BCS FAP Winter 2025

Turing's Enigma (Vibe Checker)

December 2025

1 Artificial Neural Networks

1.1 What is ANN

Artificial Neural Networks (ANNs) are a fundamental concept in machine learning and artificial intelligence. They are designed to learn patterns from data and make predictions or decisions by modeling the flow of information through interconnected computational units.

A neural network is a machine learning model composed of artificial neurons arranged in layers. It learns by adjusting weights and biases to map input data to output results, enabling the model to capture complex relationships within the data.

The structure of a neural network consists of three main layers: the input layer, hidden layers, and the output layer. The input layer receives raw features from the dataset and serves as the entry point of information. Hidden layers lie between the input and output layers and contain multiple neurons that apply weights, biases, and activation functions to transform input data into meaningful and nonlinear patterns. The output layer produces the final prediction, such as a class label or numerical value, depending on the task.

Neural networks learn through an iterative training process. During training, the network generates predictions and compares them with actual outputs using a loss function. The resulting error is propagated backward through the network using backpropagation, and parameters are updated through optimization techniques like gradient descent. This process continues until the network's performance stabilizes.

1.2 Backpropagation

The difference between the predicted output and the actual output(Error/Loss) is minimized using an algorithm called backpropagation. The algorithm works backward through the network. At each neuron, the algorithm uses the chain rule from calculus to determine how much each weight and bias influences the error. It then determines how these should be changed in order to minimize the error.

Beginning at the output layer, a backward pass takes derivatives of the loss function to measure how each individual network parameter contributes to the total error for a single input.

Gradient descent describes the general idea of how improvement happens. Conceptually, it involves making small changes that move the network toward better results. If a change reduces the loss, it is seen as a step in the right direction. By repeating many small steps, the network gradually improves over time.

2 Convolutional Neural Networks

In this section, we learn about CNNs which are primarily used in the field of pattern recognition within images just like ANN. However, we overcome the computational capabilities of ANN. This allows us to encode image-specific features into the architecture, making the network more suited for image focused tasks whilst further reducing the parameters required to set up the model. We have analyzed the MNIST database. This database contains different images of handwritten numbers and in this dataset a single neuron in the first hidden layer will contain 784 weights (28*28) (**?**).

So, while dealing with data we have created three types of layers : Convolutional layer, pooling layers and fully connected layers.

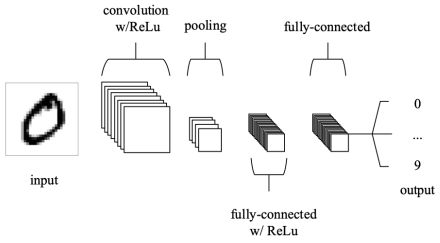


Figure 1: CNN Layers

Initially we created 3*3 kernels (which represent features such as colors and edges) and slide over the Input image of 28*28. then we get our first feature convolutional layer. Following similar path, we get more convolutional layers and we will get a Feature Map. These kernels also act as weights. After that we used Rectified linear unit(ReLu) for the activation function as sigmoid. Furthermore, Using Maxpool we created pooling layers with dimensionality 2*2 which helps in downsampling along the spatial dimensionality of the given input image. And finally we get Fully Connected layers which usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1. Thus as we move along with hierarchical structure of the CNN the later layers can see the pixels within the receptive fields of prior layers and identify the full feature.

We also learned that this feature map is impacted by number of filters, stride and padding also known as hyperparameters.

2.1 CNN Code

After loading the data we have converted the data type to floating-point numbers from (0,255) to (0,1). Because neural networks train more efficiently with smaller input values. And with consistent scale, optimization algorithms work better with normalized data.

Table 1: CNN Architecture Dimensions and Data Flow

Stage	**Layer**	**Output Shape**	**Total Values**
Input	Image	$[1, 28, 28]$	784
Feature 1	Conv + Pool	$[8, 14, 14]$	1,568
Feature 2	Conv + Pool	$[16, 7, 7]$	784
Transition	Flatten	[784]	784
Output	FC1 (Hidden)	[64]	64
Output	FC2 (Result)	[10]	10

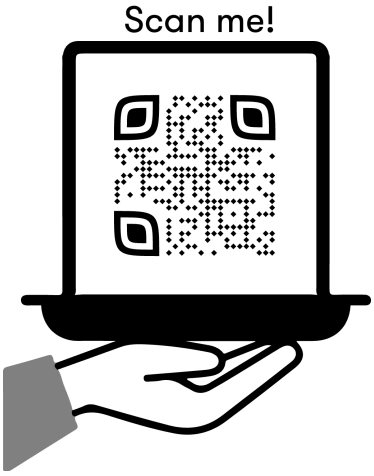


Figure 2: Information about working of CNN code using pytorch

We also changed the learning rate from 0.01 to 0.02 and analyze its impact over epochs from 5 to 10. We got the accuracy **98.10** to **98.79** percent.

