Natural Language Understanding, Generation, and Machine Translation (2021–22)

School of Informatics, University of Edinburgh
Frank Keller

Tutorial 2: Neural Network Language Models (Week 4)

This tutorial includes both calculation questions and more open-ended questions. Question 3 is intended to help you think about how to apply models you've seen before to new problems. *Most* of the points on the exam will come from questions of this form, that require you to think through a new, open-ended scenario.

1 The Softmax Function

The softmax function takes an arbitrary vector $\mathbf{v}$ as input, with $|\mathbf{v}|$ dimensions. It computes an output vector, also of $|\mathbf{v}|$ dimensions, whose ith element is given by:

$$\text{softmax}(\mathbf{v})_i = \frac{\exp(\mathbf{v}_i)}{\sum_{j=1}^{|\mathbf{v}|} \exp(\mathbf{v}_j)}$$

Question 1: Softmax Function

a. What is the purpose of the softmax function?

b. What is the purpose of the expression in the numerator?

c. What is the purpose of the expression in the denominator?

Now consider how a neural language model with a softmax output layer compares with a classic n-gram language model. Typically, we use techniques like smoothing or backoff in conjunction with n-gram models.

d. Does this problem arise in the neural model? Why or why not?

Solution 1:

a. The softmax converts an arbitrary vector of $|\mathbf{v}|$ dimensions into a valid categorical probability distribution over $|\mathbf{v}|$ possible outcomes. In particular it ensures that all individual elements (probabilities) are non-negative and sum to one.

b. The numerator ensures that all values are positive. Note that this is stronger than needed: the axioms of probability simply require all values to be non-negative. But exponentiation is only zero in the (negative) limit.

c. The denominator normalises the distribution so that all individual probabilities sum to one.

d. No—softmax ensures that the model will always return a non-zero probability for any n-gram.

2 Feedforward Language Models

Consider a **feedforward language model** of the type discussed in lecture 5. In this model, the probability $P(w_i \mid w_{i-n+1}, ..., w_{i-1})$ is given by:

$$\begin{aligned} P(w_i \mid w_{i-n+1}, ..., w_{i-1}) &= \text{softmax}(\mathbf{V}\mathbf{h}_2 + \mathbf{b}_2) \\ \mathbf{h}_2 &= \tanh(\mathbf{W}\mathbf{h}_1 + \mathbf{b}_1) \\ \mathbf{h}_1 &= \text{concatenate}(\mathbf{C}\mathbf{w}_{i-n+1}, \ldots, \mathbf{C}\mathbf{w}_{i-1}) \\ \mathbf{w}_i &= \text{onehot}(w_i) \qquad\qquad \triangleleft \text{ for all } i \end{aligned}$$

In this notation, $\mathbf{V}$, $\mathbf{W}$, and $\mathbf{C}$ are matrices, while $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{w}_{i-n+1}, \ldots, \mathbf{w}_{i-1}$ are vectors. The parameters of the model are $\mathbf{V}$, $\mathbf{W}$, $\mathbf{C}$, $\mathbf{b}_1$, and $\mathbf{b}_2$, while the remaining variables are intermediate layers computed by the network.

Now consider the number of parameters required to represent this model. This number is determined by the size of the vocabulary (given to you by the data), the order n, and the dimension of the two hidden layers, $\mathbf{h}_1$ and $\mathbf{h}_2$, which we will denote d_1 and d_2, respectively (Note that the first dimension must be divisible by n, but you can ignore this detail in your calculations). Dimensions d_1 and d_2 are modeling choices, though the practical consideration is how they impact the model's accuracy.

Question 2: Parameters of Neural Nets

a. How would you express the number of model parameters in terms of $|V|$, n, d_1, and d_2?

b. An effective size for the hidden dimension of a neural NLP model is often in the hundreds. For n from 2 to 5, how many parameters would your model have if $d_1 = d_2 = 100$? What if $d_1 = d_2 = 1000$?

c. What do you conclude about the relative memory efficiency of classic n-gram and feedforward neural language models? If you increased n even further, what would happen?

d. How would you expect the number of parameters in an RNN model to scale with $|V|$?

e. **[Advanced, for now]** Can you think of any strategies to substantially reduce the number of parameters?

