Development from Neural Network to Stochastic Recurrent Neural Network

Andong Luo 205814040

Instructor: Dr. Cristina Stoica

Abstract

The neural network is widely used in many fields such as data prediction and image recognition among the world. While it returns fascinating results, the mathematical idea behind it is simple and straightforward. In this project, I am going to illustrate the fundamental deduction from the simple neural network, recurrent neural network, gate recurrent unit, variational autoencoder, and stochastic recurrent neural network. By investigating each of these neural network models, we can see how the stochastic recurrent neural network evolves to better fit for sequential data.

At last, I will present a practical achievement of what can be done by the neural network in the prediction of stock prices. The stochastic recurrent neural network shows a significantly better prediction for the prices of AAPL than the gate recurrent unit.

Contents

Introduction

Applied Statistics, especially in the field of machine learning, focuses more on training the model in different ways to get the most precise result. Other than the neural network, there are a few famous methods that provide good predictions for different needs and circumstances. For example, regression methods like lasso regression are able to not only estimate the trend of the data but also discard less important input factors. However, the neural network is unique due to its ability to completely using all data set and discovering latent connections between each input data automatically. For a chaotic system with very less information given, the neural network will produce a better result compared with other machine learning methods.

By years of development, the neural network has evolved into different formations and usages. It is not only a whole model used directly in predicting results but takes place in the foundational component of bigger algorithms and structures. In Computer Vision and Reinforcement Learning, the neural network is used to provide values for critical variables. As its importance raising, the neural network must improve its functions to complete different tasks, which is the reason why study for the neural network is still a hot research field in recent years.

Neural Network

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote the first paper about the neural network. It has a structure that simulates how the human brain works. Due to the development of modern computers, fitting a neural network model becomes possible.

Structure of Neural Network

Neuron

The basic component of the neural network is the neuron. Like what a real neuron's duty is in the human's brain, the neuron passes information from previous neurons to the next neurons. The information is changed in a specific way in order to transform the original information to what we need. For the simple neural network, this changing procedure is defined to be in form of linear regression: Each input will be multiplied by a weight vector and sum up with an additional constant residual. In addition, the linear regression formula needs also to be translated using an activation function to produce a suitable output.

$$h = f(wx + b)$$

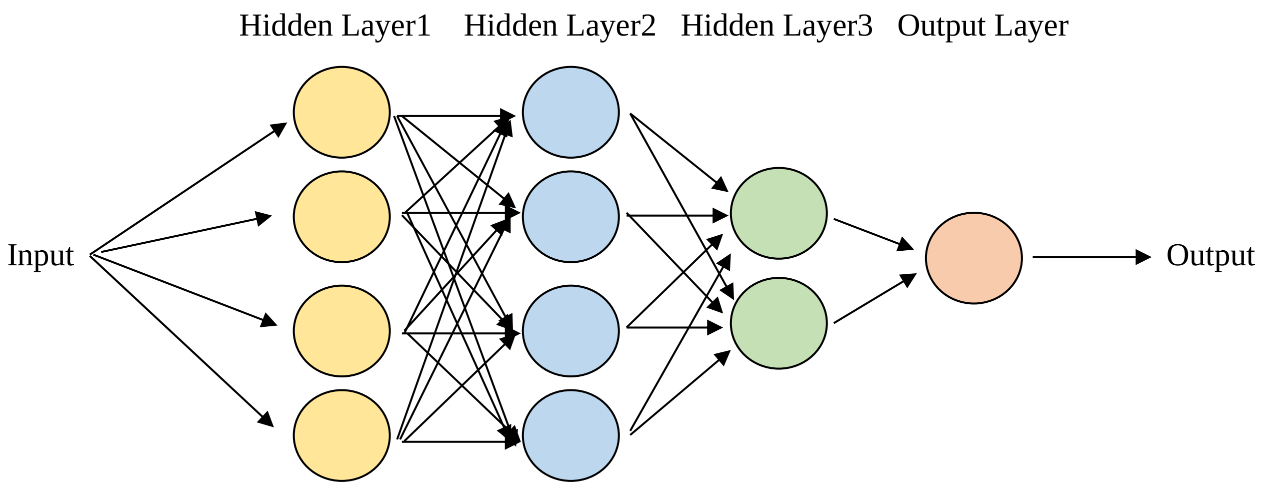
The choice of the activation functions often depends on what output is needed. For example, if we want an output value between 0 to 1 for classification, the sigmoid function is the common option for the activation function, which returns a variable with a range 0-1:

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

By defining different operations within neurons, the whole system can be modified to recurrence neural network and gate recurrent unit which will be mentioned later in this project.

