EECE-6822 Machine Learning

LSTMs

Cris Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

1

PART 1
Long-Short Term Memory (LSTM) Networks

2

LSTM Networks Introduction

By Jason Brownlee
"Long Short-Term Memory Networks With Python, Develop Sequence Prediction Models With Deep Learning, 2019.
https://machinelearningmastery.com/lstms-with-python/

3

Sequence Prediction

1.1 Sequence Prediction Problems

Sequence prediction is different to other types of supervised learning problems. The sequence imposes an order on the observations that must be preserved when training models and making predictions. Generally, prediction problems that involve sequence data are referred to as sequence prediction problems, although there are a suite of problems that differ based on the input and output sequences. In this section we will take a look at the 4 different types of sequence prediction problems:

1. Sequence Prediction.
2. Sequence Classification.
3. Sequence Generation.
4. Sequence-to-Sequence Prediction.

4

Sequence Prediction Problems

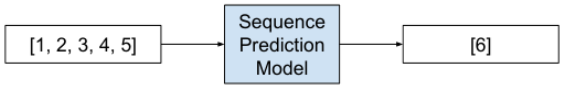


Figure 1.1: Depiction of a sequence prediction problem.

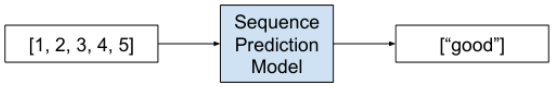


Figure 1.2: Depiction of a sequence classification problem.

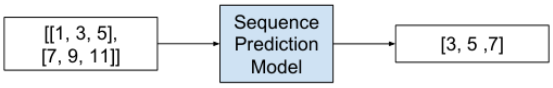


Figure 1.3: Depiction of a sequence generation problem.

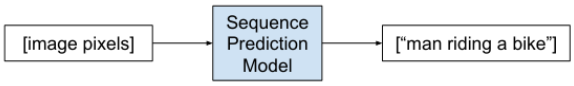


Figure 1.4: Depiction of a sequence generation problem for captioning an image.

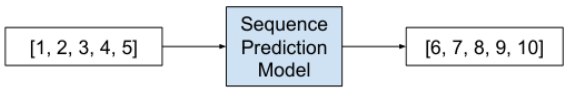


Figure 1.5: Depiction of a sequence-to-sequence prediction problem.

5

5

Limitations of Multilayer Perceptrons

The application of MLPs to sequence prediction requires that the input sequence be divided into smaller overlapping subsequences that are shown to the network in order to generate a prediction. The time steps of the input sequence become input features to the network. The subsequences are overlapping to simulate a window being slid along the sequence in order to generate the required output. This can work well on some problems, but it has 5 critical limitations.

- **Stateless**. MLPs learn a fixed function approximation. Any outputs that are conditional on the context of the input sequence must be generalized and frozen into the network weights.
- **Unaware of Temporal Structure**. Time steps are modeled as input features, meaning that network has no explicit handling or understanding of the temporal structure or order between observations.
- **Messy Scaling**. For problems that require modeling multiple parallel input sequences, the number of input features increases as a factor of the size of the sliding window without any explicit separation of time steps of series.
- **Fixed Sized Inputs**. The size of the sliding window is fixed and must be imposed on all inputs to the network.
- **Fixed Sized Outputs**. The size of the output is also fixed and any outputs that do not conform must be forced.

6

6

LSTM Networks

- The Long Short-Term Memory (LSTM) network is a type of Recurrent Neural Network.
- Recurrent Neural Networks (RNN) are a special type of neural network designed for sequence problems.
- Given a standard feedforward MLP network, an RNN can be thought of as the addition of loops to the architecture.
- The recurrent connections add **state or memory** to the network and allow it to learn and harness the ordered nature of observations within input sequences.

> ... recurrent neural networks contain cycles that feed the network activations from a previous time step as inputs to the network to influence predictions at the current time step. These activations are stored in the internal states of the network which can in principle hold long-term temporal contextual information. This mechanism allows RNNs to exploit a dynamically changing contextual window over the input sequence history

— *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*, 2014

7

7

LSTM Network

- Computational element of the LSTM network is called the memory block or memory cell (or just cell for short).
- Computationally (internal to the LSTM class), an LSTM layer can be viewed as consisting of the set of unrolled (i.e., recurrently connected) cells.

8

8

LSTM Weights, LSTM Gates

A memory cell has weight parameters for the input, output, as well as an internal state that is built up through exposure to input time steps.

