Natural Language Processing Workshop

Amy Hemmeter, MSA Class of ‘18
Artificial Intelligence Engineer, Interactions Digital Roots

Reasons NLP is hard

Reasons NLP is hard

1. Words are not numbers

Reasons NLP is hard

1. Words are not numbers
2. Input can be different lengths

Reasons NLP is hard

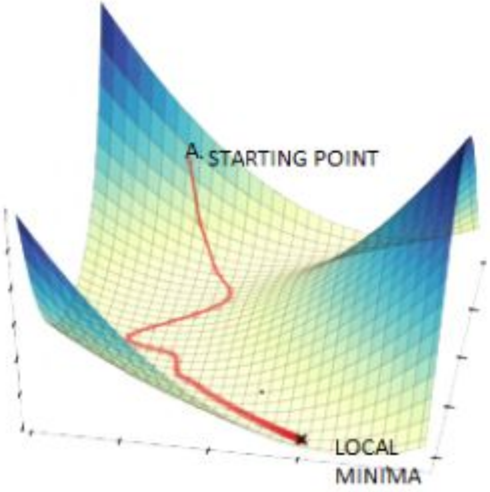
1. Words are not numbers

2. **Input can be different lengths***

*https://github.com/blester125/A2D-NLP-Talk-Feb-27-2020

Gradient Descent Review (I'm sure of it this time!)

- How to find the minimum of your loss function
- Uses the derivative of the loss function with respect to your parameters to take a step in a direction towards the minimum
- How most machine learning algorithms work
- The rate at which you move down this slope is called the learning rate

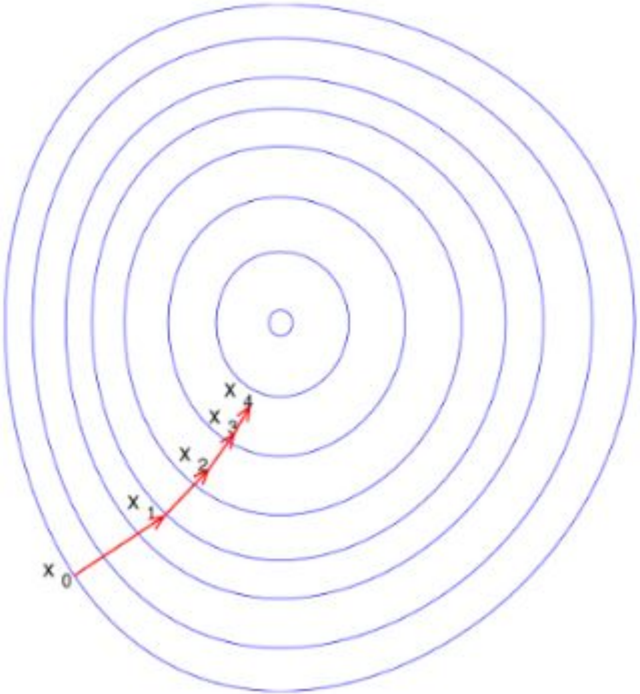


Picture from Data Science Central,
https://www.datasciencecentral.com/profiles/blogs/alternatives-to-the-gradient-descent-algorithm

Gradient Descent

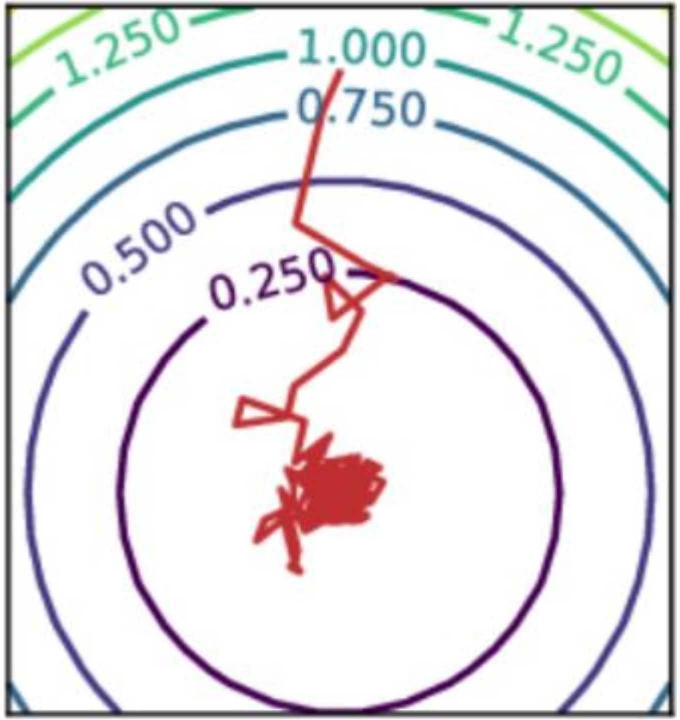
- Three ways of running gradient descent on your training data:
  - Full gradient descent
  - Stochastic gradient descent
  - Batched gradient descent

Full Gradient Descent



- This process gives us the “true gradient”
- Calculate the loss for each example in the training set and then use all of these losses to calculate the gradient
  - You need to run each example before you update any parameters
  - Very slow and compute-heavy!!

