Deep Learning Adventures TensorFlow In Practice - Presentation 4

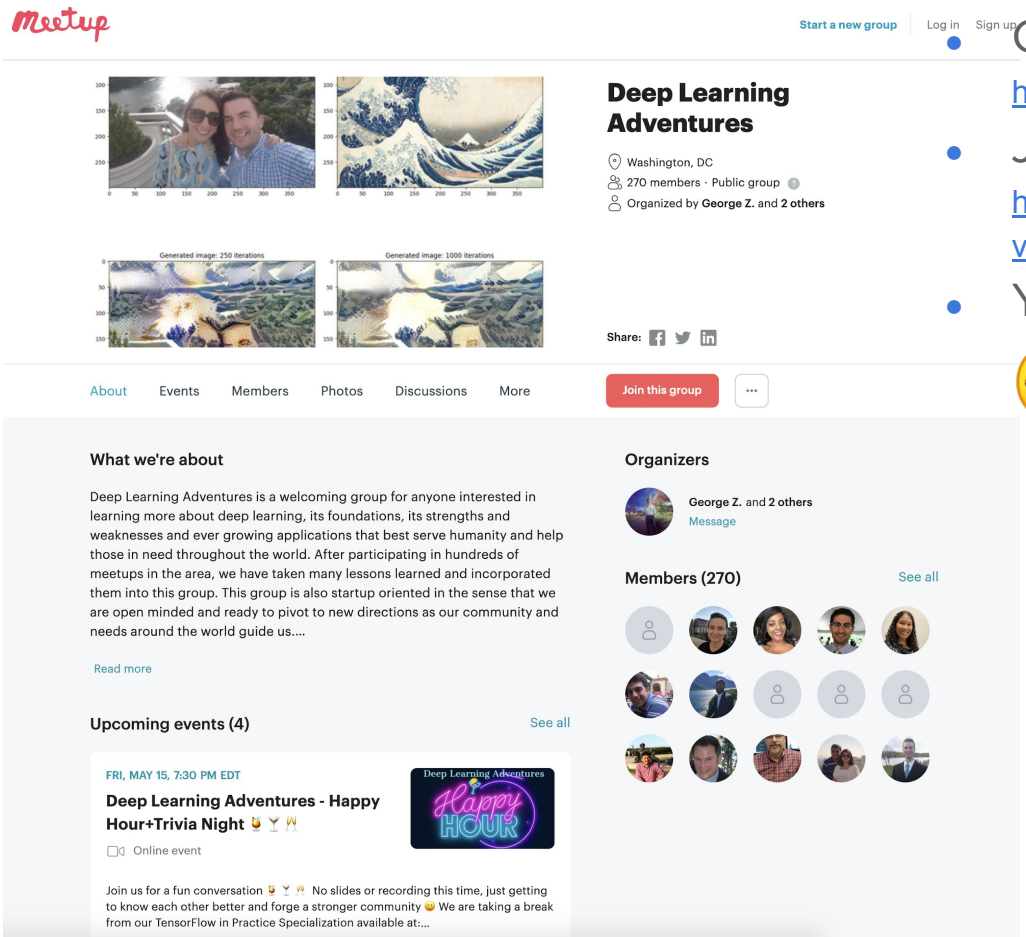
A quick overview of Coursera's Tensorflow in Practice specialization course

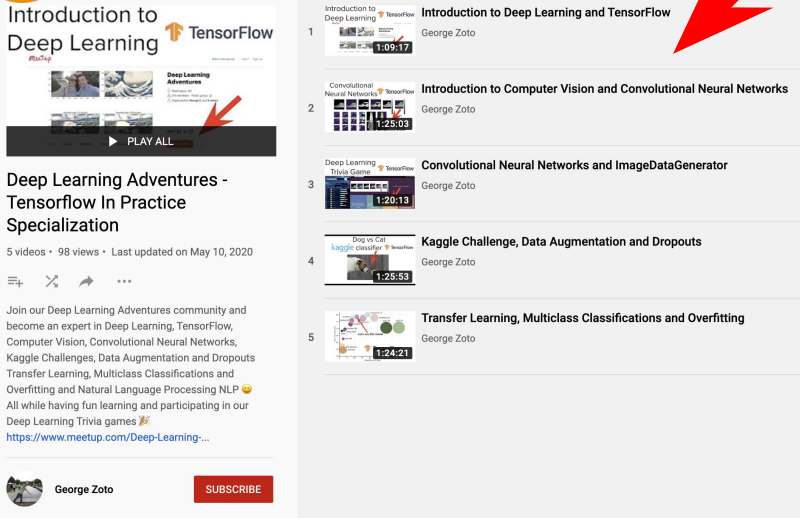
Robert Kraig, David Patton, George Zoto
https://www.meetup.com/Deep-Learning-Adventures

In the beginning...

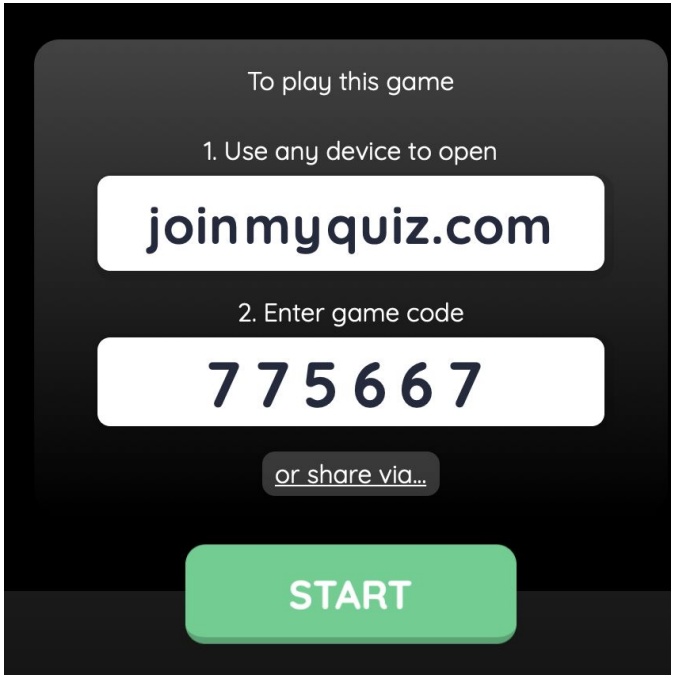
- Meetup Link
  https://www.meetup.com/Deep-Learning-Adventures
- GitHub repository
  https://github.com/georgezoto/Deep-Learning-Adventures
- Join us on Slack
  https://join.slack.com/t/deeplearninga-nmk8930/shared_invite/zt-d52h9mm9-h~Q0ZXw5PXsTDzPIlNIvog
- YouTube recordings of all Meetups

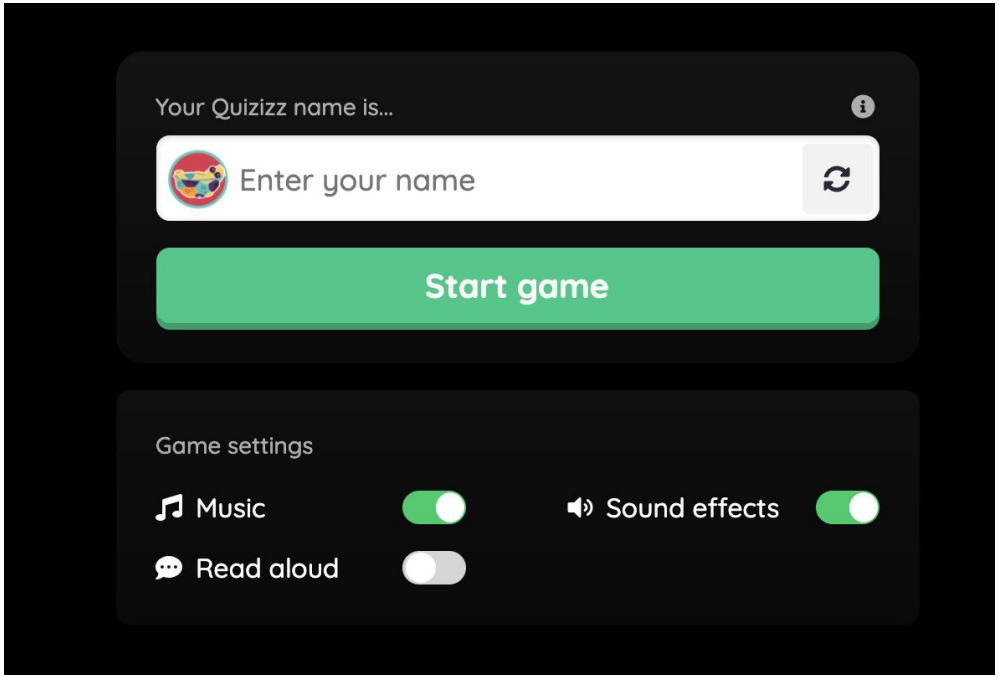
😀 https://bit.ly/deep-learning-tf



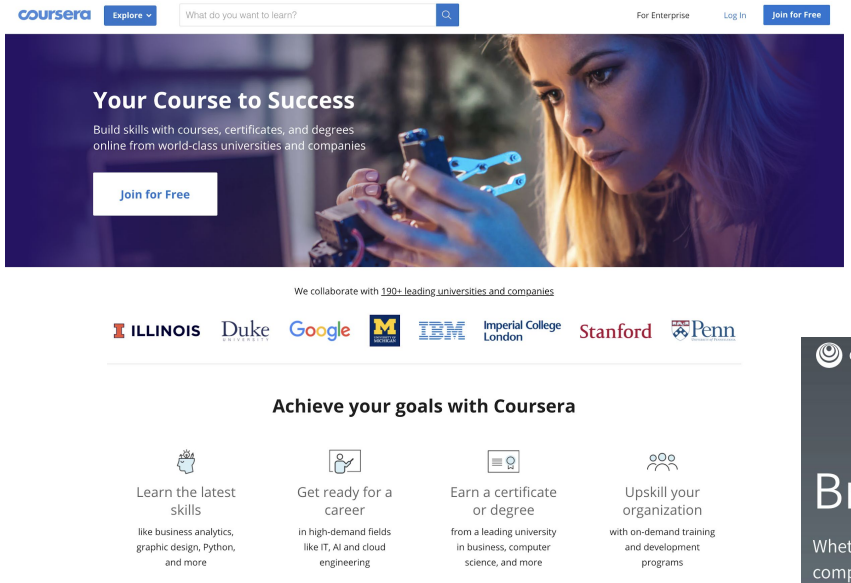


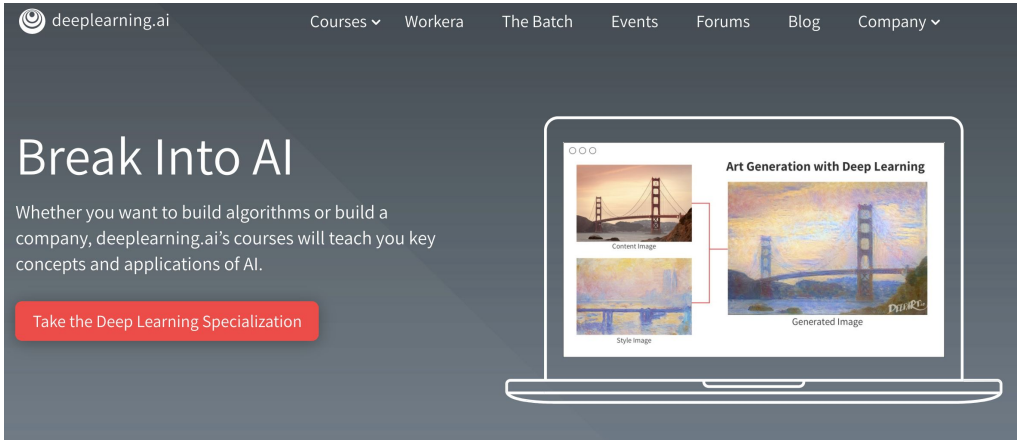
Not a typical Meetup... Get ready for a fun game on 6/5 😀🎉





Attribution to Coursera and deeplearning.ai





Source:
https://www.coursera.org/about/terms
https://www.coursera.org/
https://www.deeplearning.ai/

Chapter 1 - TensorFlow in Practice Specialization

About this Specialization

199,621 recent views

Discover the tools software developers use to build scalable AI-powered algorithms in TensorFlow, a popular open-source machine learning framework.

In this four-course Specialization, you'll explore exciting opportunities for AI applications. Begin by developing an understanding of how to build and train neural networks. Improve a network's performance using convolutions as you train it to identify real-world images. You'll teach machines to understand, analyze, and respond to human speech with natural language processing systems. Learn to process text, represent sentences as vectors, and input data to a neural network. You'll even train an AI to create original poetry!

AI is already transforming industries across the world. After finishing this Specialization, you'll be able to apply your new TensorFlow skills to a wide range of problems and projects.

Looking for more advanced TensorFlow content? Check out the new **TensorFlow: Data and Deployment Specialization**.

Source: https://www.coursera.org/specializations/tensorflow-in-practice

Chapter 1 - TensorFlow in Practice Specialization

There are 4 Courses in this Specialization

COURSE 1

Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning

★★★★½ 4.7 6,196 ratings • 1,282 reviews

If you are a software developer who wants to build scalable AI-powered algorithms, you need to understand how to use the tools to build them. This course is part of the upcoming Machine Learning in Tensorflow Specialization and will teach you best practices for using TensorFlow, a popular open-source framework for machine learning.

SHOW ALL

COURSE 2

Convolutional Neural Networks in TensorFlow

★★★★½ 4.7 2,751 ratings • 410 reviews

If you are a software developer who wants to build scalable AI-powered algorithms, you need to understand how to use the tools to build them. This course is part of the upcoming Machine Learning in Tensorflow Specialization and will teach you best practices for using TensorFlow, a popular open-source framework for machine learning.

SHOW ALL

COURSE 3

Natural Language Processing in TensorFlow

★★★★½ 4.6 2,037 ratings • 277 reviews

If you are a software developer who wants to build scalable AI-powered algorithms, you need to understand how to use the tools to build them. This Specialization will teach you best practices for using TensorFlow, a popular open-source framework for machine learning.

SHOW ALL

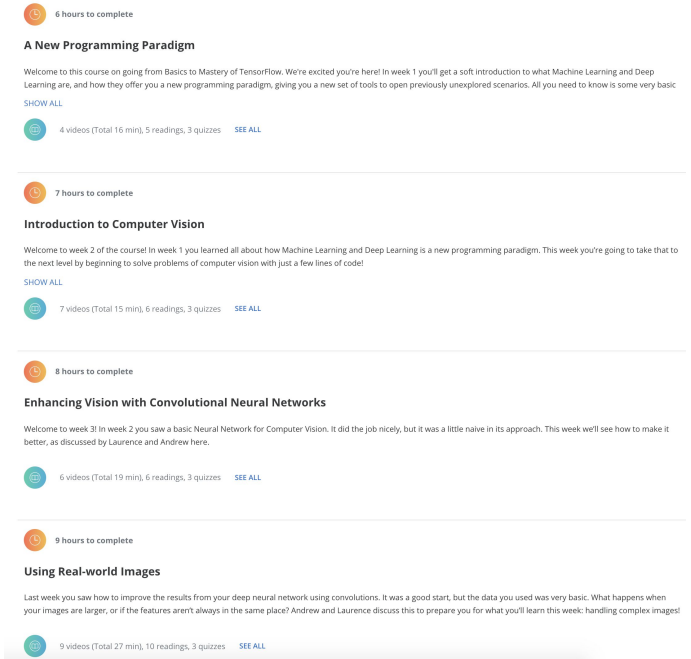
COURSE 4

Sequences, Time Series and Prediction

★★★★½ 4.6 1,374 ratings • 223 reviews

If you are a software developer who wants to build scalable AI-powered algorithms, you need to understand how to use the tools to build them. This Specialization will teach you best practices for using TensorFlow, a popular open-source framework for machine learning.

SHOW ALL



Source: https://www.coursera.org/specializations/tensorflow-in-practice

Setup

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. You can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

https://colab.research.google.com

Course 3: Natural Language Processing in TensorFlow

Week 1: Sentiment in text
Week 2: Word Embeddings
Week 3: Sequence models
Week 4: Sequence models and literature

Sentiment can be tricky. (That DJ killed me last night vs That DJ killed it last night)

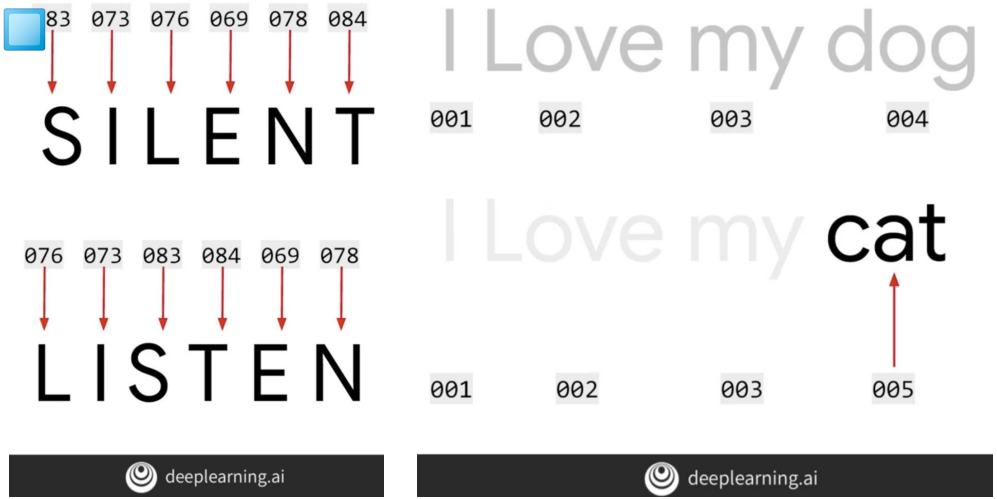
- **Problem: Models need numbers for training**
- **Solution: Tokenization**

Convert text (input and output) into numbers via a Tokenizer

1. **Characters**
2. **Words**
3. **Subwords**

Sentiment in text

1



2

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
```

3

```
sentences = [
    'I love my dog',
    'I love my cat'
]
```

4

```
tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

5

```
{'i': 1, 'my': 3, 'dog': 4, 'cat': 5, 'love': 2}
```

6

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!'
]
```

7

```
{'i': 3, 'my': 2, 'you': 6, 'love': 1, 'cat': 5, 'dog': 4}
```

Source: https://www.coursera.org/learn/natural-language-processing-tensorflow

Sentiment in text

```
from tensorflow.keras.preprocessing.text import Tokenizer

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]

tokenizer = Tokenizer(num_words = 100)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index

sequences = tokenizer.texts_to_sequences(sentences)

print(word_index)
print(sequences)
```

```
{'amazing': 10, 'dog': 3, 'you': 5, 'cat': 6,
 'think': 8, 'i': 4, 'is': 9, 'my': 1, 'do': 7,
 'love': 2}
[[4, 2, 1, 3], [4, 2, 1, 6], [5, 2, 1, 3], [7, 5,
8, 1, 3, 9, 10]]
```

Source: https://www.coursera.org/learn/natural-language-processing-tensorflow

Sentiment in text

```
test_data = [
    'i really love my dog',
    'my dog loves my manatee'
]

test_seq = tokenizer.texts_to_sequences(test_data)
print(test_seq)
```

```
[[4, 2, 1, 3], [1, 3, 1]]
```

```
{'think': 8, 'amazing': 10, 'my': 1, 'love': 2, 'dog': 3, 'is': 9, 'you': 5, 'do': 7,
'cat': 6, 'i': 4}
```

deeplearning.ai

```
tokenizer = Tokenizer(num_words = 100, oov_token="<OOV>")
```

```
{'think': 9, 'amazing': 11, 'dog': 4, 'do': 8, 'i': 5, 'cat': 7,
 'you': 6, 'love': 3, '<OOV>': 1, 'my': 2, 'is': 10}
```

```
[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]
```

Source: https://www.coursera.org/learn/natural-language-processing-tensorflow

Sentiment in text

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
sequences = tokenizer.texts_to_sequences(sentences)

padded = pad_sequences(sequences)
```

```
sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'Do you think my dog is amazing?'
]
```

```
{'do': 8, 'you': 6, 'love': 3, 'i': 5, 'amazing': 11, 'my': 2, 'is': 10, 'think': 9,
'dog': 4, '<OOV>': 1, 'cat': 7}


[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]


[[ 0  0  0  5  3  2  4]
 [ 0  0  0  5  3  2  7]
 [ 0  0  0  6  3  2  4]
 [ 8  6  9  2  4 10 11]]
```

```
padded = pad_sequences(sequences, padding='post',
                       truncating='post', maxlen=5)
```

Source: https://www.coursera.org/learn/natural-language-processing-tensorflow

Sentiment in text

Each record consists of three attributes:

- `is_sarcastic` : 1 if the record is sarcastic otherwise 0
- `headline` : the headline of the news article
- `article_link` : link to the original news article. Useful in collecting supplementary data

Source: https://www.kaggle.com/rmisra/news-headlines-dataset-for-sarcasm-detection/home

Course 3: Natural Language Processing in TensorFlow

Week 1: Sentiment in text
Week 2: Word Embeddings
Week 3: Sequence models
Week 4: Sequence models and literature

Word Embeddings

```
audio

"nsynth"

image

"abstract_reasoning"
"caltech101"
"cats_vs_dogs"
"celeb_a"
"celeb_a_hq"
"cifar10"
"cifar100"
"cifar10_corrupted"
"coco2014"
"colorectal_histology"
"cycle_gan"
"diabetic_retinopathy..."
"dsprites"
"dtd"
"emnist"
"fashion_mnist"
"horses_or_humans"
"image_label_folder"
"imagenet2012"
"imagenet2012_corrupted"
"kmnist"
"lsun"
"mnist"
"omniglot"
"open_images_v4"
"oxford_iiit_pet"
"quickdraw_bitmap"
"rock_paper_scissors"
"shapes3d"
"smallnorb"
"sun397"
"svhn_cropped"
"tf_flowers"

structured

"higgs"
"iris"
"titanic"

text

"cnn_dailymail"
"glue"
"imdb_reviews"
"lm1b"
"multi_nli"
"squad"
"wikipedia"
"xnli"

translate

"flores"
"para_crawl"
"ted_hrlr_translate"
"ted_multi_translate"
"wmt15_translate"
"wmt16_translate"
"wmt17_translate"
"wmt18_translate"
"wmt19_translate"
```

Source:
https://www.tensorflow.org/datasets/catalog/overview
https://github.com/tensorflow/datasets
https://www.tensorflow.org/datasets/catalog/imdb_reviews

TensorFlow Datasets

- Text
  - blimp
  - c4 (manual)
  - cfq
  - civil_comments
  - cos_e
  - definite_pronoun_resolution
  - eraser_multi_rc
  - esnli
  - gap
  - glue
  - imdb_reviews
  - librispeech_lm
  - lm1b
  - math_dataset
  - movie_rationales
  - multi_nli
  - multi_nli_mismatch
  - natural_questions
  - qa4mre
  - scan
  - scicite
  - snli
  - squad
  - super_glue
  - tiny_shakespeare
  - trivia_qa
  - web_questions
  - wiki40b
  - wikipedia
  - xnli
  - yelp_polarity_reviews

Word Embeddings

IMDB Reviews: Large Movie Review Dataset

This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well. Raw text and already processed bag of words formats are provided. See the README file contained in the release for more details.

