Neural Networks

Instructor: Yoav Artzi

Slides adapted from Dan Klein, Dan Jurafsky, Chris Manning, Michael Collins, Luke Zettlemoyer, Yejin Choi, and Slav Petrov
Overview

• Introduction to Neural Networks
• Word representations
• NN Optimization tricks
Some History

• Neural network algorithms date to the 80’s – Originally inspired by early neuroscience
• Historically slow, complex, and unwieldy
• Now: term is abstract enough to encompass almost any model – but useful!
• Dramatic shift in last 3-4 years away from MaxEnt (linear, convex) to “neural net” (non-linear architecture, non-convex)
The “Promise”

• Most ML works well because of human-designed representations and input features
• ML becomes just optimizing weights
• Representation learning attempts to automatically learn good features and representations
• Deep learning attempts to learn multiple levels of representation of increasing complexity/abstraction
Neuron

- Neural networks comes with their terminological baggage

- Parameters:
  - Weights: $w_i$ and $b$
  - Activation function

- If we drop the activation function, reminds you of something?
Neural Network
Neural Network

input layer

hidden layer 1

hidden layer 2

output layer
Matrix Notation

\[ \mathbf{a} \in \mathbb{R}^{1 \times 3} \]
\[ \mathbf{W}' \in \mathbb{R}^{3 \times 4} \]
\[ \mathbf{W}'' \in \mathbb{R}^{4 \times 2} \]
\[ \mathbf{b}' \in \mathbb{R}^{1 \times 4} \]
\[ \mathbf{b}'' \in \mathbb{R}^{1 \times 2} \]

Learned parameters

\[ \mathbf{h} = \mathbf{aW}' + \mathbf{b}' \]
\[ \mathbf{o} = \mathbf{hW}'' + \mathbf{b}'' \]
\[ = (\mathbf{aW}' + \mathbf{b}')\mathbf{W}'' + \mathbf{b}'' \]

No activation/non-linearity function
Matrix Notation

\[ h_1 = a_1 W'_{11} + a_2 W'_{21} + a_3 W'_{31} + b'_1 \]

\[ o_1 = h_1 W''_{11} + h_2 W''_{21} + h_3 W''_{31} + h_4 W''_{41} + b''_1 \]

\[ h_2 = a_1 W'_{14} + a_2 W'_{24} + a_3 W'_{34} + b'_4 \]

\[ o_2 = h_1 W''_{12} + h_2 W''_{22} + h_3 W''_{32} + h_4 W''_{42} + b''_1 \]
Activation Functions

• Entry-wise function: \( f : \mathbb{R} \rightarrow \mathbb{R} \)

<table>
<thead>
<tr>
<th>Activation Function</th>
<th>Formula</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td>( \sigma(x) = \frac{1}{1+e^{-x}} )</td>
<td>![Sigmoid Graph]</td>
</tr>
<tr>
<td>tanh</td>
<td>( \tanh(x) )</td>
<td>![tanh Graph]</td>
</tr>
<tr>
<td>ReLU</td>
<td>( \max(0, x) )</td>
<td>![ReLU Graph]</td>
</tr>
<tr>
<td>Leaky ReLU</td>
<td>( \max(0.1x, x) )</td>
<td>![Leaky ReLU Graph]</td>
</tr>
<tr>
<td>Maxout</td>
<td>( \max(w^T_1 x + b_1, w^T_2 x + b_2) )</td>
<td>![Maxout Graph]</td>
</tr>
<tr>
<td>ELU</td>
<td>( \begin{cases} x &amp; x \geq 0 \ \alpha(e^x - 1) &amp; x &lt; 0 \end{cases} )</td>
<td>![ELU Graph]</td>
</tr>
</tbody>
</table>
Neurons and Other Models

• A single neuron is a perceptron
• Strong connection to MaxEnt – how?
From MaxEnt to Neural Nets

• Vector form MaxEnt:

\[ P(y|x; w) = \frac{e^{w^\top \phi(x,y)}}{\sum_{y'} e^{w^\top \phi(x,y')}} \]

• For two classes:

\[ P(y_1|x; w) = \frac{e^{w^\top \phi(x,y_1)}}{e^{w^\top \phi(x,y_1)} + e^{w^\top \phi(x,y_2)}} \]

\[ = \frac{1}{1 + e^{w^\top \phi(x,y_2) - \phi(x,y_2)}} = f(w^\top z) \]

\[ z = \phi(x, y_1) - \phi(x, y_2) \]
From MaxEnt to Neural Nets

- Vector form MaxEnt:
  \[ P(y|x; w) = \frac{e^{w^\top \phi(x,y)}}{\sum_{y'} e^{w^\top \phi(x,y')}} \]

