CSCE566-DATA MINING
WEEK 6

Time-Series Data Mining: Recurrent Neural Networks

Min Shi
min.shi@louisiana.edu

Sep 30, 2024

Outline

- Introduction & Recap
- Vanilla Recurrent Neural Networks (RNNs)
- Challenges in Vanilla RNNs: Gradient Exploding and Vanishing Long
- Short-Term Memory Networks (LSTM)
- Example Application of LSTM: Forecasting Time-Series Gene
- Expression Gated Recurrent Units (GRUs)

Fixed-Sized Input and Output

The learning paradigm of (Multiple-Layer Perceptron) MLP:

1. Different data samples are independently fed into the model.
2. The sizes of input and output are invariant (e.g., input images have the same resolution).

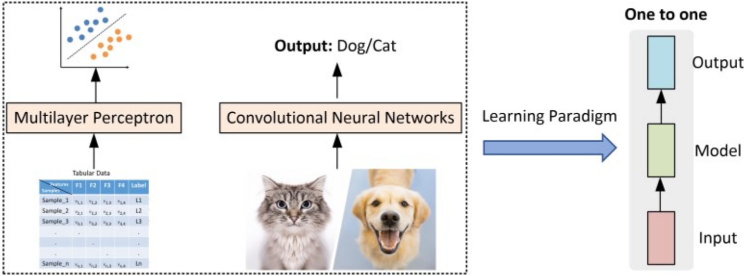


Figure: For MLP and CNN, the input and output are static and fixed-sized.

The limitation of MLP

Training: Feed adequate samples to the neural network (e.g., MLP) to train a prediction model:

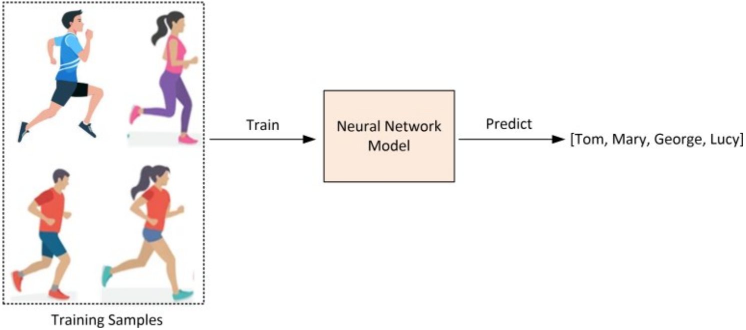


Figure: Train a neural network model to predict who is running.

The limitation of MLP

Prediction: Provide an example to the pretrained model:

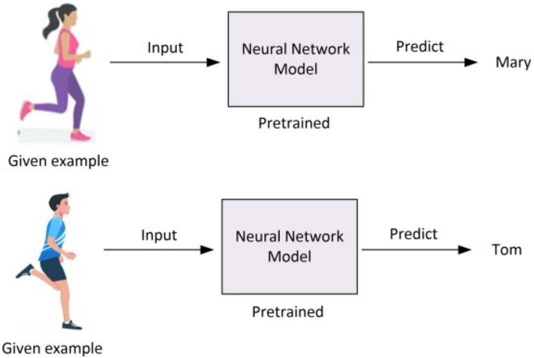


Figure: Use the pretrained neural network model to predict who is running.

The limitation of MLP

Prediction: The pretrained model is also generalizable and tolerant to unseen data:

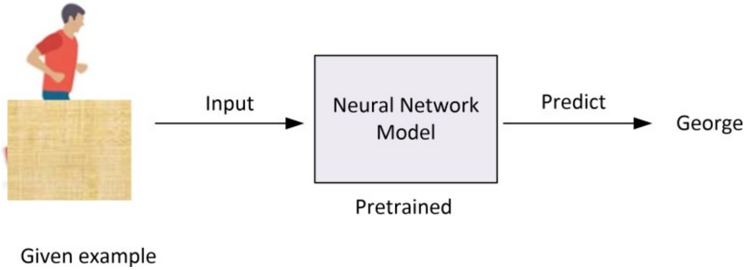


Figure: Use the pretrained neural network model to predict who is running.

The limitation of MLP

Prediction: Provide an example to the pretrained model:

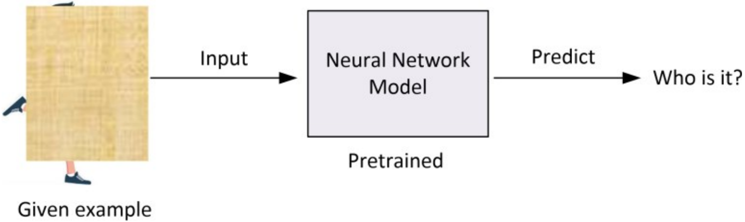


Figure: Use the pretrained neural network model to predict who is running.

