Keras

Keras for Deep Learning Research

Feedback is greatly appreciated!

Overview

- Difference between Feedforward Neural Network (FNN) and Recurrent Neural Network (RNN)
- The importance of context
- RNN
- LSTM
- Use case

Why Recurrent Neural Network

- Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words.
- You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

Why Recurrent Neural Network

- Traditional neural networks can't do this, and it seems like a major shortcoming.
- For example, imagine you want to classify what kind of event is happening at every point in a movie.
- It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.
- Recurrent neural networks address this issue.
- They are networks with loops in them, allowing information to persist.

The importance of context

- Recall the 5th digit of your phone number
- Sing your favorite song beginning at third sentence
- Recall 10th character of the alphabet

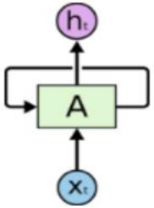
Probably you went straight from the beginning of the stream in each case…
because in sequences, order matters!

Idea: retain the information preserving the importance of order

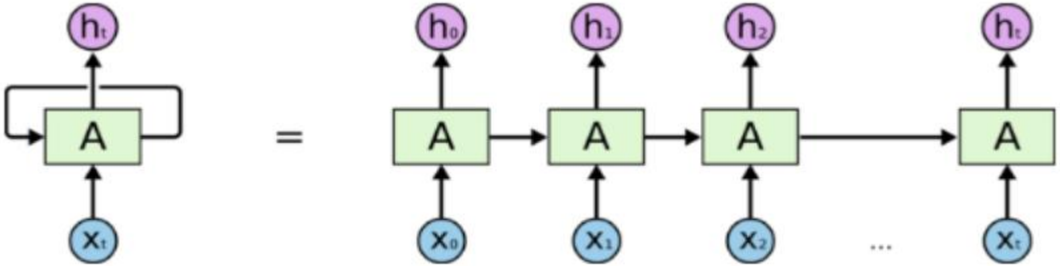
Recurrent Neural Network (RNN)

Recurrent Neural Network

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

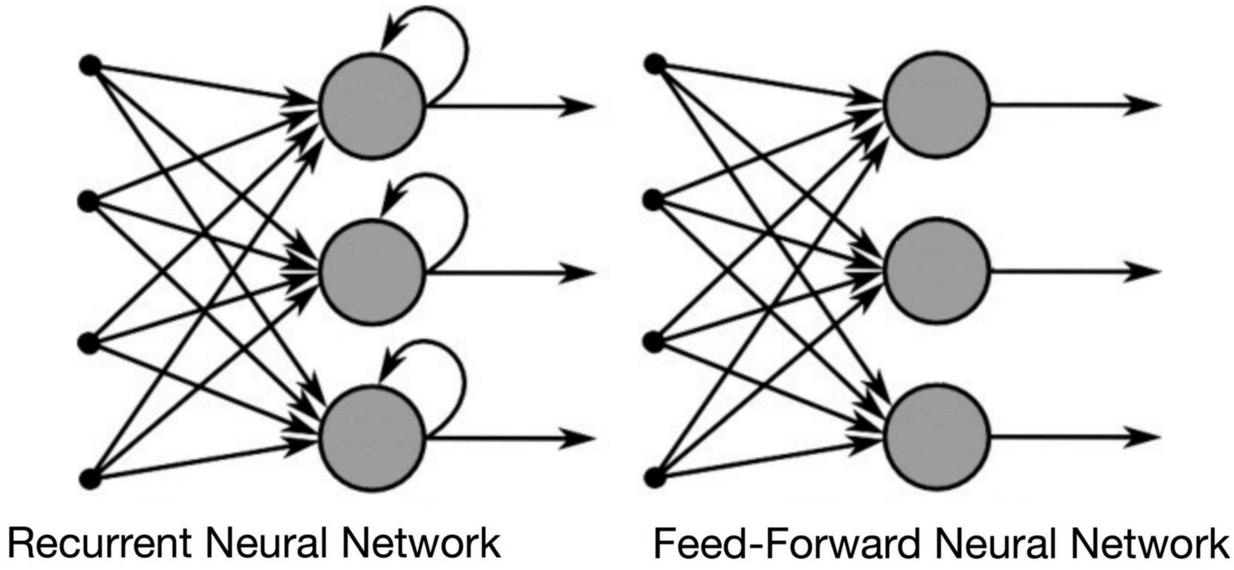


Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

Difference between RNN and feed forward



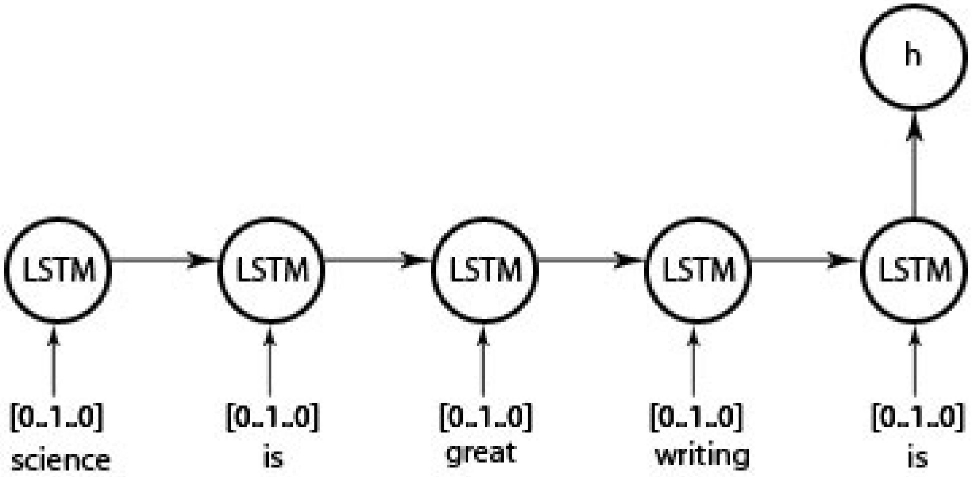
Source: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

