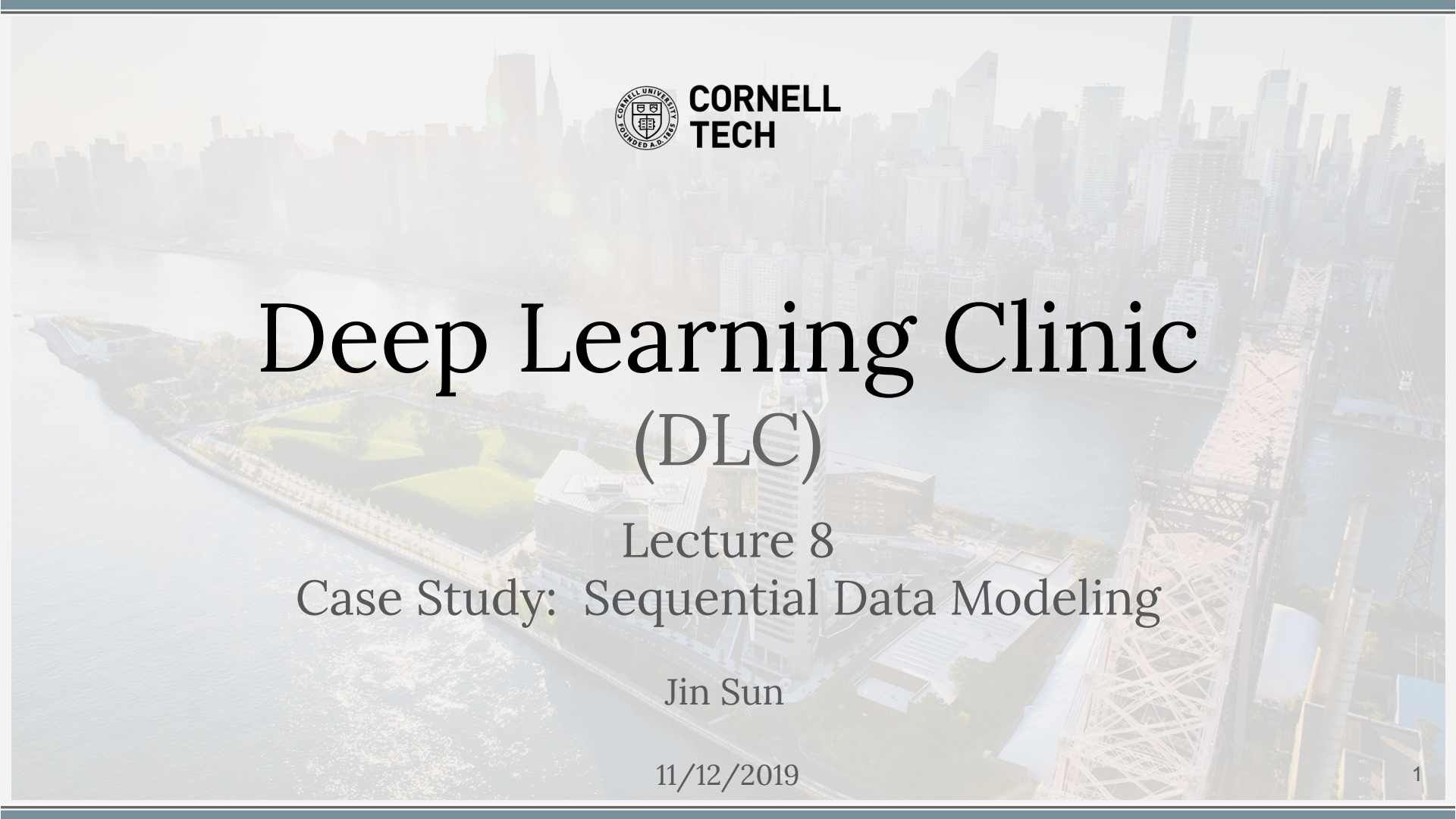


CORNELL TECH

Deep Learning Clinic (DLC)

Lecture 8
Case Study: Sequential Data Modeling

Jin Sun

11/12/2019

Today - Sequential Data Modeling

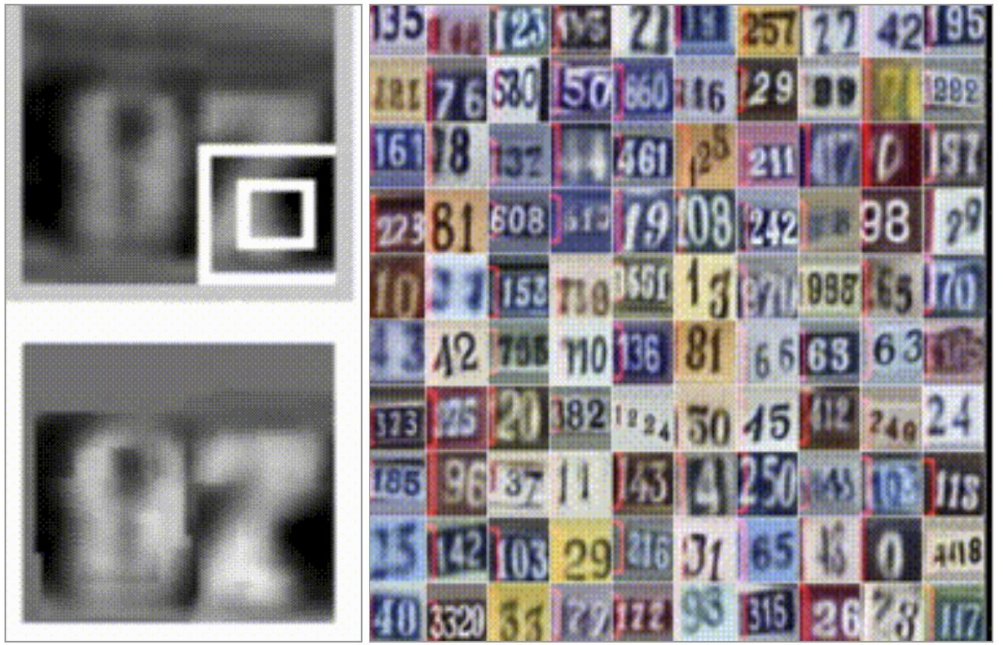
- **Overview**
- Data Preprocessing
- Put LSTM To Work
- Attention Models

Sequential Data

Distribution changes over time

- Sequence of words in an English sentence
- Acoustic features at successive time frames in speech recognition
- Successive frames in video classification
- Rainfall measurements on successive days in Hong Kong
- Daily values of current exchange rate
- Nucleotide base pairs in a strand of DNA
- **Instead of making independent predictions on samples, assume the dependency among samples and make a sequence of decisions for sequential samples**

Or sequential perspective on stationary data

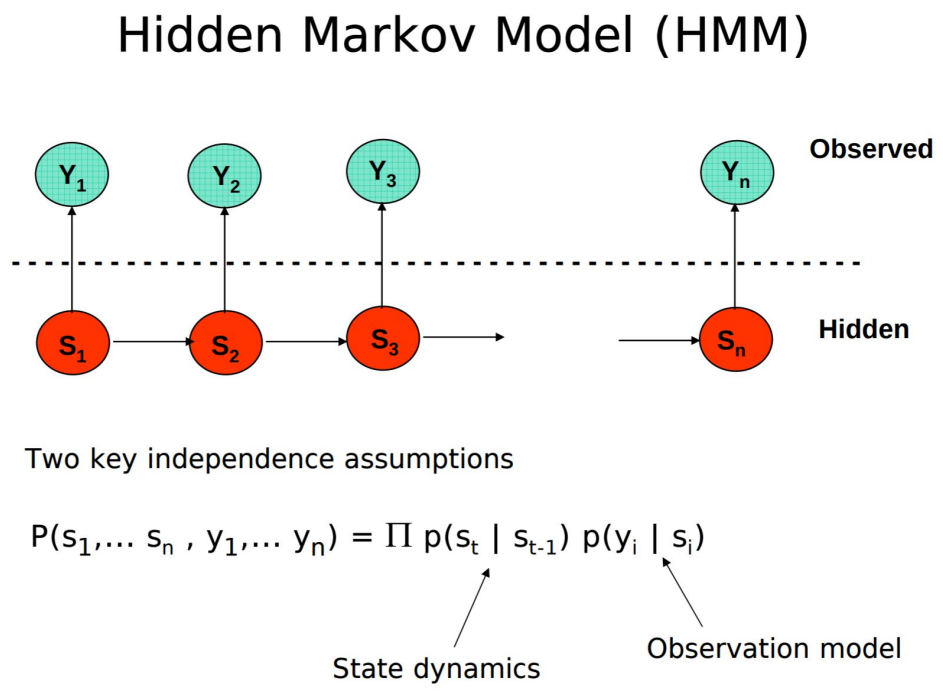


Left: RNN learns to read house numbers. Right: RNN learns to paint house numbers.

Sequential processing in absence of sequences.

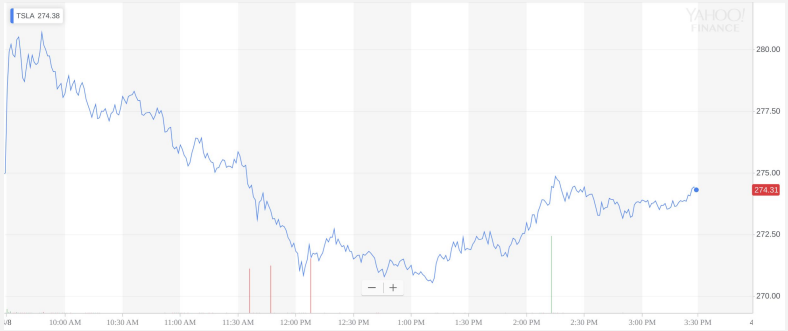
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Pre-deep learning approach



Key components - Data Representation

Raw values? Real number?



Categorical? Embedding?

CitizenX(1995) is the developing world's answer to Silence of the Lambs. Where `Silence' terrorized our peace of mind, `Citizen' exhausts and saddens us instead. This dramatization of the Chikatilo case translates rather well, thanks to a Westernized friendship between two Rostov cops who become equals.

CitizenX may also argue against(!) the death penalty far better than Kevin Spacey's The Life of David Gayle(2002).

Humans are Machiavellian mammals, under which lie limbic brains (lizard-logic). Why did two kids, who knew better, stone to death a toddler they kidnapped? Why do bloodthirsty women yell `li-lililililili' at acts of OBSCENE terrorism? -My own term for this is `limbic domination', the lizard-logic urge to dominate an `enemy'. If you have the words `enemy'/`vengeance' in your vocabulary, you're easily capable of `limbic domination'.