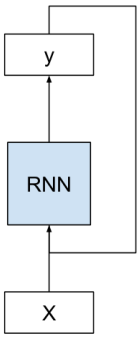
- **Input Weights**. Used to weight input for the current time step.
- **Output Weights**. Used to weight the output from the last time step.
- **Internal State**. Internal state used in the calculation of the output for this time step.

The key to the memory cell are the gates. These too are weighted functions that further govern the information flow in the cell. There are three gates:

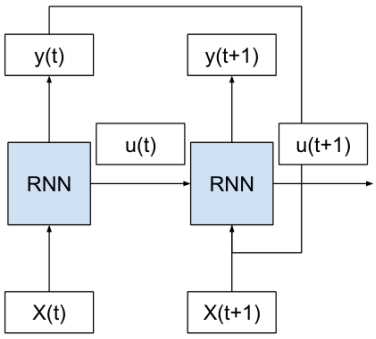
- *Forget Gate*: Decides what information to discard from the cell.
- *Input Gate*: Decides which values from the input to update the memory state.
- *Output Gate*: Decides what to output based on input and the memory of the cell.

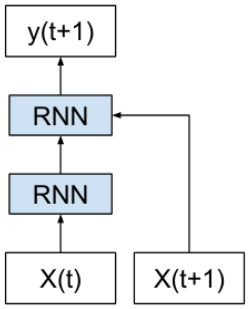
9

Unrolling Recurrent Neural Networks



Simple RNN





Unfolding the Forward Pass

We can see that the cycle is removed and that the output (`y(t)`) and internal state (`u(t)`) from the previous time step are passed on to the network as inputs for processing the next time step. Key in this conceptualization is that the network (`RNN`) does not change between the unfolded time steps. Specifically, the same weights are used for each time step and it is only the outputs and the internal states that differ. In this way, it is as though the whole network (topology and weights) are copied for each time step in the input sequence.

10

10

Models for Sequence Prediction

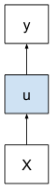


Figure 5.1: One-to-One Sequence Prediction Model.

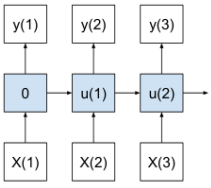


Figure 5.2: One-to-One Sequence Prediction Model Over Time.

- Predicting the next real value in a time series.
- Predicting the next word in a sentence.

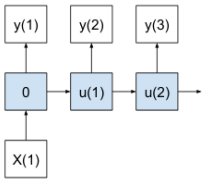


Figure 5.3: One-to-Many Sequence Prediction Model.

- Predicting a sequence of words from a single image.
- Forecasting a series of observations from a single event.

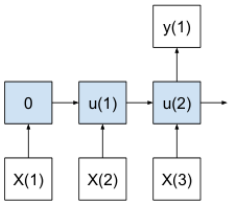


Figure 5.4: Many-to-One Sequence Prediction Model.

- Forecasting the next real value in a time series given a sequence of input observations.
- Predicting the classification label for an input sequence of words, such as sentiment analysis.

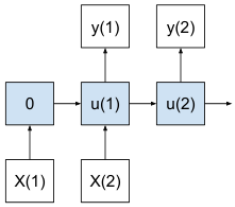


Figure 5.5: Many-to-Many Sequence Prediction Model.

- Summarize a document of words into a shorter sequence of words.
- Classify a sequence of audio data into a sequence of words.

Cardinality of the sequence prediction models defined here refers to time steps, not features.

Mapping Applications to Models

5.3.1 Time Series

- **Univariate Time Series Forecasting**. This is where you have one series with multiple input time steps and wish to predict one time step beyond the input sequence. This can be implemented as a many-to-one model.
- **Multivariate Time Series Forecasting**. This is where you have multiple series with multiple input time steps and wish to predict one time step beyond one or more of the input sequences. This can be implemented as a many-to-one model. Each series is just another input feature.
- **Multi-step Time Series Forecasting**: This is where you have one or multiple series with multiple input time steps and wish to predict multiple time steps beyond one or more of the input sequences. This can be implemented as a many-to-many model.
- **Time Series Classification**. This is where you have one or multiple series with multiple input time steps as input and wish to output a classification label. This can be implemented as a many-to-one model.

