Hands-On Machine Learning

NLP with RNNs & Attention- Vibhu Sapra

RNN & Attention

Overview

- Generate Shakespearean text - RNN
    - Stateless vs Stateful RNN
- Sentiment Analysis - IMDB
- Machine Translation
    - RNN / Bidirectional RNN / Beam Search
- Attention
- Transformers

Shakespearean Text generation

Character RNN

Shakespeare Text

Dataset

- Long text file - not really processed
  - 110k *characters* long
  - 90% training / 5% test / 5% validation
- 39 unique characters
  - Character RNN
- 1 hot encoded
- Sequential Dataset
  - Avoid overlap in train & test

Shakespeare Text

Short Samples

```
MENENIUS:
He loves your people
But tie him not to be their bedfellow.
Worthy Cominius, speak.
Nay, keep your place.

First Senator:
Sit, Coriolanus; never shame to hear
What you have nobly done.

CORIOLANUS:
Your horror's pardon:
I had rather have my wounds to heal again
Than hear say how I got them.

BRUTUS:
Sir, I hope
My words disbench'd you not.

CORIOLANUS:
No, sir: yet oft,
When blows have made me stay, I fled from words.
You soothed not, therefore hurt not: but
your people,
I love them as they weigh.
```

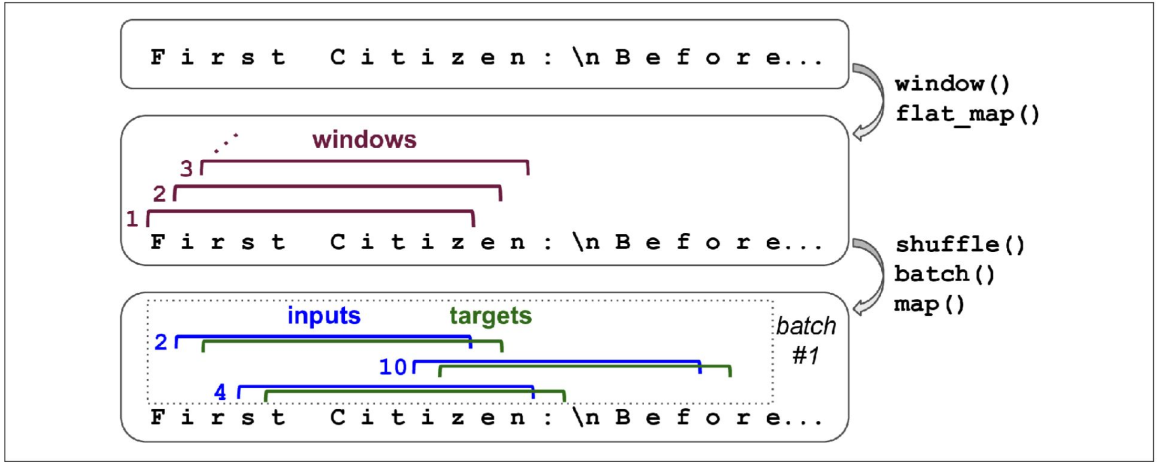
Longer Samples

```
VOLUMNIA:
Because that now it lies you on to speak
To the people; not by your own instruction,
Nor by the matter which your heart prompts you,
But with such words that are but rooted in
Your tongue, though but bastards and syllables
Of no allowance to your bosom's truth.
Now, this no more dishonours you at all
Than to take in a town with gentle words,
Which else would put you to your fortune and
The hazard of much blood.
I would dissemble with my nature where
My fortunes and my friends at stake required
I should do so in honour: I am in this,
Your wife, your son, these senators, the nobles;
And you will rather show our general louts
How you can frown than spend a fawn upon 'em,
For the inheritance of their loves and safeguard
Of what that want might ruin.

MENENIUS:
Noble lady!
Come, go with us; speak fair: you may salve so,
Not what is dangerous present, but the loss
Of what is past.
```

Text Generation

Char-RNN Model

- Goal - text generation model
  - Predict the next character

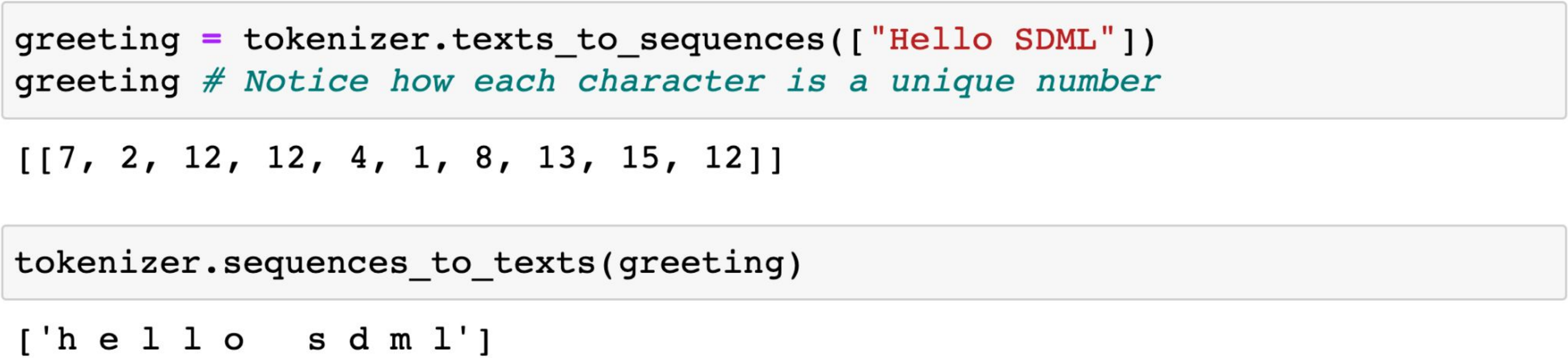


First Citizen:\nBefore...
window()
flat_map()
windows
3
2
1
First Citizen:\nBefore...
shuffle()
batch()
map()
inputs
targets
batch #1
2
10
4
First Citizen:\nBefore...

