CS 6316 Machine Learning

CNNs and RNNs

Yangfeng Ji

Information and Language Processing Lab
Department of Computer Science
University of Virginia

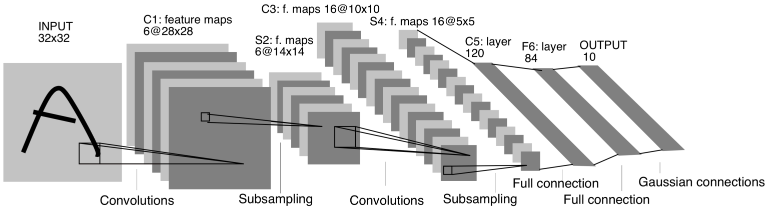
Overview

1. Convolutional Neural Networks
2. Recurrent Neural Networks
3. RNN Language Modeling
4. Challenge of Training RNNs

Convolutional Neural Networks

LeNet-5

A classical neural network architecture designed for handwritten and machine-printed character recognition.



This architecture repeat the two components twice before connecting with a fully-connected layer

- convolutional layer
- subsampling (pooling) layer

[LeCun et al., 1998]

Convolutional Operations

1-D convolutional operation is defined as

$$c_j = \boldsymbol{m}^\top \boldsymbol{x}_{j:j+n-1} \tag{1}$$

where $\boldsymbol{m} \in \mathbb{R}^n$ is convolutional filter with window size n, $\boldsymbol{x} \in \mathbb{R}^T$ input signal with size T, and $j \leq T - n + 1$.

Convolutional Operations

1-D convolutional operation is defined as

$$c_j = \boldsymbol{m}^\top \boldsymbol{x}_{j:j+n-1} \tag{1}$$

where $\boldsymbol{m} \in \mathbb{R}^n$ is convolutional filter with window size n, $\boldsymbol{x} \in \mathbb{R}^T$ input signal with size T, and $j \leq T - n + 1$.

Example

With

- $\boldsymbol{m} = [m_1, m_2, m_3]^\top$, and
- $\boldsymbol{x} = [x_1, x_2, x_3, x_4, \ldots, x_T]^\top$,

when $j = 2$

$$c_2 = m_1 x_2 + m_2 x_3 + m_3 x_4 \tag{2}$$

Convolutional Operations (II)

An example of 1-D convolutional operations

Input word embedding

0.8	1.2	3.5	1.4
0.7	0.1	0.5	0.8
1.3	2.4	0.1	0.5
2.1	1.6	3.0	4.1
3.0	0.6	0.3	1.5

Results

1.63	3.76
0.22	0.59
2.54	0.39
2.11	3.57
0.93	0.51

t=1	0.1	1.0	0.1	0
t=2	0	0.1	1.0	0.1

Convolutional filter
at each time step

Pooling

There are three popularly used pooling techniques

- Max pooling

Inputs

1.63	**3.76**
0.22	**0.59**
2.54	0.39
2.11	**3.57**
0.93	0.51

→ 1-D Max pooling

Results

3.76
0.59
2.54
3.57
0.93

Pooling

There are three popularly used pooling techniques

- Max pooling

Inputs

1.63	**3.76**
0.22	**0.59**
2.54	0.39
2.11	**3.57**
0.93	0.51

1-D Max pooling

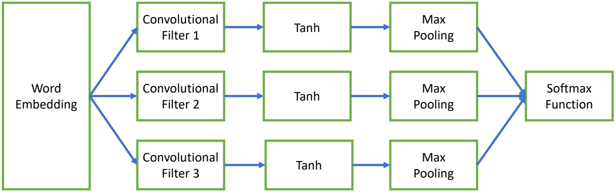
Results

3.76
0.59
2.54
3.57
0.93

- Average pooling [LeCun et al., 1998]
- Min pooling

TextCNN

A simple and effective convolutional neural network architecture for text classification [Kim, 2014]



- `torch.nn.Conv1d`: convolutional operation on each dimension of the word embeddings (no cross-dimension convolution)
- `torch.tanh`
- `torch.max`: max pooling on each dimension of the word embeddings
- `torch.cat`: concatenate three vectors from max pooling to form one single vector
- In actual implementation, the input is a 3-D tensor instead of a 2-D matrix

Advantages of CNNs

Comparing to Feed-forward NNs: Parameter sharing, sparse connections

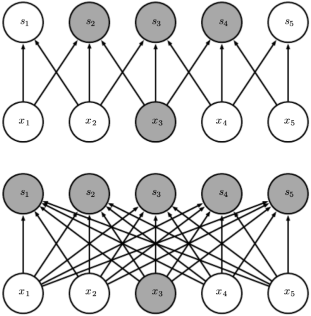


Figure: (1) upper plot: convolutional layer; (2) lower plot: fully-connected layer.

