**Neural Networks & NLP**

1. Recap of last week: language modelling
   1. Language Modeling is the task of predicting what word comes next.
   2. More formally: given a sequence of words <x1, x2, …, xt>, compute the probability distribution of the next word (x^{t+1}):  
        
      P(x^{t+1} given <xt, …, x1>)  
        
      where x^{t+1} can be any word in the vocabulary V = {w1, …., wV}
2. N-gram language models n-gram Language Models
   1. First we make a simplifying assumption: x^{t+1} depends only on the preceding n-1 words.
3. NL Generation with language models Generating text with a n-gram Language Model
   1. You can also use a Language Model to generate text.
4. Sequence Likelihood Tasks
   1. Speech recognition:
      1. I saw a van
      2. eyes awe of an
   2. Spelling correction:
      1. It’s about fifteen minuets from my house
      2. It’s about fifteen minutes from my house
   3. Machine translation
      1. vjetar će biti noćas jak:
         1. the wind tonight will be strong
         2. the wind tonight will be powerful
         3. the wind tonight will be a yak
5. Sequence Tagging Tasks in NLP:
   1. Part-of-Speech tagging:
      1. mary PN hires VBZ a DET detective CN
   2. Named Entity tagging:
      1. Today O President B-PER Donald I-PER J. I-PER Trump E-PER announced O
   3. Dialogue Act tagging:
      1. A: So do you go to college right now? YN-QUESTION
      2. B: Yeah YES-ANSWER
      3. A: Are yo- ABANDONED
      4. B: it’s my last year STATEMENT
      5. A: What did you say? CLARIFY
      6. B: my last year NP-ANSWER
      7. A: Oh good for you APPRECIATION
      8. B: uh-huh BACKCHANNEL
   4. Why are these not just (word/sentence) classification tasks?
6. Fixed window neural language models:
   1. output distribution: hat(y) = softmax(Uh + b\_2) contained within R^{|V|}
   2. hidden layer: h = f(We + b\_1)
   3. concatenated word embeddings: e = [e1; e2; e3; e4]
   4. words / one-hot vectors: x1, x2, x3, x4
7. Fixed window neural language models
   1. Advantages
      1. Do not suffer from sparsity problem
      2. No need to store all the n-grams to compute the probabilities
   2. Disadvantages
      1. Lack of flexibility: same window both when there is little context (in which case you need to pad) and when there is lots
      2. Restrictions on size now imposed by number of parameters rather than sparsity
8. Computing with context:
   1. pseudo-code
      1. state\_t = 0
      2. for input\_t in input\_sequence:
         1. output\_t = f(input\_t, state\_t)
         2. state\_t = output\_t
      3. state\_t = 0
      4. for input\_t in input\_sequence:
         1. output\_t = activation(dot(W,input\_t) + dot(U,state\_t) + b)
         2. state\_t = output\_t
9. A RNN Language Model:
   1. output distribution: hat(y) = softmax(Uh + b\_2) contained within R^{|V|}
   2. hidden states: h = sigma(W\_t \* h^{t-1} + W\_t \* e^t + b1)  
      h0 is the hidden state
   3. word embeddings: e = Ex^t
   4. words / one-hot vectors: xt contained within R^{|V|}
10. Training an RNN language model
    1. Get a big corpus of text which is a sequence of words x1, …, xt
    2. Feed into RNN-LM; compute output distribution hat(y) for every step t.
       1. i.e. predict probability dist of every word, given words so far
    3. Loss function on step t is usual cross-entropy between our predicted probability distribution ,hat(y) and the true next word y = x^{t+1}:
       1. J(theta) = CE(yt, hat(y)) = sum of (yi \* log y\_j)
    4. Average this to get overall loss for entire training set:
       1. J(theta) = 1/T \* sum of(J(theta))
       2. However: Computing loss and gradients across entire corpus is too expensive!
       3. Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
       4. In practice, consider as a sentence
       5. Compute loss J(theta) for a sentence (actually usually a batch of sentences), compute gradients and update weights. Repeat.
11. Backpropagation for RNNs
    1. What’s the derivative of J(theta) w.r.t. the repeated weight matrix Wh?
    2. Del J / Del w\_h = sum of (Del J^t / Del w\_h)
    3. “The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears”
12. Gradient computation in RNNs
    1. Given a multivariate function f(x, y) and two single variable functions x and y, here’s what the multivariable chain rule says:
       1. df(x, y) / dt = (del f / del x) \* (dx / dt) + (del f / del y) \* (dy / dt)
13. Evaluating language models:
    1. The standard evaluation metric for Language Models is perplexity.  
         