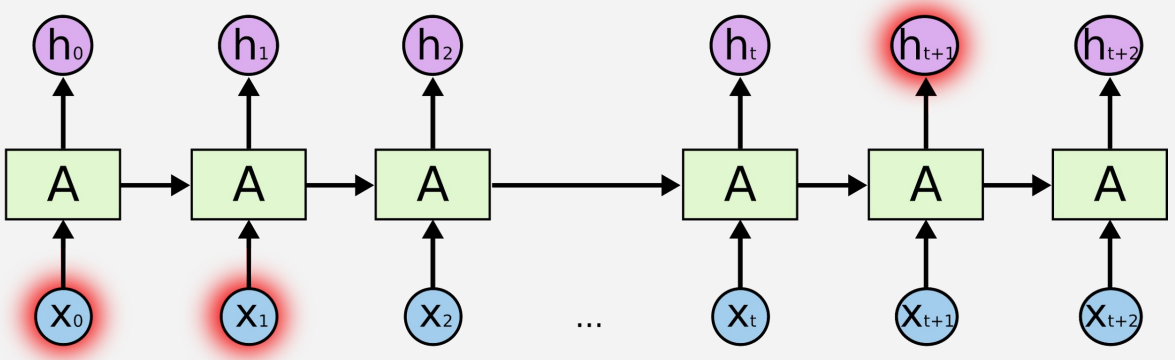
In WWII-devastated 1980s Rostov (located at the mouth of the Don river near the Black Sea)...

Key components - Models

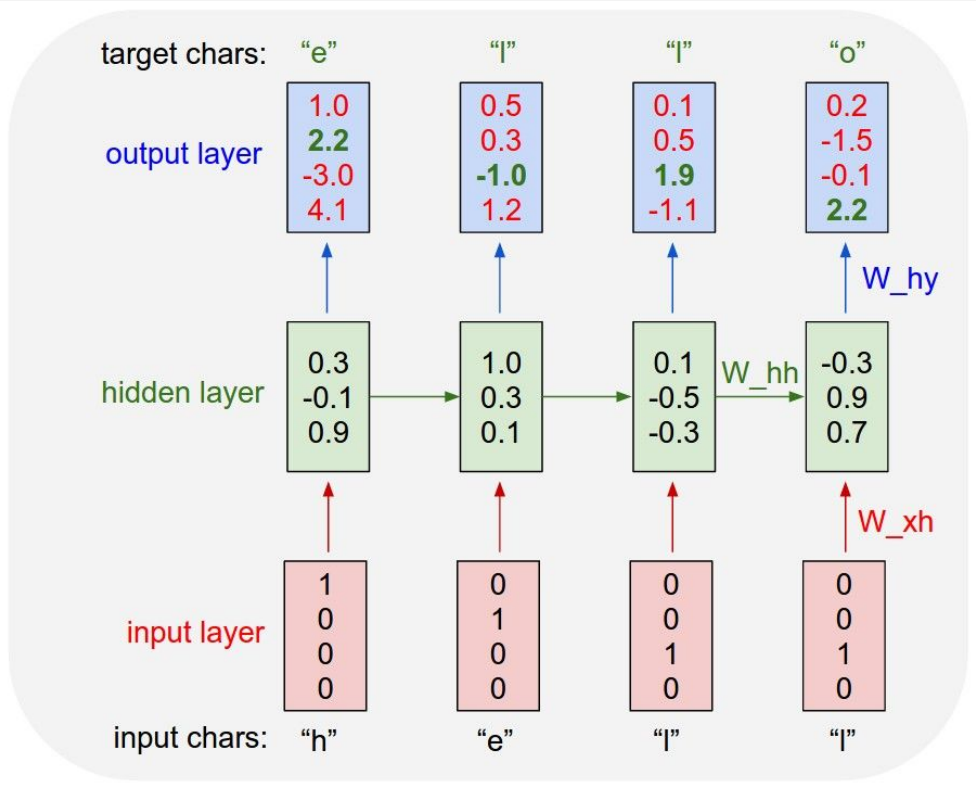
Simple RNNs

LSTM, GRU

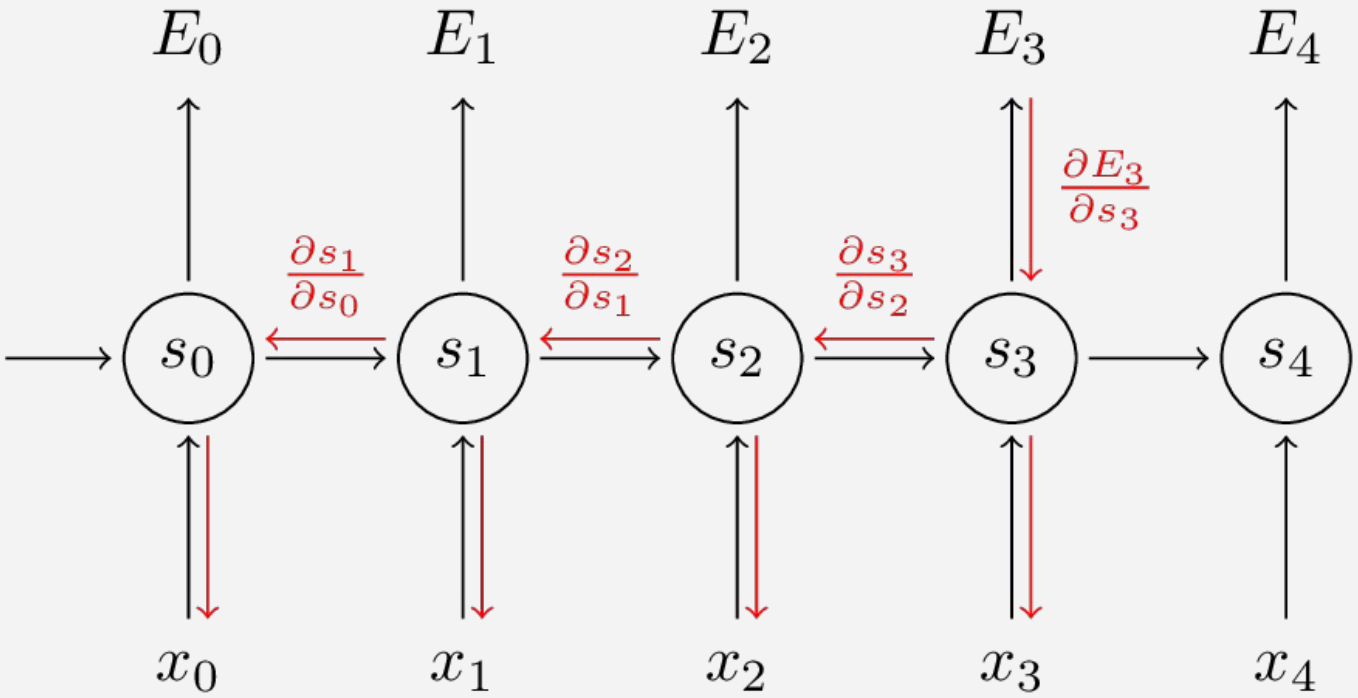
Attention Models



A Simple Recurrent Neural Network



Training on RNNs



https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/

In PyTorch - Defining a RNN

```
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)

        self.i2o = nn.Linear(input_size + hidden_size, output_size)

        self.softmax = nn.LogSoftmax(dim=1)
```

In PyTorch - Defining a RNN

```
class RNN(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

...

    def forward(self, input, hidden):

        combined = torch.cat((input, hidden), 1)

        hidden = self.i2h(combined)

        output = self.i2o(combined)

        output = self.softmax(output)

        return output, hidden

    def initHidden(self):

        return torch.zeros(1, self.hidden_size)
```

In PyTorch - Training Scripts

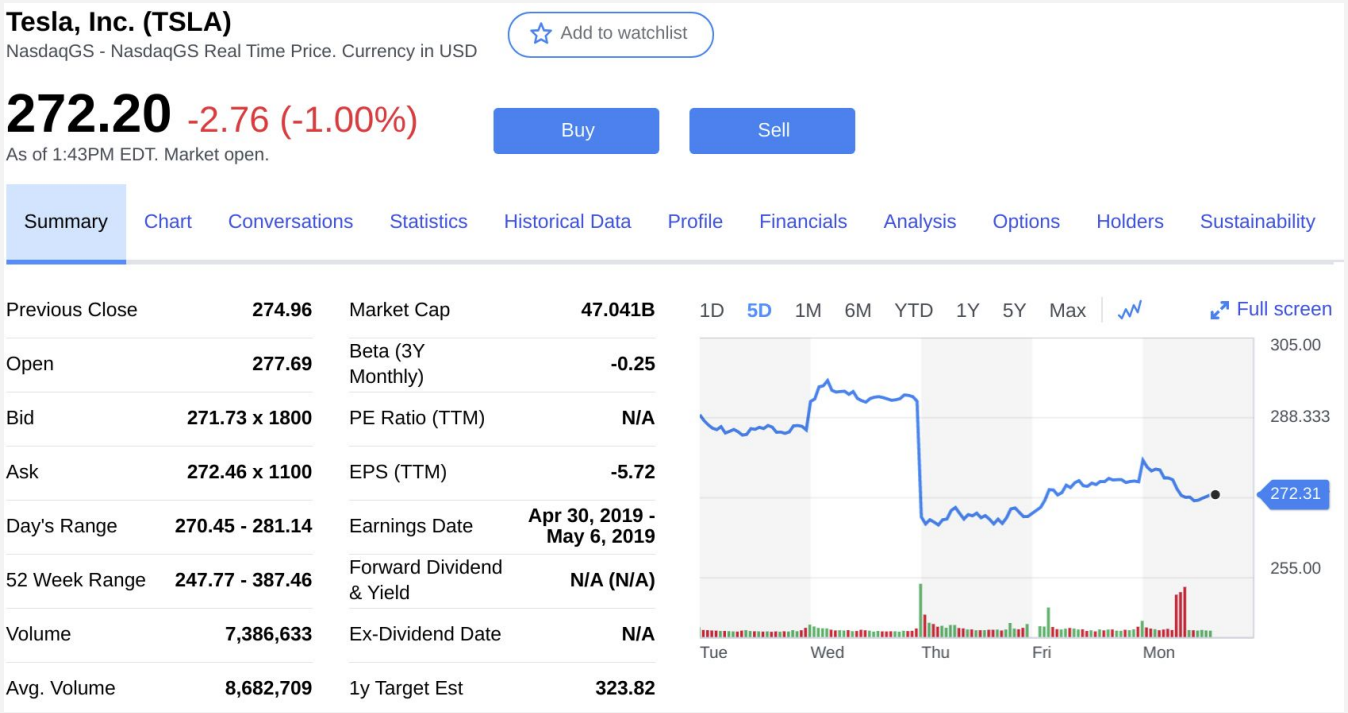
```
learning_rate = 0.005 # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    rnn.zero_grad()
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    loss = criterion(output, category_tensor)
    loss.backward()
    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)
    return output, loss.item()
```

Today - Sequential Data Modeling

- Overview
- **Data Preprocessing**
  - **Real number sequences**
  - Word sequences
- Put LSTM To Work
- Attention Models

Sequential data preprocessing

Real number sequences (temperature, stock, traffic flow, etc) - Time series



Sequential data preprocessing

Real number sequences (temperature, stock, traffic flow, etc)

