Course Outline

- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation
- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Random Forests
- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks

Topics of This Lecture

- Recap: Recurrent Neural Networks (RNNs)
  - Backpropagation through Time (BPTT)
  - Handling Vanishing Gradients
- Improved hidden units for RNNs
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRU)
- Applications of RNNs

Recall: Recurrent Neural Networks (RNNs)

- RNNs are regular NNs whose hidden units have additional connections over time.
  - You can unroll them to create a network that extends over time.
  - When you do this, keep in mind that the weights for the hidden are shared between temporal layers.
- RNNs are very powerful
  - With enough neurons and time, they can compute anything that can be computed by your computer.

Recall: Backpropagation Through Time (BPTT)

- Configuration
  - $h_i = \sigma(W_{input}x_i + W_{hidden}h_{i-1} + b)$
  - $y_i = \text{softmax}(W_{output}h_i)$
- Backpropagated gradient
  - For weight $w_{ij}$:
    $$\frac{\partial E_i}{\partial w_{ij}} = \sum_{1 \leq k \leq t} \left( \frac{\partial E_i}{\partial h_k} \cdot \frac{\partial h_k}{\partial h_{i-1}} \cdot \frac{\partial h_{i-1}}{\partial w_{ij}} \right)$$

Image source: Andrej Karpathy
Recap: Backpropagation Through Time (BPTT)

Analyzing the terms
- For weight $w_{ij}$:
  - This is the “immediate” partial derivative (with $h_{k-1}$ as constant)

Recap: Exploding / Vanishing Gradient Problem

- BPTT equations:
  \[
  \frac{\partial E_i}{\partial w_{ij}} = \sum_{t=1}^{t-k} \left( \frac{\partial E_i}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right)
  \]
  \[
  \frac{\partial h_k}{\partial h_l} = \prod_{t=1}^{t-k} \frac{\partial h_{t-1}}{\partial h_t} = \prod_{t=1}^{t-k} W_{hh}^d \text{diag}(\sigma'(h_{l-1}))
  \]
  
  (if $t$ goes to infinity and $l = t - k$)

  $\Rightarrow$ We are effectively taking the weight matrix to a high power.
  $\Rightarrow$ The result will depend on the eigenvalues of $W_{hh}$.
  - Largest eigenvalue > 1 $\Rightarrow$ Gradients may explode.
  - Largest eigenvalue < 1 $\Rightarrow$ Gradients will vanish.
  - This is very bad...

Recap: Gradient Clipping

- Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.
  
  **Algorithm 1 Pseudo-code**
  
  \[
  \text{if } \|g\| \geq \text{threshold then } g = \frac{\text{threshold}}{\|g\|} \cdot g
  \]
  
  $\Rightarrow$ This makes a big difference in RNNs

Handling Vanishing Gradients

- Vanishing Gradients are a harder problem
  - They severely restrict the dependencies the RNN can learn.
  - The problem gets more severe the deeper the network is.
  - It can be very hard to diagnose that Vanishing Gradients occur (you just see that learning gets stuck).

  Ways around the problem
  - Glorot/He initialization (see Lecture 12)
  - ReLU
  - More complex hidden units (LSTM, GRU)

ReLU to the Rescue

- Idea
  - Initialize $W_{hh}$ to identity matrix
  - Use Rectified Linear Units (ReLU)
  \[
  g(a) = \max \{0, a\}
  \]

- Effect
  - The gradient is propagated with a constant factor
  \[
  \frac{\partial y(a)}{\partial a} = \begin{cases} 
  1, & a > 0 \\
  0, & \text{else}
  \end{cases}
  \]

  $\Rightarrow$ Huge difference in practice!
Topics of This Lecture

• Recap: Recurrent Neural Networks (RNNs)
   Backpropagation through Time (BPTT)
   Problems with RNN Training
   Handling Vanishing Gradients

• Improved hidden units for RNNs
   Long-Short-Term Memory (LSTM)
   Gated Recurrent Units (GRU)

• Applications of RNNs

More Complex Hidden Units

• Target properties
   Want to achieve constant error flow through a single unit
   At the same time, want the unit to be able to pick up long-term connections or focus on short-term ones, as the problem demands.