The limitation of MLP

Prediction: In many scenarios, we have the longitudinal information of data:

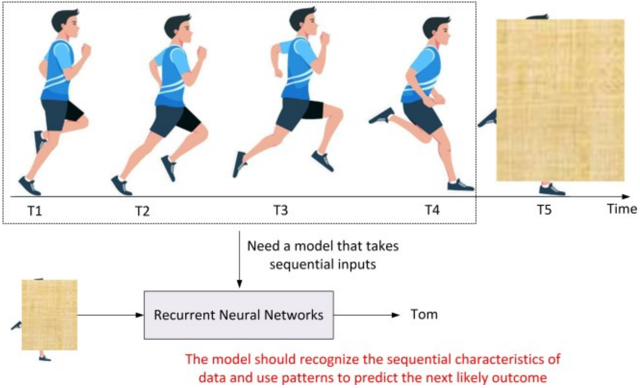


Figure: The recurrent neural networks (RNN) address the limitation of MLP.

RNN Architecture is Flexible

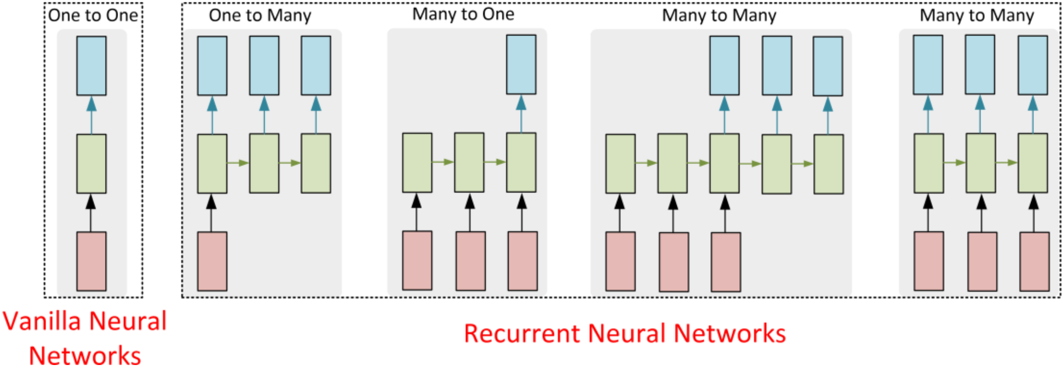


Figure: Various learning architectures of recurrent neural networks.

Sequential Applications: One-to-Many

Example: image captioning

- **Input**: fixed-size
- **Output**: Length-variable sequence

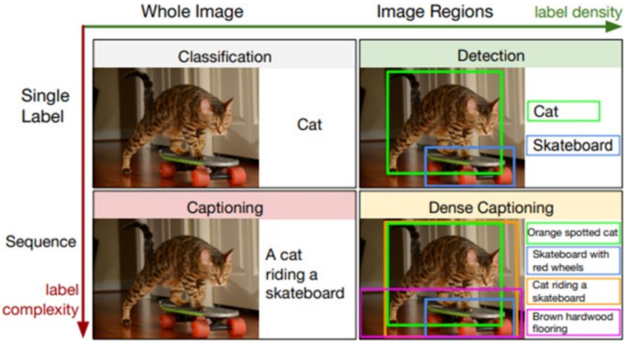


Figure: https://centricconsulting.com/blog/sentiment-analysis-way-beyond-polarity/.

Sequential Applications: Many-to-One

Example: sentiment analysis

- **Input**: Length-variable sequence
- **Output**: fixed-size

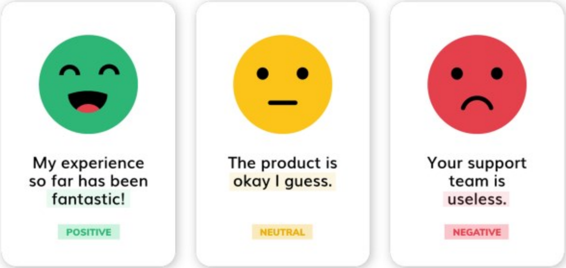


Figure: https://palashc.github.io/project/dense/.

Sequential Applications: Many-to-Many

Example: language translation

- **Input**: Length-variable sequence
- **Output**: Length-variable sequence

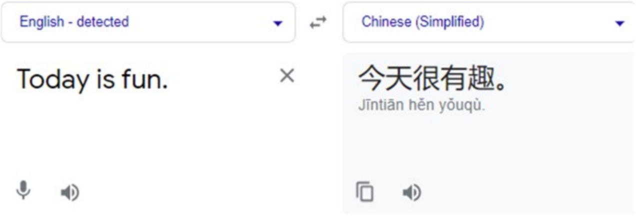


Figure: Translation from English to Chinese.