Large Movie Review Dataset v1.0

When using this dataset, please cite our ACL 2011 paper [bib].

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. *2011)*.

```
In [1]: import sys
        print(sys.version)
        import tensorflow as tf
        print(tf.__version__)

        # !pip install -q tensorflow-datasets
```

```
3.7.7 (default, Mar 23 2020, 23:19:08)
2.1.0
```

```
In [0]: # !pip install -q tensorflow-datasets
        import tensorflow_datasets as tfds
        imdb, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)
```

```
training_sentences[:10]
```

```
['b"This was an absolutely terrible movie. Don\'t be lured in by Christopher Walken or Michael Ironside.
 "b'I have been known to fall asleep during films, but this is usually due to a combination of things in
 "b'Mann photographs the Alberta Rocky Mountains in a superb fashion, and Jimmy Stewart and Walter Brenn
 "b'This is the kind of film for a snowy Sunday afternoon when the rest of the world can go ahead with i
 'b\'As others have mentioned, all the women that go nude in this film are mostly absolutely gorgeous. T
 'b"This is a film which should be seen by anybody interested in, effected by, or suffering from an eati
 'b\'Okay, you have:<br /><br />Penelope Keith as Miss Herringbone-Tweed, B.B.E. (Backbone of England.)
 'b\'The film is based on a genuine 1950s novel.<br /><br />Journalist Colin McInnes wrote a set of thre
 'b\'I really love the sexy action and sci-fi films of the sixties and its because of the actress\\\'s t
 'b\'Sure, this one isn\\\'t really a blockbuster, nor does it target such a position. "Dieter" is the f
```

```
training_labels_final[:10]
```

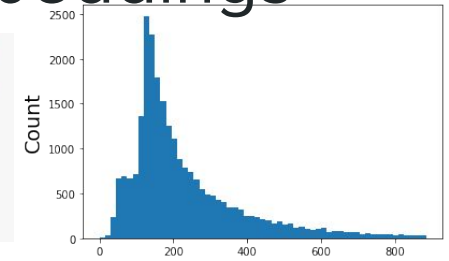
```
array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0])
```

Source: http://ai.stanford.edu/~amaas/data/sentiment/

Word Embeddings

1

```
vocab_size = 10000
embedding_dim = 16
max_length = 120
trunc_type='post'
oov_tok = "<OOV>"
```



2

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words = vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded = pad_sequences(sequences,maxlen=max_length, truncating=trunc_type)

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences,maxlen=max_length)
```

3

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 11526
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
_________________________________________________________________
```

4

```
num_epochs = 10
model.fit(padded, training_labels_final, epochs=num_epochs,
          validation_data=(testing_padded, testing_labels_final))
```

5

```
Epoch 1/10
782/782 [==============================] - 5s 7ms/step - loss: 0.4831 - accuracy: 0.7515 - val_loss: 0.3459 - val_accuracy: 0.8464
Epoch 2/10
782/782 [==============================] - 5s 6ms/step - loss: 0.2349 - accuracy: 0.9095 - val_loss: 0.3742 - val_accuracy: 0.8369
Epoch 3/10
782/782 [==============================] - 5s 7ms/step - loss: 0.0888 - accuracy: 0.9771 - val_loss: 0.4547 - val_accuracy: 0.8264
Epoch 4/10
782/782 [==============================] - 5s 7ms/step - loss: 0.0212 - accuracy: 0.9972 - val_loss: 0.5409 - val_accuracy: 0.8222
Epoch 5/10
782/782 [==============================] - 5s 6ms/step - loss: 0.0051 - accuracy: 0.9998 - val_loss: 0.5965 - val_accuracy: 0.8241
Epoch 6/10
782/782 [==============================] - 5s 6ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.6408 - val_accuracy: 0.8271
Epoch 7/10
782/782 [==============================] - 5s 6ms/step - loss: 8.2845e-04 - accuracy: 1.0000 - val_loss: 0.6797 - val_accuracy: 0.8280
Epoch 8/10
782/782 [==============================] - 5s 6ms/step - loss: 4.3873e-04 - accuracy: 1.0000 - val_loss: 0.7181 - val_accuracy: 0.8284
Epoch 9/10
782/782 [==============================] - 5s 6ms/step - loss: 2.5901e-04 - accuracy: 1.0000 - val_loss: 0.7565 - val_accuracy: 0.8286
Epoch 10/10
782/782 [==============================] - 5s 6ms/step - loss: 1.5668e-04 - accuracy: 1.0000 - val_loss: 0.7907 - val_accuracy: 0.8286
<tensorflow.python.keras.callbacks.History at 0x7f4f93c85c18>
```

6

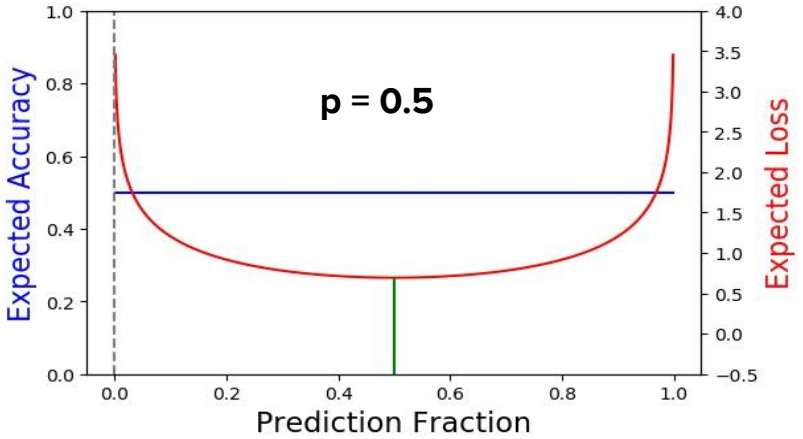
```
e = model.layers[0]
weights = e.get_weights()[0]
print(weights.shape) # shape: (vocab_size, embedding_dim)
```

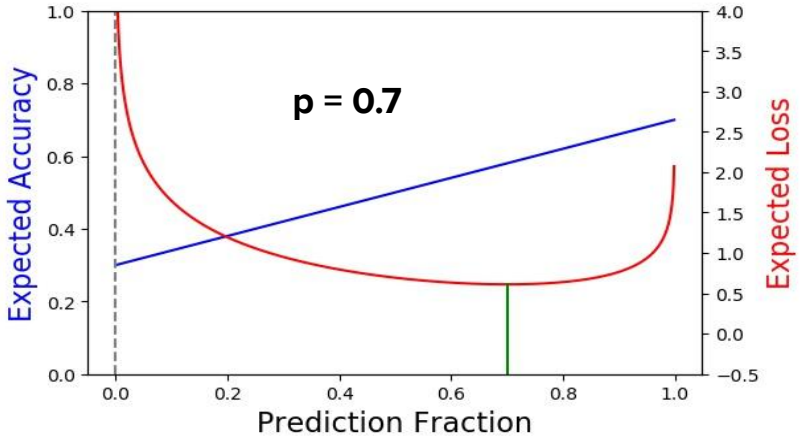
```
(10000, 16)
```

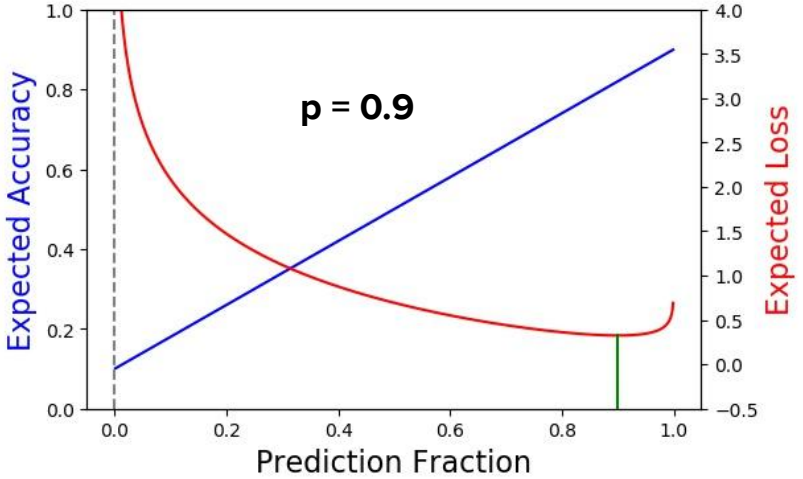
7

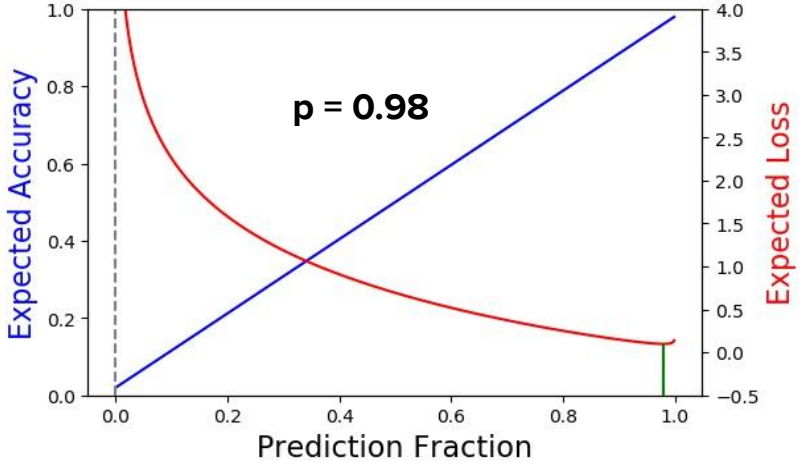
```
files.download('vecs.tsv')
files.download('meta.tsv')
```

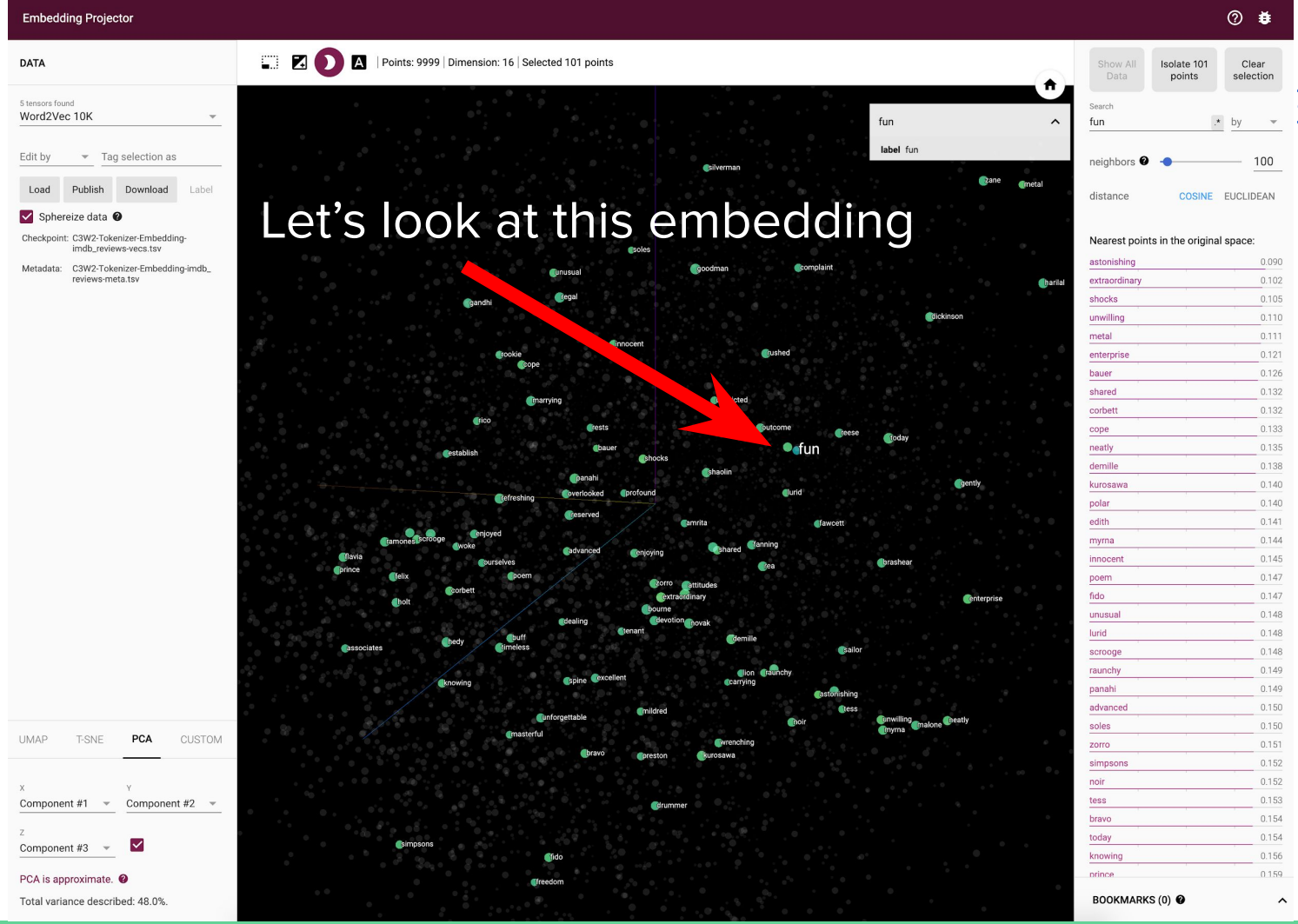
Word Embeddings: Revisit Loss vs. Accuracy



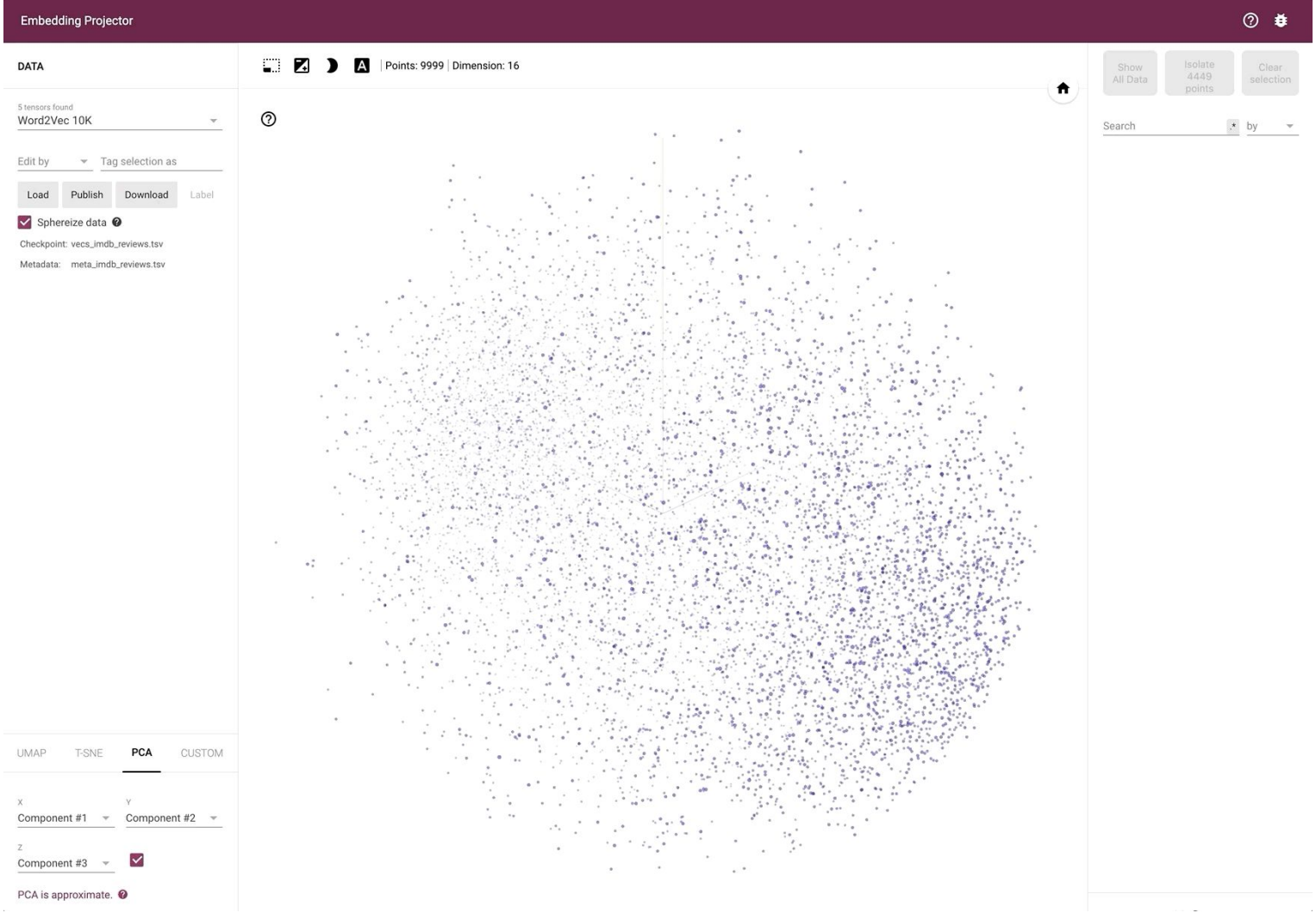








Source: http://projector.tensorflow.org/



Source: http://projector.tensorflow.org/

Word Embeddings

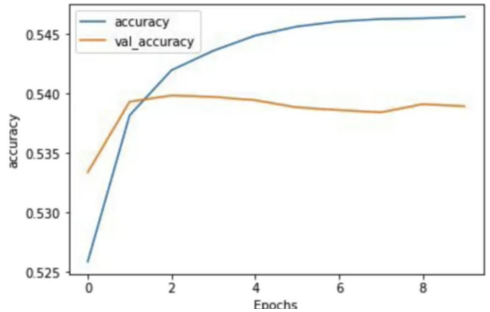
`imdb_reviews` is configured with `tfds.text.imdb.IMDBReviewsConfig` and has the following configurations predefined (defaults to the first one):

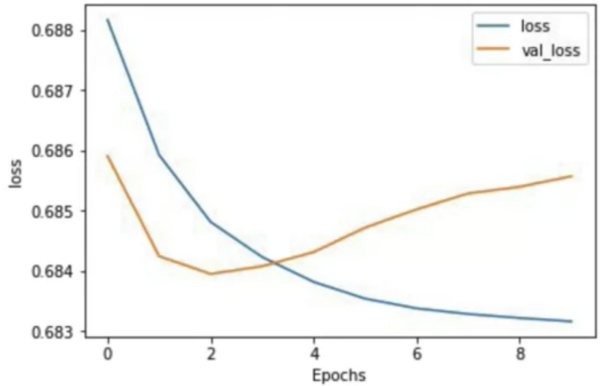
- `"plain_text"` (`v0.0.1`) (`Size: 80.23 MiB`): Plain text
- `"bytes"` (`v0.0.1`) (`Size: 80.23 MiB`): Uses byte-level text encoding with `tfds.features.text.ByteTextEncoder`
- `"subwords8k"` (`v0.0.1`) (`Size: 80.23 MiB`): Uses `tfds.features.text.SubwordTextEncoder` with 8k vocab size
- `"subwords32k"` (`v0.0.1`) (`Size: 80.23 MiB`): Uses `tfds.features.text.SubwordTextEncoder` with 32k vocab size

```
print(tokenizer.subwords)

['the_', ', ', '. ', 'a_', 'and_', 'of_', 'to_', 's_', 'is_',
'br', 'in_', 'I_', 'that_', 'this_', 'it_', … ]
```

```
6307 ----> Ten
2327 ----> sor
4043 ----> Fl
2120 ----> ow
2 ----> ,
48 ----> from
4249 ----> basi
4429 ----> cs
7 ----> to
2652 ----> master
8050 ----> y
```

"...sub-word meanings are often nonsensical and it's only when we put them together in sequences that they have meaningful semantics."





Source:
https://www.tensorflow.org/datasets/catalog/imdb_reviews
https://www.tensorflow.org/datasets/api_docs/python/tfds/features/text/SubwordTextEncoder

Let’s continue our NLP adventure 😀

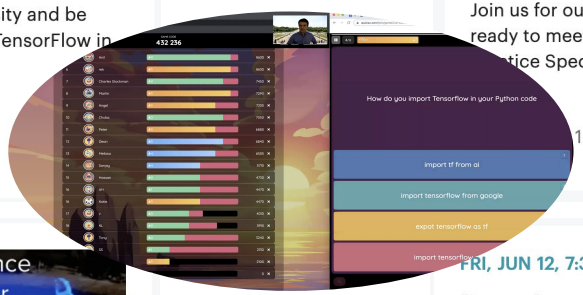
FRI, MAY 22, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 1,2 - Sentiment in text & Wor...

Online event

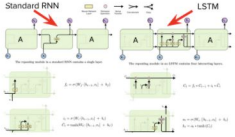
Join us for our 6th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

31 attendees



FRI, MAY 29, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 3 - Sequence models



Online event

Join us for our 7th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in tice Specialization available at:...

15 attendees

Attend

FRI, JUN 5, 7:30 PM EDT

TensorFlow in Practice - C3 Week 4 - Sequence models & literature +...

Online event

Join us for our 8th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

13 attendees

Attend

FRI, JUN 12, 7:30 PM EDT

Deep Learning Adventures - Happy Hour 🍹🍸🥂

Online event

Join us for a fun conversation 🍹🍸🥂 No slides or recording this time, just getting to know each other better and forge a stronger community 😀 We are taking a break from our TensorFlow in Practice Specialization available at:...

9 attendees

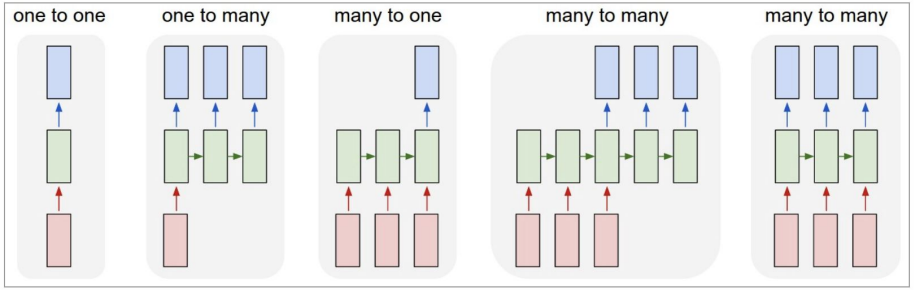
Attend

Source: https://www.meetup.com/Deep-Learning-Adventures

Course 3: Natural Language Processing in TensorFlow

Week 1: Sentiment in text
Week 2: Word Embeddings
Week 3: Sequence models
Week 4: Sequence models and literature

Sequence models



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

```
rnn = RNN()
y = rnn.step(x) # x is an input vector, y is the RNN's output vector
```

We can form a 2-layer recurrent network as follows:

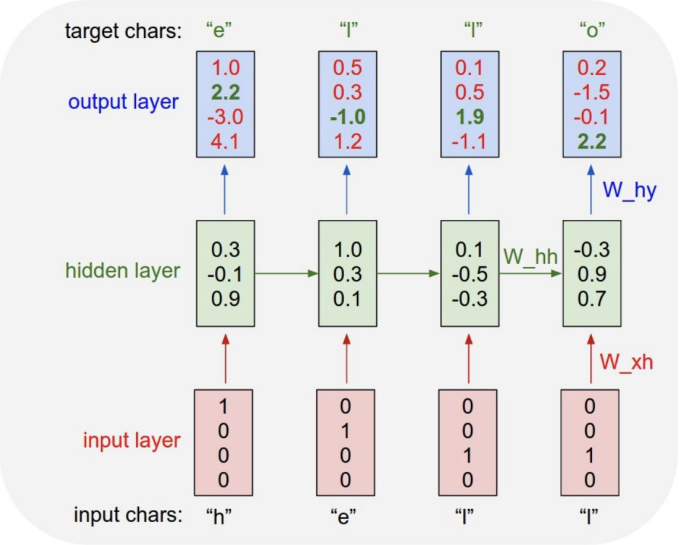
```
y1 = rnn1.step(x)
y = rnn2.step(y1)
```

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

```
self.h = ...
```

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

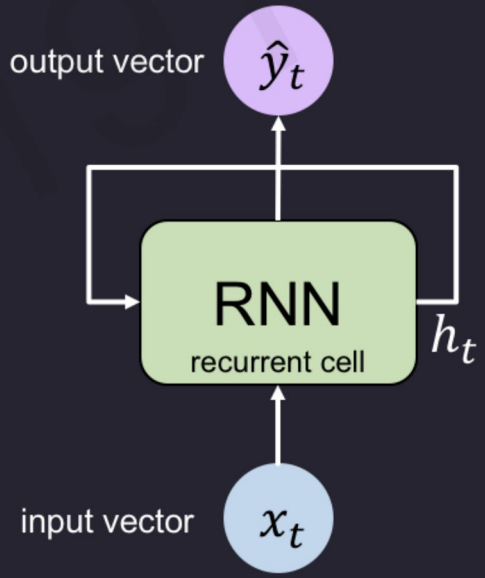
$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$, where tanh is applied elementwise.