[Goodfellow et al., 2016]

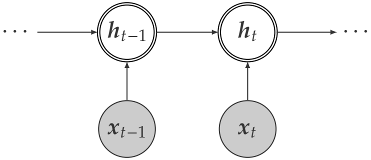
Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

A simple RNN is defined by the following recursive function

$$\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{3}$$

and depicted as



where

- $\boldsymbol{h}_{t-1}$: hidden state at time step $t-1$
- $\boldsymbol{x}_t$: input at time step t
- $\boldsymbol{h}_t$: hidden state at time step t

A Simple Transition Function

In the simplest case, the transition function f is defined with an element-wise Sigmoid function and a linear transformation of $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) = \sigma(\mathbf{W}_h \boldsymbol{h}_{t-1} + \mathbf{W}_i \boldsymbol{x}_t + \boldsymbol{b}) \quad (4)$$

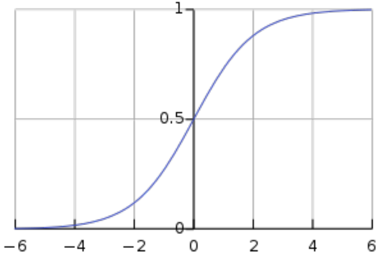
where

- $\boldsymbol{x}_t$: input word embedding
- $\boldsymbol{h}_{t-1}$: hidden statement from previous time step
- $\mathbf{W}_h$: parameter matrix for hidden states
- $\mathbf{W}_i$: parameter matrix for inputs
- $\boldsymbol{b}$: bias term (also a parameter)

Sigmoid Function

A Sigmoid function with one-dimensional input $x \in (-\infty, \infty)$

$$\sigma(x) = \frac{1}{1 - e^{-x}}$$



The potential numeric issue caused by the Sigmoid function

- $\sigma(x) \longrightarrow 1$ with $x \gg 6$
- $\sigma(x) \longrightarrow 0$, $x \ll -6$

The output of the Sigmoid function will approximate a constant, when the input value is beyond certain ranges

Unfolding RNNs

We can unfold this recursive definition of a RNN

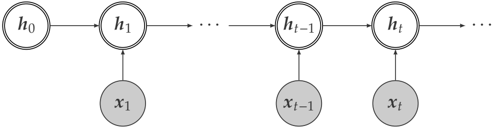
$$\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) \tag{5}$$

Unfolding RNNs

We can unfold this recursive definition of a RNN

$$h_t = f(x_t, h_{t-1}) \tag{5}$$

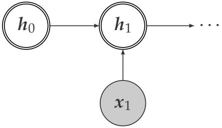
as



$$\begin{aligned} h_t &= f(x_t, f(x_{t-1}, h_{t-2})) \\ &= f(x_t, f(x_{t-1}, f(x_{t-2}, h_{t-3}))) \\ &= \cdots \\ &= f(x_t, f(x_{t-1}, f(x_{t-2}, \cdots f(x_1, h_0) \cdots))) \end{aligned} \tag{6}$$

Base Condition

Base condition defines the starting point of the recursive computation

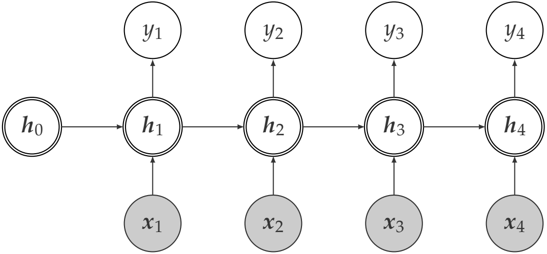


$$h_t = f(x_t, f(x_{t-1}, f(x_{t-2}, \cdots f(x_1, h_0) \cdots))) \tag{7}$$

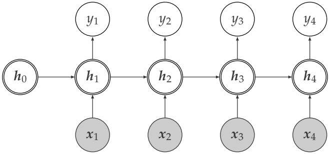
- h_0: zero vector or parameter
- x_1: input at time $t = 1$

RNN for Sequential Prediction

In general, RNNs can be used for any sequential modeling tasks



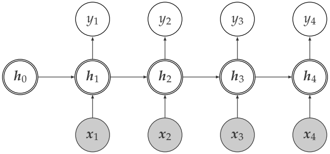
Sequential Modeling as Classification



- Prediction at each time step t

$$\hat{y}_t = \underset{y}{\operatorname{argmax}}\, P(y; \boldsymbol{h}_t) \tag{8}$$

