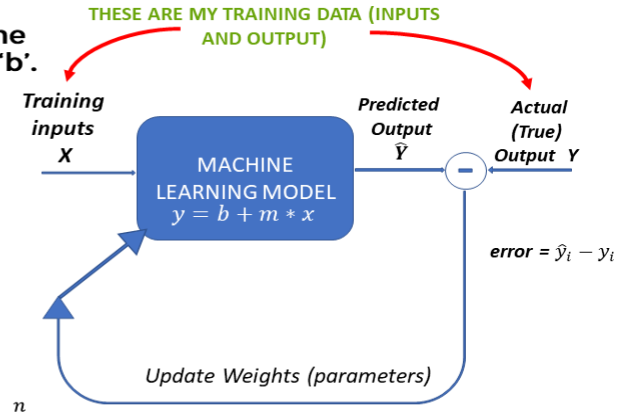
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

HOW DO ANN TRAIN? GRADIENT DESCENT

- Let's assume that we want to obtain the optimal values for parameters 'm' and 'b'.

$$y = b + m * x$$

GOAL IS TO FIND BEST PARAMETERS

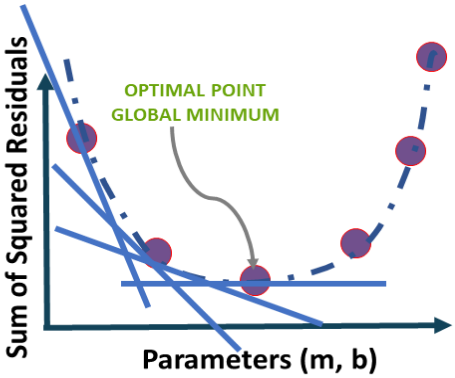


- We need to first formulate a loss function as follows:

$$Cost\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(error)^2 = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

HOW DO ANN TRAIN? GRADIENT DESCENT

$$Loss\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



GRADIENT DESCENT WORKS AS FOLLOWS:

1. Calculate the gradient (derivative) of the Loss function $\frac{\partial loss}{\partial w}$
2. Pick random values for weights (m, b) and substitute
3. Calculate the step size (how much are we going to update the parameters?)

$$Step\ size\ = learning\ rate * gradient = \alpha * \frac{\partial loss}{\partial w}$$

4. Update the parameters and repeat

$$new\ weight\ =\ old\ weight - step\ size$$

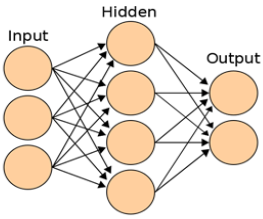
$$w_{new} = w_{old} - \alpha * \frac{\partial loss}{\partial w}$$

**Note: in reality, this graph is 3D and has three axes, one for m, b and sum of squared residuals*

RECURRENT NEURAL NETWORKS (RNN): WHAT ARE THEY?

- Feedforward Neural Networks (vanilla networks) map a fixed size input (such as image) to a fixed size output (classes or probabilities).
- A drawback in Feedforward networks is that they do not have any time dependency or memory effect.
- A RNN is a type of ANN that is designed to take temporal dimension into consideration by having a memory (internal state) (feedback loop).

FEED FORWARD ANN



RECURRENT NEURAL NETWORK

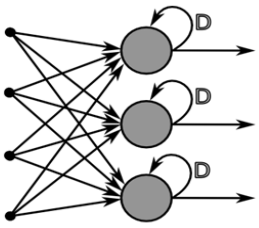
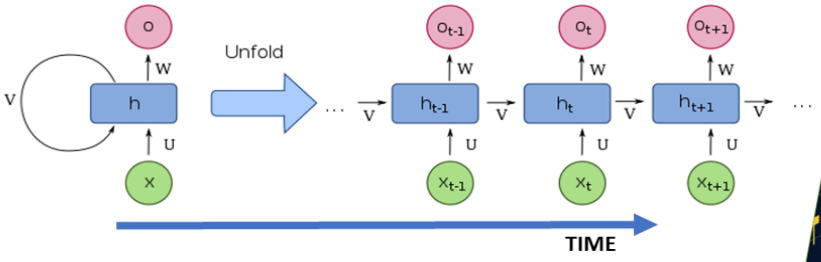


