EI320A(3) 深度學習使用 Python

Instructors

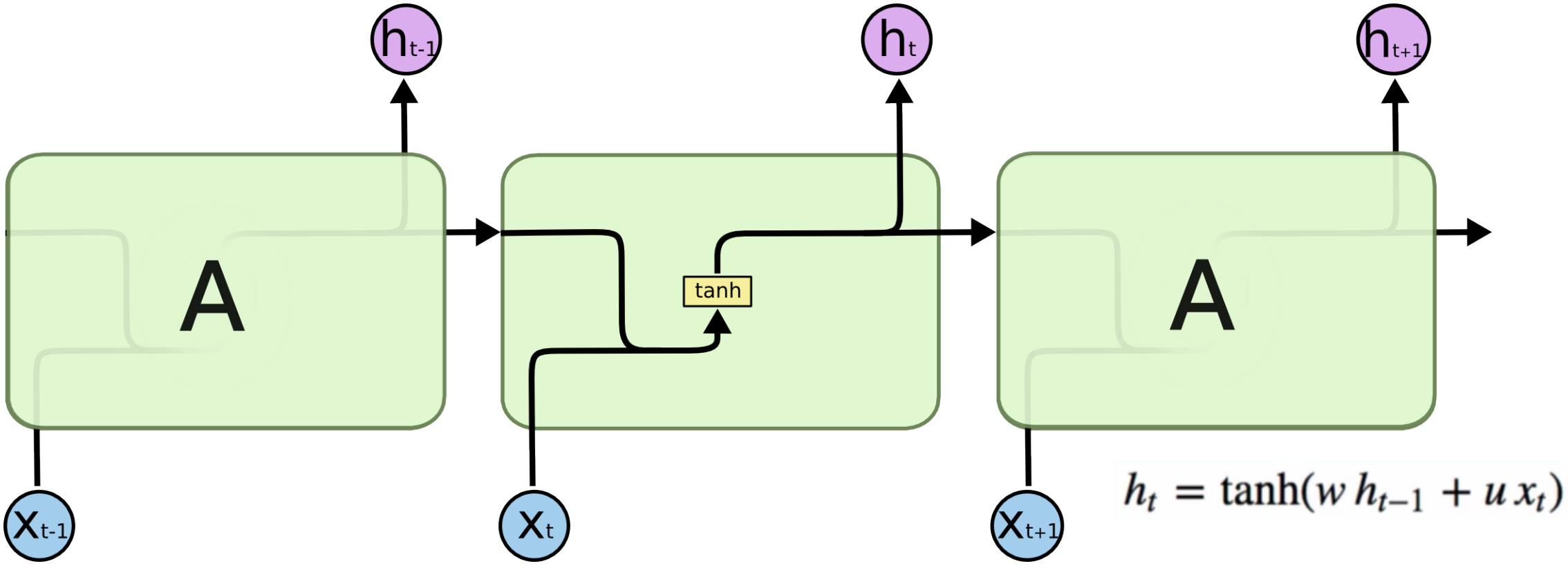
Tipajin Thaipisutikul (t.greentip@gmail.com)

Prof. Huang-Chia Shih (hcshih@Saturn.yzu.edu.tw)

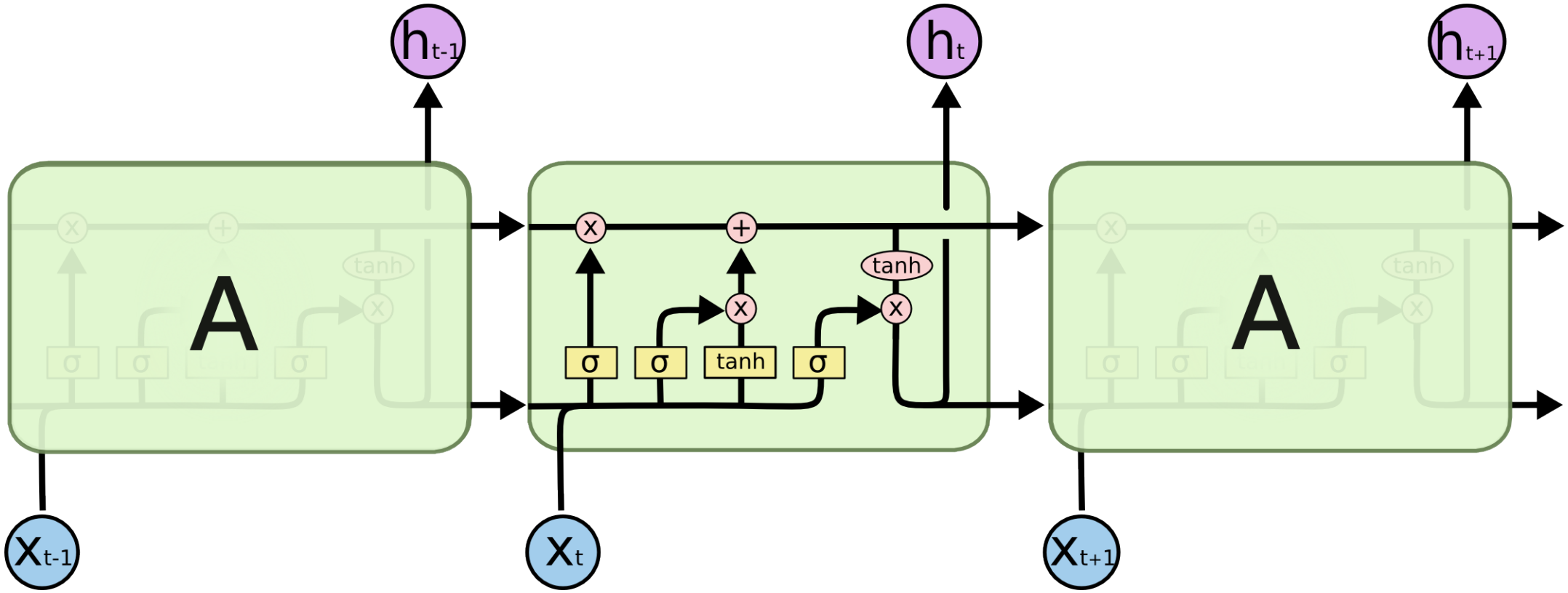
Week	Date	Content	Note	Total
1	2/26	Welcome to the course	Homework (1)	1
2	3/5	Crash Course of Python, NumPy, Pandas, and Matplotlib	In class hands-on (4)	5
3	3/12	Get to know about Data, ML: Classification Models	In class hands-on (5)	10
4	3/19	ML: Regression Models	In class hands-on (5)	15
5	3/26	ML: Clustering/Apriori Models	In class hands-on (5)	20
6	4/2	Holiday		
7	4/9	Introduction to Deep Learning (ANN)		
8	4/16	ANN Labs, Introduction to Convolutional Neural Network (CNN)	In class hands-on (10)	30
9	4/23	Convolutional Neural Network (CNN) & CNN Labs	In class hands-on (5)	35
10	4/30	Introduction to Recurrent Neural Network (RNN)	In class hands-on (5)	40
11	**5/7**	**Recurrent Neural Network (RNN) & RNN Labs**	**In class hands-on (5)**	**45**
12	5/14	Wrap Up all ANN, CNN, RNN Project Proposal Presentation	Proposal Presentation (10)	55
13	5/21	NLP & S2S & Attention Neural Network	In class hands-on (5)	60
14	5/28	N/A	In class hands-on (5)	65
15	6/4	Generative Adversarial Network (GAN)	In class hands-on (5)	70
16	6/11	Reinforcement Learning (RL)	In class hands-on (5)	75
17	6/18	Final Project Presentation	Final Presentation (30)	105

Recurrent Neural Network (RNN)

