Project Progress Report

1. What changes you have made for the task compared to the proposal, including problem/task, models, datasets, or evaluation methods? If there is any change, please explain why.

2. Describe data preprocessing process. This includes data cleaning, selection, feature generation or other representation you have used, etc.

3. What methods or models you have tried towards the project goal? And why do you choose the methods you are including related work on similar task or relevant tasks?

4. What results you have achieved up to now based on your proposed evaluation methods? What is working or what is wrong with the model?

5. How can you improve your models? What are the next steps?

Grading: For 2-5, each aspect will take 25 points.

Length: 2 page (or more if necessary). Single space if MS word is used. Or you can choose latex template, e.g. https://www.acm.org/publications/proceedings-template or http://icml.cc/2015/?page_id=151.

Each group only needs to submit one copy.

Logistics

• Progress report is due at Oct 31, 11:59pm

• If you can’t finish running on a large dataset, you can try a small dataset, but notice what the effect would be

• Start with baseline models.

• Amazon Web Service credit/Google cloud credit
  • Debug models locally, learn to debug and test

Outline

• Basics about Feedforward Neural Networks
• Neural language model (word2vec)
• Recurrent Neural Network (RNN) and LSTM

Neural Network Learning

• Learning approach based on modeling adaptation in biological neural systems.
  • Perceptron: Initial algorithm for learning simple neural networks (single layer) developed in the 1950’s.
ARTIFICIAL NEURON

Topics: connection, weights, bias, activation function

ACTIVATION FUNCTION

Topics: linear activation function
- Performs no input squashing
- Not very interesting.

\[ g(a) = a \]

ACTIVATION FUNCTION

Topics: sigmoid activation function
- Squashes the neuron's pre-activation between 0 and 1
- Always positive
- Bounded
- Strictly increasing

\[ g(a) = \text{sigmoid}(a) = \frac{1}{1 + \exp(-a)} \]

ACTIVATION FUNCTION

Topics: hyperbolic tangent ("tanh") activation function
- Squashes the neuron's pre-activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

\[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \]

ACTIVATION FUNCTION

Topics: rectified linear activation function
- Bounded below by 0 (always non-negative)
- Not upper bounded
- Strictly increasing
- Tends to give neurons with sparse activities

\[ g(a) = \text{relu}(a) = \max(0, a) \]

class Neuron:
    def forward(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.dot(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1 + np.exp(-cell_body_sum)) # sigmoid activation function
        return firing_rate
Linear Separator

• Since one-layer neuron (aka perceptron) uses linear threshold function, it is searching for a linear separator that discriminates the classes.
# Layer-wise of a 3-layer neural network:

```python
W = [randn(100, 100), randn(100, 100)]  # activation function (e.g., sigmoid)
X = np.random.randn(100, 1)  # random input vector of three samples (1 sample)
H1 = np.tanh(W[0] @ X + b1)  # calculate first hidden layer activations (40 nodes)
H2 = np.tanh(W[1] @ H1 + b2)  # calculate second hidden layer activations (20 nodes)
out = np.tanh(W[2] @ H2 + b3)  # output neuron (10 nodes)
```

---

**CAPACITY OF NEURAL NETWORK**

**Topics:** single hidden layer neural network

---

**CAPACITY OF NEURAL NETWORK**

Topics: single hidden layer neural network

---

**CAPACITY OF NEURAL NETWORK**

**Topics:** universal approximation

- Universal approximation theorem: neural networks can approximate any continuous function on a compact subset of 
  any normed vector space.
- The result applies for sigmoid, tanh, and many other hidden layer activation functions.
- This is a good result but it doesn’t mean there is a learning algorithm that can find the necessary parameter values.
How to train a neural network?