Mapping Applications to Models

5.3.2 Natural Language Processing

- **Image Captioning**. This is where you have one image and wish to generate a textual description. This can be implemented as a one-to-many model.
- **Video Description**. This is where you have a sequence of images in a video and wish to generate a textual description. This can be implemented with a many-to-many model.
- **Sentiment Analysis**. This is where you have sequences of text as input and you wish to generate a classification label. This can be implemented as a many-to-one model.
- **Speech Recognition**. This is where you have a sequence of audio data as input and wish to generate a textual description of what was spoken. This can be implemented with a many-to-many model.
- **Text Translation**. This is where you have a sequence of words in one language as input and wish to generate a sequence of words in another language. This can be implemented with a many-to-many model.
- **Text Summarization**. This is where you have a document of text as input and wish to create a short textual summary of the document as output. This can be implemented with a many-to-many model.

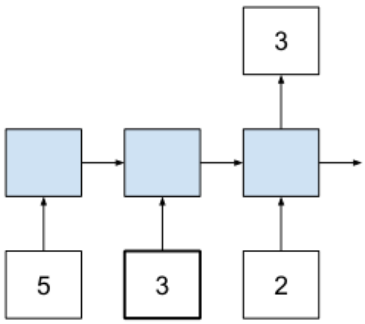
13

13

Example: Echo Sequence Prediction Problem

The echo sequence prediction problem is a contrived problem for demonstrating the memory capability of the Vanilla LSTM. The task is that, given a sequence of random integers as input, to output the value of a random integer at a specific time input step that is not specified to the model.

For example, given the input sequence of random integers `[5, 3, 2]` and the chosen time step was the second value, then the expected output is 3. Technically, this is a sequence classification problem; it is formulated as a many-to-one prediction problem, where there are multiple input time steps and one output time step at the end of the sequence.



14

14

Example: Define the model

- We will work with sequence length 5 integers
- Number of features to 10 (e.g., 0-9).
- The model must specify the expected dimensionality of the input data. In this case, in terms of **time steps (5)** and **features (10)**.
- We will use a **single hidden layer LSTM** with **25 (memory) units**, chosen with a little trial and error.
- The output layer is a fully connected layer (Dense) with 10 neurons for the 10 possible integers that may be output. A softmax activation function is used on the output layer to allow the network to learn and output the distribution over the possible output values.

15

15

The network will use the log loss function while training, suitable for multiclass classification problems, and the efficient Adam optimization algorithm. The accuracy metric will be reported each training epoch to give an idea of the skill of the model in addition to the loss.

```
# define model
length = 5
n_features = 10
out_index = 2
model = Sequential()
model.add(LSTM(25, input_shape=(length, n_features)))
model.add(Dense(n_features, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Listing 6.13: Example of defining a Vanilla LSTM for the Echo Problem.

Running the example defines and compiles the model, then prints a summary of the model structure. Printing a summary of the model structure is a good practice in general to confirm the model was defined and compiled as you intended.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 25)                3600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                260
=================================================================
Total params: 3,860
Trainable params: 3,860
Non-trainable params: 0
_________________________________________________________________
```

Listing 6.14: Example output from the defined model.

16

16

6.6 Make Predictions With the Model

Finally, we can use the fit model to make predictions on new randomly generated sequences. For this problem, this is much the same as the case of evaluating the model. Because this is more of a user-facing activity, we can decode the whole sequence, expected output, and prediction and print them on the screen.

```
# prediction on new data
X, y = generate_example(length, n_features, out_index)
yhat = model.predict(X)
print('Sequence: %s' % [one_hot_decode(x) for x in X])
print('Expected: %s' % one_hot_decode(y))
print('Predicted: %s' % one_hot_decode(yhat))
```

Listing 6.19: Example of making predictions with the fit LSTM model.

Running the example will print the decoded randomly generated sequence, expected outcome, and (hopefully) a prediction that meets the expected value. Your specific results will vary.

```
Sequence: [[7, 0, 2, 6, 7]]
Expected: [2]
Predicted: [2]
```

Listing 6.20: Example output from making predictions the fit model.

17

17

Understanding LSTM Networks

By Christopher Olah

https://colah.github.io/about.html

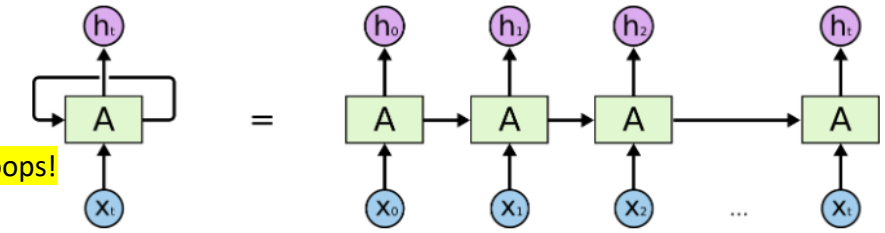
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

18

18

- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we **unroll the loop**:

RNNs have loops!