Photo Credit: https://commons.wikimedia.org/wiki/File:RecurrentLayerNeuralNetwork_english.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

RNN ARCHITECTURE

- A RNN contains a temporal loop in which the hidden layer not only gives an output but it feeds itself as well.
- An extra dimension is added which is time!
- RNN can recall what happened in the previous time stamp so it works great with sequence of text.

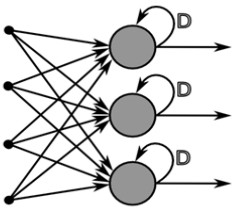


"We'll train RNNs to generate text character by character and ponder the question "how is that even possible?"
Source: The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

WHAT MAKES RNNs SO SPECIAL?

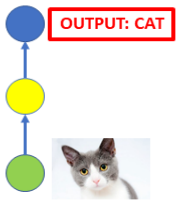
- Feedforward ANNs are so constrained with their fixed number of input and outputs.
- For example, a CNN will have fixed size image (28x28) and generates a fixed output (class or probabilities).
- Feedforward ANN have a fixed configuration, i.e.: same number of hidden layers and weights.
- Recurrent Neural Networks offer huge advantage over feedforward ANN and they are much more fun!
- RNN allow us to work with a sequence of vectors:
  - Sequence in inputs
  - Sequence in outputs
  - Sequence in both!



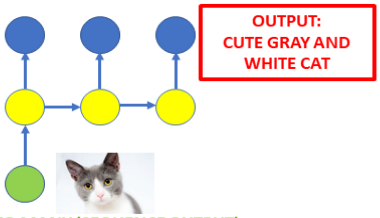
Source: The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Photo Credit: https://commons.wikimedia.org/wiki/File:RecurrentLayerNeuralNetwork_english.png

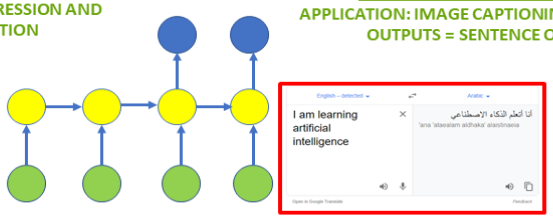
WHAT MAKES RNNs SO SPECIAL?



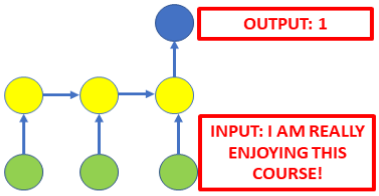
ONE TO ONE (VANILLA)
APPLICATION: REGRESSION AND CLASSIFICATION



ONE TO MANY (SEQUENCE OUTPUT)
APPLICATION: IMAGE CAPTIONING, INPUT = IMAGE
OUTPUTS = SENTENCE OF WORDS)



MANY TO MANY (SEQUENCE INPUT AND OUTPUT)
APPLICATION: LANGUAGE TRANSLATION,



MANY TO ONE (SEQUENCE INPUT)
APPLICATION: SENTIMENT ANALYSIS,
EX: IS A REVIEW POSITIVE OR NEGATIVE?

Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

RNN MATH

- A RNN accepts an input x and generate an output o.
- The output o does not depend on the input x alone, however, it depends on the entire history of the inputs that have been fed to the network in previous time steps.
- Two equations that govern the RNN are as follows:
  - **INTERNAL STATE UPDATE:**
    $$h_t = \tanh(X_t * U + h_{t-1} * V)$$
  - **OUTPUT UPDATE:**
    $$o_t = softmax(W * h_t)$$

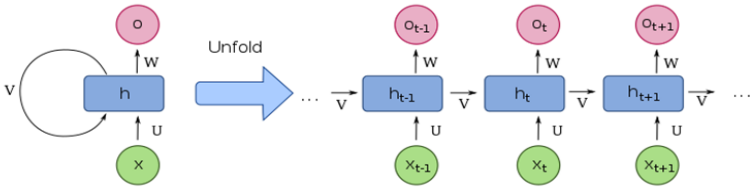


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

LET'S WATCH THIS MOVIE WRITTEN BY AN RNN!

- Let's watch a movie written by AI!
https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/
- The movie was written by an LSTM recurrent neural network
- The LSTM network was trained with a corpus of dozens of sci-fi screenplays from movies from the 1980s and 90s.

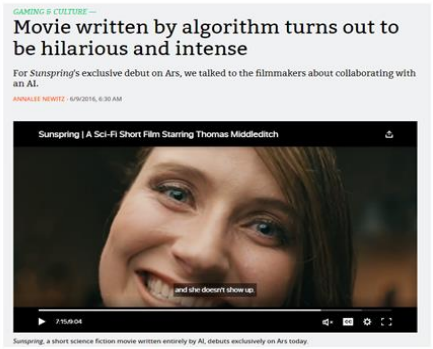


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

VANISHING GRADIENT PROBLEM

- LSTM networks work much better compared to vanilla RNN since they overcome the vanishing gradient problem.
- The error has to propagate through all the previous layers resulting in a vanishing gradient.
- As the gradient goes smaller, the network weights are no longer updated.
- As more layers are added, the gradients of the loss function approaches zero, making the network hard to train.

EACH LAYER DEPENDS ON THE OUTPUT FROM THE PREVIOUS LAYERS, THE "V" IS MULTIPLIED SEVERAL TIMES RESULTING IN VANISHING GRADIENT

0.1 * 0.1 * 0.1 *.....*0.1 =1e-10

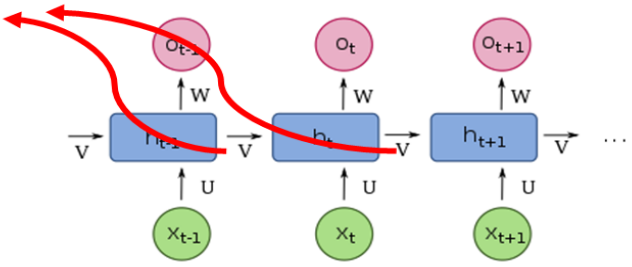
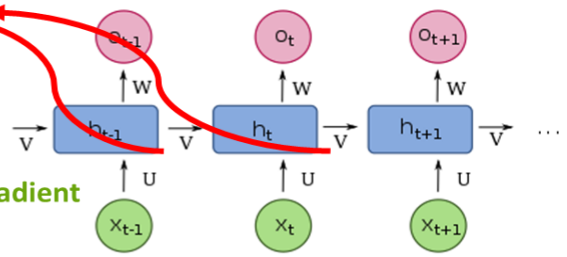


Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

VANISHING GRADIENT PROBLEM

- ANN gradients are calculated during backpropagation.
- In backpropagation, we calculate the derivatives of the network by moving from the outermost layer (close to output) back to the initial layers (close to inputs).
- The chain rule is used during this calculation in which the derivatives from the final layers are multiplied by the derivatives from early layers.
- The gradients keeps diminishing exponentially and therefore the weights and biases are no longer being updated.