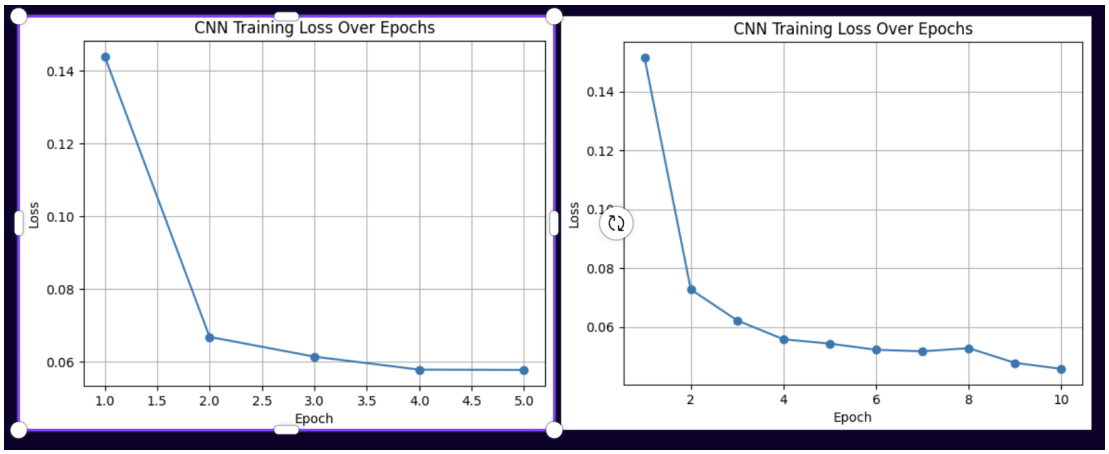


Figure 3: Epoch at Learning rate 0.01

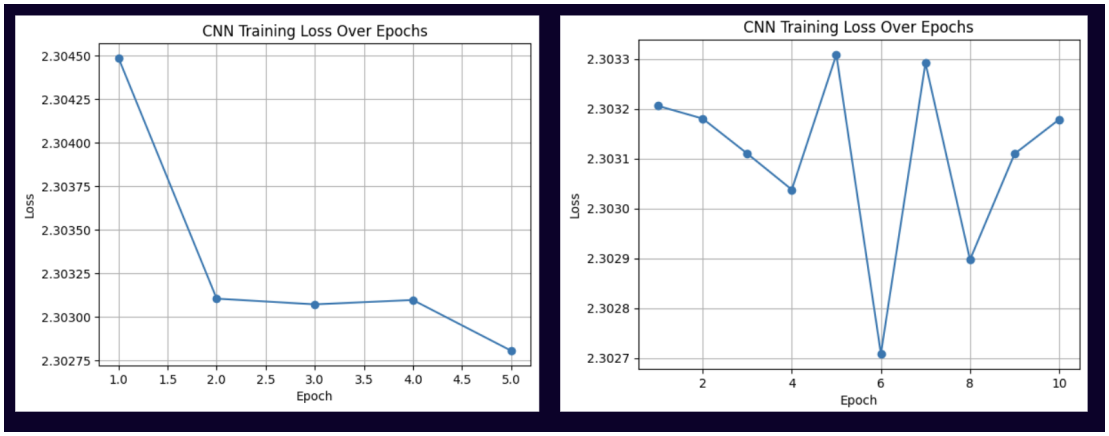


Figure 4: Epoch at Learning rate 0.02

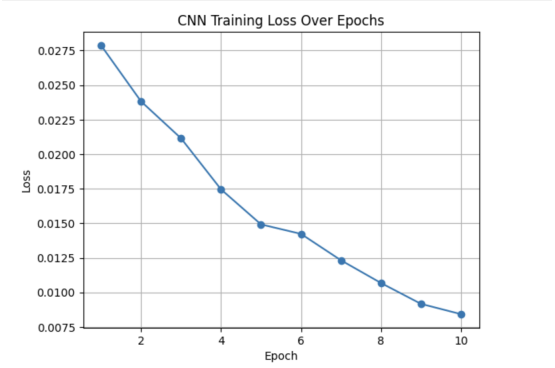


Figure 5: Epoch at Default Learning rate 0.001

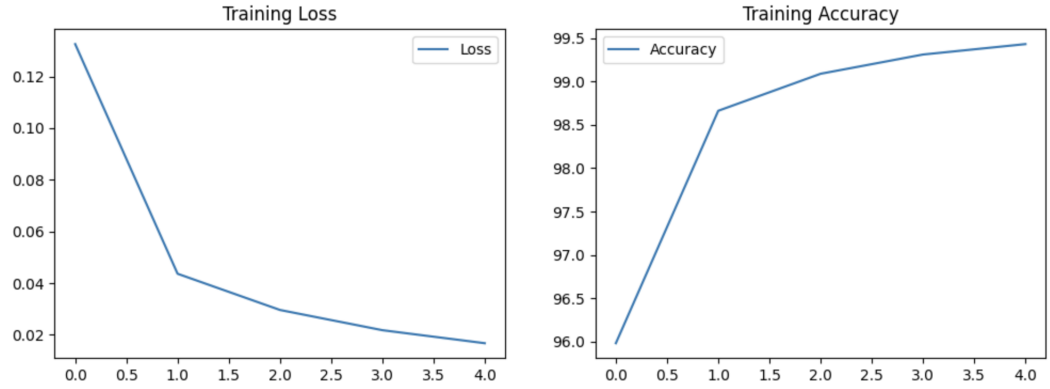


Figure 6: Training data

3 Recurrent Neural Networks and LSTMs

In this section we talk about **RNNs and LSTMs** and the need to develop them. Traditional Neural Networks which aimed to mimic Human intelligence had a major shortcoming ,Memory.Most of human tasks are sequential in nature ,it means they involve use of previous information which traditional networks had no mechanism to imitate.

RNNs address this issue with the help of hidden state which is updated and carried forward in loops with the input sequence thus allowing information to persist .RNNs can be thought of as multiple copies of the same neural network, each passing a message to a successor.

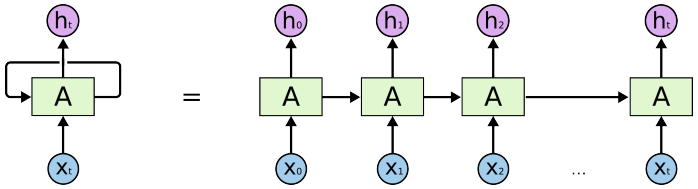


Figure 7: An unrolled recurrent neural network.

3.1 Limitations of a Vanilla RNN

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task. Unfortunately, as that gap in the context grows, RNNs become unable to learn to connect the relevant information.During Backpropagation ,there are two ways RNNs can become faulty in long term context- 1.Vanishing Gradients 2.Exploding Gradients While Exploding gradients can be solved by a simple tweaking of code called as gradient

clipping., Vanishing gradient solution requires to change almost the structure of Vanilla RNNs into a new sophisticated RNN called as LSTMS.

3.2 Need for LSTM Networks

Long Short Term Memory networks – in short "LSTMs" ,are a special kind of RNNs designed to tackle the problem of RNNs Vanishing Gradients. LSTMs do this by creating two different states hidden state and cell state . The cell state path works like a highway for information transfer involving only some linear interactions thus making it far more computationally easier to calculate gradients during backpropagation .LSTMs does have to add information to cell state that is governed by four gate layers.Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

