Language Models and RNN

Shubham Gupta

January 16, 2020

1 Introduction

- Introduce language modelling(LM)
- LM motivates building RNN

1.1 Language Modelling

- Predict what words come next
- Computes probability distribution of next work x^{t+1} given previous words $x^1, x^2...x^t$ where x is in the vocab V.
- Assign prob to a piece of text

$$\prod_{t=1}^{T} P(x^t | x(t-1), ..., x^1) \tag{1}$$

1.2 N-gram language model

- **n-gram** Chunk of n consecutive words
- Collect stats about how frequent different n-grams are, and use this to predict the next word.
- **Assumption**: x^{t+1} depends only on preceding $n-1$ words
- To get the probabilities, we will count them from a large corpus i.e *statistical approximation*

1.3 Problems

- Throws away too much context
- Sparsity. If the numerator is 0, then the chance of a valid word occuring is not possible, which is incorrect. Just because the n-words were not seen in the dataset does not mean that it is not a valid concept
- Solution? Smoothing. Add a small amount of probability *delta* for every word in the vocabulary
- If denominator is 0, cannot calculate the probability at all.

- Solution? If you cannot find n words in the dataset, backoff and just use the last n-1 or n-2 words instead. This is called **backoff**
- Sparsity problems increase with increase in n
- **Storage**: Size of model increases as the n-grams increase

2 Fixed window neural language model

- Use neural network for word prediction
- Represent window of words as a one-hot encoding
- For each word, obtain the word embedding from a model such as word2vec
- Pass this to a hidden layer and multiply it with weight matrix containing non-linearity and a smaller size
- Pass output from hidden layer to softmax to get probabilities. Softmax output will be of entire vocab size i.e V.
- **Advantages**
  - No sparsity problem
  - Don't need to store all n-grams you've seen.
- **Problems**
  - Fixed window will always be small. Increasing it will increase size of W, leading to more problems
  - No symmetry i.e because of matrix multiplication, each word vector is multiplied only by specified column of wight vectors. So you are learning something specific for each word rather than learning a general function.
  - Need neural network to process strings of arbitary length

3 RNN

- Recurrent neural network
- Any length input sequence
- Sequence of hidden states. Each state computed based on previous hidden state and the current input. Also called *timesteps*
- **Same weight matrix is applied at each step**. This helps learn a general function.
- Hidden state computed as:

$$h^t = \sigma(W_h h^{t-1} + W_e e^t + b_1) \tag{2}$$

- Initial hidden state can be either vector of 0's or any arbirary vector.

- Final output will have softmax layer. Output size will be of size vocab V
- **Advantages**
  - Any length of input
  - Computation at t can use info from previous steps
  - Symmetry in weights application i.e learns a general function
- **Disadvantages**
  - Recurrent computation is slow
  - In practice, info from many steps back is difficult to retain
- Training
  - Obtain big corpus
  - Feed into RNN-LM. Compute $\hat{y}$ for every step y
  - Loss function is cross entropy between prediced next word and the true next word y^t $J(\theta) = -log(\hat{y}^t_{x_{t+1}})$
- Computing gradient and loss over entire corpus is expensive. In practice, use input as a sentece or collection of senteces
- Apply *Stochastic gradient descent* and compute everything in batches
- Backpro in RNN will be the sum of the gradient wrt each time it appears. Words using multivariable chain rule

$$\frac{\delta J^t}{\delta W_h} = \sum_{u=1}^{t} \delta J^{\frac{t}{\delta W_h}} \tag{3}$$

- Accumulate the gradients at each timestep. Also called *backpropogation through time*
- Generating text will be the same as repeated sampling used in the n-gram model

3.1 Evaluation LM

- **Perplexity**: Inverse probability of the corpus given the language model

$$perplexity = \prod_{t=1}^{T} \left(\frac{1}{P_{LM}(x^{t+1}|x^t, ..., x^1} \right)^{\frac{1}{T}} perplexity = exp(J(\theta)) \tag{4}$$

4 Recap

- LM predicts next word
- RNN Seq input of any length, apply same weights and produce output
- RNN to build LM

- Example: RNN for POS tagging task
- RNN can also be used as a general purpose encoder model. Can be used for machine translation, question answering, etc.
- *Vanilla RNN* = RNN in this lecture
- Multilayer RNN possible