Part-of-Speech Tagging
+
Neural Networks 2

CS 287
Review: Bilinear Model

Bilinear model,

\[ \hat{y} = f((x^0 W^0)W^1 + b) \]

- \( x^0 \in \mathbb{R}^{1 \times d_0} \) start with one-hot.
- \( W^0 \in \mathbb{R}^{d_0 \times d_{in}}, \ d_0 = |\mathcal{F}| \)
- \( W^1 \in \mathbb{R}^{d_{in} \times d_{out}}, \ b \in \mathbb{R}^{1 \times d_{out}} \); model parameters

Notes:
- Bilinear parameter interaction.
- \( d_0 >> d_{in}, \) e.g. \( d_0 = 10000, \ d_{in} = 50 \)
Review: Bilinear Model: Intuition

\[(x^0 W^0) W^1 + b\]
Review: Window Model

**Goal:** predict $t_5$.

- Windowed word model.

\[ w_1 \ w_2 \ [w_3 \ w_4 \ w_5 \ w_6 \ w_7] \ w_8 \]

- $w_3, w_4$; left context
- $w_5$; Word of interest
- $w_6, w_7$; right context
- $d_{\text{win}}$; size of window ($d_{\text{win}} = 5$)
Review: Dense Windowed BoW Features

- $f_1, \ldots, f_{d_{\text{win}}}$ are words in window
- Input representation is the concatenation of embeddings

$$\mathbf{x} = [v(f_1) \ v(f_2) \ \ldots \ v(f_{d_{\text{win}}})]$$

Example: Tagging

$$w_1 \ w_2 \ [w_3 \ w_4 \ w_5 \ w_6 \ w_7] \ w_8$$

$$\mathbf{x} = [v(w_3) \ v(w_4) \ v(w_5) \ v(w_6) \ v(w_7)]$$

Rows of $\mathbf{W}^1$ encode position specific weights.
We are doing tagging with a windowed bilinear model with hinge-loss and no capitalization features. The model has $d_{\text{win}} = 5$, $d_{\text{in}} = 50$, $d_{\text{out}} = 40$, and vocabulary size 10000.

We are given the input window:

The dog walked to the

Unfortunately we incorrectly classify walked as NN as opposed to VP, in a bilinear model with a hinge-loss.

What is the maximum number of parameters that receive a non-zero gradient?
\[ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ \end{bmatrix} \begin{bmatrix} w_{1,1}^0 & \cdots & w_{0,d_{in}}^0 \\ w_{the,1}^0 & \cdots & w_{the,d_{in}}^0 \\ \vdots \\ w_{dog,1}^0 & \cdots & w_{dog,d_{in}}^0 \\ \vdots \\ w_{walked,1}^0 & \cdots & w_{walked,d_{in}}^0 \\ \vdots \\ w_{to,1}^0 & \cdots & w_{to,d_{in}}^0 \\ \vdots \\ w_{the,1}^0 & \cdots & w_{the,d_{in}}^0 \\ \vdots \\ w_{d_0,1}^0 & \cdots & w_{d_0,d_{in}}^0 \end{bmatrix} \begin{bmatrix} \cdots & \cdots & w_{1,NN}^1 & \cdots & w_{1,VP}^1 & w_{0,d_{out}}^1 \\ \cdots & \cdots \\ \vdots \\ w_{d_{in},0}^1 \cdots & w_{d_{in},NN}^1 & \cdots & w_{d_{in},VP}^1 & w_{d_{in},d_{out}}^1 \end{bmatrix} \]

\[
W^0 = 5 \times d_{in} \\
W^1 = d_{in} \times 2
\]
Consider the following windowed model, and assume for now a linear model.

\[
[w_1 \ \text{the} \ w_3 \ w_4 \ w_5]
\]

- What information do we have about the tag of \( w_3 \)?