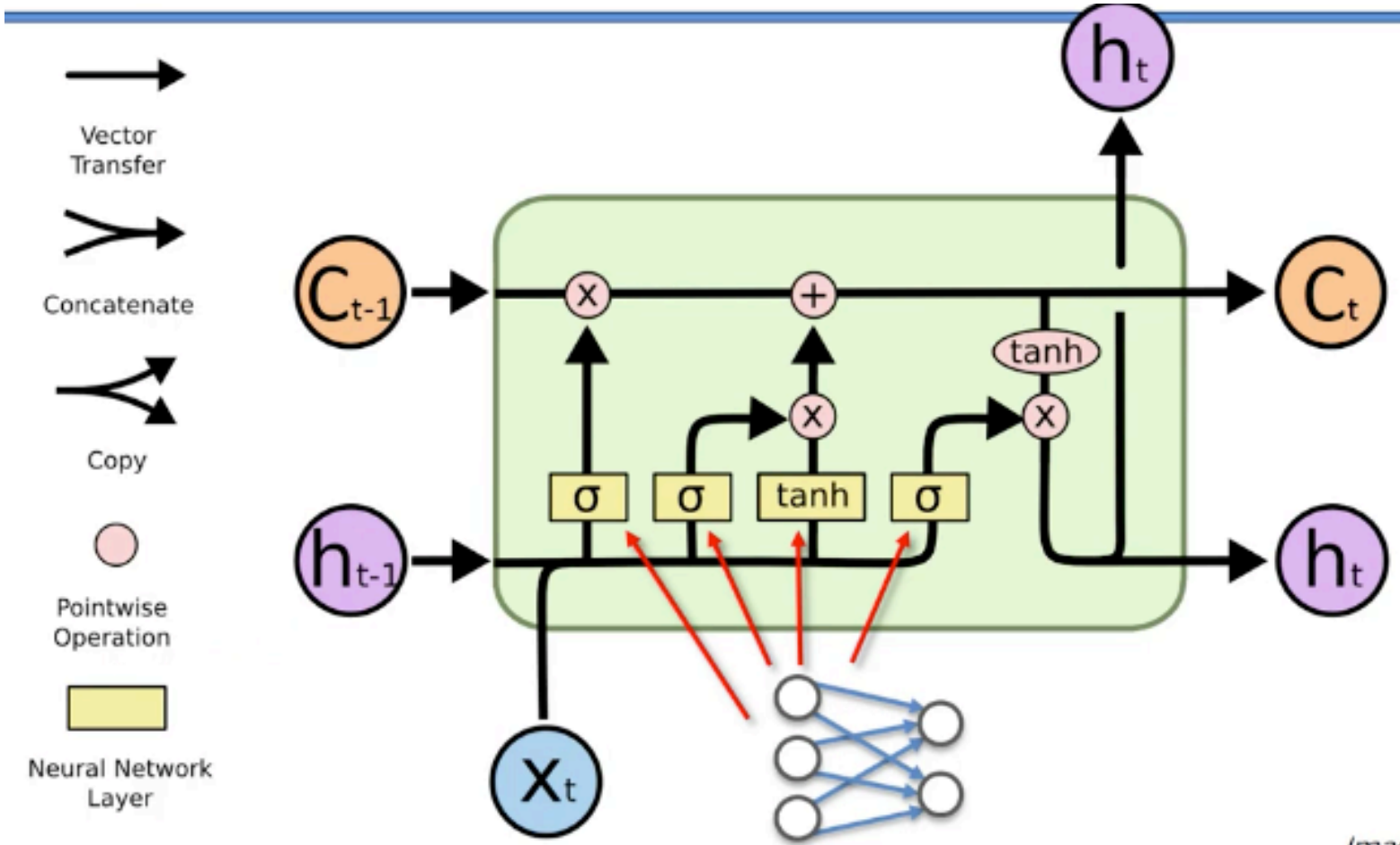
Ref: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



Long-Short Term Memory (LSTM)



Long-Short Term Memory (LSTM)

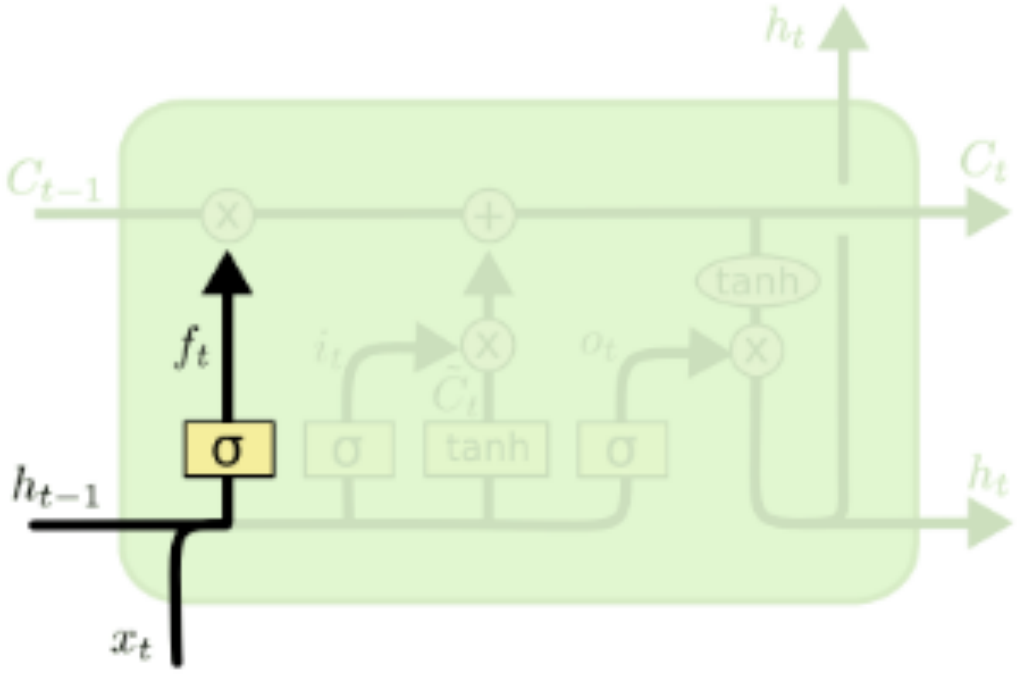


valve

T-shape

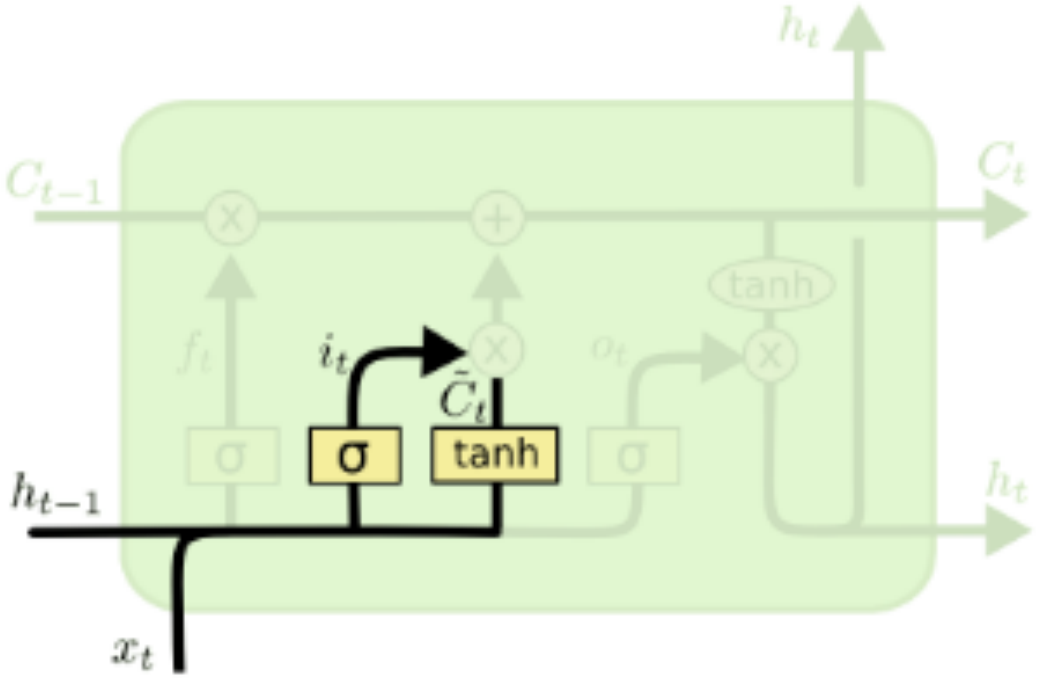
Image Source: colah.github.io

Long-Short Term Memory (LSTM)