Text Generation

Char-RNN Model

- Goal - text generation model
- Need a tokenizer
  - Character level, not word level tokenizer
- Input previous 100 characters
  - Predict the next character
- Simple RNN model
  - 2 GRU layers
  - Softmax output over the 39 unique characters

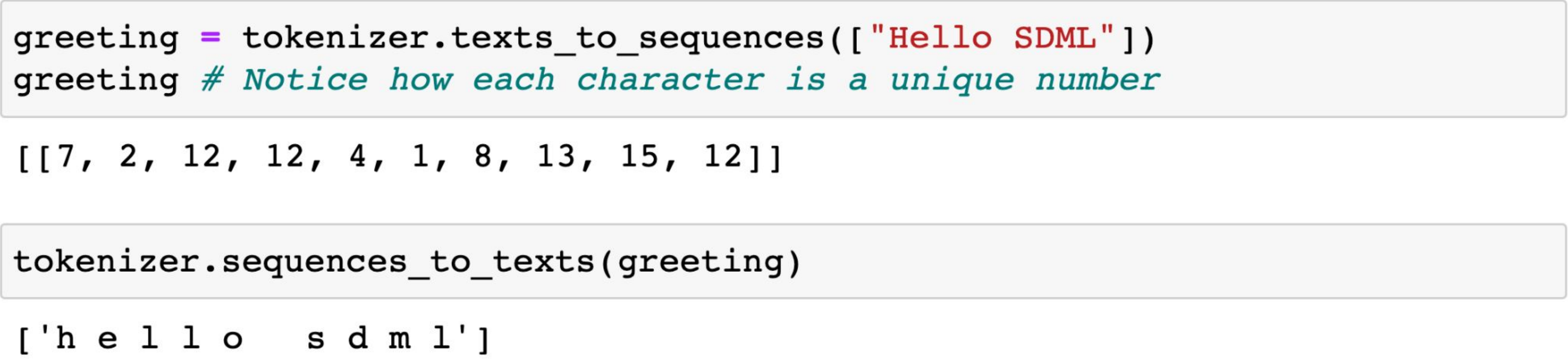


```
greeting = tokenizer.texts_to_sequences(["Hello SDML"])
greeting # Notice how each character is a unique number
```

```
[[7, 2, 12, 12, 4, 1, 8, 13, 15, 12]]
```

```
tokenizer.sequences_to_texts(greeting)
```

```
['h e l l o   s d m l']
```

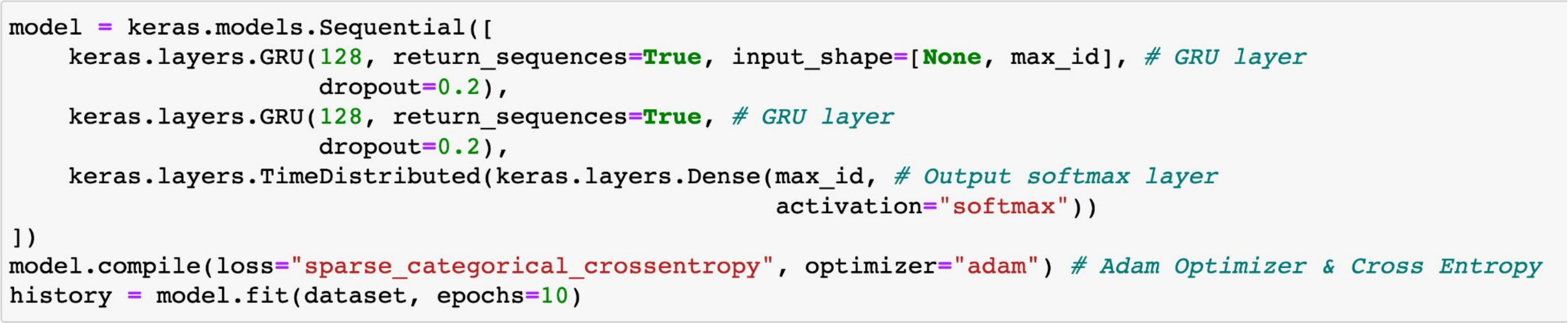


```
greeting = tokenizer.texts_to_sequences(["Hello SDML"])
greeting # Notice how each character is a unique number
```

```
[[7, 2, 12, 12, 4, 1, 8, 13, 15, 12]]
```

```
tokenizer.sequences_to_texts(greeting)
```

```
['h e l l o   s d m l']
```

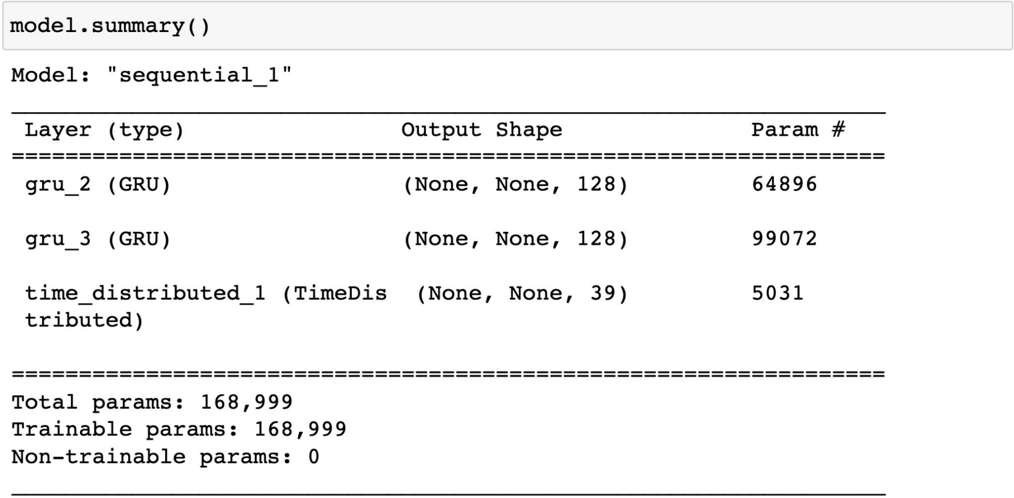


```
model = keras.models.Sequential([
    keras.layers.GRU(128, return_sequences=True, input_shape=[None, max_id], # GRU layer
                     dropout=0.2),
    keras.layers.GRU(128, return_sequences=True, # GRU layer
                     dropout=0.2),
    keras.layers.TimeDistributed(keras.layers.Dense(max_id, # Output softmax layer
                                                    activation="softmax"))
])
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam") # Adam Optimizer & Cross Entropy
history = model.fit(dataset, epochs=10)
```

Text Generation

Stateless RNN

- Hidden state is basically all 0s and useless
  - Simple model to start



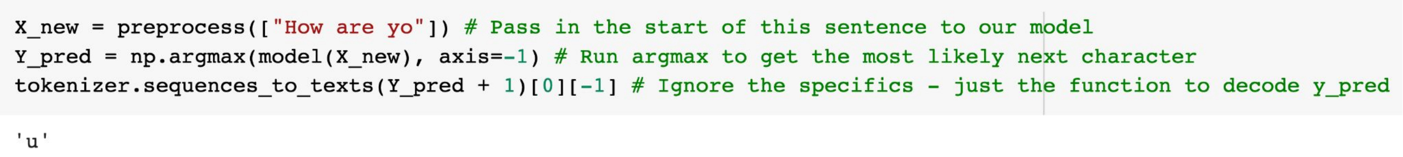
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_2 (GRU)                 (None, None, 128)         64896

 gru_3 (GRU)                 (None, None, 128)         99072

 time_distributed_1 (TimeDis  (None, None, 39)         5031
 tributed)