Layer

Layers are modified to link up all neurons. For each layer, its size determines how many neurons will be created. The more neurons for layers to have, the more sophisticated and time-consuming for the whole system. When the inputs go through one layer, the layer will create a number of outputs as the inputs for the next layer. Notice that the dimension of each layer can be different because the whole input will go through every single neuron rather than go through the layer in a one-to-one structure. Therefore, by controlling the dimensions, we can not only construct different structures for the whole neural network we want but also decide how many variables we need to be returned in the end. Figure 1 is a demonstration of a 4 layers neural network structure with numbers of neurons 4, 4, 2, and 1 respectfully.



(Figure 1)

Training Procedure

Loss function

There is one common procedure for all machine learning methods: calculate the loss. All models need a way to understand their precision and what to change to make themselves better. The loss function represents the difference between the real output and the predicted output, so the less the loss is, the better the model. The most common loss functions for the neural network are mean squared error and binary cross-entropy.

MSE (mean squared error) is used in producing quantity values:

$$mse = \frac{1}{n}\sum(y - y_{pred})$$

While binary cross-entropy is used for classification for two classes 0 and 1:

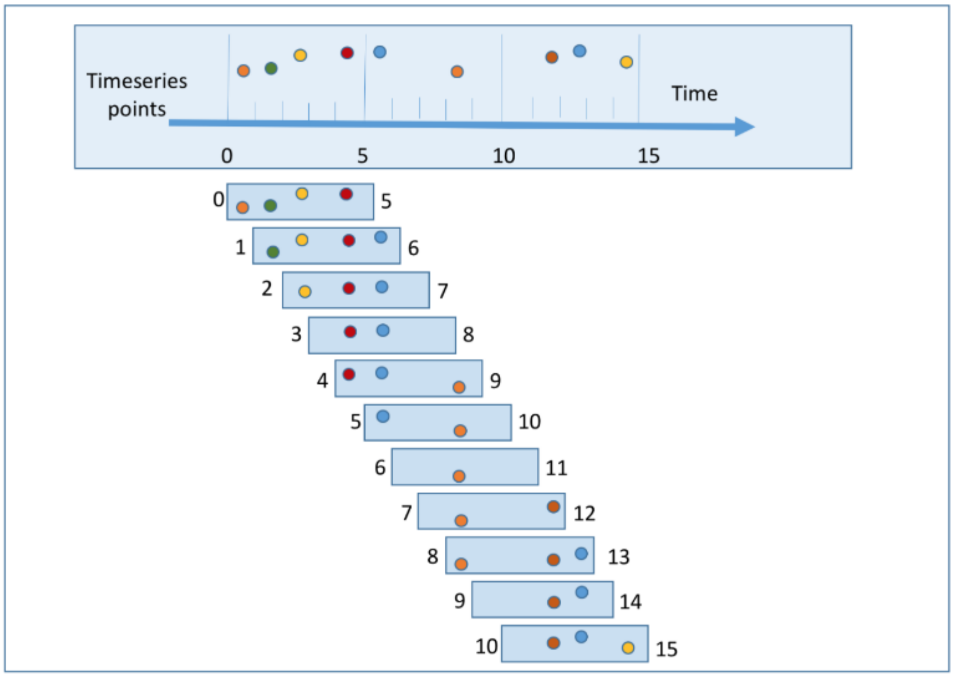
$$H_y = ylog\left(y_{pred}\right) + (1 - y)\log\left(1 - y_{pred}\right)$$

$$y = 0{,}1$$

Validation

To update variables for each neuron, a comparison method for validating is needed. The simplest method is to divide the whole data set into 3 subsets: training set, validation set, and prediction set. The goal for the training set and validation set is to build up the model. The training set will be used to predict the validation set in order to find the suitable value for each variable. By predict the prediction set using the model, we can generate a ratio like MSE to see how good our model is. Usually, to produce a better model, the size of the data set is required to be as large as possible. Therefore, it is more practical to validate the model multiple times when having a large data set. A method called rolling windows is designed to fulfill this requirement.

Rolling windows is a loop-based validation method. First, we need to decide what is the size of each window. If we assume the size for the training set is 5, and the size for the validation set is 2, then the first 5 data will be used to predict the next 2 data in the first loop. For the next loop, we discard the first data point and take the next 2-6 data points to predict data numbers 7 and 8. The size of the training set is called the time step, which in this case, the time step is 5. By continuedly discarding the data points, finally, we will predict the last two data and end this loop. Inside each loop, we will update the variables 1 time. Figure 2 is another demonstration of how rolling windows working in the graph.