Online Demo: Sketch-RNN

Once you start drawing an object, Sketch-RNN will come up with many possible ways to continue drawing this object based on where you left off.

https://magenta.tensorflow.org/assets/sketch_rnn_demo/multi_predict.html

Vanilla Recurrent Neural Networks (RNNs)

- **Feedforward Network**: It receives an input and generates an output.
- **Recurrent Network**: It receives the input at current time step and the output from previous time step, and generates an output.

The main idea is to use a hidden state to capture information about the past.

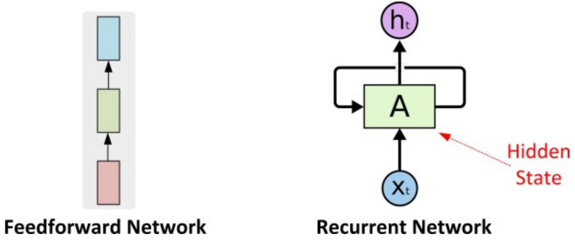


Figure: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The main idea is to use a hidden state to capture information about the past.

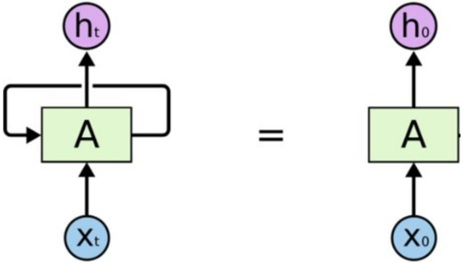


Figure: Unrolled recurrent neural networks. The input at time step 1.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The main idea is to use a hidden state to capture information about the past.

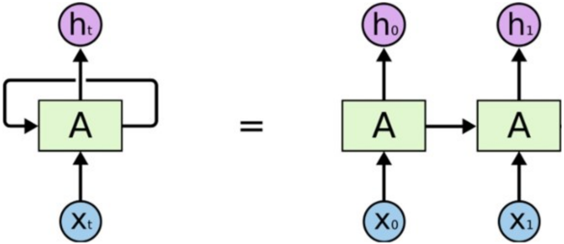


Figure: Unrolled recurrent neural networks. The input at time step 2.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The main idea is to use a hidden state to capture information about the past.

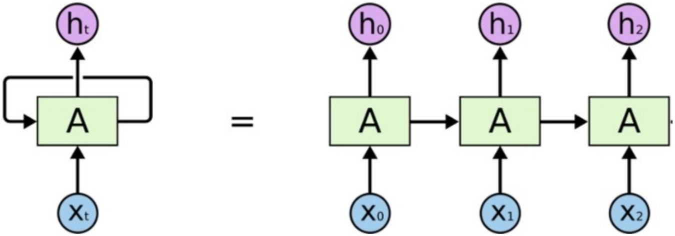


Figure: Unrolled recurrent neural networks. The input at time step 3.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The main idea is to use a hidden state to capture information about the past.

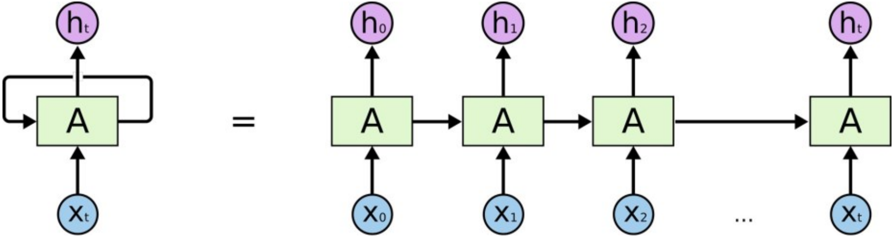


Figure: Unrolled recurrent neural networks. The input at time step *t*.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The weight parameters are shared at different time steps.

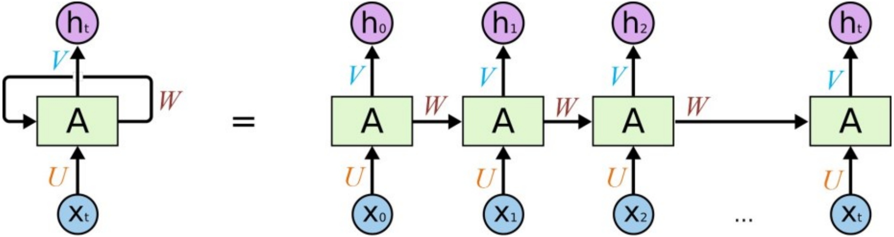


Figure: Unrolled recurrent neural networks. The input at time step *t*.
https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Vanilla Recurrent Neural Networks (RNNs)

The formulation of RNNs:

$$h_t = \phi(Wh_{t-1} + UX_t) \tag{1}$$

$$y_t = Vh_t \tag{2}$$

- X_t is the input at time t;
- h_{t-1} is the hidden state at previous time $t-1$;
- h_t is the hidden state at time t;
- y_t is the output at time t;
- W, U, V are model weights shared at all time steps;
- ϕ is the Tanh activation function.