=================================================================
Total params: 168,999
Trainable params: 168,999
Non-trainable params: 0
_________________________________________________________________
```

Text Generation

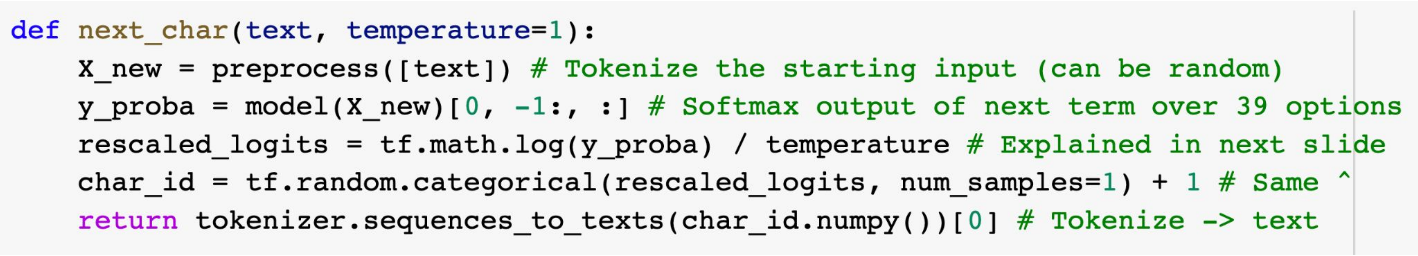
Using the Model

- Predicting one character is useless
- Write Fn() to repeatedly get a new character and input new string
  - Must explicitly give desired output length



```
X_new = preprocess(["How are yo"]) # Pass in the start of this sentence to our model
Y_pred = np.argmax(model(X_new), axis=-1) # Run argmax to get the most likely next character
tokenizer.sequences_to_texts(Y_pred + 1)[0][-1] # Ignore the specifics - just the function to decode y_pred
```

```
'u'
```

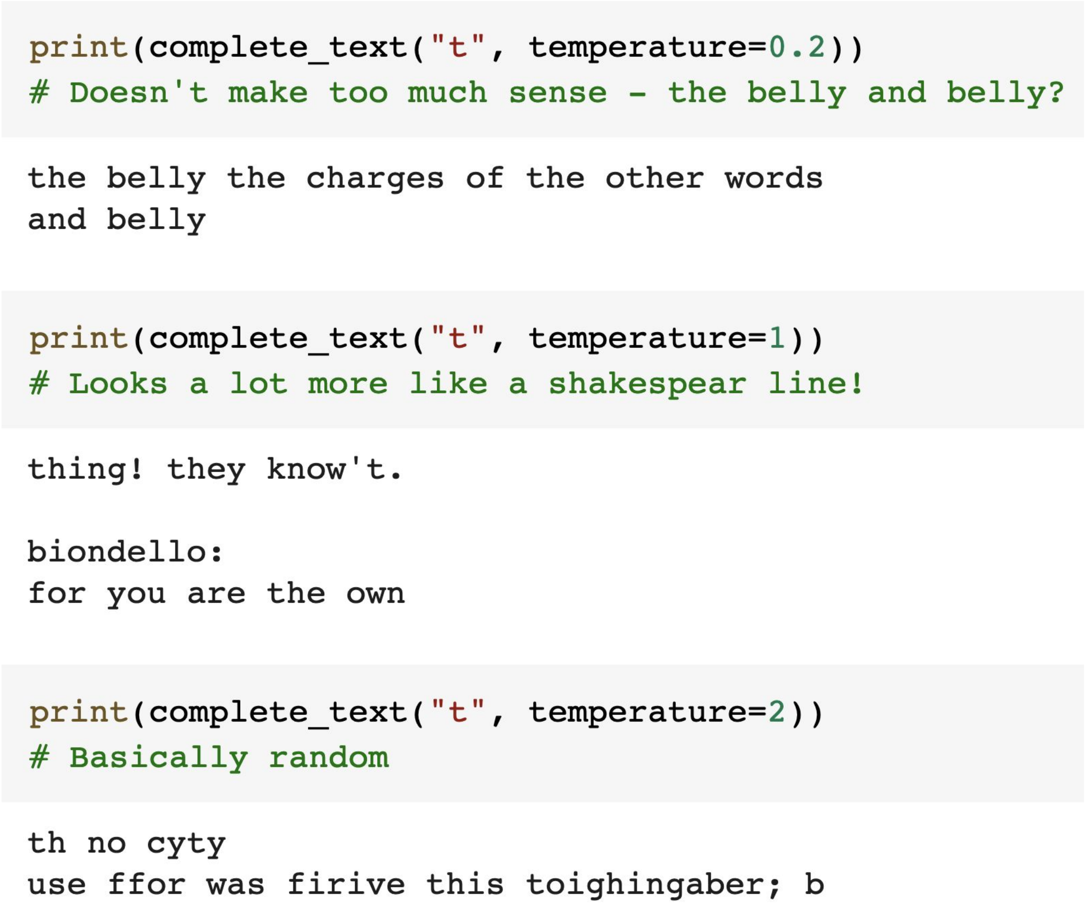


```
def next_char(text, temperature=1):
    X_new = preprocess([text]) # Tokenize the starting input (can be random)
    y_proba = model(X_new)[0, -1:, :] # Softmax output of next term over 39 options
    rescaled_logits = tf.math.log(y_proba) / temperature # Explained in next slide
    char_id = tf.random.categorical(rescaled_logits, num_samples=1) + 1 # Same ^
    return tokenizer.sequences_to_texts(char_id.numpy())[0] # Tokenize -> text
```

Text Generation

Temperature

- “rescaled_logits = tf.math.log(y_proba) / **temperature**”
- Models often get confused and predict the same letter / phrase repeatedly
- Temperature changes the output distribution / diversity in text
- High temperature - all output characters have equal prob - basically random sampling
- Low / 0 temperature - favor high probability characters
- Temperature of 1 - follow the softmax



```
print(complete_text("t", temperature=0.2))
# Doesn't make too much sense - the belly and belly?
```

