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 can 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 templates, 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

[Slides borrowed from Hugo Larochelle, Raymond Mooney, Kai-wei Chang]
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.

• **Backpropagation**: More complex algorithm for learning multi-layer neural networks developed in the 1980’s.
**ARTIFICIAL NEURON**

**Topics:** connection weights, bias, activation function

- Neuron pre-activation (or input activation):
  \[ a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x} \]

- Neuron (output) activation
  \[ h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i) \]

- \( \mathbf{w} \) are the connection weights
- \( b \) is the neuron bias
- \( g(\cdot) \) is called the activation function
ARTIFICIAL NEURON

Topics: connection weights, bias, activation function

Range determined by $g(\cdot)$

Bias $b$ only changes the position of the riff

(from Pascal Vincent's slides)
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

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

\[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1} \]
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

\[ g(a) = \text{recllin}(a) = \max(0, a) \]
class Neuron(object):
    # ...
    def forward(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.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.
Topics: capacity of single neuron

- Can solve linearly separable problems
ARTIFICIAL NEURON

Topics: capacity of single neuron

- Can't solve non-linearly separable problems...

- ... unless the input is transformed in a better representation
Topics: single hidden layer neural network

- Hidden layer pre-activation:
  \[ a(x) = b^{(1)} + W^{(1)}x \]
  \[ a(x)_i = b^{(1)}_i + \sum_j W^{(1)}_{i,j} x_j \]

- Hidden layer activation:
  \[ h(x) = g(a(x)) \]

- Output layer activation:
  \[ f(x) = o \left( b^{(2)} + w^{(2)^T} h^{(1)}x \right) \]
Topics: softmax activation function

- For multi-class classification:
  - we need multiple outputs (1 output per class)
  - we would like to estimate the conditional probability \( p(y = c | x) \)

- We use the softmax activation function at the output:
  \[
  o(a) = \text{softmax}(a) = \left[ \frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)} \right]^T
  \]
  - strictly positive
  - sums to one

- Predicted class is the one with highest estimated probability
**Topics:** multilayer neural network

- Could have $L$ hidden layers:
  - layer pre-activation for $k > 0$ \( h^{(0)}(x) = x \)
    \[
    a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)
    \]
  - hidden layer activation ($k$ from 1 to $L$):
    \[
    h^{(k)}(x) = g(a^{(k)}(x))
    \]
  - output layer activation ($k = L + 1$):
    \[
    h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)
    \]
# forward-pass of a 3-layer neural network:

```python
f = lambda x: 1.0/(1.0 + np.exp(-x))  # activation function (use sigmoid)
x = np.random.randn(3, 1)  # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1)  # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2)  # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3  # output neuron (1x1)
```
CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network
CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network

(from Pascal Vincent's slides)
CAPACITY OF NEURAL NETWORK

Topics: single hidden layer neural network

(from Pascal Vincent's slides)
CAPACITY OF NEURAL NETWORK

**Topics:** universal approximation

- Universal approximation theorem (Hornik, 1991):
  - “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”

- 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?

**Topics:** multilayer neural network

- Could have $L$ hidden layers:
  
  - layer input activation for $k > 0$: $a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)$
  
  - hidden layer activation ($k$ from 1 to $L$):
    
    $h^{(k)}(x) = g(a^{(k)}(x))$
  
  - output layer activation ($k = L+1$):
    
    $h^{(L+1)}(x) = o(a^{(L+1)}(x)) = f(x)$
**Topics:** empirical risk minimization, regularization

- **Empirical risk minimization**
  - framework to design learning algorithms

\[
\arg \min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)
\]

- \(l(f(x^{(t)}; \theta), y^{(t)})\) is a loss function
- \(\Omega(\theta)\) is a regularizer (penalizes certain values of \(\theta\))

- **Learning is cast as optimization**
  - ideally, we'd optimize classification error, but it's not smooth
  - loss function is a surrogate for what we truly should optimize (e.g. upper bound)
**Topics:** loss function for classification

- Neural network estimates $f(x)_c = p(y = c | x)$
  - we could maximize the probabilities of $y^{(t)}$ given $x^{(t)}$ in the training set

- To frame as minimization, we minimize the negative log-likelihood
  
  $l(f(x), y) = - \sum_c 1_{(y=c)} \log f(x)_c = - \log f(x)_y$

  - we take the log to simplify for numerical stability and math simplicity
  - sometimes referred to as cross-entropy
REGULARIZATION

Topics: L2 regularization

$$\Omega(\theta) = \sum_k \sum_i \sum_j \left( W_{i,j}^{(k)} \right)^2 = \sum_k \|W^{(k)}\|_F^2$$
Empirical Risk Minimization

**Topics:** empirical risk minimization, regularization

- Empirical risk minimization
  - framework to design learning algorithms

\[
\arg\min_{\theta} \frac{1}{T} \sum_{t} l(f(x^{(t)}; \theta), y^{(t)}) + \lambda \Omega(\theta)
\]

- \(l(f(x^{(t)}; \theta), y^{(t)})\) is a loss function
- \(\Omega(\theta)\) is a regularizer (penalizes certain values of \(\theta\))

- Learning is cast as optimization
  - ideally, we’d optimize classification error, but it’s not smooth
  - loss function is a surrogate for what we truly should optimize (e.g., upper bound)
[http://cs231n.github.io/neural-networks-1/]
INITIALIZATION

**Topics:** initialization

- For biases
  - initialize all to 0

- For weights
  - Can’t initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - Can’t initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: sample $W_{i,j}^{(k)}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$
    - the idea is to sample around 0 but break symmetry
    - other values of $b$ could work well (not an exact science) (see Glorot & Bengio, 2010)
Model Learning

- Backpropagation algorithm (not covered in the lecture)
Toolkits

• TensorFlow
  • https://www.tensorflow.org/

• Theano (not maintained any more)
  • http://deeplearning.net/software/theano/

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

- **Skip-gram** (Mikolov et al. 2013a), **CBOB** (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:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$
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:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Example: Natural language processing is a subarea of artificial intelligence.
Skip-grams learn 2 embeddings for each word $w$.