Stochastic Gradient Descent



Pros:

- You update your parameters after each step
- Much faster
- No need for any of the techniques we're about to talk about

Cons

- Your gradient could be wrong
- What works well for one example could hurt another example

Batched Gradient Descent

- Batching provides a happy medium
- You get to update your parameters more often
- You can get a better approximation of your gradient
- Minibatch size is now a hyperparameter for your deep learning model
- And now you need to learn some tricks….

Binary Logistic Regression

- We have two vectors of the same size representing the weights and the features respectively
- We take the sum of the features weighted by the weight to create a logit score
- In this example we’re going to ignore the activation function for now

Binary Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$
$$w = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix}$$
$$s =$$

Binary Logistic Regression

$$f = [\ 1 \quad 2 \quad 3 \quad 4\ ]$$
$$w = [\ 5 \quad 6 \quad 7 \quad 8\ ]$$
$$s = 5$$

Binary Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$
$$w = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix}$$
$$s = 17$$

Binary Logistic Regression

$$f = [\ \boxed{1 \ \ 2 \ \ 3 \ \ 4}\ ]$$
$$w = [\ \boxed{5 \ \ 6 \ \ 7 \ \ 8}\ ]$$
$$s = 70$$

Multi-Class Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} \quad \end{bmatrix}$$

Multi-Class Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 5 & \end{bmatrix}$$

Multi-Class Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 17 & \end{bmatrix}$$

Multi-Class Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & \end{bmatrix}$$

Multi-Class Logistic Regression

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & 110 \end{bmatrix}$$

Batched Multi-Class Logistic Regression

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} \\ \\ \end{bmatrix}$$

Batched Multi-Class Logistic Regression

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & \\ & \end{bmatrix}$$

Batched Multi-Class Logistic Regression

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & 110 \\ & \end{bmatrix}$$

Batched Multi-Class Logistic Regression

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & 110 \\ 382 & \end{bmatrix}$$

Batched Multi-Class Logistic Regression

$$F = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$W = \begin{bmatrix} 5 & 9 \\ 6 & 10 \\ 7 & 11 \\ 8 & 12 \end{bmatrix}$$

$$s = \begin{bmatrix} 70 & 110 \\ 382 & \boxed{614} \end{bmatrix}$$

Why did I show you that math?

- To give you an idea for how we can vectorize the inputs for the data -- this is the source of the speed gains
- To help you visualize so that padding will make more sense to you
- To show you that in order for that math to work, your feature inputs need to be the same size

A Fly in the Ointment

Consider the following sentences:

[The dog ran very fast]

[The cat slept]

What is padding?

- Meaningless tokens we can insert into our batches for sentences that are not the same length

[The dog ran very fast]

[The cat slept <PAD> <PAD>]

“dog” : [1 2 4 3 5]

“dog” : [1 2 4 3 5]

“cat” : [1 3 4 3 5]

“dog” : [1 2 4 3 5]

“cat” : [1 3 4 3 5]

<PAD> : ?

“dog” : [1 2 4 3 5]

“cat” : [1 3 4 3 5]

<PAD> : [0 0 0 0 0]

Lengths vector

- Because zeros can still mess things up for us, we need to keep track of the lengths of our original input in a lengths vector.

[The dog ran very fast]

[The cat slept <PAD> <PAD>]