- Could have $L$ hidden layers.
- Layer input activation for $l = 0$:
  \[ a^{(0)}(x) = x \]
- Layer input activation for $l > 0$:
  \[ a^{(l)}(x) = W^{(l)} a^{(l-1)}(x) + b^{(l)} \]
- Hidden layer activation (0 from 1 to L):
  \[ h^{(l)}(x) = g(a^{(l)}(x)) \]
- Output layer activation (0 to L+1):
  \[ h^{(L+1)}(x) = a(h^{(L)}(x)) = f(x) \]

Empirical Risk Minimization

- Empirical risk minimization
  - Framework to design learning algorithms
  \[ \min \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i^o; \theta), y_i^o) + \Omega(\theta) \]
- $\ell(f(x_i^o; \theta), y_i^o)$ is a loss function
- $\Omega(\theta)$ is a regularizer (penalizes certain values of $\theta$)
- Learning is cast as optimization
- Ideally, we want classification error but it's not smooth
- Loss function is a surrogate for what we truly should optimize (e.g., upper bound)

Loss Function

- Loss function for classification
  - Neural network estimates $f(x) = p(y = 1|x)$
  - We could maximize the probability of $y^o = 1$ given $x^o$ in the training set
  - To frame as minimization, we minimize the negative log likelihood
    \[ \ell(f(x), y) = -\sum_{(x,y)} \log f(x|y) \]
    - We take the log to simplify for numerical stability and make it smooth
    - Sometimes referred to as cross entropy

Regularization

- L2 regularization
  \[ \Omega(\theta) = \sum_{l=1}^{L} \sum_{i=1}^{d} (W_{i}^{l})^2 \]

Empirical Risk Minimization

- Empirical risk minimization
  - Framework to design learning algorithms
  \[ \min \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i^o; \theta), y_i^o) + \Omega(\theta) \]
- $\ell(f(x_i^o; \theta), y_i^o)$ is a loss function
- $\Omega(\theta)$ is a regularizer (penalizes certain values of $\theta$)
- Learning is cast as optimization
- Ideally, we want classification error but it's not smooth
- Loss function is a surrogate for what we truly should optimize (e.g., upper bound)
Model Learning

- Backpropagation algorithm (not covered in the lecture)

Toolkits

- TensorFlow
  - [https://www.tensorflow.org/](https://www.tensorflow.org/)
- Theano (not maintained any more)
  - [http://deeplearning.net/software/theano/](http://deeplearning.net/software/theano/)
- PyTorch
  - [http://pytorch.org/](http://pytorch.org/)

Neural language models

- Skip-grams
- Continuous Bag of Words (CBOW)
  - More details can be found at [https://cs224d.stanford.edu/lecture_notes/notes1.pdf](https://cs224d.stanford.edu/lecture_notes/notes1.pdf)

Prediction-based models:
An alternative way to get dense vectors

- Skip-gram (Mikolov et al. 2013a), CBOW (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the word2vec package
  - Including sets of pretrained embeddings!
Skip-grams

- Predict each neighboring word
  - in a context window of 2C words
  - from the current word.
- So for C=2, we are given word \( w_t \) and predicting these 4 words:

\[
\begin{bmatrix}
W_{t-2}, W_{t-1}, W_{t+1}, W_{t+2}
\end{bmatrix}
\]

Example: Natural language processing is a subarea of artificial intelligence.

Skip-grams

- Predict each neighboring word
  - in a context window of 2C words
  - from the current word.
- So for C=2, we are given word \( w_t \) and predicting these 4 words:

\[
\begin{bmatrix}
W_{t-2}, W_{t-1}, W_{t+1}, W_{t+2}
\end{bmatrix}
\]

Setup

- Walking through corpus pointing at word \( w_t \), whose index in the vocabulary is \( j \), so we’ll call it \( w_j \) \( 1 < j < |V| \).
- Let’s predict \( w_{t+1} \), whose index in the vocabulary is \( k \) \( 1 < k < |V| \).
Hence our task is to compute \( P(w_{t+1} | w_t) \).

Skip-grams learn 2 embeddings for each \( w \)

**Input embedding** \( \mathbf{v}_i \) in the input matrix \( \mathbf{W} \)
- Column \( i \) of the input matrix \( \mathbf{W} \) is the 1x\( d \) embedding \( \mathbf{v}_i \) for word \( i \) in the vocabulary.

**Output embedding** \( \mathbf{v}'_j \) in output matrix \( \mathbf{W}' \)
- Row \( j \) of the output matrix \( \mathbf{W}' \) is a \( d \times 1 \) vector embedding \( \mathbf{v}'_j \) for word \( j \) in the vocabulary.