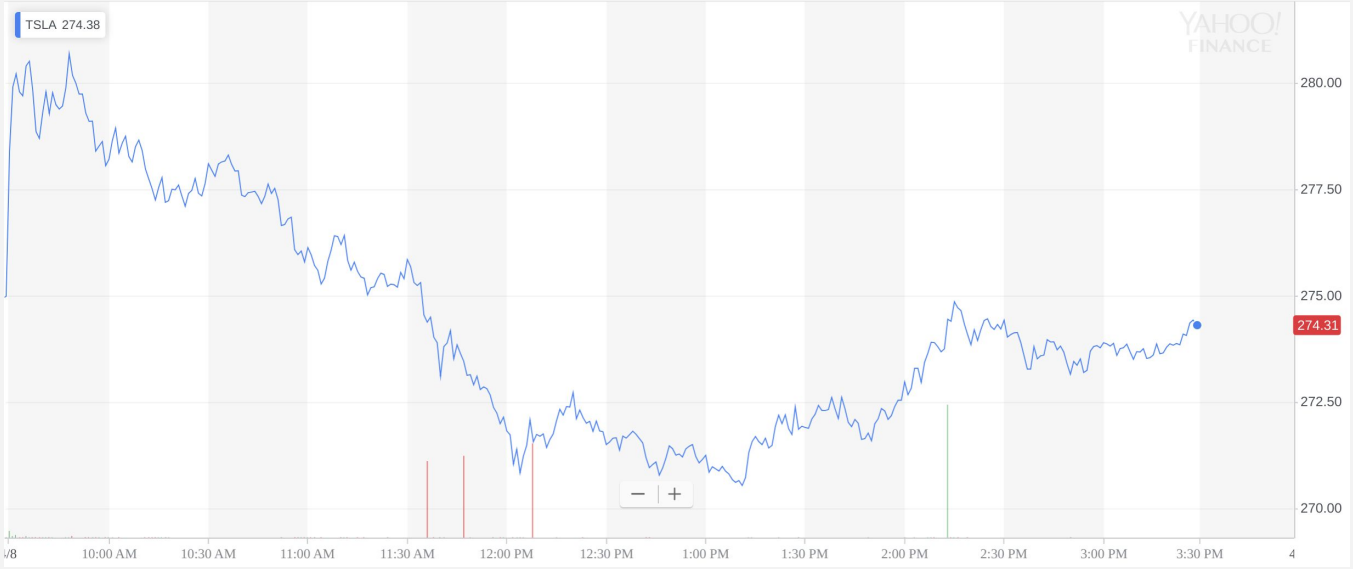
Currency in USD

Download Data

Date	Open	High	Low	Close*	Adj Close**	Volume
Apr 08, 2019	277.69	281.14	270.45	272.06	272.06	7,393,832
Apr 04, 2019	261.89	271.20	260.59	267.78	267.78	23,699,500
Apr 03, 2019	287.32	296.17	287.17	291.81	291.81	7,929,900
Apr 02, 2019	288.30	289.44	283.88	285.88	285.88	5,478,900
Apr 01, 2019	282.62	289.20	281.28	289.18	289.18	8,110,400
Mar 29, 2019	278.70	280.16	274.50	279.86	279.86	5,991,300
Mar 28, 2019	277.16	280.33	275.10	278.62	278.62	6,774,100
Mar 27, 2019	268.75	275.37	268.18	274.83	274.83	8,779,200
Mar 26, 2019	264.44	270.26	264.43	267.77	267.77	7,350,900
Mar 25, 2019	259.71	263.18	254.46	260.42	260.42	10,215,000
Mar 22, 2019	272.58	272.80	264.00	264.53	264.53	8,745,600
Mar 21, 2019	272.60	276.45	268.45	274.02	274.02	5,947,100
Mar 20, 2019	269.69	274.97	266.30	273.60	273.60	6,908,200

Temporal sequence feature engineering

Features to capture characteristics of sequence.



Day

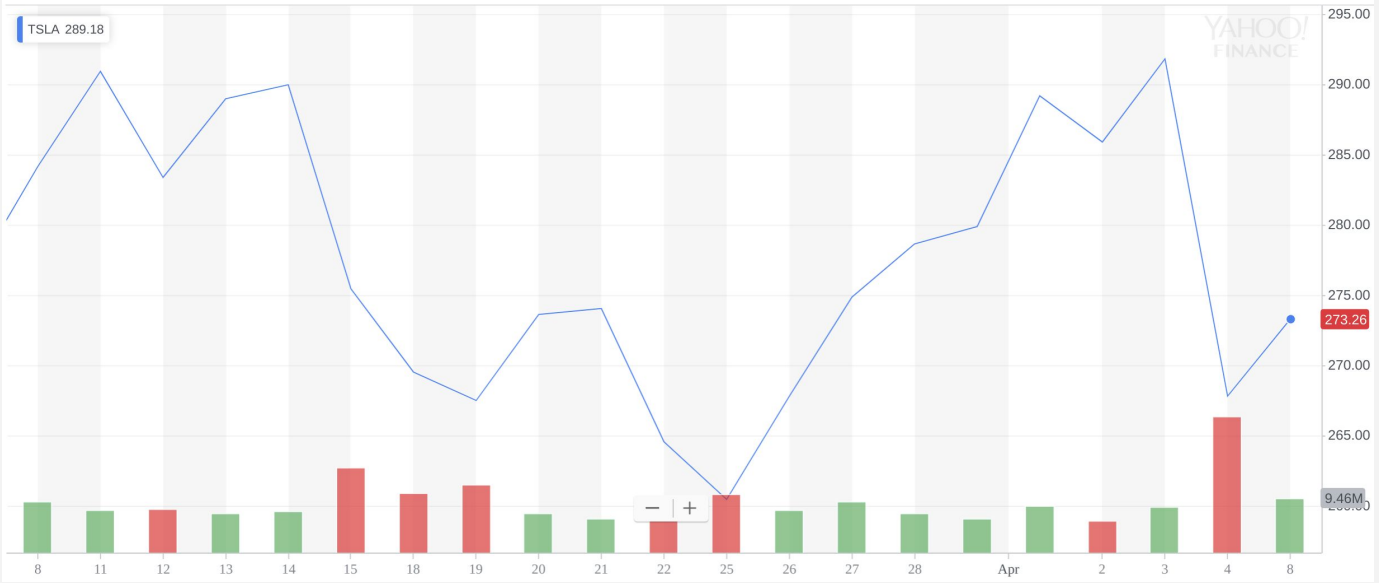
Temporal sequence feature engineering

Features to capture characteristics of sequence.

Week

Temporal sequence feature engineering

Features to capture characteristics of sequence.



Month

Temporal sequence feature engineering

Features to capture characteristics of sequence:

Learning temporal dependency is hard.

Good features make learning easier.

Example of features:

Statistics of various length (resampling, rolling window stats), autocorrelation, 1D signal analysis (e.g., wavelet)

Quick introduction to

Load data using Pandas:

In [38]:
```
import pandas as pd
import numpy as np
```

In [163]:
```
data = pd.read_csv('./data/TSLAHistorical NOCP 6m.csv', parse_dates=True, index_col='Trade DATE')
print(data.head())
```

```
            Symbol    NOCP
Trade DATE
2019-04-03   TSLA  291.81
2019-04-02   TSLA  285.88
2019-04-01   TSLA  289.18
2019-03-29   TSLA  279.86
2019-03-28   TSLA  278.62
```

Pandas stores data in special format and retrieves them by time stamp.

```
print(data['2019-04-03'])
```

```
            Symbol    NOCP
Trade DATE
2019-04-03   TSLA  291.81
```

Quick introduction to Pandas

Data resampling:

B: business day

W: week

M: month

Q: quarter

Y: year

```
In [174]: daily = data.resample('B').mean()
          print(daily.head())
```

```
                NOCP
Trade DATE
2018-10-03  294.80
2018-10-04  281.83
2018-10-05  261.95
2018-10-08  250.56
2018-10-09  262.80
```

```
In [178]: weekly = data.resample('W').mean()
          print(weekly.head())
```

```
                    NOCP
Trade DATE
2018-10-07  279.526667
2018-10-14  256.250000
2018-10-21  266.374000
2018-10-28  297.870000
2018-11-04  338.552000
```

```
In [172]: monthly = data.resample('M').mean()
          print(monthly.head())
```

```
                    NOCP
Trade DATE
2018-10-31  283.005714
2018-11-30  344.495238
2018-12-31  344.109211
2019-01-31  318.494286
2019-02-28  307.728421
```

Quick introduction to Pandas

Rolling Window Statistics (more flexible way of defining various length)

```
print(data[0:100].drop(columns='Symbol').rolling(3).mean().head(10))
```

```
                 NOCP
Trade DATE
2019-04-03        NaN
2019-04-02        NaN
2019-04-01  288.956667
2019-03-29  284.973333
2019-03-28  282.553333
2019-03-27  277.770000
2019-03-26  273.740000
2019-03-25  267.673333
2019-03-22  264.240000
2019-03-21  266.323333
```

Notice the ‘NaN’s: you won’t get result in [L-k+2, L] for rolling(k).

You can set min_periods to fill in the values.

More complex features

https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

`abs_energy` (x)	Returns the absolute energy of the time se
`absolute_sum_of_changes` (x)	Returns the sum over the absolute value of
`agg_autocorrelation` (x, param)	Calculates the value of an aggregation func
`agg_linear_trend` (x, param)	Calculates a linear least-squares regression
`approximate_entropy` (x, m, r)	Implements a vectorized Approximate entr
`ar_coefficient` (x, param)	This feature calculator fits the unconditiona
`augmented_dickey_fuller` (x, param)	The Augmented Dickey-Fuller test is a hypo
`autocorrelation` (x, lag)	Calculates the autocorrelation of the specif
`binned_entropy` (x, max_bins)	First bins the values of x into max_bins equ
`c3` (x, lag)	This function calculates the value of
`change_quantiles` (x, ql, qh, isabs, f_agg)	First fixes a corridor given by the quantiles
`cid_ce` (x, normalize)	This function calculator is an estimate for a
`count_above_mean` (x)	Returns the number of values in x that are
`count_below_mean` (x)	Returns the number of values in x that are
`cwt_coefficients` (x, param)	Calculates a Continuous wavelet transform

Today - Sequential Data Modeling

- Overview
- **Data Preprocessing**
  - Real number sequences
  - **Word sequences**
- Put LSTM To Work
- Attention Models

Sequential data preprocessing

Categorical sequences (Natural language, 8-bit images, etc)

Clean your text

Convert text to lower case

Use regular expression to clean characters from raw text

CitizenX(1995) is the developing world's answer to Silence of the Lambs. Where `Silence' terrorized our peace of mind, `Citizen' exhausts and saddens us instead. This dramatization of the Chikatilo case translates rather well, thanks to a Westernized friendship between two Rostov cops who become equals.

CitizenX may also argue against(!) the death penalty far better than Kevin Spacey's The Life of David Gayle(2002).

Humans are Machiavellian mammals, under which lie limbic brains (lizard-logic). Why did two kids, who knew better, stone to death a toddler they kidnapped? Why do bloodthirsty women yell `li-lililililili' at acts of OBSCENE terrorism? -My own term for this is `limbic domination', the lizard-logic urge to dominate an `enemy'. If you have the words `enemy'/`vengeance' in your vocabulary, you're easily capable of `limbic domination'.

In WWII-devastated 1980s Rostov (located at the mouth of the Don river near the Black Sea), nothing suppressed Andrei Chikatilo's urge for `limbic domination' from overpowering his layers of civilization. Chikatilo(Jeffrey DeMunn)'s easy victims were paupers, usually children, who rode the interurban train for fun, since they couldn't afford anything else.