EACH LAYER DEPENDS ON THE OUTPUT FROM THE PREVIOUS LAYERS, THE "V" IS MULTIPLIED SEVERAL TIMES RESULTING IN VANISHING GRADIENT, (ex: 0.1 * 0.1 * 0.1 *.....*0.1 =1e-10)



New Weight = Old Weight – Learning rate * gradient

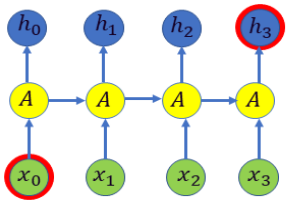
9.09999 = 10.1 – 1 * 0.001

Photo Credit: https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg

LSTM INTUITION

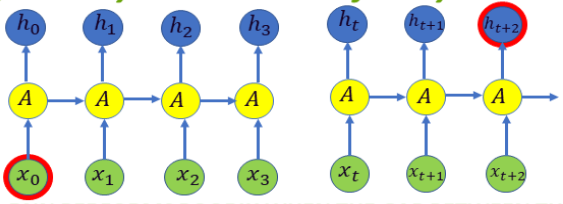
- LSTM networks work better compared to vanilla RNN since they overcome vanishing gradient problem.
- In practice, RNN fail to establish long term dependencies.
- Reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The tree color is "green"



RNN PERFORMS WELL SINCE THE GAP BETWEEN THE PREDICTION "GREEN" AND THE NECESSARY CONTEXT INFORMATION "TREE" IS SMALL

I live in Quebec in Northern Canada……where I live, the weather is generally "cold" most of the year



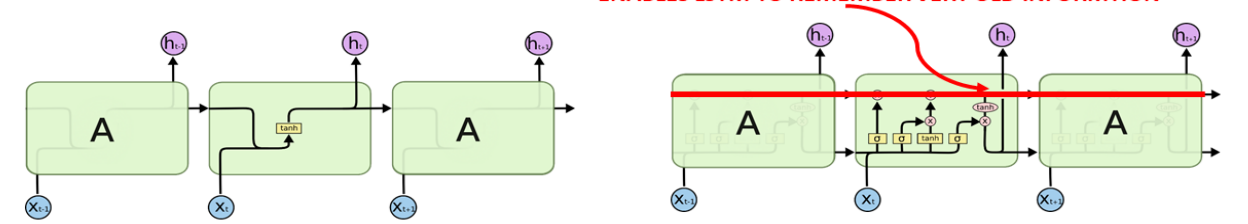
RNN PERFORMS POORLY WHEN THE GAP BETWEEN THE PREDICTION "COLD" AND THE NECESSARY CONTEXT INFORMATION "CANADA" IS LARGE

Reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM INTUITION

- LSTM networks are type of RNN that are designed to remember long term dependencies by default.
- LSTM can remember and recall information for a prolonged period of time.
- Recall that each line represents a full vector.

THIS HORIZONTAL LINE (MEMORY) OR CELL STATE ENABLES LSTM TO REMEMBER VERY OLD INFORMTION



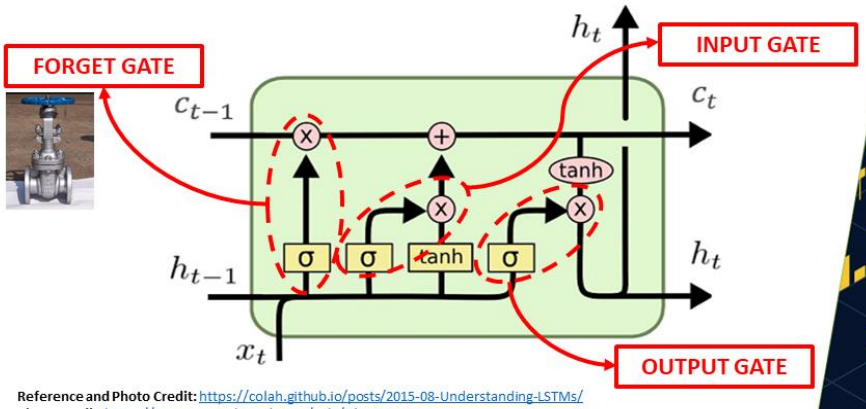
VANILLA RECURRENT NEURAL NETWORK

LONG SHORT TERM MEMORY NETWORK

Reference and Photo Credit:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM INTUITION – GATES

- LSTM contains gates that can allow or block information from passing by.
- Gates consist of a sigmoid neural net layer along with a pointwise multiplication operation.
- Sigmoid output ranges from 0 to 1:
  0 = Don't allow any data to flow
  1 = Allow everything to flow!