An unrolled recurrent neural network.

- In practice, RNNs have a hard time learning/handling "long-term dependencies."
- LSTMs do not have this problem.

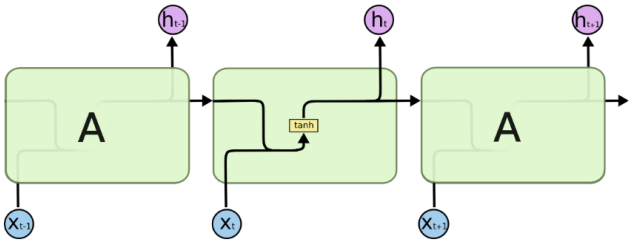
19

LSTM Networks

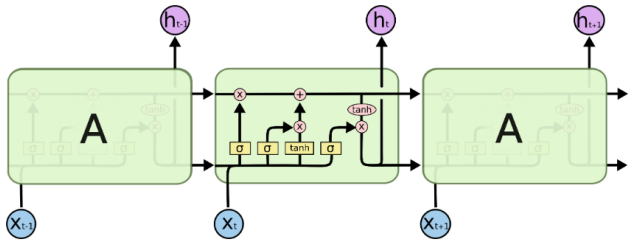
- **Long Short Term Memory networks (LSTMs)**: are a special kind of RNN, capable of learning long-term dependencies.
- Introduced by Hochreiter & Schmidhuber (1997)
- **RNNs** have the form of a chain of repeating modules of neural network - this repeating module will have a very simple structure, such as a single tanh layer.
- **LSTMs** also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

20

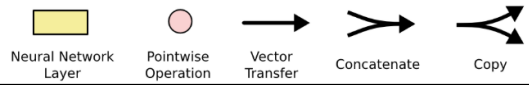
20



The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.



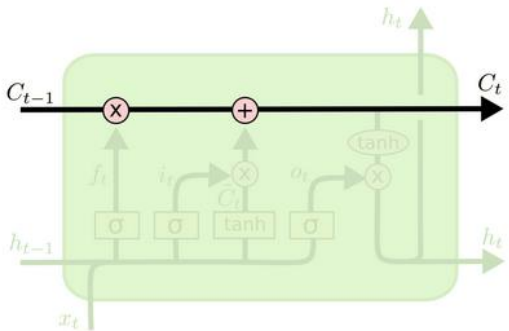
Notes:
--These figures can be easily misleading
--Keep in mind that these diagrams are unrolled – as a way to visually explain what the RNN/LSTM network "A" does!
--They are not separate concurrently existing cells. This illustrates "internal" loop calculations.

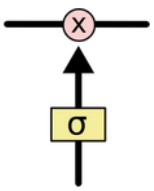
21

21

Core Idea Behind LSTMs

- The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- **Gates** are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

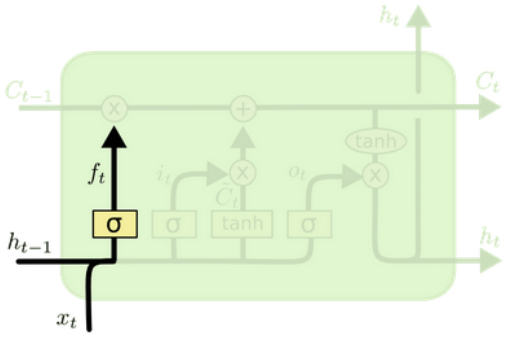




22

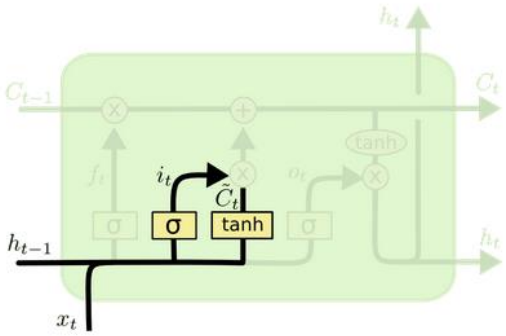
22

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between **0** and **1** for each number in the cell state C_{t-1}. A **1** represents "completely keep this" while a **0** represents "completely get rid of this."