- What weight should the features values associated with \( \text{the} \) in position \( w_2 \) take?
Next Consider the following windowed model, and assume for now a linear model.

\[
\begin{bmatrix}
w_1 & w_2 & w_3 & \text{dog} & w_5
\end{bmatrix}
\]

- What information do we have about the tag of \(w_3\)?
- What weight should the features values associated with dog in position \(w_4\) take?
Part-of-Speech Tagging 3

Now finally consider the following windowed model, and assume for now a linear model.

\[
[w_1 \text{ the } w_3 \text{ dog } w_5]
\]

- What information do we have about the tag of \( w_3 \)?
- What weight would we want if we combined both the features values?
Contents

Neural Networks

Backpropagation
Neural Network

One-layer multi-layer perceptron architecture,

\[ \text{NN}_{\text{MLP}_1}(x) = g(xW^1 + b^1)W^2 + b^2 \]

- \( xW + b; \) perceptron
- \( x \) is the dense representation in \( \mathbb{R}^{1 \times d_{\text{in}}} \)
- \( W^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}, b^1 \in \mathbb{R}^{1 \times d_{\text{hid}}}; \) first affine transformation
- \( W^2 \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{out}}}, b^2 \in \mathbb{R}^{1 \times d_{\text{out}}}; \) second affine transformation
- \( g: \mathbb{R}^{d_{\text{hid}} \times d_{\text{hid}}} \) is an activation non-linearity (often pointwise)
- \( g(xW^1 + b^1) \) is the hidden layer
Schematic
Non-Linearities: 0/1

0/1 function:

\[ 0/1(t) = \mathbf{1}(t > 0) \]

- \( 0 \) if \( t \leq 0 \)
- \( 1 \) if \( t > 0 \)

Intuition: On, if above a threshold
Exercise

Input layer to $NN_{MLP_1}$ is the sparse indicator features of the word at each position.

- Design a network to recognize

  $[w_1 \text{ the } w_3 \text{ dog } w_5]$

- Design a network to recognize where $w_2$ is not the

  $[w_1 w_2 w_3 \text{ dog } w_5]$
Feature Conjunctions

Many NLP tasks require conjunctive features, examples

- Sequence-based taggers look at last two-part of speech tags.

- Chinese part-of-speech taggers look at first character and last tag.

- Higher-level models (parses) look at tags of words and distances apart (example)

For some natural language tasks, conjunctions are painstakingly hard.

- NNs: Capacity to learn conjunctions and feature combinations.

- Also possible with other convex models such as SVMs
Feature Conjunctions

Many NLP tasks require conjunctive features, examples

- Sequence-based taggers look at last two-part of speech tags.
- Chinese part-of-speech taggers look at first character and last tag.
- Higher-level models (parses) look at tags of words and distances apart (example)

For some natural language tasks, conjunctions are painstakingly hard.

- NNs: Capacity to learn conjunctions and feature combinations.
- Also possible with other convex models such as SVMs
Simple Antecedent/Pairwise Features Not Discriminative

E.g., is [Lexus sales] the antecedent of [their sales]?

- Common pairwise features: String/Head Match, Sentences Between, Mention-Antecedent Numbers/Heads/Genders, etc.

$$\phi_p([\text{their sales}],[\text{Lexus sales}]) = \begin{cases} \text{string-match}=false \\
\text{head-match}=true \\
\text{sentences-between}=0 \\
\text{ment-ant-numbers}=\text{plur., plur.} \\
\vdots \end{cases}$$
Dealing with the Feature Problem

Finding discriminative features is a major challenge for coreference systems [Fernandes et al. 2012; Durrett and Klein 2013]

Typical to define (or search for) feature conjunction-schemes to improve predictive performance [Fernandes et al. 2012; Durrett and Klein 2013; Björkelund and Kuhn 2014]. For instance:

- string-match\((x, y) \land \text{type}(x) \land \text{type}(y)\) [Durrett and Klein 2013], where

  \[
  \text{type}(x) = \begin{cases} 
  \text{Nom.} & \text{if } x \text{ is nominal} \\
  \text{Prop.} & \text{if } x \text{ is proper} \\
  \text{citation-form}(x) & \text{if } x \text{ is pronominal}
  \end{cases}
  \]

- substring-match(head\(x\), y) \land substring-match\((x, \text{head}(y)) \land \text{coarse-type}(y) \land \text{coarse-type}(x)\) [Björkelund and Kuhn 2014]

Not just a problem for Mention Ranking systems!
Non-Linearities: 0/1

0/1 function:

\[ 0/1(t) = 1(t > 0) \]