Sequential Modeling as Classification



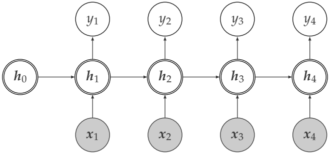
- Prediction at each time step t

$$\hat{y}_t = \operatorname*{argmax}_{y} P(y; \boldsymbol{h}_t) \tag{8}$$

- Loss at single time step t

$$L_t(y_t, \hat{y}_t) = -\log P(y_t; \boldsymbol{h}_t) \tag{9}$$

Sequential Modeling as Classification



- Prediction at each time step t

$$\hat{y}_t = \operatorname*{argmax}_y P(y; \boldsymbol{h}_t) \tag{8}$$

- Loss at single time step t

$$L_t(y_t, \hat{y}_t) = -\log P(y_t; \boldsymbol{h}_t) \tag{9}$$

- The total loss

$$\ell = \sum_{t=1}^{T} L_t(y_t, \hat{y}_t) \tag{10}$$

RNN Language Modeling

Language Models

A language model defines the probability of x_t given $\boldsymbol{x} = (x_1, x_2, \ldots, x_{t-1})$ as

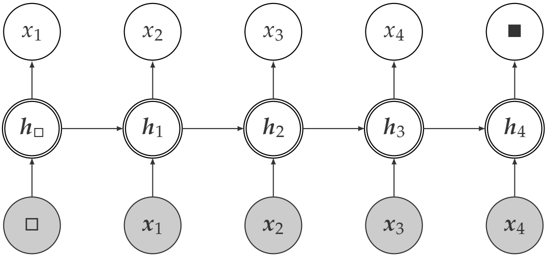
$$P(x_t \mid x_1, \ldots, x_{t-1}) \tag{11}$$

and the joint probability as

$$\begin{aligned} P(\boldsymbol{x}_{1:T}) &= P(x_1) \cdot P(x_2 \mid x_1) \\ &\quad \cdot \ \cdots \ \cdot \\ &\quad \cdot P(x_T \mid x_1, x_2, \ldots, x_{T-1}) \end{aligned}$$

Language Modeling with RNNs

Using RNNs for language modeling

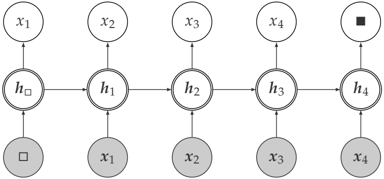


with two special tokens

$$\{\square, x_1, \ldots, x_T, \blacksquare\}$$

RNN Language Models

For a given sentence $\{x_1, \ldots, x_t\}$, the input at time t is word embedding $\boldsymbol{x}_t$



The probability distribution of next word X_t

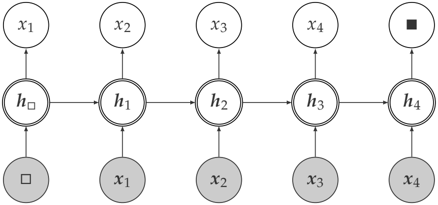
$$P(X_t = x \mid \boldsymbol{x}_{1:t-1}) = \frac{\exp(\boldsymbol{w}_{o,x}^\top \boldsymbol{h}_{t-1})}{\sum_{x' \in \mathcal{V}} \exp(\boldsymbol{w}_{o,x'}^\top \boldsymbol{h}_{t-1})} \quad (12)$$

where

- $\boldsymbol{w}_{o,x}$ is the output weight vector (parameter) associated with word x
- $\mathcal{V}$ is the word vocabulary

Special Cases

Similar to statistical language modeling, there are also two special cases that we need to consider



$$\{\square, x_1, \ldots, x_T, \blacksquare\}$$

The corresponding prediction functions are defined as

- At time $t = 1$

$$P(X_1 = x) \propto \exp(\boldsymbol{w}_{o,x}^\top \boldsymbol{h}_\square) \tag{13}$$

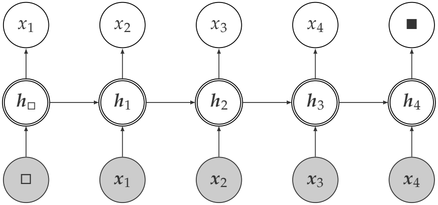
- At time $t = T$

$$P(X_T = \blacksquare \mid \boldsymbol{x}_{1:T-1}) \propto \exp(\boldsymbol{w}_{o,x}^\top \boldsymbol{h}_{T-1}) \tag{14}$$