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

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

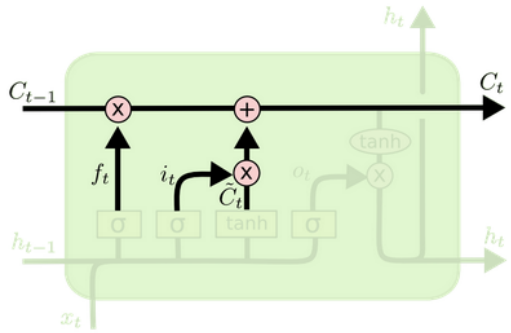


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

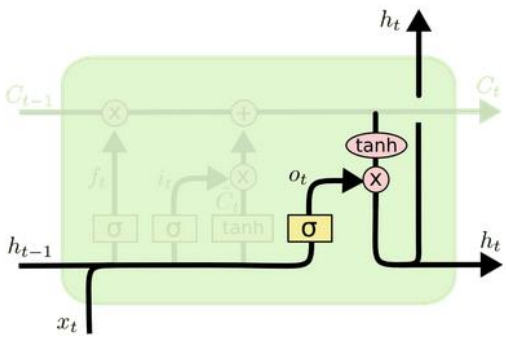


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

25

25

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



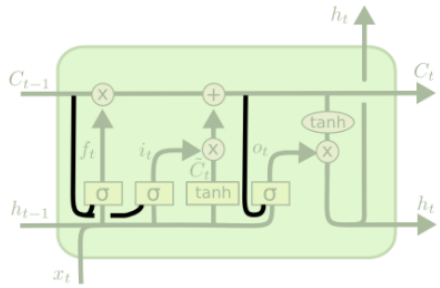
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

26

26

Many Variants of LSTM Network

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.



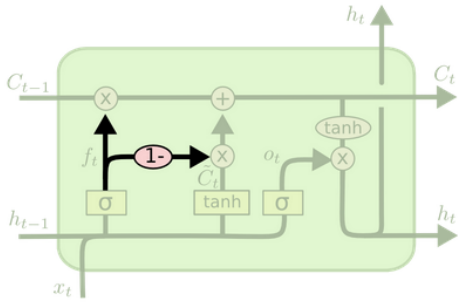
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

27

27

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.

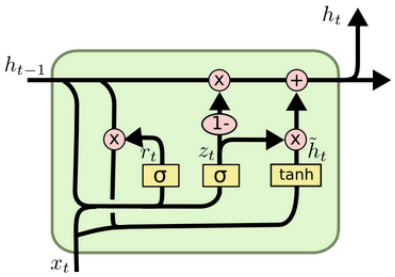


$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

28

28

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It 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 has been growing increasingly popular.



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

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

29

29

Additional Insights into LSTM Networks

Based on discussions from: Jiyang Wang, Raimi Karim

https://zhuanlan.zhihu.com/p/58854907

https://medium.com/data-science/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889

30

30

What is “units” in Keras?

- From Keras documentation at: https://keras.io/api/layers/recurrent_layers/lstm/

LSTM layer

`LSTM` class

```
keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
```

For example:

```
>>> inputs = np.random.random((32, 10, 8))
>>> lstm = keras.layers.LSTM(4)
>>> output = lstm(inputs)
>>> output.shape
(32, 4)
```

Arguments

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use. Default: hyperbolic tangent (`tanh`). If you pass `None`, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation**: Activation function to use for the recurrent step. Default: sigmoid (`sigmoid`). If you pass `None`, no activation is applied (ie. "linear" activation: `a(x) = x`).

31

31

What “units” or “latent_dim” is not

- “units” = “dimensionality of the output space” may be misleading
- Can mistakenly be thought of as number of (unrolled) units in an RNN model or network. **It is not.** If it were correct, “units” should be equal to the number of timesteps of the input sequence, Tx, **but it is not**.
- “units” is not the length of input vector **x**, nor is the timesteps **Tx**.