• Ideas behind LSTMs
   Take inspiration from the design of memory cells
   Keep around memories to capture long distance dependencies
   Allow error messages to flow at different strengths depending on the inputs

Long Short-Term Memory

• RNNs can be seen as chains of repeating modules
   In a standard RNN, the repeating module has a very simple structure (e.g., a tanh)

Long Short-Term Memory

• LSTMs
   Repeating modules have 4 layers, interacting in a special way.

LSTMs: Core Ideas

• Cell state
   This is the key to LSTMs.
   It acts like a conveyor belt, information can flow along it unchanged.

• Gates
   The cell state can be modified through gates.
   Structure: sigmoid net layer + pointwise multiplication
   The sigmoid outputs values between 0 and 1
     0: Let nothing through
     1: Let everything through
     The gate layers are learned together with all other parameters.

Elements of LSTMs

• Forget gate layer
   Look at $h_{t-1}$ and $x_t$ and output a number between 0 and 1 for each dimension in the cell state $C_{t-1}$
     0: completely delete this,
     1: completely keep this.

• Example
   Task: try to predict the next word
   Cell state could include the gender of the present subject
     When we see a new subject, want to forget the gender of the old subject.

$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$
Elements of LSTMs

- **Update gate layer**
  - Decide what information to store in the cell state.
  - Sigmoid network (input gate layer) decides which values are updated.
  - tanh layer creates a vector of new candidate values \( \tilde{C}_t \) that could be added to the state.

- In the example
  - Add the gender of the new subject to the cell state.

\[
i_t = \sigma(W_i [h_{t-1}, x_t] + b_i) \\
\tilde{C}_t = \tanh(W_{\tilde{C}} [h_{t-1}, x_t] + b_{\tilde{C}})
\]

\[
i_t \cdot \tilde{C}_t
\]

Elements of LSTMs

- **Output gate layer**
  - Output is a filtered version of our gate state.
  - First, apply sigmoid layer to decide what parts of the cell state to output.
  - Then, pass the cell state through a tanh (to push the values to be between -1 and 1) and multiply it with the output of the sigmoid gate.

- In the example
  - Since we just saw a subject, might want to output information relevant to a verb (e.g., whether the subject is singular or plural).

\[
o_t = \sigma(W_o [h_{t-1}, x_t] + b_o) \\
h_t = o_t \cdot \tanh(C_t)
\]

RNN vs. LSTM

- LSTM just changes the form of the equation for \( h \) such that:
  1. More expressive multiplicative interactions become possible
  2. Gradients flow nicer
  3. The network can explicitly decide to reset the hidden state

- Those changes have a huge effect in practice
  - LSTMs perform much better than regular RNNs
  - Many applications have become possible with LSTMs that weren’t feasible before.

LSTMs in Practice

- LSTMs are currently highly en vogue
  - Popular default model for most sequence labeling tasks.
  - Very powerful, especially when stacked and made even deeper.
  - Most useful if you have lots and lots of data.
  - Very active research field

- Here are also some other ways of illustrating them

Extension: Gated Recurrent Units (GRU)

- Simpler model than LSTM
  - Combines the forget and input gates into a single update gate \( z_t \).
  - Similar definition for a reset gate \( r_t \), but with different weights.
  - In both cases, merge the cell state and hidden state.

- Empirical results
  - Performance similar to LSTM (no clear winner yet)
  - But GRU has fewer parameters.
GRUs: Intuition

- Effects
  - If reset is close to 0, ignore previous hidden state.
    ⇒ Allows model to drop information that is irrelevant in the future.
  - Update gate \( z \) controls how much of past state should matter now.
    ⇒ If \( z \) is close to 0, then we can copy information in that unit through many time steps!
    ⇒ Less vanishing gradients!