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

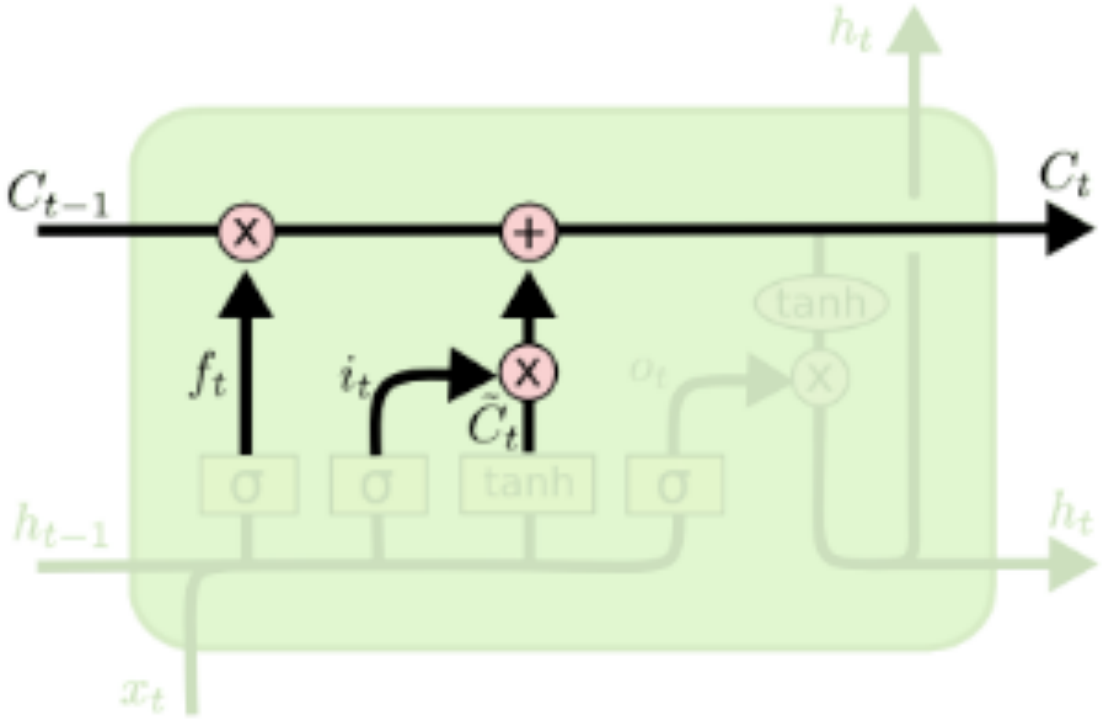
Long-Short Term Memory (LSTM)



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

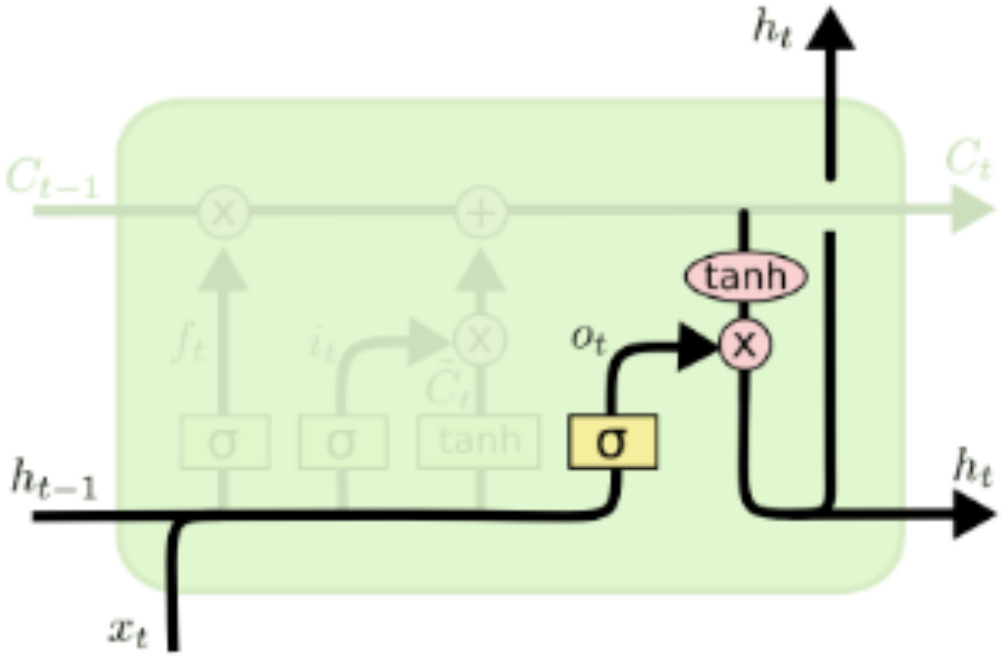
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long-Short Term Memory (LSTM)



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

Long-Short Term Memory (LSTM)

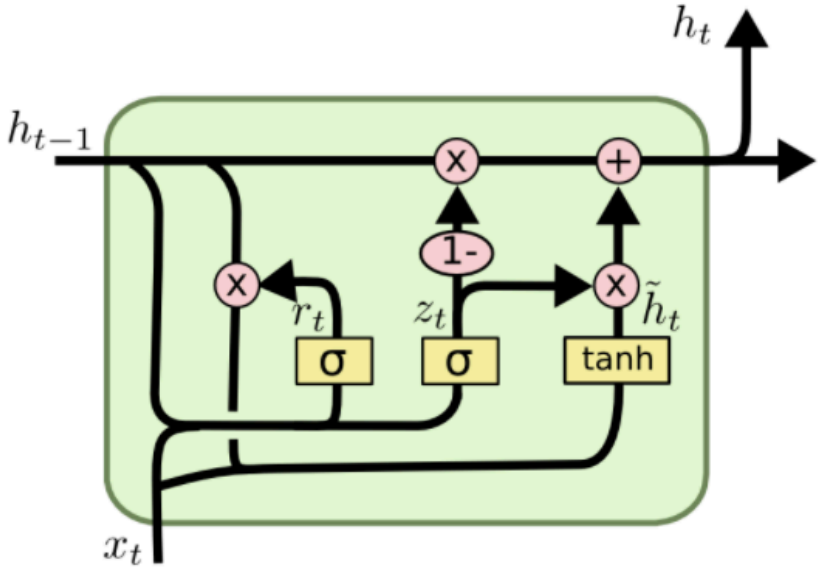


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Gated Recurrent Unit (GRU)

1. A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU.
2. It **combines the forget and input gates into a single "update gate." (z_t)**
3. It also **merges the cell state and hidden state**, and makes some other changes.
4. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



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

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

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

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

LSTM Foundation Plan of Attack

- What are LSTMs?
- How to prepare data for LSTM?
- How to develop Vanilla & Stacked LSTMs?
- How to develop Bi-directional LSTMs?
- How to Develop Encoder-Decoder LSTMs?

What are LSTMs

Sequence Prediction Problems

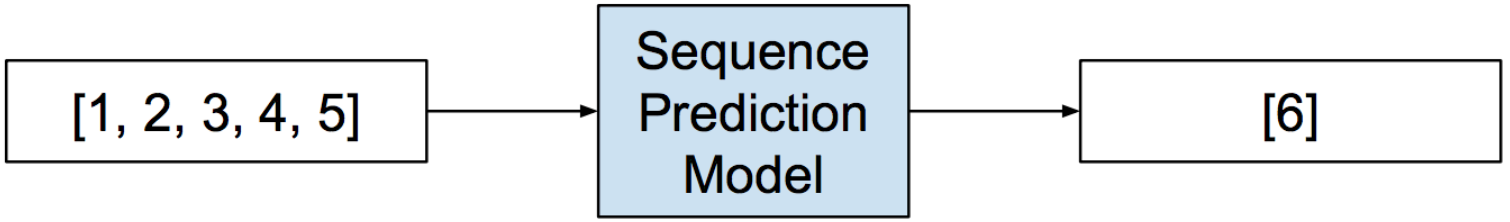
Some examples of **sequence prediction problems** include:

- **Weather Forecasting**. Given a sequence of observations about the weather over time, predict the expected weather tomorrow.
- **Stock Market Prediction**. Given a sequence of movements of a security over time, predict the next movement of the security.
- **Product Recommendation**. Given a sequence of past purchases for a customer, predict the next purchase for a customer.

Sequence prediction involves predicting the next value for a given input sequence. For example:

```
Input Sequence: 1, 2, 3, 4, 5
Output Sequence: 6
```

Example of a sequence prediction problem.



Depiction of a sequence prediction problem.

What are LSTMs

Sequence Prediction Problems

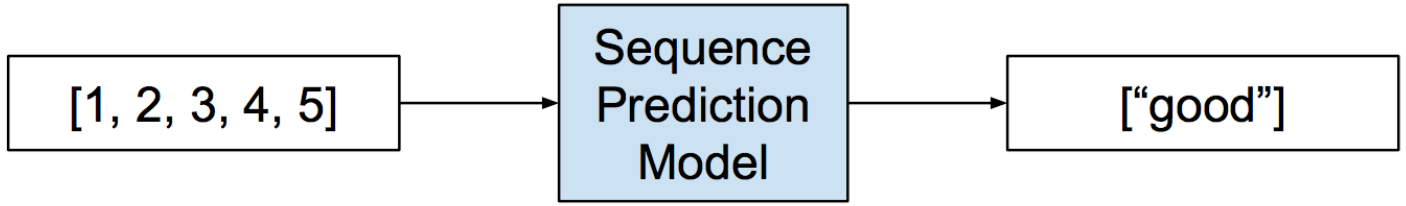
The input sequence may be comprised of real values or discrete values. In the latter case, such problems may be referred to as discrete sequence classification problems. Some examples of **sequence classification** problems include:

- **DNA Sequence Classification**. Given a DNA sequence of A, C, G, and T values, predict whether the sequence is for a coding or non-coding region.
- **Anomaly Detection**. Given a sequence of observations, predict whether the sequence is anomalous or not.
- **Sentiment Analysis.** Given a sequence of text such as a review or a tweet, predict whether the sentiment of the text is positive or negative.

