Outline

Morning program
  Preliminaries
  Semantic matching
  Learning to rank
  Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Outline

Morning program
  Preliminaries
    Feedforward neural network
    Distributed representations
    Recurrent neural networks
    Sequence-to-sequence models
    Convolutional neural networks

  Semantic matching
  Learning to rank
  Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Multi-layer perceptron a.k.a. feedforward neural network

target: y
output/prediction: \( \hat{y} \)

output layer
weights

hidden layer
weights

input: x

\( y_1 \)
\( y_2 \)
\( y_3 \)

\( \hat{y}_1 \)
\( \hat{y}_2 \)
\( \hat{y}_3 \)

cost function
e.g.: 1/2 (\( \hat{y} - y \))^2

\( x_{i-1,1} \)
\( x_{i-1,2} \)
\( x_{i-1,3} \)
\( x_{i-1,4} \)

node \( j \) at level \( i \)

\( y_1 \)
\( y_2 \)
\( y_3 \)

\( \phi: \) activation function
e.g.: sigmoid

\( \frac{1}{1 + e^\phi} \)

\( o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k} \)

\( x_{i,j} = \phi(o_{i,j}) \)
Multi-layer perceptron a.k.a. feedforward neural network

input layer: $\mathbf{x}_1$

hidden layer: $\mathbf{x}_2$

output layer/predictions $\hat{\mathbf{x}}_3$ ($\mathbf{x}_3$)

ground truth: $\mathbf{y}$

$\mathbf{y}[0]$ | $\mathbf{y}[1]$ | $\mathbf{y}[2]$

activation

$\mathbf{x}_3 = \sigma(\mathbf{o}_2) = [1 \times 4]$

$\mathbf{x}_2 \cdot W_2 = \mathbf{o}_2$

$[1 \times 4] \cdot [4 \times 4] = [1 \times 4]$

activation

$\mathbf{x}_2 = \sigma(\mathbf{o}_1) = [1 \times 4]$

$\mathbf{x}_1 \cdot W_1 = \mathbf{o}_1$

$[1 \times 4] \cdot [4 \times 4] = [1 \times 4]$

activation

$\mathbf{x}_1 = \sigma(\mathbf{o}_0)$

$[1 \times 4] \cdot [4 \times 4] = [1 \times 4]$

$\mathbf{y}[0]$ | $\mathbf{y}[1]$ | $\mathbf{y}[2]$

$\mathbf{x}_3[0]$ | $\mathbf{x}_3[1]$ | $\mathbf{x}_3[2]$

$\mathbf{x}_2[0]$ | $\mathbf{x}_2[1]$ | $\mathbf{x}_2[2]$ | $\mathbf{x}_2[3]$

$\mathbf{x}_1[0]$ | $\mathbf{x}_1[1]$ | $\mathbf{x}_1[2]$ | $\mathbf{x}_1[3]$

$\phi$: activation function
e.g.: sigmoid

$\hat{y} = \mathbf{ŷ}$

$y_i = \sigma(\mathbf{o}_i)$

$\frac{1}{1 + \mathbf{e}^\phi}$

node $j$ at level $i$
Outline

Morning program
Preliminaries
  Feedforward neural network
  Distributed representations
  Recurrent neural networks
  Sequence-to-sequence models
  Convolutional neural networks

Semantic matching
Learning to rank
Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Distributed representations

- Represent units, e.g., words, as vectors
- Goal: words that are similar, e.g., in terms of meaning, should get similar embeddings

Cosine similarity to determine how similar two vectors are:

$$cosine(\vec{v}, \vec{w}) = \frac{\vec{v}^\top \cdot \vec{w}}{\|\vec{v}\|_2 \|\vec{w}\|_2}$$

$$= \frac{\sum_{i=1}^{\|v\|} v_i \cdot w_i}{\sqrt{\sum_{i=1}^{\|v\|} v_i^2} \sqrt{\sum_{i=1}^{\|w\|} w_i^2}}$$

newspaper = <0.08, 0.31, 0.41>
magazine = <0.09, 0.35, 0.36>
biking = <0.59, 0.25, 0.01>
Distributed representations

How do we get these vectors?