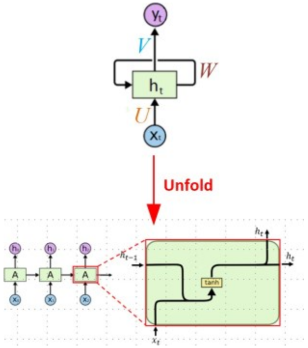


Figure: Recurrent neural networks with loop.

Training RNNs: Backpropagation Through Time (BPTT)

- For every time step t, the output error e_t will be backpropagated throgh all the previous steps.
- Compute partial gradients $\frac{\partial e_t}{\partial U}, \frac{\partial e_t}{\partial V}, \frac{\partial e_t}{\partial W}$; update weights, e.g., $U \leftarrow U - \eta \sum \frac{\partial e_t}{U}$

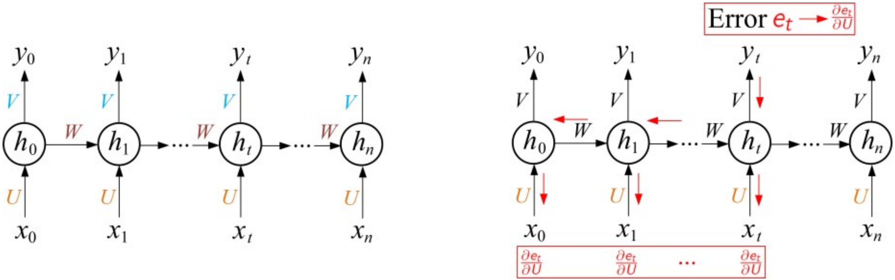


Figure: U, V, W are weight parameters. BPTT is a gradient training strategy for recurrent neural network.

Training RNNs: Backpropagation Through Time (BPTT)

Compute partial derivative $\frac{\partial e}{\partial U}$ at each time step:

At time n, $\frac{\partial e}{\partial U} = \frac{\partial e}{\partial y_n} \times \frac{\partial y_n}{\partial h_n} \times \frac{\partial h_n}{\partial U}$.

$n-1$, $\frac{\partial e}{\partial U} = \frac{\partial e}{\partial y_n} \times \frac{\partial y_n}{\partial h_n} \times \frac{\partial h_n}{\partial h_{n-1}} \times \frac{\partial h_{n-1}}{\partial U}$

...

1, $\frac{\partial e}{\partial U} = \frac{\partial e}{\partial y_n} \times \frac{\partial y_n}{\partial h_n} \times \frac{\partial h_n}{\partial h_{n-1}} \times \ldots \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial U}$

0, $\frac{\partial e}{\partial U} = \frac{\partial e}{\partial y_n} \times \frac{\partial y_n}{\partial h_n} \times \cdots \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial h_0} \times \frac{\partial h_0}{\partial U}$

The total $\frac{\partial e}{\partial U}$ is represented as sum-of-product:

$$\frac{\partial e}{\partial U} = \sum_{0}^{n} \frac{\partial e}{\partial U}$$

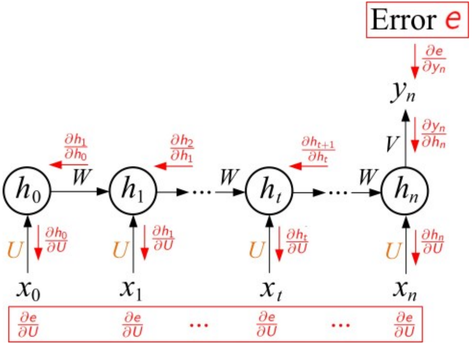


Figure: Compute gradients backward through the sequence.

Training RNNs: Backpropagation Through Time (BPTT)

- Forward through entire sequence to compute the total prediction loss.
- For each time t, backward through all dependent steps to compute gradients.

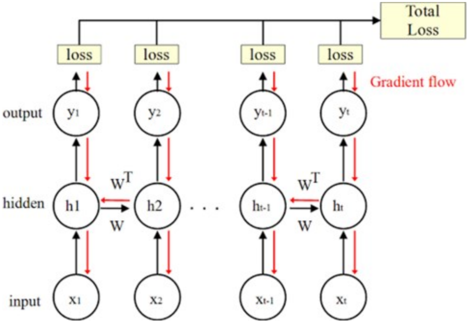
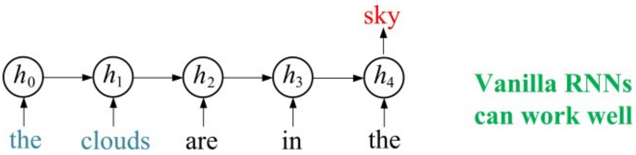