Sequence classification involves predicting a class label for a given input sequence. For example:

```
Input Sequence: 1, 2, 3, 4, 5
Output Sequence: "good"
```

Example of a sequence classification problem.



Depiction of a sequence classification problem.

Sequence Prediction Problems

What are LSTMs

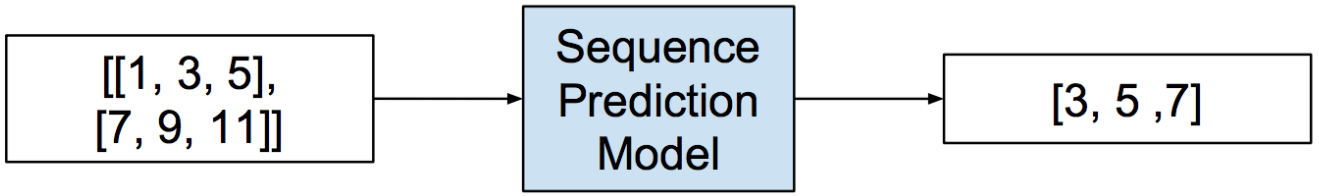
Some examples of **sequence generation problems** include:

- **Text Generation**. Given a corpus of text, such as the works of Shakespeare, generate new sentences or paragraphs of text that read they could have been drawn from the corpus.
- **Handwriting Prediction**. Given a corpus of handwriting examples, generate handwriting for new phrases that has the properties of handwriting in the corpus.
- **Music Generation**. Given a corpus of examples of music, generate new musical pieces that have the properties of the corpus.

Sequence generation involves generating a new output sequence that has the same general characteristics as other sequences in the corpus. For example:

```
Input Sequence: [1, 3, 5], [7, 9, 11]
Output Sequence: [3, 5 ,7]
```

Example of a sequence generation problem.



Depiction of a sequence generation problem.

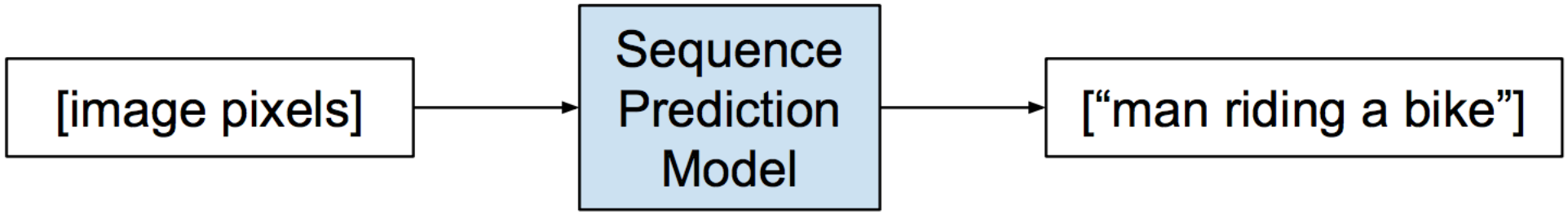
What are LSTMs

Sequence generation may also refer to the generation of a sequence given a single observation as input.
An example is the automatic textual description of images.
Image Caption Generation. Given an image as input, generate a sequence of words that describe an image.

For example:

```
Input Sequence: [image pixels]
Output Sequence: ["man riding a bike"]
```

Example of a sequence generation problem.



Depiction of a sequence generation problem for captioning an image.

What are LSTMs

Sequence Prediction Problems

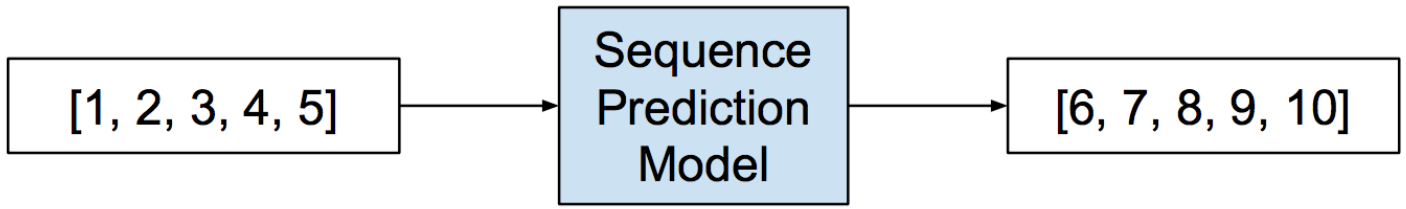
Some examples **of sequence-to-sequence problems** include:

- **Multi-Step Time Series Forecast**ing. Given a time series of observations, predict a sequence of observations for a range of future time steps.
- **Text Summarization**. Given a document of text, predict a shorter sequence of text that describes the salient parts of the source document.
- **Program Execution.** Given the textual description program or mathematical equation predict the sequence of characters that describes the correct output.

Sequence-to-sequence prediction involves predicting an output sequence given an input sequence. For example:

```
Input Sequence: 1, 2, 3, 4, 5
Output Sequence: 6, 7, 8, 9, 10
```

Example of a sequence-to-sequence prediction problem.



Depiction of a sequence-to-sequence prediction problem.

Applications of LSTMs

What are LSTMs

This section provides 3 examples to give you a snapshot of the results that LSTMs are capable of achieving.

1. **Automatic Image Caption Generation**

Example of LSTM generated captions, taken from *Show and Tell: A Neural Image Caption Generator*, 2014.

Applications of LSTMs

What are LSTMs

This section provides 3 examples to give you a snapshot of the results that LSTMs are capable of achieving.

2. **Automatic Translation of Text**

Type	Sentence
Our model	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
Truth	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
Our model	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
Truth	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
Our model	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
Truth	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

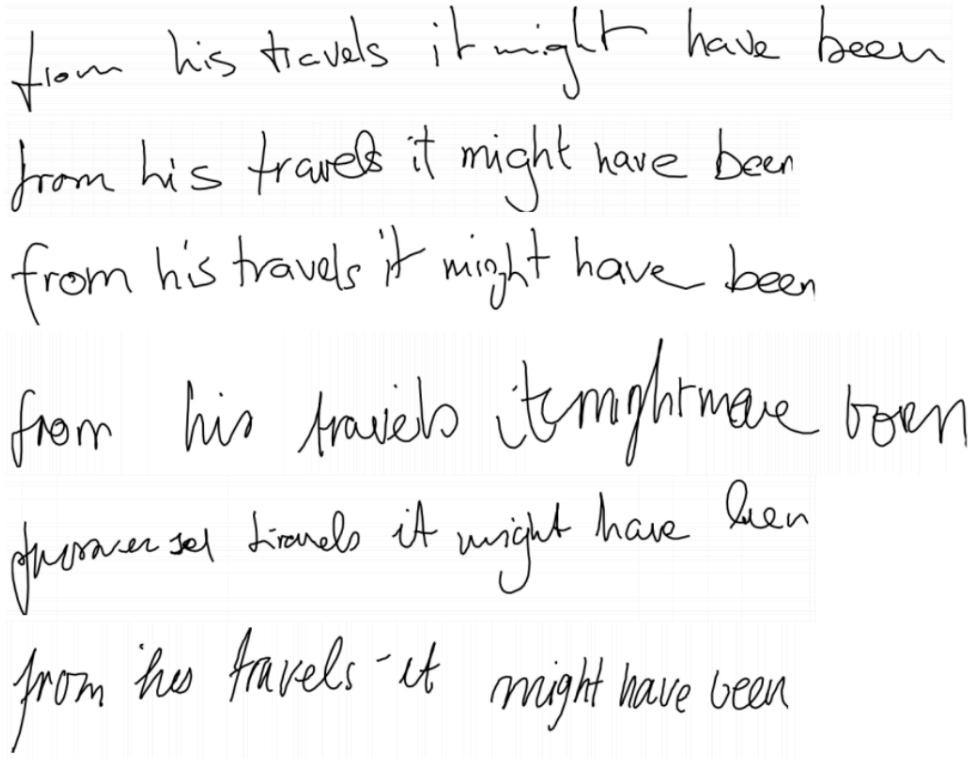
Example of English text translated to French comparing predicted to expected translations, taken from *Sequence to Sequence Learning with Neural Networks*, 2014.

Applications of LSTMs

What are LSTMs

This section provides 3 examples to give you a snapshot of the results that LSTMs are capable of achieving.

3. **Automatic Handwriting Generation:** From this corpus, the relationship between the pen movement and the letters is learned and new examples can be generated. What is fascinating is that different styles can be learned and then mimicked.