3.3 LSTM in detail

In short, LSTM has to take some part of previous cell state and current cell state candidate values and create a new current cell state out of which some information is communicated to the hidden state. First, we have forget gate layer that governs how much previous cell state information is kept .

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

Then there is an input gate layer which governs how much the new candidate values should be kept in cell state .

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_i) \tag{3}$$

Then the cell state is updated with help of these two gates surrounding it with a tanh function

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C_t} \tag{4}$$

then there is the the output gate which governs how much cell state information is to provided to hidden state

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t \odot \tanh(C_t) \tag{6}$$

3.4 GRU

GRU(Gated Recurrent Units) is a variation of LSTMs that combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and gives similar results while being computationaly faster.

3.5 Our Implementation

We created a stock prediction model to test these sequential models. We pulled Tesla stock prices' specifically close prices as it represents the market's overall sentiment of the day using yfinance api . then we created a dataset from stock prices and reshaped it into a 2d numpy array then applied MinMaxscaler to squash prices from -1 to 1. We used MinMaxscaler because it standardizes the log1p-transformed stock prices to a range of -1 to 1, which significantly improves the training stability, speed, and overall performance of the LSTM and GRU models by preventing issues related to unscaled input data.After that we created sequences from the data using the sliding window method after that we created the LSTM and GRU model extending on the pytorch's nn.lstm and nn.gru classes and trained hem on the sequential data and plotted the results confirming that GRU performs quite better and faster than LSTM on small datasets. These are the following practices we employed to further optimize the models.