```
the belly the charges of the other words
and belly
```

```
print(complete_text("t", temperature=1))
# Looks a lot more like a shakespear line!
```

```
thing! they know't.

biondello:
for you are the own
```

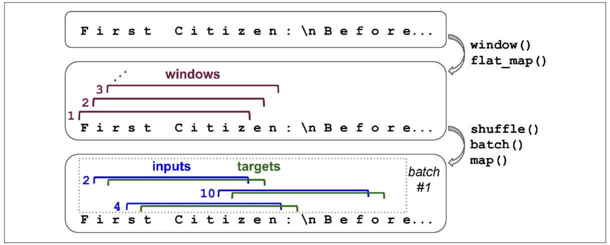
```
print(complete_text("t", temperature=2))
# Basically random
```

```
th no cyty
use ffor was firive this toighingaber; b
```

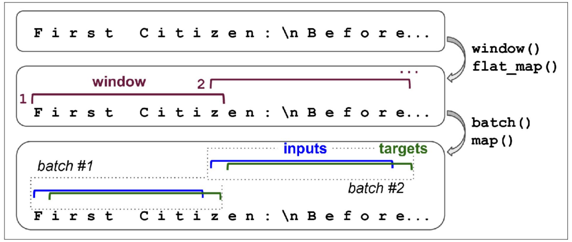
Text Generation

Stateful RNN

- Preserve hidden state
  - Learn long term patterns
- Batching needs special attention
  - No overlap
- Rest of model setup / train / use / temperature is similar



F i r s t   C i t i z e n : \n B e f o r e...
window()
flat_map()
windows
3
2
1
F i r s t   C i t i z e n : \n B e f o r e...
shuffle()
batch()
map()
inputs
targets
2
10
4
batch #1
F i r s t   C i t i z e n : \n B e f o r e...



F i r s t   C i t i z e n : \n B e f o r e...
window()
flat_map()
window
2
1
...
F i r s t   C i t i z e n : \n B e f o r e...
batch()
map()
inputs
targets
batch #1
batch #2
F i r s t   C i t i z e n : \n B e f o r e...

Sentiment Analysis

IMDB Dataset

Sentiment Analysis

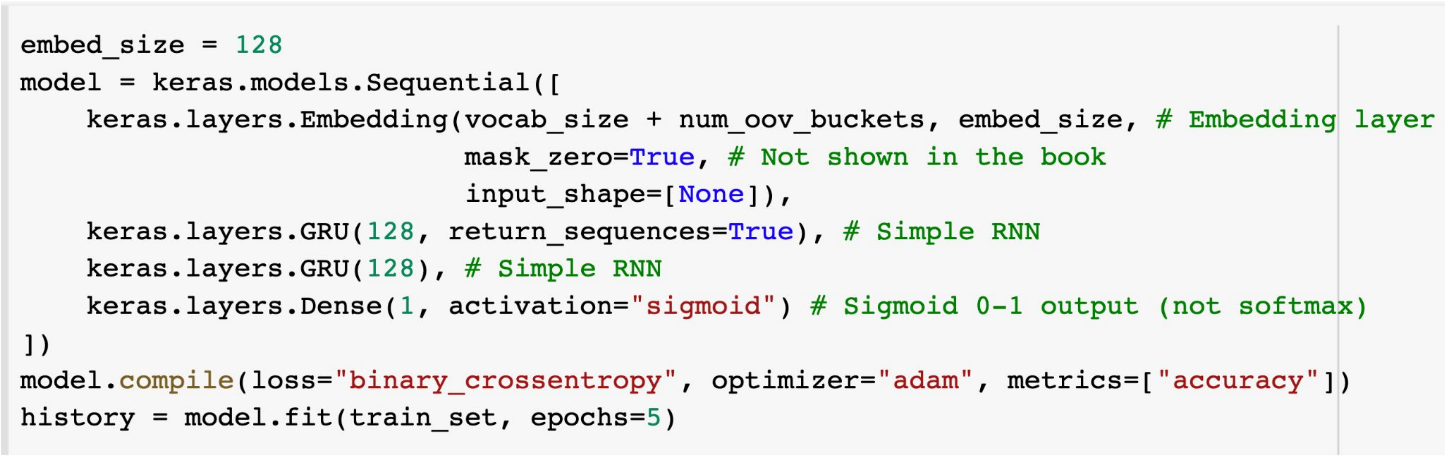
IMDB Data

- 50k Movie Reviews
  - 25k training, 25k testing
- Binary target for each review
  - Negative (0), Positive (1)
- Longer sequences to use
- Somewhat preprocessed
  - We’ll use the first 300 characters of each review
  - Can gauge sentiment from the start of a review

Sentiment Analysis

Model Setup

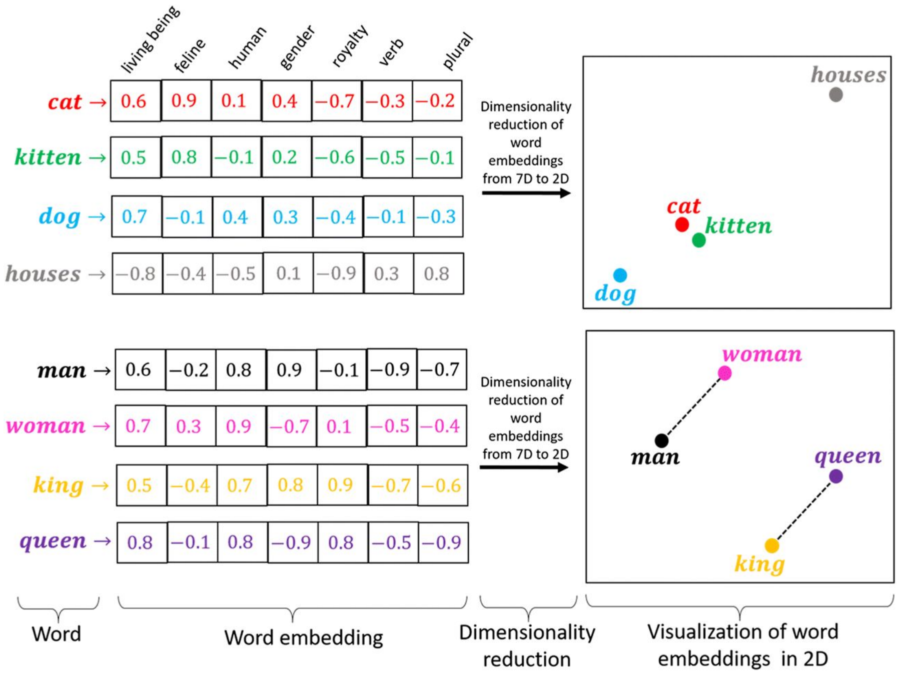
- Preprocess
  - Only use first 300 characters
  - Build vocabulary lookup table of the 10,000 most common words
  - Encode words to tokens with vocab table
    - Add PAD / OOV tokens
- Model Architecture
  - Adding an embedding layer
    - Convert word ID to feature rich embeddings
  - Same 2 basic GRU layer
  - Sigmoid output (not softmax)
    - Predict 1/0 - positive or negative