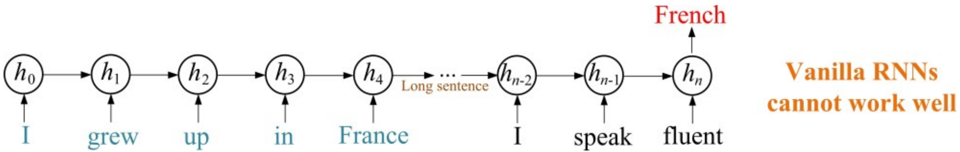
Figure: Multiple outputs. BPTT algorithm is a gradient training strategy for recurrent neural network. https://sonsnotation.blogspot.com/2020/11/10-recurrent-neural-networkrnn.html.

The Problem of Long-Term Dependencies

Short-Term Dependencies: Require recent context to perform the prediction.



Long-Term Dependencies: Require distant context to perform the prediction.



Vanishing Gradients

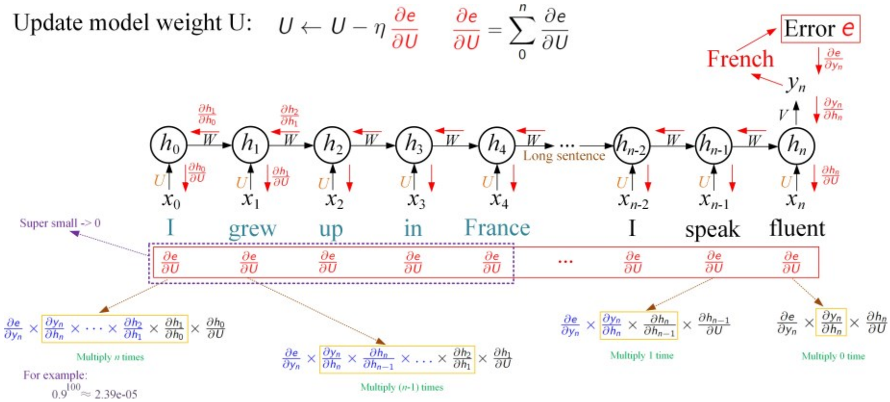


Figure: Gradient signal can end up being multiplied many times. Long term components goes exponentially fast to norm 0, meaning that distant inputs have no contribution to the final prediction.

Exploding Gradients

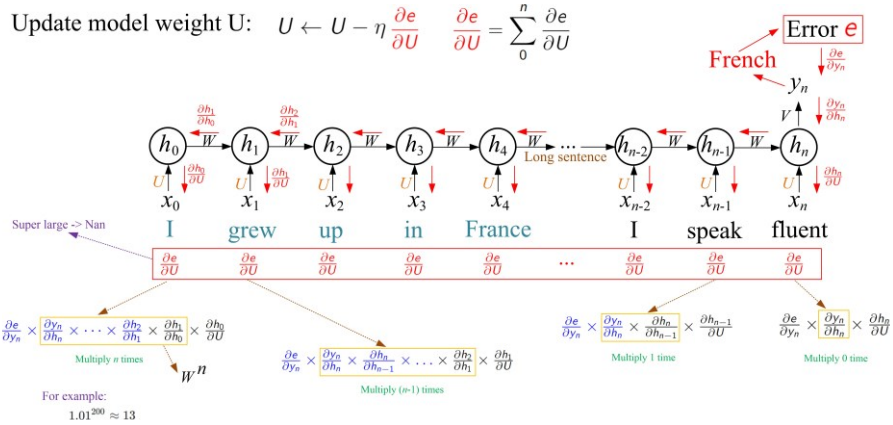


Figure: Gradient signal can end up being multiplied many times. Long term components goes exponentially fast to be vary large.

Solutions to Vanishing/Exploding Gradients

Exploding gradients:

- Clip the gradients to a certain max value.
- Try to set smaller learning rate η.

Vanishing gradients:

- Introducing memory in RNNs, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks.

Long Short-Term Memory Networks (LSTM)

- LSTM networks are RNNs capable of learning long-term dependencies.
- **Cell states** transport the information through the units.
- **Cell gates** control what information can pass through a specific unit.