(Figure 2)

For time-series data, it is also important to normalize it before fitting models. Data before normalization will have trends and different variances, whether observable or not.

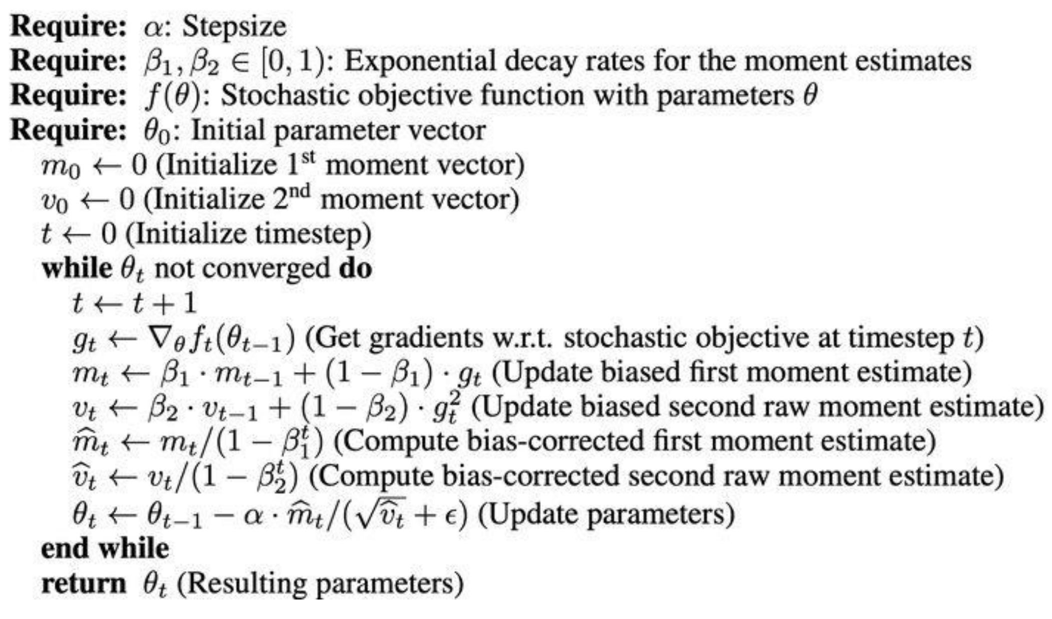
Gradient Descent

The next step is about using the derivative of the loss function with respect to each constant variable. For each loop of the validation, each constant will be updated by minus a multiplication of a learning rate and the derivative.

$$w = w - ƀ\frac{dL}{dw}$$

Where ƀ represents a learning rate which controls how fast the w will descend.