- For two classes:
  \[ P(y_1|x; w) = \frac{1}{1 + e^{-w^\top z}} = f(w^\top z) \]

- Neuron:
  - Add an “always on” feature for class prior \( \rightarrow \) bias term \( b \)
  \[ h_{w,b}(z) = f(w^\top z + b) \]
  \[ f(u) = \frac{1}{1 + e^{-u}} \]
Neural Net = Several MaxEnt Models

• Feed a number of MaxEnt models → vector of outputs
• And repeat …
Neural Net = Several MaxEnt Models

- But: how do we tell the hidden layer what to do?
  - Learning will figure it out
How to Train?

• No hidden layer:
  – Supervised
  – Just like MaxEnt

• With hidden layers:
  – Latent units $\rightarrow$ not convex
  – What do we do?
    • Back-propagate the gradient
    • About the same, but no guarantees
Probabilistic Output from Neural Nets

• What if we want the output to be a probability distribution over possible outputs?
• Normalize the output activations using softmax:

\[ y = \text{softmax}(o) \]

\[ \text{softmax}(o_i) = \frac{\exp(o_i)}{\sum_{j=1}^{k} \exp(o_j)} \]

  – Where \( o \) is the output layer
  – Usually: no non-linearity before softmax
Word Representations

• So far, atomic symbols:
  – “hotel”, “conference”, “walking”, “___ing”
• But neural networks take vector input
• How can we bridge the gap?
• One-hot vectors

hotel = [0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
conference = [0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]

  – Dimensionality:
    • Size of vocabulary
    • 20K for speech
    • 500K for broad-coverage domains
    • 13M for Google corpora
Word Representations

• One-hot vectors:

\[
\begin{align*}
\text{hotel} &= [0 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\text{conference} &= [0 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0] \\
\text{hotels} &= [0 \ 0 \ 0 \ 0 \ \ldots \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]
\end{align*}
\]

– Problems?
– Information sharing?
  • “hotel” vs. “hotels”
Word Embeddings

• Each word is represented using a dense low-dimensional vector
  – Low-dimensional $\ll$ vocabulary size
• If trained well, similar words will have similar vectors
• How to train? What objective to maximize?
  – Soon …
Word Embeddings as Features

• Example: sentiment classification
  – very positive, positive, neutral, negative, very negative

• Feature-based models: bag of words

• Any good neural net architecture?
  – Concatenate all the vectors
    • Problem: different document → different length
  – Instead: sum, average, etc.
Neural Bag-of-words

Deep Averaging Networks

IMDB sentiment analysis

BOW + fancy smoothing + SVM

NBOW + DAN

[Iyyer et al. 2015; Wang and Manning 2012]
Classify Word Pair

• Goal: build a classifier that given a pair of words, classify if they are the full name of a person or not
• The classifier is a multi-layer-perceptron with three layers
• Make a drawing!
• Write the matrix notation, including dimensionality of matrices (choose as you wish, and as needed)
• What are the parameters to be learned

Inputs: $x_l, x_r$
Input vocabulary: $\mathcal{V}$
Embedding function: $\phi: \mathcal{V} \rightarrow \mathbb{R}^{256}$
Weight matrices: $\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$
Bias vectors: $\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3$
Operations: $2 \times \sigma: \mathbb{R}^* \rightarrow \mathbb{R}^*, 1 \times \text{softmax}$
Practical Tips

- Select network structure appropriate for the problem
  - Window vs. recurrent vs. recursive
  - Non-linearity function
- Gradient checks to identify bugs
  - If you build from scratch
- Parameter initialization
- Model is powerful enough?
  - If not, make it larger
  - Yes, so regularize, otherwise it will overfit
- Know your non-linearity function and its gradient
  - Example $\tanh(x)$

\[ \frac{\partial}{\partial x} \tanh(x) = 1 - \tanh^2(x) \]
Debugging

• Verify value of initial loss when using softmax
• Perfectly fit a single mini-batch
• If learning fails completely, maybe gradients stuck
  – Check learning rate
  – Verify parameter initialization
  – Change non-linearity functions
Avoiding Overfitting

• Reduce model size (but not too much)
• L1 and L2 regularization
• Early stopping (e.g., *patience*)
• Dropout (Hinton et al. 2012)
  – Randomly set 50% of inputs in each layer to 0