CS6501: Deep Learning for Visual Recognition

Recurrent Neural Networks (RNNs)
Today’s Class

• Recurrent Neural Network Cell
• Recurrent Neural Networks (RNNs)
• Bi-Directional Recurrent Neural Networks (Bi-RNNs)
• Multiple-layer / Stacked / Deep Bi-Direction Recurrent Neural Networks
• LSTMs and GRUs.
• Applications in Vision: Caption Generation.
ACM Turing Award 2019

Yann LeCun
CNNs

Geoff Hinton
Backpropagation

Yoshua Bengio
GANs
Recurrent Neural Network Cell
Recurrent Neural Network Cell

\[ h_1 = \tanh(W_{hh} h_0 + W_{hx} x_1) \]
Recurrent Neural Network Cell

\[
h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)
\]

\[
y_1 = \text{softmax}(W_{hy}h_1)
\]
Recurrent Neural Network Cell

\[ y_1 = [0.1, 0.05, 0.05, 0.1, 0.7] \]
\[ h_1 = [0.1, 0.2, 0.3, -0.1] \]
\[ h_0 = [0, 0, 0, 0, 0, 0] \]
\[ x_1 = [0, 0, 1, 0, 0] \]
Recurrent Neural Network Cell

\[ h_0 \rightarrow RNN \rightarrow h_1 \]

\[ x_1 \rightarrow RNN \rightarrow y_1 \]
Recurrent Neural Network Cell

\[ h_0 \rightarrow RNN \rightarrow h_1 \]

\[ h_1 \rightarrow h_0 \]

\[ x_1 \rightarrow RNN \rightarrow h_1 \]
(Unrolled) Recurrent Neural Network

\[ h_0 \rightarrow RNN \rightarrow h_1 \rightarrow RNN \rightarrow h_2 \rightarrow RNN \rightarrow h_3 \]

\[ x_1 \rightarrow h_1 \rightarrow y_1 \]

\[ x_2 \rightarrow h_2 \rightarrow y_2 \]

\[ x_3 \rightarrow h_3 \rightarrow y_3 \]
(Unrolled) Recurrent Neural Network

cat

\( y_1 \)

\( h_1 \)

\( x_1 \)

the

\( RNN \)

\( h_0 \)

likes

\( y_2 \)

\( h_2 \)

\( x_2 \)

cat

\( RNN \)

\( h_1 \)

eating

\( y_3 \)

\( h_3 \)

\( x_3 \)

likes

\( RNN \)

\( h_2 \)
(Unrolled) Recurrent Neural Network

The cat likes positive / negative sentiment rating
Bidirectional Recurrent Neural Network

\[ \text{gato} \rightarrow y_1 \rightarrow h_1 \rightarrow h_2 \rightarrow y_2 \rightarrow \text{quieres} \rightarrow h_2 \rightarrow h_3 \rightarrow y_3 \rightarrow \text{comer} \]

\[ \text{the} \rightarrow x_1 \rightarrow \text{cat} \rightarrow x_2 \rightarrow \text{wants} \rightarrow x_3 \]
Stacked Recurrent Neural Network
Stacked Bidirectional Recurrent Neural Network

\[ y_1, y_2, y_3 \]

\[ h_0, h_1, h_2, h_3 \]

\[ \tilde{h}_0, \tilde{h}_1, \tilde{h}_2, \tilde{h}_3 \]

\[ x_1, x_2, x_3 \]
RNN in Pytorch

Recurrence layers

```python
class torch.nn.RNN(*args, **kwargs)
```

Applies a multi-layer Elman RNN with tanh or ReLU non-linearity to an input sequence.

For each element in the input sequence, each layer computes the following function:

\[ h_t = \tanh(w_{ih} \ast x_t + b_{ih} + w_{hh} \ast h_{t-1} + b_{hh}) \]

where \( h_t \) is the hidden state at time \( t \), and \( x_t \) is the hidden state of the previous layer at time \( t \) or \( \text{input}_t \) for the first layer. If nonlinearity='relu', then ReLU is used instead of tanh.

Parameters:
- **input_size** – The number of expected features in the input \( x \)
- **hidden_size** – The number of features in the hidden state \( h \)
- **num_layers** – Number of recurrent layers.
- **nonlinearity** – The non-linearity to use ['tanh'|'relu']. Default: 'tanh'
- **bias** – If False, then the layer does not use bias weights \( b_{ih} \) and \( b_{hh} \). Default: True
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If True, becomes a bidirectional RNN. Default: False
LSTM Cell (Long Short-Term Memory)

\[
\begin{align*}
    i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
    f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
    c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
    o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
    h_t &= o_t \tanh(c_t)
\end{align*}
\]
Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

\[ i_t = \text{sigmoid}(W_i x_t + b_i + W_h h_{(t-1)} + b_i) \]
\[ f_t = \text{sigmoid}(W_f x_t + b_f + W_h h_{(t-1)} + b_f) \]
\[ g_t = \tanh(W_g x_t + b_g + W_h h_{(t-1)} + b_g) \]
\[ o_t = \text{sigmoid}(W_o x_t + b_o + W_h h_{(t-1)} + b_o) \]
\[ c_t = f_t \odot c_{(t-1)} + i_t \odot g_t \]
\[ h_t = o_t \odot \tanh(c_t) \]

where \( h_t \) is the hidden state at time \( t \), \( c_t \) is the cell state at time \( t \), \( x_t \) is the hidden state of the previous layer at time \( t \) or input, for the first layer, and \( i_t, f_t, g_t, o_t \) are the input, forget, cell, and output gates, respectively.

Parameters:

- `input_size` – The number of expected features in the input \( x \)
- `hidden_size` – The number of features in the hidden state \( h \)
- `num_layers` – Number of recurrent layers.
- `bias` – If False, then the layer does not use bias weights \( b_i, b_f, b_g, b_o \). Default: True
- `batch_first` – If True, then the input and output tensors are provided as (batch, seq, feature).
- `dropout` – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer.
- `bidirectional` – If True, becomes a bidirectional RNN. Default: False
GRU in Pytorch

```python
class torch.nn.GRU(*args, **kwargs) [source]

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

\[
\begin{align*}
r_t &= \text{sigmoid}(W_{ir} x_t + b_{ir} + W_{hr} h_{t-1} + b_{hr}) \\
z_t &= \text{sigmoid}(W_{iz} x_t + b_{iz} + W_{hz} h_{t-1} + b_{hz}) \\
n_t &= \text{tanh}(W_{in} x_t + b_{in} + r_t \times (W_{hn} h_{t-1} + b_{hn})) \\
h_t &= (1 - z_t) \times n_t + z_t \times h_{t-1}
\end{align*}
\]

where \( h_t \) is the hidden state at time \( t \), \( x_t \) is the hidden state of the previous layer at time \( t \) or input \( t \) for the first layer, and \( r_t, z_t, n_t \) are the reset, input, and new gates, respectively.

Parameters:
- `input_size` – The number of expected features in the input \( x \)
- `hidden_size` – The number of features in the hidden state \( h \)
- `num_layers` – Number of recurrent layers.
- `bias` – If False, then the layer does not use bias weights \( b_{ih} \) and \( b_{hh} \). Default: True
- `batch_first` – If True, then the input and output tensors are provided as (batch, seq, feature)
- `dropout` – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- `bidirectional` – If True, becomes a bidirectional RNN. Default: False
```
Questions?