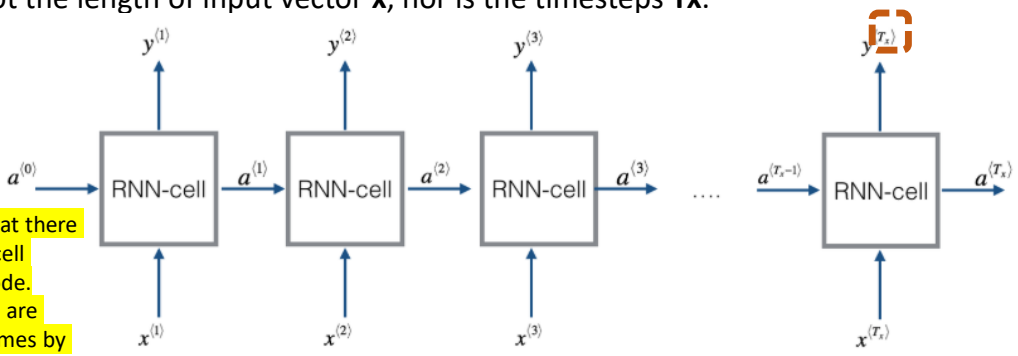


Figure 1 - An RNN Network

--Keep in mind that there is only one RNN cell created by the code.
--RNN operations are **repeated** by *Tx* times by the class itself

32

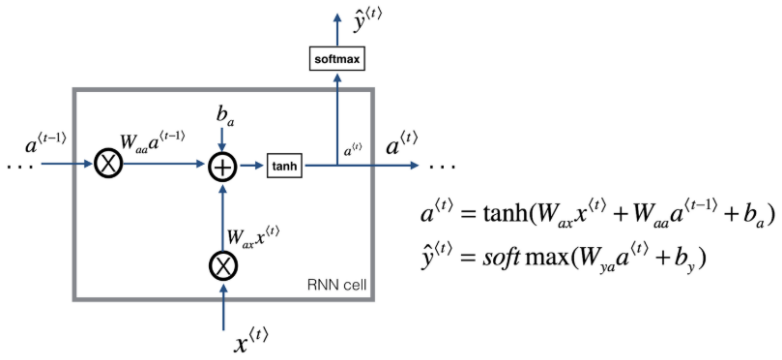


Figure 2 - A Conventional RNN Cell

Keep in mind that implementation of neural networks is by matrix computations between different layers. For example, in the two formulas in the diagram above that implement an RNN cell for one timestep, $W_{ax}x^{<t>}$ is the multiplication of the hidden layer and the input layer at timestep *t*, where W_{ax} is the hidden state matrix between the input vector *x* and the hidden layer vector *a*. The hidden state $a^{<t>}$ (also called activation) and the output $\hat{y}^{<t>}$ at different timesteps *t* in an RNN network shown in the 1st diagram is just the result of the same matrix computations in a for-loop for *t* from *1* to T_x .

Unrolling

33

33

Input *x* into LSTM cell has the shape of *(#timsteps, #feature*). Note that it is different from gray-scale image-based input which is also 2-dimensional (can you tell why?). The 2nd dimension, *#feature*, equals to the length of input vector at each timestep. For example, if *x* is an English word that is fed into RNN cell at each timestep, and it is represented by one-hot encoding, then *#feature* equals to the length of one-hot encoding which is in turn equal to the length of the vocabulary. The reason for calling the 2nd dimension "feature" is because input vector is also called feature vector. Here is another example. In featurized word embedding[+] (instead of one-hot encoding), *x* is an embedding code and $x = (x_1, x_2, \ldots, x_n) \in \Re^n$ in which x_1 represents gender, x_2 represents age, and x_n represents height, etc. So, x is a feature vector and the length of this vector, n , is *#feature*. Note that input x is outside of RNN cell.

When programming, we usually use *Tx* to represent the number of timesteps of input sequence, and *n_x* the number of features (i.e., length of input vector). We define input as the following:

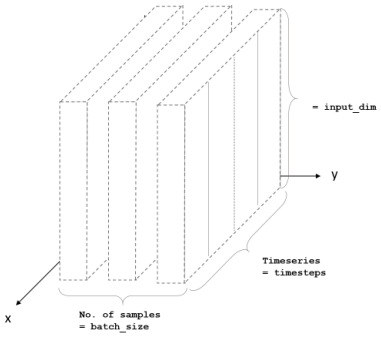
Input

X = **keras.layers.Input**(shape=(*Tx, n_x*))

34

34

Input to LSTM



- Input of the LSTM is always is a 3D array:
  - **(batch_size, time_steps, seq_len)**
- Example in Keras:
  - model = keras.models.Sequential()
  - model.add(keras.layers.LSTM(units=3, input_shape=(2,10)))
  - Though it looks like that input_shape requires a 2D array, it actually requires a 3D array. Here, you can give any size for a batch.
- We can give a fixed batch size like this:
  - model = keras.models.Sequential()
  - model.add(keras.layers.LSTM(units=3, batch_input_shape=(8,2,10)))

35

35

- There appear to be multiple LSTM units, but, they are just used to illustrate more clearly what is going on between timesteps.

