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
Neural network structures

Multiclass: output 3 values, only one = 1 in training data
Neural network structures

output 3 values, several = 1 in training data
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

\[
\begin{align*}
W_1 & \quad W_2 & \quad V \\
X_1 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & y \\
X_2 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & y \\
X_3 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & y \\
X_3 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & y \\
X_3 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & y \\
X_3 & \rightarrow & h_1 & \rightarrow & h_2 & \rightarrow & 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

Blurring

[Matrix]

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

1D Convolution

convolution $K$

$X$

$$
\begin{array}{ccc}
1/3 & 1/3 & 1/3 \\
\hline
0 & 1 & 3 & -1 & 4 & 2 & 0 \\
1\frac{1}{3} & 1 & 2 & 1\frac{2}{3} & 2
\end{array}
$$
I hated it

I really hated it

I really hated it

Convolutional networks

\[ h_1 = f(I, \text{hated}, \text{it}) \]

\[ h_2 = f(\text{it}, I, \text{really}) \]

\[ h_3 = f(\text{really, hated, it}) \]

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

\[ h_2 = \sigma(x_3 W_1 + x_4 W_2 + x_5 W_3) \]

\[ h_3 = \sigma(x_5 W_1 + x_6 W_2 + x_7 W_3) \]
Indicator vector

- Every token is a V-dimensional vector (size of the vocab) with a single 1 identifying the word
$$h_1 = \sigma (x_1 W_1 + x_2 W_2 + x_3 W_3)$$
$$h_2 = \sigma (x_3 W_1 + x_4 W_2 + x_5 W_3)$$
$$h_3 = \sigma (x_5 W_1 + x_6 W_2 + x_7 W_3)$$
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 \( x_n^{id} \) 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

- Very common for computer vision problems.
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.
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.
CNN as important ngram detector

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

<table>
<thead>
<tr>
<th>N-grams</th>
<th>Unique Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>unigrams</td>
<td>50921</td>
</tr>
<tr>
<td>bigrams</td>
<td>451,220</td>
</tr>
<tr>
<td>trigrams</td>
<td>910,694</td>
</tr>
<tr>
<td>4-grams</td>
<td>1,074,921</td>
</tr>
</tbody>
</table>

Unique ngrams (1-4) in Cornell movie review dataset
Keras

• We’ll be using keras to implement several neural architectures over the next few weeks

• Today: Functional models
Sequential

- Useful for models of limited complexity where the input to every layer is the output of the previous layer.
model = Sequential()

model.add(Embedding(input_dim=vocab_size, output_dim=word_embedding_dim, weights=[[embeddings]], trainable=False))

model.add(Conv1D(filters=50, kernel_size=2, strides=1, padding="same", activation="tanh"))

model.add(GlobalMaxPooling1D())

model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))
Functional

- Useful for complex models where a single layer can have multiple inputs/output that don’t need to be linearly related to each other
- Layers here are functions that give you more control over the inputs and outputs
word_sequence_input = Input(shape=(None,), dtype='int32')

word_embedding_layer = Embedding(vocab_size, word_embedding_dim, weights=[embeddings], trainable=False)

embedded_sequences = word_embedding_layer(word_sequence_input)
• Explore CNN using keras