Deep Learning for Computer Vision: Sequences – Tutorial
RNNs, LSTM, Transformers

Akhiad Bercovich
Agenda

• RNNs recap and issues
• LSTM
• Transformers in computer vision
(Simple) Recurrent Neural Network

The state consists of a single "hidden" vector $h$:

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

Slides credit: Justin Johnson (EECS-498-007, UMich)
Q. Multilayer RNNs

\[ h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \]

\[ y_t = W_{hy} h_t \]
Searching for interpretable cells
Searching for interpretable cells

```c
/* unpack a filter field's string representation from user-space */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* of the currently implemented string fields, PATH_MAX defines the longest valid length. */
```
Searching for interpretable cells

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell
Searching for interpretable cells

Cell sensitive to position in line:
The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission
Searching for interpretable cells

```c
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask, siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig){
        if (current->notifier)
            if (sigismember(current->notifier_mask, sig)) {
                if ((current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
    }
    collect_signal(sig, pending, info);
    return sig;
}
```

if statement cell
Searching for interpretable cells

code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]

LSTM

\[
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix}
= \begin{pmatrix}
  \sigma \\
  \sigma \\
  \sigma \\
  \tanh
\end{pmatrix}
W \begin{pmatrix}
  h_{t-1} \\
  x_t
\end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \tanh(c_t)
\]

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Vanilla RNN Gradient Flow

\[ h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \]
\[ = \tanh \left( (W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]
\[ = \tanh \left( W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \]

Bengio et al., "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al., "On the difficulty of training recurrent neural networks", ICML 2013
Vanilla RNN Gradient Flow

Backpropagation from $h_t$ to $h_{t-1}$ multiplies by $W$ (actually $W_{hh}^T$)

$$
\begin{align*}
  h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
  &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\
  &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)
\end{align*}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}
$$

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \\
\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh} h_{t-1} + W_{xh} x_t)W_{hh}
\]

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left( \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}
\]
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W^{T-1} \frac{\partial h_1}{\partial W}
\]

• What if we assumed no non-linearity?

• Largest singular value > 1:
  Exploding gradients

• Largest singular value < 1:
  Vanishing gradients
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

What if we assumed no non-linearity?

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

Largest singular value > 1: Exploding gradients

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \begin{pmatrix} W^T \end{pmatrix}_{-1} \frac{\partial h_1}{\partial W}
\]

