Natural Language Processing

Info 159/259
Lecture 4: Text classification 2 (Jan 27, 2022)

David Bamman, UC Berkeley
Binary logistic regression

$$P(y = 1 \mid x, \beta) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

output space $\mathcal{Y} = \{0, 1\}$
Multiclass logistic regression

\[ P(Y = y \mid X = x; \beta) = \frac{\exp(x^\top \beta_y)}{\sum_{y' \in \mathcal{Y}} \exp(x^\top \beta_{y'})} \]

output space \( \mathcal{Y} = \{1, \ldots, K\} \)
As a discriminative classifier, logistic regression doesn’t assume features are independent.

Its power partly comes in the ability to create richly expressive features without the burden of independence.

We can represent text through features that are not just the identities of individual words, but any feature that is scoped over the entirety of the input.
We want to find the value of $\beta$ that leads to the highest value of the conditional log likelihood:

$$\ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta)$$

---

**Algorithm 2** Logistic regression stochastic gradient descent

1: Data: training data $x \in \mathbb{R}^F, y \in \{0, 1\}$
2: $\beta = 0^F$
3: while not converged do
4:   for $i = 1$ to $N$ do
5:     $\beta_{t+1} = \beta_t + \alpha (y_i - \hat{p}(x_i)) x_i$
6:   end for
7: end while
L2 regularization

\[
\ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta) - \eta \sum_{j=1}^{F} \beta_j^2
\]

- We can do this by changing the function we're trying to optimize by adding a penalty for having values of \( \beta \) that are high.

- This is equivalent to saying that each \( \beta \) element is drawn from a Normal distribution centered on 0.

- \( \eta \) controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data).
L1 regularization

\[ \ell(\beta) = \sum_{i=1}^{N} \log P(y_i \mid x_i, \beta) - \eta \sum_{j=1}^{F} |\beta_j| \]

- L1 regularization encourages coefficients to be exactly 0.

- \(\eta\) again controls how much of a penalty to pay for coefficients that are far from 0 (optimize on development data)
https://www.forbes.com/sites/kevinmurnane/2016/04/01/what-is-deep-learning-and-how-is-it-useful
History of NLP

- Foundational insights, 1940s/1950s
- Two camps (symbolic/stochastic), 1957-1970
- Four paradigms (stochastic, logic-based, NLU, discourse modeling), 1970-1983
- Empiricism and FSM (1983-1993)
- Field comes together (1994-1999)
- Neural networks (~2014–today)
“Word embedding” in NLP papers

Data from ACL papers in the ACL Anthology (https://www.aclweb.org/anthology/)
Neural networks in NLP

• Language modeling [Mikolov et al. 2010]

• Text classification [Kim 2014; Iyyer et al. 2015]

• Syntactic parsing [Chen and Manning 2014, Dyer et al. 2015, Andor et al. 2016]

• CCG super tagging [Lewis and Steedman 2014]


• Dialogue agents [Sordoni et al. 2015, Vinyals and Lee 2015, Ji et al. 2016]

• (for overview, see Goldberg 2017, 1.3.1)
Neural networks

• Discrete, high-dimensional representation of inputs (one-hot vectors) -> low-dimensional “distributed” representations.

• Static representations -> contextual representations, where representations of words are sensitive to local context.

• Non-linear interactions of input features

• Multiple layers to capture hierarchical structure
Neural network libraries

TensorFlow
Theano
Keras
PyTorch
DyNet
Logistic regression

\[
P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}
\]

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>β</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>bad</td>
<td>1</td>
<td>-1.7</td>
</tr>
<tr>
<td>movie</td>
<td>0</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Calculate the derivative of some loss function with respect to parameters we can change, update accordingly to make predictions on training data a little less wrong next time.
Logistic regression

$$P(\hat{y} = 1) = \frac{1}{1 + \exp \left( - \sum_{i=1}^{F} x_i \beta_i \right)}$$
Feedforward neural network

