NLP Programming Tutorial 10 - Neural Networks

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Prediction Problems

Given $x$, predict $y$
Example we will use:

• Given an introductory sentence from Wikipedia
• Predict whether the article is about a person

<table>
<thead>
<tr>
<th>Given</th>
<th>Predict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gonso was a Sanron sect priest (754-827) in the late Nara and early Heian periods.</td>
<td>Yes!</td>
</tr>
<tr>
<td>Shichikuzan Chigogataki Fudomyoo is a historical site located at Magura, Maizuru City, Kyoto Prefecture.</td>
<td>No!</td>
</tr>
</tbody>
</table>

• This is binary classification (of course!)
Linear Classifiers

\[ y = \text{sign}(\mathbf{w} \cdot \varphi(x)) \]
\[ = \text{sign} \left( \sum_{i=1}^{I} w_i \cdot \varphi_i(x) \right) \]

- \( x \): the input
- \( \varphi(x) \): vector of feature functions \( \{\varphi_1(x), \varphi_2(x), \ldots, \varphi_I(x)\} \)
- \( \mathbf{w} \): the weight vector \( \{w_1, w_2, \ldots, w_I\} \)
- \( y \): the prediction, +1 if “yes”, -1 if “no”
  - (\text{sign}(v) \text{ is } +1 \text{ if } v \geq 0, -1 \text{ otherwise})
Example Feature Functions: Unigram Features

- Equal to “number of times a particular word appears”

\[ x = \text{A site, located in Maizuru, Kyoto} \]

\[
\begin{align*}
\varphi_{\text{unigram "A"}}(x) &= 1 \\
\varphi_{\text{unigram "site"}}(x) &= 1 \\
\varphi_{\text{unigram "","}}(x) &= 2 \\
\varphi_{\text{unigram "located"}}(x) &= 1 \\
\varphi_{\text{unigram "in"}}(x) &= 1 \\
\varphi_{\text{unigram "Maizuru"}}(x) &= 1 \\
\varphi_{\text{unigram "Kyoto"}}(x) &= 1 \\
\varphi_{\text{unigram "the"}}(x) &= 0 \\
\varphi_{\text{unigram "temple"}}(x) &= 0 \\
\end{align*}
\]

The rest are all 0

- For convenience, we use feature names (\( \varphi_{\text{unigram "A"}} \)) instead of feature indexes (\( \varphi_1 \))
**Calculating the Weighted Sum**

\[ x = \text{A site, located in Maizuru, Kyoto} \]

<table>
<thead>
<tr>
<th>Unigram</th>
<th>( \phi )</th>
<th>( w )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>“A”</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“site”</td>
<td>1</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>“located”</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“Maizuru”</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“,”</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“in”</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“Kyoto”</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>“priest”</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>“black”</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \cdots = -3 \rightarrow \text{No!} \]
The Perceptron

- Think of it as a “machine” to calculate a weighted sum

\[
\text{sign}\left(\sum_{i=1}^{I} w_i \cdot \phi_i(x)\right)
\]

\[
\begin{align*}
\phi_{\text{"A"}} &= 1 \\
\phi_{\text{"site"}} &= 1 \\
\phi_{\text{"located"}} &= 1 \\
\phi_{\text{"Maizuru"}} &= 1 \\
\phi_{\text{"","}} &= 2 \\
\phi_{\text{"in"}} &= 1 \\
\phi_{\text{"Kyoto"}} &= 1 \\
\phi_{\text{"priest"}} &= 0 \\
\phi_{\text{"black"}} &= 0
\end{align*}
\]
Problem: Linear Constraint

- Perceptron cannot achieve high accuracy on non-linear functions

\[
\begin{array}{cc}
X & O \\
O & X
\end{array}
\]
Neural Networks

- Neural networks connect multiple perceptrons together

 Motivation: Can express non-linear functions
Example:

- Build two classifiers:

  \[ \varphi(x_1) = \{-1, 1\} \quad \varphi(x_2) = \{1, 1\} \]

  \[ \varphi(x_3) = \{-1, -1\} \quad \varphi(x_4) = \{1, -1\} \]
Example:

- These classifiers map the points to a new space

\[ \varphi(x_1) = \{-1, 1\} \quad \varphi(x_2) = \{1, 1\} \quad y(x_3) = \{-1, 1\} \]

\[ \varphi(x_3) = \{-1, -1\} \quad \varphi(x_4) = \{1, -1\} \quad y(x_1) = \{-1, -1\} \quad y(x_2) = \{1, -1\} \quad y(x_4) = \{-1, -1\} \]
Example:

- In the new space, examples are classifiable!

\[ \varphi(x_1) = \{-1, 1\} \quad \varphi(x_2) = \{1, 1\} \]

\[ \varphi(x_3) = \{-1, -1\} \quad \varphi(x_4) = \{1, -1\} \]
Example:

- Final neural network:
Representing a Neural Network

- Assume network is fully connected and in layers
- Each perceptron:
  - A layer ID
  - A weight vector

network = [(1, w₀),
           (1, w₁),
           (1, w₂),
           (2, w₃)]
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer

\[
\begin{align*}
\varphi_{\text{"A"}} &= 1 \\
\varphi_{\text{"site"}} &= 1 \\
\varphi_{\text{"located"}} &= 1 \\
\varphi_{\text{"Maizuru"}} &= 1 \\
\varphi_{\text{"in"}} &= 2 \\
\varphi_{\text{"Kyoto"}} &= 1 \\
\varphi_{\text{"priest"}} &= 0 \\
\varphi_{\text{"black"}} &= 0
\end{align*}
\]
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer

\[
\begin{align*}
\phi_{"A"} &= 1 \\
\phi_{"site"} &= 1 \\
\phi_{"located"} &= 1 \\
\phi_{"Maizuru"} &= 1 \\
\phi_{"in"} &= 2 \\
\phi_{"Kyoto"} &= 1 \\
\phi_{"priest"} &= 0 \\
\phi_{"black"} &= 0
\end{align*}
\]
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer

\[
\begin{align*}
\varphi_{\text{“A”}} &= 1 \\
\varphi_{\text{“site”}} &= 1 \\
\varphi_{\text{“located”}} &= 1 \\
\varphi_{\text{“Maizuru”}} &= 1 \\
\varphi_{\text{“,”}} &= 2 \\
\varphi_{\text{“in”}} &= 1 \\
\varphi_{\text{“Kyoto”}} &= 1 \\
\varphi_{\text{“priest”}} &= 0 \\
\varphi_{\text{“black”}} &= 0
\end{align*}
\]
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer

\[
\begin{align*}
\phi_{"A"} &= 1 \\
\phi_{"site"} &= 1 \\
\phi_{"located"} &= 1 \\
\phi_{"Maizuru"} &= 1 \\
\phi_{"\"\"} &= 2 \\
\phi_{"in"} &= 1 \\
\phi_{"Kyoto"} &= 1 \\
\phi_{"priest"} &= 0 \\
\phi_{"black"} &= 0
\end{align*}
\]
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer
Review:
Pseudo-code for Perceptron Prediction

```
PREICT_ONE(w, phi)
    score = 0
    for each name, value in phi # score = w*φ(x)
        if name exists in w
            score += value * w[name]
    if score >= 0
        return 1
    else
        return -1
```
Pseudo-Code for NN Prediction

\[
\text{PREDICT\_NN} (\text{network}, \phi) \\
y = [\phi, \emptyset, \emptyset \ldots] \quad \# \text{activations for each layer} \\
\text{for each node } i:\ \\
\quad \text{layer, weight} = \text{network}[i] \\
\quad \# \text{predict the answer with the previous perceptron} \\
\quad \text{answer} = \text{PREDICT\_ONE} (\text{weight}, y[\text{layer}-1]) \\
\quad \# \text{save this answer as a feature for the next layer} \\
\quad y[\text{layer}][i] = \text{answer} \\
\text{return the answer for the last perceptron}
\]
Neural Network Activation Functions

- Previously described NN uses step function

\[ y = \text{sign}(w \cdot \varphi(x)) \]

- Step function is not differentiable → use tanh

\[ y = \text{tanh}(w \cdot \varphi(x)) \]

**Python:**
```python
from math import tanh
tanh(x)
```
Learning a Perceptron w/ tanh

• First, calculate the error:

\[ \delta = y' - y \]

\[ \text{correct tag} \quad \text{system output} \]

• Update each weight with:

\[ w \leftarrow w + \lambda \cdot \delta \cdot \varphi(x) \]

• Where \( \lambda \) is the learning rate

• (For step function perceptron \( \delta = -2 \) or \(+2, \lambda = 1/2\)
Problem: Don't Know Correct Answer!

• For NNs, only know correct tag for last layer

\[
\begin{align*}
\phi \text{"A"} &= 1 \\
\phi \text{"site"} &= 1 \\
\phi \text{"located"} &= 1 \\
\phi \text{"Maizuru"} &= 1 \\
\phi \text{"."} &= 2 \\
\phi \text{"in"} &= 1 \\
\phi \text{"Kyoto"} &= 1 \\
\phi \text{"priest"} &= 0 \\
\phi \text{"black"} &= 0
\end{align*}
\]

\[
\begin{align*}
y' &= ? \\
y &= 1 \\
y &= 1 \\
y' &= 1 \\
y &= -1 \\
y' &= ? \\
y &= 1
\end{align*}
\]
Answer: Back-Propagation

- Pass error backwards along the network

\[ \delta_j = \sum_i \delta_i w_{j,i} \]

- Also consider gradient of tanh

\[ d \tanh(\varphi(x) * w) = 1 - (\varphi(x) * w)^2 = 1 - y_j^2 \]

- Combine:

\[ \delta_j = (1 - y_j^2) \sum_i \delta_i w_{j,i} \]
Back Propagation Code

```python
UPDATE_NN(network, phi, y')
    create array δ
    calculate y using PREDICT_NN
    for each node j in reverse order:
        if j is the last node
            δ_j = y' - y_j
        else
            δ_j = (1 - y_j^2) \sum_i δ_i w_{j,i}
        for each node j:
            layer, w = network[j]
            for each name, val in y[layer-1]:
                w[name] += λ * δ_j * val
```
Training process

create \textit{network}
randomize \textit{network} weights
for / iterations
   for each labeled pair $x$, $y$ in the data
      $\phi = \text{CREATE\_FEATURES}(x)$
      \text{UPDATE\_NN}(w, \phi, y)

- For previous perceptron, we initialized weights to zero
- In NN: randomly initialize weights (so not all perceptrons are identical)
Exercise
Exercise (1)

- **Write two programs**
  - train-nn: Creates a neural network model
  - test-nn: Reads a neural network model
- **Test** train-nn
  - Input: test/03-train-input.txt
  - Use one iteration, one hidden layer, two hidden nodes
  - Calculate updates by hand and make sure they are correct
Exercise (2)

- **Train** a model on data/titles-en-train.labeled
- **Predict** the labels of data/titles-en-test.word
- **Grade** your answers
  - script/grade-prediction.py data-en/titles-en-test.labeled your_answer
- **Compare:**
  - With a single perceptron/SVM classifiers
  - With different neural network structures
Thank You!