Deep Learning Final Notes

Andrew Chen

May 2, 2019

1 Machine Learning Basics

1.1 Dimensionality Reduction

- Supervised: Linear Discriminant Analysis
- Unsupervised: PCA and Autoencoder

2 CNN

2.1 Tricks to Know For Better Model Generalization

1. Dropout
   (a) Works because it forces the model to make a decision with limited information, thereby eliminating a lot of unnecessary parameters
2. Data Augmentation
3. Pre-train on different Larger Dataset
4. Ensemble methods
   (a) Have multiple models make a prediction and take the majority vote
5. Multitask learning
   (a) Training a set of lower level layers and then later on training top level Conv layers for a specific task
      i. ie: Train low level layers for learning low level features, and then train conv layers to recognize face, age, race, etc.

2.2 Tricks for Better Optimization

1. Batch Normalization
   (a) Change the scale of the metrics which reduces condition number of matrix
   (b) Better conditioned Hessian -> Faster Convergence
   (c) Every feature has 0 mean and unit variance
2. Gradient Injection
3. Skip Connections

3 Misc ML Facts

1. In higher dimensions, it becomes improbable to find global minimum
   (a) You will find saddle points and local minima
   (b) If you use GD, you will probably find a Saddle point because gradient near saddle points is near 0
   (c) SGD can find local minima because it's random and noisy
2. Large Batch Size -> poor generalization
   (a) Large Batch size -> faster training

