CS224n: NLP with Deep Learning

Lecture 3: Word Window Classification, Neural Networks and Calculus

---

Classification review/intro

- Our learned line between red and green is our classifier

Softmax classifier

$$\boxed{p(y|x) = \frac{exp(W_y \cdot x)}{\sum_{c=1}^{C} exp(W_c \cdot x)}}$$

2 steps:

1. the y^{th} row of W is for the y^{th} class
   Multiplying it by x gives us a score :
2. Then we apply the softmax function to get a probability distribution

Objective: minimize our negative log-probability $-logp(y|x)$

Cross-entropy loss

- $p =$ true probability distribution
- $q =$ computed probability distribution

Cross-entropy:

$$H(p,q) = -\sum_{c=1}^{C} p(c)\, log\, q(c)$$

Cross-entropy loss:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

Neural nets intro

- Logistic regression, SVM, Naïve Bayes are all simple classifiers:
  they just draw a line in some high-dimensional space
- But oftentimes we'd like a more sophisticated classifier, especially on natural data

NLP Deep Learning

- We're simultaneously changing both the weights, and the words vector representations as we're learning
- We're optimising both of them at once
- Representation Learning

We can interpret the word embeddings as the first hidden layer of a network that takes as inputs the 1-hot encoded vectors of the words

Biological Neuron analogy

Dendrites -> Enough Signal -> Body of the Neuron -> Axon -> Dendrites of other Neurons

A little bit of Neural Net History

- 50s: Perceptron
- 80s-90s: 1 hidden layer neural net

We need to introduce a non-linearity for our neural net to learn something interesting, otherwise we just get another linear function

---

Named Entity Recognition (NER)

Task = find the names of things, and classify them

Possible usages:

- track the names of companies and people
- answers of questions often are named entities
- information often is just an association of 2 named entities

Difficulties

- Hard to determine the boundaries of an entity
- Hard to know if something is an entity
- Hard to know the class of a new entity

Binary true vs corrupted word window clasification

Idea:
Classify a word in its context window

We could design a classifier that classifies the middle word according to the context window words: this way, the order information would be preserved

(as opposite to taking the sum vector of all the context words)

NER of center-of-window word

We want to have a system that returns a score if there is a Location name in the middle

1. Concatenate the context window words' vector representations
2. Pass it through a first non-linearity
3. Multiply it by a big `U^` matrix

Putting our extra hidden layer allows us to calculate **non-linear** things, from our input vector, such as:

- The first word is 'museum'
- AND the 2nd word is 'in'

Then, our 3rd word (here, our center word), is a Location name.

Matrix calculus intro

We will use matrix calculus to calculate our gradients

Gradient

$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_n)$

Gradient:

$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial \mathbf{x_1}}, \frac{\partial f}{\partial \mathbf{x_2}}, \ldots, \frac{\partial f}{\partial \mathbf{x_n}} \right]$

Jacobian Matrix

The Jacobian Matrix is the Generalization of the Gradient

$\mathbf{f}(\mathbf{x}) = [f_1(x_1, \ldots, xn), \ldots, f_m(x_1, \ldots, xn)]$

Jacobian:

$$\boxed{\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{f_i}}{\partial \mathbf{x_j}} \right)_{i,j}}$$

$

\frac{\partial \mathbf{f}} {\partial \mathbf{x}}

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$

Chain Rule

For 1-variable functions: **Multiply Derivatives**

For mutliple variable functions: **Multiply JACOBIANS**

$\mathbf{h} = f(\mathbf{z})$

$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$

$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \cdots$

Examples

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \operatorname{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \quad \text{(Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

Practicing on a small example

1. Break up the equations into simple smaller pieces
   (**Recommended**: define new variables for each step!)

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boxed{\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}}$$
$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \; \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \; \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$
$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z})) \boldsymbol{I}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$
$$\boxed{\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}}$$

Now, we want to calculate the derivative with respect to W

We notice that, during the calculation of the derivative, most of the terms are the same as with respect to *b*!

δ is the error signal

Re-using Computation

- Suppose we now want to compute $\dfrac{\partial s}{\partial \boldsymbol{W}}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{\delta}$$

$$\boldsymbol{\delta} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

δ is local error signal

We are using the shape convention:
The shape of the gradient is the same shape as that of the inputs

Derivative with respect to Matrix

- Remember $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$
  - $\boldsymbol{\delta}$ is going to be in our answer
  - The other term should be $\boldsymbol{x}$ because $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- It turns out $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$

δ is local error signal at z
x is local input signal

71

Shape contradictions

It appears that we are trying to follow 2 contradictory shape conventions:

1. The Jacobian form, which is easy for calculations
2. The parameter convention, where we want our gradient's shape to match our parameter's, so that we can use it in the gradient descent update

What shape should derivatives be?

- Two options:
- 1. Use Jacobian form as much as possible, reshape to follow the convention at the end:
    - What we just did. But at the end transpose $\frac{\partial s}{\partial \boldsymbol{b}}$ to make the derivative a column vector, resulting in $\boldsymbol{\delta}^T$
- 2. Always follow the convention
    - Look at dimensions to figure out when to transpose and/or reorder terms.