Recurrent Neural Network (RNN)

- Good for Sequential data, or ordered data
- Internal memory: remember important things about input
- Produces output, copies and output, loop it back into input

UMKC

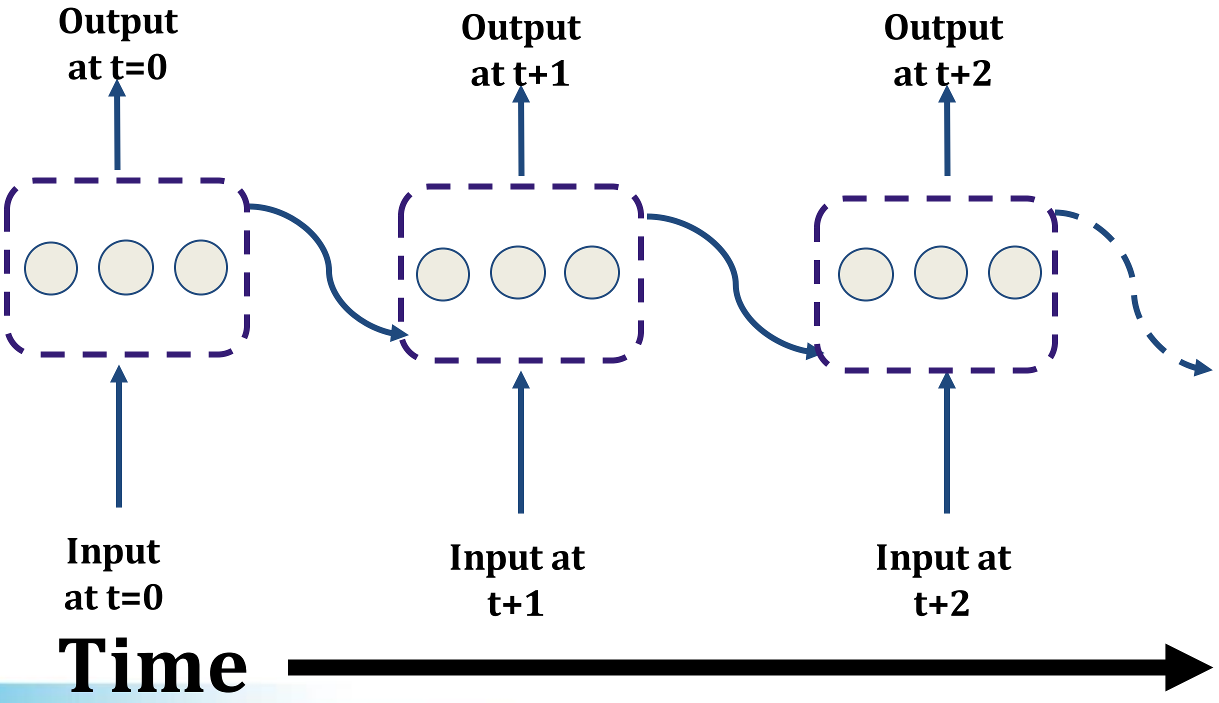
Example



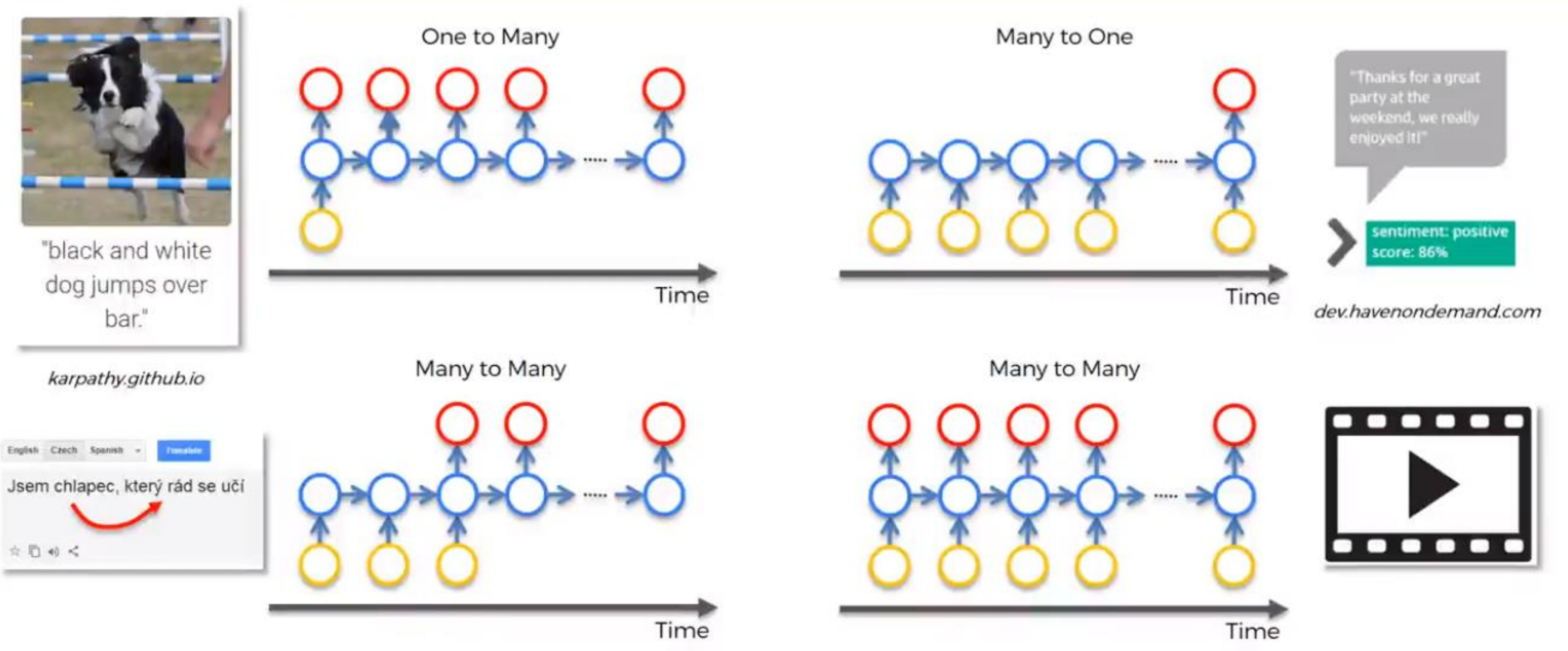
h
LSTM
LSTM
LSTM
LSTM
LSTM
[0..1..0]
science
[0..1..0]
is
[0..1..0]
great
[0..1..0]
writing
[0..1..0]
is

Recurrent Neural Network (RNN)

"Unrolled" layer

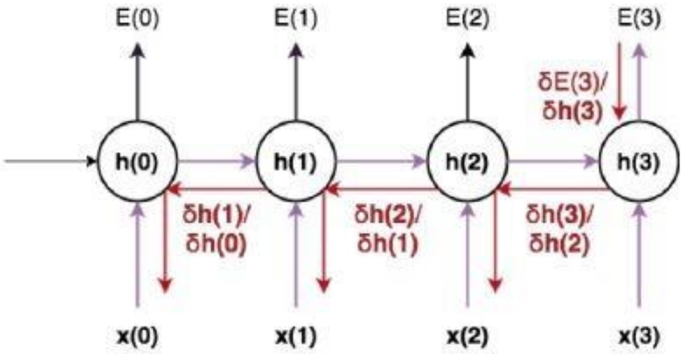


Different kind of RNN



Recurrent Neural Network (RNN)

