Natural Language Processing
Lecture 6

Lecture by Dr. Kfir Bar
Typeset by Steven Karas

2019-04-30
Last edited 20:57:50 2019-04-30

Disclaimer These notes are based on the lectures for the course Natural Language Processing, taught by Dr. Kfir Bar at IDC Herzliyah in the spring semester of 2018/2019. Sections may be based on the lecture slides prepared by Dr. Kfir Bar.

1 Agenda

- Intro to machine learning
- Deep learning (LSTM, etc)
- Word embedding

2 Intro to machine learning

2.1 Neural Networks

Last time, we stopped off at describing logistic regression as a neuron. Logistic regression is defined as a set of inputs $x_1, \ldots, x_n$ linearly combined with weights $w_1, \ldots, w_n$. We compute $z = w^\intercal x$ and then pass it off to the activation function (usually sigmoid):

$$a = \frac{1}{1 = e^{-w^\intercal x}}$$

We typically provide a dummy input 1 with weight $b = w_0$.

To extend this to a neural network, we can use the same inputs to train more neurons at the same time and train for multiple classifiers. This can be generalized as:

$$A = Wx$$

Where W is the weight matrix.

We can add extra layers that use the output of previous layers as their input. Depending on the number of layers and their shape, we refer to this as deep learning. The last layer has as many neurons as we need outputs (e.g. for credit card fraud detection, there is a single neuron at the end).

Denote $W^{[i]}$ as the weight matrix and $a^{[i]}$ as the output for the i-th layer. $x = a^{[0]}$.

2.1.1 Training

$$w_i' \leftarrow w_i - \eta \frac{dL}{dw_i}$$

This is called gradient descent. The current state of the art is called ADAM, which adjusts the learning rate η automatically while training.

2.1.2 Different activation functions

Sigmoid is an older activation function, and great for some purposes, but the derivative changes lots. In practice, ReLU (sets negative values to 0) gives us the nonlinear activation we need, but the derivative is stable, so the loss function is easier/quicker to train on.

2.1.3 Loss functions

Maximum squared error is a good loss function to start with, but when we have a closed set of labels to assign and not a generic output value, we can use the logistic regression loss function, which is much better in practice:

For each instance, if $y_i = 1$ then we want $\Pr[y_i \mid x_i] = h_\theta(x_i)$ to be maximized. If $y_i = 0$, then we want $\Pr[y_i \mid x_i] = 1 - h_\theta(x_i)$ to be maximized. So we can write this more compactly as:

$$\Pr[y_i \mid x_i] = (h_\theta(x_i))^{y_i} \, (1 - h_\theta(x_i))^{1-y_i}$$

And as such the loss function is negative log likelihood:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \log \Pr[y_i \mid x_i] = -\frac{1}{m} \sum_{i=1}^{m} y_i \log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i))$$

2.2 Computational graph

A computational graph is a graph where edges represent the flow of data, and vertices represent operations on their inputs. A neural network can be represented as a computational graph. Tensorflow and pytorch both allow us to define computational graphs and execute them quickly and easily.

Forwarding data through the computational graph gives us the result of the operations. We can also pull data backwards through the graph to get the derivatives. In the literature, it is common to denote the loss of w as dw. This is usually already implemented by the library as two opaque operations: forward and backward.

2.3 PyTorch

Similar to numpy, but builds a computational graph. The basic building block is `torch.Tensor`. There are many prebuilt helpers that let us build neural networks at a very high level.

It is strongly recommended to follow the PyTorch tutorial

3 Word Embeddings

Generally speaking, words make bad inputs to neural networks. So we want to convert words into vectors. Since the 70s, the wordnet project provides a dictionary of english words with many different meanings. However taxonomies such as this are subjective and can't be complete. In 2013, word2vec created the breakthrough by embedding words into a vector by their semantic similarity. Application wise, we can use embeddings for word similarity, such as search query suggestions.

3.1 Building the embeddings

A naive approach uses a one-hot encoding, by defining a vector of size $|\Sigma|$. This doesn't encode word similarity at all, and isn't a very good approach.

Distributional similarity encodes words based on the adjacent words. We typically use a defined window size and construct a co-occurrence matrix. The scoring for co-occurrence can be simple frequency, or something more advanced such as tf-idf. We then use dimensionality reduction such as PCA or t-SNE to get the final embeddings.

3.1.1 Principal Component Analysis

Find the eigenvectors of the covariance matrix and use them as the new basis. Drop dimensions with small eigenvalues.

3.1.2 Latent Semantic Analysis

Singular Value Decomposition (SVD) on W gives us $W = U\Sigma V^\intercal$, where U holds the eigenvectors of the context similarity matrix $W^\intercal W$, V holds the eigenvectors of the word similarity matrix $WW^\intercal$, and Σ holds the square roots of the eigenvalues of W.

LSA runs SVD and only keeps the top k dimensions. In practice, $k \in [300, 5000]$, using whole documents as context, and more advanced counting techniques such as tf-idf (term frequency - inverse document frequency) or PPMI (positive pointwise mutual information).

3.2 Word similarity

There are many similarity measures, but in this case we typically use Euclidean distance (inner product). This tells us how close the meanings are to each other.

4 Next week

Due to the national holiday, there will not be a lecture next week. Notice will be sent regarding a makeup lecture.

References

[1] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[2] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., 2009.

[3] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.