```
embed_size = 128
model = keras.models.Sequential([
    keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size, # Embedding layer
                           mask_zero=True, # Not shown in the book
                           input_shape=[None]),
    keras.layers.GRU(128, return_sequences=True), # Simple RNN
    keras.layers.GRU(128), # Simple RNN
    keras.layers.Dense(1, activation="sigmoid") # Sigmoid 0-1 output (not softmax)
])
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train_set, epochs=5)
```

Sentiment Analysis

Embeddings



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D

Dimensionality reduction of word embeddings from 7D to 2D

Word

Word embedding

Dimensionality reduction

Visualization of word embeddings in 2D

Embeddings

- Capture context and word meaning
- Train once and reusable
  - IMDB example trained on 25k reviews
  - Learned basic word understanding
- Transfer learning
  - Pretrain on general tasks to learn language fundamentals
- Companies train massive embeddings
  - Word2vec - Google 2013
  - Glove - Stanford 2014
  - BERT - Google 2019

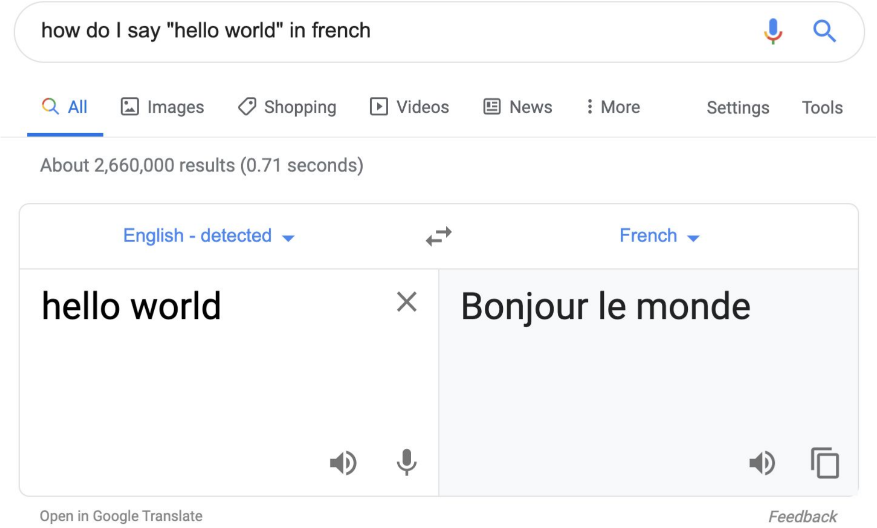
Machine Translation

English to French

Machine Translation

English - French

- English - French translation



how do I say "hello world" in french

About 2,660,000 results (0.71 seconds)

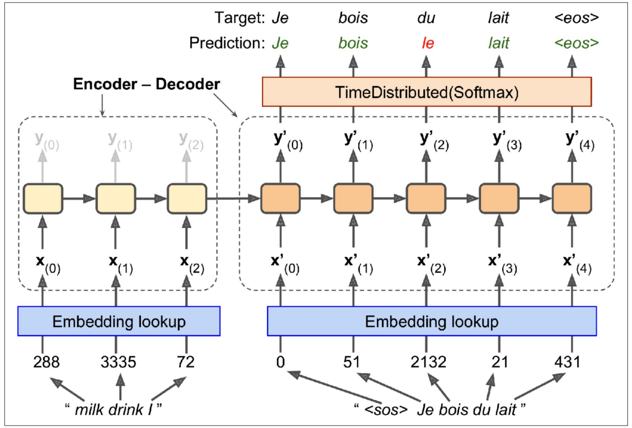
hello world

Bonjour le monde

Machine Translation

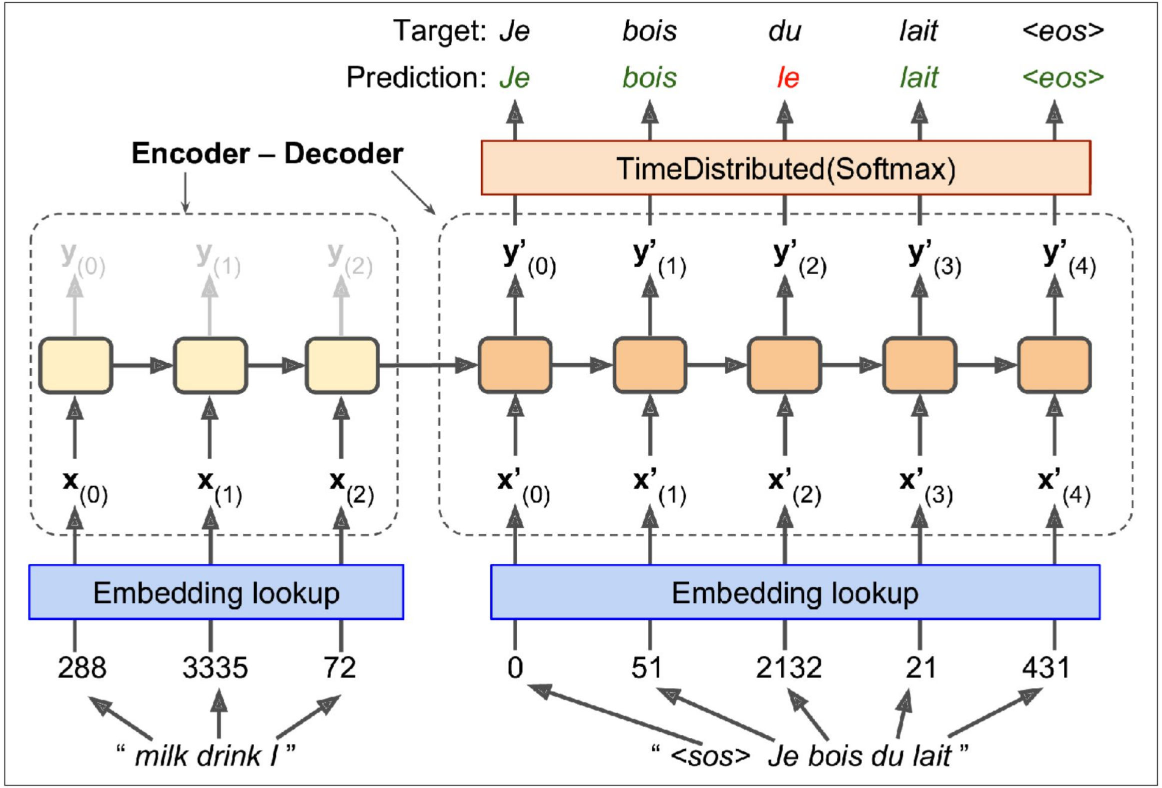
Model Setup

- Encoder - Decoder Model
  - English sentences -> encoder