       perplexity = product sum of (1 / P\_LM(x^{t+1} given xt, …, x1)) ^ {1/T}  
         
       where,   
       product sum of (1 / P\_LM(x^{t+1} given xt, …, x1)) = Inverse probability of corpus, according to Language Model  
         
       1/T = Normalized by number of words
    2. This is equal to the exponential of the cross-entropy loss:  
         
       = exp(1/T \* -sum of (log hat(y)\_x\_{t+1})) = exp(J(theta))  
         
       Lower perplexity is better!
14. RNN applications:
    1. Automatic Speech Recognition (ASR):
       1. Mikolov et al. (2011) train RNN on some standard corpora, and it outperforms existing models in WER
    2. Using RNNs for tagging.
    3. Using RNNs for sentence classification (e.g., sentiment analysis) RNNs can be used for sentence
    4. How to compute sentence encoding? Basic way: Use final hidden state; Usually better: Take element-wise max or mean of all hidden states.
    5. The sequence-to-sequence model
       1. Encoding of the source sentence. Provides initial hidden state for Decoder RNN.
       2. Encoder RNN produces an encoding of the source sentence.
       3. Decoder RNN is a Language Model that generates target sentence conditioned on encoding.
    6. Using RNNs for generation:
       1. Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step’s input.
    7. Generating text with an RNN language model:
       1. You can train a RNN-LM on any kind of text, then generate text in that style.
15. Vanishing gradient
    1. In many NLP applications, the information on which interpretation / prediction depends is far away:
       1. Language modelling: I grew up in France… I speak fluent French.
       2. Sentiment analysis: This is an amazing film. It tells the story of ……
       3. Machine translation
    2. Theoretically, RNNs would be ideally suited for this
    3. But in practice, as the gap grows, RNNs become less and less able to maintain the relevant information in its state.
    4. Vanishing gradient problem: When gradient are small, the gradient signal gets smaller and smaller as it backpropagates further
    5. Gradient signal from faraway is lost because it’s much smaller than gradient signal from close-by. So model weights are only updated only with respect to near effects, not long-term effects.
    6. Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we’d like [Linzen et al 2016]
    7. Vanishing gradient: solutions
       1. Researchers have been aware of this problem with RNNs almost since they were proposed (Bengio et al, 1994)
       2. A number of heuristic solutions were proposed
          1. Choose initialization carefully
          2. This works better for exploding gradient (see ‘clipping gradient’ method by Mikolov et al 2012)
       3. Use ReLUs instead of sigmoid
       4. But the best solution is the use of models that have more control over what is forgotten and what is passed:
          1. LSTMs (Hochreiter & Schmidhuber, 1997)
          2. GRUs (Cho et al, 2014)
       5. RELUs as a solution to the VGP:  
            