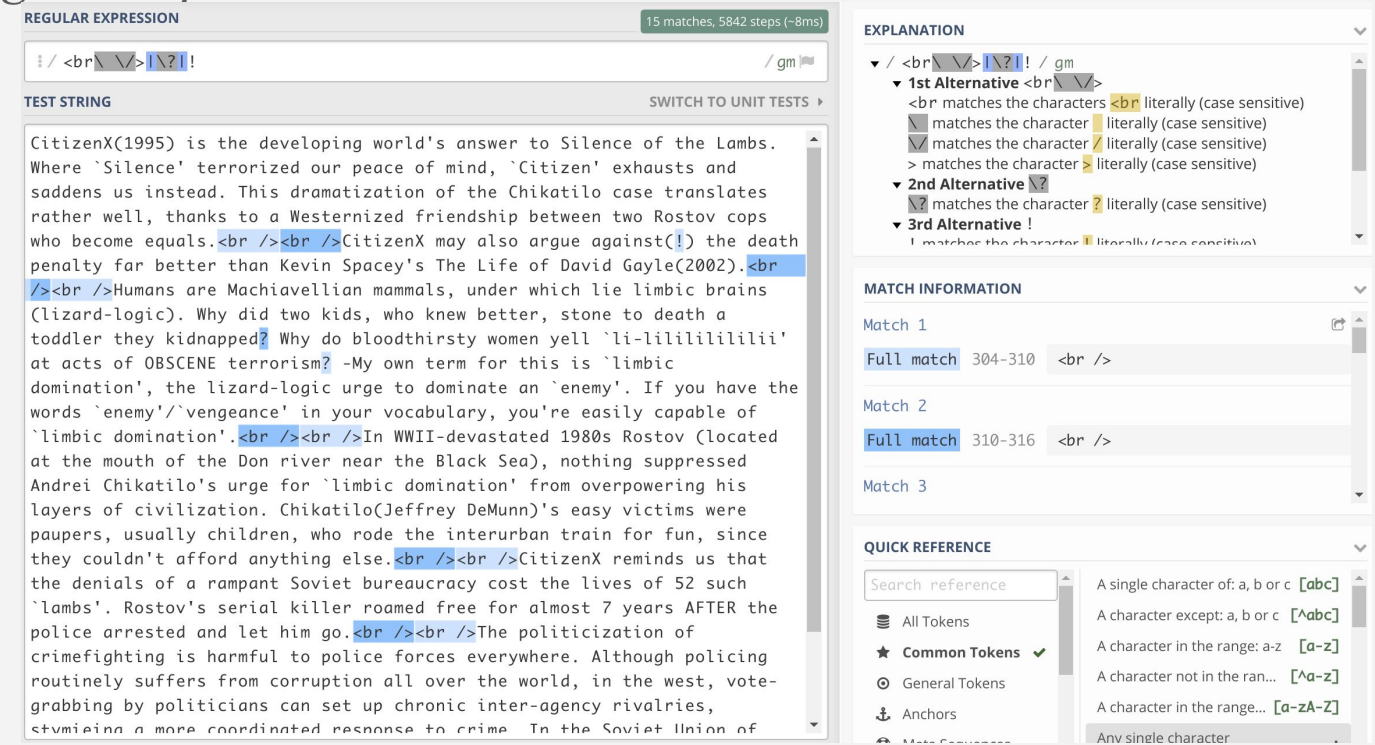
CitizenX reminds us that the denials of a rampant Soviet bureaucracy cost the lives of 52 such `lambs'. Rostov's serial killer roamed free for almost 7 years AFTER the police arrested and let him go.

The politicization of crimefighting is harmful to police forces everywhere. Although policing routinely suffers from corruption all over the world, in the west, vote-grabbing by politicians can set up chronic inter-agency rivalries, stymieing a more coordinated response to crime. In the Soviet Union of CitizenX, however, Viktor Burakov(Stephen Rea)'s Killer Department was suffering from a repressive bureaucracy.

Geoffrey DeMunn plays the psychosexually inadequate Chikatilo with faultless but understated

Clean your text

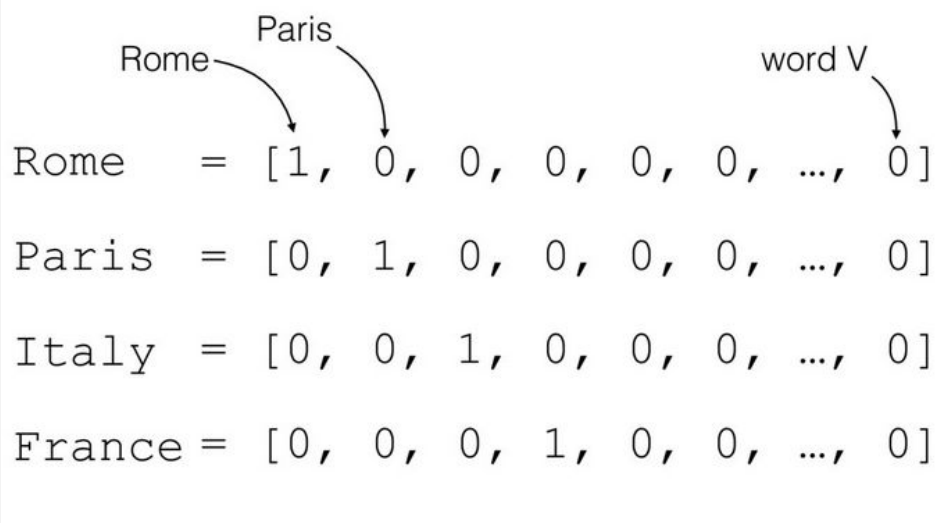
Use regular expression to clean characters from raw text https://regex101.com/



Encode words

Neural networks only understand numbers.

Each word needs to be represented by a one-hot vector.



Build a word vocabulary

This requires a dictionary that maps a word to a vector.

```
word_idx = my_vocab['Paris']

one_hot_encoding_vec[word_idx] = 1    #[0, 1, 0, 0, …, 0]
```

You can build a straight-forward dict by keeping records of unique words.

Or you can use the following procedure:

1. Collect frequencies of appearance of each word.
2. Build a python dict based on the freq count so that more common words have smaller index values.

   You might see performance difference for large dict.

Word embedding in Pytorch

Drawback of one-hot encoding:

Dimension is the same as the vocabulary/dictionary. Think about how many words are there in English!

[More Importantly] It assumes words are independent from each other, which they aren't.

For example:

'Paris' and 'Rome' are similar words; 'Paris' and 'Cat' are much different.

Word embedding in Pytorch

Usually it is better to represent word as continuous vectors called embedding that:

1) It has arbitrary dimension (a hyper-parameter).
2) Similar words are close to each other in the embedded space.

This can be learned by using nn.Embedding as a neural network layer:

...embeddings are stored as a |V|×D matrix, where D is the dimensionality of the embeddings, such that the word assigned index i has its embedding stored in the i'th row of the matrix.

Word embedding in Pytorch

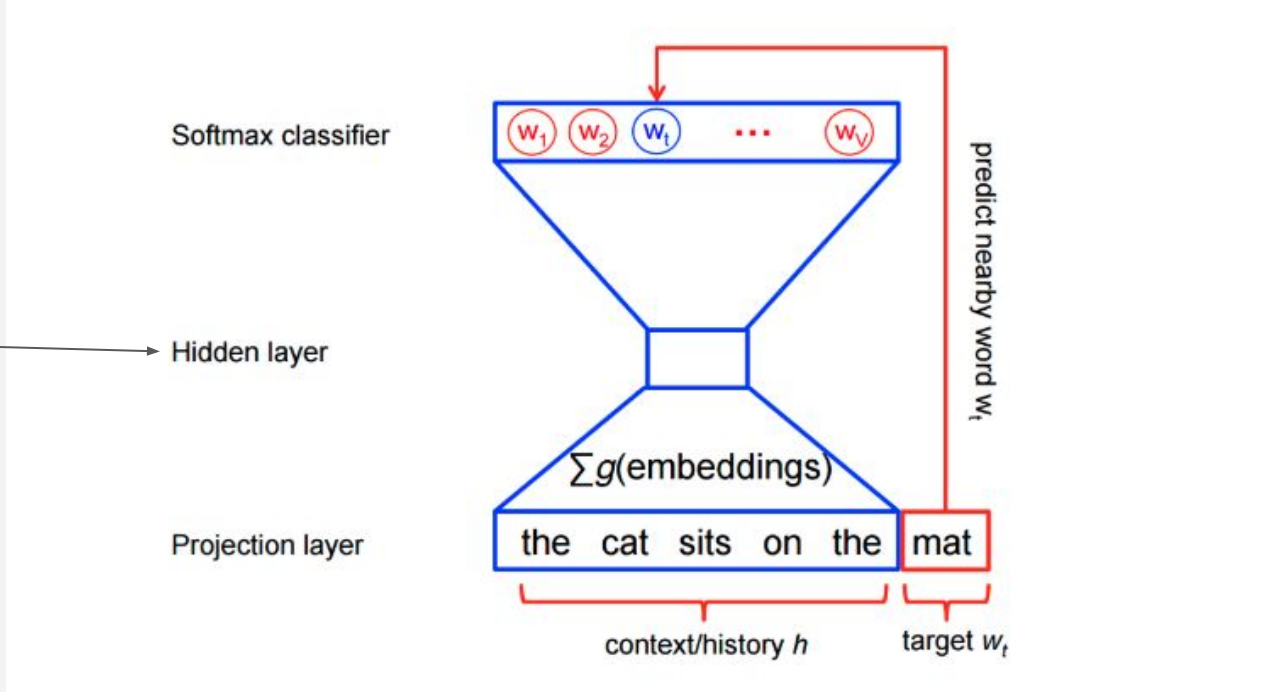
```
word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5)  # 2 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.tensor([word_to_ix["hello"]], dtype=torch.long)
hello_embed = embeds(lookup_tensor)
print(hello_embed)
```

Out:

```
tensor([[ 0.6614,  0.2669,  0.0617,  0.6213, -0.4519]],
       grad_fn=<EmbeddingBackward>)
```

Word2vec model

Learned embedding



Embedded vector space is very intuitive. For example: ‘king’ - ‘man’ + ‘woman’ = ‘queen’

Load a pre-trained embedding in Pytorch

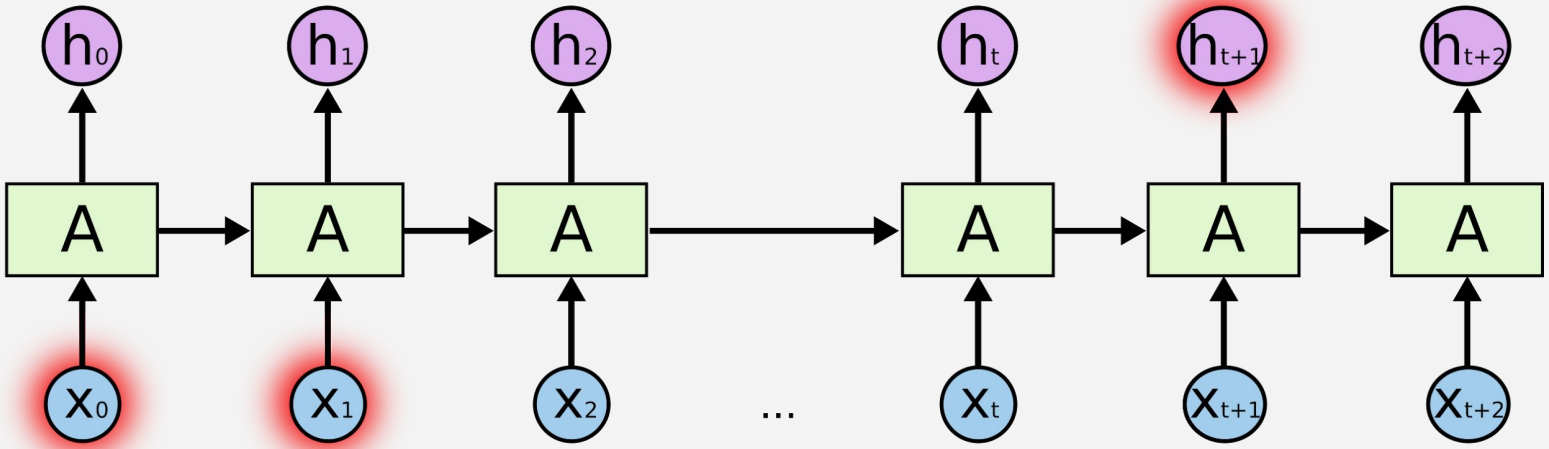
```
>>> # FloatTensor containing pretrained weights
>>> weight = torch.FloatTensor([[1, 2.3, 3], [4, 5.1, 6.3]])
>>> embedding = nn.Embedding.from_pretrained(weight)
>>> # Get embeddings for index 1
>>> input = torch.LongTensor([1])
>>> embedding(input)
tensor([[ 4.0000,  5.1000,  6.3000]])
```

You can download any pre-trained weight embedding matrix and use them in your pytorch code.