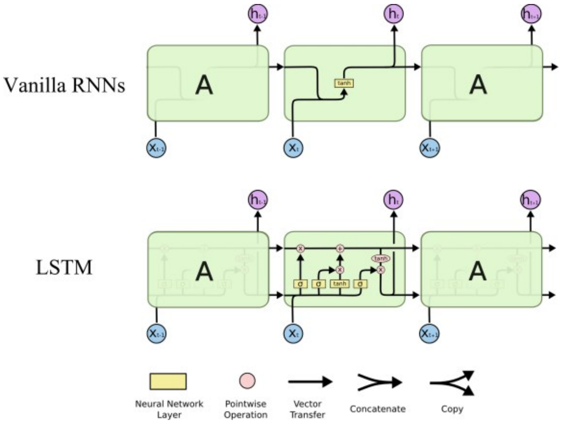
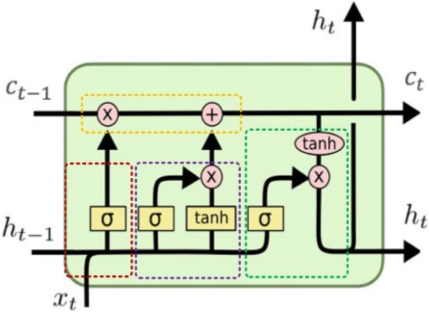


Figure: Vanilla RNNs versus LSTM.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Long Short-Term Memory Networks (LSTM)

Overview of the LSTM structure:



Cell State

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Input Gate

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Forget Gate

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

Output Gate

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

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure: The LSTM unit contains four major components.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

LSTM: Step by Step

Cell: Transport information through the units.

- The horizontal line running through the top that links LSTM units at different time steps.
- Gates are used to control what information can pass thought the current unit.

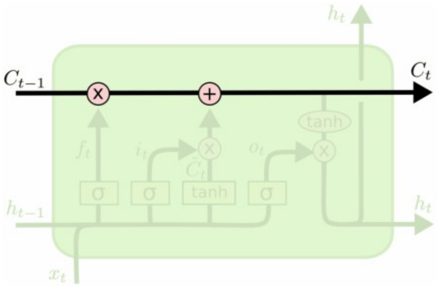
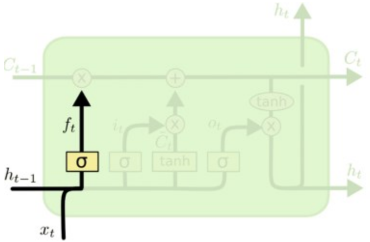


Figure: Cell states transport the information through the units. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

LSTM: Step by Step

Forget Gate: Decide how much information to throw away or remember from the cell state based on new input x_t.

- The Sigmoid layer (θ) outputs a value f_t between 0 and 1. 0 means completely forget while 1 means completely keep.
- x_t is the input at time t. h_{t-1} is the output of previous time step.



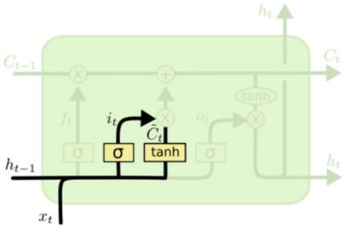
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure: Forget gate decides what information to forget from the cell state.

LSTM: Step by Step

Input Gate: Decide how much information to store in the cell state based on new input x_t.

- A tanh layer transforms the new input to candidate information that will be added to the cell state.
- A Sigmoid layer outputs a value between 0 and 1 deciding what new information to add.



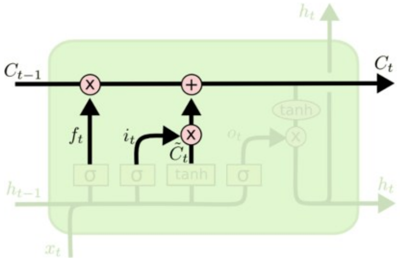
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure: Input gate decides what new information to store in the cell state.

LSTM: Step by Step

Cell Update: Update cell state to be current.

- f_t decides what information to keep from the old cell state C_{t-1}.
- i_t decides what new information $\tilde{C}_t$ (transformed from the new input x_i) to add .



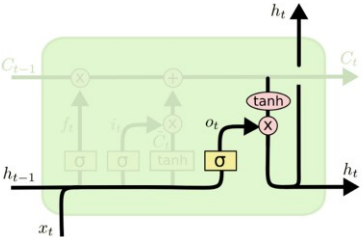
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure: Update cell state by incorporating what old information to forget and what new information to add.

LSTM: Step by Step

Output Gate: Generate the output at current time step t, which is the filtered version of the cell state.

- A tanh layer transforms cell state C_t to candidate output values.
- A Sigmoid layer outputs a value o_t between 0 and 1 deciding what part of the cell state to output.



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

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure: Update cell state by incorporating what old information to forget and what new information to add.

How LSTM Address Vanishing Gradient Problem?

