Overview

- Handling Overfitting
- Recurrent Neural Networks
Handling Overfitting
How to know your model overfits?

- Separate data into training, validation, and test set
- Train on training set only and evaluate on validation set
- Fine-tune hyperparameters and train again
- DO NOT touch the test set until the model is trained with a good performance on the training and test set
- Overfitting indicator: high performance on training set, low performance on validation set
Reduce Overfitting

## Regularization

Any modification to a learning algorithm intended to reduce its generalization error (NOT the training error).

Regularization methods:

- norm constraint
- dropout
- batch normalization
- data augmentation
- early stopping
- ...

Norm Constraint

- method for penalizing model complexity by adding a constraint to the cost function that balances against complexity
- minimize($\text{Loss}(\text{Data} \mid \text{Model}) + \text{complexity}(\text{Model})$)
- $L_2$ regularization (ridge regularization):
  - penalize very big weights
  - add a constraint to the loss function
  - $L(w, x) + \lambda \|w\|^2_2$
    - $w$ are weights and $x$ are input data
    - where $\frac{2}{2}$ represents the sum of squares of all feature weights
    - $L$ is the loss function
    - $\lambda$ is the coefficient that regulates the tradeoff between correct prediction and model simplification
Dropout

Powerful method to avoid overfitting by randomly ignoring subsets of neurons in a given layer during training, which means those neurons do not participate in producing a prediction for the data.

Why does that make sense?
A single neuron cannot fully rely on any one feature, so it is forced to spread its weights more, which has the effect of shrinking the squared norm of the weights.
The higher the value in dropout, the more neurons you keep per layer (1.0 - no dropout at all).
Data Augmentation

Add new training examples that are similar to the existing data but reasonably modified so that more variants can be learned.

Example of Modification

Recurrent Neural Networks
Why “recurrent”? 

- specialized for processing a sequence of inputs (of variable length)
- parameters are shared through different parts of the model across different time steps
- this parameter sharing allows information to persist across layers
- networks with loops, that is, repetitive (“recurring”) structures
Unfolding Recurrent Neural Networks

Unfolded RNN

Dagmar Gromann, 21 December 2018
Semantic Computing
Forward Propagation in this RNN

Assumptions:

• activation function on hidden layer is tanh
• output layer uses softmax
• output = discrete (e.g. words or characters)
• \( t \) represents the time steps:
  \( t = 1, \ldots, \tau \)
• \( U, W, V \) are weight matrices
• \( b \) and \( c \) are bias vectors

\[
\begin{align*}
a^{(t)} &= Wh^{(t-1)} + Ux^{(t)} + b \\
h^{(t)} &= \tanh(a^{(t)}) \\
o^{(t)} &= Vh^{(t)} + c \\
\hat{y}^{(t)} &= \text{softmax}(o^{(t)}) \\
L(\hat{y}, y) &= -\sum_{t=1}^{\tau} y^{(t)} \log \hat{y}^{(t)}
\end{align*}
\]
Networks with Output Recurrence

Flexibility of Recurrent Neural Networks (RNNs)

one to one
Vanilla NN (no R)
e.g. image classification

one to many
Sequence output
e.g. image captioning

many to one
Sequence input
e.g. sentiment analysis

many to many
Sequence input and sequence output
e.g. Machine Translation (NMT)

many to many
Synced sequence input and sequence output
e.g. video classification on frame level

Image adapted from Fei-Fei Li, Andrej Karpathy & Justin Johnson
Character-Level RNN Example

input chars:  

```
input layer
```

```
<table>
<thead>
<tr>
<th></th>
<th>“h”</th>
<th>“e”</th>
<th>“l”</th>
<th>“l”</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

hidden layer

```
<table>
<thead>
<tr>
<th></th>
<th>“e”</th>
<th>“l”</th>
<th>“l”</th>
<th>“l”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3</td>
<td>1.0</td>
<td>0.1</td>
<td>-0.3</td>
</tr>
<tr>
<td>0</td>
<td>-0.1</td>
<td>0.3</td>
<td>-0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>0</td>
<td>0.9</td>
<td>0.1</td>
<td>-0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>
```

output layer

```
<table>
<thead>
<tr>
<th></th>
<th>“e”</th>
<th>“l”</th>
<th>“l”</th>
<th>“l”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>0</td>
<td>2.2</td>
<td>3.0</td>
<td>-1.0</td>
<td>-1.5</td>
</tr>
<tr>
<td>0</td>
<td>-3.0</td>
<td>-0.3</td>
<td>1.2</td>
<td>-0.1</td>
</tr>
<tr>
<td>0</td>
<td>4.1</td>
<td>1.0</td>
<td>-1.1</td>
<td>2.2</td>
</tr>
</tbody>
</table>
```

W_{hh} = W_{xh} = W_{hy}
Backpropagation Through Time (BPTT)

Variant of backpropagation that works on a sequence in time:

- For each training epoch: start by training on shorter sequences and then progressively train on longer sequences
- For each length of sequence k: unfold the network (see previous slide) so that it resembles a normal feedforward network with k hidden layers
- Proceed with the standard backpropagation algorithm
- Issue: high cost of a single parameter update
- **Truncated BPTT**: process the sequence one timestep at a time, and every $t_1$ timesteps it runs BPTT for $t_2$ timesteps, which makes a parameter update cheaper if $t_2$ is small
Problems of RNNs

• Powerful but difficult to train
  – in backpropagation the gradient is multiplied a large number of times by the same weight matrix associated with the recurrent connections
  – if weights are small (< 1) => vanishing gradient problem
  – if weights are big (> 1) => exploding gradient problem

• Solution:
  – exploding gradient: clipping the norm of the exploded gradient when it is too large
  – vanishing gradient: relax non-linear dependency to linear dependency (LSTM, GRU, etc.)
Review of Lecture 9

• On which basis do cost function and activation function interact?
• What is backpropagation?
• What is the purpose of regularization in a neural network? How can this be achieved?
• What is a Recurrent Neural Network?
• How does backpropagation work with an RNN?
• What is a typical application scenario of an RNN?
• How can the problem of exploding gradients be treated in this type of network?