4 RNN

4.1 Basic Steps of Text Processing

1. Tokenize corpus
2. Encode tokens
3. Align encodings by padding shorter encodings with 0s in the front
4. Convert encodings (vectors) to work embeddings (matrix)
   (a) Train to create a matrix of (vocab_size x embedding dimension)
   (b) For each encoding (scalar), convert it to a vector, and the final result will be a matrix where each vector within represents a word (in vector form)
5. Rules of Thumb
   (a) Always use LSTM over SimpleRNN
   (b) Always use LSTM dropout to alleviate overfitting
   (c) Use Bi-LSTM whenever possible
   (d) Use stacked LSTM if sample size is big
   (e) Pretrain the embedding layer if sample size is small
6. SimpleRNN Implementation
7. Attention
   (a) For Seq2Seq models.
      i. encoder -> final state.
         A. For each state in decoder, we look at each state in the original encoding and we choose the one that looks most similar
         B. Attention has time complexity of O(l1 * l2) instead of O(l1 + l2) (compared to w/o it)
8. Self Attention
   (a) For RNN/LSTM/GRU layers.
      i. For each state in RNN, we look back one state to generate new hidden state
         —
      ii. Calculate weights by getting similarity of current hidden state and previous context vector
         —

```python
from keras.models import Model
from keras.layers import Input, LSTM, Dense

# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Figure 1: Simple LSTM Implementation

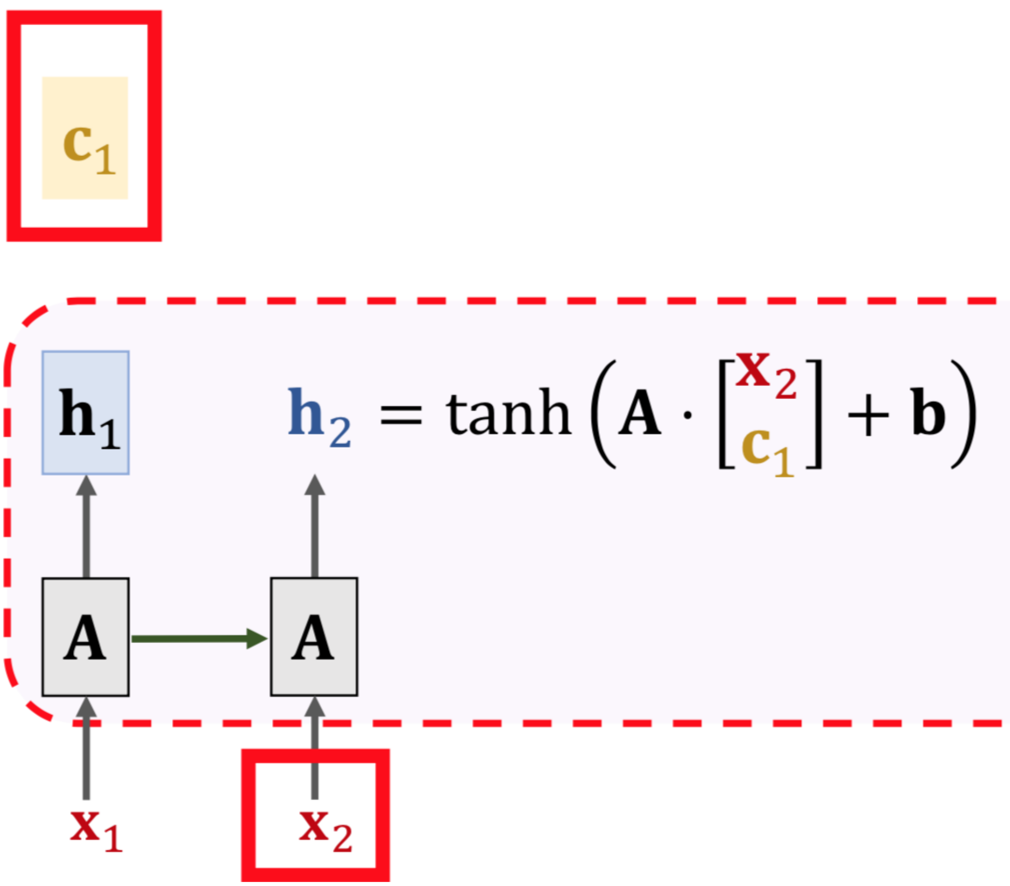

Figure 2: 1. Calculating a hidden state

iii. To Calculate Context vectors, we use use current hidden state and weight and all those from states before

9. Transformer Model

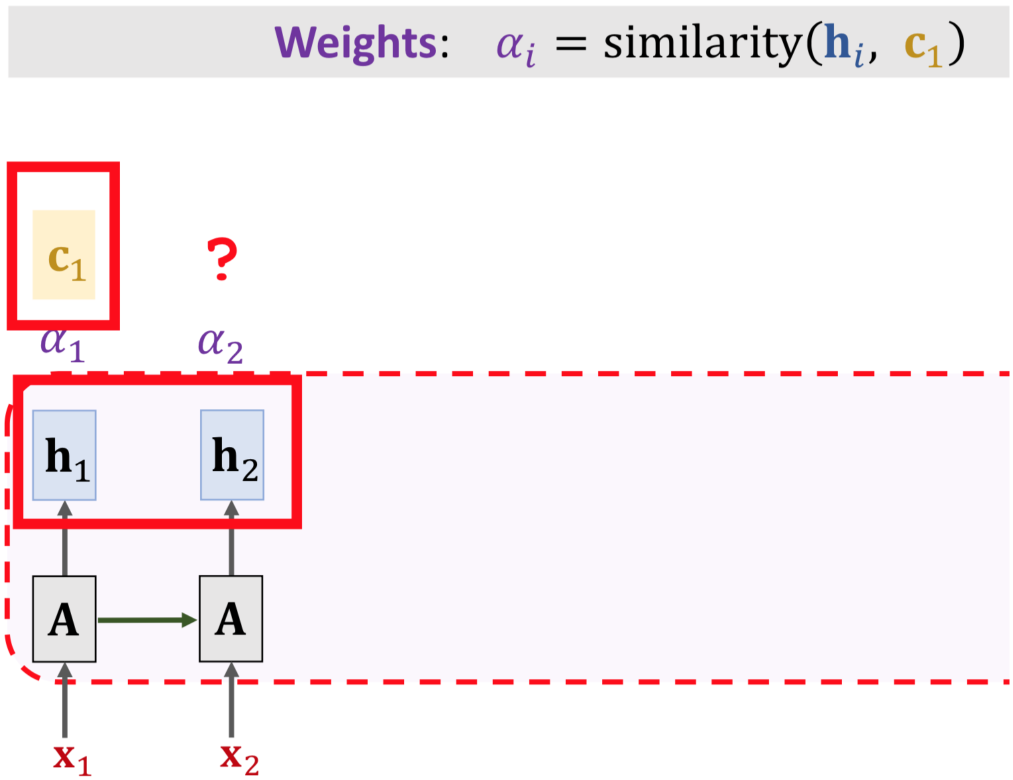

Figure 3: 1. Calculating current weight

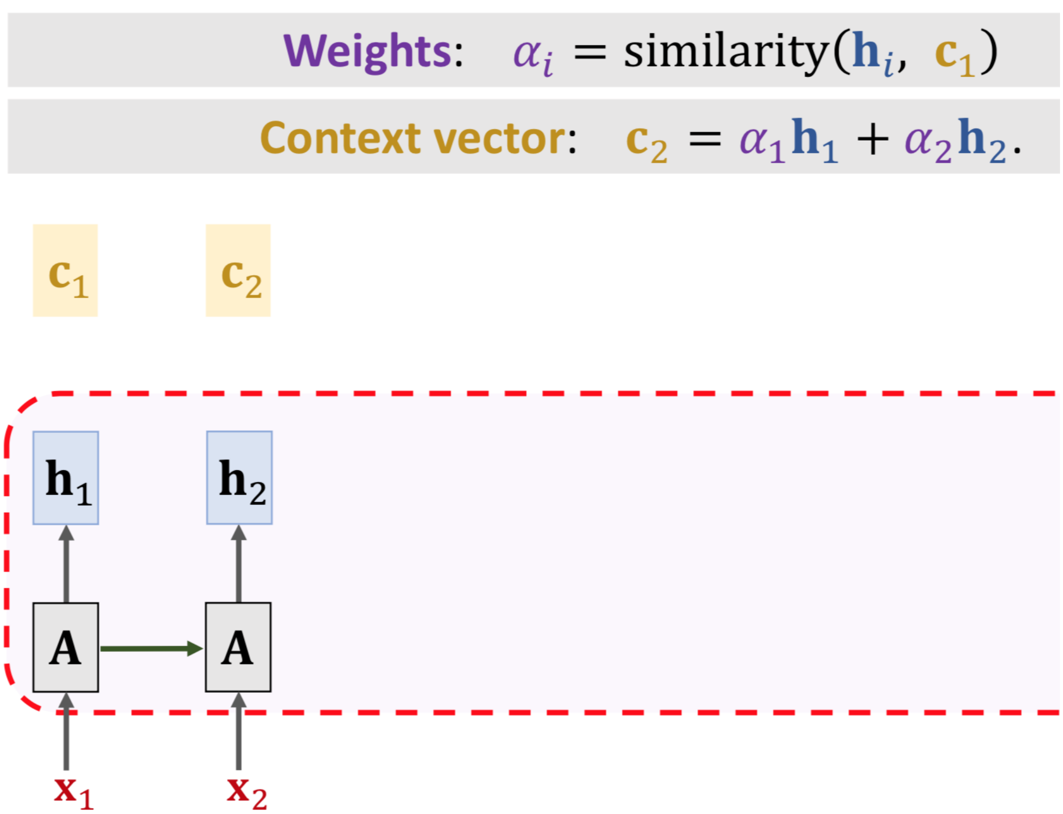

Figure 4: 2. Calculating a context vector based on all previous states

(a) Is a seq2seq model

(b) Uses Multihead attention

(c) Not RNN

(d) Purely Attention and FC layers

   i. More computation than RNNs

   ii. Better performance on larger datasets than RNNs