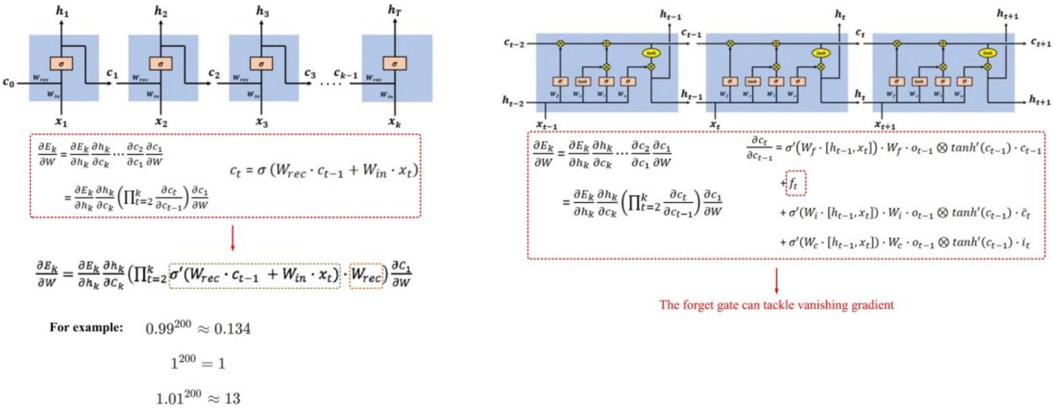


Figure: Vanilla RNNs (left) versus LSTM (right).
https://www.codingninjas.com/codestudio/library/solving-the-vanishing-gradient-problem-with-lstm.

Example: Forecasting Time-Series Gene Expression with LSTM

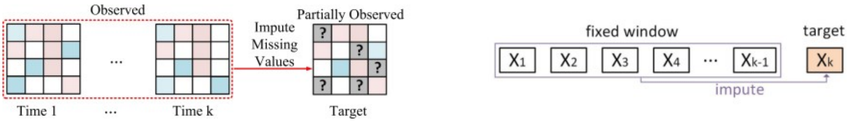
Two time-series gene expression datasets:

- **Reverse phase protein array proteomics data**: consists of highly sensitive and selective antibody-based measurements of 295 proteins and phosphoproteins in breast epithelial cells after individual treatments with six different growth ligands: epidermal growth factor (EGF), hepatocyte growth factor (HGF), oncostatin M (OSM), bone morphogenetic protein 2 (BMP2), transforming growth factor beta (TGFB), and interferon gamma-1b (IFNG). The gene expression at 1, 4, 8, 24, and 48 hours after treated with ligands were retrained.
- **Genome-wide gene expression data**: Human estrogen-responsive breast cancer cells (ZR-75.1) cultured in steroid-free medium for 4 days were stimulated with a mitogenic dose (10nM) of 17 -estradiol and RNA was extracted before hormonal stimulation or after hormonal stimulation. The expression data from 644 genes for 4-, 8-, 12-, 16-, 20-, 24-, 28- and 32- hours were retained.

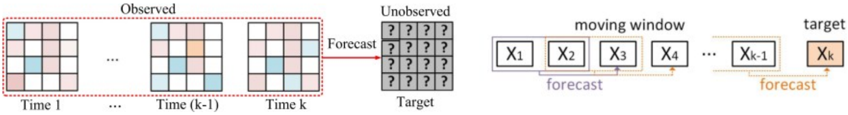
Example: Forecasting Time-Series Gene Expression with LSTM

Using LSTM to perform the following two tasks:

- **Task 1**: Missing gene expression value imputation.

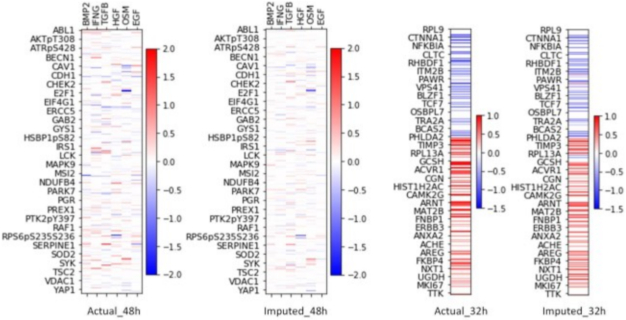


- **Task 2**: Gene expression value forecasting.



Example: Forecasting Time-Series Gene Expression with LSTM

Result: Missing gene expression value imputation:

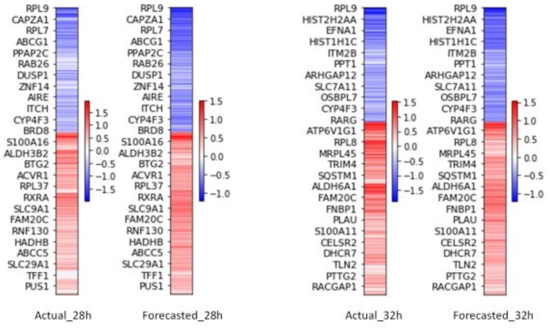


(a) The imputed RPPA data at 48 hour (b) The imputed GE data at 32 hour

Figure: The imputed data (missing ratio 0.5) compared with the actual data. (a) 295 (proteins) × 6 (ligands). (b) 644 (genes) × 1, where the labels for 30 proteins and 33 genes are shown respectively. The color bars indicate the intensity of expression values.