Example of LSTM generated captions, taken from *Generating Sequences With Recurrent Neural Networks*, 2014.

LSTM Foundation Plan of Attack

- What are LSTMs?
- How to prepare data for LSTM?
- How to develop Vanilla & Stacked LSTMs?
- How to develop Bi-directional LSTMs?
- How to Develop Encoder-Decoder LSTMs?

How to prepare data for LSTM?

Prepare **Numeric** Data.

This lesson is divided into 4 parts; they are:

1. Prepare **Numeric** Data.
2. Prepare **Categorical** Data.
3. Prepare **Sequences with Varied Lengths.**
4. Sequence Prediction as **Supervised Learning.**

```
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
# define contrived series
data = [10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
series = Series(data)
print(series)
# prepare data for normalization
values = series.values
values = values.reshape((len(values), 1))
# train the normalization
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
print('Min: %f, Max: %f' % (scaler.data_min_, scaler.data_max_))
# normalize the dataset and print
normalized = scaler.transform(values)
print(normalized)
# inverse transform and print
inversed = scaler.inverse_transform(normalized)
print(inversed)
```

Example of normalizing a sequence.

```
0     10.0
1     20.0
2     30.0
3     40.0
4     50.0
5     60.0
6     70.0
7     80.0
8     90.0
9    100.0

Min: 10.000000, Max: 100.000000

[[ 0.       ]
 [ 0.11111111]
 [ 0.22222222]
 [ 0.33333333]
 [ 0.44444444]
 [ 0.55555556]
 [ 0.66666667]
 [ 0.77777778]
 [ 0.88888889]
 [ 1.       ]]

[[  10.]
 [  20.]
 [  30.]
 [  40.]
 [  50.]
 [  60.]
 [  70.]
 [  80.]
 [  90.]
 [ 100.]]
```

Example output from normalizing a sequence.

How to prepare data for LSTM?

Prepare **Numeric** Data.

This lesson is divided into 4 parts; they are:

1. Prepare **Numeric** Data.

```
from pandas import Series
from sklearn.preprocessing import StandardScaler
from math import sqrt
# define contrived series
data = [1.0, 5.5, 9.0, 2.6, 8.8, 3.0, 4.1, 7.9, 6.3]
series = Series(data)
print(series)
# prepare data for normalization
values = series.values
values = values.reshape((len(values), 1))
# train the normalization
scaler = StandardScaler()
scaler = scaler.fit(values)
print('Mean: %f, StandardDeviation: %f' % (scaler.mean_, sqrt(scaler.var_)))
# normalize the dataset and print
standardized = scaler.transform(values)
print(standardized)
# inverse transform and print
inversed = scaler.inverse_transform(standardized)
print(inversed)
```

Example of standardizing a sequence.

```
0    1.0
1    5.5
2    9.0
3    2.6
4    8.8
5    3.0
6    4.1
7    7.9
8    6.3

Mean: 5.355556, StandardDeviation: 2.712568

[[-1.60569456]
 [ 0.05325007]
 [ 1.34354035]
 [-1.01584758]
 [ 1.26980948]
 [-0.86838584]
 [-0.46286604]
 [ 0.93802055]
 [ 0.34817357]]

[[ 1. ]
 [ 5.5]
 [ 9. ]
 [ 2.6]
 [ 8.8]
 [ 3. ]
 [ 4.1]
 [ 7.9]
 [ 6.3]]
```

Example output from standardizing a sequence.

How to prepare data for LSTM?

Prepare **Categorical** Data.

2. Prepare **Categorical** Data.

Categorical data are variables that **contain label values** rather than numeric values.
The number of possible values is often limited to a fixed set. Categorical variables are often called nominal.
Some examples include:

- A pet variable with the values: **dog and cat.**
- A color variable with the values: **red, green, and blue**.
- A place variable with the values**: first, second, and third.**

How to Convert Categorical Data to Numerical Data
This involves 2 steps:

1. Integer Encoding.
2. One Hot Encoding.

How to prepare data for LSTM?

Prepare **Categorical** Data.

2. Prepare **Categorical** Data.

Integer Encoding

As a first step, each unique category value is assigned an integer value. For example, `red` is 1, `green` is 2, and `blue` is 3. This is called label encoding or an integer encoding and is easily reversible. For some variables, this may be enough.

One Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough. In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value. In the `color` variable example, there are 3 categories and therefore 3 binary variables are needed. A `1` value is placed in the binary variable for the color and `0` values for the other colors. For example:

```
red, green, blue
1,   0,    0
0,   1,    0
0,   0,    1
```

Example of one hot encoded color.

How to prepare data for LSTM?

Prepare **Categorical** Data.

2. Prepare **Categorical** Data.

One Hot Encode with scikit-learn

In this example, we will assume the case where you have an output sequence of the following 3 labels: `cold`, `warm`, `hot`. An example sequence of 10 time steps may be:

```
cold, cold, warm, cold, hot, hot, warm, cold, warm, hot
```

Example of categorical temperature sequence.

In this example, we will use the encoders from the scikit-learn library. Specifically, the `LabelEncoder` of creating an integer encoding of labels and the `OneHotEncoder` for creating a one hot encoding of integer encoded values. The complete example is listed below.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot']
values = array(data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
# binary encode
onehot_encoder = OneHotEncoder(sparse=False, categories='auto')
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print(inverted)
```

Example of one hot encoding a sequence.

By default, the `OneHotEncoder` class will return a more efficient sparse encoding. This may not be suitable for some applications, such as use with the Keras deep learning library. In this case, we disabled the sparse return type by setting the `sparse=False` argument. If we receive a prediction in this 3-value one hot encoding, we can easily invert the transform back to the original label.

First, we can use the `argmax()` NumPy function to locate the index of the column with the largest value. This can then be fed to the `LabelEncoder` to calculate an inverse transform back to a text label. This is demonstrated at the end of the example with the inverse transform of the first one hot encoded example back to the label value `cold`. Again, note that input was formatted for readability.

```
['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']

[0 0 2 0 1 1 2 0 2 1]

[[ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]

['cold']
```

Example output of one hot encoded color.

How to prepare data for LSTM?

3. Prepare **Sequences with Varied Lengths**

Pre-Sequence Padding

Pre-sequence padding is the default (`padding='pre'`) The example below demonstrates pre-padding 3-input sequences with 0 values.

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [
  [1, 2, 3, 4],
     [1, 2, 3],
        [1]
  ]
# pad sequence
padded = pad_sequences(sequences)
print(padded)
```

Example of pre-sequence padding.

Running the example prints the 3 sequences pre-pended with zero values.

```
[[1 2 3 4]
[0 1 2 3]
[0 0 0 1]
```

Example output of pre-sequence padding.

Post-Sequence Padding

Padding can also be applied to the end of the sequences, which may be more appropriate for some problem domains. Post-sequence padding can be specified by setting the `padding` argument to `post`.

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [
  [1, 2, 3, 4],
     [1, 2, 3],
        [1]
  ]
# pad sequence
padded = pad_sequences(sequences, padding='post')
print(padded)
```

Example of post-sequence padding.

Running the example prints the same sequences with zero-values appended.

```
[[1 2 3 4]
[1 2 3 0]
[1 0 0 0]]
```

Example output of post-sequence padding.

How to prepare data for LSTM?

3. Prepare **Sequences with Varied Lengths**

Pre-Sequence Truncation

The default truncation method is to remove time steps from the beginning of sequences. This is called pre-sequence truncation. The example below truncates sequences to a desired length of 2.

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [
  [1, 2, 3, 4],
     [1, 2, 3],
        [1]
  ]
```

```
# truncate sequence
truncated= pad_sequences(sequences, maxlen=2)
print(truncated)
```

Example of pre-sequence truncating.

Running the example removes the first two time steps from the first sequence, the first time step from the second sequence, and pads the final sequence.

```
[[3 4]
[2 3]
[0 1]]
```

Example output of pre-sequence truncating.

Post-Sequence Truncation

Sequences can also be trimmed by removing time steps from the end of the sequences. This approach may be more desirable for some problem domains. Post-sequence truncation can be configured by changing the `truncating` argument from the default `pre` to `post` as follows:

```
from keras.preprocessing.sequence import pad_sequences
# define sequences
sequences = [
  [1, 2, 3, 4],
     [1, 2, 3],
        [1]
  ]
# truncate sequence
truncated= pad_sequences(sequences, maxlen=2, truncating='post')
print(truncated)
```

Example of post-sequence truncating.

Running the example removes the last two time steps from the first sequence, the last time step from the second sequence, and again pads the final sequence.

```
[[1 2]
[1 2]
[0 1]]
```

Example output of post-sequence truncating.

How to prepare data for LSTM?