- You shall know a word by the company it keeps [Firth, 1957]
- The vector of a word should be similar to the vectors of the words surrounding it

$\overrightarrow{all} \overrightarrow{you} \overrightarrow{need} \overrightarrow{is} \overrightarrow{love}$
Embedding methods

**target distribution**

<table>
<thead>
<tr>
<th>all</th>
<th>answer</th>
<th>amtrak</th>
<th>is</th>
<th>need</th>
<th>you</th>
<th>what</th>
<th>zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**vocabulary size probability distribution**

```
... ...
```

?? turn this into a probability distribution ??

**vocabulary size layer**

```
... ...
```

**embedding size hidden layer**

**vocabulary size x embedding size weight matrix**

**vocabulary size inputs**

<table>
<thead>
<tr>
<th>all</th>
<th>answer</th>
<th>amtrak</th>
<th>is</th>
<th>love</th>
<th>need</th>
<th>you</th>
<th>what</th>
<th>zero</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**vocabulary size x embedding size weight matrix**

```
... ...
```

**vocabulary size inputs**

```
... ...
```

**vocabulary size inputs**

```
... ...
```

**vocabulary size inputs**

```
... ...
```

**vocabulary size inputs**

```
... ...
```
### Preliminaries

**Probability distributions**

**softmax** = normalize the logits

\[
\text{softmax} = \frac{e^{\text{logits}[i]}}{\sum_{j=1}^{|\text{logits}|} e^{\text{logits}[j]}}
\]

**cost** = cross entropy loss

\[
\text{cost} = - \sum_x p(x) \log \hat{p}(x)
\]

\[
= - \sum_i p_{\text{ground truth}}(\text{word} = \text{vocabulary}[i]) \log p_{\text{predictions}}(\text{word} = \text{vocabulary}[i])
\]

\[
= - \sum_i y_i \log \hat{y}_i
\]
Outline

Morning program
  Preliminaries
    Feedforward neural network
    Distributed representations
    Recurrent neural networks
    Sequence-to-sequence models
    Convolutional neural networks
  Semantic matching
  Learning to rank
  Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Recurrent neural networks

- Recurrent neural networks (RNNs) are typically used in scenarios where a sequence of inputs and/or outputs is being modelled.
- RNNs have memory that captures information about what has been computed so far.
- RNNs are called recurrent because they perform the same task for every element of a sequence, with output dependent on previous computations.
- RNNs can, in theory, make use of information in arbitrarily long sequences – in practice, however, they are limited to looking back only a few steps.
Recurrent cell

- word vector
- input vector
- output vector
- hidden state vector: $f$ (word vector, input vector)
- target distribution
- distribution over vocabulary
- next word
- cross entropy loss
- softmax
- distribution over vocabulary
- output vector 2
- next word

**rnn cell (language modelling)**
Recurrent cell (language modelling)

- **Preliminaries**
  - RNN cell
  - Word vector
  - Hidden state $t-1$
  - Hidden state $t$
  - Softmax
  - Cross entropy loss
  - Target distribution
  - Distribution over vocabulary
  - Output vector 2
  - Hidden state vector: $f(\text{hidden state}_{t-1}, \text{input vector})$
  - Next word
  - Target distribution
  - Softmax
  - Cross entropy loss
  - Next word
Recurrent neural networks

- RNN being unrolled (or unfolded) into full network
- **Unrolling**: write out network for complete sequence

Formulas governing computation:

- \( x_t \) input at time step \( t \)
- \( s_t \) hidden state at time step \( t \) – memory of the network, calculated based on
  - input at the current step
  - previous hidden state
- \( f \) usually nonlinearity, e.g., tanh or ReLU
  \( f \) can also be LSTM or GRU.

