APS360 Fundamentals of AI

Lisa Zhang

Lecture 12; July 4, 2019

Agenda

- Midterm
- Plan for rest of course
- RNN to generate text

Midterm

Midterm

- Very well done
  - Average: 72%
  - Median: 74%
- Midterm paper scan available on Quercus
- If you want the physical copy, please come to office hours
- I'll accept remark requests by tomorrow July 5th

Plan for Rest of Course

Lectures

- Generative RNN
- Generative Adversarial Network
- Reinforcement Learning (???)
- Ethics and Fairness in AI

Alternative Plan. Thoughts?

- Generative RNN
- Generative Adversarial Network
- ~~Reinforcement Learning~~ Interpreting Neural Networks
- Ethics and Fairness in AI

Lab will be for Project Help

All your TAs will be present on Thursdays 7pm-8pm

- July 11
- July 18
- July 25

Office Hours

Move office hours to Monday 4pm-5pm?

Tutorial Next Week on Google Cloud

Andrew will be delivering the tutorial next week 6pm-7pm using Google Cloud

Project

- Proposals were generally very well done (79% average)
- Try to have a "minimum submittable project" early on
- Look at the writing feedback – can help you create a more impressive report that people are more likely to read

Project

TA Mentor allocations:

- Jake: Pokemon, News, Stock, Objects
- Farzaneh: Colour, Faces, Books, Painting
- Huan: Instrumental, Music, Audio, Chatbot, Cars
- Andrew: Pets, Food, Medical, Font

Reach out to your mentors by July 9th.

Text Generation with RNN

Today's Task: Generate Trump Tweets

- Dataset: ~20000 Trump Tweets from 2018
- At most 140 characters
- Remove tweets that starts with "http" (tweet with link only)

To help with training, we will

- prepend all tweets with a special "" token (beginning of string)
- append all tweets with a special "" token (end of string)

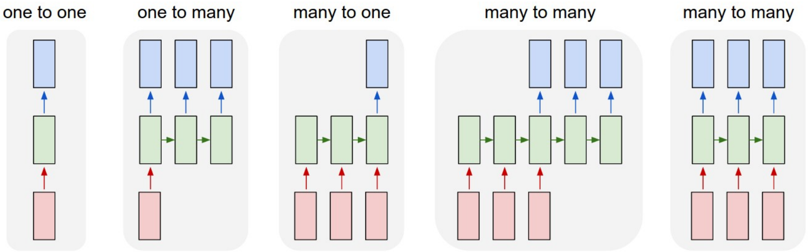
Let's look at some data!

(Follow along in Colab: http://bit.ly/GenRNN)

Lecture Struture

- Difference betwen predictive -> generative RNN
- Test-time behaviour (how to generate a tweet)
- Training-time behaviour (what loss to use)
  - Teacher-forcing
- Jupyter Notebook (coding!)
- More test-time behaviour
  - Temperature

RNN Review



RNN Hidden States

RNN For Prediction:

- Process tokens one at a time
- Hidden state is a representation of **all the tokens read thus far**

RNN Hidden States

RNN For Prediction:

- Process tokens one at a time
- Hidden state is a representation of **all the tokens read thus far**

RNN For Generation:

- Generate tokens one at a time
- Hidden state is a representation of **all the tokens to be generated**

RNN Functions

RNN For Prediction:

- Update hidden state with new input (token)
  - `hidden = update_function(hidden, input)`
- Get prediction (e.g. distribution over possible labels):
  - `output_distribution = prediction_function(hidden)`

RNN Functions

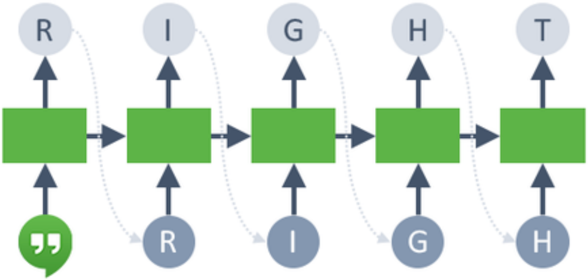
RNN For Prediction:

- Update hidden state with new input (token)
  - `hidden = update_function(hidden, input)`
- Get prediction (e.g. distribution over possible labels):
  - `output_distribution = prediction_function(hidden)`

RNN For Generation:

- Get prediction distribution of next token
  - `token_distribution = prediction_function(hidden)`
- Generate a token from the distribution
  - `token = sample_from(token_distribution)`
- Update the hidden state with new token:
  - `hidden = update_function(hidden, input)`

Text Generation Diagram



- Get prediction distribution of next token
  - `token_distribution = prediction_function(hidden)`
- Generate a token from the distribution
  - `token = sample_from(token_distribution)`
- Update the hidden state with new token:
  - `hidden = update_function(hidden, input)`