Notice that if the derivative is positive, then it means that the loss function is increasing, therefore minus this derivative will decrease the loss function. If the derivative is negative, then the loss function is decreasing, so minus this derivative will still decrease the loss function by increasing the constant. Since all variables will always go deep until they converge, this method is called gradient descent. The ADAM method is a gradient descent method with more efficiency and speed. Figure 3 is the algorithm of the ADAM method.



```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

(Figure 3)

Now the remaining question would be how can we evaluate this derivative.

Backpropagation

If we name the procedure from feeding the input through the layers and getting the output to be the forward propagation, then the backpropagation is a backward inference of the derivative with respect to the loss function. Take the simple neural network as an example: Assume we want to know the derivative of the weight variable inside the neuron with loss function mean-square error:

$$\frac{dL}{dw} = \frac{dL}{dy} * \frac{dy}{dh} * \frac{dh}{dw}$$

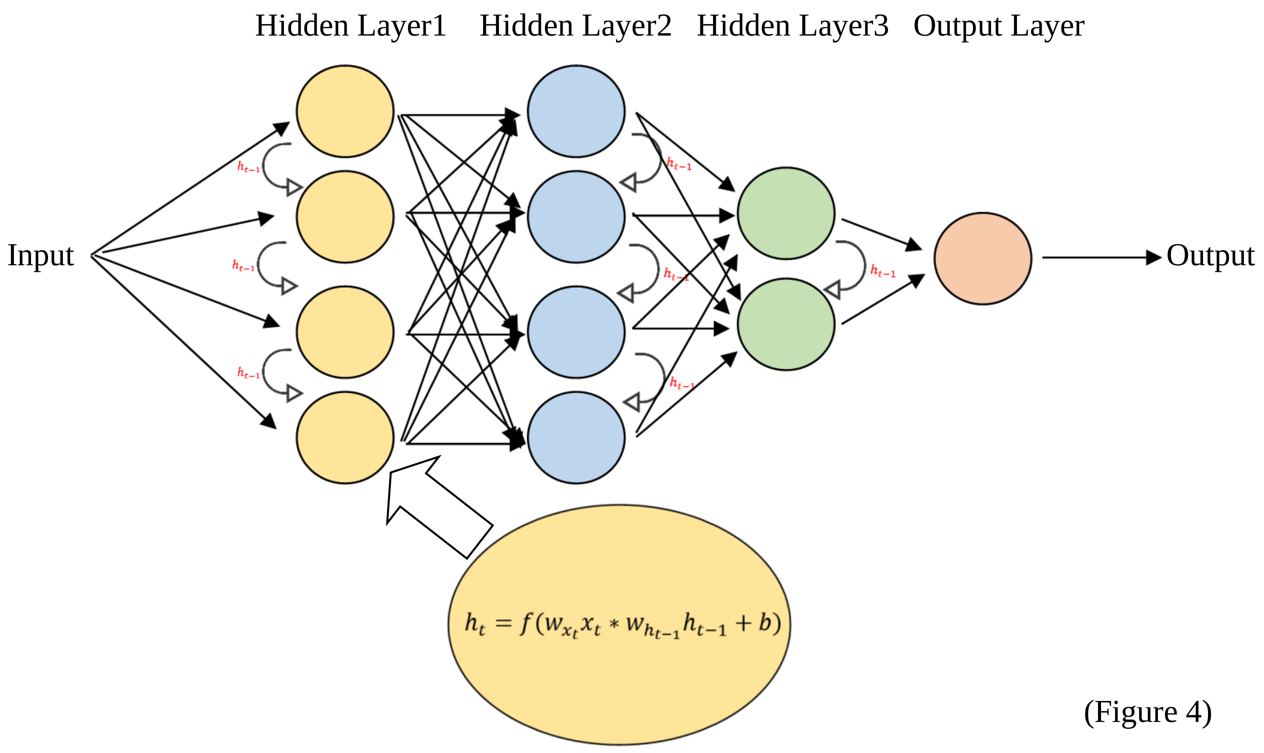
From the equation, we get $\frac{dL}{dw}$ from each neuron to the output and to the loss function backwardly, and this is the reason why it is called backpropagation.

Recurrent Neural Network

For sequential data or time-series data, one of the biggest issues for the simple neural network is the memoryless problem. Since all neurons in a single layer are independent, the simple neural network cannot memorize information but only treat each of the data in a sequence independently. The recurrent neural network is created to solve this issue.

Structure

The whole structure for the RNN is the same as the simple neural network, but the difference is in each neuron. The neuron for the RNN includes another variable, which is the output from the previous neuron. RNN is named recurrent because of this feature that the last output will always be considered in the calculation of the next neuron. Figure 4 is a demonstration of the RNN.



(Figure 4)

Backpropagation

Since each of constants for the output h is related to the previous neuron, it is a more complicated procedure to get the value of its derivative. Take the constant for the first h as an example. To get $\frac{dL}{dw_{h_1}}$, the equation is:

$$\frac{dL}{dw_{h_1}} = \frac{dL}{dy} * \frac{dy}{dh_t} * \frac{dh_t}{dh_{t-1}} * \dots * \frac{dh_2}{dh1} * \frac{dh_1}{dw_{h_1}}$$

$$= \frac{dL}{dy} * \frac{dy}{dh_t} * \prod_1^t \frac{dh_n}{dh_{n-1}} * \frac{dh_1}{dw_{h_1}}$$

By knowing $h_t = \sigma(w_{x_t}x_t + w_{h_{t-1}} * h_{t-1} + c_t)$, then:

$$\frac{dh_t}{dh_{t-1}} = \sigma'\left(w_{x_t}x_t + w_{h_{t-1}} * h_{t-1} + c_t\right) * w_{h_{t-1}}$$