  - Encoder output + actual previous French-> Decoder
  - Decoder -> French sentences
  - SOS & EOS tokens
  - Embedding layer before Encoder & Decoder layer


Target: Je bois du lait <eos>
Prediction: Je bois le lait <eos>
Encoder – Decoder
TimeDistributed(Softmax)
$y_{(0)}$ $y_{(1)}$ $y_{(2)}$
$y'_{(0)}$ $y'_{(1)}$ $y'_{(2)}$ $y'_{(3)}$ $y'_{(4)}$
$x_{(0)}$ $x_{(1)}$ $x_{(2)}$
$x'_{(0)}$ $x'_{(1)}$ $x'_{(2)}$ $x'_{(3)}$ $x'_{(4)}$
Embedding lookup
Embedding lookup
288 3335 72
0 51 2132 21 431
" milk drink I "
" <sos> Je bois du lait "

Machine Translation

Encoder-Decoder



Target: Je bois du lait <eos>
Prediction: Je bois le lait <eos>
Encoder – Decoder
TimeDistributed(Softmax)
$y_{(0)}$ $y_{(1)}$ $y_{(2)}$
$y'_{(0)}$ $y'_{(1)}$ $y'_{(2)}$ $y'_{(3)}$ $y'_{(4)}$
$x_{(0)}$ $x_{(1)}$ $x_{(2)}$
$x'_{(0)}$ $x'_{(1)}$ $x'_{(2)}$ $x'_{(3)}$ $x'_{(4)}$
Embedding lookup
Embedding lookup
288 3335 72
0 51 2132 21 431
“ milk drink I ”
“ <sos> Je bois du lait ”

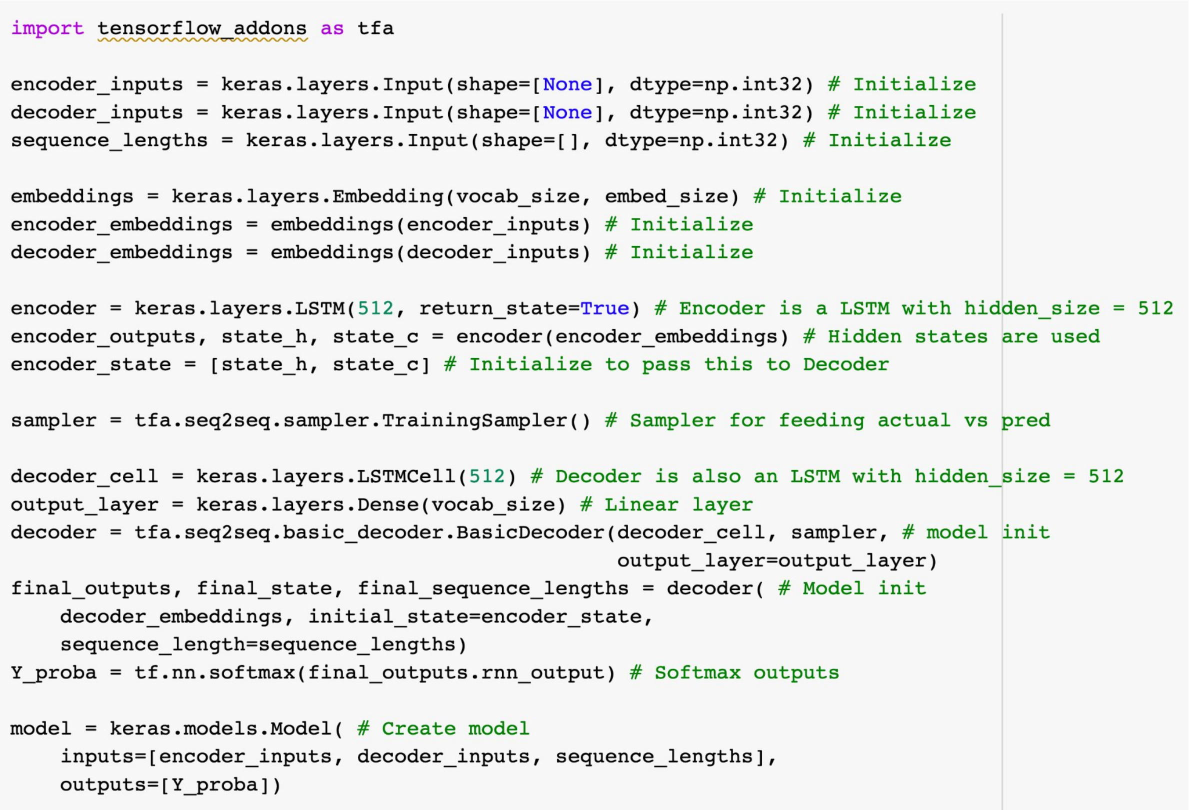
Machine Translation

Complex Model

- Need to PAD inputs to max_len
- In use, can’t feed “actual” previous term
  - Feed back in actual predicted word
- Ignore output after EOS token
- Output Vocab very large ~ 50,000 words
  - Sampled softmax - sample actual word + random sample of other words
    - Can’t be used during inference - use reg
- Eventually stop training with actual previous word and use encoder output
  - Try ratios - 70/30 - 50/50 - 20/80

Machine Translation

TF Model Code

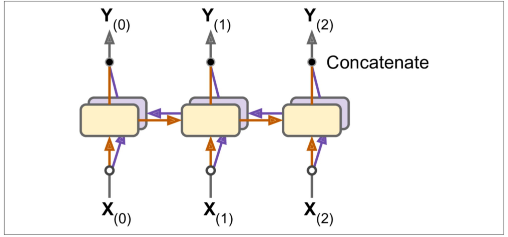