Today - Sequential Data Modeling

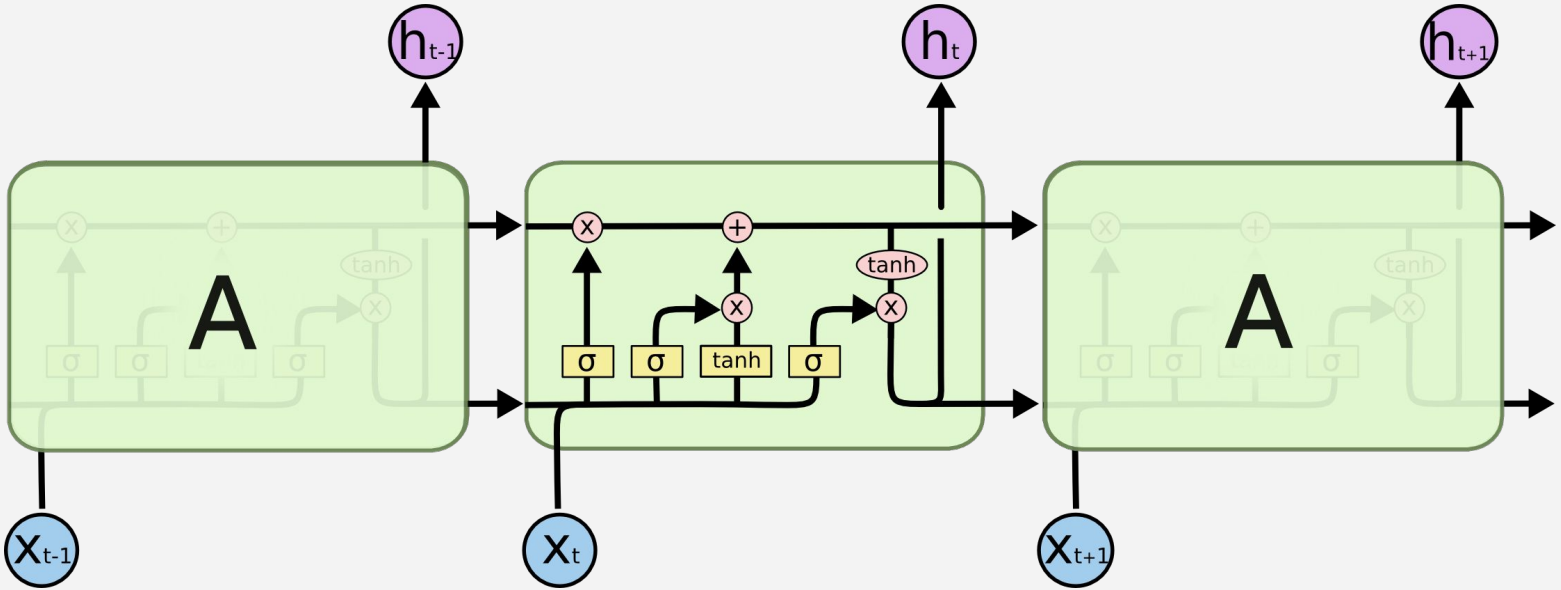
- Overview
- Data Preprocessing
- **Put LSTM To Work**
- Attention Models

Learning long term dependency is hard!



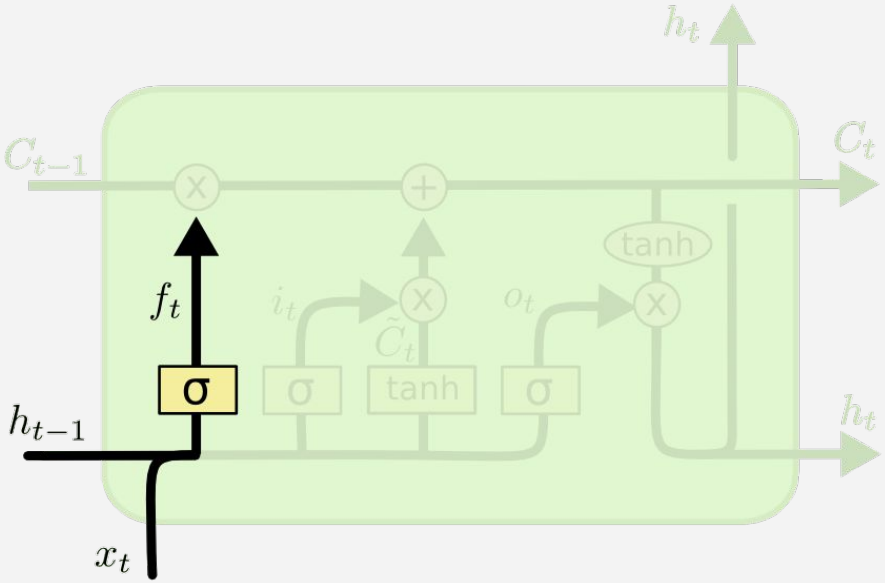
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory (LSTM)



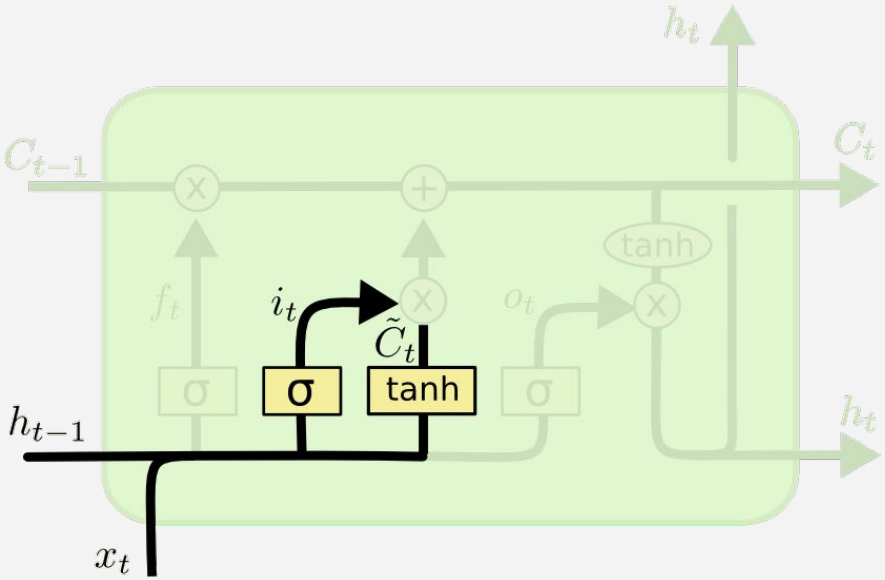
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget gate



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

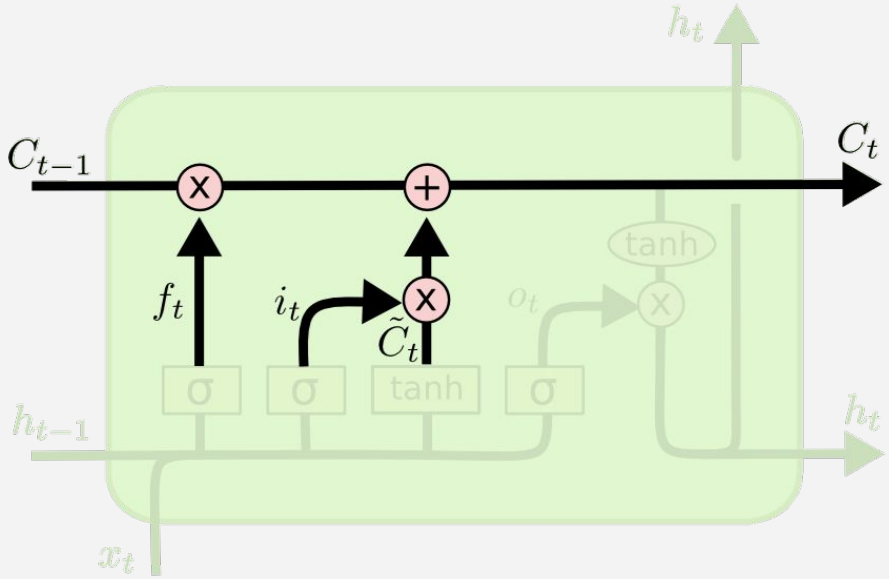
Store info in cell states



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

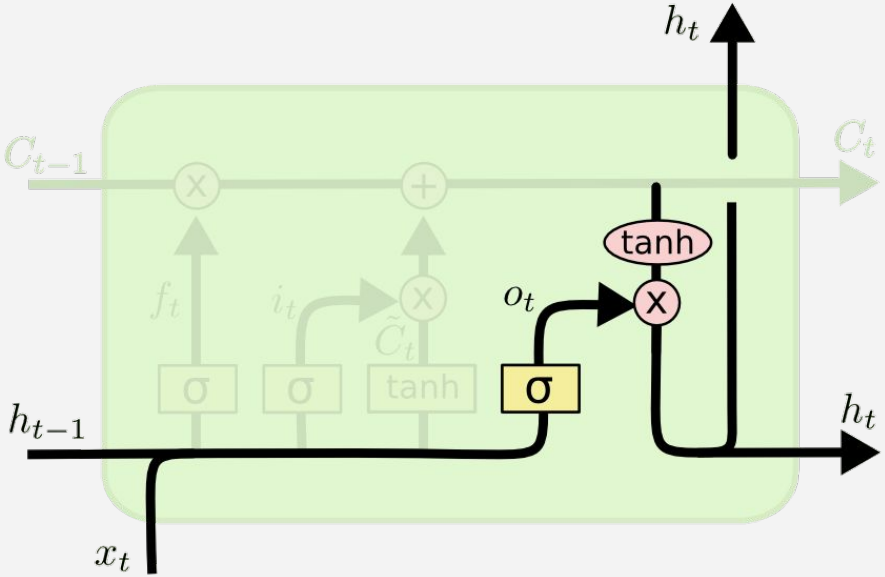
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Update cell states