Sequence models

RNN Intuition

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```



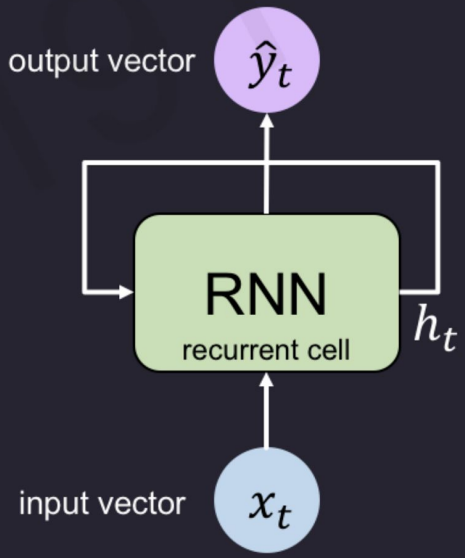
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

Sequence models

RNNs from Scratch

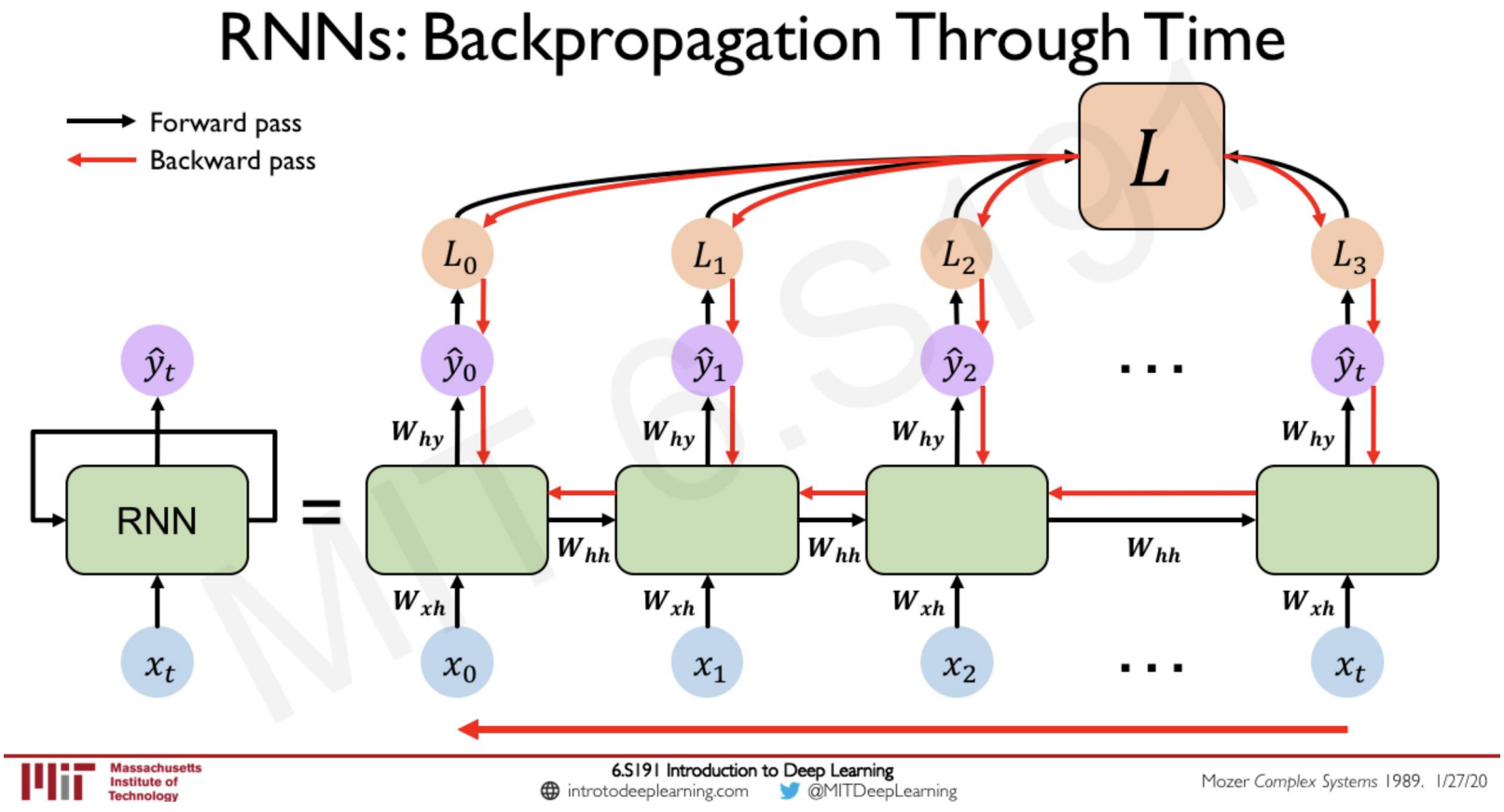
`tf.keras.layers.SimpleRNN(rnn_units)`

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



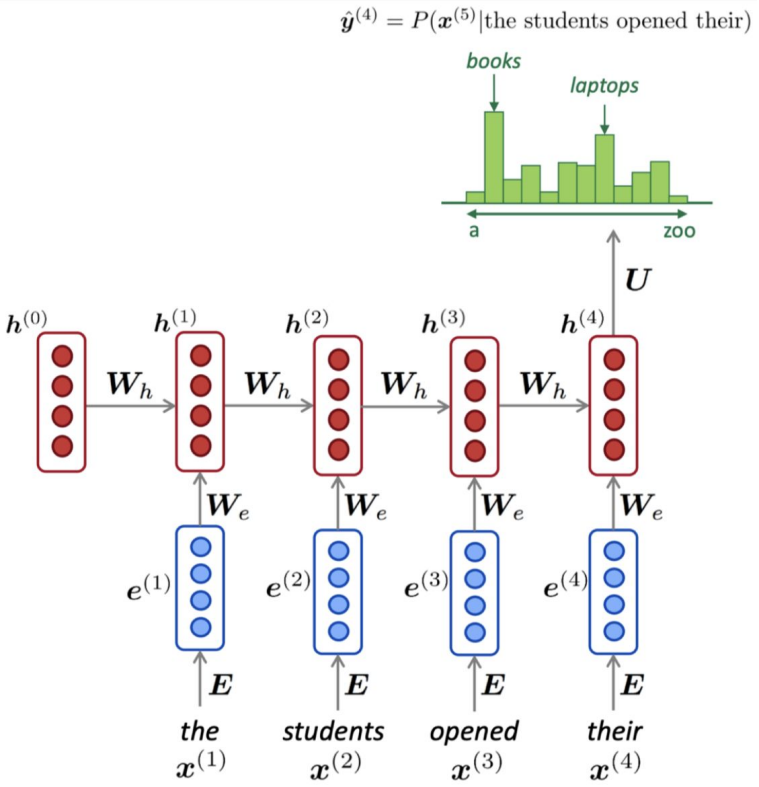
Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

Sequence models

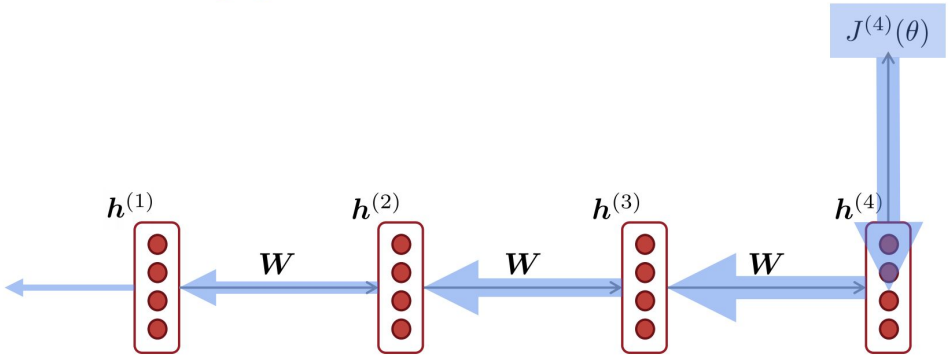


Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

Sequence models - new



Vanishing gradient intuition



What happens if these are small?

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Sequence models - new

$$h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}) \quad (5)$$

$$\hat{y}_t = softmax(W^{(S)} h_t) \quad (6)$$

The loss function used in RNNs is often the cross entropy error introduced in earlier notes. Equation 7 shows this function as the sum over the entire vocabulary at time-step t.

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j}) \quad (7)$$

The cross entropy error over a corpus of size T is:

$$J = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{|V|} y_{t,j} \times log(\hat{y}_{t,j}) \quad (8)$$

Equation 9 is called the *perplexity* relationship; it is basically 2 to the power of the negative log probability of the cross entropy error function shown in Equation 8. Perplexity is a measure of confusion where lower values imply more confidence in predicting the next word in the sequence (compared to the ground truth outcome).

$$Perplexity = 2^J \quad (9)$$

- $x_1, ..., x_{t-1}, x_t, x_{t+1}, ...x_T$: the word vectors corresponding to a corpus with T words.
- $h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$: the relationship to compute the hidden layer output features at each time-step t
  - $x_t \in \mathbb{R}^d$: input word vector at time t.
  - $W^{hx} \in \mathbb{R}^{D_h \times d}$: weights matrix used to condition the input word vector, x_t
  - $W^{hh} \in \mathbb{R}^{D_h \times D_h}$: weights matrix used to condition the output of the previous time-step, h_{t-1}
  - $h_{t-1} \in \mathbb{R}^{D_h}$: output of the non-linear function at the previous time-step, $t-1$. $h_0 \in \mathbb{R}^{D_h}$ is an initialization vector for the hidden layer at time-step $t = 0$.
  - $\sigma()$: the non-linearity function (sigmoid here)
- $\hat{y}_t = softmax(W^{(S)} h_t)$: the output probability distribution over the vocabulary at each time-step t. Essentially, $\hat{y}_t$ is the next predicted word given the document context score so far (i.e. h_{t-1}) and the last observed word vector $x^{(t)}$. Here, $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$ and $\hat{y} \in \mathbb{R}^{|V|}$ where $|V|$ is the vocabulary.

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Sequence models - new

Consider Equations 5 and 6 at a time-step t; to compute the RNN error, dE/dW, we sum the error at each time-step. That is, dE_t/dW for every time-step, t, is computed and accumulated.

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \tag{10}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \tag{11}$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W} \tag{14}$$

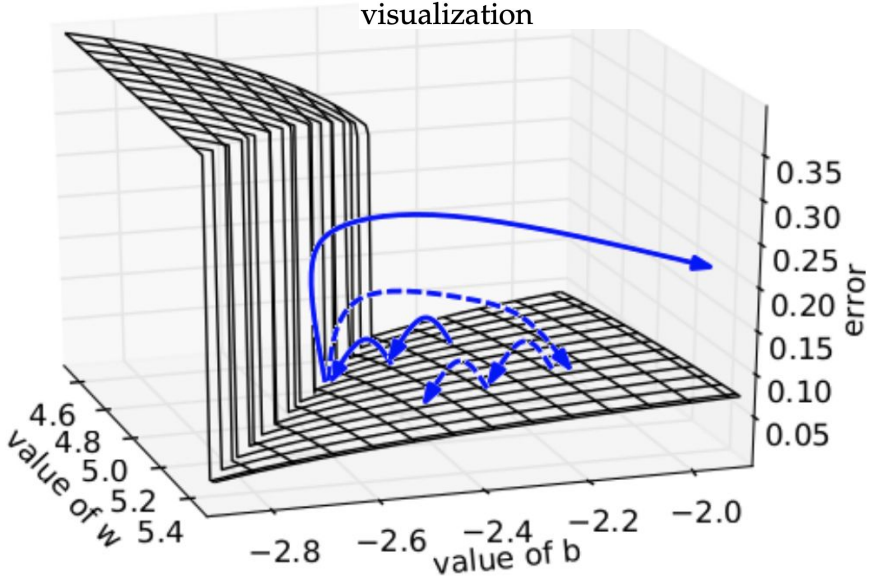
$$\hat{g} \leftarrow \frac{\partial E}{\partial W}$$
if $\| \hat{g} \| \geq threshold$ **then**
$$\hat{g} \leftarrow \frac{threshold}{\| \hat{g} \|} \hat{g}$$
end if

Algorithm 1: Pseudo-code for norm clipping in the gradients whenever they explode

$$\| \frac{\partial h_t}{\partial h_k} \| = \| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \| \leq (\beta_W \beta_h)^{t-k} \tag{16}$$

The exponential term $(\beta_W \beta_h)^{t-k}$ can easily become a very small or large number when $\beta_W \beta_h$ is much smaller or larger than 1 and $t-k$ is sufficiently large. Recall a large $t-k$ evaluates the cross entropy error due to faraway words. The contribution of faraway words to predicting the next word at time-step t diminishes when the gradient vanishes early on.

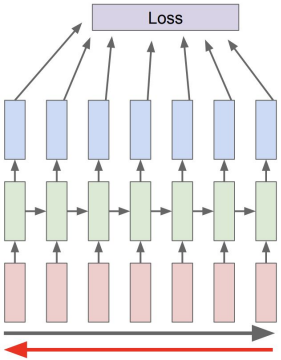
Figure 7: Gradient explosion clipping visualization



Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Sequence models - new

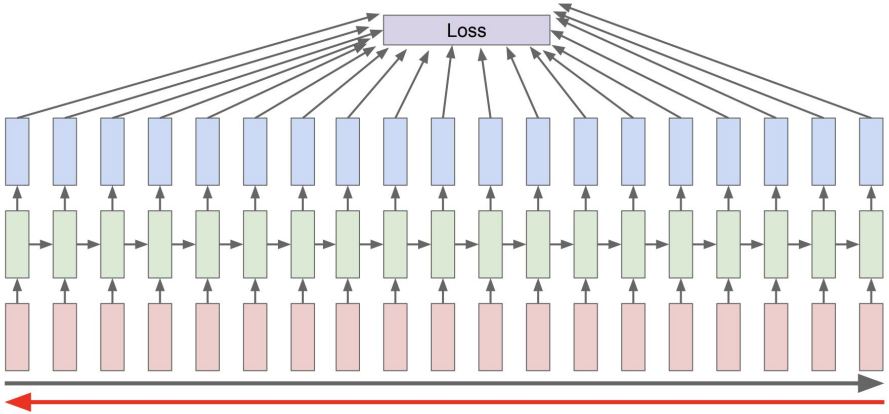
Truncated Backpropagation through time



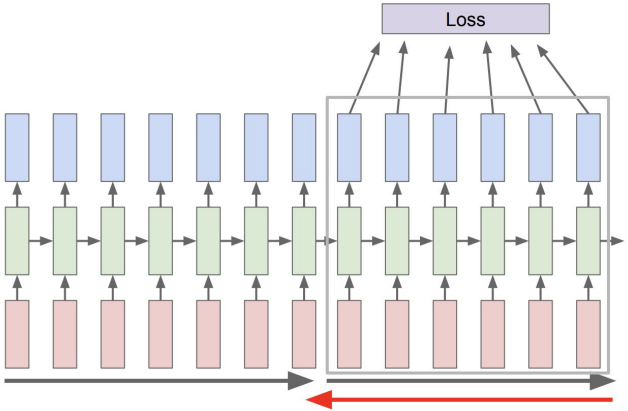
Run forward and backward through chunks of the sequence instead of whole sequence

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

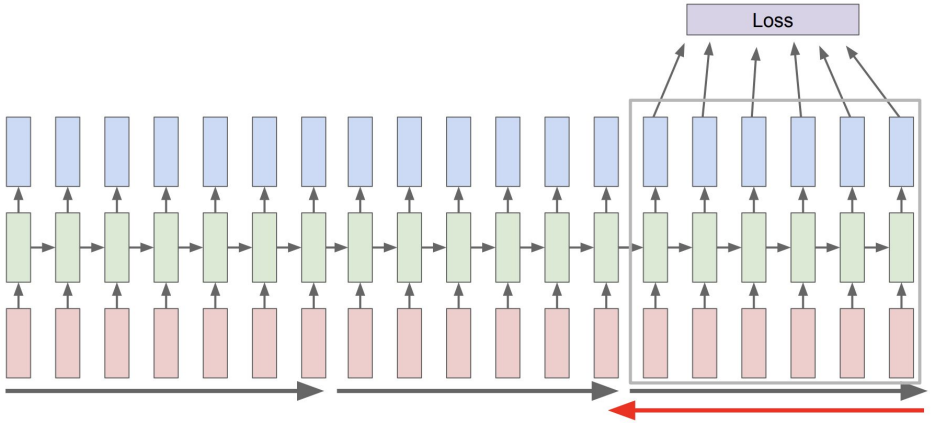


Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

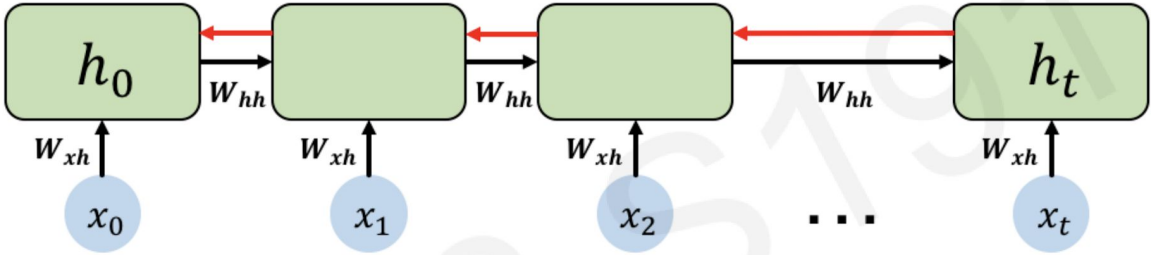
Truncated Backpropagation through time



Source: http://cs231n.stanford.edu/slides/2020/lecture_10.pdf

Sequence models

Standard RNN Gradient Flow: Exploding Gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

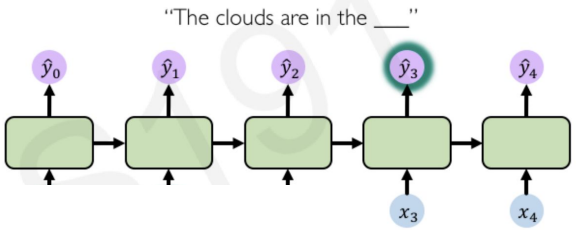
Many values < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

Sequence models - new

The Problem of Long-Term Dependencies



A Simple Way to Initialize Recurrent Networks of Rectified Linear Units

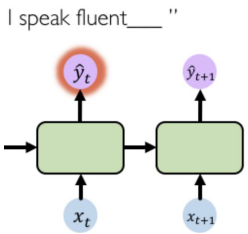
Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton
Google

2 The initialization trick

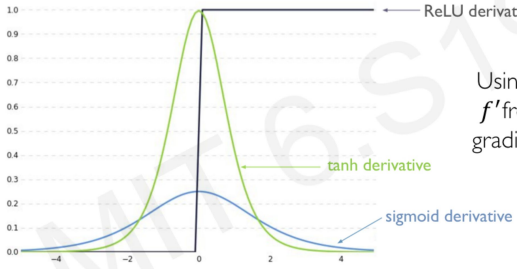
In this paper, we demonstrate that, with the right initialization of the weights, RNNs composed of rectified linear units are relatively easy to train and are good at modeling long-range dependencies. The RNNs are trained by using backpropagation through time to get error-derivatives for the weights and by updating the weights after each small mini-batch of sequences. Their performance on test data is comparable with LSTMs, both for toy problems involving very long-range temporal structures and for real tasks like predicting the next word in a very large corpus of text.

We initialize the recurrent weight matrix to be the identity matrix and biases to be zero. This means that each new hidden state vector is obtained by simply copying the previous hidden vector then adding on the effect of the current inputs and replacing all negative states by zero. In the absence of input, an RNN that is composed of ReLUs and initialized with the identity matrix (which we call an IRNN) just stays in the same state indefinitely. The identity initialization has the very desirable property that when the error derivatives for the hidden units are backpropagated through time they remain constant provided no extra error-derivatives are added. This is the same behavior as LSTMs when their forget gates are set so that there is no decay and it makes it easy to learn very long-range temporal dependencies.

We also find that for tasks that exhibit less long range dependencies, scaling the identity matrix by a small scalar is an effective mechanism to forget long range effects. This is the same behavior as LTSMs when their forget gates are set so that the memory decays fast.



Trick #1: Activation Functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Solution #3: Gated Cells

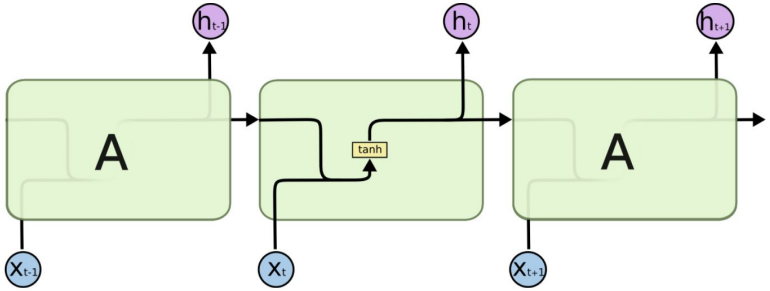
Idea: use a more **complex recurrent unit with gates** to control what information is passed through

gated cell
LSTM, GRU, etc.