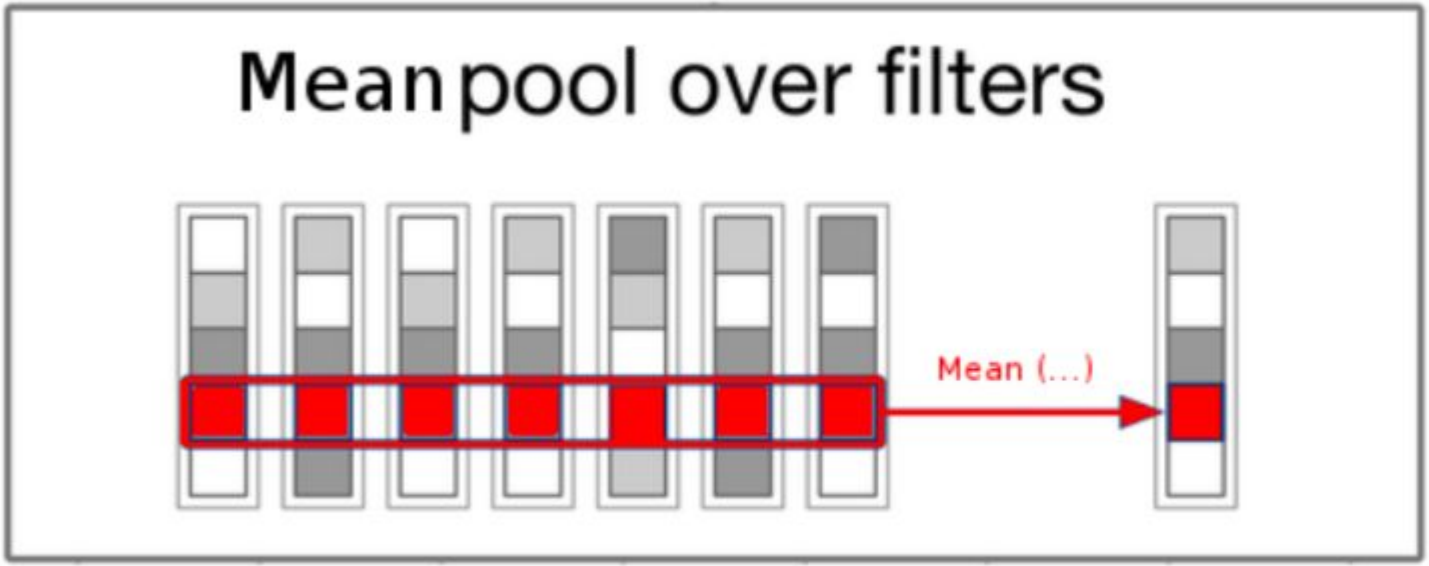
L = [5 3]

Quiz

Knowing what we know about padding and stochastic gradient descent, why don’t we need to use padding for stochastic gradient descent?

You have to be careful

Mean Pooling



Mean Pooling

$$\begin{bmatrix} 1 & 10 & 8 & 17 & 13 & 17 \end{bmatrix} = \frac{66}{6} = 11.0$$

Mean Pooling

$$\begin{bmatrix} 1 & 10 & 8 & 17 & 13 & 17 \end{bmatrix} = \frac{66}{6} = 11.0$$

$$\begin{bmatrix} 22 & 24 & 9 & 13 \end{bmatrix} = \frac{68}{4} = 17.0$$

Mean Pooling

$$\begin{bmatrix} 1 & 10 & 8 & 17 & 13 & 17 \end{bmatrix} = \frac{66}{6} = 11.0$$

$$\begin{bmatrix} 22 & 24 & 9 & 13 \end{bmatrix} = \frac{68}{4} = 17.0$$

$$\begin{bmatrix} 5 & 4 & 8 & 9 & 10 & 34 \\ 6 & 3 & 1 & 4 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{66}{6} \\ \frac{68}{6} \end{bmatrix} = \begin{bmatrix} 11.0 \\ 11.\bar{3} \end{bmatrix}$$

Mean Pooling

```
>>> x
array([[ 1, 10,  8, 17, 13, 17],
       [22, 24,  9, 13,  0,  0]])
>>> np.mean(x, axis=1)
array([11.        , 11.33333333])
>>> lengths
array([6, 4])
>>> np.sum(x, axis=1) / lengths
array([11., 17.])
```

Now that we have a more holistic view of the pipeline...

Let’s look at some code!

Things you already know that are helpful!

-

Things you already know that are helpful!

- CLUSTERING

Things you already know that are helpful!

- CLUSTERING
- Cosine similarity

Things you already know that are helpful!

- CLUSTERING
- Cosine similarity
- Naive Bayes, Logistic Regression, Support Vector Machines

What I would teach you if I had more time

- Contextual embeddings
- Sequence tagging
- Named Entity Recognition
- Dependency Parsing
- Machine Translation and other applications of Encoder-Decoder Models
- Transformers
- Attention and self-attention
- And more!!!

Mead-Baseline

- MEAD-Baseline -- open-source tool for Modeling, Experimentation and Development that provides good baselines against which to compare results of your experiment
- It’s also a great tool for production-quality models
- Does a lot of the work we’ve discussed in this class for you

Any last questions? :)