Back Propagation Through Time (BPTT): The training method has to take into account the time operations → a cost function E is defined to train our RNN, and in this case the total error at the output of the network is the sum of the errors at each time-step



Example back-prop in time with 3 time-steps

Recurrent Neural Network (RNN)

Main problems:

- Sometimes, we only need to look at recent information to perform the present task - "the clouds are in the *sky*"
- But there are also cases where we need more context - "I grew up in France… I speak fluent *French*."

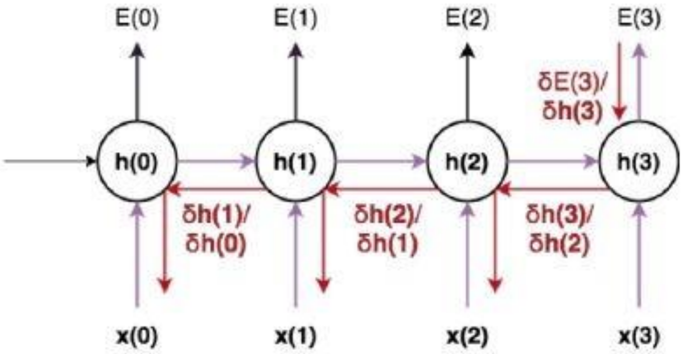
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

UMKC

Recurrent Neural Network (RNN)

Main problems:

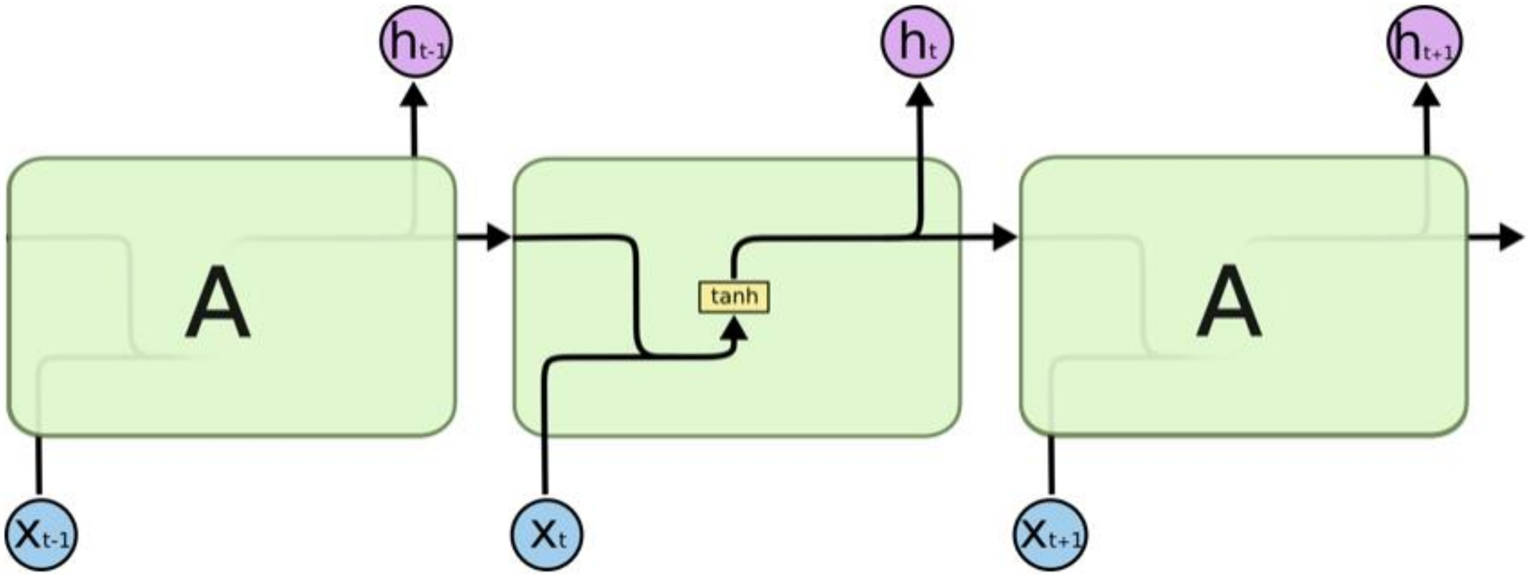
- **Exploding Gradients:** weights assigns high importance: unstable network.
- **Vanishing Gradients:** values of gradients are too small: model stops learning



Example back-prop in time with 3 time-steps

Long Short Term Memory (LSTM)

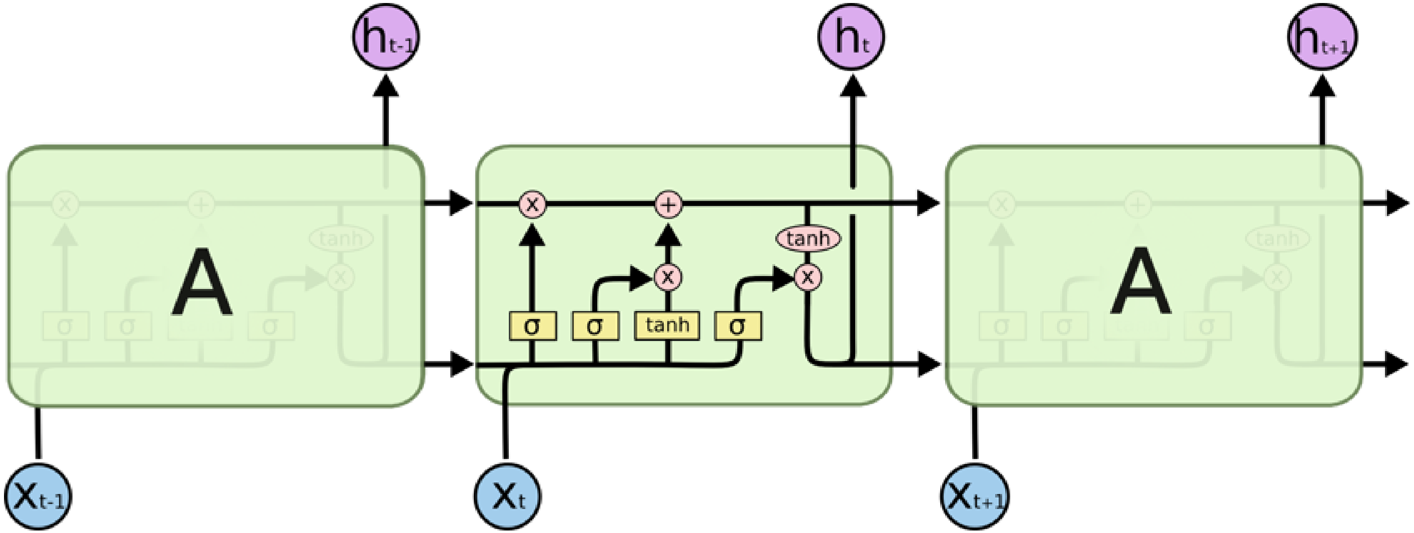
RNN vs LSTM



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

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

RNN vs LSTM

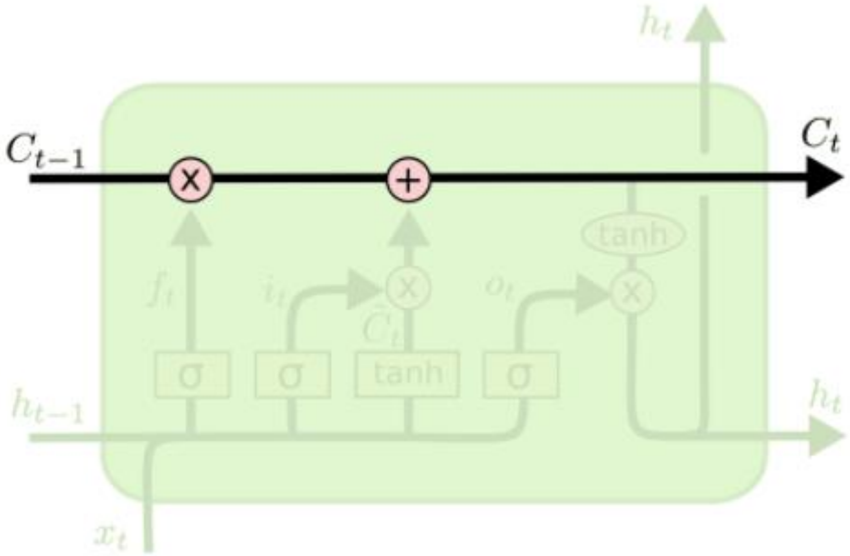


The repeating module in an LSTM contains four interacting layers.