$$\frac{dL}{dw_{h_1}} = \frac{dL}{dy} * \frac{dy}{dh_t} * \prod_2^t \frac{dh_n}{dh_{n-1}} * \frac{dh_1}{dw_{h_1}}$$

$$= \frac{dL}{dy} * \frac{dy}{dh_t} * \prod_2^t \sigma'\left(w_{x_t}x_t + w_{h_{t-1}} * h_{t-1} + c_t\right) * w_{h_{t-1}} * \frac{dh_1}{dw_{h_1}}$$

Gradient Vanishing and Exploding

Assuming any of the $w_{h_{t-1}}$ is very large or very small, then $\frac{dL}{dw_{h_1}}$ is inevitably becomes very large or close to 0. The problem of a very large $\frac{dL}{dw_{h_1}}$ is called gradient exploding which will cause the update of each step will be too large. The problem of $\frac{dL}{dw_{h_1}}$ close to 0 is called gradient vanishing. The harm for gradient vanishing is more serious because the update may be close to 0 so that no significant learning will be done in a reasonable time. The gated recurrent unit is one of the methods which can solve these issues.

Gated Recurrent Unit

In 2014, Kyunghyun Cho et al. introduced the gated recurrent unit for the first time. It is known along with another similar method called LSTM (long short-term memory), which has an additional output gate.

Structure

Similar as the RNN, GRU (Gated Recurrent Unit) has the same structure but different neurons. For each neuron, GRU added a reset gate, and an update gate. The reset gate creates a h' using reset previous h. The update gate generates the new h using h' and the previous h to determine which part to be updated. Below is the demonstration of the neuron for the GRU.

$$u = \sigma\,(wx_t + wh_{t-1} + b)$$

$$r = \sigma\,(wx_t + wh_{t-1} + b)$$

$$h'_t = tanh\,(wx_t + rwh_{t-1} + b)$$

$$h_t = (1-u)h'_t + uh_{t-1}$$

Backpropagation

Similarly, for $\frac{dL}{dw_{h_1}}$:

$$\frac{dL}{dw_{h_1}} = \frac{dL}{dy} * \frac{dy}{dh_t} * \prod_{2}^{t} \frac{dh_n}{dh_{n-1}} * \frac{dh_1}{dw_{h_1}}$$

However, the gradient for GRU is different from RNN due to the reset gate and the update gate.

$$h_t = (1-u)h'_t + uh_{t-1} = h'_t - uh'_t + uh_{t-1}$$

$$\frac{dh_t}{dh_{t-1}} = {h'_t}'\sigma'_r w_{h'_t} w_r - u' w_u h'_t - u{h'_t}'\sigma'_r w_{h'_t} w_r + u + u' w_u h_{t-1}$$

$$= ({h'_t}'\sigma'_r) w_{h'_t} w_r + u'(h_{t-1} - h'_t) w_u + u\left(1 - {h'_t}'\sigma'_r w_{h'_t} w_r\right)$$

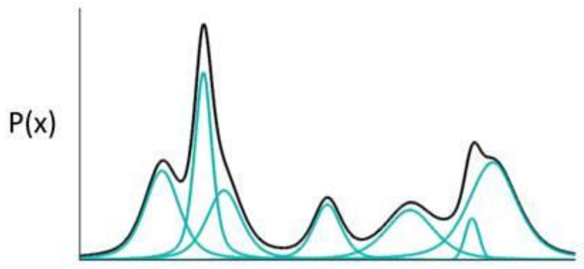
$$= u'(h_{t-1} - h'_t) w_u + (1-u)({h'_t}'\sigma'_r) w_{h'_t} w_r + u$$

u acts as an important role here. Since u is a number between 0 and 1, $\frac{dh_t}{dh_{t-1}}$ will never be a very large number or a number close to 0. Therefore, we prevent the gradient vanishing and exploding problem successfully.

Variational Autoencoder

