Long Short-Term Memory in Recurrent Neural Network

Shixiang Gu, Andrey Malinin

sg717@cam.ac.uk, am969@cam.ac.uk

November 20, 2014
Outline

NN: Review
Early RNN
LSTM
Modern RNN
Discriminative Model learns $P(Y|X)$

Generative Model learns $P(X, Y)$, or $P(X)$

Supervised Learning uses pairs of input $X$ and output $Y$, and aims to learn $P(Y|X)$ or $f: X \rightarrow Y$

Unsupervised Learning uses input $X$ only, and aims to learn $P(X)$ or $f: X \rightarrow H$, where $H$ is “better” representation of $X$

Models can be stochastic or deterministic

In this presentation, we focus on deterministic, supervised, discriminative model based on Recurrent Neural Network (RNN)/Long Short-Term Memory (LSTM)

Applications: Speech Recognition, Machine Translation, Online Handwriting Recognition, Language Modeling, Music Composition, Reinforcement Learning, etc.
input: $X \in \mathbb{R}^Q$
target output: $Y \in \mathbb{R}^D$
predicted output: $\hat{Y} \in \mathbb{R}^D$
hidden: $H, Z \in \mathbb{R}^P$
weights: $W_X \in \mathbb{R}^{P\times Q}, W_Y \in \mathbb{R}^{D\times P}$
activations: $f : \mathbb{R}^P \rightarrow \mathbb{R}^P, g : \mathbb{R}^D \rightarrow \mathbb{R}^D$
loss function: $L : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$
loss: $E \in \mathbb{R}$

**objective:** minimize total loss $E$ for $(X^{(i)}, Y^{(i)})_{i=1}^N$ training examples

Forward Propagation:
$Z = W_X \cdot X$
$H = f(Z)$
$\hat{Y} = g(W_Y \cdot H)$
$E = L(Y, \hat{Y})$
Activation functions

Common activation functions \( f : X \longrightarrow Y \) on \( X, Y \in \mathbb{R}^D \):

**Linear:** \( f(X) = X \)

**Sigmoid/Logistic:** \( f(X) = \frac{1}{1+e^{-X}} \)

**Rectified Linear (ReLU):** \( f(X) = \max(0, X) \)

**Tanh:** \( f(X) = \tanh(X) = \frac{e^X - e^{-X}}{e^X + e^{-X}} \)

**Softmax:** \( f : Y_i = \frac{e^{X_i}}{\sum_{j=1}^{D} e^{X_j}} \)

Most activations are **element-wise** and **non-linear**

Derivatives are easy to compute
Backpropagation (i.e. chain rule):
assume \( g(X) = X \),
\[ L(Y, \hat{Y}) = \frac{1}{2} \sum_{i=1}^{D} (Y_i - \hat{Y}_i)^2: \]

\[
\frac{\partial E}{\partial \hat{Y}} = \hat{Y} - Y, \quad \frac{\partial \hat{Y}}{\partial H} = W_Y, \quad \frac{\partial H}{\partial Z} = f'(Z)
\]

\[
\frac{\partial E}{\partial Z} = \frac{\partial E}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial H} \frac{\partial H}{\partial Z} = (W_Y^T \cdot (\hat{Y} - Y)) \odot f'(Z)
\]

\[
\frac{\partial E}{\partial W_Y} = \frac{\partial E}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial W_Y} = (\hat{Y} - Y) \otimes H
\]

\[
\frac{\partial E}{\partial W_X} = \frac{\partial E}{\partial Z} \frac{\partial Z}{\partial W_X} = \frac{\partial E}{\partial Z} \otimes X
\]
*abusing matrix calculus notations

Stochastic Gradient Descent
learning rate: \( \epsilon > 0 \)
\[ W = W - \epsilon \frac{\partial E}{\partial W}, \forall W \in \{W_Y, W_X\} \]
Why Recurrent?

- Human brain is a recurrent neural network, i.e. has feedback connections
- A lot of data is sequential and dynamic (can grow or shrink)
- RNNs are Turing-Complete
  [Siegelmann and Sontag, 1995, Graves et al., 2014]
Unrolling RNNs

Figure: RNN

Figure: Unrolled RNN
RNN (1980s-)

Forward Propagation:
Given $H_0$

$Z_t = W_X \cdot X_t + W_H \cdot H_{t-1}$

$H_t = f(Z_t)$

$\hat{Y}_t = g(W_Y \cdot H_t)$

$E_t = L(Y_t, \hat{Y}_t)$

$E = \sum_{t=1}^{T} E_t$

RNN=Very Deep NN w/ tied weights

Sequence learning: unknown input len $\rightarrow$ unknown output len
Vanishing/exploding gradients (1991)