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

Get output



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

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

LSTM in Pytorch

CLASS `torch.nn.LSTM(*args, **kwargs)`

```
>>> rnn = nn.LSTM(10, 20, 2) # (input, hidden, num_stack)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(1, 3, 20)
>>> c0 = torch.randn(1, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

Parameters explained

Parameters:

- **input_size – The number of expected features in the input x**
- **hidden_size – The number of features in the hidden state h**
- num_layers – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- bias – If `False`, then the layer does not use bias weights b_ih and b_hh. Default: `True`
- batch_first – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- dropout – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- bidirectional – If `True`, becomes a bidirectional LSTM. Default: `False`

The input size is the word embedding vector's dimension. The hidden size is a hyper-parameter.

Parameters explained

Parameters:

- input_size – The number of expected features in the input x
- hidden_size – The number of features in the hidden state h
- **num_layers – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1**
- bias – If `False`, then the layer does not use bias weights b_ih and b_hh. Default: `True`
- batch_first – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- dropout – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- bidirectional – If `True`, becomes a bidirectional LSTM. Default: `False`

LSTM can be stacked to each other. They are not shared-weights.

Parameters explained

Parameters:

- input_size – The number of expected features in the input x
- hidden_size – The number of features in the hidden state h
- num_layers – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- bias – If `False`, then the layer does not use bias weights b_ih and b_hh. Default: `True`
- **batch_first – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`**
- dropout – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- bidirectional – If `True`, becomes a bidirectional LSTM. Default: `False`

Recommend to set batch_first to True so that a batch_size=3, embedding_dim=300, sequence_length=10 sequence has shape [3,10,300]

Parameters explained

Parameters:

- input_size – The number of expected features in the input x
- hidden_size – The number of features in the hidden state h
- num_layers – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a stacked LSTM, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- bias – If `False`, then the layer does not use bias weights b_ih and b_hh. Default: `True`
- batch_first – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout – If non-zero, introduces a Dropout layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0**
- **bidirectional – If `True`, becomes a bidirectional LSTM. Default: `False`**

Some extra things to tweak if you want to get maximum performance.

LSTM example code and init_hidden

https://www.jessicayung.com/lstms-for-time-series-in-pytorch/

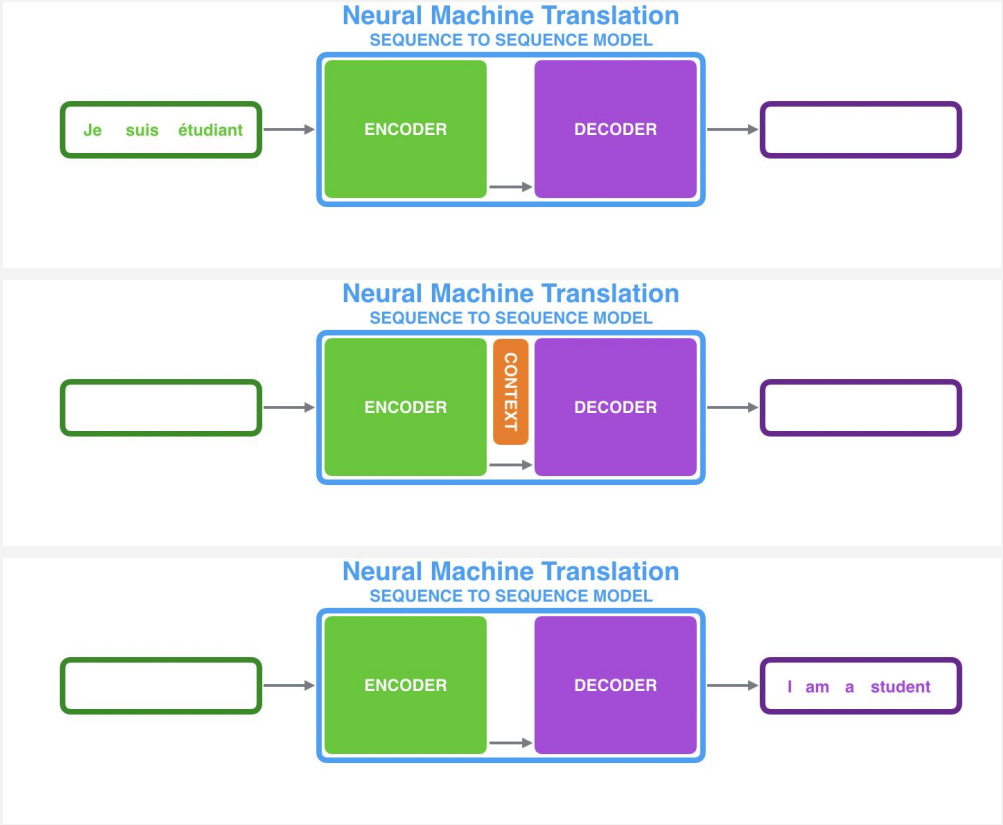
```
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, batch_size,
output_dim=1, num_layers=2):
        super(LSTM, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.num_layers = num_layers
        # Define the LSTM layer
        self.lstm = nn.LSTM(self.input_dim, self.hidden_dim,
self.num_layers)
        # Define the output layer
        self.linear = nn.Linear(self.hidden_dim, output_dim)

    def init_hidden(self):
        return (torch.zeros(self.num_layers,self.batch_size,
self.hidden_dim), torch.zeros(self.num_layers,
self.batch_size, self.hidden_dim))
```

```
    def forward(self, input):
        # Forward pass through LSTM layer
        # shape of lstm_out: [input_size, batch_size,
hidden_dim]
        # shape of self.hidden: (a, b), where a and b both
        # have shape (num_layers, batch_size, hidden_dim).
        lstm_out, self.hidden =
self.lstm(input.view(len(input), self.batch_size, -1))

