CSC321 Lecture 7
Neural language models

Roger Grosse and Nitish Srivastava

February 1, 2015

Overview

We've talked about neural nets and backpropagation in the abstract. Now let's see our first real example, a neural language model.

This also serves to introduce two big ideas:

- probabilistic modeling, where the network learns to predict a probability distribution
  - The cross-entropy loss function encourages the model to assign high probability to the observed data.
  - It also avoids the problem of saturation in the output units.
- dimensionality reduction using linear hidden units (i.e. reducing the number of trainable parameters)

Language modeling

Motivation: suppose we want to build a speech recognition system.

We'd like to be able to infer a likely sentence $\mathbf{s}$ given the observed speech signal $\mathbf{a}$. The generative approach is to build two components:

- An observation model, represented as $p(\mathbf{a} \mid \mathbf{s})$, which tells us how likely the sentence $\mathbf{s}$ is to lead to the acoustic signal $\mathbf{a}$.
- A prior, represented as $p(\mathbf{s})$, which tells us how likely a given sentence $\mathbf{s}$ is. E.g., it should know that "recognize speech" is more likely that "wreck a nice beach."

Language modeling

Motivation: suppose we want to build a speech recognition system.

We'd like to be able to infer a likely sentence $\mathbf{s}$ given the observed speech signal $\mathbf{a}$. The generative approach is to build two components:

- An observation model, represented as $p(\mathbf{a} \mid \mathbf{s})$, which tells us how likely the sentence $\mathbf{s}$ is to lead to the acoustic signal $\mathbf{a}$.
- A prior, represented as $p(\mathbf{s})$, which tells us how likely a given sentence $\mathbf{s}$ is. E.g., it should know that "recognize speech" is more likely that "wreck a nice beach."

Given these components, we can use Bayes' Rule to infer a posterior distribution over sentences given the speech signal:

$$p(\mathbf{s} \mid \mathbf{a}) = \frac{p(\mathbf{s})p(\mathbf{a} \mid \mathbf{s})}{\sum_{\mathbf{s}'} p(\mathbf{s}')p(\mathbf{a} \mid \mathbf{s}')}.$$

Language modeling

In this lecture, we focus on learning a good distribution $p(\mathbf{s})$ of sentences. This problem is known as language modeling.

Assume we have a corpus of sentences $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(N)}$. The maximum likelihood criterion says we want our model to maximize the probability our model assigns to the observed sentences. We assume the sentences are independent, so that their probabilities multiply.

$$\max_p \prod_{i=1}^{N} p(\mathbf{s}^{(i)}).$$

Language modeling

In maximum likelihood training, we want to maximize $\prod_{i=1}^{N} p(\mathbf{s}^{(i)})$.

The probability of generating the whole training corpus is vanishingly small — like monkeys typing all of Shakespeare.

- The log probability is something we can work with more easily. It also conveniently decomposes as a sum:

$$\log \prod_{i=1}^{N} p(\mathbf{s}^{(i)}) = \sum_{i=1}^{N} \log p(\mathbf{s}^{(i)}).$$

- Let's use *negative* log probabilities, so that we're working with positive numbers.

Language modeling

In maximum likelihood training, we want to maximize $\prod_{i=1}^{N} p(\mathbf{s}^{(i)})$.

The probability of generating the whole training corpus is vanishingly small — like monkeys typing all of Shakespeare.

- The log probability is something we can work with more easily. It also conveniently decomposes as a sum:

$$\log \prod_{i=1}^{N} p(\mathbf{s}^{(i)}) = \sum_{i=1}^{N} \log p(\mathbf{s}^{(i)}).$$

- Let's use *negative* log probabilities, so that we're working with positive numbers.

Intuition: slightly better trained monkeys are slightly more likely to type *Hamlet*!

Neural language model

Probability of a sentence ? What does that even mean ?

Neural language model

Probability of a sentence ? What does that even mean ?
A sentence is a sequence of words $w_1, w_2, \ldots, w_M$. Using the chain rule of conditional probability, we can decompose the probability as

$$p(\mathbf{s}) = p(w_1, \ldots, w_M) = p(w_1)p(w_2 \mid w_1) \cdots p(w_M \mid w_1, \ldots, w_{M-1}).$$

Therefore, the language modeling problem is equivalent to being able to predict the next word!

Neural language model

Probability of a sentence ? What does that even mean ?
A sentence is a sequence of words $w_1, w_2, \ldots, w_M$. Using the chain rule of conditional probability, we can decompose the probability as

$$p(\mathbf{s}) = p(w_1, \ldots, w_M) = p(w_1)p(w_2 \mid w_1) \cdots p(w_M \mid w_1, \ldots, w_{M-1}).$$

Therefore, the language modeling problem is equivalent to being able to predict the next word!