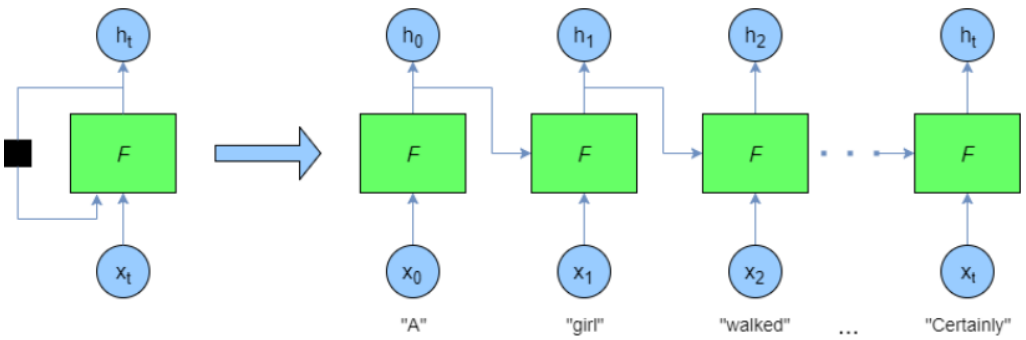


Figure 5 - An RNN cell Repeated (Rolled-over) Tx Times

36

36

Verification

```
from keras.layers import Model, Input, LSTM
Tx = 30
n_x = 3
n_s = 64
X = Input(shape=(Tx, n_x))
s, a, c = LSTM(n_s, return_sequences=True, return_state=True)(X)
model_LSTM = Model(inputs=X, outputs=[s, a, c])
model_LSTM.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 30, 3)             0
_________________________________________________________________
lstm_2 (LSTM)                [(None, 30, 64), (None, 6 17408
=================================================================
Total params: 17,408
Trainable params: 17,408
Non-trainable params: 0
```

We can see that shape of the output of LSTM layer is **(None, 30, 64)**

- "None" is the batch dimension for the number of samples
- Tx=30 is the timesteps
- N_x=3 is the number of input features
- N_s=64 is the length of state vector, n_s, which is assigned as the 1st argument of LSTM class. That is is the number of "units".

We do not explicitly assign the number of timesteps in the definition of LSTM layer, but LSTM layer knows how many times it should repeat itself once it is applied to input X that has Tx in its shape.

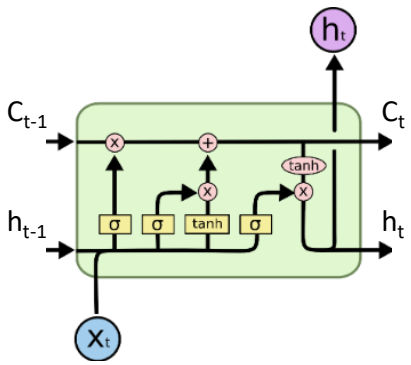
We can conclude that "units" = "dimensionality of the output space" in the Keras documentation - refers to the **dimension of hidden state vector *a*** that is the state output from RNN cell.

37

37

Abstraction of LSTM Cell

- Abstracting is a way to handle complexity. The visual representation of one copy of the LSTM cell may be misleading too because it is abstracted too much.
- When looking at this LSTM Cell representation, remember:
  - Input Xt gets **concatenated** with the hidden state and then fed inside into **four NNs** with a single layer each, and, we have a number of "units" or "neurons" per layer!
  - **See next slide.**