Loss Function (MSE): We relied on Mean Squared Error. It strongly penalizes large prediction errors, which is crucial when working with high-value stock prices where big mistakes are very costly.

Optimizer (Adam): We selected the Adam optimizer because it's one of the most dependable choices for deep learning. It adjusts the "learning rate" on its own so the model avoids getting trapped in poor solutions or overshooting the optimum.

The 80/20 Split: We reserved 20% of the data as a completely unseen set during training. This "validation set" serves as a final exam to confirm that the model has learned general patterns rather than simply memorizing historical data (overfitting).

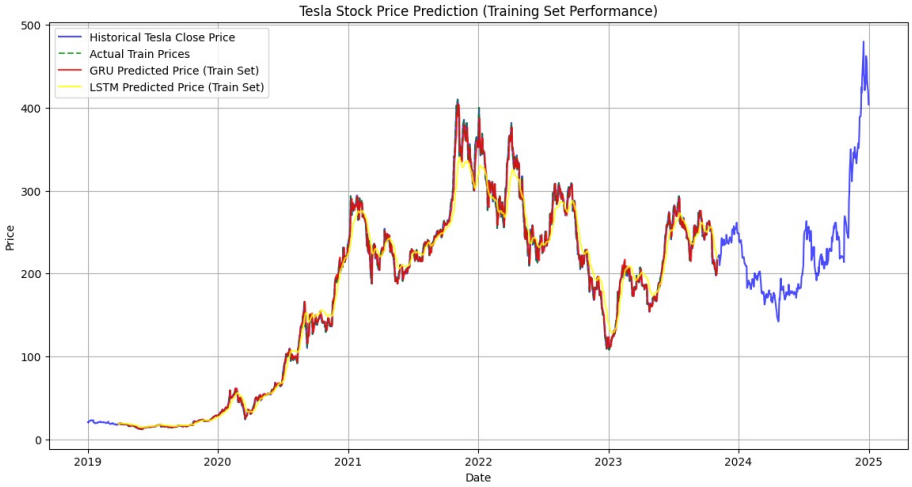
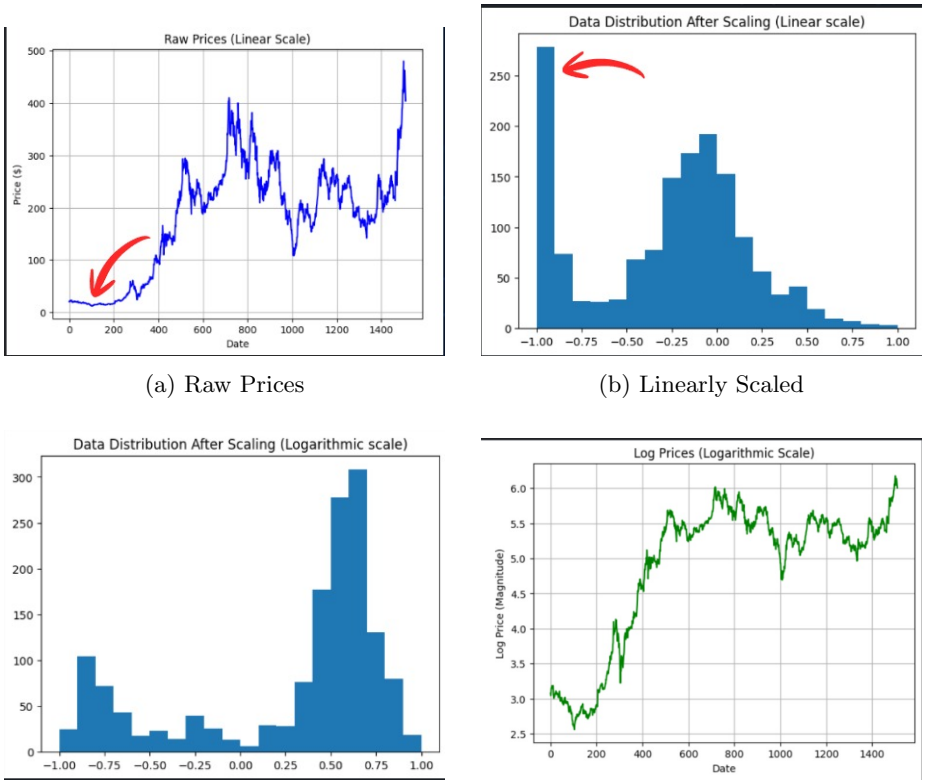