Solution 2:

a.
- For $\mathbf{V}$: $d_2|V|$
- For $\mathbf{W}$: $d_1 d_2$ because d_1 is predefined, and will be equal to $(n-1) * |embedding|$
- For $\mathbf{C}$: $|V|d_1/(n-1)$ because $|embedding| = d_1/(n-1)$
- For $\mathbf{b}_1$: d_2
- For $\mathbf{b}_2$: $|V|$
- For the complete model, add up the above:
  $(1 + d_1/(n-1) + d_2)|V| + d_1 d_2 + d_2$
- The key here is that $(1 + d_1/(n-1) + d_2)|V|$ dominates, and this is determined by the mapping between the one-hot vocabulary vectors and the hidden dimensions. A reasonable guess for $|V|$ in most neural network language models is 20000 and $d_1/(n-1) = d_2 = 100$ should give parameters of about 4M parameters. If $d_1/(n-1) = d_2 = 1000$

then you have about 40M parameters. However if we try to model an open vocabulary in this model we will struggle to fit this in memory on a GPU whose memory is generally far smaller than a CPU machine. In tutorial 1 we saw $|V| = 2.6 \times 10^9$ which would give us about 500 billion parameters when $d_1/(n-1) = d_2 = 100$, and 50 trillion parameters when $d_1/(n-1) = d_2 = 1000$.

b. Thinking about tutorial 1, the numbers of parameters for an n-gram language model when $n = 5$ were 500 billion (5×10^{11}). The numbers of parameters with the same vocabulary size are similar for a FFNN when $n = 5$. But consider what happens as n increases or decreases: The number of parameters in the n-gram model is highly sensitive to changes in n, while the number of parameters in the neural model is *almost unchanged.* For $n < 5$, the n-gram model is more compact, but for $n > 5$, the feedforward model is. Hence, the feedforward model can be easily extended to larger n, which might be advantageous.

c. An RNN scales in the same way as the feedforward model: the dominant factor is the vocabulary size. It's entirely insensitive to n since it (theoretically) models $n = \infty$.

d. For RNN models, the key to reducing the number of parameters is to reduce vocabulary size. This can be done with subword modeling (discussed in lecture 9). Notice that this is inappropriate for the n-gram model, since it would be conditioning on less information! Note that the feedforward model has a similar limitation, though it is easier to increase the order n of the feedforward model.

   Note: In the deep learning literature, it's common to replace rare words with an unknown word token (often denoted UNK) in order to keep the vocabulary size down. This *might* be ok for certain types of lab experiments focusing on algorithmic differences, but has no place in real use cases, because you simply cannot have a machine translation or NLG system that generates UNK everywhere.

The next problem looks at how to apply neural models you've just learned about to a problem you've seen in previous courses: part-of-speech tagging. Given an input sentence $x = x_1 \ldots x_{|x|}$, we want to predict the corresponding tag sequence $y = y_1 \ldots y_{|x|}$. Let x_i denote the ith word of x, y_i denote the ith word of y, and $|x|$ denote the length of x. Note that $|y| = |x|$. For example:

$i =$	1	2	3	4	5	6	7	8	9	10
$x_i =$	Each	day	starts	with	one	or	two	lectures	by	researchers
$y_i =$	DT	NN	VBZ	IN	CD	CC	JJR	NNS	IN	NNS

We have access to many training examples like this, and our goal is to model the conditional probability of the tag sequence given the sentence, that is: $P(y \mid x)$. There are many possible choices here. To simplify the problem, let's *assume* that each element of y is conditionally independent of each other. That is, we want to model:

$$P(y \mid x) = \prod_{i=1}^{|y|} P(y_i \mid x)$$

Question 3: Model Design

a. Design a feedforward neural network to model $P(y_i \mid x)$. Identify any independence assumptions you make. Draw a diagram that illustrates how the model computes probabilities for the tag of the word “with”: What is the input, and how is the output distribution computed from the input? Write out the basic equations of the model, and explain your choices.

b. Design an RNN to model $P(y_i \mid x)$. Identify any independence assumptions you make. Draw a diagram that illustrates how the model computes probabilities for the tag of the word “with”: What is the input, and how is the output distribution computed from the input? Write out the basic equations of the model, and explain your choices.

c. [**Advanced, for now**] Can you model $P(y_i \mid x)$ without independence assumptions, using multiple RNNs?