Image credits: Nature
Preliminaries

Language modeling using RNNs

- Language model allows us to predict probability of observing sentence (in a given dataset) as:
  \[ P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \]

- In RNN, set \( o_t = x_{t+1} \): we want output at step \( t \) to be actual next word

- Cross-entropy loss as loss function

- Training RNN similar to training a traditional NN: backpropagation algorithm, but with small twist:
  parameters shared by all time steps, so gradient at each output depends on calculations of previous time steps: Backpropagation Through Time
Vanishing and exploding gradients

- For training RNNs, calculate gradients for $U$, $V$, $W$ – ok for $V$ but for $W$ and $U$ ...

- Gradients for $W$:

$$\frac{\partial \mathcal{L}_3}{\partial W} = \frac{\partial \mathcal{L}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} = \sum_{k=0}^{3} \frac{\partial \mathcal{L}_3}{\partial o_3} \frac{\partial o_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

- More generally: $\frac{\partial \mathcal{L}}{\partial s_t} = \frac{\partial \mathcal{L}}{\partial s_m} \cdot \frac{\partial s_m}{\partial s_{m-1}} \cdot \frac{\partial s_{m-1}}{\partial s_{m-2}} \cdots \frac{\partial s_{t+1}}{\partial s_t} \Rightarrow \ll 1$

  $\quad < 1 \quad < 1 \quad < 1$

- Gradient contributions from far away steps become zero: state at those steps doesn't contribute to what you are learning

Long Short Term Memory [Hochreiter and Schmidhuber, 1997]

LSTMs designed to combat vanishing gradients through **gating** mechanism

\[
\begin{align*}
    i_t &= \sigma(x_t U^i + s_{t-1} W^i) \\
    f_t &= \sigma(x_t U^f + s_{t-1} W^f) \\
    o_t &= \sigma(x_t U^o + s_{t-1} W^o) \\
    g_t &= \tanh(x_t U^g + s_{t-1} W^g) \\
    c_t &= c_{t-1} \odot f + g \odot i \\
    s_t &= \tanh(c_t) \odot o
\end{align*}
\]

(\(\odot\) is elementwise multiplication)

Image credits:
https://commons.wikimedia.org/wiki/File:Peephole_Long_Short-Term_Memory.svg
Gated Recurrent Units

- GRU layer quite similar to that of LSTM layer, as are the equations:

\[
\begin{align*}
z &= \sigma(x_t U^z + s_{t-1} W^z) \\
r &= \sigma(x_t U^r + s_{t-1} W^r) \\
h &= \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\
s_t &= (1 - z) \circ h + z \circ s_{t-1}
\end{align*}
\]

- GRU has two gates: reset gate \( r \) and update gate \( z \).
  - Reset gate determines how to combine new input with previous memory; update gate defines how much of the previous memory to keep around
  - Set reset to all 1’s and update gate to all 0’s to get plain RNN model

- On many tasks, LSTMs and GRUs perform similarly
Bidirectional RNNs

- **Bidirectional RNNs** based on idea that output at time $t$ may depend on previous *and future* elements in sequence
  - Example: predict missing word in a sequence
- Bidirectional RNNs are two RNNs stacked on top of each other
- Output is computed based on hidden state of both RNNs

Outline

Morning program
Preliminaries
  Feedforward neural network
  Distributed representations
  Recurrent neural networks
  Sequence-to-sequence models
  Convolutional neural networks
Semantic matching
Learning to rank
Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Sequence-to-sequence models

Increasingly important: not just retrieval but also generation

- Machine translation, spoken results, chatbots, conversational interfaces, ... , but also snippets, query suggestion, query correction, ...