```
import tensorflow_addons as tfa

encoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32) # Initialize
decoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32) # Initialize
sequence_lengths = keras.layers.Input(shape=[], dtype=np.int32) # Initialize

embeddings = keras.layers.Embedding(vocab_size, embed_size) # Initialize
encoder_embeddings = embeddings(encoder_inputs) # Initialize
decoder_embeddings = embeddings(decoder_inputs) # Initialize

encoder = keras.layers.LSTM(512, return_state=True) # Encoder is a LSTM with hidden_size = 512
encoder_outputs, state_h, state_c = encoder(encoder_embeddings) # Hidden states are used
encoder_state = [state_h, state_c] # Initialize to pass this to Decoder

sampler = tfa.seq2seq.sampler.TrainingSampler() # Sampler for feeding actual vs pred

decoder_cell = keras.layers.LSTMCell(512) # Decoder is also an LSTM with hidden_size = 512
output_layer = keras.layers.Dense(vocab_size) # Linear layer
decoder = tfa.seq2seq.basic_decoder.BasicDecoder(decoder_cell, sampler, # model init
                                                 output_layer=output_layer)
final_outputs, final_state, final_sequence_lengths = decoder( # Model init
    decoder_embeddings, initial_state=encoder_state,
    sequence_length=sequence_lengths)
Y_proba = tf.nn.softmax(final_outputs.rnn_output) # Softmax outputs