Example: Forecasting Time-Series Gene Expression with LSTM

Result: Gene expression value forecasting:



(a) The forecasted GE data at 28 hour (b) The forecasted GE data at 32 hour

Figure: The forecasted data compared with the respective actual data, where 33 genes are shown due to the space limit. The color bars indicate the intensity of expression values.

Example: Forecasting Time-Series Gene Expression with LSTM

Code in Tensorflow

Missing gene expression value imputation:
https://colab.research.google.com/drive/ 1YRf7wVrNI49ZoGk6n-AqRJxeZBHJq6bs?usp=sharing

Gene expression value forecasting:
https://colab.research.google.com/drive/11abn8z0zJVgo4gBLkqYniQYtLXjVlZni

Bidirectional Long Short-Term Memory Networks (BiLSTM)

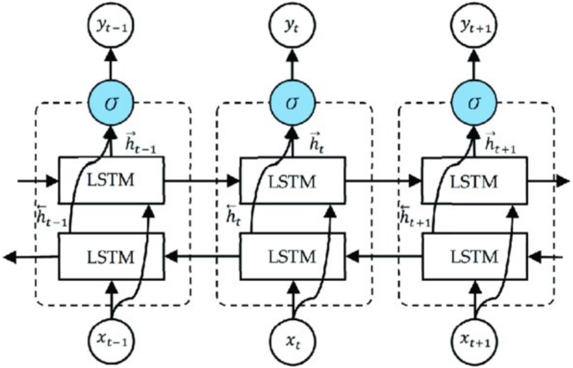
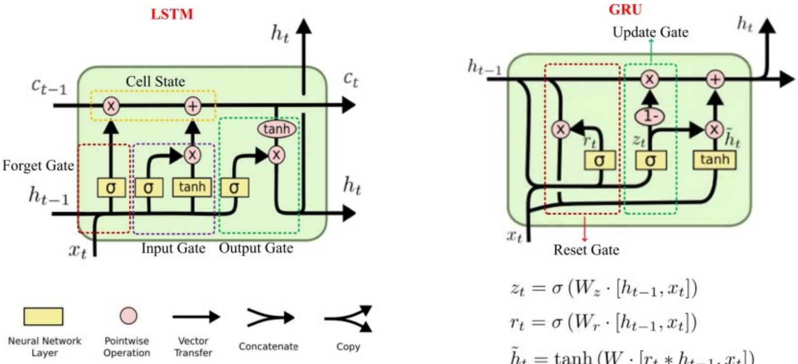


Figure: The input flows in both directions, and it's capable of utilizing information from both sides. Therefore, the output layer can get information from past and future states simultaneously.
DOI: 10.3390/s20195606

Gated Recurrent Units (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

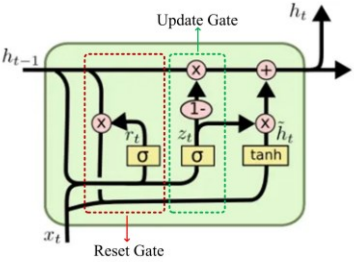
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure: LSTM (left) versus GRU (right). DOI: 10.1109/DS-RT52167.2021.9576137

Gated Recurrent Units (GRU)

- **Reset gate** determines how much of the past information to forget.
- **Update gate** determines how much of the past information needs to be passed along into the future.
- z_t is the output of the update gate. r_t is the output of the reset gate. h_t is the output of the current memory content gate. h_t is the output of the GRU cell.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gated Recurrent Units (GRU)

- GRU contains fewer parameters than LSTM.
- GRU trains faster and perform better than LSTMs on less training data.
- LSTM remembers longer sequences than GRU and outperforms in tasks requiring modeling long-distance relations.

GRU	LSTM
Two gates are reset and update gate.	Three gates are input, output and forget gates.
No use of an internal memory unit.	Use of a memory unit.
Simpler	Complex
Easy to modify.	Complicated to modify
Remember short memory.	Remember longer sequences
Train faster with less data.	Training time consume with larger data.

Figure: GRU vs. LSTM. DOI: 10.1109/DS-RT52167.2021.9576137

Summary

- RNNs are used in handling sequential data, such as sentiment analysis, language modeling, speech recognition, and video analysis.
- Vanilla RNNs are simple but do not work well.
- LSTM and GRU are commonly used and they achieve similar performance.
- Backward flow of gradients in RNN can explode and vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with special gates in LSTM and GRU.

Quesions

You can contact Dr. Min Shi via email for further questions: min.shi@louisiana.edu