4. **Sequence Prediction as Supervised Learning** (Pandas **shift()** Function)

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
print(df)
```

Running the example prints the time series data with the row indices for each observation.

```
   t
0  0
1  1
2  2
3  3
4  4
5  5
6  6
7  7
8  8
9  9
```

Example output of the created series.

We can shift all the observations down by one time step by inserting one new row at the top. Because the new row has no data, we can use `NaN` to represent *no data*. The shift function can do this for us and we can insert this shifted column next to our original series.

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
# shift forward
df['t-1'] = df['t'].shift(1)
print(df)
```

Example of shifting the series forward.

Running the example gives us two columns in the dataset. The first with the original observations and a new shifted column. We can see that shifting the series forward one time step gives us a primitive supervised learning problem, although with `X` and `y` in the wrong order. Ignore the column of row labels. The first row would have to be discarded because of the `NaN` value. The second row shows the input value of 0.0 in the second column (input or `X`) and the value of 1 in the first column (output or `y`).

```
   t  t-1
0  0  NaN
1  1  0.0
2  2  1.0
3  3  2.0
4  4  3.0
5  5  4.0
6  6  5.0
7  7  6.0
8  8  7.0
9  9  8.0
```

Example output of shifting the series forward.

How to prepare data for LSTM?

4. **Sequence Prediction as Supervised Learning** (Pandas **shift()** Function)

The shift operator can also accept a negative integer value. This has the effect of pulling the observations up by inserting new rows at the end. Below is an example:

```
from pandas import DataFrame
# define the sequence
df = DataFrame()
df['t'] = [x for x in range(10)]
# shift backward
df['t+1'] = df['t'].shift(-1)
print(df)
```

Example of shifting the series backward.

Running the example shows a new column with a `NaN` value as the last value. We can see that the forecast column can be taken as an input (`X`) and the second as an output value (`y`). That is the input value of 0 can be used to forecast the output value of 1.

```
   t  t+1
0  0  1.0
1  1  2.0
2  2  3.0
3  3  4.0
4  4  5.0
5  5  6.0
6  6  7.0
7  7  8.0
8  8  9.0
9  9  NaN
```

Example output of shifting the series backward.

LSTM Foundation Plan of Attack

- What are LSTMs?
- How to prepare data for LSTM?
- How to develop Vanilla & Stacked LSTMs?
- How to develop Bi-directional LSTMs?
- How to Develop Encoder-Decoder LSTMs?

How to develop Vanilla & Stacked LSTMs?

```
#3D tensor with shape (batch_size, timesteps, input_dim)
```

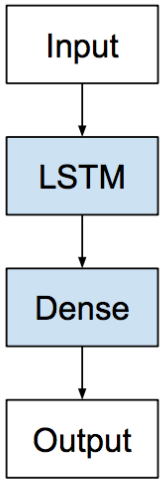


Figure 6.1: Vanilla LSTM Architecture.

The choice of activation function is most important for the output layer as it will define the format that predictions will take. For example, below are some common predictive modeling problem types and the structure and standard activation function that you can use in the output layer:

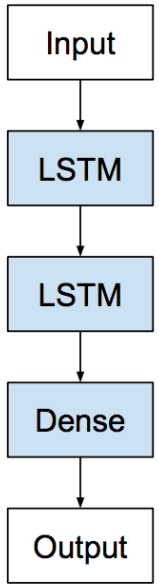
- **Regression**: Linear activation function, or `linear`, and the number of neurons matching the number of outputs. This is the default activation function used for neurons in the `Dense` layer.
- **Binary Classification (2 class)**: Logistic activation function, or `sigmoid`, and one neuron the output layer.
- **Multiclass Classification (> 2 class)**: Softmax activation function, or `softmax`, and one output neuron per class value, assuming a one hot encoded output pattern.

The type of predictive modeling problem imposes constraints on the type of loss function that can be used. For example, below are some standard loss functions for different predictive model types:

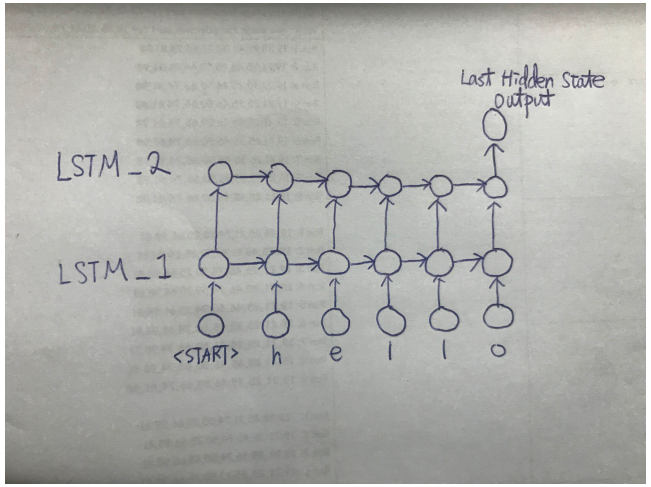
- **Regression**: Mean Squared Error or `mean_squared_error`, `mse` for short.
- **Binary Classification (2 class)**: Logarithmic Loss, also called cross entropy or `binary_crossentropy`.
- **Multiclass Classification (> 2 class)**: Multiclass Logarithmic Loss or `categorical_crossentropy`.

How to develop Vanilla & Stacked LSTMs?

Stacked LSTM: The Stacked LSTM is a model that has multiple hidden LSTM layers where each layer contains multiple memory cells. We will refer to it as a Stacked LSTM here to differentiate it from the unstacked LSTM (Vanilla LSTM) and a variety of other extensions to the basic LSTM model.



Stacked LSTM Architecture.



To stack LSTM layers, we need to change the configuration of the prior `LSTM` layer to output a 3D array as input for the subsequent layer. We can do this by setting the `return_sequences` argument on the layer to `True` (defaults to `False`). This will return one output for each input time step and provide a 3D array. Below is the same example as above with `return_sequences=True`.

```
# Example of one output for each input time step
from keras.models import Sequential
from keras.layers import LSTM
from numpy import array
# define model where LSTM is also output layer
model = Sequential()
model.add(LSTM(1, return_sequences=True, input_shape=(3,1)))
model.compile(optimizer='adam', loss='mse')
# input time steps
data = array([0.1, 0.2, 0.3]).reshape((1,3,1))
# make and show prediction
print(model.predict(data))
```

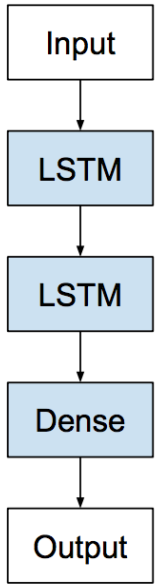
Example of an layer LSTM that returns sequences.

Running the example outputs a single value for each time step in the input sequence.

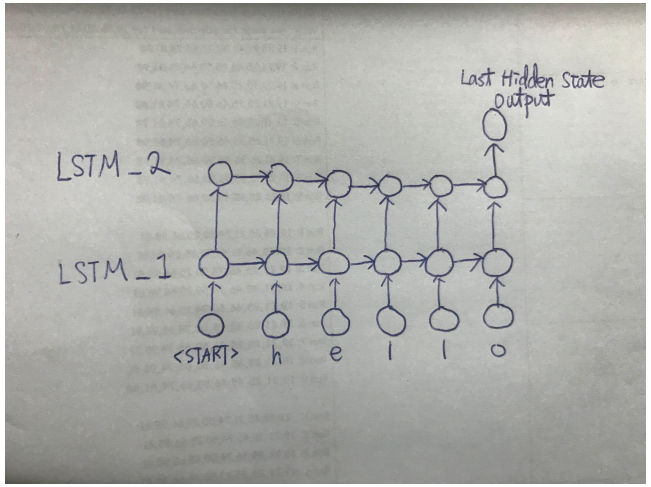
```
[[[-0.02115841]
  [-0.05322712]
  [-0.08976141]]]
```

Example output from an LSTM model that returns sequences.

How to develop Vanilla & Stacked LSTMs?



