Deep Learning for Sequences

Lecture 2
Recurrent Neural Networks

David Kriegman
Computer Science & Engineering
UCSD

Announcements

• Syllabus of topics/papers “finalized”

• A google form for selecting topics of interest and potential partners for paper presentation is now available. Check Piazza. Please complete by noon on Friday.

• Schedule will be finalized by Monday. Preferences are not guaranteed.

Last Lecture

• Talked about sequences

• Sequence processing tasks \((T_x = \{0, 1, n\}, T_y = \{1, n, m\}\)}

• Review of feedforward networks
  • Fully connected networks
  • activation function (sigmoid, tanh, RELU)

• Loss functions (softmax + cross entropy)

Representing words as 1-hot vectors

Sequence of words: “Triton engineers rock the free world”

10,000 Word Dictionary

1: “a”
2: “aaron”
2023: “engineers”
2893: “free”
7317: “Triton”
9167: “world”
10000: “xylophone”
Using d-layer fully connected network for six word sequence

Number of Parameters
- $6 \times 10,000 \times n_1$
- $n_1 \times n_2$
- $n_2 \times n_3$
- $\vdots$
- $n_{d-1} \times n_d$

For $n_1=200$, weight matrix for first layer has 12M parameters

Problems with Feedfoward Network
- Number of parameters can become explosive in sequence length
- Input size is fixed.
- Padding is possible, but then maximum sequence length means network is big. (First layer: Dictionary size * largest sequence length)
- Output length also can’t vary
- Doesn’t share features (weights) across sequence positions.

Representing Chars as 1-hot vectors

Sequence of words: “Triton engineers rock the free world”

Instead of a six word sequence, we have a 36 character sequence

Vanilla Recurrent Neural Nets
Start with two layer feed forward network

Input $x \in \mathbb{R}^n$: n-D vector representing sequence element
Output $y \in \mathbb{R}^c$: C classes
Hidden layer $a \in \mathbb{R}^h$: h-nodes

$$a = g_1(W_{xa}x + b_a)$$
$$y = g_2(W_{ya}a + b_y)$$
$$g_c(W_{yc}g_1(W_{xa}x + b_a) + b_y)$$

where $g_1$ and $g_2$ are activation functions.
Vanilla Recurrent Neural Nets
Elman Network

Add in recurrent loop

Input $x \in \mathbb{R}^n$: n-D vector representing sequence element
Output $y \in \mathbb{R}^c$: C classes
Hidden layer $a \in \mathbb{R}^h$: h-nodes

$x \in \mathbb{R}^n$
$y \in \mathbb{R}^c$
$a \in \mathbb{R}^h$

$W_{xa} \in \mathbb{R}^{n \times h}$
$W_{ay} \in \mathbb{R}^{h \times c}$
$W_{aa} \in \mathbb{R}^{h \times h}$

$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$

where $g_1$ is usually tanh or sigmoid
$g_2$ is usually softmax

Note:
$a_0$: State must be initialized, usually vector zeros

Training RNN: Forward Pass

How do we train?
We use backdrop with a trick

Initialize:
$a_0 = 0$
First input: $x_1$
$a_1 = g_1(W_{xa}x_1 + W_{aa}a_0 + b_a)$
$y_1 = g_2(W_{ay}a_1 + b_y)$

Second input: $x_2$
$a_2 = g_1(W_{xa}x_2 + W_{aa}a_1 + b_a)$
$y_2 = g_2(W_{ay}a_2 + b_y)$

$i$-th input: $x_i$
$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$

Training RNN: Unrolling

We use backdrop with a trick
Unroll network over time for sequence of length $T_x$

$x_1$ $x_2$ $x_3$ $x_T$
$a_0 = 0$ $a_1$ $a_2$ $a_T$
$W_{xa}$ $W_{ay}$ $W_{aa}$
$W_{xa}$ $W_{ay}$ $W_{aa}$

Training RNN: BPTT

- This looks like a feedforward network
- Train using Stochastic Gradient Descent with ADAM/RMSprop
- Back Propogation Through Time (BPTT)
- The unrolled network is deep — Number of layers $= T_x + 1$
This unrolling looks complicated to implement!

- No worries, it’s already implemented as part of training in the major deep learning frameworks (Tensorflow, Pytorch, etc.)

**Statistical Language Model**

\[ P(x_1, x_2, x_3, x_4, x_5) = P(x_5 | x_1, x_2, x_3, x_4)P(x_1, x_2, x_3, x_4) = P(x_5 | x_1, x_2, x_3, x_4)P(x_4 | x_1, x_2, x_3)P(x_1, x_2, x_3) \]

or more generally

\[ P(x_1, \ldots, x_m) = \prod_{i=1}^{m} P(x_i | x_1, \ldots, x_{i-1}) \]

\( P(x_1, \ldots, x_m) \) is usually a pretty small number, especially as length becomes longer, so often work with log probabilities:

\[ \log P(x_1, \ldots, x_m) = \sum_{i=1}^{m} \log P(x_i | x_1, \ldots, x_{i-1}) \]

**Toward Training a character language model**

- A statistical language model is a probability distribution over sequence of words.
- \( P(x_1, x_2, x_3, x_4, x_5) \) is the probability of a sequence of five words being used in the language.
- Examples
  - \( P(\text{“the”, “cat”, “in”, “the”, “hat”}) \) — The probability of the “The cat in the hat” appearing in a corpus of english text.
  - \( P(\text{“a”, “q”, “t”, “i”, “u”}) \) — The probability of the letter sequence “aqtui” appearing in a corpus of english text

**Character RNN implementing a language model**

- \( x \) : is one-hot encoding of characters and includes punctuation, space (end of word)
- \( \hat{y} \) : has the same number of outputs as \( x \) and the final activation function is softmax
- The outputs can be viewed as the conditional probability of the next character

When a sequence \( x_1, x_2, \ldots, x_n \) has been input to a trained network, the output \( y \) is the probability distribution for \( x_{n+1} \)
Training Character RNN

The network is trained using cross entropy loss to compare the next character from the training set $x_{n+1}$ to the output $\hat{y}$ from the network.

Take sequences of characters, and put them in one at a time, looking comparing the predicted character to the next one.

\[
W_{ax} \in \mathbb{R}^{h \times c}, \\
W_{y} \in \mathbb{R}^{h \times c}, \\
W_{aa} \in \mathbb{R}^{h \times h}
\]

Inference with trained RNN

\[
P(x_2 | 'h') \ 
\hat{y}_1, \ 
P(x_3 | 'h', 'e') \ 
\hat{y}_2, \ 
P(x_4 | 'h', 'e', 'l') \ 
\hat{y}_3, \ 
\ldots
\]

Examples from Karpathy’s Blog

“The Unreasonable Effectiveness of Recurrent Neural Networks”, Andrej Karpathy, 2015

- Shakespeare
  - Trained on all works of William Shakespeare
  - 512 Hidden units
  - 1000's of epochs
  - Characters only!!!

- Results as Function of number of epochs (Tolstoy data)
- Latex (produces mathy looking latex)
  - 16MB of latex.
- C code
  - Trained on 474MB of code from Linux Github Repo

And how complicated is this to code?

My Keras code for creating a network

```python
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.01))
```