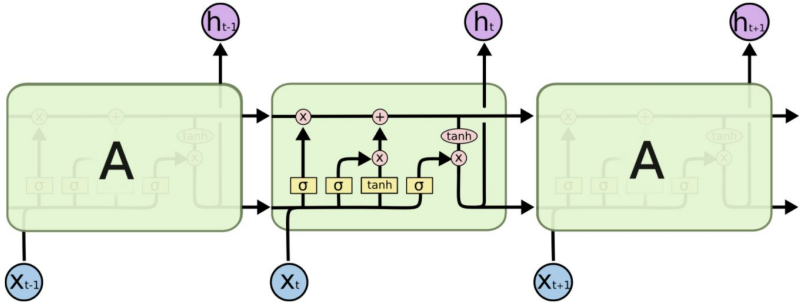
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf
https://arxiv.org/pdf/1504.00941.pdf

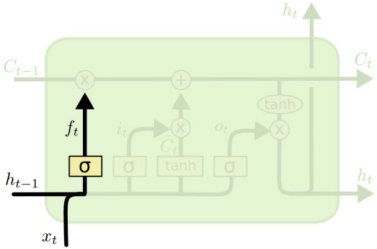
Sequence models



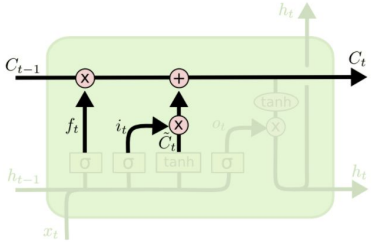
The repeating module in a standard RNN contains a single layer.



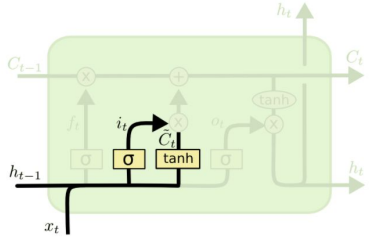
The repeating module in an LSTM contains four interacting layers.



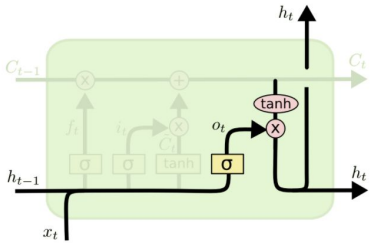
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



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



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



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

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

Sequence models - new

Long Short-Term Memory (LSTM)

We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

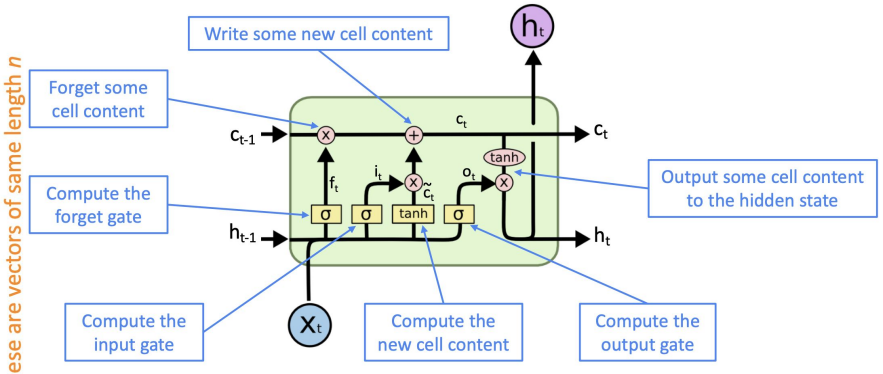
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length n



Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Sequence models - new

Gated Recurrent Units (GRU)

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.

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep *t* we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

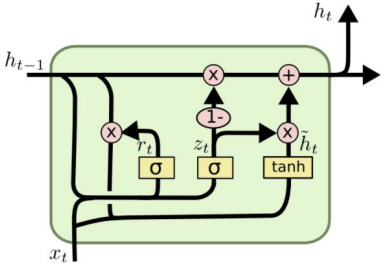
$$u^{(t)} = \sigma\left(W_u h^{(t-1)} + U_u x^{(t)} + b_u\right)$$

$$r^{(t)} = \sigma\left(W_r h^{(t-1)} + U_r x^{(t)} + b_r\right)$$

$$\tilde{h}^{(t)} = \tanh\left(W_h(r^{(t)} \circ h^{(t-1)}) + U_h x^{(t)} + b_h\right)$$

$$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)



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

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

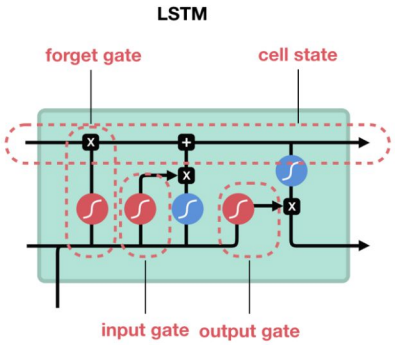
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

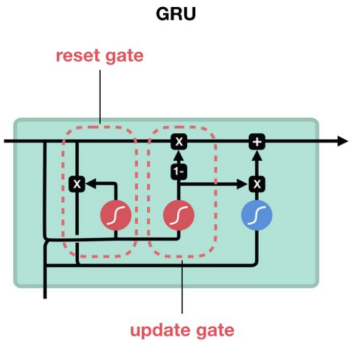
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

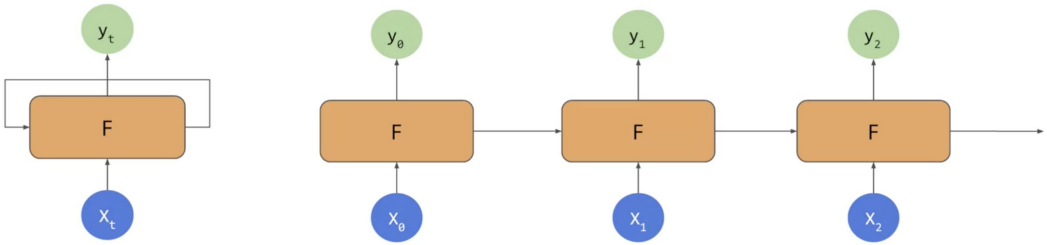
"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

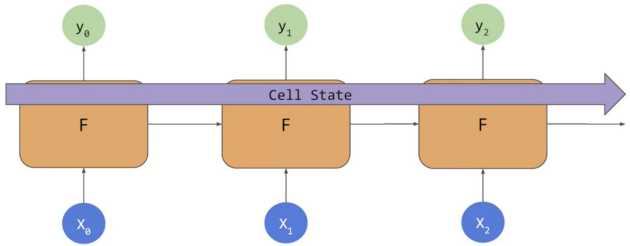
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequence models







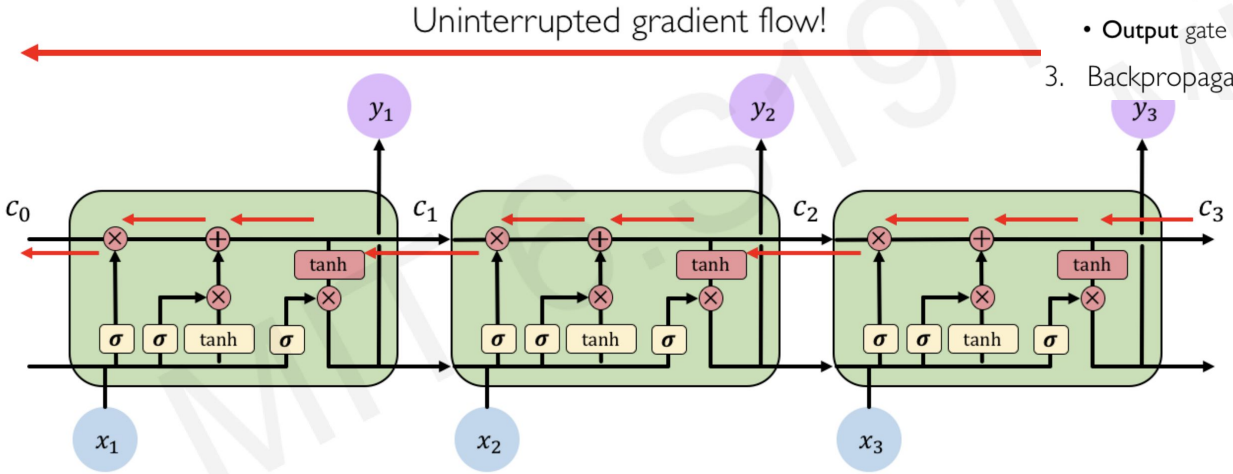


Source: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
https://www.coursera.org/learn/natural-language-processing-tensorflow/lecture/09WN5/introduction

Sequence models

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with **uninterrupted gradient flow**

LSTM Gradient Flow




```
tf.keras.layers.LSTM(num_units)
```

Source: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

Sequence models - new

Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:

- Residual cc
- Also knowı
- The identit preserves i
- This makes easier to tı

30

For example:

- Dense con
- Directly cc to all futu

31

For example:

- Highway co
- Similar to re transformat
- Inspired by

32

- Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

34

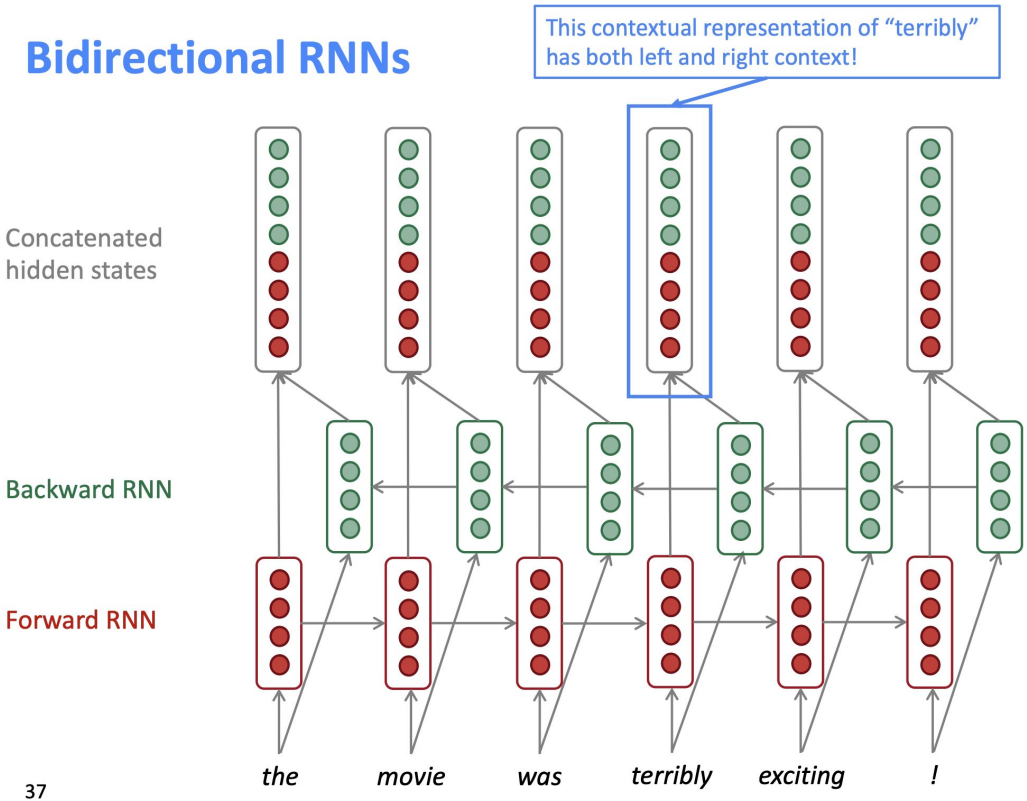
"Learning Long-Term Dependencies with Gradient Descent is Difficult", Bengio et al. 1994, http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf

"Highway Networks", Srivastava et al, 2015. https://arxiv.org/pdf/1505.00387.pdf

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf

Sequence models - new

Bidirectional RNNs



On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

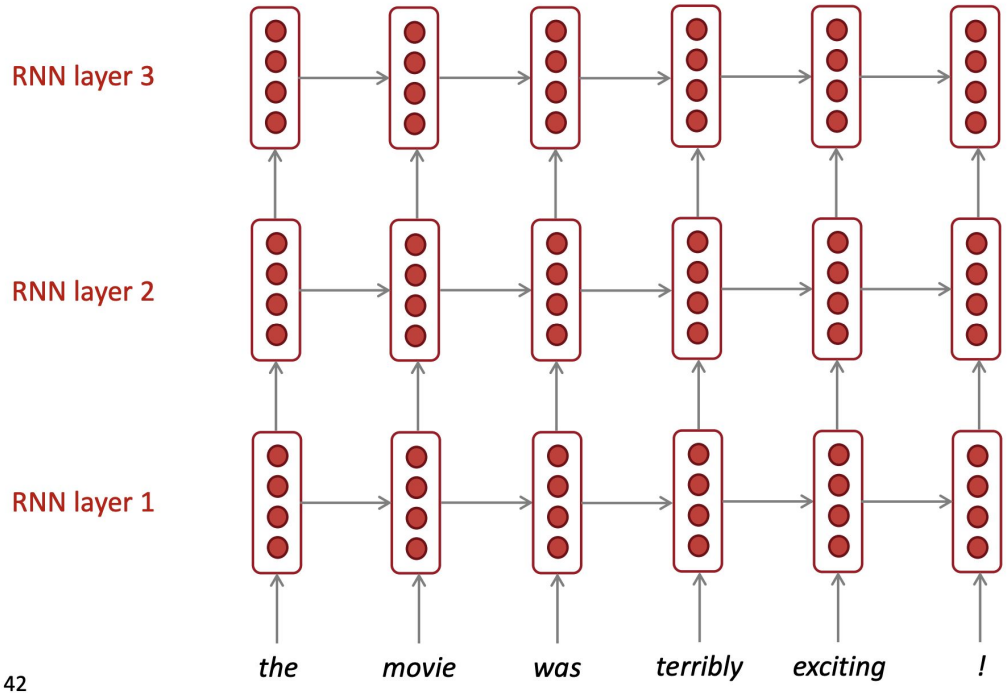
- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence.
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).
- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional

Sequence models - new

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*



- RNNs are already "deep" on one dimension (they unroll over many timesteps)
- We can also make them "deep" in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.
- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers)
- Transformer-based networks (e.g. BERT) are frequently deeper, like 12 or 24 layers.
  - You will learn about Transformers later; they have a lot of skipping-like connections

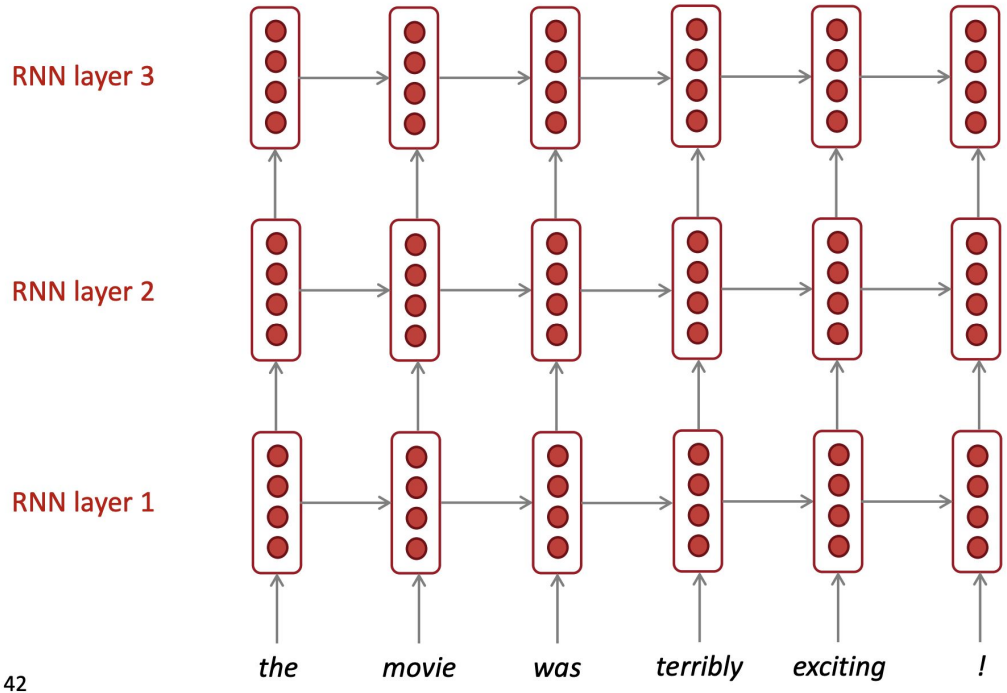
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf
http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

"Massive Exploration of Neural Machine Translation Architecutres", Britz et al, 2017. https://arxiv.org/pdf/1703.03906.pdf

Sequence models - new

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*



A note on terminology

The RNN described in this lecture = simple/vanilla/Elman RNN

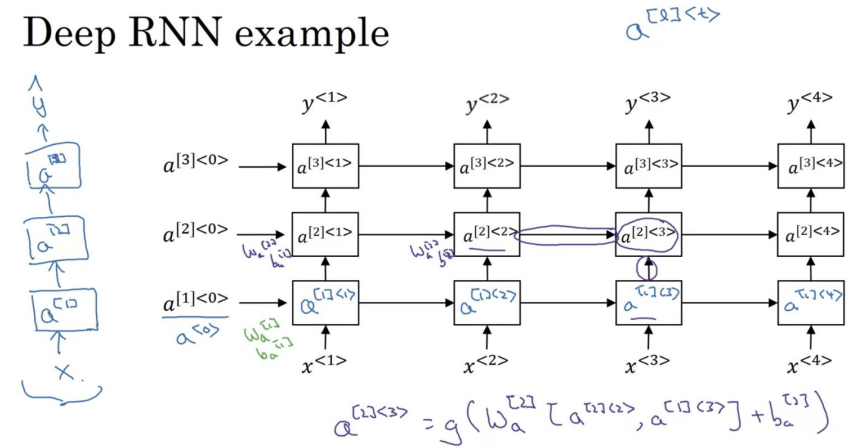
Next lecture: You will learn about other RNN flavors like GRU  and LSTM  and multi-layer RNNs 

By the end of the course: You will understand phrases like *"stacked bidirectional LSTM with residual connections and self-attention"*

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf
http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

Sequence models

Deep RNN example



Sequence Models

deeplearning.ai

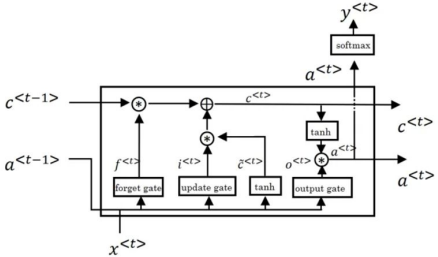
4.8 (20,428 ratings) | 180K Students Enrolled

Course 5 of 5 in the Deep Learning Specialization

Enroll for Free

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



Sequence Models

deeplearning.ai

4.8 (20,446 ratings) | 180K Students Enrolled

Course 5 of 5 in the Deep Learning Specialization

Enroll for Free

Source: https://www.coursera.org/lecture/nlp-sequence-models/deep-rnns-ehs0S
https://www.coursera.org/lecture/nlp-sequence-models/long-short-term-memory-lstm-KXoay

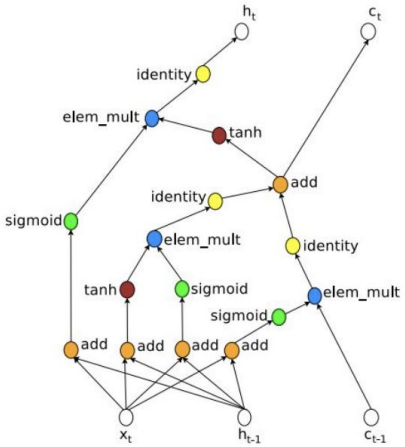
Sequence models - new

LSTM

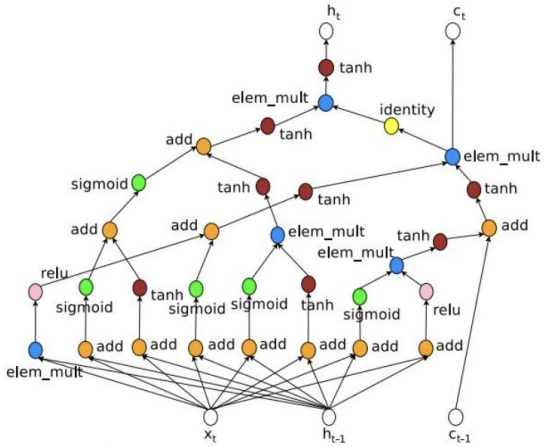
Klaus Greff, Rupesh

This paper re
variants of the
most commonly
performs reasona
investigated moc
However, certain
forget gates (CI
simplified LSTM
decreasing perfo
because they r
computational c

Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

Zoph et Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Source: https://www.youtube.com/watch?v=6niqTuYFZLQ
https://arxiv.org/abs/1503.04069
http://proceedings.mlr.press/v37/jozefowicz15.pdf
http://cs231n.stanford.edu/slides/2020/lecture_10.pdf

ion of
ectures

network ar-
reliably out-
tectures that
we were un-
at the LSTM

h procedure
tly different
highest per-
milar to the
would have
st of evalu-
bility of do-
arch proce-
dure failed to dramatically improve over the LSTM sug-
gests that, at the very least, if there are architectures that
are much better than the LSTM, then they are not trivial to
find.

Sequence models - new

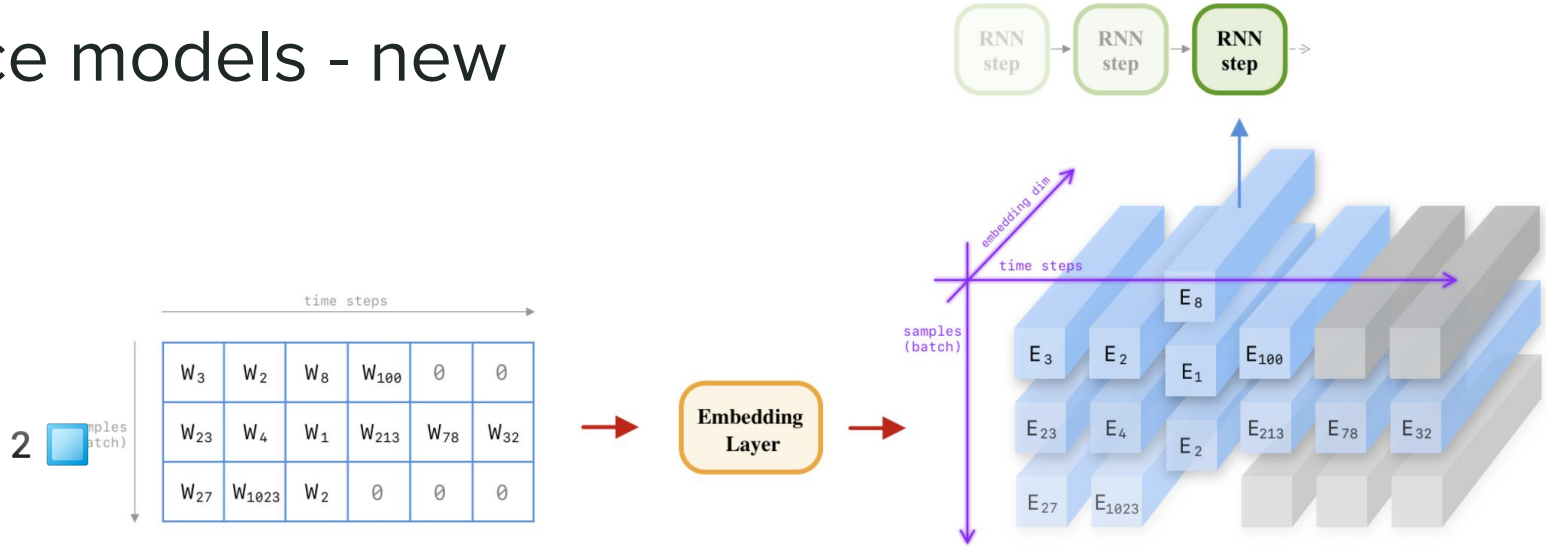


Figure 2. E_i is the embedding vector for word W_i. This tensor has [batch = 3, max_len = 6, features = 300] shape, and is ready to be fed into an RNN network directly.