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

Core idea behind LSTM

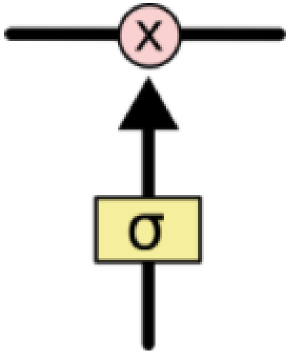
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.



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

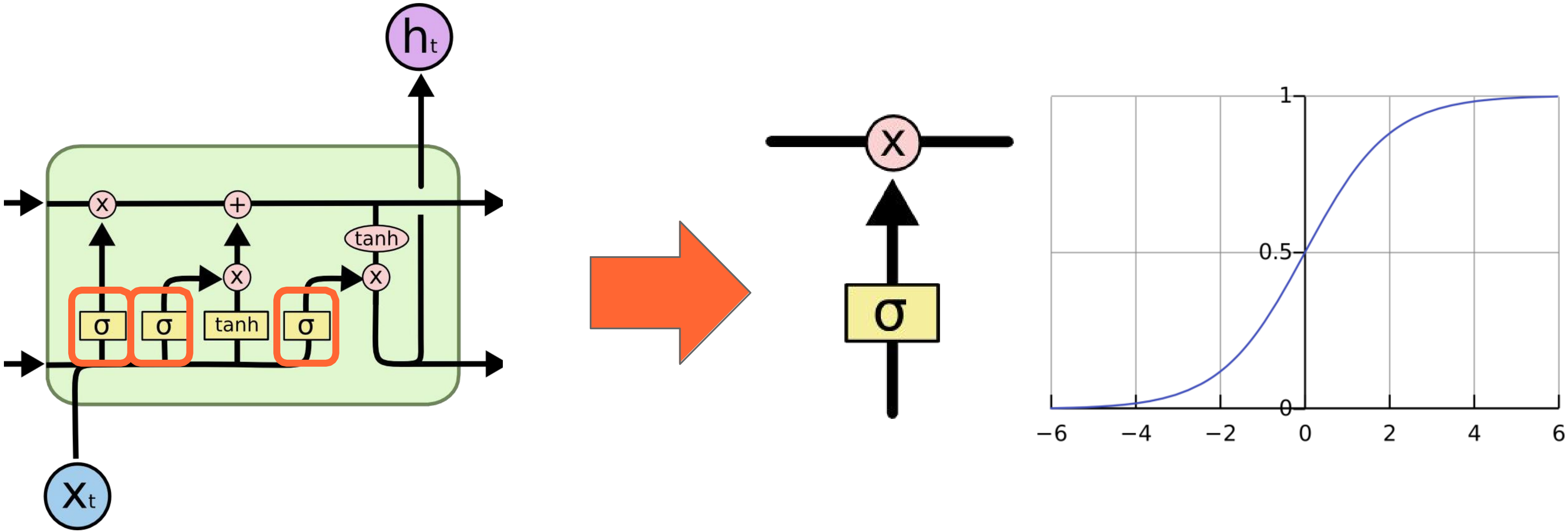
Gate

- Gates are a way to optionally let information through.
- They are composed out of a sigmoid neural net layer and a pointwise multiplication operation
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
- A value of zero means "let nothing through," while a value of one means "let everything through!"
- An LSTM has three of these gates, to protect and control the cell state.



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

Long Short Term Memory (LSTM)

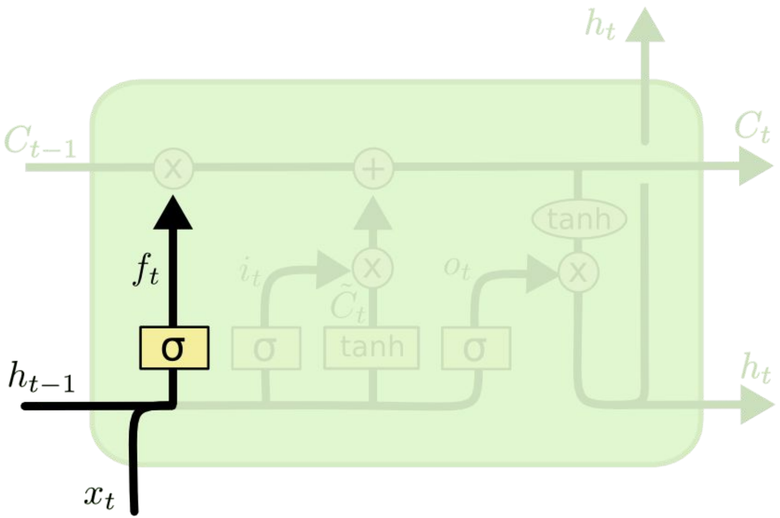


Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Meaning Of the Symbols

- + : Adding information
- σ : Sigmoid layer
- tanh: tanh layer
- h(t-1) : Output of last LSTM unit
- c(t-1) : Memory from last LSTM unit
- X(t) : Current input
- c(t) : New updated memory
- h(t) : Current output