- Input and output are mediated by at least one hidden layer.
*For simplicity, we’re leaving out the bias term, but assume most layers have them as well.*
A neural network diagram with nodes labeled $x_1$, $x_2$, $x_3$, $h_1$, and $h_2$ connected by edges. Below the diagram is a table with rows labeled "not", "bad", and "movie" and columns labeled $x$, $W$, $V$, and $y$. The table values include:

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$W$</th>
<th>$V$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>not</td>
<td>1</td>
<td>-0.5</td>
<td>1.3</td>
<td>4.1</td>
</tr>
<tr>
<td>bad</td>
<td>1</td>
<td>0.4</td>
<td>0.08</td>
<td>-0.9</td>
</tr>
<tr>
<td>movie</td>
<td>0</td>
<td>1.7</td>
<td>3.1</td>
<td>1</td>
</tr>
</tbody>
</table>
The hidden nodes are completely determined by the input and weights.

\[ h_j = f \left( \sum_{i=1}^{F} x_i W_{i,j} \right) \]
\[ h_1 = f \left( \sum_{i=1}^{F} x_i W_{i,1} \right) \]
Activation functions

\[ \sigma(z) = \frac{1}{1 + \exp(-z)} \]
Logistic regression

We can think about logistic regression as a neural network with no hidden layers.

\[
P(\hat{y} = 1) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}
\]

\[
P(\hat{y} = 1) = \sigma\left(\sum_{i=1}^{F} x_i \beta_i\right)
\]

We can think about logistic regression as a neural network with no hidden layers.
Activation functions

$$tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$
Activation functions

ReLU($z$) = \text{max}(0, \ z)$
• ReLU and tanh are both used extensively in modern systems.

• Sigmoid is useful for final layer to scale output between 0 and 1, but is not often used in intermediate layers.
\[
W \quad V
\]

\[
\begin{align*}
    h_1 &= \sigma \left( \sum_{i=1}^{F} x_i W_{i,1} \right) \\
    h_2 &= \sigma \left( \sum_{i=1}^{F} x_i W_{i,2} \right) \\
    \hat{y} &= \sigma [V_1 h_1 + V_2 h_2]
\end{align*}
\]
we can express \( y \) as a function only of the input \( x \) and the weights \( W \) and \( V \)

\[
\hat{y} = \sigma \left[ V_1 \left( \sigma \left( \sum_{i}^{F} x_i W_{i,1} \right) \right) + V_2 \left( \sigma \left( \sum_{i}^{F} x_i W_{i,2} \right) \right) \right]
\]
This is hairy, but differentiable

\[
\hat{y} = \sigma \left[ V_1 \left( \sigma \left( \sum_{i}^{F} x_i W_{i,1} \right) \right) \right] + V_2 \left( \sigma \left( \sum_{i}^{F} x_i W_{i,2} \right) \right)
\]

Backpropagation: Given training samples of \(<x,y>\) pairs, we can use stochastic gradient descent to find the values of \(W\) and \(V\) that minimize the loss.
Neural networks are a series of functions chained together.

The loss is another function chained on top.

\[
xW \rightarrow \sigma (xW) \rightarrow \sigma (xW) V \rightarrow \sigma (\sigma (xW) V)
\]

\[
\log (\sigma (\sigma (xW) V))
\]
Chain rule

\[
\frac{\partial}{\partial V} \log (\sigma (\sigma (xW) V)) = \frac{\partial \log (\sigma (\sigma (xW) V))}{\partial \sigma (\sigma (xW) V)} \frac{\partial \sigma (\sigma (xW) V)}{\partial \sigma (xW) V} \frac{\partial \sigma (xW) V}{\partial V}
\]

Let's take the likelihood for a single training example with label \(y = 1\); we want this value to be as high as possible.
Chain rule

\[
\frac{\partial}{\partial \sigma(hV)} \log(\sigma(hV)) \frac{\partial \sigma(hV)}{\partial hV} \frac{\partial hV}{\partial V} = \frac{1}{\sigma(hV)} \times \sigma(hV) \times (1 - \sigma(hV)) \times h
\]

\[
= (1 - \sigma(hV))h
\]

\[
= (1 - \hat{y})h
\]
Neural networks

- Tremendous flexibility on design choices (exchange feature engineering for model engineering)
- Articulate model structure and use the chain rule to derive parameter updates.
Neural network structures