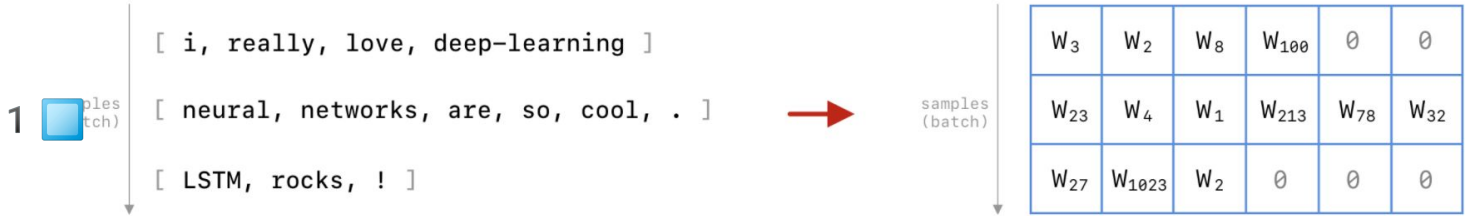


Figure 1. Batch size = 3, and W_i = Word ID assigned by the vocabulary, and 0 is reserved for padding; Here max_len = 6, and sentence 1 and 3 are padded to match up with the longest sentence

Source: https://iust-deep-learning.github.io/972/static_files/assignments/assignment_05_preview.html

Sequence models - new

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
```

```
tf.keras.layers.Embedding(
    input_dim, output_dim, embeddings_initializer='uniform',
    embeddings_regularizer=None, activity_regularizer=None,
    embeddings_constraint=None, mask_zero=False, input_length=None, **kwargs
)
```

```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
# the model will take as input an integer matrix of size (batch,
# input_length).
# the largest integer (i.e. word index) in the input should be no larger
# than 999 (vocabulary size).
# now model.output_shape == (None, 10, 64), where None is the batch
# dimension.

input_array = np.random.randint(1000, size=(32, 10))

model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
assert output_array.shape == (32, 10, 64)
```

TensorFlow > API > TensorFlow Core v2.2.0 > Python

tf.keras.layers.Embedding

Turns positive integers (indexes) into dense vectors of fixed size.

Inherits From: `Layer`

`input_dim`	int > 0. Size of the vocabulary, i.e. maximum integer index + 1.
`output_dim`	int >= 0. Dimension of the dense embedding.

Input shape:

2D tensor with shape: `(batch_size, input_length)`.

Output shape:

3D tensor with shape: `(batch_size, input_length, output_dim)`.

Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

Sequence models - new

TensorFlow > API > TensorFlow Core v2.2.0 > Python

tf.keras.layers.Flatten

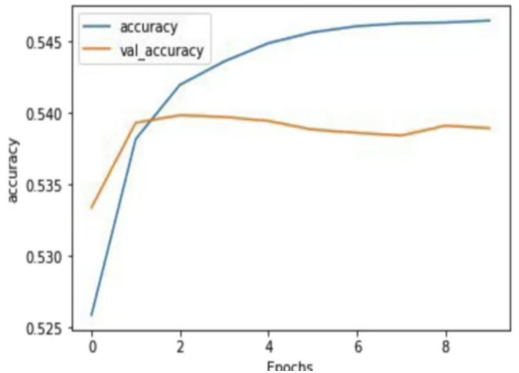
Flattens the input. Does not affect the batch size.

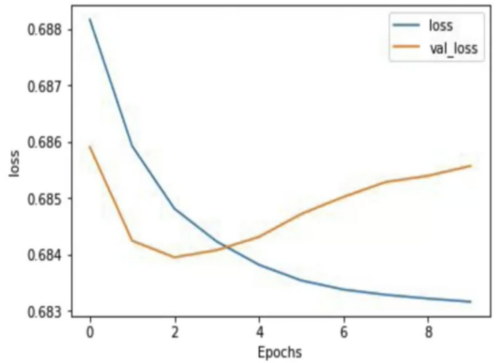
Inherits From: `Layer`

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(6, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 16)           160000
_________________________________________________________________
flatten (Flatten)            (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 11526
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 7
=================================================================
Total params: 171,533
Trainable params: 171,533
Non-trainable params: 0
_________________________________________________________________
```





Sequence models

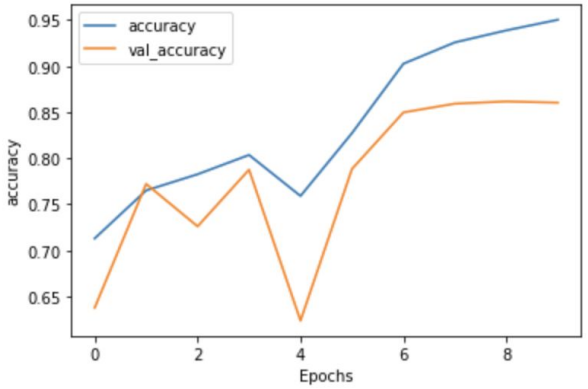
```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

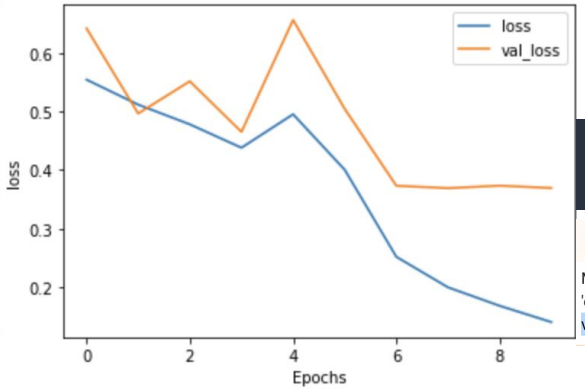
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
NUM_EPOCHS = 10
history = model.fit(train_dataset, epochs=NUM_EPOCHS, validation_data=test_dataset)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          523840
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               66048
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
_________________________________________________________________
```





TensorFlow > API > TensorFlow Core v2.2.0 > Python

tf.keras.layers.Bidirectional

```
tf.keras.layers.Bidirectional(
    layer, merge_mode='concat', weights=None, backward_layer=None, **kwargs
)
```

merge_mode

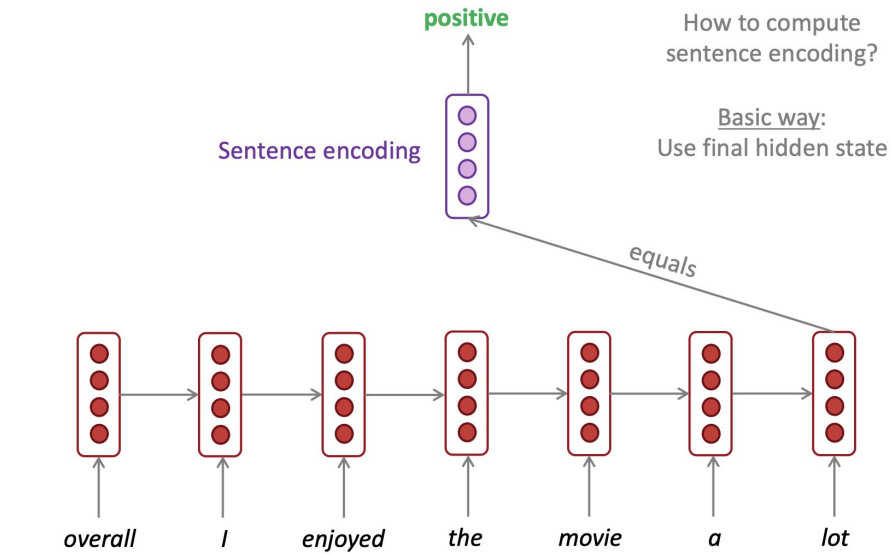
Mode by which outputs of the forward and backward RNNs will be combined. One of {'sum', 'mul', 'concat', 'ave', None}. If None, the outputs will not be combined, they will be returned as a list. Default value is 'concat'.

Sequence models - new

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True, as_supervised=True)
train_dataset, test_dataset = dataset['train'], dataset['test']
```

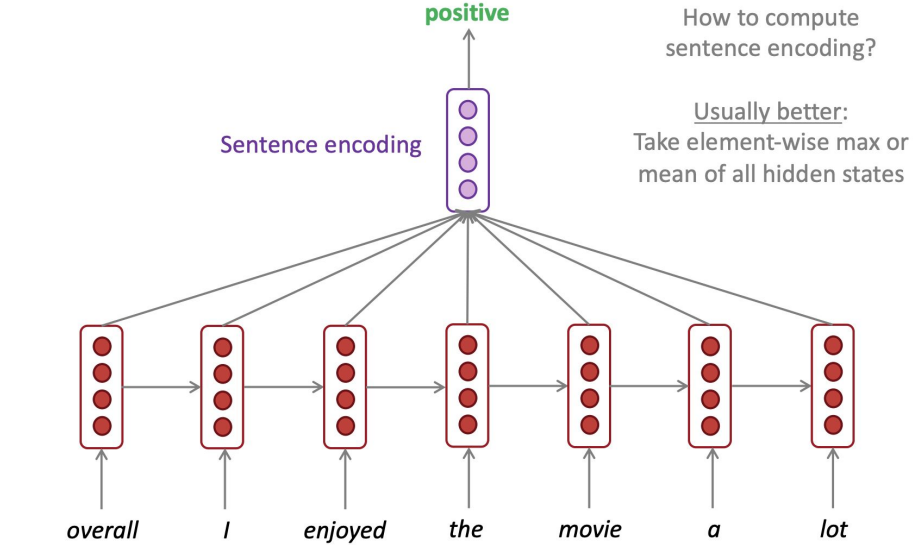
RNNs can be used for sentence classification

e.g. sentiment classification



RNNs can be used for sentence classification

e.g. sentiment classification



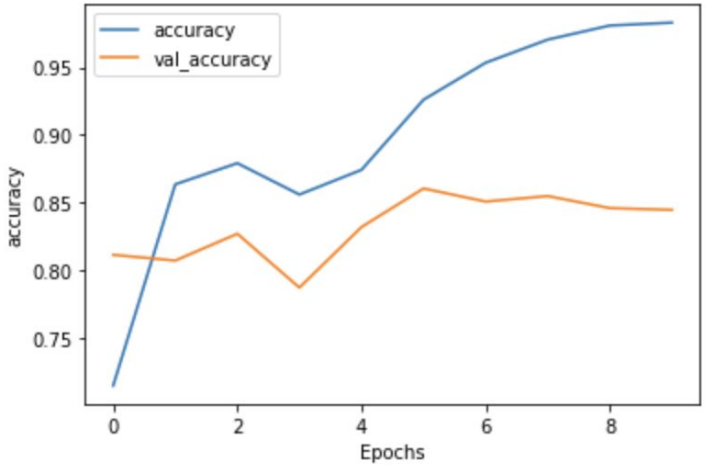
Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

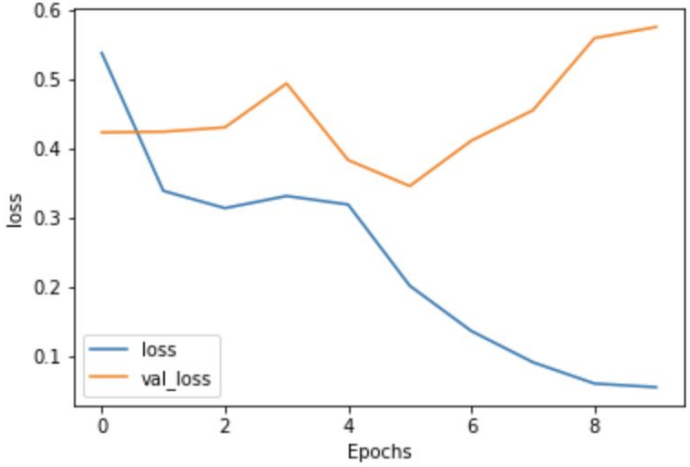
```
tf.keras.layers.Bidirectional(
    layer, merge_mode='concat', weights=None, backward_layer=None, **kwargs
)
```

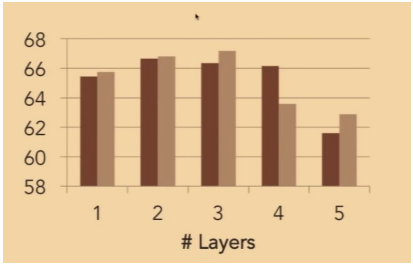
Sequence models

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 128)         66048
_________________________________________________________________
bidirectional_2 (Bidirection (None, 64)                41216
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 635,329
Trainable params: 635,329
Non-trainable params: 0
_________________________________________________________________
```

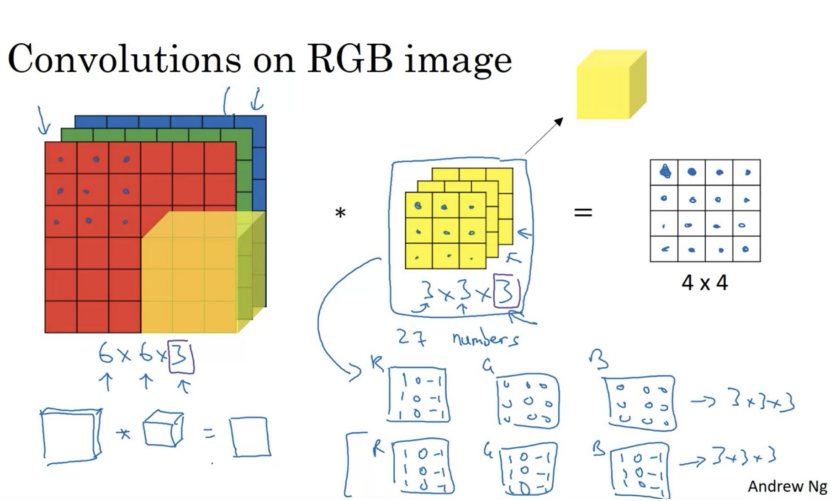


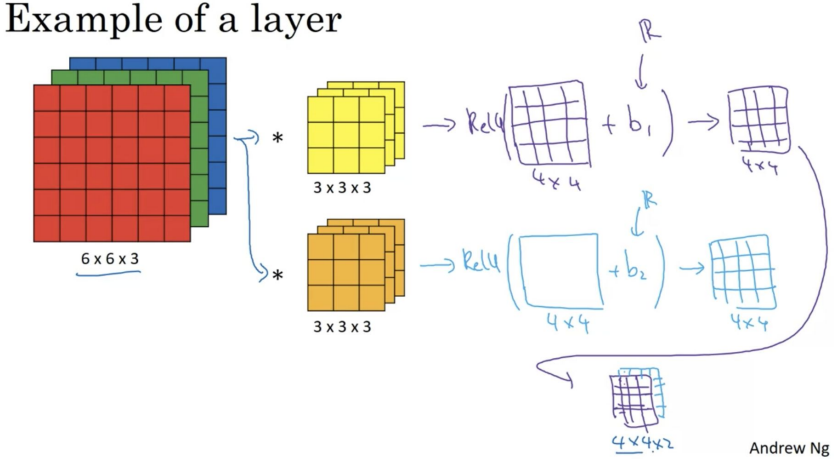


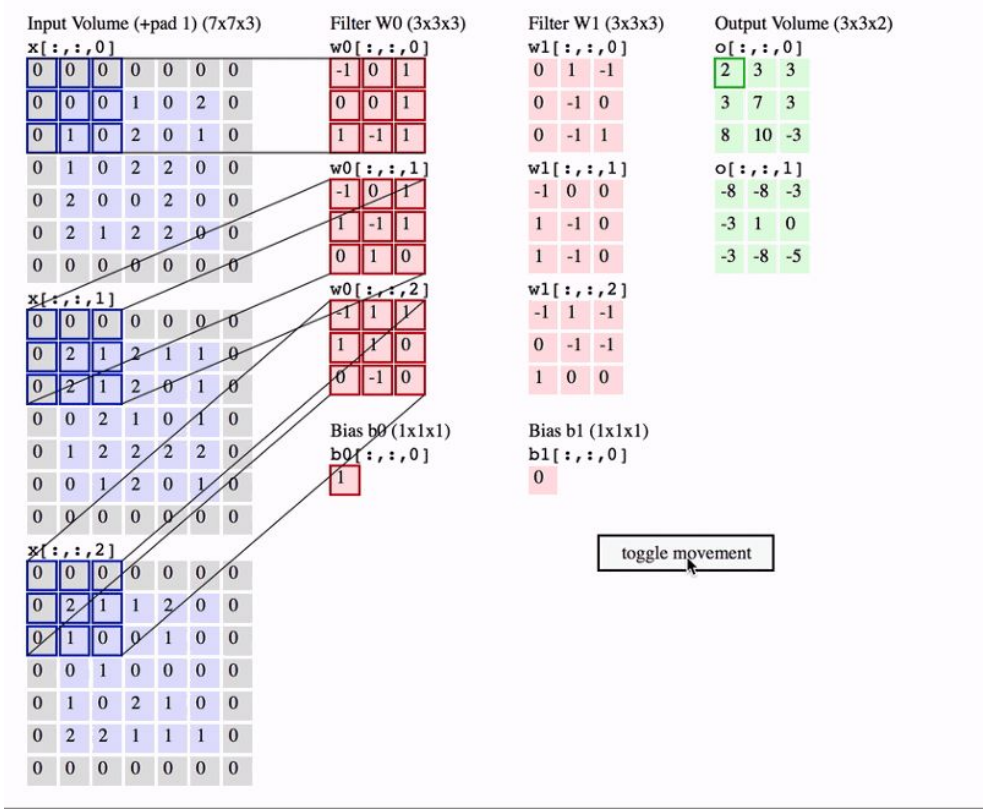


Source: https://www.youtube.com/watch?v=Keqep_PKrY8

Reminder from Presentation-3 on CNNs



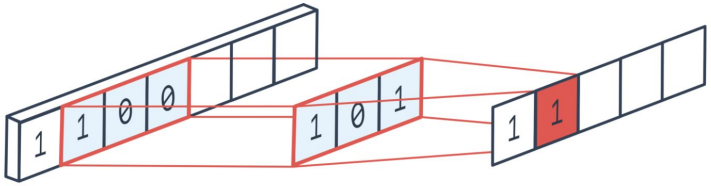




Source: https://www.coursera.org/learn/convolutional-neural-networks
https://images.app.goo.gl/Z4xDvjzi3Gcerpm27

Sequence models - new

TensorFlow > API > TensorFlow Core v2.2.0 > Python



Conv1d-Input1d Example [Image [12]]

tf.keras.layers.Conv1D

1D convolution layer (e.g. temporal convolution).

```
tf.keras.layers.Conv1D(
    filters, kernel_size, strides=1, padding='valid', data_format='channels_last',
    dilation_rate=1, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)
```

```
>>> # The inputs are 128-length vectors with 10 timesteps, and the batch size
>>> # is 4.
>>> input_shape = (4, 10, 128)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv1D(
... 32, 3, activation='relu',input_shape=input_shape)(x)
>>> print(y.shape)
(4, 8, 32)
```

`filters`	Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
`kernel_size`	An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
`strides`	An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.

Input shape:

3D tensor with shape: `(batch_size, steps, input_dim)`

Output shape:

3D tensor with shape: `(batch_size, new_steps, filters)` `steps` value might have changed due to padding or strides.

Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D
https://towardsdatascience.com/pytorch-basics-how-to-train-your-neural-net-intro-to-cnn-26a14c2ea29

Sequence models - new

TensorFlow > API > TensorFlow Core v2.2.0 > Python

tf.keras.layers.AveragePooling1D

Average pooling for temporal data.

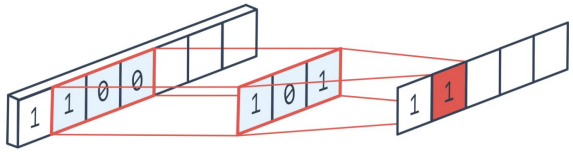
```
tf.keras.layers.AveragePooling1D(
    pool_size=2, strides=None, padding='valid', data_format='channels_last',
    **kwargs
)
```

Input shape:

- If `data_format='channels_last'`: 3D tensor with shape `(batch_size, steps, features)`.
- If `data_format='channels_first'`: 3D tensor with shape `(batch_size, features, steps)`.

Output shape:

- If `data_format='channels_last'`: 3D tensor with shape `(batch_size, downsampled_steps, features)`.
- If `data_format='channels_first'`: 3D tensor with shape `(batch_size, features, downsampled_steps)`.



GlobalAveragePooling1D is same as **AveragePooling1D** with **pool_size=steps**

TensorFlow > API > TensorFlow Core v2.2.0 > Python

tf.keras.layers.GlobalAveragePooling1D

Global average pooling operation for temporal data.

```
tf.keras.layers.GlobalAveragePooling1D(
    data_format='channels_last', **kwargs
)
```

```
>>> input_shape = (2, 3, 4)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.GlobalAveragePooling1D()(x)
>>> print(y.shape)
(2, 4)
```

Input shape:

- If `data_format='channels_last'`: 3D tensor with shape: `(batch_size, steps, features)`
- If `data_format='channels_first'`: 3D tensor with shape: `(batch_size, features, steps)`

Output shape:

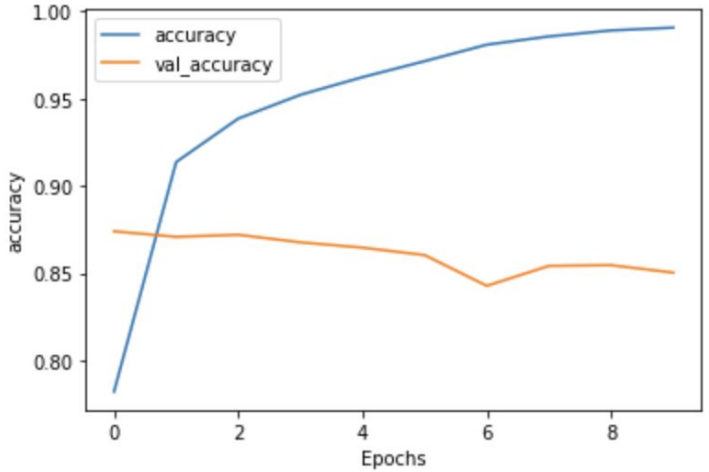
2D tensor with shape `(batch_size, features)`.