        # Only take the output from the final timetep
        # Can pass on the entirety of lstm_out to the next
layer if it is a seq2seq prediction
        y_pred =
self.linear(lstm_out[-1].view(self.batch_size, -1))
        return y_pred.view(-1)
```

Today - Sequential Data Modeling

- Overview
- Data Preprocessing
- Put LSTM To Work
- **Attention Models**

Seq2seq review

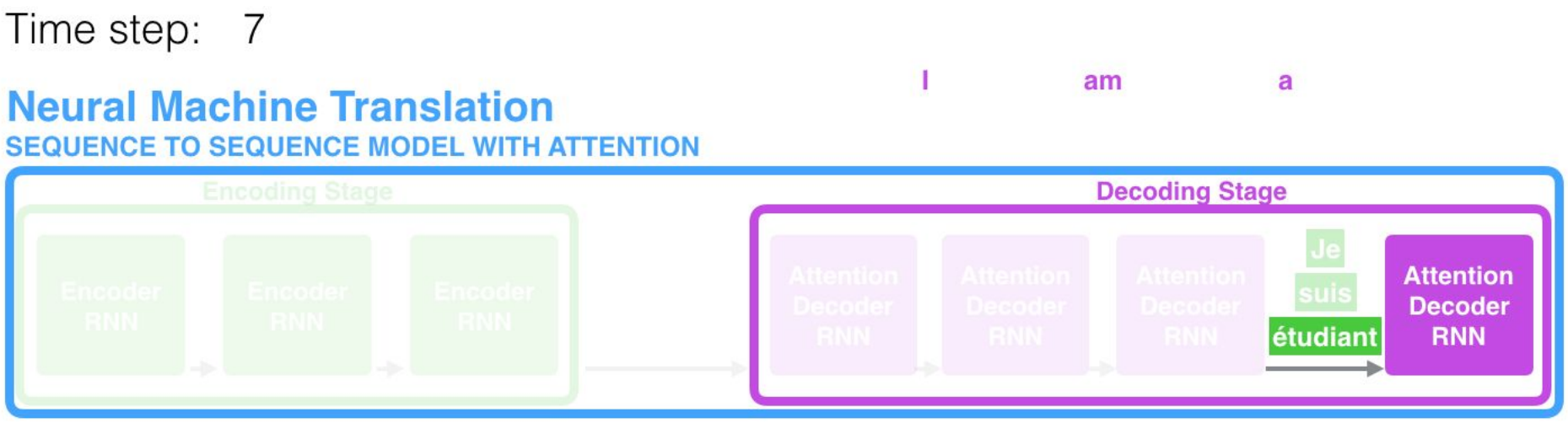


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Seq2seq

Attention model in neural translation

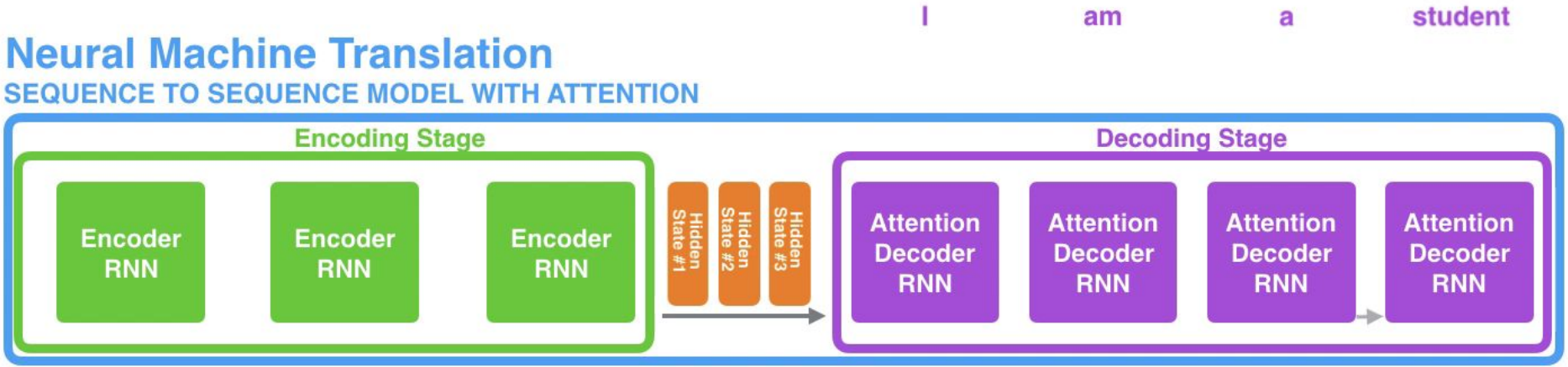
Use attention in neural translation (high level idea)



Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473* (2014).

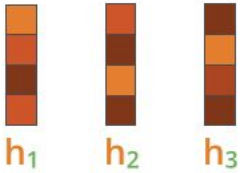
Attention model vs Seq2seq

1. Instead of passing over one hidden state, pass them all:



2. Focus on part of all hidden states each time:

1. Prepare inputs



Encoder hidden states

Decoder hidden state at time step 4

2. Score each hidden state

13	9	9

scores
Attention weights for decoder time step #4

3. Softmax the scores

0.96	0.02	0.02

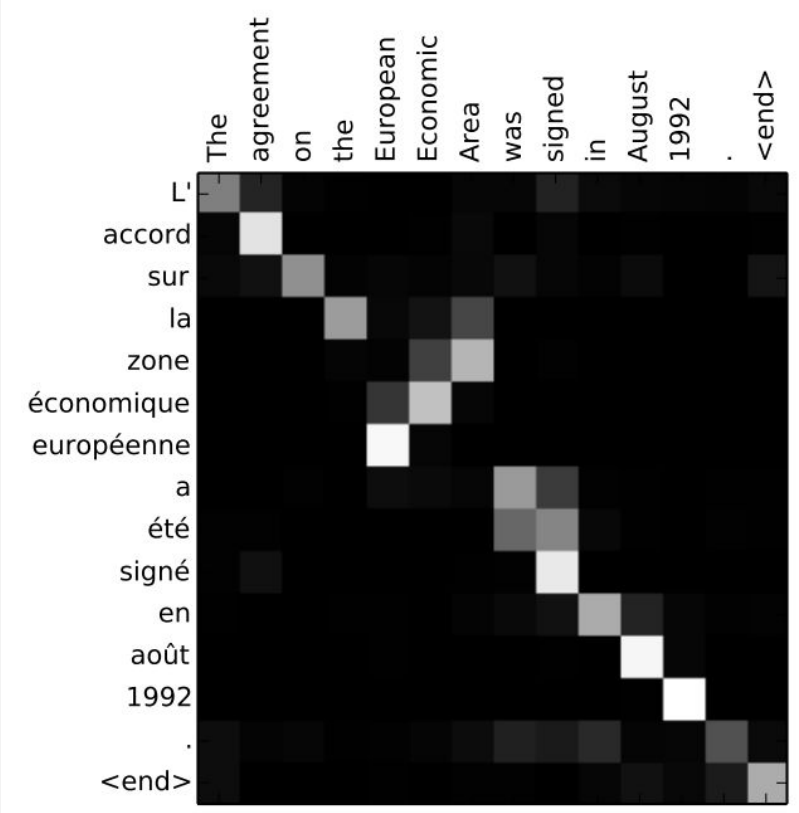
softmax scores

4. Multiply each vector by its softmaxed score

