CS395T: Structured Models for NLP
Lecture 13: Neural Networks

Greg Durrett

Administrivia

› Project 2 due on Tuesday
› Project 1 samples posted on website

This Lecture

› Neural network history
› Neural network basics
› Feedforward neural networks
› Backpropagation
› Applications

A brief history of (modern) NLP


“AI winter”
rule-based, expert systems
Penn treebank
Collins vs. Charniak parsers
Unsup: topic models, grammar induction
earliest stat MT work at IBM
Ratnaparkhi tagger
Sup: SVMs, CRFs, NER, Sentiment
Semi-sup, structured prediction
Neural
History: NN “dark ages”

- Convnets: applied to MNIST by LeCun in 1998
- LSTMs: Hochreiter and Schmidhuber (1997)

2008-2013: A glimmer of light...

- Collobert and Weston 2011: “NLP (almost) from scratch”
  - Feedforward neural nets induce features for sequential CRFs (“neural CRF”)
  - 2008 version was marred by bad experiments, claimed SOTA but wasn’t, 2011 version tied SOTA
- Krizhevsky et al. (2012): AlexNet for vision
- Socher: tree-structured RNNs
  - Started working well for sentiment in 2013, but only worked for weird tasks before that, some lackluster parsing results

2014: Stuff starts working

  - Basic convnets work pretty well for NLP
- Sutskever et al., Bahdanau et al. seq2seq for neural MT
  - LSTMs actually do well at NLP problems
- Chen and Manning transition-based dependency parser
  - Feedforward neural networks for parsing
- 2015: explosion of neural nets for everything under the sun

Why didn’t they work before?

- Datasets too small: for MT, not really better until you have 1M+ parallel sentences (and really need a lot more)
- Optimization not well understood: good initialization, per-feature scaling + momentum (Adagrad / Adadelta / Adam) work best out-of-the-box
  - Regularization: dropout was very important
- Computers not big enough: can’t run for enough iterations
- Inputs: need word representations to have the right continuous semantics
  - Dealing with unknown words: word pieces, use character LSTMs, ... complex stuff!
Neural Nets Basics

Neural Networks

- Linear classification: \( \operatorname{argmax}_y w^T f(x, y) \)
- How can we do nonlinear classification?
- Polynomial, etc. from kernels, but these are slow!
- Kernels are neither necessary nor sufficient: not every pair of features interacts, might need to go beyonds pairs
- Instead, want to learn intermediate conjunctive features of the input

Neural Networks: XOR

- Let’s see how we can use neural nets to learn a simple nonlinear function
- Inputs  \( x_1, x_2 \)
  (generally \( x = (x_1, \ldots, x_m) \))
- Output  \( y \)
  (generally \( y = (y_1, \ldots, y_n) \))

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( y = x_1 \text{ XOR } x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)
\]

(looks like action potential in neuron)
Neural Networks: XOR

\[ y = a_1 x_1 + a_2 x_2 \]

\[ y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2) \]

\[ y = -x_1 - x_2 + 2 \tanh(x_1 + x_2) \]

"or"

the movie was not good

Neural Networks

(Linear model: \( y = \mathbf{w} \cdot \mathbf{x} + b \))

\[ y = g(\mathbf{w} \cdot \mathbf{x} + b) \]

\[ y = g(\mathbf{Wx} + \mathbf{b}) \]

Nonlinear transformation

Warp space

Shift

Linear classifier

Neural network

…possible because we transformed the space!
Deep Neural Networks

(this was our neural net from the XOR example)

\[ y_1 = g(w_1 \cdot x + b_1) \]

Deep Neural Networks

\[ y_1 = g(w_1 \cdot x + b_1) \quad y = g(Wx + b) \]

Deep Neural Networks

Output of first layer

\[ z = g(Vy + c) \]

\[ z = g(Vg(Wx + b) + c) \]

“Feedforward”: computation “feeds forward” (not recurrent)

Check: what happens if no nonlinearity?

More powerful than basic linear models?

\[ z = V(Wx + b) + c \]

Deep Neural Networks

Linear classifier

Neural network

…possible because we transformed the space!

Deep Neural Networks

- Using multiple layers of processing to induce deep representations parallels visual processing in the brain

Feedforward Networks, Backpropagation

Logistic Regression with NNs

\( P(y|x) = \frac{\exp(w^T f(x, y))}{\sum_{y'} \exp(w^T f(x, y'))} \)

- Single scalar probability

\( P(y|x) = \text{softmax}_y(w^T f(x, y)) \)

- \( \text{softmax}_y \): score vector -> prob of \( y \)

\( P(y|x) = \text{softmax}_y(w^y g(V f(x))) \)

- Feature function no longer looks at label — same shared processing for each label.