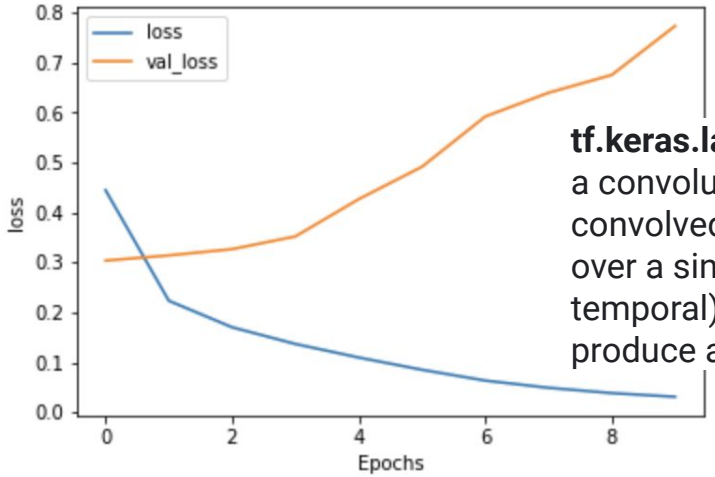
Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D
https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling1D
https://stackoverflow.com/questions/54493738/keras-difference-between-averagepooling1d-layer-and-globalaveragepooling1d-laye

Sequence models

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
conv1d (Conv1D)              (None, None, 128)         41088
_________________________________________________________________
global_average_pooling1d (Gl (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 573,249
Trainable params: 573,249
Non-trainable params: 0
_________________________________________________________________
```



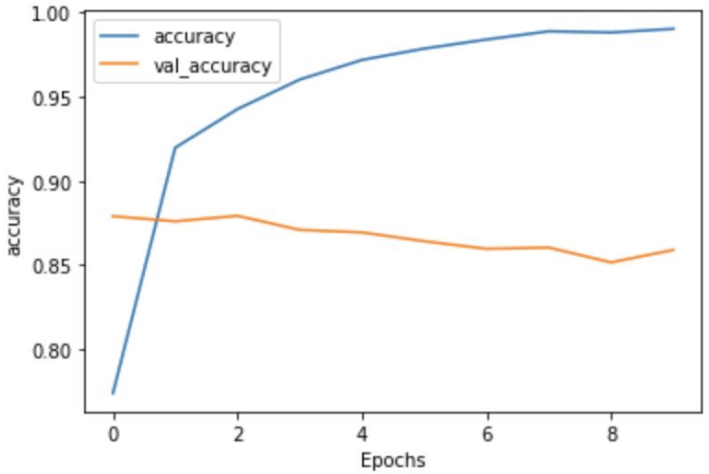


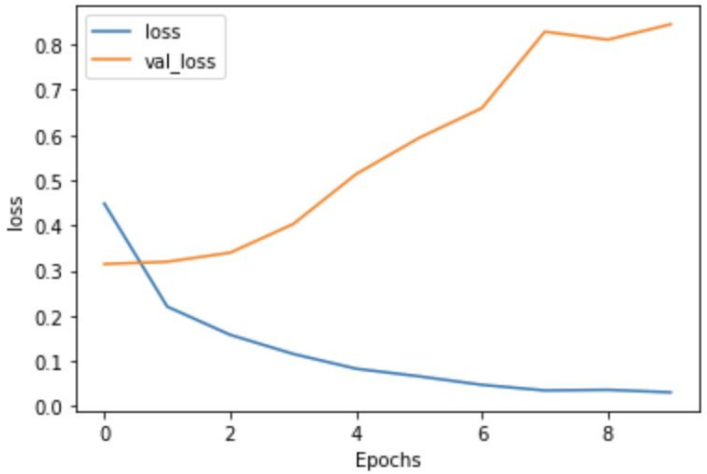
tf.keras.layers.Conv1D:
a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs

Sequence models

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.Conv1D(128, 3, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
conv1d_1 (Conv1D)            (None, None, 128)         41088
_________________________________________________________________
conv1d_2 (Conv1D)            (None, None, 128)         49280
_________________________________________________________________
global_average_pooling1d_1 ( (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 65
=================================================================
Total params: 622,529
Trainable params: 622,529
Non-trainable params: 0
_________________________________________________________________
```

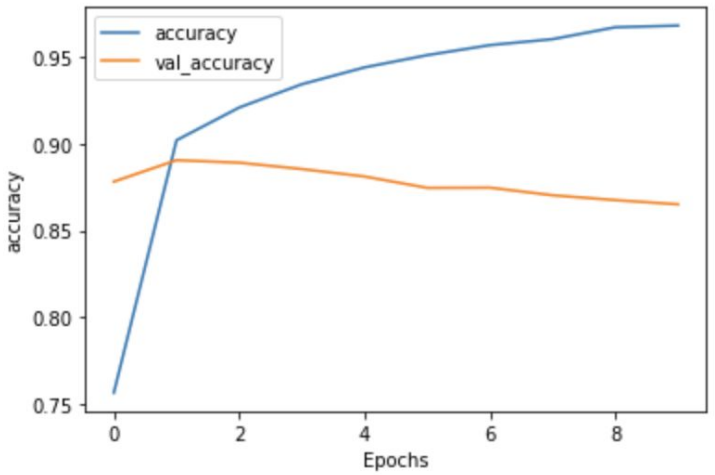


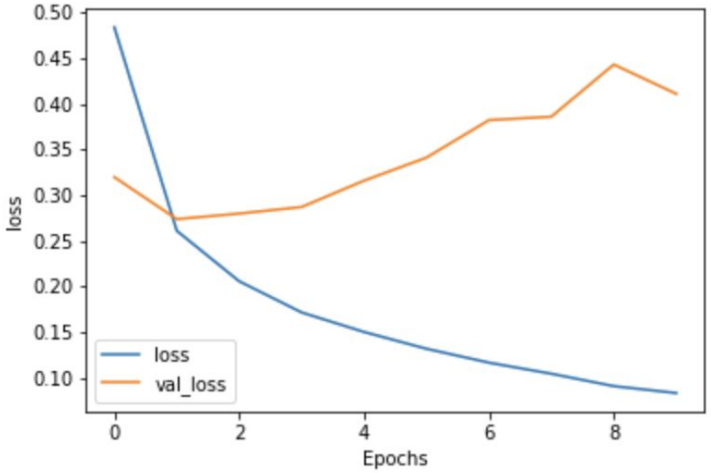


Sequence models

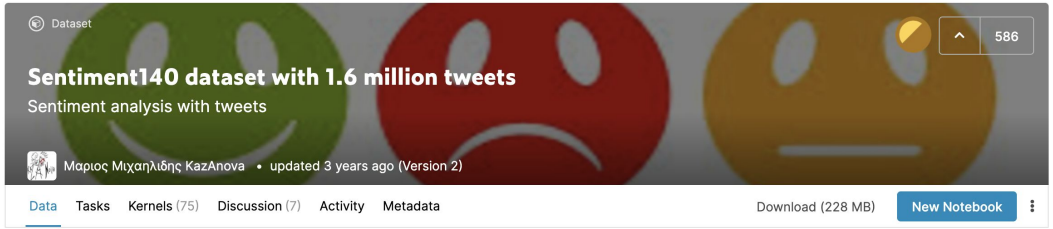
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(tokenizer.vocab_size, 64),
    tf.keras.layers.Conv1D(128, 5, activation='relu'),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
conv1d_3 (Conv1D)            (None, None, 128)         41088
_________________________________________________________________
global_average_pooling1d_2 ( (None, 128)               0
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 65
=================================================================
Total params: 573,249
Trainable params: 573,249
Non-trainable params: 0
_________________________________________________________________
```





Sequence models



1. **target**: the polarity of the tweet (*0* = negative, *2* = neutral, *4* = positive)
2. **ids**: The id of the tweet (*2087*)
3. **date**: the date of the tweet (*Sat May 16 23:58:44 UTC 2009*)
4. **flag**: The query (*lyx*). If there is no query, then this value is NO_QUERY.
5. **user**: the user that tweeted (*robotickilldozr*)
6. **text**: the text of the tweet (*Lyx is cool*)

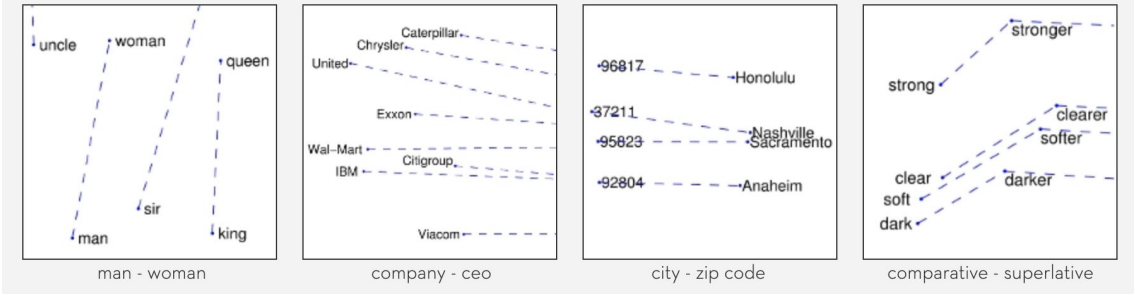
```
1600000
1600000
['0', "@alydesigns i was out most of the day so didn't get much done "]
['0', "one of my friend called me, and asked to meet with her at Mid Valley today...but i've no time *sigh* "]
['0', '@angry_barista I baked you a cake but I ated it ']
[['0', "@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer.  You shoulda got David Carr of Third Day to do it. ;D"], ['0', "is upset that he can't upda
{0, 1}
138824
1
The Second Edition of the 20-volume Oxford English Dictionary, published in 1989, contains full entries for 171,476 words in current use
138825
```

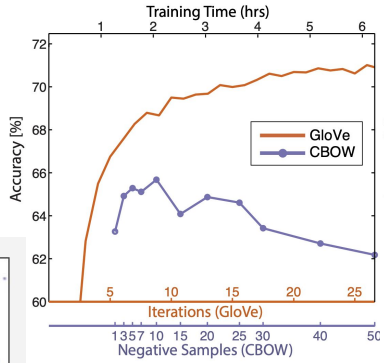
Source: https://www.kaggle.com/kazanova/sentiment140

Sequence models

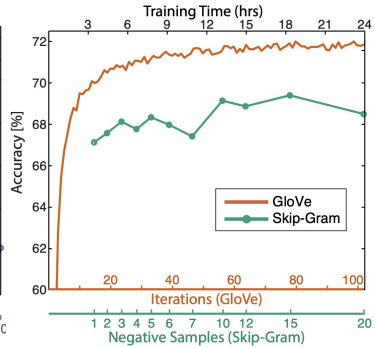
GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning





(a) GloVe vs CBOW



(b) GloVe vs Skip-Gram

Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.

GloVe is an unsupervised learning algorithm for obtaining **vector** representations for **words**. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Source: https://nlp.stanford.edu/projects/glove
https://nlp.stanford.edu/pubs/glove.pdf

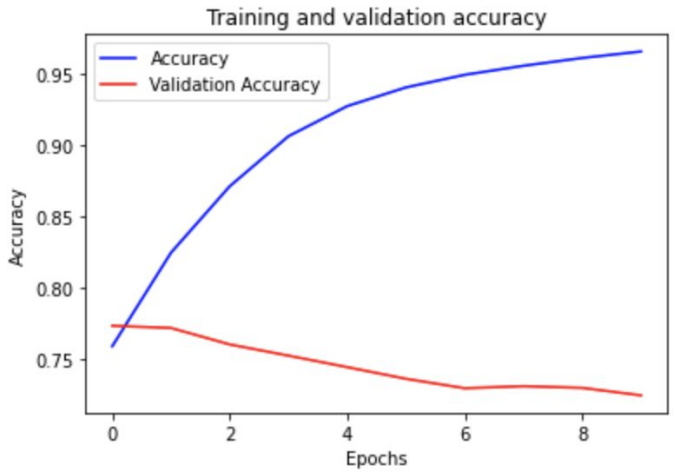
Tokenize,lowercase,Stanford tokenizer, build a vocabulary of the 400,000 most frequent words , and then construct a matrix of cooccurrence counts X.

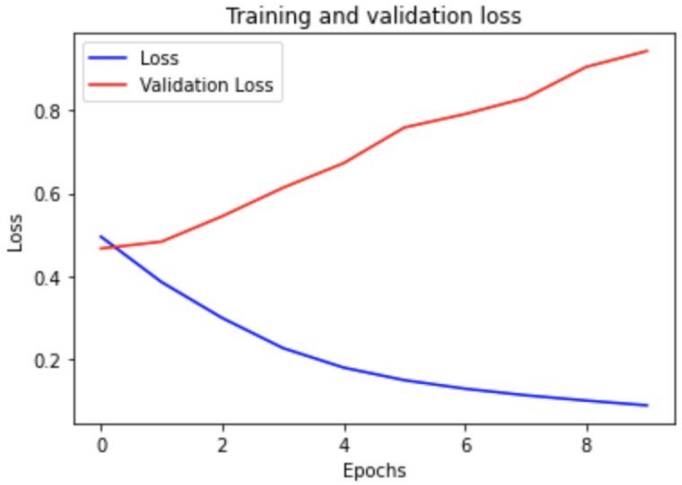
We trained our model on five corpora of varying sizes: a 2010 Wikipedia dump with 1 billion tokens; a 2014 Wikipedia dump with 1.6 billion tokens; Gigaword 5 which has 4.3 billion tokens; the combination Gigaword5 + Wikipedia2014, which has 6 billion tokens; and on 42 billion tokens of web data, from Common Crawl.

Sequence models

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 16, 100)           13910600
_________________________________________________________________
global_average_pooling1d (Gl (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 24)                2424
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 25
=================================================================
Total params: 13,913,049
Trainable params: 13,913,049
Non-trainable params: 0
_________________________________________________________________
```





Sequence models

```
with open('/tmp/glove.6B.100d.txt') as f:
    for line in f:
        values = line.split();
        word = values[0];
        coefs = np.asarray(values[1:], dtype='float32');
        embeddings_index[word] = coefs;
```

```
embeddings_matrix = np.zeros((vocab_size+1, embedding_dim));
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word);
    if embedding_vector is not None:
        embeddings_matrix[i] = embedding_vector;
```

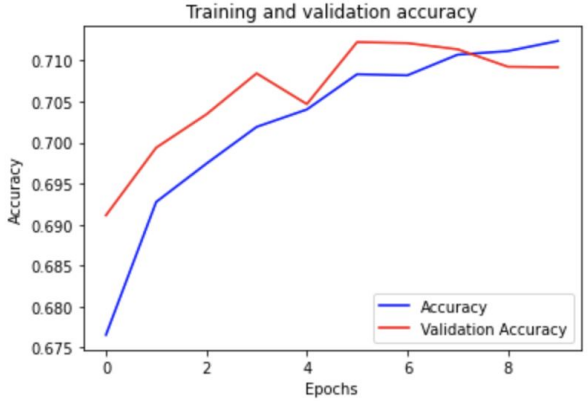
```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_16 (Embedding)     (None, 16, 100)           13870500
_________________________________________________________________
global_average_pooling1d_13  (None, 100)               0
_________________________________________________________________
dense_32 (Dense)             (None, 24)                2424
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 25
=================================================================
Total params: 13,872,949
Trainable params: 2,449
Non-trainable params: 13,870,500
_________________________________________________________________
```

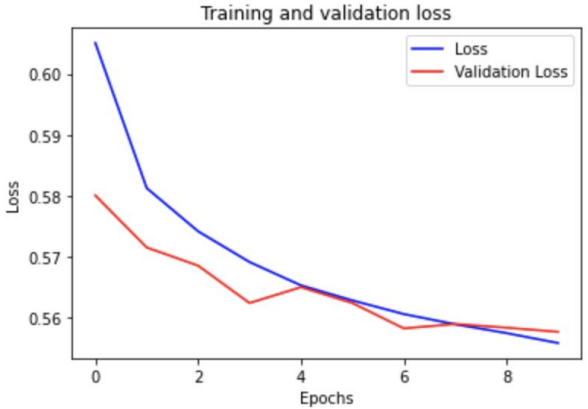
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length, weights=[embeddings_matrix], trainable=False),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

len(embeddings_matrix)
138,705
The Second Edition of the 20-volume Oxford English Dictionary, published in 1989, contains full entries for 171,476 words in current use

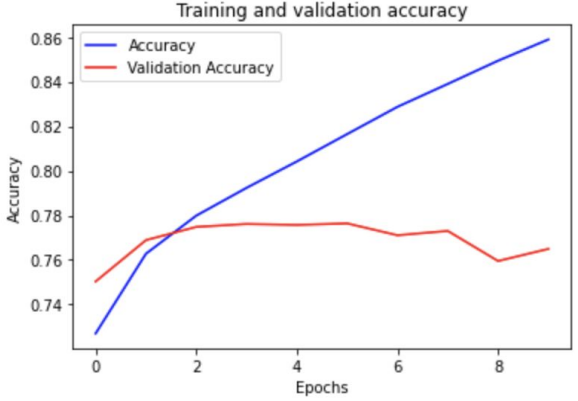
Sequence models

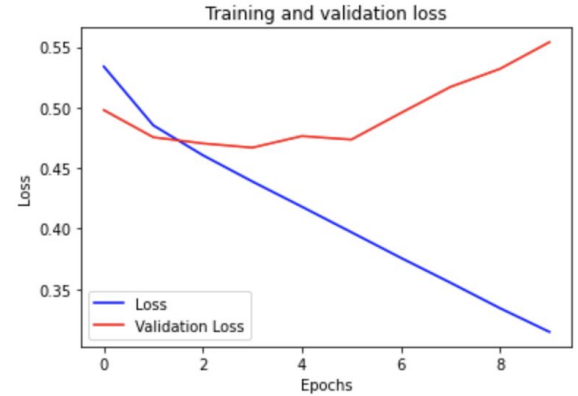
GlobalAveragePooling1D()



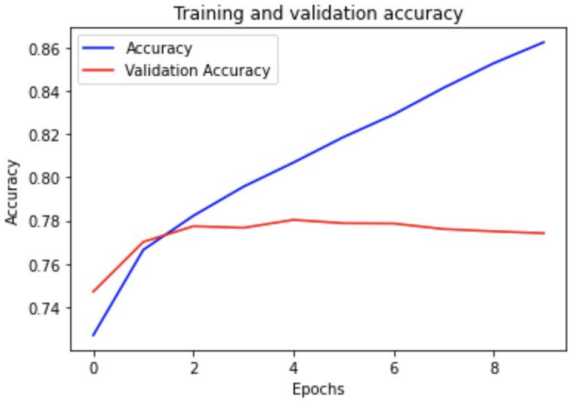


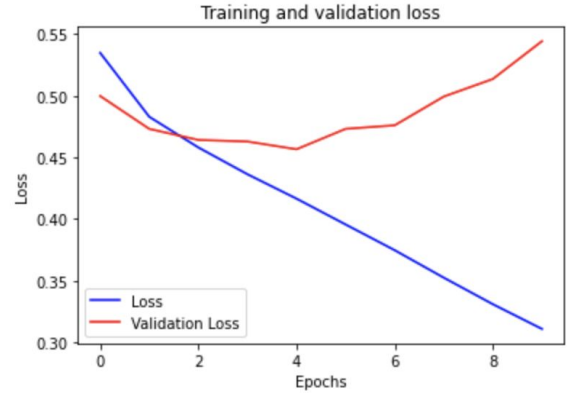
Bidirectional(LSTM(64))





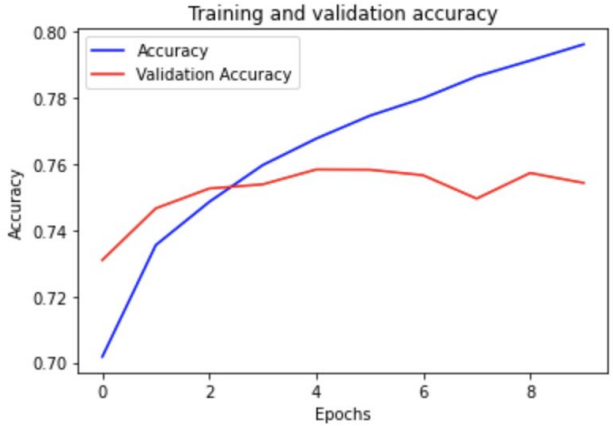
2 x Bidirectional(LSTM(64)), Dropout

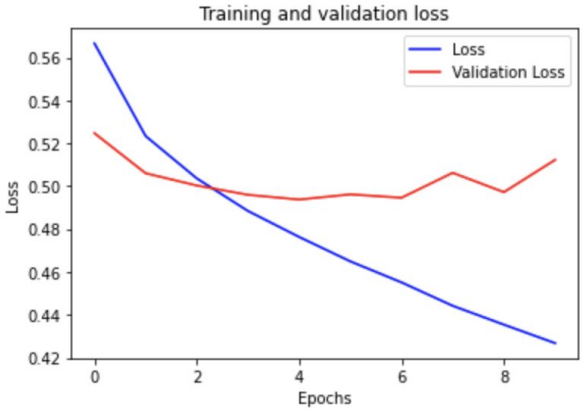




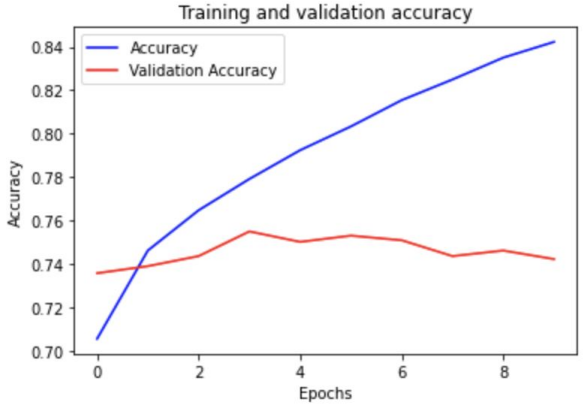
Sequence models

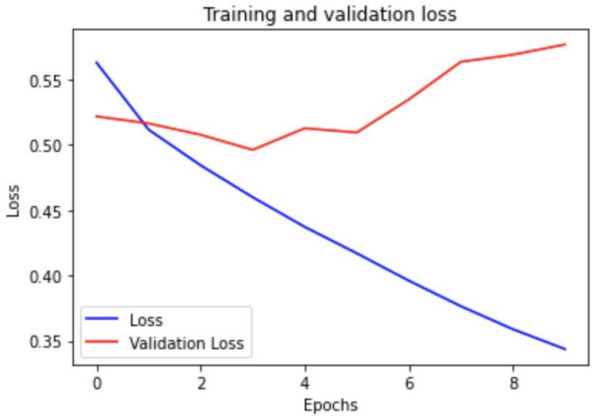
Conv1D, GlobalAveragePooling1D()



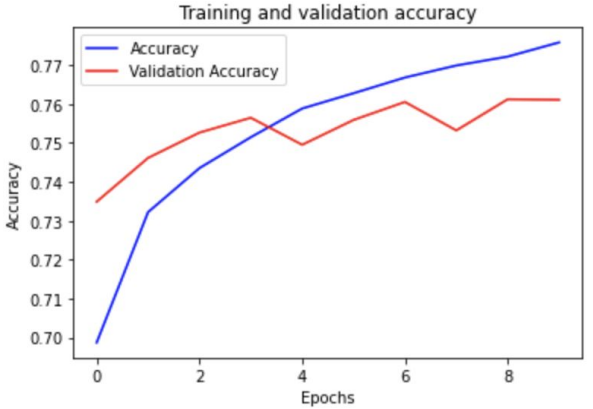


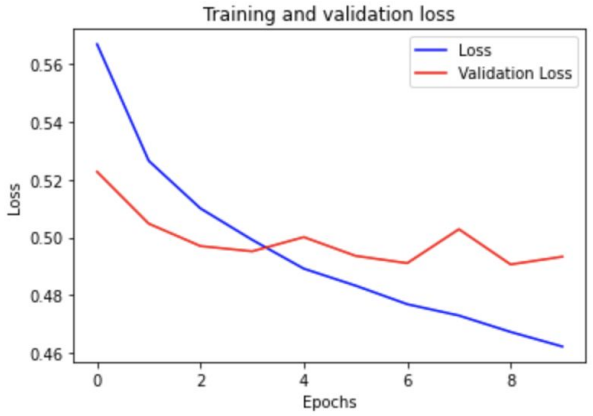
2 x Conv1D, GAP1D(), Dropout





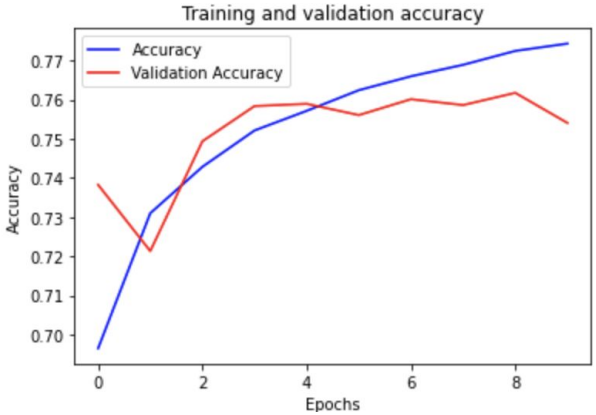
Dropout, Conv1D, MaxP1D(), LSTM(64)

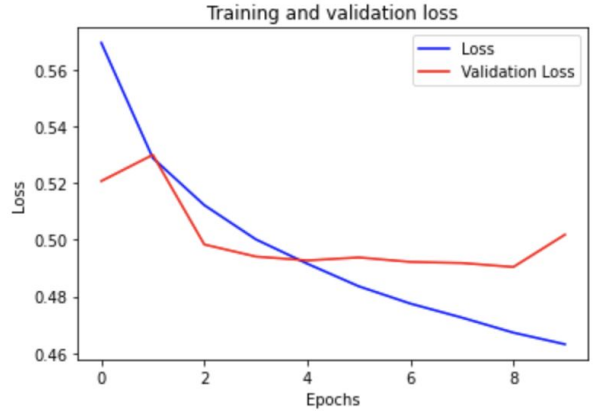




Sequence models

Dropout, Conv1D, MaxP1D(), 2 x Bidirectional(LSTM(64))





Let’s continue our NLP adventure 😀

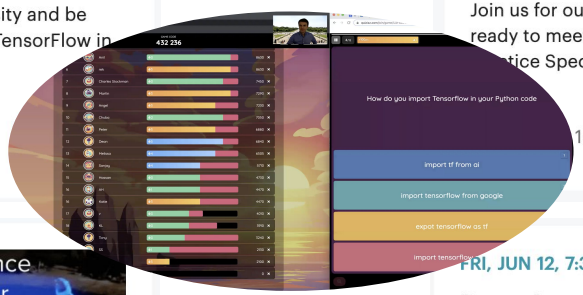
FRI, MAY 22, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 1,2 - Sentiment in text & Wor...