Challenge of Training RNNs

Objective

The training objective for each timestep is to predict the next token in the text

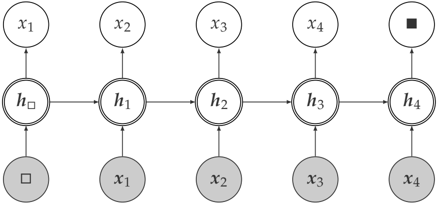


- Prediction at step t, $P(X_t = x \mid \boldsymbol{x}_{1:t-1}) = \frac{\exp(\boldsymbol{w}_{o,x}^{\top} \boldsymbol{h}_{t-1})}{\sum_{x' \in \mathcal{V}} \exp(\boldsymbol{w}_{o,x'}^{\top} \boldsymbol{h}_{t-1})}$
- Loss at step t, $L_t = -\log P(X_t = x \mid \boldsymbol{x}_{1:t-1})$

Gradients

Let $\boldsymbol{\theta}$ denote all model parameters

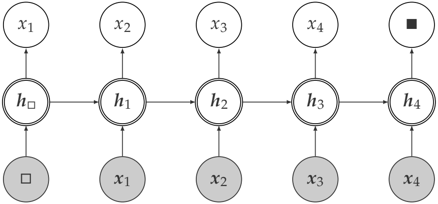
$$\frac{\partial \ell}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \boldsymbol{\theta}} \tag{15}$$



Backpropagation Through Time [Rumelhart et al., 1985, BPTT]

Model Parameters

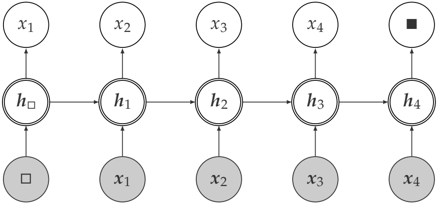
Before computing the gradient of each L_t with respect to model parameters, let us count how many parameters that we need consider



- Output parameter matrix $\boldsymbol{W}_o = (\boldsymbol{w}_{o,1}, \ldots, \boldsymbol{w}_{o,V})$

Model Parameters

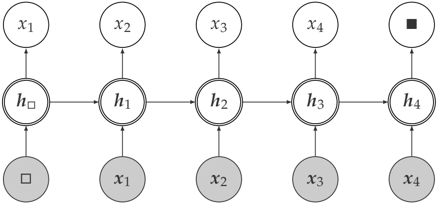
Before computing the gradient of each L_t with respect to model parameters, let us count how many parameters that we need consider



- Output parameter matrix $\boldsymbol{W}_o = (\boldsymbol{w}_{o,1}, \ldots, \boldsymbol{w}_{o,V})$
- Input word embedding matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_V)$

Model Parameters

Before computing the gradient of each L_t with respect to model parameters, let us count how many parameters that we need consider



- Output parameter matrix $\boldsymbol{W}_o = (\boldsymbol{w}_{o,1}, \ldots, \boldsymbol{w}_{o,V})$
- Input word embedding matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_V)$
- Neural network parameters $\boldsymbol{W}_h, \boldsymbol{W}_i, \boldsymbol{b}$

Backpropagation Through Time

Take time step t as an example, we can take a look the gradient computation of some specific parameters

- Output model parameter $\frac{\partial L_t}{\partial \boldsymbol{w}_{o,\cdot}}$

Backpropagation Through Time

Take time step t as an example, we can take a look the gradient computation of some specific parameters

- Output model parameter $\frac{\partial L_t}{\partial \boldsymbol{w}_{o,\cdot}}$
- Neural network parameters, for example $\boldsymbol{W}_h$

$$\frac{\partial L_t}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left\{ \frac{\partial L_t}{\partial \boldsymbol{h}_t} \cdot \left(\prod_{j=i}^{t-1} \frac{\partial \boldsymbol{h}_{j+1}}{\partial \boldsymbol{h}_j} \right) \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_h} \right\} \tag{16}$$

Similar patterns for the other two neural network parameters $\boldsymbol{W}_i$ and $\boldsymbol{b}$

Backpropagation Through Time

Take time step t as an example, we can take a look the gradient computation of some specific parameters