- \( \text{softmax} \): score vector -> probability vector

\( P(y|x) = \text{softmax}(W g(V f(x))) \)

- Assumes that the labels \( y \) are indexed and associated with coordinates in a vector space
Neural Networks for Classification

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

Training Neural Networks

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

- Maximize log likelihood of training data
  \[ \log P(y = i^*|x) = \log (\text{softmax}(Wg(Vf(x))) \cdot e_{i^*}) \]
  - \( i^* \): index of the gold label
  - \( e_i \): 1 in the \( i \)th row, zero elsewhere. Dot by this = select \( i \)th index

\[ \mathcal{L}(x, i^*) = Wg(Vf(x)) \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wg(Vf(x)) \cdot e_j) \]

Computing Gradients

\[ \mathcal{L}(x, i^*) = Wg(Vf(x)) \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wg(Vf(x)) \cdot e_j) \]

- Gradient with respect to \( W \)
  \[ \frac{\partial}{\partial W_{ij}} \mathcal{L}(x, i^*) = \begin{cases} z_j - P(y = i|x)z_j & \text{if } i = i^* \\ -P(y = i|x)z_j & \text{otherwise} \end{cases} \]

- Looks like logistic regression with \( z \) as the features!
CompuKng Gradients: BackpropagaKon

\[ \mathcal{L}(x, i^*) = Wz \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wz \cdot e_j) \]

\[ z = g(Vf(x)) \]

Activations at hidden layer

\[ \nabla \mathcal{L}(x, i^*) = \frac{\partial \mathcal{L}(x, i^*)}{\partial z} \nabla z = W_{i^*} - \sum_j P(y = j|x) W_j \]

\[ \text{weights(gold) - } E[\text{weights(guess)}], \text{ like LR with weights and features flipped!} \]

Or:

\[ \text{err(root)} = e_{i^*} - P(y|x) \]

\[ \frac{\partial \mathcal{L}(x, i^*)}{\partial z} = \text{err}(z) = W^\top \text{err(root)} \]

\[ \text{dim} = m \]

\[ \text{dim} = d \]

BackpropagaKon: Picture

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

CompuKng Gradients: BackpropagaKon

\[ \mathcal{L}(x, i^*) = Wz \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wz \cdot e_j) \]

\[ z = g(Vf(x)) \]

Activations at hidden layer

\[ \frac{\partial \mathcal{L}(x, i^*)}{\partial V_{ij}} = \frac{\partial \mathcal{L}(x, i^*)}{\partial z} \frac{\partial z}{V_{ij}} = W_{i^*} - \sum_j P(y = j|x) W_j \]

\[ \text{weights(gold) - } E[\text{weights(guess)}], \text{ like LR with weights and features flipped!} \]

\[ \text{Or: } \text{err(root)} = e_{i^*} - P(y|x) \]

\[ \frac{\partial \mathcal{L}(x, i^*)}{\partial z} = \text{err}(z) = W^\top \text{err(root)} \]

\[ \text{dim} = m \]

\[ \text{dim} = d \]

BackpropagaKon: Picture

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]
### Backpropagation

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

- Step 1: compute \( err(\text{root}) = e_i - P(y|x) \) (vector)
- Step 2: compute derivatives of \( W \) using \( err(\text{root}) \) (matrix)
- Step 3: compute \( \frac{\partial L(x, i^*)}{\partial z} = err(z) = W^T err(\text{root}) \) (vector)
- Step 4: compute derivatives of \( V \) using \( err(z) \) (matrix)
- Step 5+: continue backpropagation (compute \( err(f(x)) \) if necessary...)

### Backpropagation: Takeaways

- Gradients of output weights \( W \) are easy to compute — looks like logistic regression with hidden layer \( z \) as feature vector
- Can compute derivative of loss with respect to \( z \) to form an “error signal” for backpropagation
- Easy to update parameters based on “error signal” from next layer, keep pushing error signal back as backpropagation
- Need to remember the values from the forward computation

### Applications

### NLP with Feedforward Networks

- Part-of-speech tagging with FFNNs
  \( f(x) \)
  \[ Fed \text{ raises interest rates in order to ... } \]
  Previous word
  Curr word
  Next word
  Other words, feats, etc.

- Word embeddings for each word form input
- ~1000 features here — smaller feature vector than in sparse models, but every feature fires on every example
- Weight matrix learns position-dependent processing of the words
NLP with Feedforward Networks

- Hidden layer mixes these different signals and learns feature conjunctions

Botha et al. (2017)

NLP with Feedforward Networks

- Multilingual tagging results:

<table>
<thead>
<tr>
<th>Model</th>
<th>Acc.</th>
<th>Wts.</th>
<th>MB</th>
<th>Ops.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillick et al. (2016)</td>
<td>95.06</td>
<td>900k</td>
<td>-</td>
<td>6.63m</td>
</tr>
<tr>
<td>Small FF</td>
<td>94.76</td>
<td>241k</td>
<td>0.6</td>
<td>0.27m</td>
</tr>
<tr>
<td>+Clusters</td>
<td>95.56</td>
<td>261k</td>
<td>1.0</td>
<td>0.31m</td>
</tr>
<tr>
<td>1/2 Dim.</td>
<td>95.39</td>
<td>143k</td>
<td>0.7</td>
<td>0.18m</td>
</tr>
</tbody>
</table>

- Gillick used LSTMs; this is smaller, faster, and better

Botha et al. (2017)

Sentiment Analysis

- Deep Averaging Networks: feedforward neural network on average of word embeddings from input

Iyyer et al. (2015)

Sentiment Analysis

Bag-of-words

- Wang and Manning (2012)

Tree RNNs / CNNS / LSTMS

- Kim (2014)

Iyyer et al. (2015)
Coreference Resolution

- Feedforward networks identify coreference arcs

Clark and Manning (2015), Wiseman et al. (2015)

Next Time

- How to implement neural networks for NLP
- Tensorflow
- Practical training techniques
- Word representations / word vectors
- word2vec, GloVe