Online event

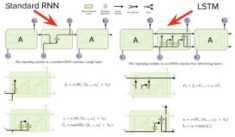
Join us for our 6th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

31 attendees



FRI, MAY 29, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 3 - Sequence models



Online event

Join us for our 7th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in tice Specialization available at:...

15 attendees

Attend

FRI, JUN 5, 7:30 PM EDT

TensorFlow in Practice - C3 Week 4 - Sequence models & literature +...

Online event

Join us for our 8th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

13 attendees

Attend

FRI, JUN 12, 7:30 PM EDT

Deep Learning Adventures - Happy Hour 🍹🍸🥂

Online event

Join us for a fun conversation 🍹🍸🥂 No slides or recording this time, just getting to know each other better and forge a stronger community 😀 We are taking a break from our TensorFlow in Practice Specialization available at:...

9 attendees

Attend

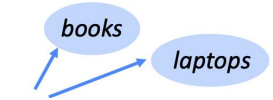
Source: https://www.meetup.com/Deep-Learning-Adventures

Course 3: Natural Language Processing in TensorFlow

Week 1: Sentiment in text
Week 2: Word Embeddings
Week 3: Sequence models
Week 4: Sequence models and literature

Sequence models and literature

- **Language Modeling** is the task of predicting what word comes next.



the students opened their

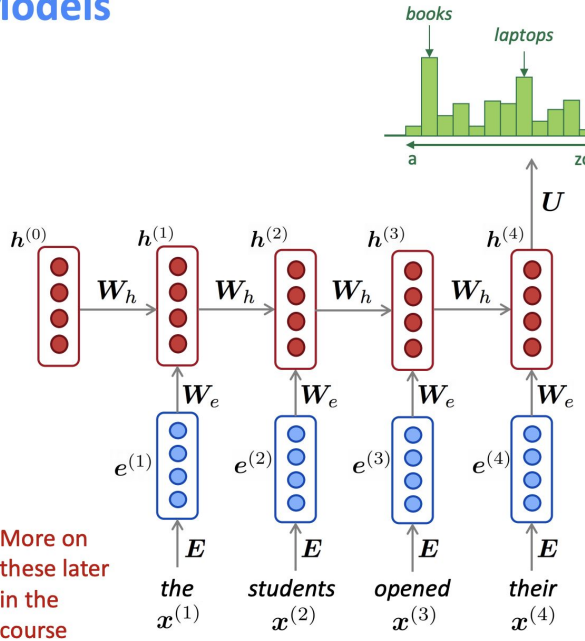
$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$

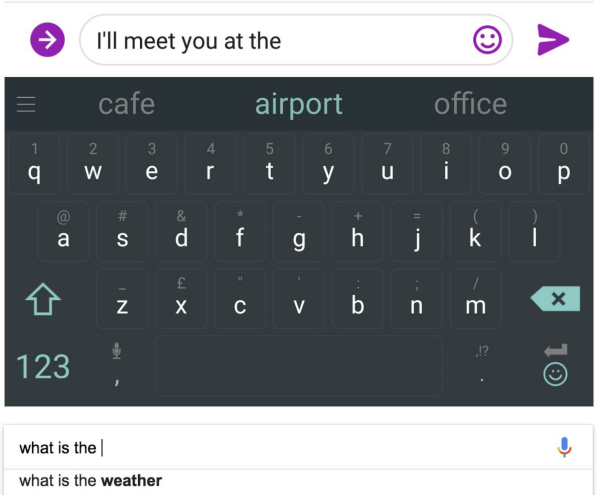
RNN Language Models

RNN **Advantages**:
- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:
- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course





Question: How to learn a Language Model?

Answer (pre- Deep Learning): learn an *n*-gram Language Model!

Definition: A *n*-gram is a chunk of *n* consecutive words.
- unigrams: "the", "students", "opened", "their"
- bigrams: "the students", "students opened", "opened their"
- trigrams: "the students opened", "students opened their"
- 4-grams: "the students opened their"

Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

Sequence models and literature

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

53

RNNs have greatly improved perplexity

n-gram model

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

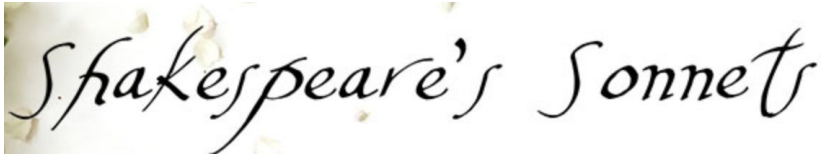
Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

Sequence models and literature



From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And, tender churl, mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

```
data = open('/tmp/sonnets.txt').read()

corpus = data.lower().split("\n")
print(corpus[:3])

tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
```

```
# create input sequences using list of tokens
input_sequences = []
for line in corpus:
  #print([line])
  #print(tokenizer.texts_to_sequences([line]))
  #print(tokenizer.texts_to_sequences([line])[0])
  #break
  token_list = tokenizer.texts_to_sequences([line])[0]
  for i in range(1, len(token_list)):
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)


# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
```

```
#https://keras.io/api/utils/python_utils/#to_categorical-function
label = ku.to_categorical(label, num_classes=total_words)
```

```
label[0]
```

```
array([0., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

```
len(label[0])
```

```
3211
```

```
total_words
```

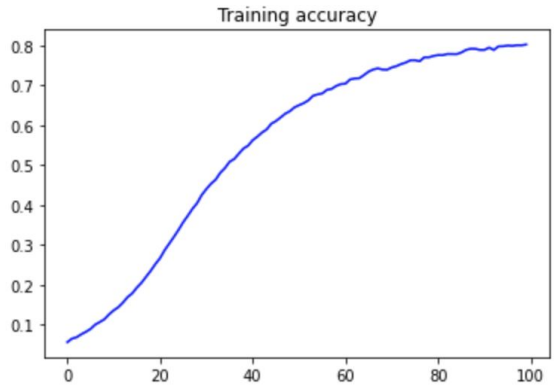
```
3211
```

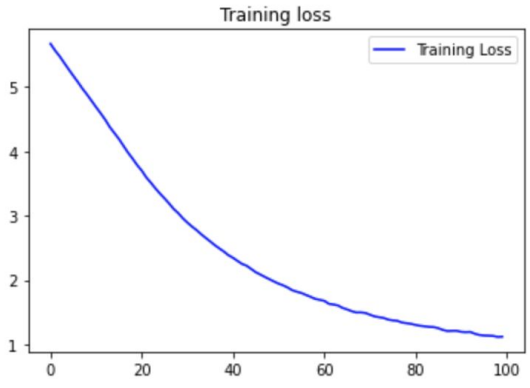
Source: http://www.shakespeares-sonnets.com/sonnet/1

Sequence models and literature


```
model = Sequential()
model.add(Embedding(input_dim=total_words, output_dim=100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150, return_sequences=True)))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dense(total_words/2, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))

my_optimizer = Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=my_optimizer, metrics=['accuracy'])
history = model.fit(predictors, label, epochs=100, verbose=1)
```

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And, tender churl, mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.





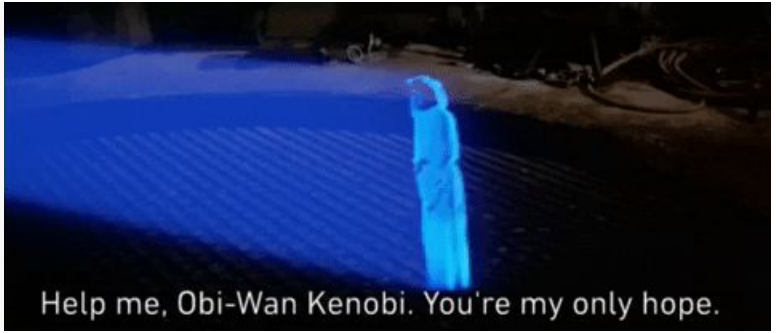

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 10, 100)           321100
_________________________________________________________________
bidirectional_6 (Bidirection (None, 10, 300)           301200
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 300)           0
_________________________________________________________________
lstm_7 (LSTM)                (None, 100)               160400
_________________________________________________________________
dense_4 (Dense)              (None, 1605)              162105
_________________________________________________________________
dense_5 (Dense)              (None, 3211)              5156866
=================================================================
Total params: 6,101,671
Trainable params: 6,101,671
Non-trainable params: 0
_________________________________________________________________
```

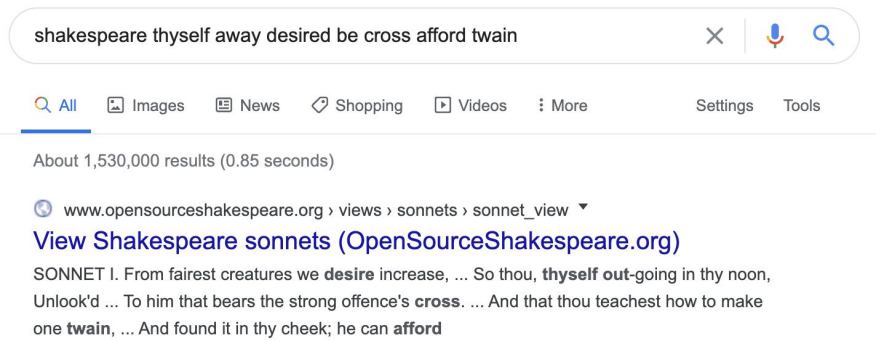
Source: http://www.shakespeares-sonnets.com/sonnet/1

Sequence models and literature

```
seed_text = "Help me Obi Wan Kenobi, you're my only hope"
next_words = 100

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      break
  seed_text += " " + output_word
print(seed_text)
```

```
tf.keras.preprocessing.sequence.pad_sequences(
    sequences, maxlen=None, dtype='int32', padding='pre', truncating='pre',
    value=0.0
)
```

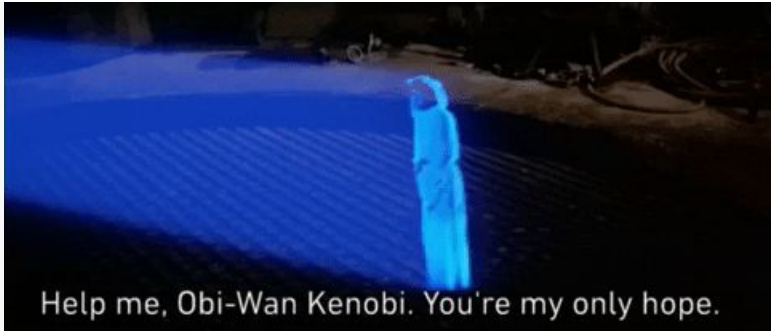


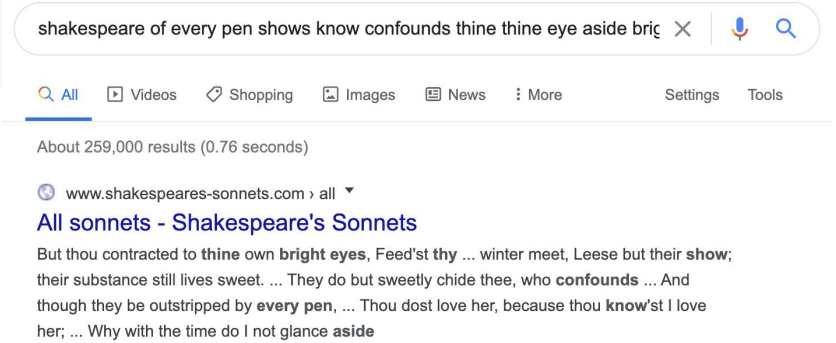


Help me Obi Wan Kenobi, you're my only hope thyself away desired be cross afford twain date of men ' doth groan ' be free free live away away your days here brought of one one more must grow of free ride hits green thee burn pleasure pleasure me of days much brought of pleasure pleasure of pleasure green die burn'd hits dwells beard ward fall by days another rolling ' doth ride ride ride ride ' her view free grew to ride cherish plight grow thus days did tell away away night exchanged mad so meant free his senses can alive pleasure ride ride light decease translate must

Sequence models and literature

```
seed_text = "Help me Obi Wan Kenobi, you're my only hope"
next_words = 100

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      break
  seed_text += " " + output_word
print(seed_text)
```





Help me Obi Wan Kenobi, you're my only hope of every pen shows know confounds thine thine eye aside bright than gently cheeks express'd place in mind keep something new hate of die much date cheeks day was me away the rhyme ' may be bevel disdain erred light light stand bright ' write me flowers so look me doth grow ill old old glory still say thee but now come away a gainer his part torn torn away away ill young new date lies be express'd rare rare ill ill near feast so 'will ' hast me well seen live up memory prove keep invention spent kind check

Sequence models and literature

TensorFlow > Learn > TensorFlow Core > Tutorials

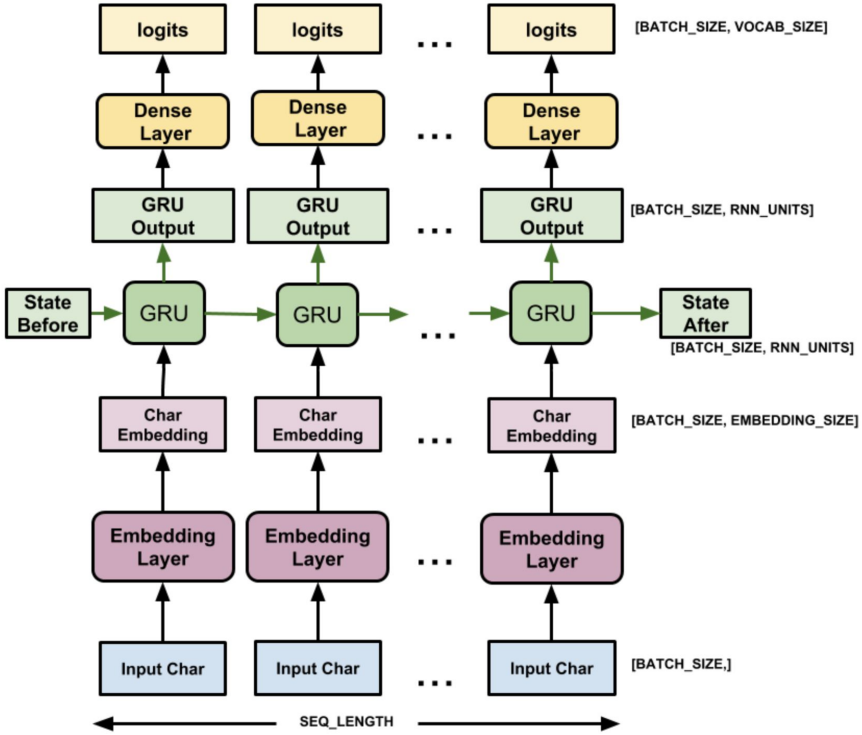
Text generation with an RNN

Run in Google Colab | View source on GitHub | Download notebook

This tutorial demonstrates how to generate text using a character-based RNN. We will work with a dataset of Shakespeare's writing from Andrej Karpathy's The Unreasonable Effectiveness of Recurrent Neural Networks. Given a sequence of characters from this data ("Shakespear"), train a model to predict the next character in the sequence ("e"). Longer sequences of text can be generated by calling the model repeatedly.

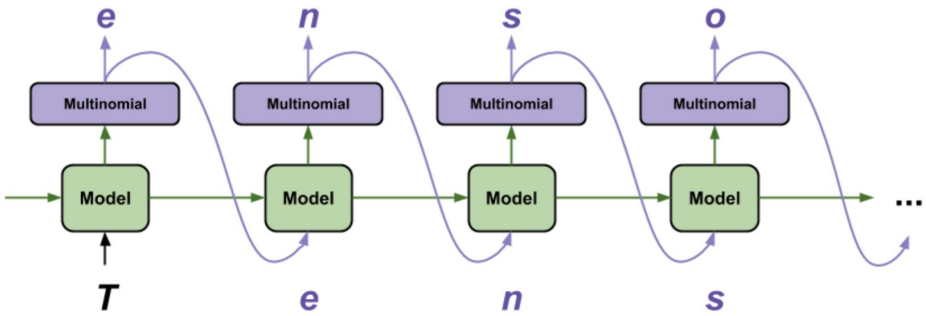
```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
  model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              batch_input_shape=[batch_size, None]),
    tf.keras.layers.GRU(rnn_units,
                        return_sequences=True,
                        stateful=True,
                        recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(vocab_size)
  ])
  return model
```

For each character the model looks up the embedding, runs the GRU one timestep with the embedding as input, and applies the dense layer to generate logits predicting the log-likelihood of the next character:



Source: https://www.tensorflow.org/tutorials/text/text_generation

Sequence models and literature



The prediction loop

The following code block generates the text:

- It Starts by choosing a start string, initializing the RNN state and setting the number of characters to generate.
- Get the prediction distribution of the next character using the start string and the RNN state.
- Then, use a categorical distribution to calculate the index of the predicted character. Use this predicted character as our next input to the model.
- The RNN state returned by the model is fed back into the model so that it now has more context, instead than only one character. After predicting the next character, the modified RNN states are again fed back into the model, which is how it learns as it gets more context from the previously predicted characters.

```
print(generate_text(model, start_string=u"ROMEO: "))
```

Source: https://www.tensorflow.org/tutorials/text/text_generation

```
ROMEO: I am unplume, shalt to the
Francies would women cluncime against it.

MENENIUS:
Pemprescorce that you shall not be thus; let's in
justices and by
have thoughts, to test the stars as great
As open lack gawned raged
Duke of Northumberland, this manner of his prince.

MARCIUS:
How do thou wast forced;
The endempily east enought than whence, or, bear
headed me aple, to-morrow why I rue,
My own brothers on't, but stins abbooon of
so sours; or ghinf purnicy in base as as
Two kings at my heart?

DUKE OF AUMERLE:
Which dost thou be in this person? peace: but whoreson
hat a not flee
Whose honour and athe stampet's presence in sorrow is
there?

BIONDELLO:
What can you gone, and suck'd your prunishen.
They, we beseech you, sister, whilst Margaret
pleasant, gentle speed?
About! following proclaim the end,
And whom unto his rooate passion,
Go apperial court.
Now, sir, but 'tain for Richard, whom is that all.

PETRUCHIO:
Now, there, i' for the !
With welt repared on Rome, to
     Grimy, sad I fought. Whe
```

Sequence models applicatio

Character level language model - Dinos

Welcome to Dinosaurus Island! 65 million years ago, dinosaurs
task. Leading biology researchers are creating new breeds of d
these dinosaurs. If a dinosaur does not like its name, it might g

```
Iteration: 30000, Loss: 22.532136

Oryxisaurus
Loca
Lustreoenliusaures
Pacaisig
Xussancophydps
Eg
Stochisaurus


Iteration: 32000, Loss: 22.298402

Nivusaurus
Knacalosaurus
Lustrasaurus
Ndaairus
Xuspandon
Eg
Trohaliomumsaurus


Iteration: 34000, Loss: 22.429814

Onyxmigomalex
Kica
Lutrolomia
Ola
Xosalonghauglosaurus
Ehadon
Stlerdor
```

1 - Problem statement

You would like to create a jazz music piece specially for a friend's birthday. However, you don't know any instruments or music composition. Fortunately, you know deep learning and will solve this problem using an LSTM netwok.

You will train a network to generate novel jazz solos in a style representative of a body of performed work.

Source: https://www.youtube.com/watch?v=ggQ1y1UHOvc

Sequence models applications

Gen

Let's

- Yo
  in
- RN

Gene

Let's ha

- You
  in th
- RNN

Generating text

Let's have some fun!

- You can train a RNN
  in that style.
- RNN-LM trained on

The
peop
and
be ai

"Sor
they

"No i
last
com
spid

```
      Title: CHOCOLATE RANCH
Categories: Game, Casseroles
      Yield: 6 Servings

      2 tb Parmesan cheese --
      1 c  Coconut milk
      3    Eggs, beaten

   Place each pasta over layer
   until firm. Serve hot in bo

   Combine the cheese and salt
   and stir in the chocolate a
```

Generating text with a RNN Language Model

Let's have some fun!

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:

Ghasty Pink 231 137 165	Sand Dan 201 172 143
Power Gray 151 124 112	Grade Bat 48 94 83
Navel Tan 199 173 140	Light Of Blast 175 150 147
Bock Coe White 221 215 236	Grass Bat 176 99 108
Horble Gray 178 181 196	Sindis Poop 204 205 194
Homestar Brown 133 104 85	Dope 219 209 179
Snader Brown 144 106 74	Testing 156 101 106
Golder Craam 237 217 177	Stoner Blue 152 165 159
Hurky White 232 223 215	Burble Simp 226 181 132
Burf Pink 223 173 179	Stanky Bean 197 162 171
Rose Hork 230 215 198	Turdly 190 164 116

This is an example of a character-level RNN-LM (predicts what character comes next)

49 50 51 52

Source: http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

Source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture06-rnnlm.pdf

Sequence models applications

Andrej Karpathy blog

About Hacker's guide to Neural Networks

The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

There's something magical about Recurrent Neural Networks (RNNs). I still remember when I trained my first recurrent network for Image Captioning. Within a few dozen minutes of training my first baby model (with rather arbitrarily-chosen hyperparameters) started to generate very nice looking descriptions of images that were on the edge of making sense. Sometimes the ratio of how simple your model is to the quality of the results you get out of it blows past your expectations, and this was one of those times. What made this result so shocking at the time was that the common wisdom was that RNNs were supposed to be difficult to train (with more experience I've in fact reached the opposite conclusion). Fast forward about a year: I'm training RNNs all the time and I've witnessed their power and robustness many times, and yet their magical outputs still find ways of amusing me. This post is about sharing some of that magic with you.