          RELU = 0 if x < 0; x if x >= 0
16. LSTMs
    1. LSTMs (Long-Short-Term-Memory Networks) are a form of RNN introduced in 1997 by Hochreiter & Schmidhuber
    2. The crucial innovations with respect to basic RNNs are that
       1. They maintain a `LONG TERM’ state in addition to the SHORT TERM state maintained by basic RNNs
       2. They use special sub-NNs called GATES to decide what to forget of the past state and what to keep of the current input
17. LSTM – the core ideas: GATES
    1. Gates are composed of a SIGMOID LAYER and a POINTWISE MULTIPLICATION
    2. The sigmoid layer outputs values between 0 and 1
       1. 0: “let nothing through”
       2. 1: “let everything through”
    3. The forget gate: controls what is kept vs forgotten, from previous cell state
       1. F = sigma (W \* [h\_{t-1}, xt] + b)
    4. the input gate: controls what parts of the new cell content are written to cell
       1. i = sigma (W \* [h\_{t-1}, xt] + b)
    5. New cell content: this is the new content to be written to the cell
       1. C\_t = tanh (W \* [h\_{t-1}, xt] + b)
    6. updating the cell state: erase (“forget”) some content from last cell state, and write (“input”) some new cell content
       1. C\_t = f \* C\_{t-1} + t \* C\_t
       2. Crucial point here: old cell state (as modified by forget gate) and input update are ADDED not MULTIPLIED!
    7. Output: controls what parts of cell are output to hidden state
       1. o = sigma (W \* [h\_{t-1}, xt] + b)
    8. Hidden state: read (“output”) some content from the cell
       1. h = 0 \* tanh(C\_t)
    9. Gates are applied using element-wise product
    10. All these are vectors of same length n
18. How do LSTMs solve the VGP?
    1. Remember the reason for the problem: with RNNs, the updates to the hidden state can become vanishingly small
    2. With LSTMs, the update to the cell state is:  
         
       del C / del C\_{t-1} = C\_{t-1} \* sigma’ (.) W \* o\_{t-1} tanh’(C\_{t-1}) +   
       C\_{t} \* sigma’ (.) W \* o\_{t-1} tanh’(C\_{t-1}) +   
       i\_{t} \* tanh’ (.) W \* o\_{t-1} tanh’(C\_{t-1}) +   
       f\_t
    3. Which can get smaller or larger, but its growth is controlled by the forget gate
19. LSTMs applications
    1. LSTMs were the basic building block of most neural-based NLP systems from around 2000 to around 2018. They are still used as key ingredients in architectures
       1. To learn word embeddings
       2. For text classification
       3. For machine translation (in a few weeks from now)
20. LSTMs for embeddings
    1. State-of-the-art NLP models use word embeddings computed using (Bi)LSTMs instead of word2vec
       1. Coreference: Lee et al, 2017 (see lecture 7)
       2. The ELMO model (Peters et al, 2018)
21. LSTMs for text classification
    1. LSTMs are competitive with CNNs for text classification
22. GRUs
    1. Introduced by Cho et al 2014
    2. Can be seen either as a drastic simplification of LSTMs, or as an intermediate step between Basic RNNs and LSTMs:
       1. Only one state as RNNs
       2. But two gates to control what goes through / what gets added
23. GRUs: the gates
    1. Update gate: z = sigma(W \* x(t) + U \* h\_{t-1})
    2. Reset gate: r = sigma(W \* x(t) + U \* h\_{t-1})
    3. New memory content: h = tanh (W \* x(t) + r \* U \* h\_{t-1}); If reset gate unit is ~0, then this ignores previous memory and only stores the new word information
    4. Final memory at time step combines current and previous time steps: h = z \* h\_{t-1} + (1 - z) \* h\_t
24. GRU intuition
    1. If reset is close to 0, ignore previous hidden state
       1. Allows model to drop information that is irrelevant in the future
    2. Update gate z controls how much of past state should matter now.
       1. If z close to 1, then we can copy information in that unit through many time steps! Less vanishing gradient!
    3. Units with short-term dependencies often have reset gates very active
25. Bidirectional RNNs Bidirectional RNNs
    1. Problem: For classification you want to incorporate information from words both preceding and following  
         
       right\_arrow\_vector(h) = f(rightarrow(W)\*x\_t + rightarrow(V)\* rightarrow(h\_{t-1}) + rightarrow(b))  
         
       left\_arrow\_vector(h) = f(leftarrow(W)\*x\_t + leftarrow(V)\* left rrow(h\_{t-1}) + leftarrow(b))  
         
       t = g(U[rightarrow(h\_t); leftarrow(h\_t)] + c)  
         
       h = [rightarrow(h\_t); leftarrow(h\_t)] now represents (summarizes) the past and future around a single token.
26. Deep Bidirectional RNNs
    1. Each memory layer passes an intermediate sequential representation to the next.