5. Sum up the weighted vectors

Context vector for decoder time step #4

Each output attends to different part of input



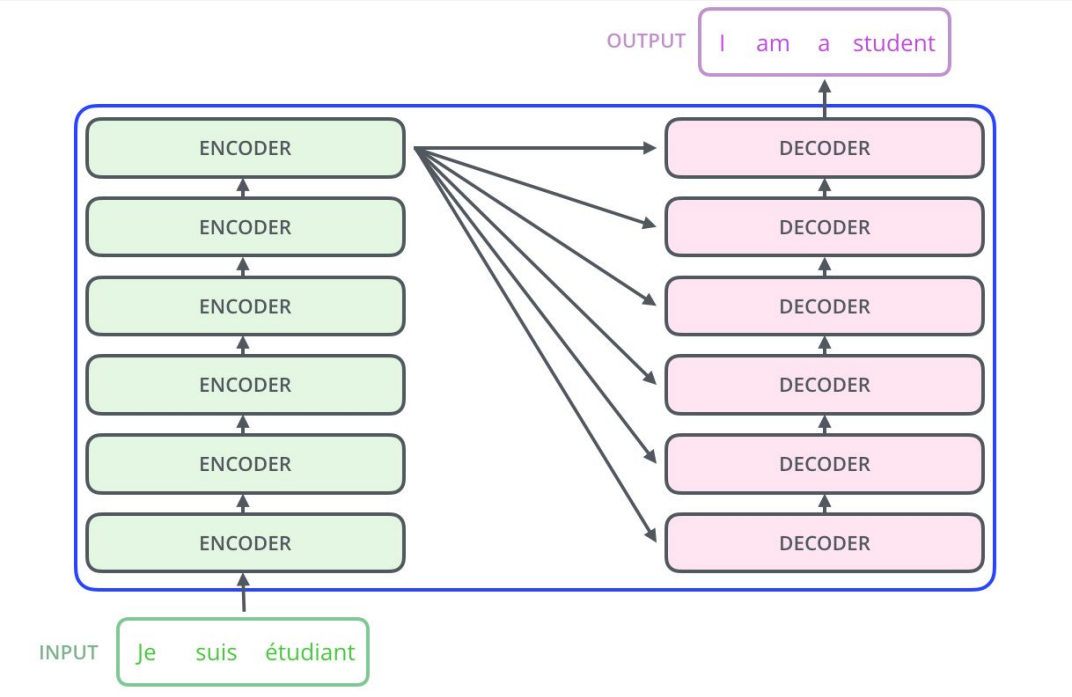
Attention is all you need

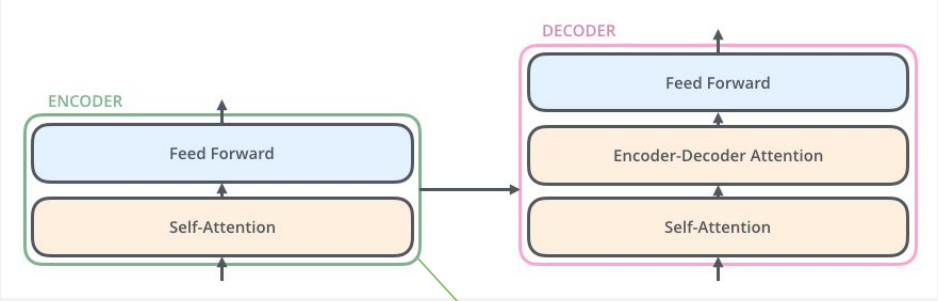
Resources:

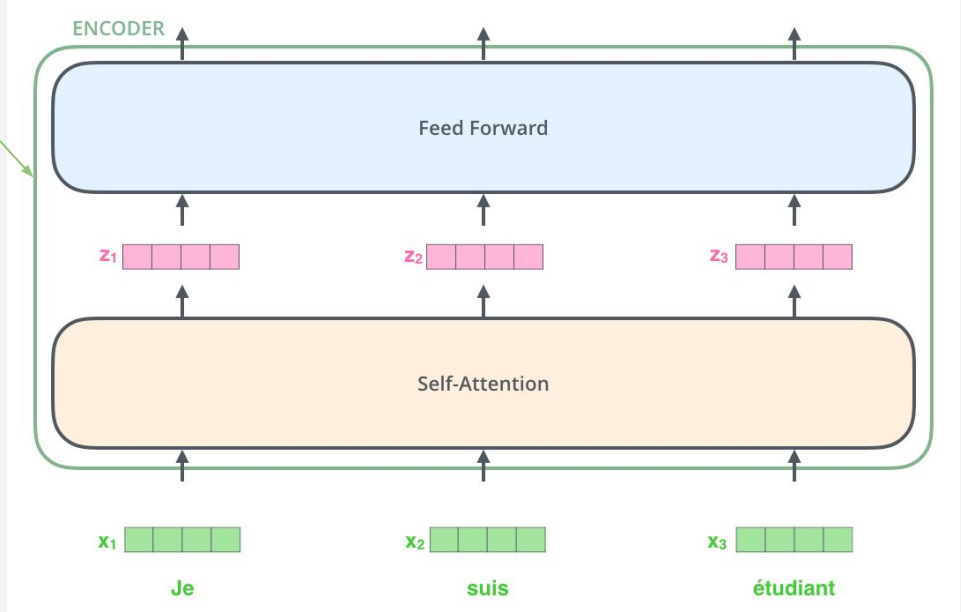
http://jalammar.github.io/illustrated-transformer/

https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

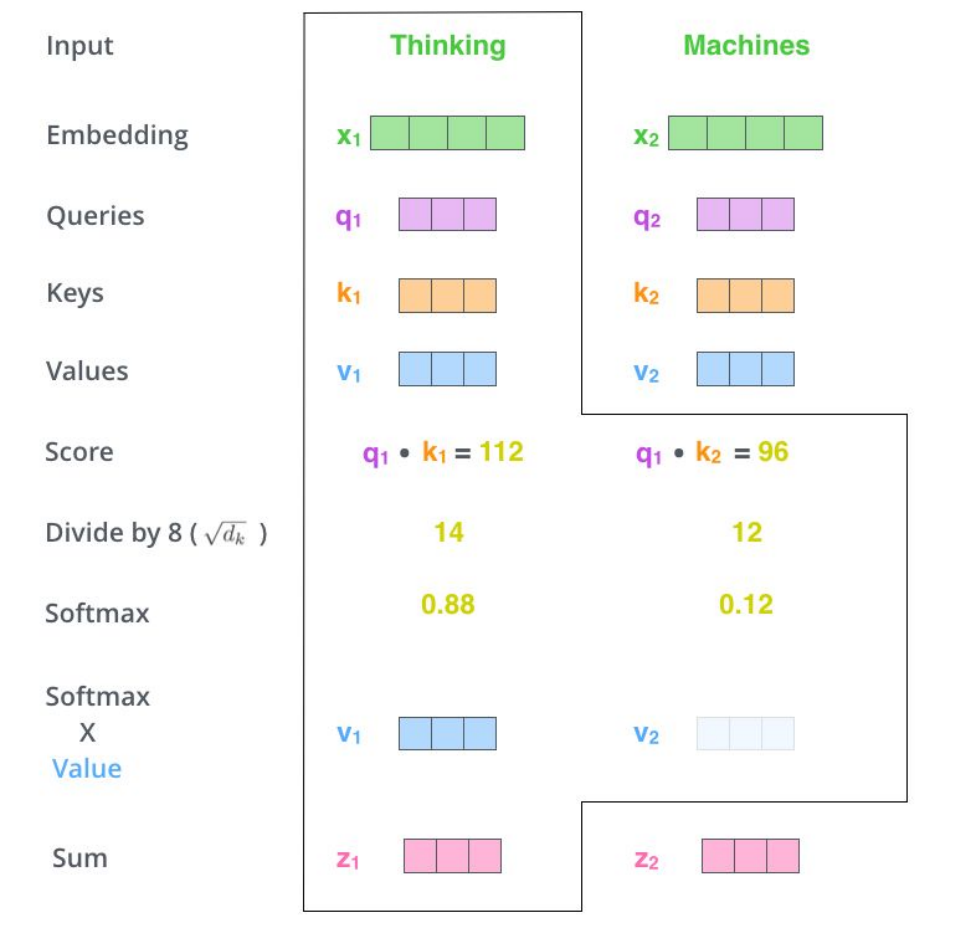
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

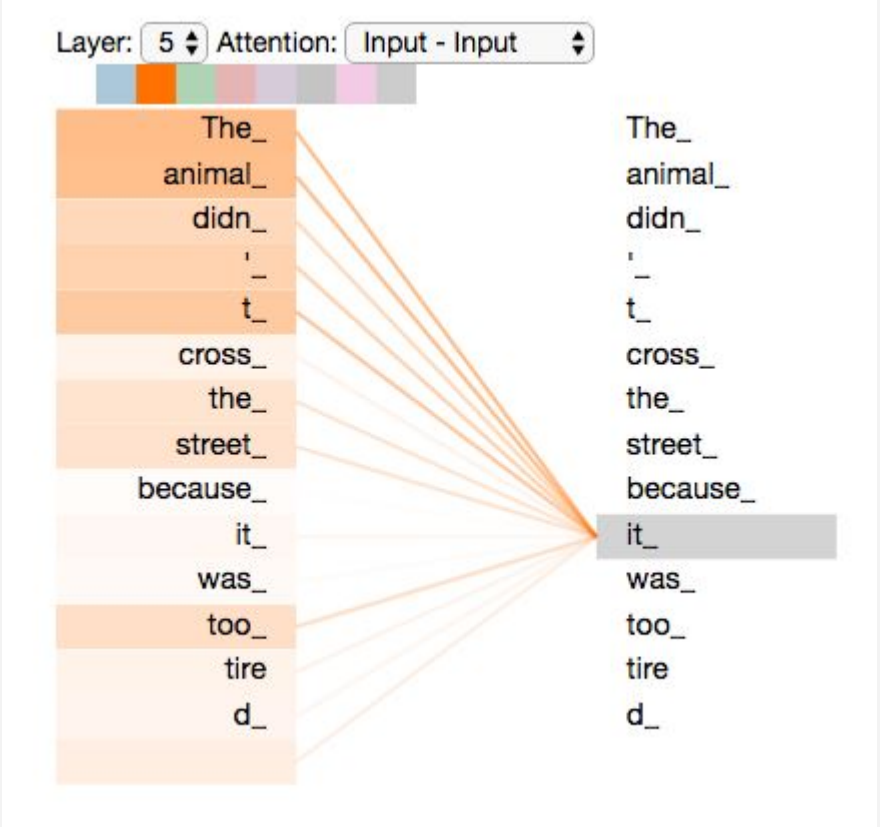



DECODER
ENCODER
Feed Forward
Encoder-Decoder Attention
Feed Forward
Self-Attention
Self-Attention

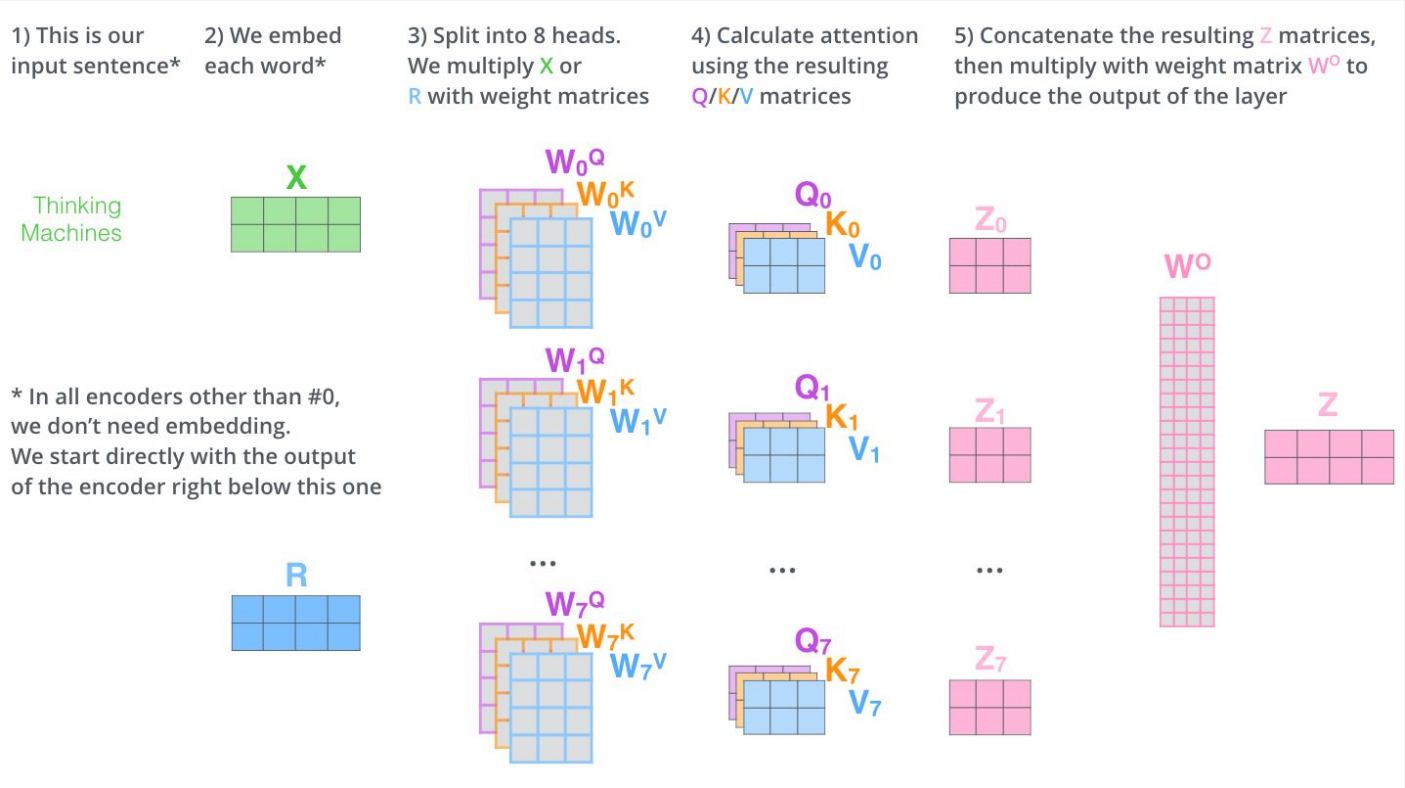

ENCODER
Feed Forward
z1
z2
z3
Self-Attention
x1
x2
x3
Je
suis
étudiant

Self Attention



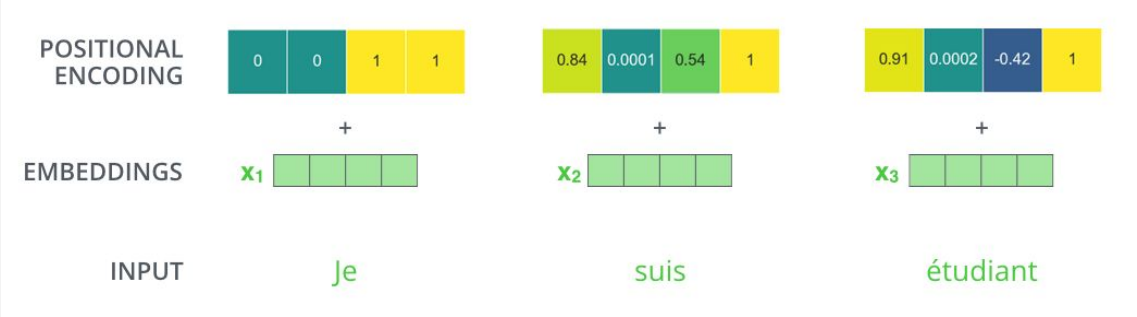

Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

Multi-head attention

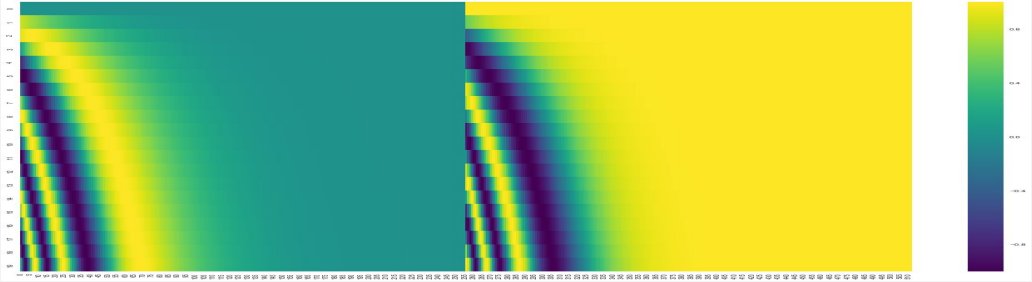


Adding back sequential order

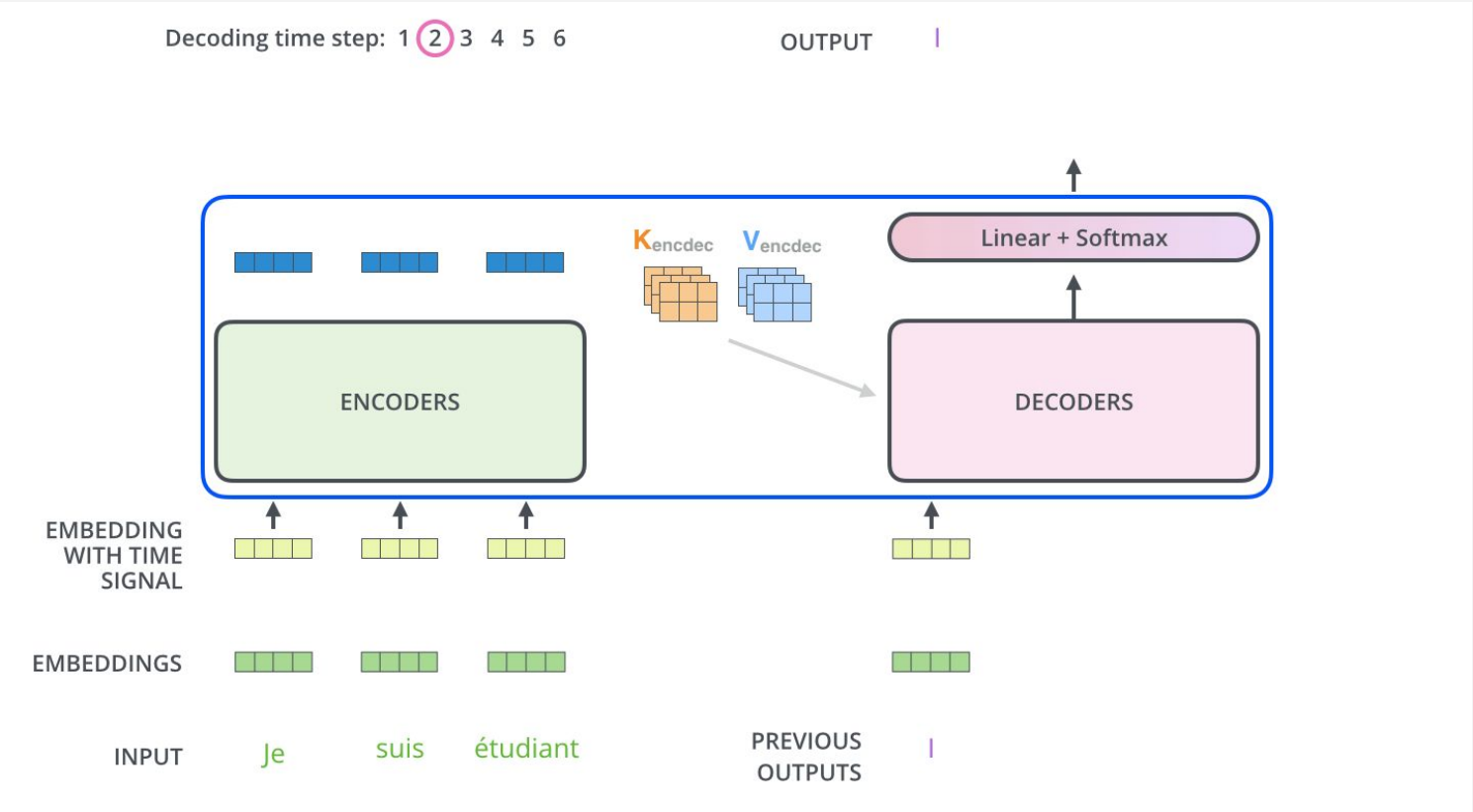
Positional encoding:



Adding position-dependent signal to word embeddings



The full picture



Summary - Sequential Data Modeling

- Overview
- Data Preprocessing
- Put LSTM To Work
- Attention Models