Reference and Photo Credit: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
Photo Credit: https://commons.wikimedia.org/wiki/File:LSTM.png

Conclusion: -

Stock Price Predictions Using regression & LSTM.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect Editing

```
# Create the model
inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
x = keras.layers.LSTM(150, return_sequences= True)(inputs)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150, return_sequences=True)(x)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150)(x)
outputs = keras.layers.Dense(1, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss="mse")
model.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1, 1)]            0
_________________________________________________________________
lstm (LSTM)                  (None, 1, 150)            91200
_________________________________________________________________
dropout (Dropout)            (None, 1, 150)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 150)            180600
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 150)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 150)               180600
_________________________________________________________________
dense (Dense)                (None, 1)                 151
=================================================================
Total params: 452,551
Trainable params: 452,551
Non-trainable params: 0
```

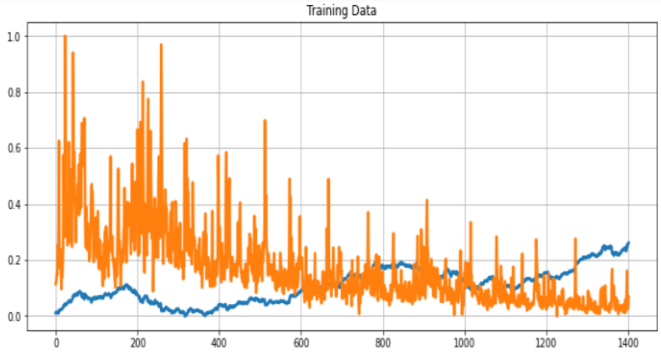
Stock Price Predictions Using regression & LSTM.ipynb

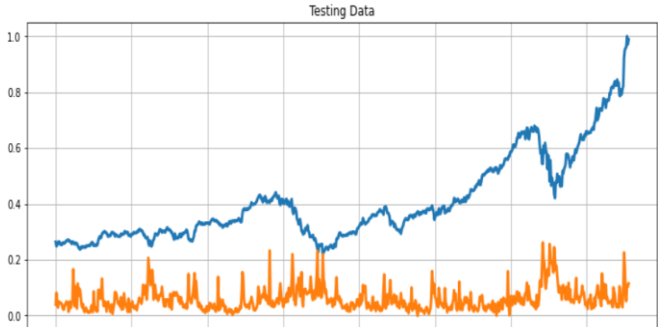
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