> *We'll train RNNs to generate text character by character and ponder the question "how is that even possible?"*

Paul Graham generator

Shakespeare

Wikipedia

Algebraic Geometry (Latex)

Linux Source Code

Generating Baby Names

Visualizing the predictions and the "neuron" firings in the RNN

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

Sequence models and computer vision

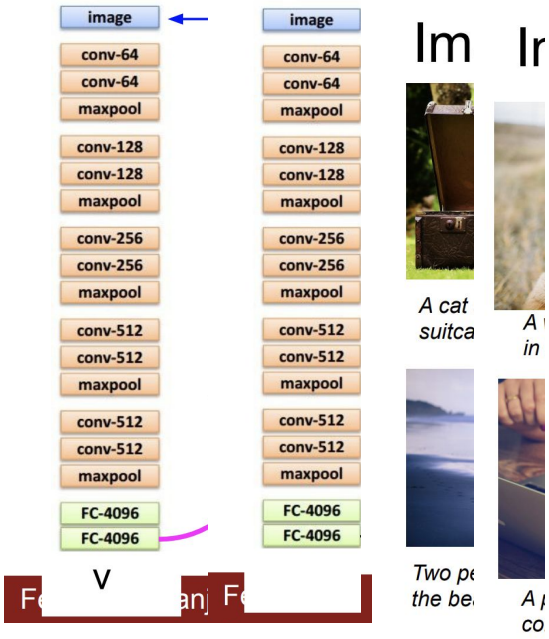
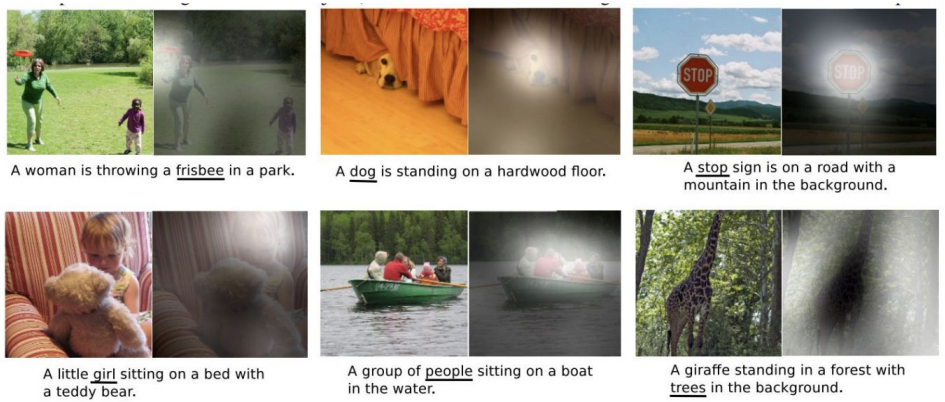


Image Captioning with Attention



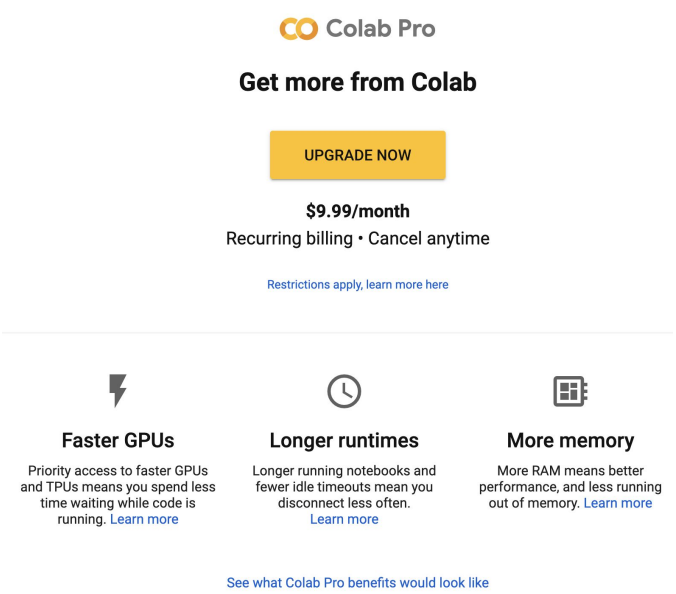
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

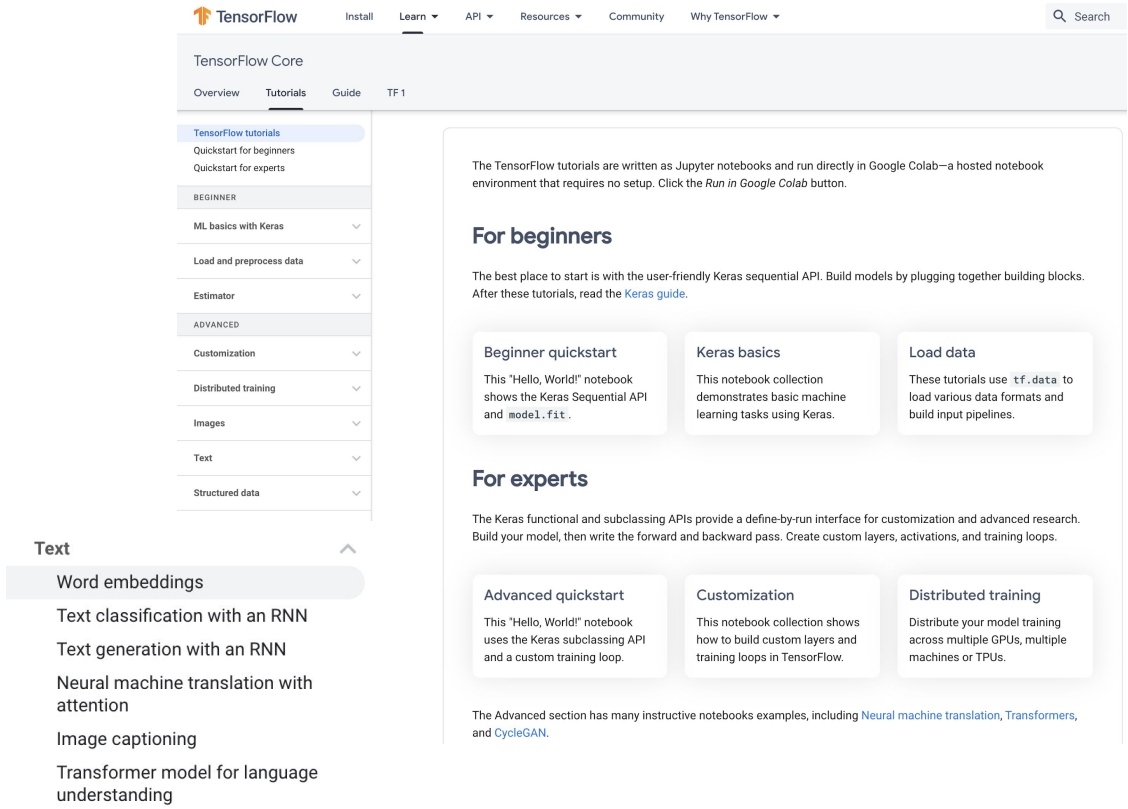
Source: http://cs231n.stanford.edu/slides/2020/lecture_10.pdf

Course 3: Natural Language Processing in TensorFlow

Week 1: Sentiment in text
Week 2: Word Embeddings
Week 3: Sequence models
Week 4: Sequence models and literature

Check out these resources





Source:
https://colab.research.google.com/signup
https://www.tensorflow.org/tutorials/

Check out these AI for Healthcare resources

deeplearning.ai **AI for Medicine Specialization**

Enroll for Free
Starts May 22

About | How It Works | Courses | Instructors | Enrollment Options | FAQ

There are 3 Courses in this Specialization

COURSE 1

AI for Medical Diagnosis

4.6 410 ratings • 102 reviews

AI is transforming the practice of medicine. It's helping doctors diagnose patients more accurately, make predictions about patients' future health, and recommend better treatments. As an AI practitioner, you have the opportunity to join in this transformation of modern medicine. If you're already familiar with some of the math and coding behind AI

SHOW ALL

COURSE 2

AI for Medical Prognosis

4.6 113 ratings • 27 reviews

AI is transforming the practice of medicine. It's helping doctors diagnose patients more accurately, make predictions about patients' future health, and recommend better treatments. This Specialization will give you practical experience in applying machine learning to concrete problems in medicine.

SHOW ALL

COURSE 3

AI For Medical Treatment

AI is transforming the practice of medicine. It's helping doctors diagnose patients more accurately, make predictions about patients' future health, and recommend better treatments. This Specialization will give you practical experience in applying machine learning to concrete problems in medicine.

SHOW ALL

SYLLABUS

AI for Healthcare

Learn to build, evaluate, and integrate predictive models that have the power to transform patient outcomes. Begin by classifying and segmenting 2D and 3D medical images to augment diagnosis and then move on to modeling patient outcomes with electronic health records to optimize clinical trial testing decisions. Finally, build an algorithm that uses data collected from wearable devices to estimate the wearer's pulse rate in the presence of motion.

PREREQUISITE KNOWLEDGE

Intermediate Python, and Experience with Machine Learning See detailed requirements.

HIDE DETAILS

Applying AI to 2D Medical Imaging Data

Learn the fundamental skills needed to work with 2D medical imaging data and how to use AI to derive clinically-relevant insights from data gathered via different types of 2D medical imaging such as x-ray, mammography, and digital pathology. Extract 2D images from DICOM files and apply the appropriate tools to perform exploratory data analysis on them. Build different AI models for different clinical scenarios that involve 2D images and learn how to position AI tools for regulatory approval.

PNEUMONIA DETECTION FROM CHEST X-RAYS

Applying AI to 3D Medical Imaging Data

Learn the fundamental skills needed to work with 3D medical imaging datasets and frame insights derived from the data in a clinically relevant context. Understand how these images are acquired, stored in clinical archives, and subsequently read and analyzed. Discover how clinicians use 3D medical images in practice and where AI holds most potential in their work with these images. Design and apply machine learning algorithms to solve the challenging problems in 3D medical imaging and how to integrate the algorithms into the clinical workflow.

HIPPOCAMPUS VOLUME QUANTIFICATION FOR ALZHEIMER'S PROGRESSION

Applying AI to EHR Data

Learn the fundamental skills to work with EHR data and build and evaluate compliant, interpretable models. You will cover EHR data privacy and security standards, how to analyze EHR data and avoid common challenges, and cover key industry code sets. By the end of the course, you will have the skills to analyze an EHR dataset, transform it to the right level, build powerful features with TensorFlow, and model the uncertainty and bias with TensorFlow Probability and Aequitas.

PATIENT SELECTION FOR DIABETES DRUG TESTING

Applying AI to Wearable Device Data

Learn how to build algorithms that process the data collected by wearable devices and surface insights about the wearer's health. Cover the sensors and signal processing foundation that are critical for success in this domain, including IMU, PPG, and ECG that are common to most wearable devices, and learn how to build three algorithms from real-world sensor data.

MOTION COMPENSATED PULSE RATE ESTIMATION

Source:
https://www.coursera.org/specializations/ai-for-medicine
https://www.udacity.com/course/ai-for-healthcare-nanodegree--nd320

Check out this certification and books

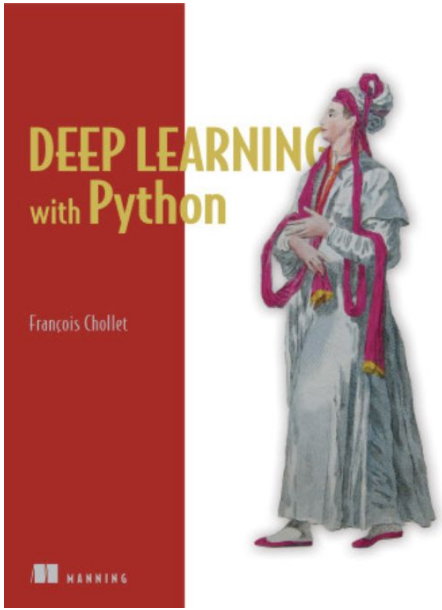
TensorFlow Core

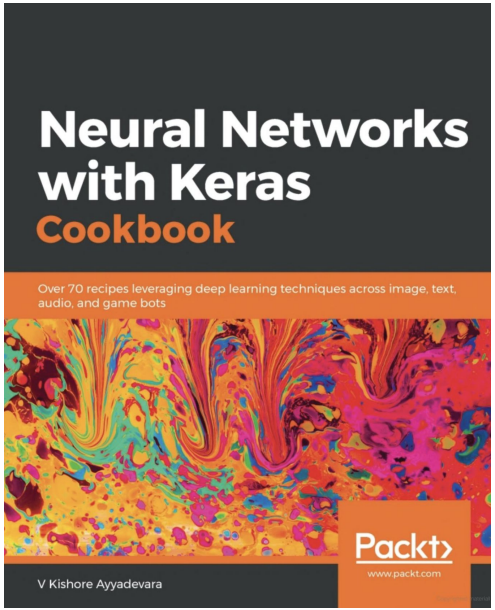
Introducing the TensorFlow Developer Certificate!

March 12, 2020

Posted by Alina Shinkarsky, on behalf of the TensorFlow Team

In the AI world today, more and more companies are looking to hire machine learning talent, and simultaneously, an increasing number of students and developers are looking for ways to gain and showcase their ML knowledge with formal recognition. In addition to the courses and learning resources available online, we want to help developers showcase their ML proficiency and help companies hire ML developers to solve challenging problems.





Source:
https://blog.tensorflow.org/2020/03/introducing-tensorflow-developer-certificate.html?m=1
https://www.manning.com/books/deep-learning-with-python
https://books.google.com/books?id=5guLDwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
https://github.com/PacktPublishing/Neural-Networks-with-Keras-Cookbook/tree/master/Chapter11

Let’s continue our NLP adventure 😀

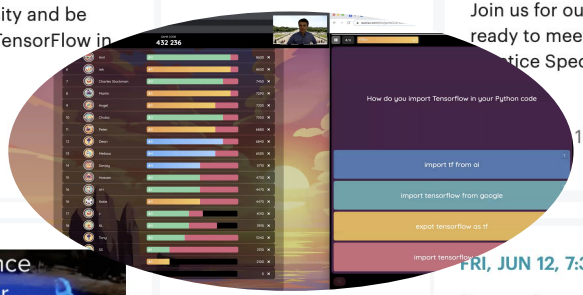
FRI, MAY 22, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 1,2 - Sentiment in text & Wor...

Online event

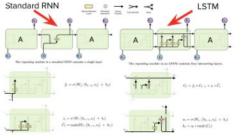
Join us for our 6th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

31 attendees



FRI, MAY 29, 7:30 PM EDT

TensorFlow in Practice - Course 3 Week 3 - Sequence models



Online event

Join us for our 7th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in tice Specialization available at:...

15 attendees

Attend

FRI, JUN 5, 7:30 PM EDT

TensorFlow in Practice - C3 Week 4 - Sequence models & literature +...

Online event

Join us for our 8th adventure in Deep Learning! Just bring your curiosity and be ready to meet our growing community 😀 We are taking Course 3 of TensorFlow in Practice Specialization available at:...

13 attendees

Attend

FRI, JUN 12, 7:30 PM EDT

Deep Learning Adventures - Happy Hour 🍹🍸🥂

Online event

Join us for a fun conversation 🍹🍸🥂 No slides or recording this time, just getting to know each other better and forge a stronger community 😀 We are taking a break from our TensorFlow in Practice Specialization available at:...

9 attendees

Attend

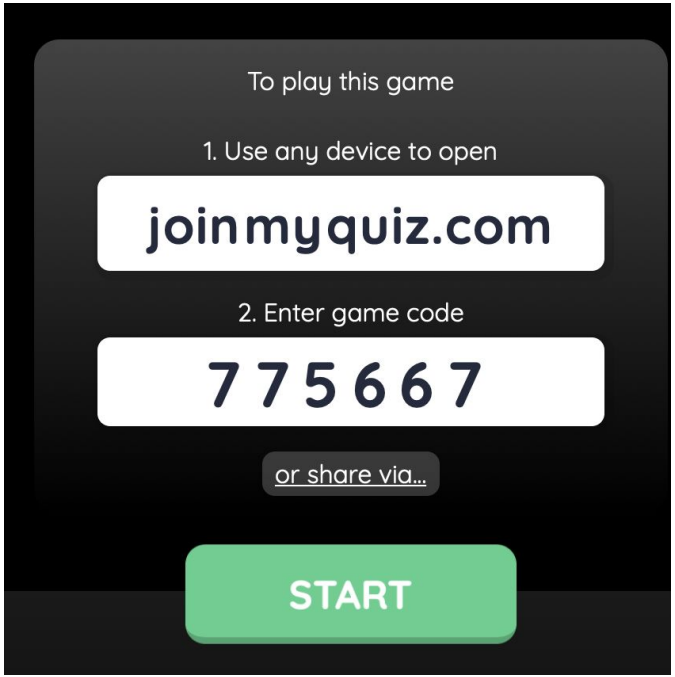
Source: https://www.meetup.com/Deep-Learning-Adventures

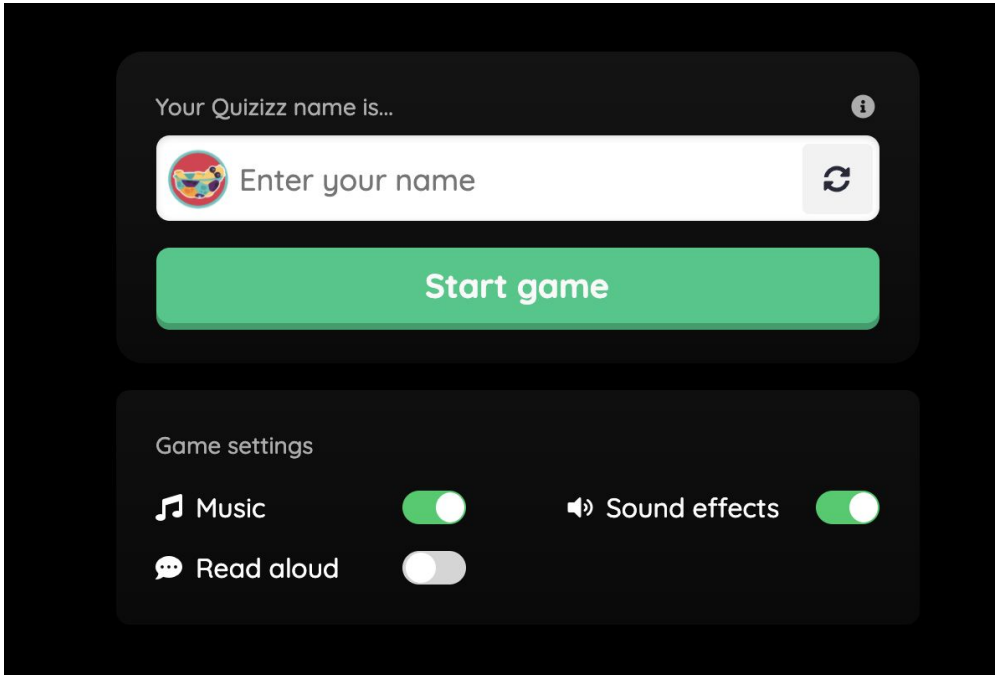
Time for a fun game 😀🎉

Deep Learning Adventures - Quiz 2

Professional Development · 3 times

Other

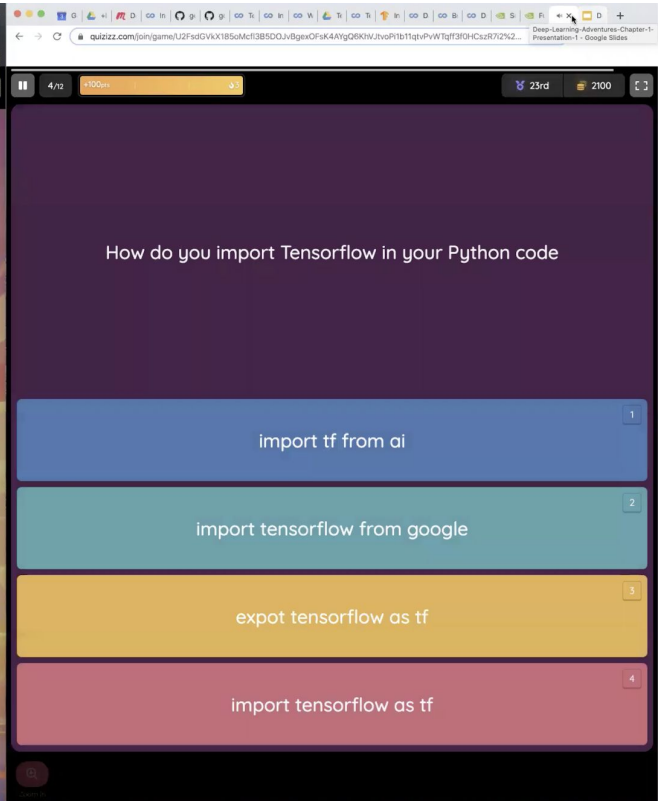




Practice here or use Flashcards:
https://quizizz.com/join/quiz/5e90e8e631fb32001f63510b/start?from=soloLinkShare&referrer=5d921444d0fa99001a135336

Time for a fun game 😀🎉

Practice here or use Flashcards:
quizizz.com/join/quiz/5e87bdbc07fa7f001b120404/start?from=soloLinkShare&referrer=5d921444d0fa99001a135336



Questions

Discussion

Standard notations for Deep Learning

This document has the purpose of discussing a new standard for deep learning mathematical notations.

1 Neural Networks Notations.

General comments:

· superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

·m : number of examples in the dataset

·n_x : input size

·n_y : output size (or number of classes)

·$n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h{}^{[\text{number of layers } +1]}$.

·L : number of layers in the network.

Objects:

·$X \in \mathbb{R}^{n_x \times m}$ is the input matrix

·$x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th}example represented as a column vector

·$Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

·$y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

·$W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

·$b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

·$\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

Common forward propagation equation examples:

$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1)$ where $g^{[l]}$ denotes the l^{th} layer activation function

$\hat{y}^{(i)} = softmax(W_h h + b_2)$

· General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

· $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Examples of cost function:

· $J_{CE}(\hat{y}, y) = -\sum_{i=0}^{m} y^{(i)} \log \hat{y}^{(i)}$

· $J_1(\hat{y}, y) = \sum_{i=0}^{m} \mid y^{(i)} - \hat{y}^{(i)} \mid$

2 Deep Learning representations

For representations:

· nodes represent inputs, activations or outputs

· edges represent weights or biases

Here are several examples of Standard deep learning representations

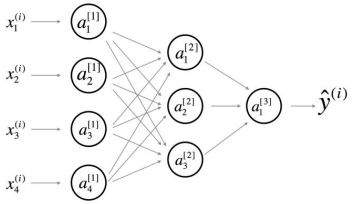


Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

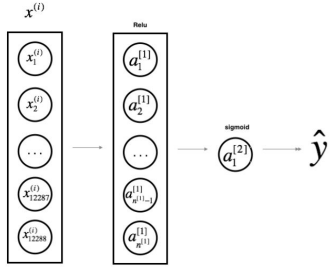


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.