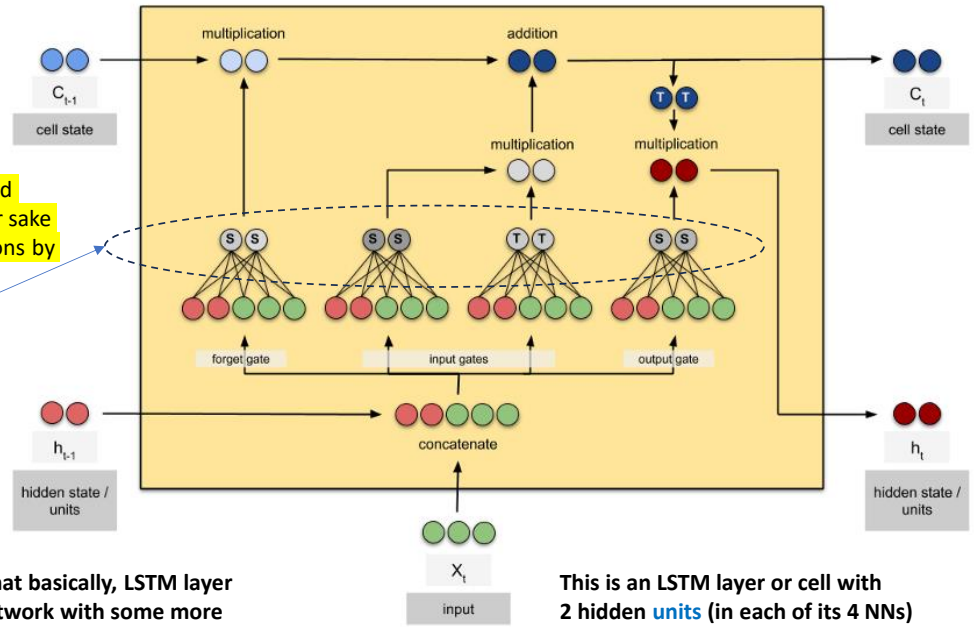


38

38

Details of LSTM **Cell / Module / Block / Unit**

--This Cell is repeated during "unrolling for sake of internal calculations by class".
--It contains **four** interacting layers

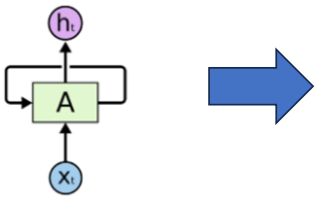


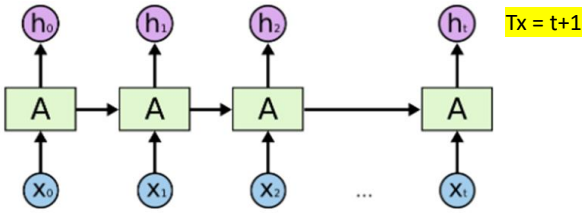
NOTE: We can see that basically, LSTM layer is a feed-forward network with some more Hadamard products and matrix operations

This is an LSTM layer or cell with 2 hidden units (in each of its 4 NNs) and input dimension 3

39

Remember





LSTM Layer = LSTM cell

Number of units is often selected by trial and error to see what works better.

Just a way to visually explain the inner workings of an LSTM cell. Number of unrolls is given indirectly by Tx (class, inside its implementation, will know to repeat/roll Tx times). Tx is the num_time_steps. That is how many times **internally calculations are rolling**.

40

40

PART 2

Code Time!

41

41

Code Time

- See demonstration and discussion in class.
- See also links in the “Code Examples” for this Lecture.

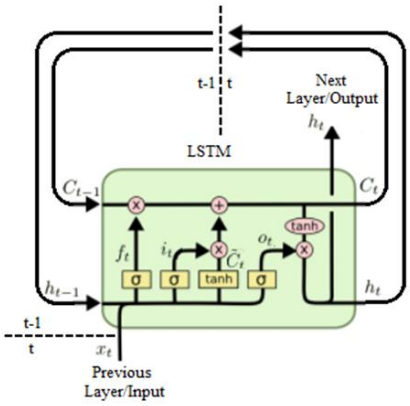
42

42

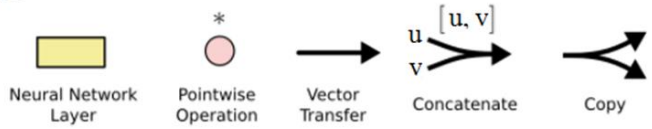
Conclusion
Takeaways

43

Understanding LSTM Networks



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



44

References and Credits

Many of the teaching materials for this course have been adapted from various sources. We are very grateful and thank the following professors, researchers, and practitioners for sharing their teaching materials (in no particular order):

- Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. https://amlbook.com/slides.html
- Ethem Alpaydin. https://www.cmpe.boun.edu.tr/~ethem/i2ml3e/
- Natasha Jaques. https://courses.cs.washington.edu/courses/cse446/25sp/
- Lyle Ungar. https://alliance.seas.upenn.edu/~cis520/dynamic/2022/wiki/index.php?n=Lectures.Lectures
- Aurelien Geron. https://github.com/ageron/handson-ml3
- Sebastian Raschka. https://github.com/rasbt/machine-learning-book
- Trevor Hastie. https://www.statlearning.com/resources-python
- Andrew Ng. https://www.youtube.com/playlist?list=PLoROMvodv4rMiGQp3WXShtMGgzqpfVfbU
- Richard Povineli. https://www.richard.povinelli.org/teaching
- … and many others.

45