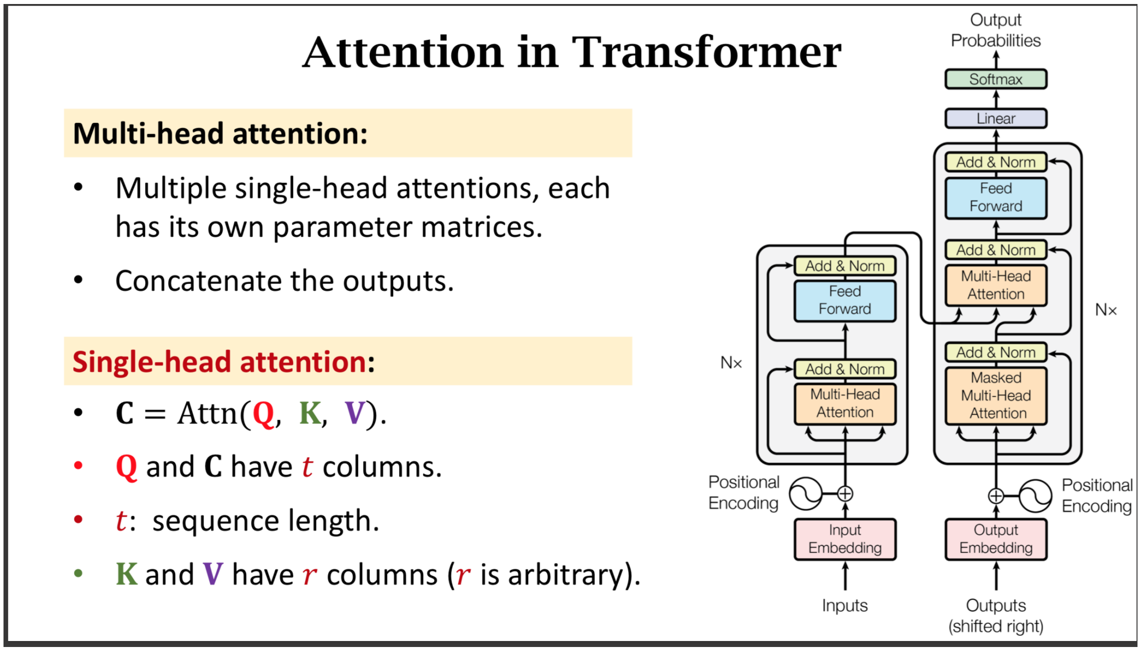

Figure 5: Transformer Model Attention Parameters

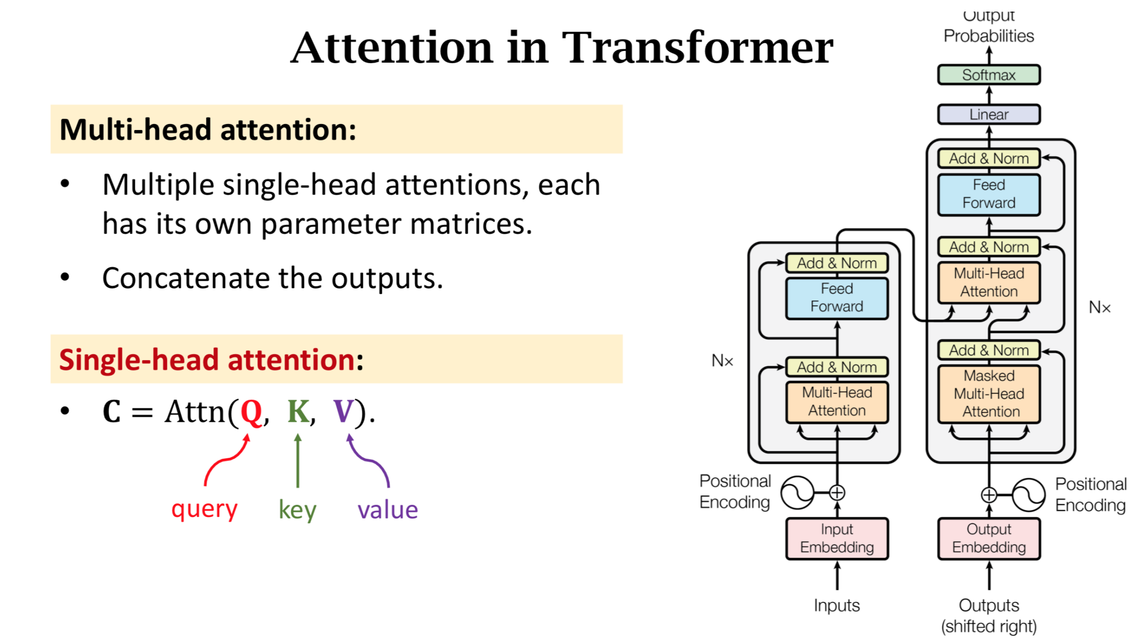

Figure 6: Transformer Model

5 Number of Trainable parameters

- Dense: `output_size * (input_size + 1)`

- Conv2D: `output_channels * (input_channels (kernel_size + 1))`
- BatchNormalization: `4 * input_channels`
- RNN: `output_shape * (output_shape + input_channels) + output_shape`
- LSTM: `4 * RNN`

6 Facial Recognition

1. Softmax classifier is bad bc its a Dense Output Layer w/ activation function of Softmax
   (a) # trainable parameters for Dense is `output_size * (input_size + 1)`
   (b) If # faces ~= 10M, and input_size = 1000, then # trainable parameters = 10M * 1000 = 10G

7 Definitions

1. Precision: How many selected items are relevant?
   (a) `relevant items / all items`
2. Recall: How many relevant items are selected?
   (a) `relevant items / all relevant items`
3. Positive Semidefinite
   (a) For convex functions, the Hessian Matrix is positive semidefinite everywhere