Long Short Term Memory (LSTM)

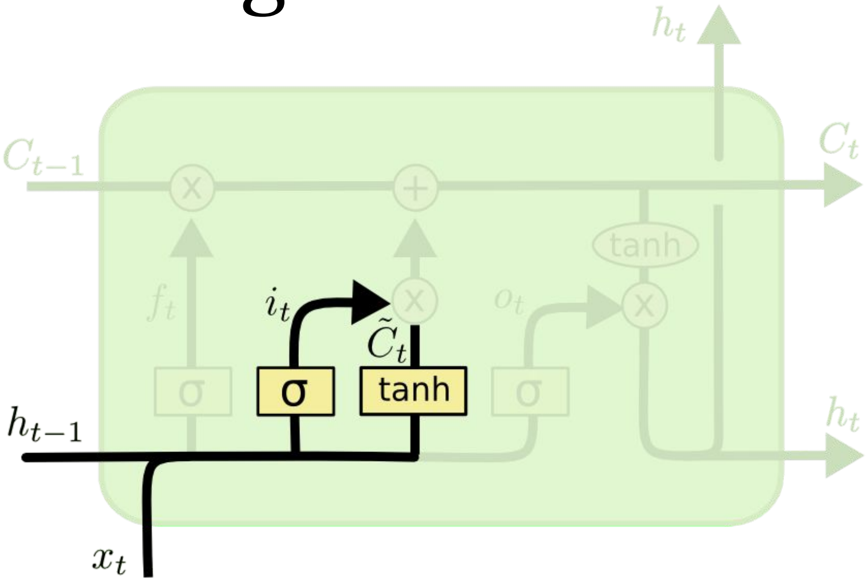


Forget Gate Layer

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

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

Long Short Term Memory (LSTM)



Input Gate Layer

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

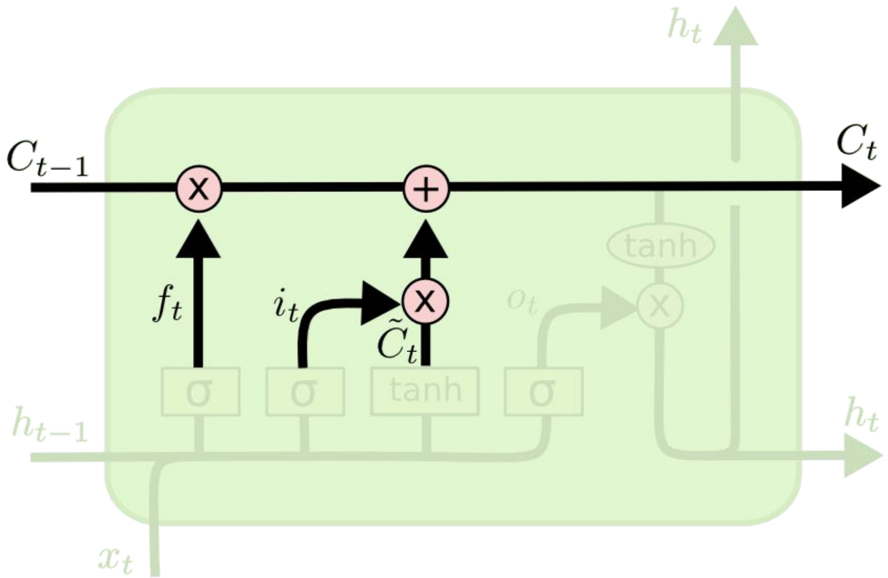
New contribution to cell state

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

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

UMKC

Long Short Term Memory (LSTM)

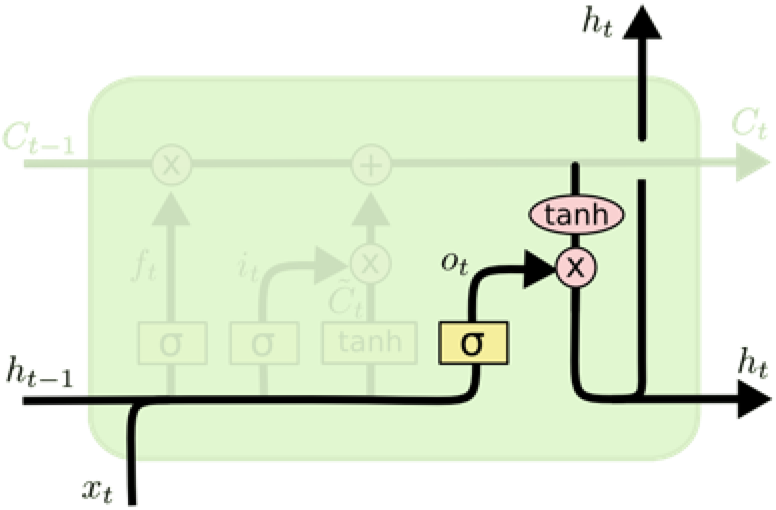


Update Cell State (memory)

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

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

Long Short Term Memory (LSTM)