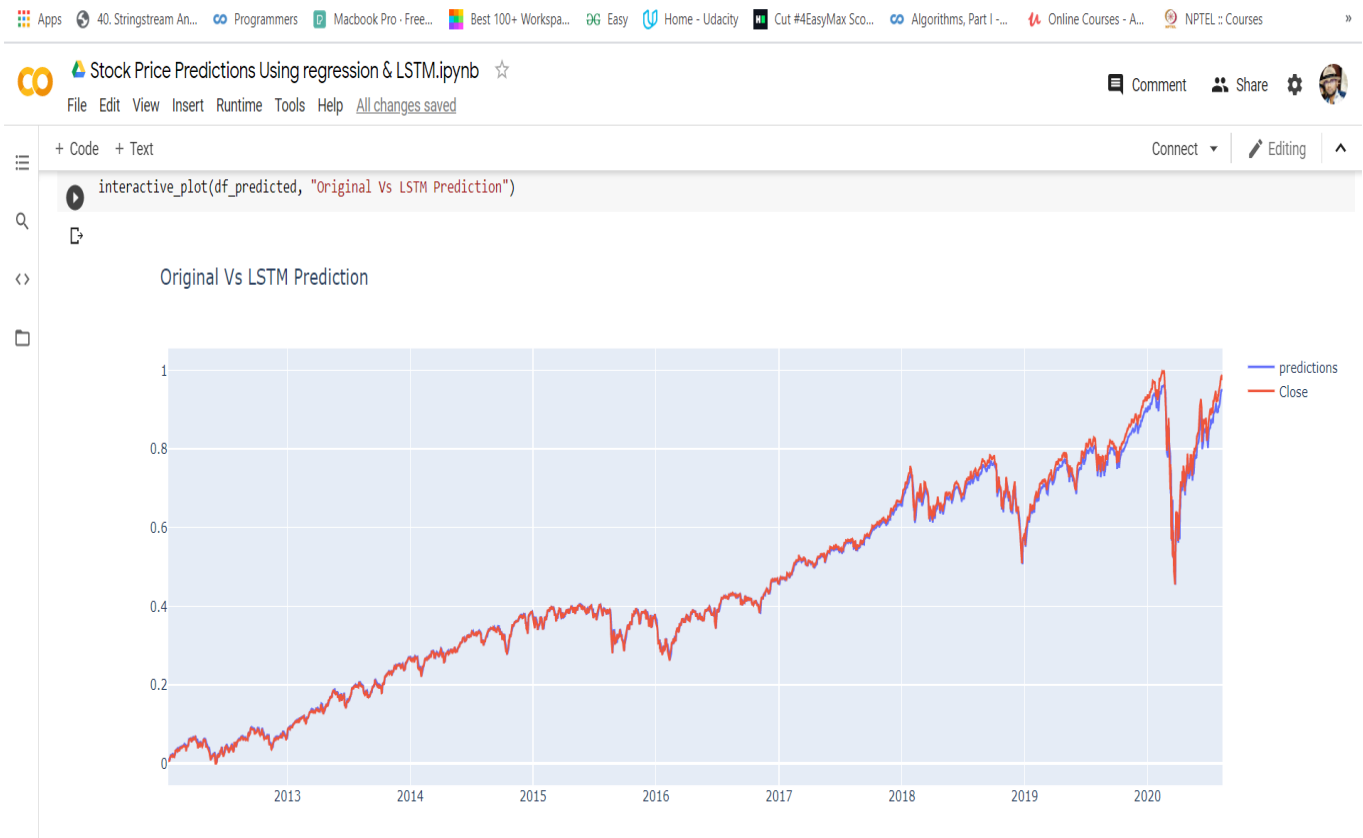
+ Code + Text

Connect Editing





Final Prediction: -



Earlier, I mentioned the remarkable results people are achieving with RNNs. Essentially all of these are achieved using LSTMs. They really work a lot better for most tasks!

Written down as a set of equations, LSTMs look pretty intimidating. Hopefully, walking through them step by step in this essay has made them a bit more approachable.

LSTMs were a big step in what we can accomplish with RNNs. It's natural to wonder: is there another big step? A common opinion among researchers is: "Yes! There is a next step and it's attention!" The idea is to let every step of an RNN pick information to look at from some larger collection of information. For example, if you are using an RNN to create a caption describing an image, it might pick a part of the image to look at for every word it outputs. In fact, Xu, *et al.* (2015) do exactly this – it might be a fun starting point if you want to explore attention! There's been a number of really exciting results using attention, and it seems like a lot more are around the corner…

Attention isn't the only exciting thread in RNN research. For example, Grid LSTMs by Kalchbrenner, *et al.* (2015) seem extremely promising. Work using RNNs in generative models – such as Gregor, *et al.* (2015), Chung, *et al.* (2015), or Bayer & Osendorfer (2015) – also seems very interesting. The last few years have been an exciting time for recurrent neural networks, and the coming ones promise to only be more so!

References: -

Udemy Courses by Dr. Ryan Ahmad

https://colah.github.io/posts/2015-08-Understanding-LSTMs/