Stacked LSTM Architecture.




```
model = Sequential()
model.add(LSTM(..., return_sequences=True, input_shape=(...)))
model.add(LSTM(...))
model.add(Dense(...))
```

Example of defining a Stacked LSTM with 2 hidden layers.

```
model = Sequential()
model.add(LSTM(..., return_sequences=True, input_shape=(...)))
model.add(LSTM(..., return_sequences=True))
model.add(LSTM(..., return_sequences=True))
model.add(LSTM(...))
model.add(Dense(...))
```

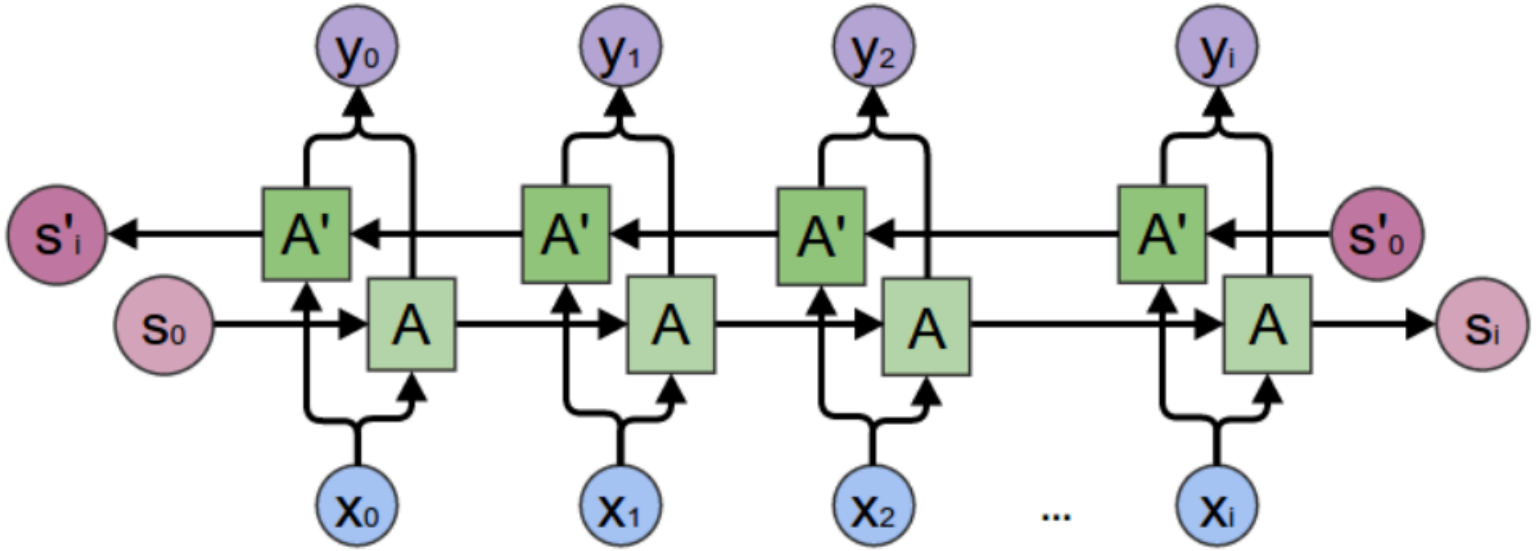
Example of defining a Stacked LSTM with 4 hidden layers.

LSTM Foundation Plan of Attack

- What are LSTMs?
- How to prepare data for LSTM?
- How to develop Vanilla & Stacked LSTMs?
- How to develop Bi-directional LSTMs?
- How to Develop Encoder-Decoder LSTMs?

How to develop Bi-directional LSTMs?

- To enable straight (past) and reverse traversal of input (future), Bidirectional RNNs, or BRNNs, are used.
- A BRNN is a combination of 2 RNNs - **one RNN moves forward, beginning from the start of the data sequ**ence, and the **other, moves backward, beginning from the end of the data sequence**.
- The network blocks in a BRNN can either be simple RNNs, GRUs, or LSTMs.

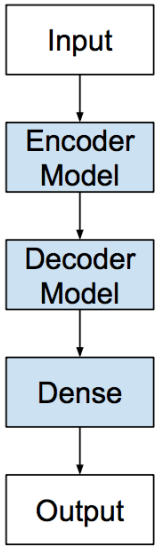


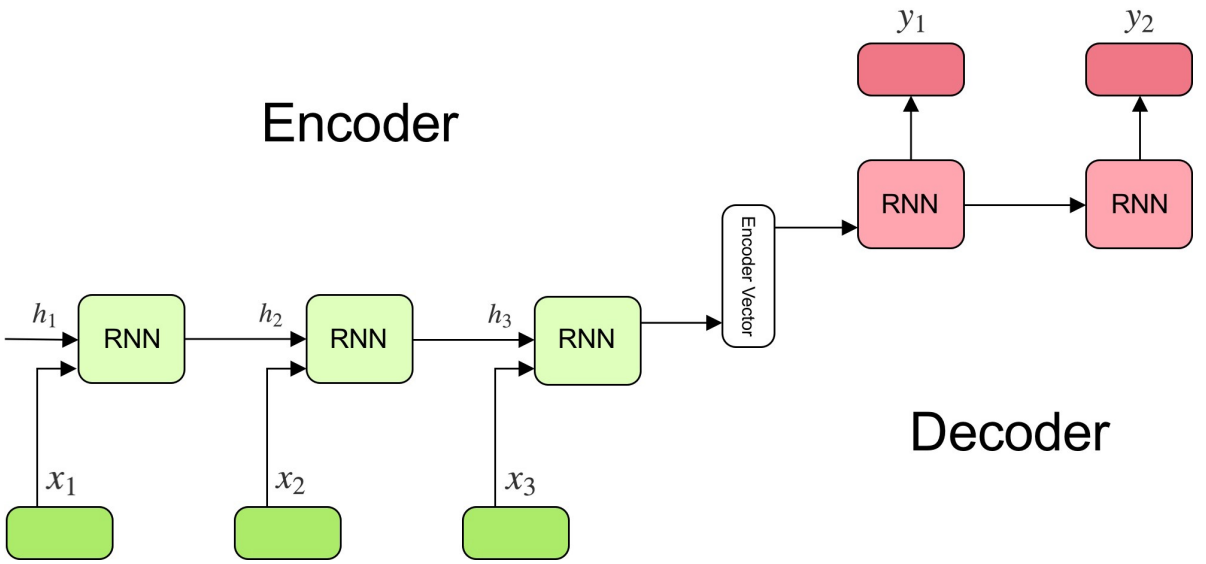
LSTM Foundation Plan of Attack

- What are LSTMs?
- How to prepare data for LSTM?
- How to develop Vanilla & Stacked LSTMs?
- How to develop Bi-directional LSTMs?
- How to Develop Encoder-Decoder LSTMs?

How to Develop Encoder-Decoder LSTMs?

- There is a more challenging type of sequence prediction problem that takes **a sequence as input and requires a sequence prediction as output.** These are called sequence-to-sequence prediction problems, or **seq2seq.**
- One modeling concern that makes these problems challenging is that **the length of the input and output sequences may vary.**
- Given that there are **multiple input time steps** and **multiple output time steps**, this form of problem is referred to as **many-to-many type sequence prediction problem**.
- This architecture is comprised of **2 models: one for reading the input sequence and encoding it into a fixed-length vector**, and **a second for decoding the fixed-length vector and outputting the predicted sequence**.





: Encoder-decoder LSTM Architecture.

How to Develop Encoder-Decoder LSTMs?

- **The encoder**: One or more **LSTM** layers can be used to **implement the encoder model**.

```
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
```

Example of a Vanilla LSTM model.

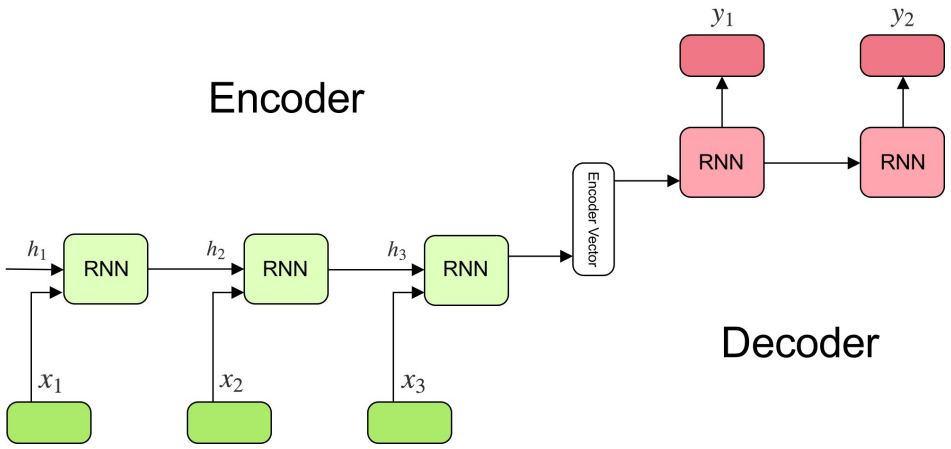
- **The decoder** must transform the learned internal representation of the input sequence into the correct output sequence. One or more **LSTM layers can also be used to implement the decoder model.**
- This model **reads from the fixed sized output** from the encoder model.
- As with the Vanilla LSTM, a **Dense layer is used as the output for the network.**
- The same weights can be used to output each time step in the output sequence **by wrapping the Dense layer** in a **TimeDistributed** wrapper.

```
model.add(LSTM(..., return_sequences=True))
model.add(TimeDistributed(Dense(...)))
```