The autoencoder is a generative method that is often used for reproducing similar graphs. It has two parts: the encoder and the decoder. The idea behind any Autoencoder system is to use the encoder to extract important information and save it in a lower dimension, then use the decoder to generate different output related to the input. If we mapping each of the points in a graph to an x-y axis quadrant, then it may have n*m dimensions. What autoencoder does is to transfer these n*m dimensions to lower dimensions like 2, and then use the 2 dimension's information to create another graph with n*m dimensions which has something in common with the previous graph.

The main goal of the autoencoder is to find a p(x) so that we can know the probability for every point x in a sequence and generate new graphs with respect to p(x). However, in real life, it is nearly impossible to find this probability. Instead of finding p(x), the VAE (variational autoencoder) looks into $p(x) = \int_z p(x|z)p(z)\,dz$ for a latent distribution z, which is usually a normal distribution, so that for every number x, there will be a z projected into it to get $p(x|z)$. The idea behind it is every probability p(x) can be reformed into a combination of multiple normal distributions as shown in the figure 5.

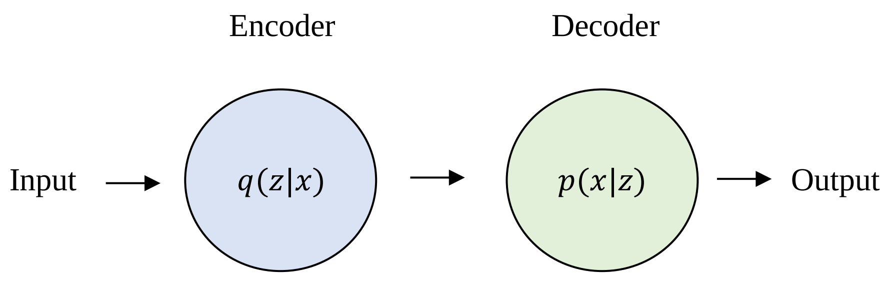


(Figure 5)

Notice that if we assume $z \sim N(0, I)$, then $p(x|z) \sim N(\mu(z), \sigma(z))$.

Structure

As the autoencoder, VAE has an encoder and a decoder. The purpose of the encoder now is to generate $\mu(z)$ and $\sigma(z)$, so that the decoder can use them to simulate $p(x|z)$. A new distribution $q(z|x)$ is introduced for this purpose. It means that a normal distribution z is generated for each of the x in the data. Neural Network also contributes a lot in the process of generating $q(z|x)$ and $p(x|z)$. The encoder and the decoder will be two simple neural networks. The structure is shown in the figure 6.

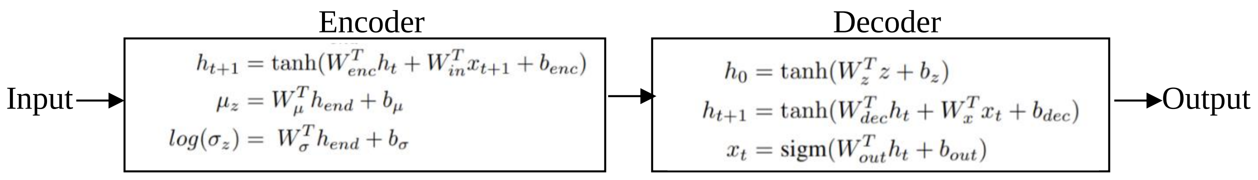


(Figure 6)

Reparameterization Trick

By knowing VAE has neural network structures, generating a normal distribution directly will be hard because of its involvement in gradient descent. The paper shows an alternative way to generate $q(z|x)$ and it is called the reparameterization trick. The idea is to simulate μ and σ directly from a neural network, then write $z = \mu + \sigma * \epsilon$ where $\epsilon \sim N(0, I)$ is a noise variable. Therefore, the gradient descent is only about two random variables μ and σ instead of a gaussian variable z.

The final structure is shown in the figure 7:


Encoder
Decoder
Input
$h_{t+1} = \tanh(W_{enc}^T h_t + W_{in}^T x_{t+1} + b_{enc})$
$\mu_z = W_\mu^T h_{end} + b_\mu$
$log(\sigma_z) = W_\sigma^T h_{end} + b_\sigma$
$h_0 = \tanh(W_z^T z + b_z)$
$h_{t+1} = \tanh(W_{dec}^T h_t + W_x^T x_t + b_{dec})$
$x_t = \mathrm{sigm}(W_{out}^T h_t + b_{out})$
Output

(Figure 7)

Loss Function

The loss function is different from an MSE for VAE due to its complicated structure. Since we aim to find p(x) in the beginning, we use the maximum log-likelihood function to get the loss function. The bigger the MLE (maximum log-likelihood) is, the better our model will be.