**Input embedding** $v$, in the input matrix $W$:
- Column $i$ of the input matrix $W$ is the $1 \times d$ embedding $v_i$ for word $i$ in the vocabulary.

**Output embedding** $v'$, in output matrix $W'$:
- Row $i$ of the output matrix $W'$ is the $d \times 1$ vector embedding $v'_i$ for word $i$ in the vocabulary.
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 $P(w_k | w_j)$. 
One-hot vectors

- A vector of length $|V|$
- 1 for the target word and 0 for other words
- So if “popsicle” is vocabulary word 5
- The one-hot vector is
- $[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]$
Skip-gram

**Input layer**
- 1-hot input vector
- \( w_t \)
- \( x_1, x_2, \ldots, x_{|V|} \)

**Projection layer**
- Embedding for \( w_t \)
- \( W \) \( |V| \times d \)

**Output layer**
- Probabilities of context words
- \( W' \) \( d \times |V| \)
- \( y_1, y_2, \ldots, y_{|V|}, w_{t-1}, w_{t+1} \)
Skip-gram

Input layer

1-hot input vector

\[ w_t \]

\[ x_1 \]

\[ x_2 \]

\[ \vdots \]

\[ x_j \]

\[ \vdots \]

\[ x_{|V|} \]

\[ 1 \times |V| \]

Projection layer

embedding for \( w_t \)

\[ W \]

\[ |V| \times d \]

Output layer

probabilities of context words

\[ o = W' h \]

\[ y_1 \]

\[ y_2 \]

\[ \vdots \]

\[ y_k \]

\[ \vdots \]

\[ y_{|V|} \]

\[ 1 \times d \]

\[ W'_{d \times |V|} \]

\[ 1 \times |V| \]

\[ o = W' h \]

\[ w_{t-1} \]

\[ w_{t+1} \]
Skip-gram

Input layer

- 1-hot input vector
- Embedding for $w_t$

Projection layer

- $h = v_j$

Output layer

- Probabilities of context words

- $o_k = v'_k h \ (h = v_j)$
- $o_k = v'_k \cdot v_j$

Equations:

- $h = v_j$
- $o_k = v'_k h \ (h = v_j)$
- $o_k = v'_k \cdot v_j$
Turning outputs into probabilities

- $o_k = v'_k \cdot v_j$
- We use softmax to turn into probabilities

$$p(w_k | w_j) = \frac{\exp(v'_k \cdot v_j)}{\sum_{w' \in |V|} \exp(v'_w \cdot v_j)}$$
Embeddings from W and W’

• Since we have two embeddings, $v_j$ and $v'_j$ for each word $w_j$

• We can either:
  • Just use $v_j$
  • 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?

\[
\text{argmax} \ \log p(\text{Text})
\]

\[
\begin{align*}
\text{argmax}_\theta & \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log \frac{\exp(v'(t+j) \cdot v(t))}{\sum_{w \in |V|} \exp(v'_w \cdot v(t))} \\
\text{argmax}_\theta & \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \left[ v'(t+j) \cdot v(t) - \log \sum_{w \in |V|} \exp(v'_w \cdot v(t)) \right]
\end{align*}
\]
CBOW (Continuous Bag of Words)

- **Input layer**: 1-hot input vectors for each context word
- **Projection layer**: sum of embeddings for context words
- **Output layer**: probability of $w_t$

The diagram shows the structure of the CBOW model with input, projection, and output layers. The input layer consists of 1-hot vectors for each context word. The projection layer aggregates the embeddings of the context words. The output layer outputs the probability of the target word $w_t$. The operations involve matrix multiplications and 1-hot vectors.
Properties of embeddings

• Nearest words to some embeddings (Mikolov et al. 20131)
Embeddings capture relational meaning!

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

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 _?_.

\[ h_t \]  
\[ A \]  
\[ X_t \]  

\[ h_0 \]  
\[ h_1 \]  
\[ h_2 \]  
\[ h_t \]  

\[ A \]  
\[ A \]  
\[ A \]  
\[ A \]  

\[ X_0 \]  
\[ X_1 \]  
\[ X_2 \]  
\[ X_t \]  

\[ \ldots \]
Recurrent Neural Networks

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

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

**Feed-forward NN**

\[
\begin{align*}
    h &= g(Vx + c) \\
    \hat{y} &= Wh + b
\end{align*}
\]

**Recurrent NN**

\[
\begin{align*}
    h_t &= g(Vx_t \| Uh_{t-1} \| c) \\
    h_t &= g(V[x_t; h_{t-1}] + c) \\
    \hat{y}_t &= Wh_t + b
\end{align*}
\]
Long-Short Term Memory Networks (LSTMs)
Another Visualization

Capable of modeling long-distant dependencies between states.

Figure: Christopher Olah
Long-Short Term Memory Networks (LSTMs)

Use gates to control the information to be added from the input, forgot from the previous memories, and outputted. \( \sigma \) and \( f \) are sigmoid and tanh function respectively, to map the value to \([-1, 1]\).
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
Summary of LSTM Application Architectures

- **Image Captioning**
- **Video Activity Recognition**
- **Text Classification**
- **Video Captioning**
- **Machine Translation**
- **POS Tagging**
- **Language Modeling**
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 (Sustkever, 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.
Homework

• Neural language models: https://web.stanford.edu/~jurafsky/slp3/7.pdf, 3rd ed
• Project progress report is due on Oct 31.