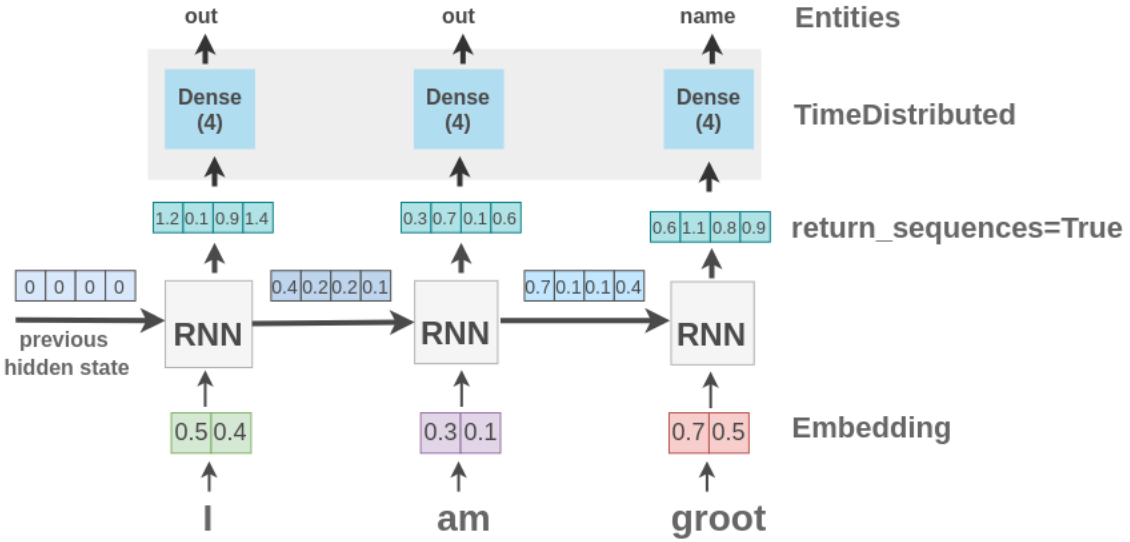
Example of a LSTM model with `TimeDistributed` wrapped `Dense` layer.

How to Develop Encoder-Decoder LSTMs?

- The same weights can be used to output each time step in the output sequence **by wrapping the Dense layer** in a **TimeDistributed** wrapper.



Or



How to Develop Encoder-Decoder LSTMs?

- **There's a problem though.** We must connect the encoder to the decoder, and they do not fit.
- That is, the encoder will produce a 2-dimensional matrix of outputs, where the length is defined by the number of memory cells in the layer.
- **The decoder is an LSTM layer that expects a 3D input of [samples, time steps, features] in order to produce a decoded sequence of some different length defined by the problem.**
- We can solve this using a **RepeatVector** layer. This layer simply repeats the provided 2D input multiple times to create a 3D output.
- The **RepeatVector** layer can be used like an adapter to fit the encoder and decoder parts of the network together.
- To summarize, the **RepeatVector** is used as an adapter to fit the fixed-sized 2D output of the encoder to the differing length and 3D input expected by the decoder.
- The **TimeDistributed** wrapper allows the same output layer to be reused for each element in the output sequence.

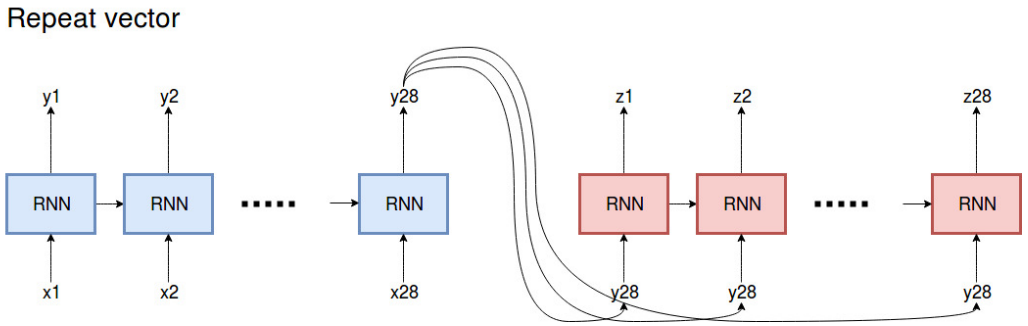
Putting this together, we have:

```
model = Sequential()
model.add(LSTM(..., input_shape=(...)))
model.add(RepeatVector(...))
model.add(LSTM(..., return_sequences=True))
model.add(TimeDistributed(Dense(...)))
```

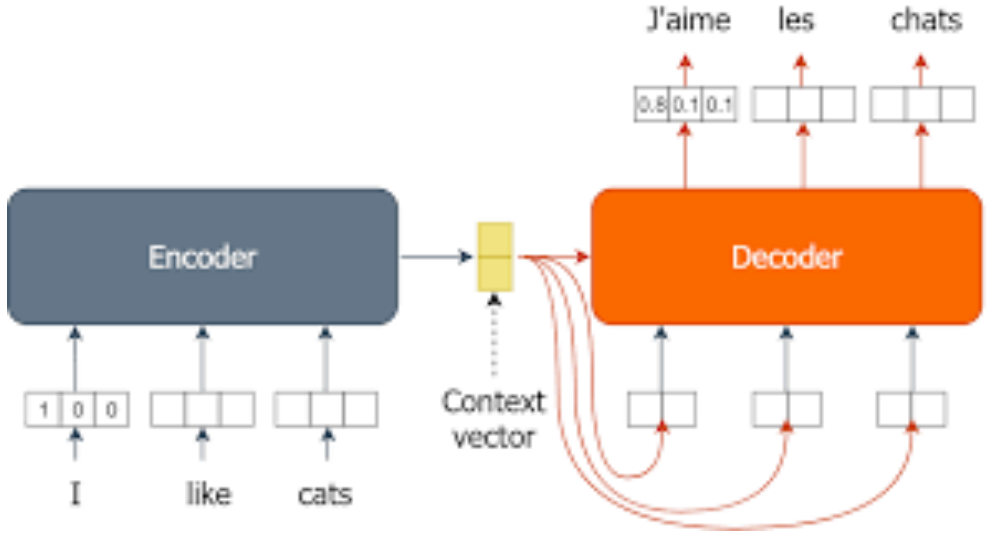
Listing 9.4: Example of an Encoder-Decoder model.

How to Develop Encoder-Decoder LSTMs?

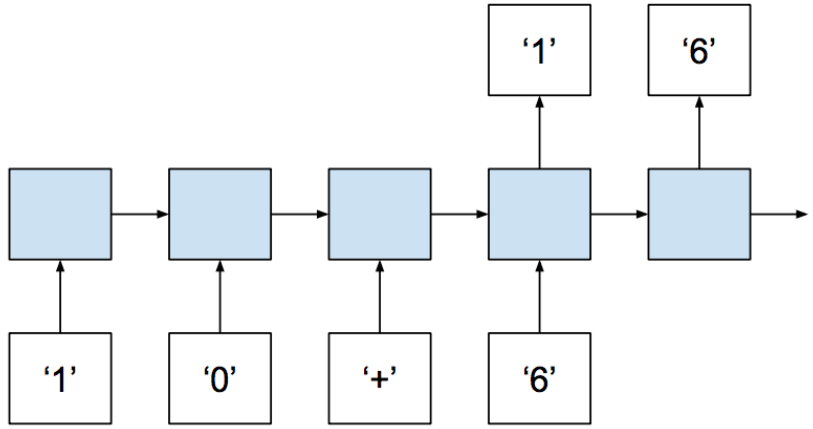
- The **RepeatVector** layer can be used like an adapter to fit the encoder and decoder parts of the network together.



Or



How to Develop Encoder-Decoder LSTMs?



Addition prediction problem framed with a many-to-many prediction model.

```
Input: ['1', '0', '+', '6']
Output: ['1', '6']
```

Example of input and output sequences for the addition problem.

This problem can be implemented in Python. We will divide this into the following steps:

1. Generate Sum Pairs.
2. Integers to Padded Strings.
3. Integer Encoded Sequences.
4. **One Hot Encoded Sequences.**
5. **Sequence Generation Pipeline.**
6. **Decode Sequences.**

Demo….

Procedure of Project Proposal Presentation

- Each group has 10 mins to present plus 5-10mins for Q&A
- PPT presentation file must be uploaded directly on the portal by the end of 14 May.
- Please make sure your code for demonstration is working before the presentation.
- For this presentation, Full score is 10 and it will be given as a group.
- There is 1 hour lecture from 9.10-10.10 and 1 quiz worth (2) score for w13
- The presentation will be started at 10.30 am.