model = keras.models.Model( # Create model
    inputs=[encoder_inputs, decoder_inputs, sequence_lengths],
    outputs=[Y_proba])
```

Machine Translation

Advanced NMT

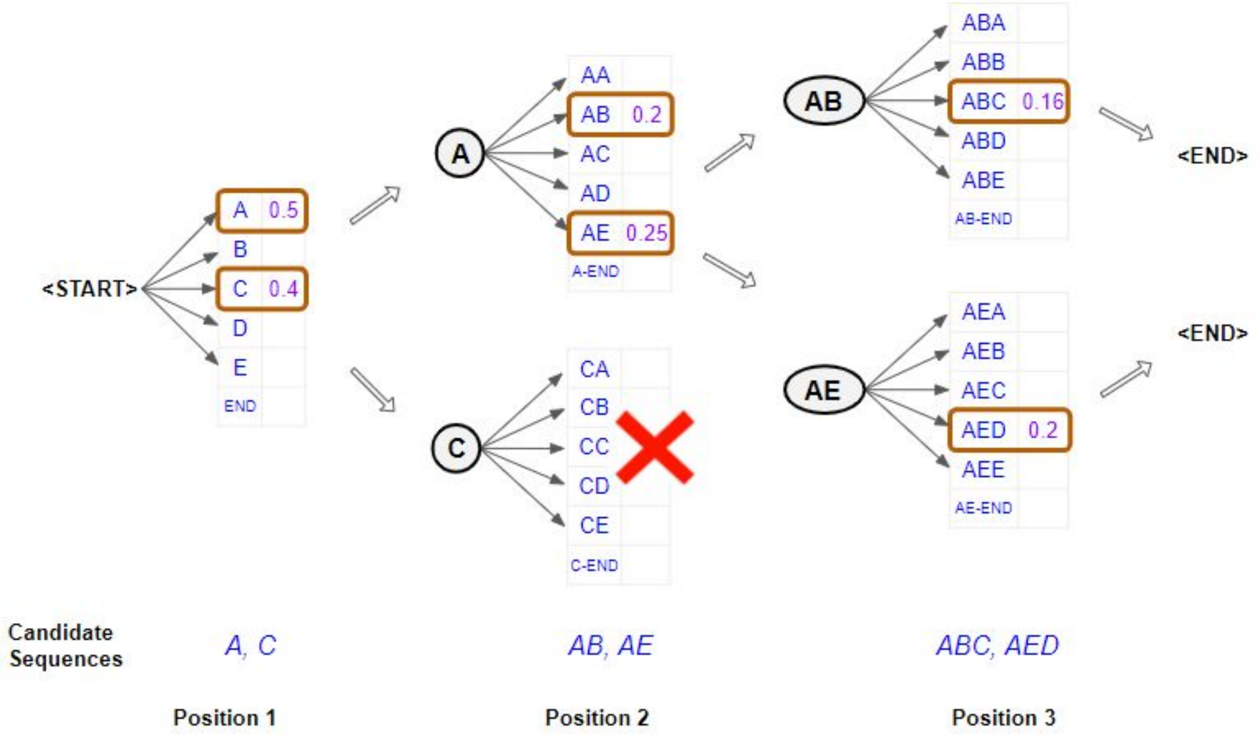
- Basic LSTM + Softmax struggles
  - Only looks forward
    - "The Queen of the UK"
    - "The Queen Bee"
    - "The Queen of Hearts
  - "Queen" will have the same encoding
- Try Bidirectional RNNs
  - Look forward and backwards



- Beam Search
  - Sometimes softmax messed up and we can't fix it
  - Keep a list of K most promising outputs
    - Compute list of possible outputs over time
    - Pick best solution

Machine Translation

Beam Search



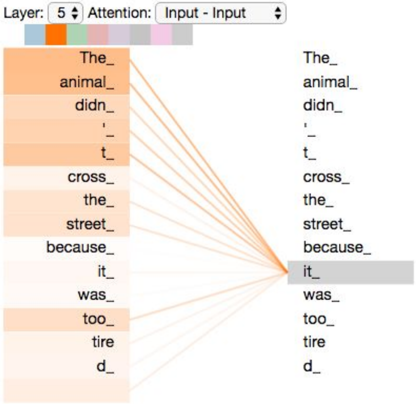
Attention Mechanism

Short term memory

Attention

NMT Issues

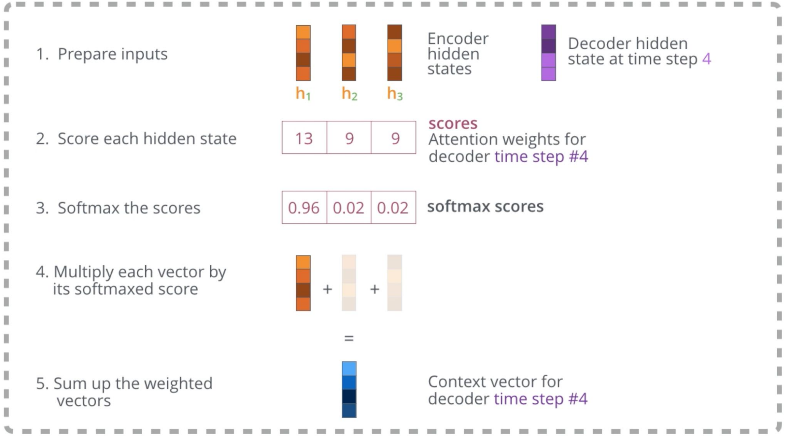
- Beam Search works well for short length
  - Struggles with longer text
  - RNN short term memory
- Attention Mechanism
  - Weight matrix between each word of input & output



Layer: 5 Attention: Input - Input

Attention

- Send all encoder weights to decoder
  - Send a lot more data than just one hidden state
- Each token has its own hidden state
  - Decoder gives a score to each token passed in
    - Softmax scores to ignore useless words



1. Prepare inputs

Encoder hidden states

Decoder hidden state at time step 4

2. Score each hidden state

13	9	9

scores
Attention weights for decoder time step #4

3. Softmax the scores

0.96	0.02	0.02

softmax scores

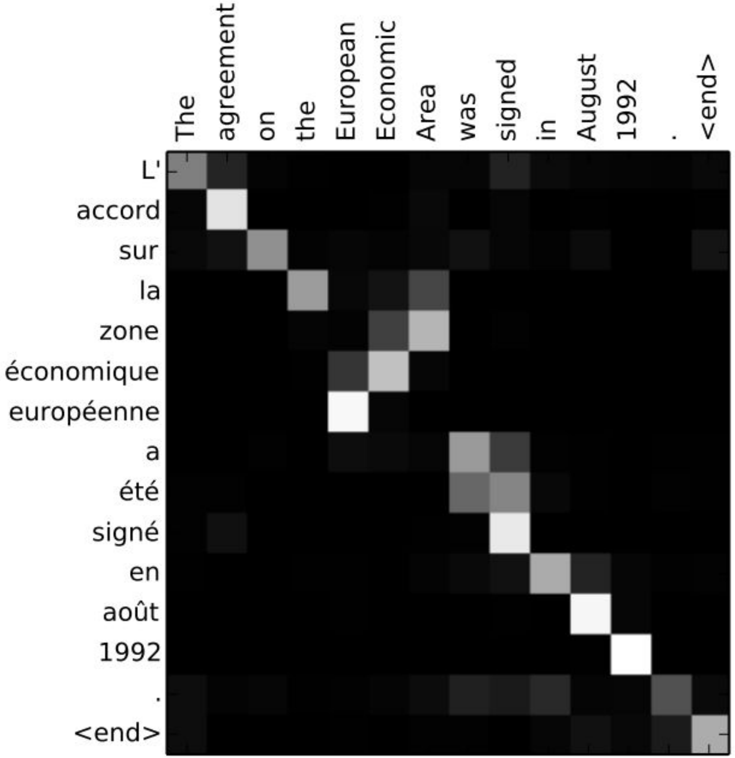
4. Multiply each vector by its softmaxed score

5. Sum up the weighted vectors

Context vector for decoder time step #4

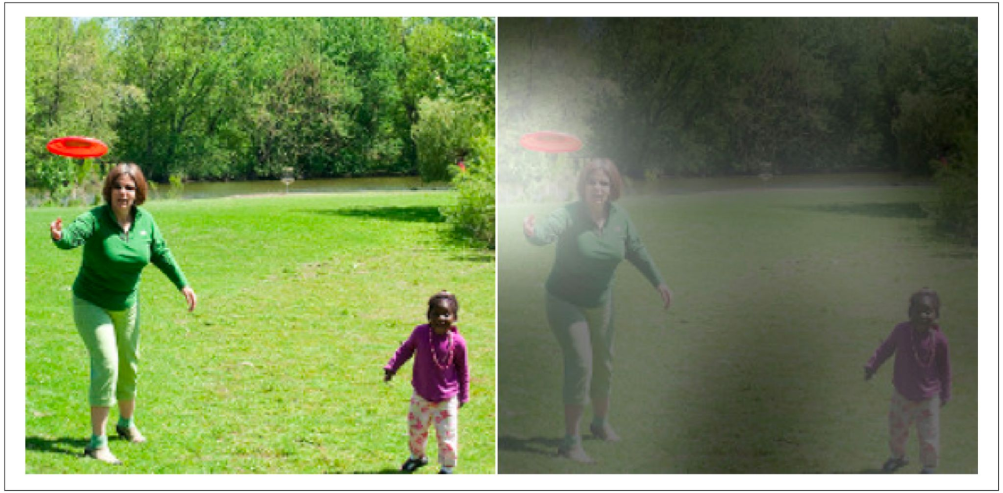
Attention

Attention



Attention Uses

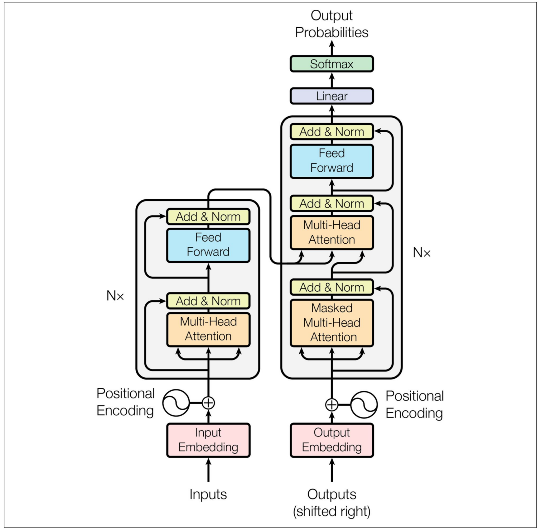
- NLP
  - Translation, summarization, generation, etc
- Vision
  - Explainable and easy to visualize



Attention

Transformer

- Next generation / SOTA RNN / LSTM
  - Backbone is attention mechanism stacked layers



Attention

Transformer

- Very large models trained over weeks
  - Trained on huge webscrapes
    - Entire wikipedia / reddit / twitter / etc
  - Cost $$$$ and time
- Great for transfer learning
  - Trained on general language understanding
  - Very unique pre-training tasks
    - Masked language pred
    - Next sentence pred
  - Open source and easy to use from large companies
- Apply to many tasks
  - Summarization, Translation, Classification, Sentiment Analysis, QA, POS, Vision, etc.

Transformer

- Easy to use
  - Apply a new layer at the end and fine tune
  - Essentially called like an RNN / LSTM layer
  - Hugging face library / Jay Alammar blog
- Teach your model the fundamentals of a language
- BERT, GPT-3, Roberta, Pegasus

Learn More

Missing Info

- Examples using transformers
  - Can share jupyter notebook
- Transformer layer visualization
- Math behind attention
  - Can do future section

THANKS!

Questions?