Output one real value; sigmoid function for output gives single probability between 0 and 1
Neural network structures

Multiclass: output 3 values, only one = 1 in training data; softmax function for output gives probability between 0 and 1 for each class (all class probabilities sum to 1); classes compete with each other.
Neural network structures

output 3 values, several = 1 in training data; sigmoid function for each output gives probability of presence of that label; classes do not compete with each other since multiple can be present together.

nsubj_walks

nsubj_says

dobj_hits

1 glad

1 happy

0 sad
Regularization

- Increasing the number of parameters = increasing the possibility for overfitting to training data
Regularization

• L2 regularization: penalize W and V for being too large

• Dropout: when training on a $<x, y>$ pair, randomly remove some node and weights.

• Early stopping: Stop backpropagation before the training error is too small.
Deeper networks

\[ W_1 \quad W_2 \quad V \]

\[
\begin{align*}
x_1 \quad h_1 \quad h_2 \quad h_2 \quad \cdots \quad h_2 \quad h_2 \quad h_2 \quad y \\
x_2 \quad h_2 \quad h_2 \quad h_2 \quad \cdots \quad h_2 \quad h_2 \quad h_2 \quad y \\
x_3 \quad h_2 \quad h_2 \quad h_2 \quad \cdots \quad h_2 \quad h_2 \quad h_2 \quad y \\
x_3 \quad h_2 \quad h_2 \quad h_2 \quad \cdots \quad h_2 \quad h_2 \quad h_2 \quad y \\
\end{align*}
\]
Densely connected layer

\[ h = \sigma(xW) \]
Convolutional networks

- With convolution networks, the *same* operation is (i.e., the same set of parameters) is applied to *different* regions of the input
2D Convolution

| 0 0 0 0 0 |
| 0 1 1 1 0 |
| 0 1 1 1 0 |
| 0 1 1 1 0 |
| 0 1 1 1 0 |
| 0 0 0 0 0 |

blurring
1D Convolution

convolution $K$

\[
\begin{array}{ccc}
1/3 & 1/3 & 1/3 \\
\hline
0 & 1 & 3 & -1 & 4 & 2 & 0 \\
\hline
1\frac{1}{2} & 1 & 2 & 1\frac{1}{2} & 2 \\
\end{array}
\]
Convolutional networks

\[ h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3) \]
Indicator vector

- Every token is a V-dimensional vector (size of the vocab) with a single 1 identifying the word.
\[ h_1 = \sigma(x_1W_1 + x_2W_2 + x_3W_3) \]
For indicator vectors, we're just adding these numbers together

\[ h_1 = \sigma(W_{1,x_1^{id}} + W_{2,x_2^{id}} + W_{3,x_3^{id}}) \]

(Where \(x_i^{id}\) specifies the location of the 1 in the vector — i.e., the vocabulary id)
For dense input vectors (e.g., embeddings), full dot product

\[ h_1 = \sigma(x_1 W_1 + x_2 W_2 + x_3 W_3) \]
Pooling

- Down-samples a layer by selecting a single point from some set
- **Max-pooling** selects the largest value
Global pooling

- Down-samples a layer by selecting a single point from some set
- **Max-pooling over time** (global max pooling) selects the largest value over an entire sequence
- Very common for NLP problems.
Convolutional networks

This defines one filter.

convolution  max pooling
We can specify multiple filters; each filter is a separate set of parameters to be learned.

\[ h_1 = \sigma(x^\top W) \in \mathbb{R}^4 \]
Convolutional networks

- With max pooling, we select a single number for each filter over all tokens.
- (e.g., with 100 filters, the output of max pooling stage = 100-dimensional vector)
- If we specify multiple filters, we can also scope each filter over different window sizes.
Zhang and Wallace 2016, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification"
Higher-order ngrams are much more informative than just unigrams (e.g., “I don’t like this movie” [“I”, “don’t”, “like”, “this”, “movie”])

We can think about a CNN as providing a mechanism for detecting important (sequential) ngrams without having the burden of creating them as unique features.