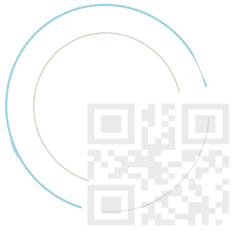
文本分类实践

玖强

Agenda

- Word Embeddings
- Classification
  - RNN-based model
  - CNN-based model
- Experiments



Word Embeddings (or 词向量)

- Traditional NLP: Words are treated as indices (or "one-hot" vectors in $\mathbf{R}^V$)
  - Every word is orthogonal to one another.
  - $\mathbf{w}_{mother} \cdot \mathbf{w}_{father} = 0$
- Can we embed words in $\mathbf{R}^D$ with $D \leq V$ such that semantically close words are likewise 'close' in $\mathbf{R}^D$? (i.e. $\mathbf{w}_{mother} \cdot \mathbf{w}_{father} > 0$)
  - Yes!
  - Don't (necessarily) need deep learning for this: Latent Semantic Analysis, Latent Dirichlet Allocation, or simple context counts all give dense representations.

Neural Language Models (NLM)

- Another way to obtain word embeddings.
- Words are projected from $\mathbf{R}^V$ to $\mathbf{R}^D$ via a hidden layer.
- D is a hyperparameter to be tuned.
- Various architectures exist. Simple ones are popular these days (right).
- Very fast—can train on billions of tokens in one day with a single machine.

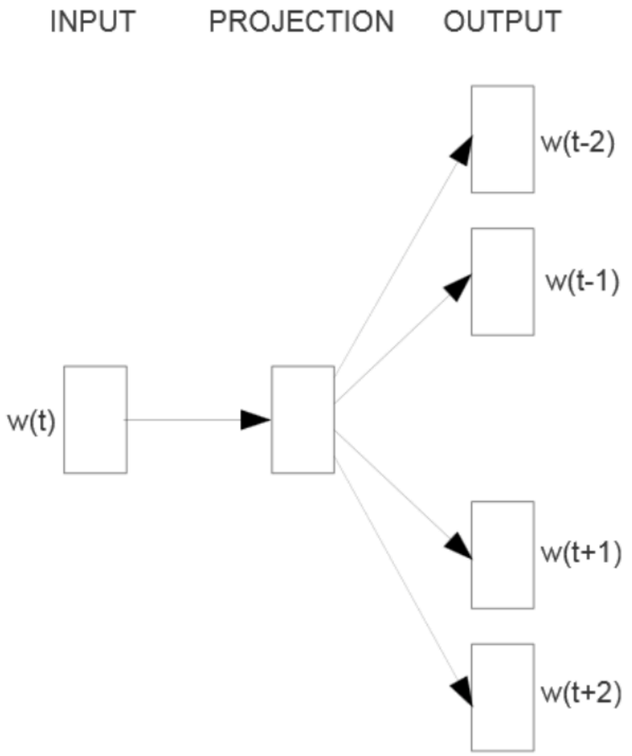


Figure 1: Skipgram architecture of Mikolov et al. (2013)

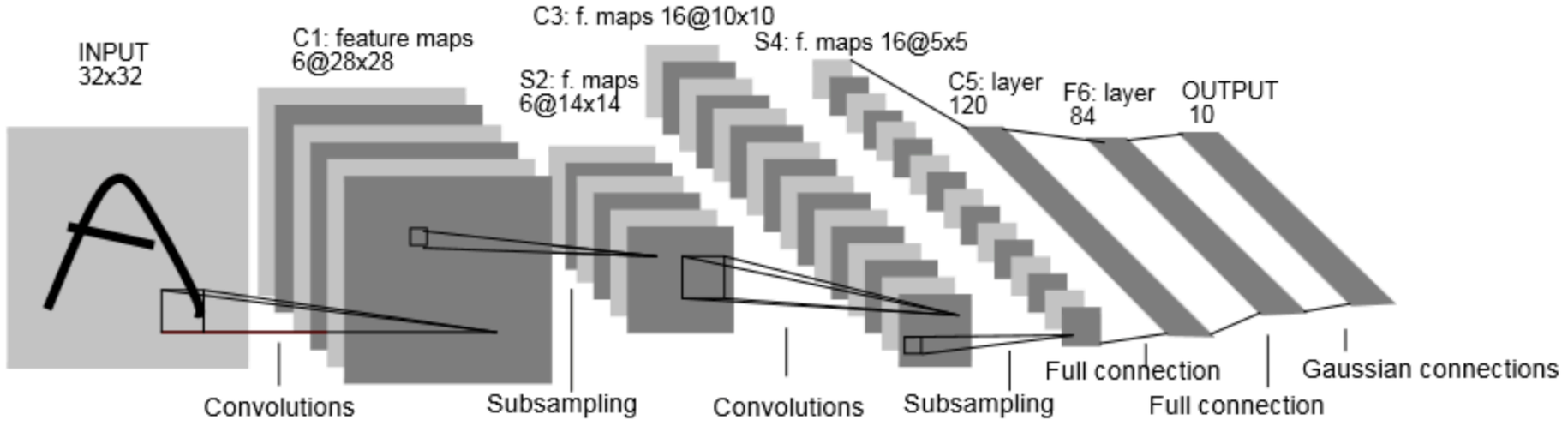
Using word embeddings as features in classification

