Lecture 12: Recurrent Neural Networks
Recap of Lecture 11

- Convolutional neural networks (CNNs)
- Convolutional Layer and Pooling layer
- Workflow for Deep Learning
Outline for today

- Recurrent Neural Networks (RNNs)
- Teacher forcing
- Bi-directional RNNs
- Deep RNNs
- Long term dependencies and gated recurrent units

References: Deep Learning Book
Recurrent Neural Networks

- Processing time-sequenced data, though the time step $t$ needs not refer to the passage of time. RNNs can be applied to images.

- Need not respect the notion of causality: the network may have connections that go backward in time.

- Like CNNs, essential to RNNs is parameter sharing — this enables generalization across different sequence lengths/positions in time.

- Example: “I started my PhD study at UW-Madison in 2020” and “In 2020, I started my PhD study at UW-Madison” have the same info.

- Can be used in combination of ConvNet:
Basic Idea

- Include cycles in computational graph which capture the influence of the present value of a variable on its own value at a future step.

\[ h(t) = f(h(t-1), x(t); \theta), \]

where \( h(t) \) represents the state (hidden), and \( x(t) \) external signal.

![Diagram showing the basic idea of recurrent neural networks](image)
Advantages of RNNs

- Regardless of the sequence length, the learned model always has the same input size ⇒ generalization to different sequence lengths.

\[
\begin{align*}
    h^{(t)} &= g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \ldots, x^{(2)}, x^{(1)}) \\
    &= f(h^{(t-1)}, x^{(t)}; \theta).
\end{align*}
\]

- Parameter sharing: it is possible to use the same transition function function \( f \) with the same parameters at every time step ⇒ learn a single model, rather than a separate model for all possible time steps.

Any function involving recurrence can be considered a recurrent neural network.
RNN Types: Output at each time step

Example:

\[ a^{(t)} = b + W h^{(t-1)} + U x^{(t)}, \]
\[ h^{(t)} = \tanh(a^{(t)}), \]
\[ o^{(t)} = c + V h^{(t)}, \]
\[ \hat{y}^{(t)} = \text{softmax}(o^{(t)}), \]

Learnable parameters: \( b, c, U, V, W \)

Gradient of loss function can be computed by back propagation through time (BBTT)
Figure 10.4: An RNN whose only recurrence is the feedback connection from the output to the hidden layer. At each time step $t$, the input is $x(t)$, the hidden layer activation $h(t)$, the target $y(t)$, and the loss $L(t)$. (Left) Circuit diagram. (Right) Unfolded computational graph. Such an RNN is less powerful (can express a smaller set of functions) than those in the family represented by figure 10.3. The RNN in figure 10.3 can choose to put any information it wants about the past into its hidden representation $h$ and transmit $h$ to the future. The RNN in this figure is trained to put as specific output value in $o$, and $o$ is the only information it is allowed to send to the future. There are no direct connections from $h$ going forward. The previous $h$ is connected to the present only indirectly, via the predictions it was used to produce. Unless $o$ is very high-dimensional and rich, it will usually lack important information from the past. This makes the RNN in this figure less powerful, but it may be easier to train because each time step can be trained in isolation from the others, allowing greater parallelization during training.

Forward propagation begins with a specification of the initial state $h(0)$. Then, for each time step from $t=1$ to $t=\tau$, we apply the following update equations:

$$ a(t) = b + W h(t-1) + U x(t), \quad (10.8) $$

$$ h(t) = \tanh(a(t)), \quad (10.9) $$

$$ o(t) = c + V h(t), \quad (10.10) $$

Less expressive, but may be easier to train because each time step can be trained in isolation from the others, allowing greater parallelization during training.
RNN Types: Output after entire sequence

Figure 10.5: Time-unfolded recurrent neural network with a single output at the end of the sequence. Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target right at the end (as depicted here), or the gradient on the output can be obtained by back-propagating from further downstream modules.