Output Gate Layer

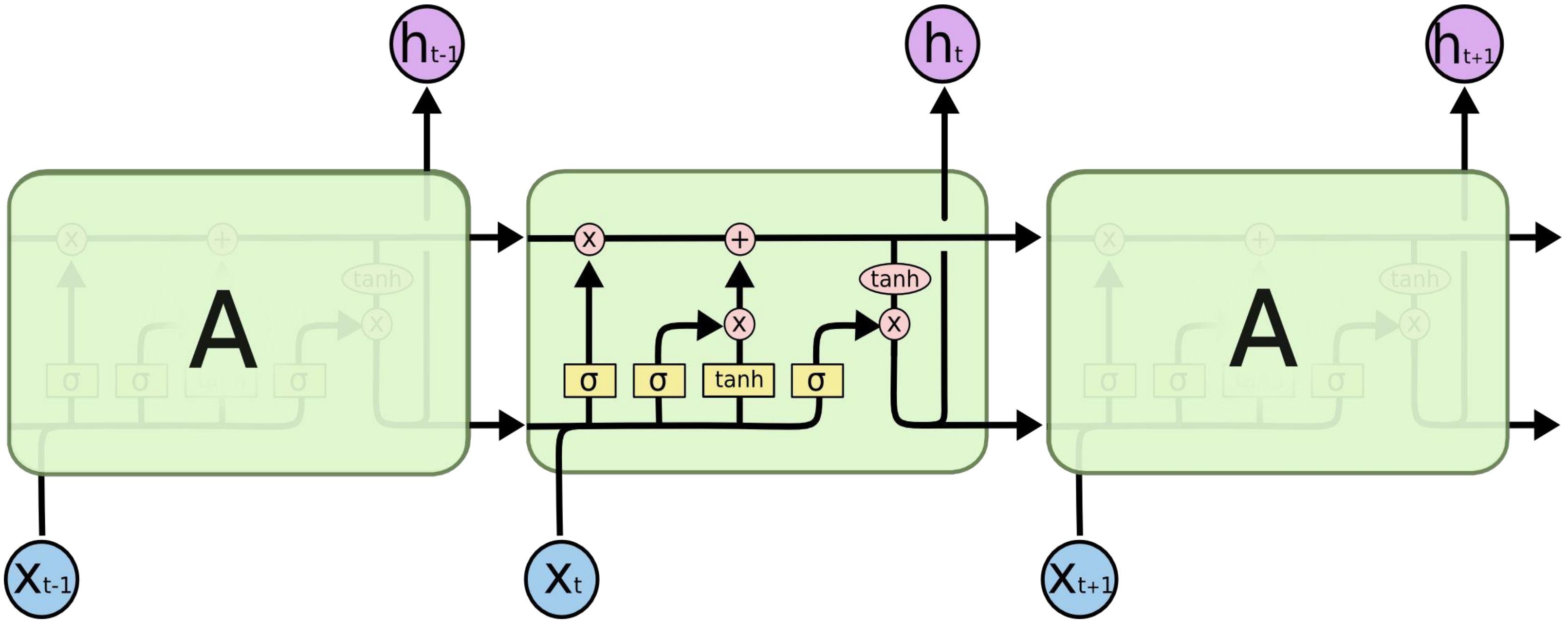
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

Output to next layer

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

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

Long Short Term Memory (LSTM)



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

Overfitting

- Overfitting occurs when you achieve a good fit of your model on the training data, while it does not generalize well on new, unseen data.
- In other words, the model learned patterns specific to the training data, which are irrelevant in other data

Different ways to avoid overfitting

- Get more data
- Reduce the network’s capacity
- Apply regularization
- Use Dropout layers

Regularization techniques

- Regularization: adding an extra element to the loss function, which punishes our model for being too complex or, for using too high values in the weight matrix
  - L1: Least Absolute Deviations (Lasso)
  - L2: Least Square Errors (Ridge)

L1: Least Absolute Deviations (Lasso)

- Lasso shrinks the less important feature's coefficient to zero
- Removing some feature altogether.
- This works well for **feature selection** in case we have a huge number of features.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

L2:Least Square Errors (Ridge)

- Adds "*squared magnitude*" of coefficient as penalty term to the loss function

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

Hyperparameter Optimization

- Grid Search
- Random Search
- Hand-tuning
- Gaussian Process with Expected Improvement
- Tree-structured Parzen Estimators (TPE)

Hyperparameter Optimization

- **Grid search** capability from the scikit-learn python machine learning library
- Grid search is a model hyperparameter optimization technique
- It tune the hyperparameters of deep learning models

How to Use Keras Models in scikit-learn

- Keras models can be used in scikit-learn by wrapping them with the **KerasClassifier** or **KerasRegressor** class.