One-hot vectors

- A vector of length \( |V| \)
- 1 for the target word and 0 for other words
- So if “popsicle” is vocabulary word \( s \)
- The one-hot vector is
  - \([0,0,0,0,1,0,0,0,...,0]\)

Skip-gram

- Input layer
  - 1-hot input vector
- Projection layer
  - Embedding for \( w_t \)
- Output layer
  - Probabilities of context words
Turning outputs into probabilities

- \( \mathbf{o}_k = \mathbf{v}'_k \mathbf{v}_j \)
- We use softmax to turn into probabilities

\[
p(w_k|w_j) = \frac{\exp(\mathbf{v}'_k \cdot \mathbf{v}_j)}{\sum_{w' \in |V|} \exp(\mathbf{v}'_{w'} \cdot \mathbf{v}_j)}
\]

Embeddings from \( W \) and \( W' \)

- Since we have two embeddings, \( \mathbf{v}_j \) and \( \mathbf{v}'_j \) for each word \( w_j \)
- We can either:
  - Use \( \mathbf{v}_j \)
  - Sum them
  - Concatenate them to make a double-length embedding

But wait; how do we learn the embeddings?

\[
\arg\max_{\theta} \log p(\text{Text})
\]

But wait; how do we learn the embeddings?

\[
\arg\max_{\theta} \sum_{i=1}^{c} \sum_{v_i \in \mathcal{I}} \log \frac{\exp(\mathbf{v}_k \cdot \mathbf{v}_i)}{\sum_{v'_i \in \mathcal{I}} \exp(\mathbf{v}'_k \cdot \mathbf{v}'_i)}
\]

\[
\arg\max_{\theta} \sum_{i=1}^{c} \sum_{v_i \in \mathcal{I}} \left[ \log \frac{\exp(\mathbf{v}_k \cdot \mathbf{v}_i)}{\sum_{v'_i \in \mathcal{I}} \exp(\mathbf{v}'_k \cdot \mathbf{v}'_i)} - \log \sum_{v'_i \in \mathcal{I}} \exp(\mathbf{v}'_k \cdot \mathbf{v}'_i) \right]
\]
Recurrent Neural Networks

Feed-forward NN
\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

Recurrent NN
\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]

Properties of embeddings

- Nearest words to some embeddings (Mikolov et al. 2013a)

<table>
<thead>
<tr>
<th>Target</th>
<th>Redmond Wash.</th>
<th>Velvet Revolution</th>
<th>spray paint</th>
<th>graffiti</th>
<th>capitalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redmond Wash.</td>
<td>Vector Mark</td>
<td>president Vaclav</td>
<td>sales</td>
<td>graffiti</td>
<td>capitalized</td>
</tr>
<tr>
<td>Wash.</td>
<td>Vaclav Havel</td>
<td>Havel</td>
<td>Redmond</td>
<td>spray</td>
<td>Wash.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>paint</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>graffiti</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>capitalized</td>
<td></td>
</tr>
</tbody>
</table>

Long Distance Dependencies

- It is very difficult to train NNs to retain information over many time steps.
- This makes it very difficult to handle long-distance dependencies, such as subject-verb agreement.
- E.g., Jane walked into the room. John walked in too. It was late in the day. Jane said hi to whom?

Embeddings capture relational meaning!

vector('king') - vector('man') + vector('woman') \approx vector('queen')
vector('Paris') - vector('France') + vector('Italy') \approx vector('Rome')

Recurrence Neural Networks

Feed-forward NN
\[ h = g(Vx + c) \]
\[ \hat{y} = Wh + b \]

Recurrent NN
\[ h_t = g(Vx_t + Uh_{t-1} + c) \]
\[ \hat{y}_t = Wh_t + b \]
Long-Short Term Memory Networks (LSTMs)

Another Visualization

Figure: Christopher Olah

Sequence to Sequence

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.

This is my cat
C'est mon chat

Successful Applications of LSTMs

- Speech recognition: Language and acoustic modeling
- Sequence labeling
- POS Tagging
- NER
- Phrase Chunking
- Neural syntactic and semantic parsing
- Image captioning
- Sequence to Sequence
  - Machine Translation (Sutskever, Vinyals, & Le, 2014)
  - Video Captioning (input sequence of CNN frame outputs)
Bi-directional LSTM (Bi-LSTM)

- Separate LSTMs process sequence forward and backward and hidden layers at each time step are concatenated to form the cell output.

![Bi-directional LSTM Diagram]

Homework

- Project progress report is due on Oct 31.