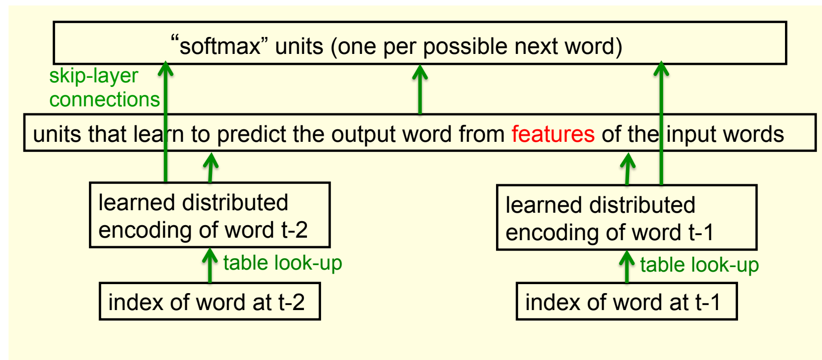
We typically make a Markov assumption, i.e. that the distribution over the next word only depends on the preceding few words. I.e., if we use a context of length 3,

$$p(w_i \mid w_1, \ldots, w_{i-1}) = p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}).$$

Now it's like a supervised prediction problem. The inputs are $(w_{i-3}, w_{i-2}, w_{i-1})$, and the target is w_i.

Neural language model

Bengio's neural language model (from Geoff's lecture)



Neural language model

When we train a neural language model, is that supervised or unsupervised learning? Does it have elements of both?

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?
Consider these two functions -

- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

Here t and y are probabilities, so then lie in $[0, 1]$.

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?
Consider these two functions -

- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

Here t and y are probabilities, so then lie in $[0, 1]$.
If $y = t = 0$ or $y = t = 1$, both loss functions are zero.

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?
Consider these two functions -

- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

Here t and y are probabilities, so then lie in $[0, 1]$.
If $y = t = 0$ or $y = t = 1$, both loss functions are zero.
Suppose $t = 1$. Consider these two situations -

- The model predicts $y = 0.01$.
- The model predicts $y = 0.0001$.

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?
Consider these two functions -

- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

Here t and y are probabilities, so then lie in $[0, 1]$.
If $y = t = 0$ or $y = t = 1$, both loss functions are zero.
Suppose $t = 1$. Consider these two situations -

- The model predicts $y = 0.01$.
- The model predicts $y = 0.0001$.

The first case is a LOT better than the second. (We are a hundred times less likely to pick 0). So we need to be a lot more unhappy about the second case, than the first. Which loss function does that ?

A good loss function

Squared error worked nicely for regression.
What loss function should we use when predicting probabilities ?
Consider these two functions -

- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

Here t and y are probabilities, so then lie in $[0, 1]$.
If $y = t = 0$ or $y = t = 1$, both loss functions are zero.
Suppose $t = 1$. Consider these two situations -

- The model predicts $y = 0.01$. $C_{\mathrm{sq}} = 0.99^2$, $C_{\mathrm{CE}} = \log(100)$
- The model predicts $y = 0.0001$. $C_{\mathrm{sq}} = 0.9999^2$, $C_{\mathrm{CE}} = \log(10000)$

The first case is a LOT better than the second. (We are a hundred times less likely to pick 0). So we need to be a lot more unhappy about the second case, than the first. Which loss function does that ?

Question 1: Squared error vs. cross-entropy

Geoff said that squared error is the wrong cost function to use with logistic or softmax outputs because of saturation. Let's analyze this in the case of a logistic unit.

Recall the logistic activation function:

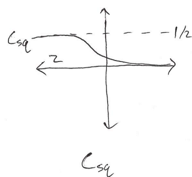
$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

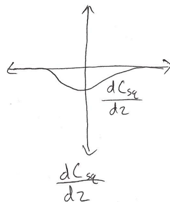
Suppose our target is $t = 1$. Sketch C and $\mathrm{d}C/\mathrm{d}z$ as a function of z for the cost functions:

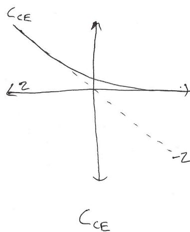
- **Squared error:** $C_{\mathrm{sq}} = \frac{1}{2}(y - t)^2$
- **Cross-entropy:** $C_{\mathrm{CE}} = -t \log y - (1 - t) \log(1 - y)$

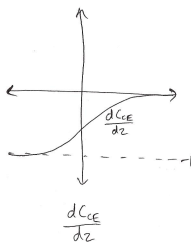
Hint: consider the behavior as $z \to \pm\infty$

Question 1: Squared error vs. cross-entropy









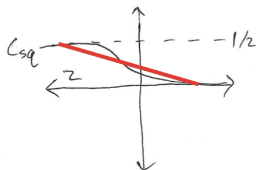
Question 1: Squared error vs. cross-entropy

The region where C_{sq} is flat for negative z is a plateau.