How to Tune Batch Size and Number of Epochs

```
model = KerasClassifier(build_fn=model,verbose=0)
batch_size = [10, 20, 40]
epochs = [1, 2, 3]
param_grid = dict(batch_size=batch_size, epochs=epochs)
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

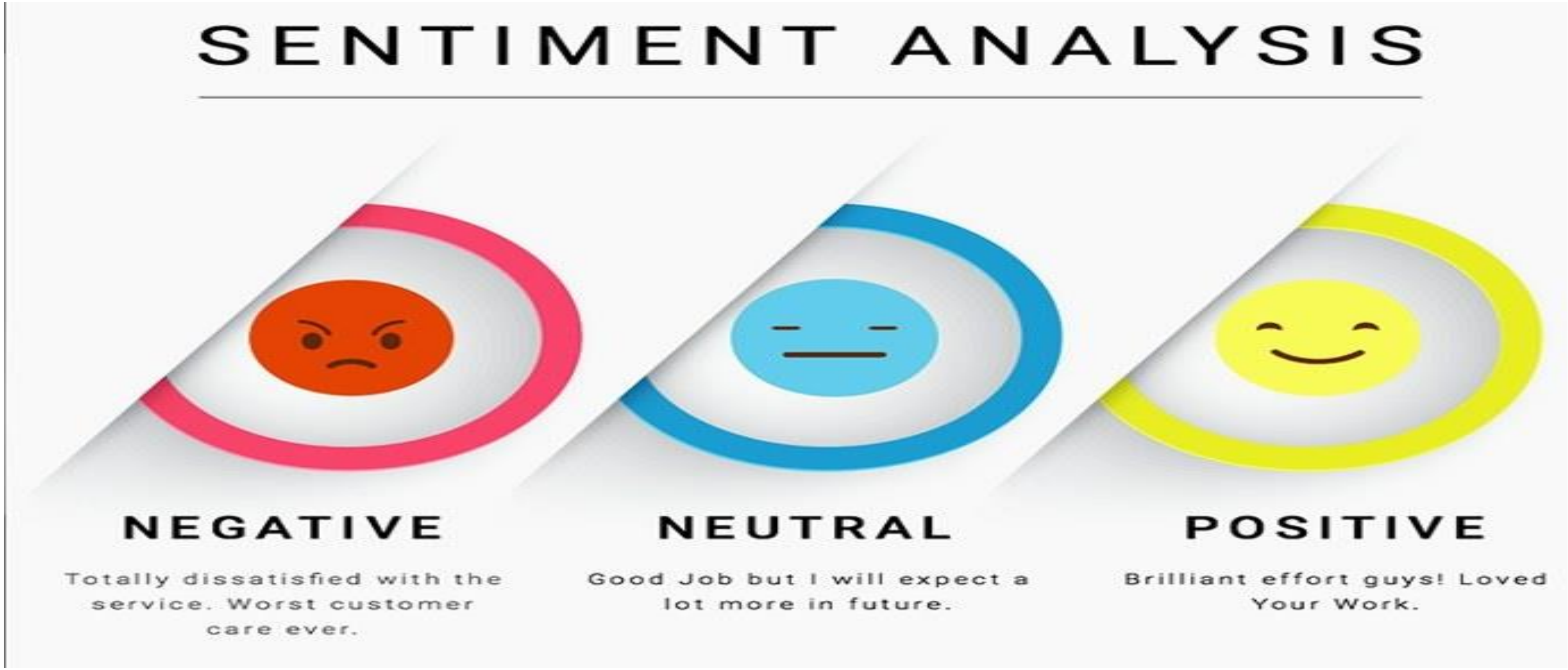
Results

Best: 0.846411 using {'batch_size’: 40, 'epochs’: 3}

List of hyperparameters to be tuned

- **Optimization Algorithm**
  - optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam’]
- **Network Weight Initialization**
  - init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform’]
- **Neuron Activation Function**
  - activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear’]
- **Dropout Regularization**
  - dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

Use case: Sentiment Analysis using LSTM



Sentiment Classification

- The process of computationally identifying and categorizing opinions expressed in a piece of text
- In order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

First GOP Debate Twitter Sentiment

- Analyze tweets on the first 2016 GOP Presidential Debate

keeping the necessary columns

- Only keeping the necessary columns.

```
data = pd.read_csv('../input/Sentiment.csv’)
# Keeping only the neccessary columns
data = data[['text','sentiment']]
```

Filtering the tweets, using Tokenizer to vectorize, convert text into Sequences

```
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))

for idx, row in data.iterrows():
            row[0] = row[0].replace('rt',' ‘)

max_features = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ‘)
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

composing the LSTM Network

```
embed_dim = 128
 lstm_out = 196
model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3,activation='softmax’))
model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
(model.summary())
```

Defining training and test data

```
labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment’])
Y = to_categorical(integer_encoded)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.33, random_state = 42)
(X_train.shape,Y_train.shape)
(X_test.shape,Y_test.shape)
```

```
batch_size = 32
model.fit(X_train, Y_train, epochs = 7, batch_size=batch_size, verbose = 2)
```

Evaluating the model

```
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = batch_size)
("score: %.2f" % (score))
("acc: %.2f" % (acc))
```

References

- https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
- https://stackoverflow.com/questions/47788799/grid-search-the-number-of-hidden-layers-with-keras
- https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767
- https://towardsdatascience.com/exploring-activation-functions-for-neural-networks-73498da59b02
- https://medium.com/lingvo-masino/introduction-to-recurrent-neural-network-d77a3fe2c56c
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://www.slideshare.net/ashraybhandare/deep-learning-cnn-and-rnn?qid=0f39b4da-a290-4e9e-9d31-ed21816598e3&v=&b=&from_search=27
- https://machinelearningmastery.com/truncated-backpropagation-through-time-in-keras/
- https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912
- https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

Cheat Code

- Code for saving the model: model.save(‘model.h5’)
- Code for loading the model:
  - from keras.models import load_model
  - model = load_model(‘model.h5’)
- Code for predicting on new data:
  - model.predict([‘’new text’]