$$MLE = \max L = max \sum_{z} \log\left(p(x)\right)$$

$$\log\big(p(x)\big) = \int_z q(z|x) \log\big(p(x)\big)\, dz = \int_z q(z|x) \log \frac{(p(z,x)q(z|x))}{q(Z|X)p(Z|X)} dz$$

$$= \int_z q(z|x) \log \frac{p(z,x)}{q(Z|X)} dz + \int_z q(z|x) \log \frac{q(z|x)}{p(Z|X)} dz$$

$$= \int_z q(z|x) \log \frac{p(x|z)p(z)}{q(Z|X)} dz + \int_z q(z|x) \log \frac{q(z|x)}{p(Z|X)} dz$$

$$= \int_z q(z|x) \log \frac{p(z)}{q(Z|X)} dz + \int_z q(z|x) \log p(x|z)\, dz + \int_z q(z|x) \log \frac{q(z|x)}{p(Z|X)} dz$$

The first term is a Kullback-Leibler divergence: $-KL(q(z|x)||p(z))$, which is equal to $\sum(1 + \log(\sigma^2)) - \mu^2 - \sigma^2)$

The second term $\int_z q(z|x) \log p(x|z)\, dz$ is equal to $\frac{1}{L}\sum_L \log\left(p(x|z)\right)$

The last term $\int_z q(z|x) \log \frac{q(z|x)}{p(Z|X)} dz$ is also a Kullback-Leibler divergence: $KL(q(z|x)||p(z|x))$, which is always greater or equal to 0. Because what we want is the maximum, this term will not be considered in the loss function.

Finally, the loss function is:

$$\mathcal{L}(\theta; \mathbf{x}^{(i)}) \simeq \sum_{j=1}^{J} (1 + \log((\sigma^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L} \log p(x^{(i)}|z^{(i,l)})$$

Stochastic Recurrent Neural Network

SRNN (Stochastic Recurrent Neural Network) is a combination of VAE and GRU. Instead of using neural networks in the encoder and decoder, SRNN uses GRU to improve the performance of VAE in time series data like stock prices or weather information. It is proved that the model can be used in sophisticated deep learning models for sequential modeling.

Structure

Just as VAE, SRNN shares the same loss function and model structure except that each neuron is in format of GRU:

Encoder:

$$u \ = \sigma(Wy + Cz + Mh + b)$$

$$r \ = \sigma(Wy + Cz + Mh + b)$$

$$h' \ = \ tanh\ (Wy + Cz + rMh + b)$$

$$h_t = u_t h_{t-1} + (1 - u_t) h_t^{'}$$

$$\mu_z = W_\mu^T h_t + b_\mu$$

$$\log\left(\sigma_z\right) = W_\sigma^T h_t + b_\sigma$$

Decoder:

$$\begin{aligned}
\boldsymbol{u}_t &= \sigma(\boldsymbol{W}_u \cdot \boldsymbol{x}_t + \boldsymbol{C}_u \cdot z_t + \boldsymbol{M}_u \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_u) \\
\boldsymbol{r}_t &= \sigma(\boldsymbol{W}_r \cdot \boldsymbol{x}_t + \boldsymbol{C}_r \cdot z_t + \boldsymbol{M}_r \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_r) \\
\tilde{\boldsymbol{h}}_t &= tanh(\boldsymbol{W}_h \cdot \boldsymbol{x}_t + \boldsymbol{C}_h \cdot z_t + \boldsymbol{r}_t \odot \boldsymbol{M}_h \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \\
\boldsymbol{h}_t &= \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \odot \tilde{\boldsymbol{h}}_t,
\end{aligned}$$

The algorithm for model prediction is showed in the figure 8.

Algorithm 1 Prediction algorithm for stochastic GRU