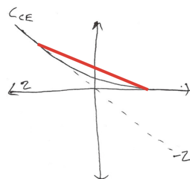
A function is convex if line segments joining points on the graph of f lie above f. Mathematically,

$$C(\lambda w_1 + (1-\lambda)w_2) \leq \lambda C(w_1) + (1-\lambda)C(w_2).$$

Convex cost functions are usually easier to optimize because there aren't any local optima or plateaux.



not convex in z



convex in z

Question 1: Squared error vs. cross-entropy

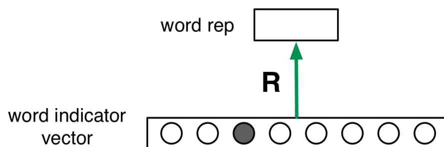
We just saw that plateaux are a problem for logistic output units with squared error loss, but not cross-entropy.

We often use logistic hidden units in multilayer neural nets. Do you think we have plateaux when training these units?

Question 2: Linear hidden units

The neural language model is the first example we've seen of an embedding layer.

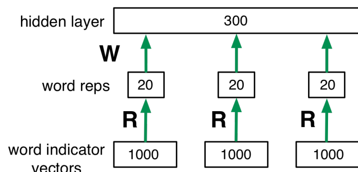
Geoff describes it as a lookup table. But we can also think of it as a linear hidden layer.

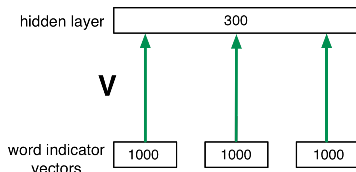


Multiplying **R** by an indicator vector selects a column of **R**.

Question 2: Linear hidden units

Here we have two architectures. Model A is similar to the neural language model, while Model B eliminates the embedding layer.

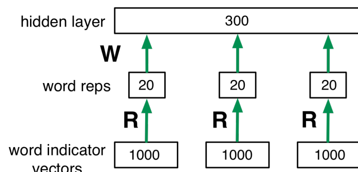


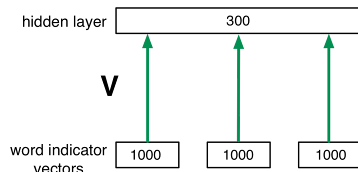


Show that Model B can compute any function that Model A can compute. If I give you weight matrices **R** and **W** for Model 1, compute an equivalent weight matrix **V** for Model B.

Question 2: Linear hidden units

Here we have two architectures. Model A is similar to the neural language model, while Model B eliminates the embedding layer.





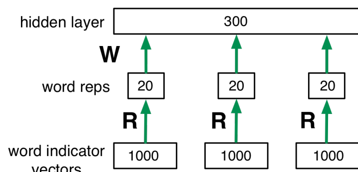
Show that Model B can compute any function that Model A can compute. If I give you weight matrices **R** and **W** for Model 1, compute an equivalent weight matrix **V** for Model B.

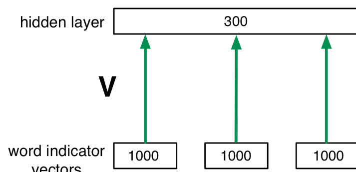
Solution: Model B can match Model A's function by setting $\mathbf{V} = \mathbf{W}\tilde{\mathbf{R}}$, where

$$\tilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R} & & \\ & \mathbf{R} & \\ & & \mathbf{R} \end{pmatrix}.$$

Question 2: Linear hidden units

Here we have two architectures. Model A is similar to the neural language model, while Model B eliminates the embedding layer.





Assume there are 1000 words in the vocabulary, 20-dimensional embeddings, and 300 hidden units.

1. Compute the number of trainable parameters in the layers shown for each model.
2. Approximately compute the number of arithmetic operations required to compute the hidden activations for a given input. **Assume we compute the matrix-vector products explicitly** rather than using a lookup table.
   - Hint: how many operations are needed to compute a matrix-vector product?

Question 2: Linear hidden units

Number of trainable parameters:

- **Model A**: **R** is a matrix of size 20×1000 and **W** is of size 300×60, for $20 \cdot 1000 + 300 \cdot 60 = 38{,}000$ learnable parameters
- **Model B**: **V** is a matrix of size 300×3000, for 900,000 learnable parameters.

Number of computations:

- A matrix-vector product where the matrix is of size $M \times N$ involves approximately MN adds and MN multiplies.
- **Model A**: Three matrix-vector products of size 20×1000 and one of size 300×60 for a total of 78,000 multiples and adds.
- **Model B**: One matrix-vector product of size 300×3000, for 900,000 multiplies and adds.
- Note: since the inputs are indicator vectors, in practice you would use a lookup table. But we asked about explicit multiplications since most situations where we use linear hidden layers won't involve indicator vectors.

Question 2: Linear hidden units

To recap:

- Nonlinear hidden units allow a network to compute more complex functions.
- Linear hidden units don't increase the expressive power of a network. But they can introduce a bottleneck which reduces the number of learnable parameters or the number of computations required.
  - Corresponds to a low-rank factorization of the weight matrix