Figure 8: Comparison of GRU and LSTM model based training data

Why Log Scalling

In the preprocessing part we found a problem that model treats the difference between high prices and low prices the same but the these differences can be significant or miniscule based on percent changes ,To account for that we log scalled the prices thus caring about the ratios involved. For example: In 2019, Tesla fell from \$30 to \$20. That's a 33% drop — a huge crash! Panic! Yet in absolute terms, the price only moved by \$10. In 2024, Tesla slipped from \$400 to \$390. That's just a 2.5% decline — barely anything — but again the absolute change is \$10.

The issue is that, as far as our model is concerned, these two situations look exactly the same. It only registers "minus 10" in both cases. It treats the major 2019 crash as if it were a small, insignificant fluctuation and fails to recognize the crash pattern because the raw difference is too small.



(a) Raw Prices

(b) Linearly Scaled

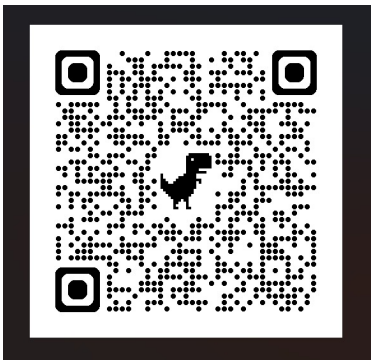
(c) Logarithmically Scaled

(d) Log Prices

Figure 9: Comparison of GRU and LSTM model performance across different price scalings.

Logarithms focus on ratios (multiplicative changes), not raw differences (additive changes). When we apply the log transform, the behavior of the numbers changes: 2019 crash ($30 to $20): ln(30) = 3.40 and ln(20) = 2.99, so the model observes a change of 0.41. 2024 dip ($400 to $390): ln(400) = 5.99 and ln(390) = 5.96, so the model observes a change of 0.03.

Now, the AI interprets the 0.41 shift as a large move (a real crash) and the 0.03 shift as a very small move (a minor dip).



3.6 References

1. **Colah's Blog:** *Understanding LSTM Networks*
   `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

2. **Medium:** *Stock Price Prediction with PyTorch*
   `https://medium.com/swlh/stock-price-prediction-with-pytorch-37f52ae84632`

4 Transformer Architecture and AGI

In this section, we analyze the architectural evolution from Recurrent Neural Networks (discussed in the previous section) to the **Transformer** model. Guided by resources from IBM and Hugging Face, we examined how this architecture has become the foundation for modern Generative AI.

4.1 Overcoming RNN Limitations

Our research highlights that the primary limitation of the RNN and LSTM architectures was their sequential nature. Processing tokens one by one prevented parallelization and led to the "vanishing gradient" problem, making it difficult to retain context over long sequences. The Transformer solves this by processing the entire sequence simultaneously using a mechanism called **Self-Attention**.

4.2 The Self-Attention Mechanism

We identified Self-Attention as the core innovation. The model assigns three vectors to every input token: a *Query (Q)*, a *Key (K)*, and a *Value (V)*.

- By calculating the dot product of the Q and K vectors, the model generates an **Attention Score**.
- This score determines how much focus the model places on other tokens when encoding the current word.

- Combined with **Positional Encodings** (to retain word order), this allows the model to capture long-range dependencies effectively.

4.3 The Path to Artificial General Intelligence (AGI)

Finally, we explored how Transformers serve as a stepping stone toward AGI. While current models (LLMs) are probabilistic predictors, the scale of these architectures is leading to "emergent" capabilities. The path to AGI involves moving beyond simple text generation to **Multimodal Systems** (integrating vision and audio) and **Agentic Workflows**, where the AI can reason, plan, and execute tasks autonomously rather than just responding to prompts.

5 References

1. **IBM Topics:** *What is a Neural Network?* `https://www.ibm.com/think/topics/neural-networks#741977106`
2. **IBM Topics:** *What is a Transformer Model?* `https://www.ibm.com/think/topics/transformer-model`
3. **IBM Topics:** *What is GPT?* `https://www.ibm.com/think/topics/gpt`
4. **Medium:** *Guide to Large Language Models with Hugging Face* `https://medium.com/@nimritakoul01`
5. **Vaswani et al. (2017):** *Attention Is All You Need.* (Foundational Paper).