```
Input: $\tau, \boldsymbol{h}_{last}, \boldsymbol{x}_{T+1:T+\tau}$
Output: $y_{T+1:T+\tau}$
1: for $t \leftarrow 1$ to $\tau$ do
2:     $z_t \sim p_{\theta_3}(z_t|\boldsymbol{h}_{last})$
3:     $\boldsymbol{h}_t \leftarrow GRU(\boldsymbol{h}_{last}, z_t, \boldsymbol{x}_t)$
4:     $y_t \sim p_{\theta_1}(y_t|\boldsymbol{h}_t)$
5:     $\boldsymbol{h}_{last} \leftarrow \boldsymbol{h}_t$
6: end for
```

(Figure 8)

Stock Price Prediction

The stock market is a chaotic system with very limited information for prediction. Normally investors trade stocks based on the report from companies but not the current prices. The only thing assured to happen by knowing the underlying price changes for each day is what is the beginning price of tomorrow. In addition, the stock price is easy to be manipulated by capital. The stock prices are impossible to predict due to their complexity and the human power behinds them. However, with increasing computational power, deep learning methods have more potential to solve the mystery of the stock market.

It has been shown that SRNN provides a significantly better prediction than the AR model and the LSTM model. In this project, I will use SRNN to predict the stock prices of AAPL.

Data

The data using for the project is the AAPL stock prices data from yahoo finance. It is a clean resource so that it has no missing values or any type of errors. The project will use the adjusted close prices for analysis, which is the real stock prices at each end of the day. The data is selected from the beginning of 2013 to the end of 2020 and separated into training data and test data, which the training data is from 2013 to 2019, and the test data is after 2019.

The training data was used by the method of rolling windows. The time steps are 2 and the validation data has the same size 2.

Normalization is also important to keep the whole data in the same structure. The data is set to have mean 0 and variance 1.

Model Structure

The time step for the model is 2, and the validation size is also 2. The number of features is 1 because the prices are the only data we input. The dimension for the latent variable z is 50. The dimension for each GRU layer is 64.

Just as any SRNN, the stock price prediction model has an encoder and a decoder with 7 layers respectfully.

Encoder:

Input (sample size, 2, 1) → GRU (sample size, 2, 64) × 3 → GRU (sample size, 64)

→ μ = Dense (sample size, 50), σ = Dense (sample size, 50), Z = Lambda (sample size, 50)

The input is in format of sample size, time steps, and number of features. The dimension of the input changes from (sample size, 2, 1) to (sample size, 50), which is the dimension for z.

Decoder:

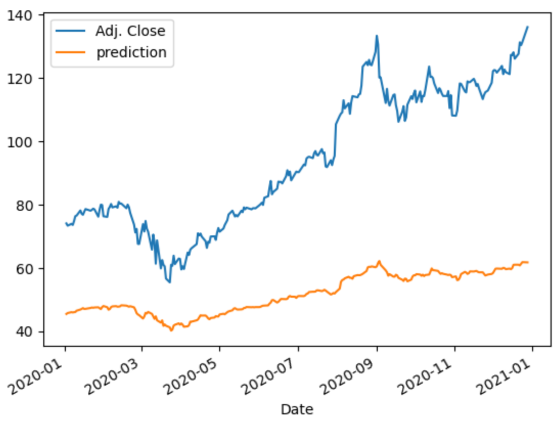
→Repeat_Vector (sample size, 2, 50) → GRU (sample size, 2, 64) x3 → GRU (sample size, 2, 1)

→ Time_Distributed (sample size, 2, 2) → Output = Time_Distributed (sample size, 2, 1)

In this process the dimension has change back to (sample size, 2, 1) which is the same as the input.

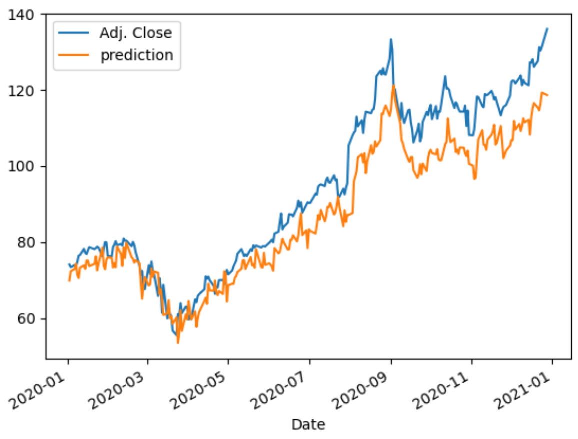
Result

First, a 4-layers GRU model with 50 neurons each is created for comparison. The MSE for the GRU model is 2030.5582. The result is showed in the figure 9, which is overall not a great prediction.



(Figure 9)

On the contrary, the SRNN model provides a good estimation at the beginning of the year. It reduces the precision after, but still describes the trend successfully. The MSE for the SRNN model is 75.831436 and the result is showed in the figure 10.



(Figure 10)

Although the graph and MSE seems to be a good result for SRNN, it still cannot prove that SRNN can predict the stock price perfectly. The prediction procedure for the neural network is an updating process, which means that the model uses the actual 2-day values to predict the next two future predictions, instead of what it has predicted previously. This is the reason why we can see the predicted value is a few days later than the real value. There is still space for improvement of the model for stock prices.

Conclusion

With the development of computational power, more possible machine learning models are available from the paper. The gradient descent and backpropagation had been calculated long before 1943, but the neural network model can be used widely all over the world in just recent decades. The development also inspires the understanding of problems like gradient vanishing and exploding. To adapt to more circumstances in real life, the neural network also changed itself to different structures like recurrent neural networks and variational autoencoders. In modern researches, a combination of those methods gives us more possibility for more precise models like the stochastic recurrent neural network. This project provides an example of the stochastic recurrent neural network, which provides a better prediction than the gated recurrent unit. In the future, neural networks will still be a hot research field. We look forward to more research results that can make human predictions of unknown events more and more accurate.

Reference

1. Luo A. (2021) Stochastic-RNN https://github.com/andongluoo/Stochastic-RNN

2. *Using moving and tumbling windows to highlight trends*. Using Moving and Tumbling Windows to Highlight Trends | Tanzu Observability Documentation. (n.d.). https://docs.wavefront.com/query_language_windows_trends.html.

3. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, & Yoshua Bengio. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

4. Otto Fabius, & Joost R. van Amersfoort. (2015). Variational Recurrent Auto-Encoders.

5. Diederik P. Kingma, & Jimmy Ba. (2017). Adam: A Method for Stochastic Optimization.

6. Diederik P Kingma, & Max Welling. (2014). Auto-Encoding Variational Bayes.

7. Koehrsen, W. (2018, November 5). *Recurrent neural networks by example in python*. Medium. https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470.

8. Victor Zhou. (n.d.). *Machine learning for beginners: An introduction to neural networks*. Victor Zhou. https://victorzhou.com/blog/intro-to-neural-networks/.

9. Brownlee, J. (2021, January 21). *How to choose an activation function for deep learning*. Machine Learning Mastery. https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.