For each question, the goal is to design a *simple* model for the distribution. You solution should only use architectures that we discussed in the first two weeks of the course. If you are aware of other architectures, you should not use them here.

Solution 3: The goal of this exercise is for you to use feedforward networks and RNNs (which you've seen for language modeling) and repurpose them for *tagging* problems, which are very common in NLP.

a. An effective design for the feedforward network is to model $P(y_i \mid x_{i-k}, \ldots, x_{i+k})$ for some fixed window size k. You might, for example, use something like this:

$$\begin{aligned} P(y_i \mid x_{i-k}, \ldots, x_{i+k}) &= \text{softmax}(\mathbf{W}\mathbf{h} + \mathbf{b}_2) \\ \mathbf{h} &= \tanh(\mathbf{V}\mathbf{x} + \mathbf{b}_1) \\ \mathbf{x} &= \text{onehot}(x_{i-k}); \ldots; \text{onehot}(x_{i+k}) \end{aligned}$$

Here, the semicolon (;) denotes concatenation. The choice of non-linearity is not important for this question, but since it asks for a feedforward network, you should have a hidden layer. This is about the simplest possible model.

Note that your solution *should not* depend on previous tags, since the question explicitly assumes that the tags are conditionally independent.

b. One design for the RNN is to model $P(y_i \mid x_1, \ldots, x_i)$. That is, the RNN reads x_1 through x_i one step at a time, and at the ith step produces a distribution for possible tags y_i. For simplicity, let's use RNN to denote a unit that receives an input and a previous hidden state, and produces a new hidden state; it can easily be replaced with an LSTM or other recurrent unit of your choice:

$$\begin{aligned} P(y_i \mid x_1 \ldots x_i) &= \text{softmax}(\mathbf{W}\mathbf{h}_i + \mathbf{b}) \\ \mathbf{h}_i &= \text{RNN}(\text{onehot}(x_i), \mathbf{h}_{i-1}) \end{aligned}$$

One thing you might notice here is that, while this model conditions on all words the left of x_i, it does not use *any* words to its right! Because of this, the feedforward might have an advantage. Can you think of a reason why you should not use k words of right context in this model?

c. A *bidirectional* RNN can model $P(y_i \mid x_1, \ldots, x_{|x|})$. It consists of two RNNs, each with its own set of parameters: one that encodes from left to right (RNN), and one that encodes from right to left (RNN$'$). Again using the semicolon to denote concatenation:

$$\begin{aligned} P(y_i \mid x_1 \ldots x_{|x|}) &= \text{softmax}(\mathbf{W}(\mathbf{h}_i; \mathbf{h}'_i) + \mathbf{b}) \\ \mathbf{h}_i &= \text{RNN}(\text{onehot}(x_i), \mathbf{h}_{i-1}) \\ \mathbf{h}'_i &= \text{RNN}'(\text{onehot}(x_i), \mathbf{h}'_{i+1}) \end{aligned}$$

A *poor* answer to any of these questions is to use a sequence-to-sequence model with attention for tagging. Why do you think that sequence-to-sequence models are inappropriate for tagging?

Pos tag is fixed.
A simple LSTM would be the better choice (and have fewer parameters and be easier to train). The model would take a word sequence as an input, and out a tag sequence. It would work exactly like the LSTM language models we discussed in the lectures, just that the **output vocabulary would be a set of labels (the PoS tags), rather than a set of words (as for language modeling).**