- Output model parameter $\frac{\partial L_t}{\partial \boldsymbol{w}_{o,\cdot}}$
- Neural network parameters, for example $\boldsymbol{W}_h$

$$\frac{\partial L_t}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left\{ \frac{\partial L_t}{\partial \boldsymbol{h}_t} \cdot \left(\prod_{j=i}^{t-1} \frac{\partial \boldsymbol{h}_{j+1}}{\partial \boldsymbol{h}_j} \right) \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_h} \right\} \quad (16)$$

  Similar patterns for the other two neural network parameters $\boldsymbol{W}_i$ and $\boldsymbol{b}$

- Word embedding $\frac{\partial L_t}{\partial \boldsymbol{x}_{t'}}$
  - E.g., word embedding $\boldsymbol{x}_{t'}$ is the input of $\boldsymbol{h}_t$ if $t' \leq t$, so ...

Challenges

For each timestep, we need to compute the gradient using the chain rule:

$$\frac{\partial L_t}{\partial \boldsymbol{W}_h} = \sum_{i=1}^{t} \left\{ \frac{\partial L_t}{\partial \boldsymbol{h}_t} \cdot \left(\prod_{j=i}^{t-1} \frac{\partial \boldsymbol{h}_{j+1}}{\partial \boldsymbol{h}_j} \right) \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{W}_h} \right\} \tag{17}$$

The chain rule of gradient will cause two potential problems in training RNNs

- vanishing gradients: $\frac{\partial L_t}{\partial \boldsymbol{\theta}} \longrightarrow 0$
- exploding gradients: $\frac{\partial L_t}{\partial \boldsymbol{\theta}} \geq M$

[Pascanu et al., 2013]

Exploding Gradients

Solution: **norm clipping** [Pascanu et al., 2013].

Consider the gradient $\boldsymbol{g} = \frac{\partial \ell}{\partial \boldsymbol{\theta}}$,

$$\hat{\boldsymbol{g}} \leftarrow \tau \cdot \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|} \tag{18}$$

when $\|\boldsymbol{g}\| > \tau$.

- Usually, $\tau = 3$ or 5 in practice.
- Smaller gradient will cause slower learning progress

Vanishing Gradients

Solution:

- initialize parameters carefully
- replace hidden state transition function $\sigma(\cdot)$ with other options

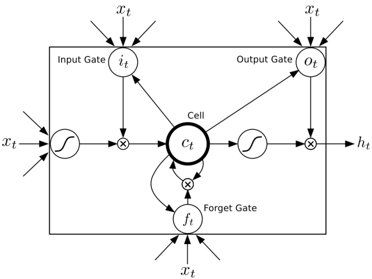
$$f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}) = \boldsymbol{\sigma}(\mathbf{W}_h \boldsymbol{h}_{t-1} + \mathbf{W}_i \boldsymbol{x}_t + \boldsymbol{b}) \tag{19}$$

  - LSTM [Hochreiter and Schmidhuber, 1997]
  - GRU [Cho et al., 2014]

Long Short-Term Memory

Rather than directly taking input and hidden state as simple transition function, LSTM relies on three cates to control *how much* information it should take from input and hidden state before combining them together

$$
\begin{aligned}
\boldsymbol{i}_t &= \sigma(\mathbf{W}_{xi}\boldsymbol{x}_t + \mathbf{W}_{hi}\boldsymbol{h}_{t-1} + \mathbf{W}_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma(\mathbf{W}_{xf}\boldsymbol{x}_t + \mathbf{W}_{hf}\boldsymbol{h}_{t-1} + \mathbf{W}_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tanh(\mathbf{W}_{xc}\boldsymbol{x}_t + \mathbf{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\mathbf{W}_{xo}\boldsymbol{x}_t + \mathbf{W}_{ho}\boldsymbol{h}_{t-1} + \mathbf{W}_{co}\boldsymbol{c}_t + \boldsymbol{b}_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned}
$$



where $\circ$ is the element-wise multiplication, $\{W_\cdot\}$ and $\{b_\cdot\}$ are parameters. [Graves, 2013]

Reference

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014).
On the properties of neural machine translation: Encoder-decoder approaches.
arXiv preprint arXiv:1409.1259.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016).
Deep Learning, volume 1.
MIT press Cambridge.

Graves, A. (2013).
Generating sequences with recurrent neural networks.
arXiv preprint arXiv:1308.0850.

Hochreiter, S. and Schmidhuber, J. (1997).
Long short-term memory.
Neural computation, 9(8):1735–1780.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013).
On the difficulty of training recurrent neural networks.
In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985).
Learning internal representations by error propagation.
Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.