Basic sequence-to-sequence (seq2seq) model consists of two RNNs: an encoder that processes input and a decoder that generates output:

![Encoder Diagram](https://www.tensorflow.org/tutorials/seq2seq)

Each box represents cell of RNN (often GRU cell or LSTM cell). Encoder and decoder can share weights or, as is more common, use a different set of parameters
Attention mechanism Bahdanau et al. [2014]

Normal decoder model:

\[ h_t = f(x, h_{t-1}; \theta), \]

Decoder with attention:

\[ h_{dec}^t = g(x_{dec}, H^{encoder}, h_{dec}^{t-1}) \]
Decoder with attention  Luong and Manning [2016], Vinyals et al. [2015]:

\[ h_{t}^{\text{dec}} = g(x_{t}^{\text{dec}}, H^{\text{encoder}}, h_{t-1}^{\text{dec}}) \]
\[ = W_{\text{proj}} \cdot d_{t} \| \hat{h}_{t}^{\text{dec}}, \]

\| is the concatenation operator, and
\[ h_{t}^{\text{dec}} = f(x_{t}^{\text{dec}}, h_{t-1}^{\text{dec}}; \theta^{\text{dec}}) \] from above.

\[ d_{t} \text{ is calculated from } H^{\text{encoder}} \text{ by:} \]
\[ d_{t} = \sum_{i=1}^{n} a_{t,i} h_{i}^{\text{encoder}} \]
\[ a_{t} = \text{softmax}(u_{t}) \]
\[ u_{t,i} = v^{T} \tanh(W_{1} h_{i}^{\text{encoder}} + W_{2} h_{t}^{\text{dec}}), \]
Outline

Morning program
  Preliminaries
    Feedforward neural network
    Distributed representations
    Recurrent neural networks
    Sequence-to-sequence models
    Convolutional neural networks
  Semantic matching
  Learning to rank
  Entities

Afternoon program
  Modeling user behavior
  Generating responses
  Recommender systems
  Industry insights
  Q & A
Convoluational neural networks

What is a convolution? Intuition: sliding window function applied to a matrix

Example: convolution with $3 \times 3$ filter

Multiply values element-wise with original matrix, then sum. Slide over whole matrix.

Image credits:
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution
Convolutional neural networks

- Each layer applies different filters and combines results
- Pooling (subsampling) layers
- During training, CNN learns values of filters

- First layer may learn to detect edges from raw pixels in first layer
- Use edges to detect simple shapes in second layer
- Use shapes to detect higher-level features, such as facial shapes in higher layers
- Last layer: classifier using high-level features
CNNs in text

I like this movie very much!

- Instead of image pixels, inputs typically are word embeddings.
- For a 10 word sentence using a 100-dimensional embedding we would have a $10 \times 100$ matrix as our input.
- That's our "image"
- Typically 1-dimensional convolutions are used. I.e. filters slide over rows of the matrix (words).

Architecture for sentiment analysis Zhang and Wallace [2015]
CNNs in text

Example uses in IR

- MSR: how to learn semantically meaningful representations of sentences that can be used for Information Retrieval
- Recommending potentially interesting documents to users based on what they are currently reading
- Sentence representations are trained based on search engine log data
- Modeling interestingness with deep neural networks Gao et al. [2014]
  A latent semantic model with convolutional-pooling structure for information retrieval [Shen et al., 2014]
Take aways

1. **Information Retrieval (IR)** systems help people find the right (most useful) information in the right (most convenient) format at the right time (when they need it).

2. **Neural Network (NN)** is a function $F(x; \Theta)$ with (a large number of) parameters $\Theta$ that maps an input object $x$ (which can be text, image or arbitrary vector of features) to an output object $y$ (class label, sequence of class labels, text, image).

3. There three main architectures (classes of $F(x; \Theta)$): (i) feed-forward NN (FFNN), (ii) recurrent NN (RNN), (iii) convolutional NN (CNN).

4. **Embeddings** are vector representations of objects in a high-dimensional space that are learned during training. In many practical applications, these vectors reflect similarities between objects that are important for solving the task.

5. Other stuff, such as seq2seq, vanishing and exploding gradients, LSTM and GRU, attention mechanism, softmax, cross-entropy.