- Typical learned behaviors
  - Units with short-term dependencies often have active reset gate
  - Units with long-term dependencies have inactive update gates.

Topics of This Lecture

- Recap: Recurrent Neural Networks (RNNs)
  - Backpropagation through Time (BPTT)
  - Problems with RNN Training
  - Handling Vanishing Gradients
- Improved hidden units for RNNs
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRU)
- Applications of RNNs

Applications

- Machine Translation [Sutskever et al., 2014]

Language Model Results

- Example: Generating Shakespeare
  - Trained on all works of Shakespeare (4.4 MB of data)
  - Using a 3-Layer RNN with 512 hidden units per layer
Language Model Results

Example: Generating Wikipedia pages
- Trained on 100MB of Wikipedia data
- Using an LSTM

Example: Hallucinating Algebraic Geometry
- Trained on an Algebraic Geometry book
- Using a multilayer LSTM

Applications: Image Tagging

Simple combination of CNN and RNN
- Use CNN to define initial state $h_0$ of an RNN.
- Use RNN to produce text description of the image.

Results: Image Tagging

Spectacular results!
Results: Image Tagging

- Wrong, but one can still see why those results were selected...

Fun Application: Image to Story

Later on the eighth day, Billy was a friend of a man who lived on his own. He didn’t know how far away they were, and if he was to survive the fall. His mind raced, trying not to show any signs of weakness. The wind ruffled the snow and ice in the snow. He had no idea how many times he was going to climb into the mountains. He told me to stay on the ground for a while, but if I find out what’s going on, we should go on foot. Sam and Si Lei joined us in the army.

- Example: Generating a story from an image
  - Trained on corpus of adventure novels

More Results

Only Prince Darin knew how to run from the mountains, and once more, he could see the outline of a rider on horseback. The wind ruffled his hair in an attempt to locate the forest. He hadn’t been in such a state of mind before, but it was a good thing. All of them seemed to be doing the same thing. They didn’t know where they came from. The wind blew up the mountain peaks and disappeared into the sky, leaving trails behind the peaks of the mountains on Mount Fuji.

Application: Video to Text Description
Memory Networks

- Soft, differentiable memory
  - Stores <key, value> pairs
  - Input is matched to the stored keys
  - Output is the average over all values that correspond to the matched keys

- Key Idea
  - Make all steps differentiable.
  - Then all parameters (including access keys, stored values, etc.) can be learned with end-to-end supervised learning.

End-to-End Memory Networks

- A closer look at the memory mechanism
  - Values $c_i$
  - Keys $m_i$
  - Selection $p_i = \text{softmax}(u^T m_i)$
  - Output $o = \sum_i p_i c_i$
  - Input query $u$

  ⇒ Rely on sparsity of softmax to select a unique output value.

Improved Design

- Gated memory (e.g., Recurrent Entity Network)
  - Gating mechanism
  - Goal: Enable general computation with Neural Nets
  - Again key is to make all operations differentiable.
  - Memory + Access operators + Controller
  - Learn entire algorithms from examples.

Memory Networks

- Problem with this design
  - Softmax used for the selection involves a normalization over all stored keys.
  - Memory cells that are not accessed get almost zero gradient.
  - When a backpropagation step causes the accessed memory cell to change, this massively affects the gradient flow.

  ⇒ Together, this results in bad gradient propagation during learning.
  ⇒ Very finicky behavior...

Improved Design

- Gated memory (e.g., Recurrent Entity Network)
  - Gating mechanism
  - Goal: Enable general computation with Neural Nets
  - Again key is to make all operations differentiable.
  - Memory + Access operators + Controller
  - Learn entire algorithms from examples.

Neural Turing Machines

- Goal: Enable general computation with Neural Nets
  - Again key is to make all operations differentiable.
  - Memory + Access operators + Controller
  - Learn entire algorithms from examples.
References and Further Reading

- **RNNs**

- **LSTM**
  - C. Olah, *Understanding LSTM Networks*, blog post, August 2015.