Largest singular value < 1: Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

```python
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```
Vanilla RNN Gradient Flow

Gradients over multiple time steps:

\[
\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}
\]

• What if we assumed no non-linearity?

\[
\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left[ W^{T-1} \frac{\partial h_1}{\partial W} \right]
\]

• Largest singular value > 1:

Exploding gradients

Largest singular value < 1:

Vanishing gradients

→ Change RNN architecture
Long Short Term Memory (LSTM)

Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1} \right) \right) \]

LSTM

\[
\begin{pmatrix}
  i \\
  f \\
  o \\
  g
\end{pmatrix} =
\begin{pmatrix}
  \sigma \\
  \sigma \\
  \sigma \\
  \tanh
\end{pmatrix}
W
\begin{pmatrix}
  h_{t-1} \\
  x_t
\end{pmatrix}
\]

\[ c_t = f \odot c_{t-1} + i \odot g \]

\[ h_t = o \odot \tanh(c_t) \]

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997
Vanilla RNN

\[ h_t = \tanh \left( W \left( h_{t-1}, x_t \right) \right) \]
LSTM

\[ \begin{align*}
C_t &= f \odot C_{t-1} + i \odot g \\
h_t &= o \odot \tanh(c_t)
\end{align*} \]

sigmoid \quad \tanh \quad \text{pointwise multiplication} \quad \text{pointwise addition} \quad \text{vector concatenation}
LSTM

\[
\begin{align*}
C_t &= f \odot C_{t-1} + i \odot g \\
C_t &= o \odot \tanh(c_t) \\
C_t &= o \odot \tanh(c_t)
\end{align*}
\]

\[
(i) = \begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix}, \quad \sigma = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \end{pmatrix}, \quad W = \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}
\]

sigmoid \quad \tanh \quad \text{pointwise multiplication} \quad \text{pointwise addition} \quad \text{vector concatenation}
LSTM

\[
\begin{align*}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \\
\mathbf{c}_t &= \sigma(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{x}_t + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{U}_h \mathbf{x}_t + \mathbf{b}_h)
\end{align*}
\]

where \(\sigma\) is the sigmoid activation function and \(\mathbf{W}, \mathbf{U}, \mathbf{b}\) are weight matrices and biases respectively.
LSTM

\[
\begin{align*}
    c_t &= f \odot c_{t-1} + i \odot g \\
    h_t &= o \odot \tanh(c_t)
\end{align*}
\]

- forget gate
- cell state
- input gate
- output gate

sigmoid

tanh

pointwise multiplication

pointwise addition

vector concatenation
LSTM

\[
\begin{align*}
  c_t &= f \odot c_{t-1} + i \odot g \\
  h_t &= o \odot \tanh(c_t)
\end{align*}
\]

sigmoid \hspace{1cm} \text{tanh} \hspace{1cm} \text{pointwise multiplication} \hspace{1cm} \text{pointwise addition} \hspace{1cm} \text{vector concatenation}
LSTM

\[ c_{t-1} \text{ only elementwise multiplication by } f, \text{ no matrix multiply } \]

\[
\begin{pmatrix}
i \\
f \\
o \\
g
\end{pmatrix} = \begin{pmatrix}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{pmatrix} W \begin{pmatrix} h_{t-1} \\
x_t \end{pmatrix}
\]

\[ c_t = f \odot c_{t-1} + i \odot g \]

\[ h_t = o \odot \tanh(c_t) \]
Long Short Term Memory (LSTM)
[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell

\[
\begin{pmatrix}
i \\
f \\
o \\
g
\end{pmatrix} =
\begin{pmatrix}
\sigma \\
\sigma \\
\sigma \\
tanh
\end{pmatrix}
W
\begin{pmatrix}
h_{t-1} \\
x_t
\end{pmatrix}
\]

\[
c_t = f \odot c_{t-1} + i \odot g
\]

\[
h_t = o \odot \text{tanh}(c_t)
\]
Multilayer LSTMs
LSTM in Pytorch

```
nn.LSTM(embedding_dim, hidden_dim, n_layers, dropout=drop_prob)
```
Do LSTMs solve the vanishing gradient problem?

YES!
The LSTM architecture makes it easier for RNNs to preserve information over many timesteps
- e.g. if $f = 1$ and $i = 0$, then the information of that cell is preserved indefinitely
- By contrast, it’s harder for vanilla RNNs to learn a recurrent weight matrix $W_h$ that preserves info in hidden state

Actually NO…
LSTM doesn’t guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies
Three Ways of Processing Sequences

Recurrent Neural Network
- **Works on** Ordered Sequences
- **(+)** large and adaptive receptive field via hidden state
- **(-)** Not parallelizable: need to process states sequentially

1D Convolution
- **Works on** Multidimensional Grids
- **(-)** Fixed receptive field. Need to stack many layers to have a decent one
- **(+)** Highly parallelizable

Self-Attention
- **Works on** Sets
- **(+)** receptive filed = entire sequence
- **(+)** parallelizable
- **(-)** Very memory intensive

Slide credit: Justin Johnson (EECS-498-007, UMich)
Transformer Layer

**Input:** $x_1, \ldots, x_n$ ($n$ tokens in $D$ dimensions)

**Output:** $y_1, \ldots, y_n$ ($n$ tokens in $D$ dimensions)

Highly scalable

Highly parallelizable
nn.Transformer(d_model=512, nhead=8, num_encoder_layers=6, num_decoder_layers=6, 
   dim_feedforward=2048, dropout=0.1, activation='relu')
Vision Transformers (ViT)

Image Transformer Parmar et al. 2018

Sequences? Pixel conditioned on generated pixels
Image Transformer Parmar et al. 2018

Queries – current pixel embeddings
Keys – neighbor pixel embeddings
Values - neighbor pixel embeddings
**Images static**

Perceptron → Convolution → CNN

**Sequences time-dependent**

ViT → Recurrent NN → Memory → Attention

Arrow connects static images to time-dependent sequences.