- Issue: No gradient anywhere
Non-Linear Functions: Sigmoid

Logistic sigmoid function:

\[ \sigma(t) = \frac{1}{1 + \exp(-t)} \]

- \( \sigma((xW^1 + b^1)_i) \)
- Intuition: Each hidden dimension ("neuron") is result of logistic regression.
Other Non-Linearities: ReLU

Rectified Linear Unit:

\[ \text{ReLU}(t) = \max\{0, t\} \]

- Intuition: Each hidden-unit gives activation margin
- No gradient (saturation) when below 0.
Saturation: Intuition

\[ x_1, x_2, x_3, \ldots, x_d, h_1, h_2, h_d_{\text{hid}}, z_1, z_2, z_3, \ldots, z_d_{\text{out}} \]
Other Non-Linearities: Tanh

Hyperbolic Tangent:

\[ \text{tanh}(t) = \frac{\exp(t) - \exp(-t)}{\exp(t) + \exp(-t)} \]

▶ Intuition: Similar to sigmoid, but range between 0 and -1.
Other Non-Linearities: Hard Tanh

Hyperbolic Tangent:

\[
\text{hardtanh}(t) = \begin{cases} 
-1 & t < -1 \\ 
t & -1 \leq t \leq 1 \\ 
1 & t > 1 
\end{cases}
\]

Intuition: Similar to sigmoid, but range between 0 and -1.
Other Non-Linearities: Cube

Cube non-linearity (directly encourage parameter interaction):

\[ \text{cube}(t) = t^3 \]

- Intuition: Directly encourage higher-order interactions.
Tagging from Scratch
MLP1 is a universal approximator

*Can approximate with any desired non-zero amount of error a family of functions that include all continuous functions on a closed and bounded subset of $\mathbb{R}^n$, and any function mapping from any finite dimensional discrete space to another (YG)*

Caveats:

- Does not give size of hidden layer.
- Does not specify how hard this is to learn.
Deep Neural Networks (DNNs)

Can stack MLPs, create deep fully connected networks,

\[ NN_{MLP1}(x) = g(xW^1 + b^1)W^2 + b^2 \]

\[ NN_{MLP2}(x) = g(NN_{MLP1}(x)W^1 + b^1)W^2 + b^2 \]

- Can have multiple hidden layers, etc.
- Benefit: may be able to find better function
- Known to be harder to train (although other approaches)
We will discuss many other neural network layers,

- convolutional
- attention-based
- gated layers
- ...
Highway Network

\[ y : \text{output from CharCNN} \]

**Multilayer Perceptron**

\[ z = g(Wy + b) \]

**Highway Network**

(Srivastava, Greff, and Schmidhuber 2015)

\[ z = t \odot g(W_H y + b_H) + (1 - t) \odot y \]

\( W_H, b_H : \text{Affine transformation} \)

\( t = \sigma(W_T y + b_T) : \text{transform gate} \)

\( 1 - t : \text{carry gate} \)

Hierarchical, adaptive composition of character \( n \)-grams.
Highway Network

\[ z = t \odot g(W_H y + b_H) + (1 - t) \odot y \]

Input from CharCNN

Input to LSTM

Input from CharCNN

\[ g(W_H y + b_H) \]

\[ \sigma(W_T y + b_T) \]
Contents

Neural Networks

Backpropagation
Sequential Neural Network

Sequential neural networks consist of a series of composed functions, Consider a vector-valued parameterized functions $f_1, \ldots, f_k$ where

- $f_i(x; \theta_i) : \mathbb{R}^{n_{i-1}} \mapsto \mathbb{R}^{n_i}$; function
- $\theta \in \mathbb{R}^{d_i}$; function parameters

Consider a scalar-valued loss function $L(y, \hat{y})$ where

- $L(y, *) : \mathbb{R}^{n_k} \mapsto \mathbb{R}$; loss for input
Backpropagation

- **Forward Step (f-prop):**
  
  Compute
  
  $$L(f_k(\ldots f_1(x^0)))$$

  Saving intermediary values
  
  $$f_i(\ldots f_1(x^0)))$$

- **Backward Step (b-prop):**
  
  $$\frac{\partial L}{\partial f_i(\ldots f_1(x^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial f_i(\ldots f_1(x^0))} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}$$

  $$\frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial \theta_i} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}$$
Backpropagation