\[
\hat{y}(t) = \text{softmax}(o(t)), \quad (10.11)
\]

where the parameters are the bias vectors \(b\) and \(c\) along with the weight matrices \(U\), \(V\) and \(W\), respectively, for input-to-hidden, hidden-to-output and hidden-to-hidden connections. This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. The total loss for a given sequence of \(x\) values paired with a sequence of \(y\) values would then be just the sum of the losses over all the time steps. For example, if \(L(t)\) is the negative log-likelihood of \(y(t)|\{x^{(1)}, \ldots, x^{(t)}\}\), then

\[
L(\tau) = \sum_{t} L(t) \quad (10.12)
\]

\[
= \sum_{t} \log p_{\text{model}}(y(t)|\{x^{(1)}, \ldots, x^{(t)}\}) \quad (10.13)
\]

where \(p_{\text{model}}(y(t)|\{x^{(1)}, \ldots, x^{(t)}\})\) is given by reading the entry for \(y(t)\) from the model's output vector \(\hat{y}(t)\). Computing the gradient of this loss function with respect to the parameters is an expensive operation. The gradient computation involves performing a forward propagation pass moving left to right through our illustration.
Teacher Forcing

- Problem of **hidden-to-hidden recurrence**: long computation & not parallelizable as the computation is sequential ⇒ Expensive to train

- **Output-to-hidden**: not expressive.

- Use **ground truth output** at $t$ as input at $t + 1$ for the training set.

- **Disadvantage**: input that network sees during training might be quite different from that during testing.

Some models can be trained with both teacher forcing and BBTT randomly (Bengio et al, 2015)
Bidirectional RNNs

- Example: speech recognition. Current sound may depend on the next few sounds (articulation, posing a question).
- Another example: Handwriting recognition.
- **Combination of 2 RNNs:** one moving forward in time, another backward in time.
- Generalization: multiple directions (e.g. 2D grid up, down, left, right).
- Compared with CNNs, RNNs applied to images are typically more expensive but allow for long-range lateral interactions between features in the same feature map.
Deep RNNs

- Make each computation $U$, $V$, $W$ deep.
- Replace “linear function” with some deep network.
- Problem: optimization more involved (not a priori clear whether model is trainable).

![Diagram of deep RNNs](image)

(a) The hidden recurrent state can be broken down into groups organized hierarchically.
(b) Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden, and hidden-to-output parts. This may lengthen the shortest path linking different time steps.
(c) The path-lengthening effect can be mitigated by introducing skip connections.
Challenge of long-term dependencies

- The recurrence relation can be described as matrix multiplication:

\[ h^{(t)} = W^\top h^{(t-1)} = (W^t)^\top h^{(0)} \]

- Using the eigendecomposition of \( W \):

\[ W = Q \Lambda Q^\top \]

- The recurrence is amplified by the number of time steps:

\[ h^{(t)} = Q^\top \Lambda^t Q h^{(0)} \]

- Let \( \lambda \) be the eigenvalues of \( \Lambda \), \( \lambda < 1 \) (\( \lambda > 1 \)) \( \Rightarrow h^{(t)} \) decays to zero (explodes); \( h^{(0)} \) not aligned with the largest eigenvector discarded.

- Network trainable if gradient \( \approx 0 \), gradient of long-term interaction \( \ll \) gradient of a short-term interaction.
Gated RNNs

• How to avoid difference between long- and short-term memory?

• Idea: Create paths through time that neither vanish nor explode.

• Change connection weights at each time step.

• Idea: Once relevant information of past states has been used, forget the old state. Use trainable parameters to decide when to do it.

• LSTM-layer (long-short-term-memory-layer)
The LSTM block diagram is illustrated in figure 10.16. The corresponding system of gating units that controls the flow of information. The most important component is the state unit $s_i$, which has a linear self-loop similar to the leaky component is the state unit $s_i$, which has a linear self-loop similar to the leaky 

**Forget gate:**

$$f^{(t)}_i = \sigma \left( b^f_i + \sum_j U^f_{i,j} x_j^{(t)} + \sum_j W^f_{i,j} h_j^{(t-1)} \right)$$

**Internal state update:**

$$s_i^{(t)} = f^{(t)}_i s_i^{(t-1)} + g^{(t)}_i \sigma \left( b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

**External input gate:**

$$g^{(t)}_i = \sigma \left( b^g_i + \sum_j U^g_{i,j} x_j^{(t)} + \sum_j W^g_{i,j} h_j^{(t-1)} \right)$$

**Output gate:**

$$h_i^{(t)} = \tanh \left( s_i^{(t)} \right) q_i^{(t)} , $$

$$q_i^{(t)} = \sigma \left( b^o_i + \sum_j U^o_{i,j} x_j^{(t)} + \sum_j W^o_{i,j} h_j^{(t-1)} \right)$$
Summary

- Recurrent Neural Networks (RNNs)
- Teacher forcing
- Bi-directional RNNs
- Deep RNNs
- Long term dependencies and gated recurrent units