Test Time Behaviour of Generative RNN

Unlike other models we discussed so far, the training time behaviour of Generative RNNs will be **different** from the test time behaviour

Test time behaviour:

- At each time step:
  - `token_distribution = prediction_function(hidden)`
  - `token = sample_from(token_distribution)`
  - `hidden = update_function(hidden, token)`

Training Time Behaviour of Generative RNN

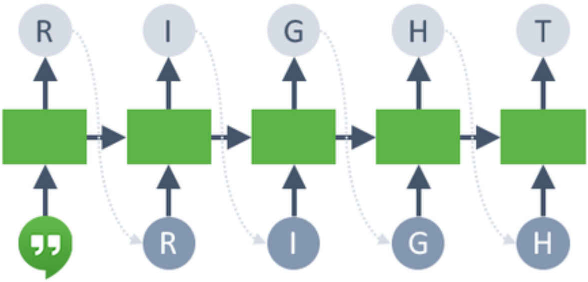
During training, we try to get the RNN to generate one particular sequence in the training set:

- At each time step:
  - `token_distribution = prediction_function(hidden)`
  - Compare the `token_distribution` with the *actual* next token

Q1: What kind of a problem is this? (regression or classification?)

Q2: What loss function should we use during training?

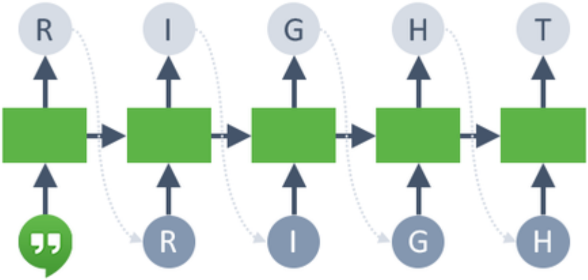
Text Generation: Step 1



First classification problem:

- Start with an initial hidden state
- Update the hidden state with a "<BOS>" (beginning of string) token, so that the hidden state becomes meaningful (not just zeros)
- Get the distribution over the first character
- Compute the cross-entropy loss against the ground truth (R)

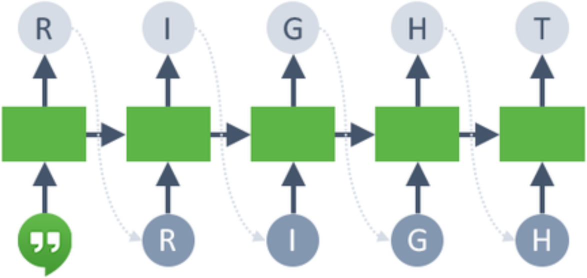
Text Generation with Teaching Forcing



Second classification problem:

- Update the hidden state with the **ground truth** token (R) regardless of the prediction from the previous step
  - This technique is called **teaching forcing**
- Get the distribution over the second character
- Compute the cross-entropy loss against the ground truth (I)

Text Generation: Later Steps



Continue until we get to the “<EOS>” (end of string) token

Example Code

- We'll build a first generative RNN model
- Then, we'll start off with a very inefficient training code that computes the loss one time step at a time
- Then, when we understand what should happen under the hood, we'll switch to a more performant version of the code

(One more slide before Jupyter)

RNN Model

```
class TextGenerator(nn.Module):
  def __init__(self, vocab_size, hidden_size):
    super(TextGenerator, self).__init__()
    self.ident = torch.eye(vocab_size)
    self.rnn = nn.GRU(vocab_size,
                      hidden_size,
                      batch_first=True)
    self.decoder = nn.Linear(hidden_size, vocab_size)

  def forward(self, inp, hidden=None):
    inp = self.ident[inp]
    output, hidden = self.rnn(inp, hidden)
    output = self.decoder(output)
    return output, hidden
```

Jupyter Notebook!

Sampling a Token during Test Time

Unlike in an actual classification problem, always generating the token with the highest probability **won't work**.

Q: Why?

Sampling from a multinomial distribution

Suppose that the RNN's predicted (softmax) distribution of the first token was:

- A $=$ 60%, B $=$ 40%, everything else $=$ 0%

Then,

- If temperature $= 1$, probability of sampling A $= 60\%$
- If temperature < 1, probability of sampling A $> 60\%$
  - **Low temperature** $=$ less random
- If temperature > 1, probability of sampling A $< 60\%$
  - **High temperature** $=$ more random

Temperature Tradeoff

- Low temperature:
  - Higher quality samples
  - Less variety
- High temperature:
  - More variety
  - Lower quality samples

Training

- Training on CPU is quite slow
- Let's train using a GPU on Google Colab!