**Fundamental Problem in Deep Learning [Hochreiter, 1991]:**

Let  \( v_{j,t} = \frac{\partial E}{\partial Z_{j,t}} \)

\[ v_{i,t-1} = f'(Z_{i,t-1}) \cdot \sum_{j=1}^{P} w_{ji} \cdot v_{j,t} \]

\[ \frac{\partial v_{i,t-q}}{\partial v_{j,t}} = \sum_{l_1=1}^{P} \cdots \sum_{l_{q-1}=1}^{P} \prod_{m=1}^{q} f'_m(Z_{lm,t-m}) \cdot w_{lm,l_{m-1}} \]

| \( f'_m(Z_{lm,t-m}) \cdot w_{lm,l_{m-1}} \) | > 1.0  \( \forall m \rightarrow \) explodes

| \( f'_m(Z_{lm,t-m}) \cdot w_{lm,l_{m-1}} \) | < 1.0  \( \forall m \rightarrow \) vanishes

Gradient vanishes/increases exponentially in terms of time steps \( T \)

e.g. “butterfly effect”
Early work on RNN (1980s-90s)

Exact gradient (TOO HARD!(back then)):

- **Backpropagation Through Time** (BPTT) [Werbos, 1990]

Approximate gradient:

- **Real Time Recurrent Learning** (RTRL)
  [Robinson and Fallside, 1987, Williams and Zipser, 1989]

- **Truncated BPTT** [Williams and Peng, 1990]

Non-gradient methods:

- **Random Guessing** [Hochreiter and Schmidhuber, 1996]

Unsupervised “pretraining”:

- **History Compressor** [Schmidhuber, 1992]

Others:

- **Time-delay Neural Network** [Lang et al., 1990], **Hierarchical RNNs** [El Hihi and Bengio, 1995], **NARX** [Lin et al., 1996] and more...

History compressor (HC) [Schmidhuber, 1992] = unsupervised, greedy layer-wise pretraining of RNN
(Recall in standard DNN: unsupervised pretraining (AE, RBM) [Hinton and Salakhutdinov, 2006] → supervised
[Krizhevsky et al., 2012])
Forward Propagation:
Given $H_0, X_0, Y_t = X_{t+1}$
$Z_t = W_{XH} \cdot X_{t-1} + W_H \cdot H_{t-1}$
$H_t = f(Z_t)$
$\hat{Y}_t = g(W_Y \cdot H_t + W_{XY} \cdot X_t)$
$E_t = L(Y_t, \hat{Y}_t)$
$E = \sum_{t=0}^{T-1} E_t$

$X' \leftarrow \{ X_0, H_0, (t, X_t) \land t \ni X_t \neq \hat{Y}_{t-1} \}$
→ train new HC using $X'$

Greedily pretrain RNN, using “surprises” from previous step
Basic “LSTM” (1991-95)

Idea: Have separate linear units that simply deliver errors
[Hochreiter and Schmidhuber, 1997]

Forward Propagation*:
Let $C_t \in \mathbb{R}^P = \text{“cell state”}$,
Given $H_0, C_0$

\[ C_t = C_{t-1} + f_1(W_X \cdot X_t + W_H \cdot H_{t-1}) \]
\[ H_t = f_2(C_t) \]

BPTT: Truncate gradients outside the cell

*approximately
Idea: Have additional controls for R/W access (input/output gates) [Hochreiter and Schmidhuber, 1997]

Forward Propagation*:
Given $H_0, C_0$

$C_t = C_{t-1} + i_t \odot f_1(W_X \cdot X_t + W_H \cdot H_{t-1})$

$H_t = o_t \odot f_2(C_t)$

$i_t = \sigma(W_{i,X} \cdot X_t + W_{i,C} \cdot C_{t-1} + W_{i,H} \cdot H_{t-1})$

$o_t = \sigma(W_{o,X} \cdot X_t + W_{o,C} \cdot C_t + W_{o,H} \cdot H_{t-1})$

BPTT: Truncate gradients outside the cell

*approximately
Modern LSTM (full version) (2000s-)

Idea: Have control for forgetting cell state (forget gate)
[Gers, 2001, Graves, 2006]

Forward Propagation:
Given $H_0$, $C_0$
$C_t = f_t \odot C_{t-1} + i_t \odot f_1(W_X \cdot X_t + W_H \cdot H_{t-1})$
$H_t = o_t \odot f_2(C_t)$
$i_t = \sigma(W_{i,X} \cdot X_t + W_{i,C} \cdot C_{t-1} + W_{i,H} \cdot H_{t-1})$
$o_t = \sigma(W_{o,X} \cdot X_t + W_{o,C} \cdot C_t + W_{o,H} \cdot H_{t-1})$
$f_t = \sigma(W_{f,X} \cdot X_t + W_{f,C} \cdot C_{t-1} + W_{f,H} \cdot H_{t-1})$
BPTT: Use exact gradients
Recent Progress in RNN

Echo-State Network (ESN): [Jaeger and Haas, 2004]
- only train hidden-output weights
- other weights drawn from carefully-chosen distribution and fixed

Hessian-Free (HF) optimization: [Martens and Sutskever, 2011]
- approximate second-order method
- outperformed LSTM on small-scale tasks

Momentum methods: [Sutskever, 2013]
- Could we get some good results without HF?
- Yes! With “good” initialization & aggressive momentum scheduling
- Nesterov’s accelerated gradient?
DEMO!


Imagenet classification with deep convolutional neural networks.