- **Forward Step (f-prop):**
  
  Compute
  
  \[
  L(f_k(\ldots f_1(x^0)))
  \]
  
  Saving intermediary values
  
  \[
  f_i(\ldots f_1(x^0)))
  \]

- **Backward Step (b-prop):**

\[
\frac{\partial L}{\partial f_i(\ldots f_1(x^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial f_i(\ldots f_1(x^0))} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}
\]

\[
\frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial \theta_i} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}
\]
Backpropagation

- **Forward Step (f-prop):**
  
  Compute
  
  \[
  L(f_k(\ldots f_1(x^0)))
  \]

  Saving intermediary values
  
  \[
  f_i(\ldots f_1(x^0))
  \]

- **Backward Step (b-prop):**
  
  \[
  \frac{\partial L}{\partial f_i(\ldots f_1(x^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial f_i(\ldots f_1(x^0))} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}
  \]

  \[
  \frac{\partial L}{\partial \theta_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\ldots f_1(x^0))_j}{\partial \theta_i} \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))_j}
  \]
Backpropagation: Data flow

\[ f_i(\ldots f_1(x^0)) \]

\[ f_{i+1}(f_i(\ldots f_1(x^0))) \]

\[ f_{i+1}(\ast; \theta_{i+1}) \]

\[ \frac{\partial L}{\partial \theta_{i+1}} \]

\[ \frac{\partial L}{\partial f_i(\ldots f_1(x^0))} \]

\[ \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))} \]
Torch Implementation

Torch uses declarative unit-based specification of NN

- Every function is a represented as a unit.

- Responsibilities:
  1. Expose any parameters $\theta_{i+1}$ as tensors
  2. Compute $f_{i+1}(x, \theta_{i+1})$ (fprop)
  3. Compute any necessary state needed for bprop
  4. Compute chain-rule given $\frac{\partial L}{\partial f_{i+1}(...)f_1(x^0)}$ and $f_i(\ldots f_1(x^0))$
  5. Compute parameter gradient $\frac{\partial L}{\partial \theta_{i+1}}$

- Contract: forward will always be called before backward.
Torch Units

input
\( f_i(\ldots f_1(x^0)) \)

weights
\( f_{i+1}(\ast; \theta_{i+1}) \)

self.output
\( f_{i+1}(f_i(\ldots f_1(x^0))) \)

self.gradInput
\( \frac{\partial L}{\partial f_i(\ldots f_1(x^0))} \)

self.gradWeight
\( \frac{\partial L}{\partial \theta_{i+1}} \)

gradOutput
\( \frac{\partial L}{\partial f_{i+1}(\ldots f_1(x^0))} \)
Loss Criterions

\[ L(\hat{y}, y) \]

- **Prediction**: \( \hat{y} \)
- **Loss**: \( L(\hat{y}, y) \)
- **Weights**: \( L(\ast, \ast) \)
- **Target**: \( y \)
- **Self.gradInput**: \( \frac{\partial L}{\partial \hat{y}} \)
Torch Internals

[Web]
Torch Units: Batch

\[ f_i(\ldots f_1(X^0)) \]

input

\[ f_{i+1}(\ast; \theta_{i+1}) \]

self.output

\[ f_{i+1}(f_i(\ldots f_1(X^0))) \]

self.gradInput

\[ \frac{\partial L}{\partial f_i(\ldots f_1(X^0))} \]

self.gradWeight

\[ \sum \frac{\partial L}{\partial \theta_{i+1}} \]

gradOutput

\[ \frac{\partial L}{\partial f_{i+1}(\ldots f_1(X^0))} \]
Loss Criterions

$$L(\ast, \ast)$$

loss

$$L(Y, \hat{Y})$$

target

$$Y$$

prediction

$$\hat{Y}$$

self.gradInput

$$\frac{\partial L}{\partial \hat{Y}}$$
Today

- Benefits of neural networks
- Training neural networks

Next time: Pretraining and word embeddings