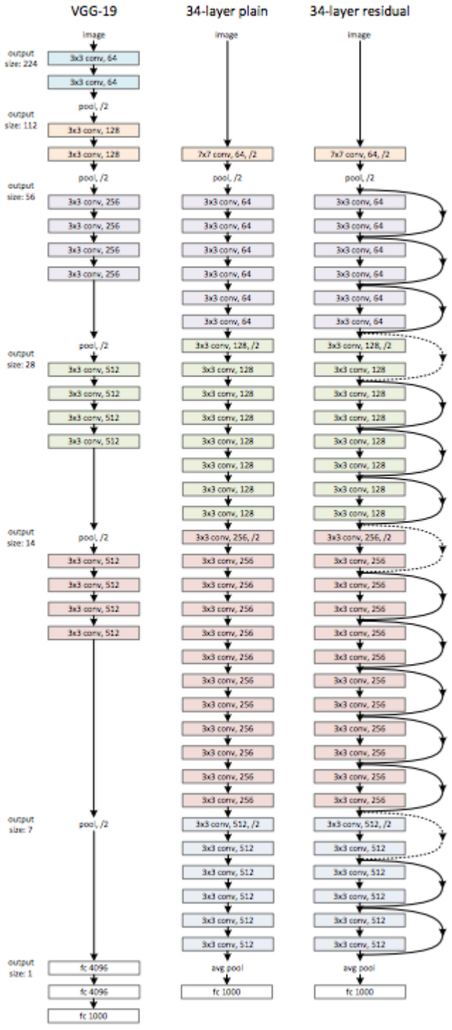
- The embeddings can be used as features in a classifier.
- For multi-word composition (e.g. sentences and phrases), one could (for example) take the average.
- 均值看起来很粗糙，有更好的办法吗？

CNN-Based model

Convolutional Neural Networks (CNN)

- 初衷是为了解决图像的问题(Lecun et al, 1989).
- Pretty much all modern vision systems use CNNs.

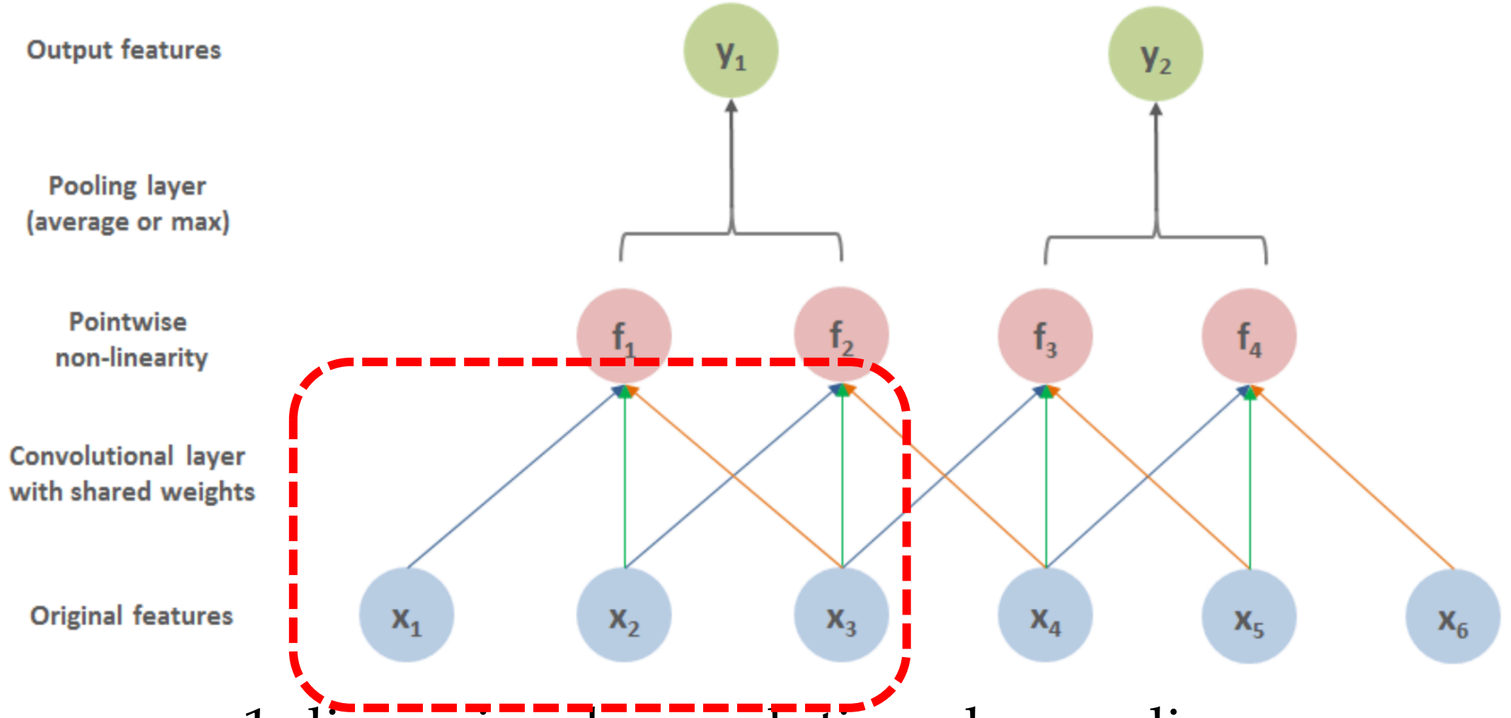




LeCun et al., “Gradient-based learning applied to document recognition”, IEEE 1998

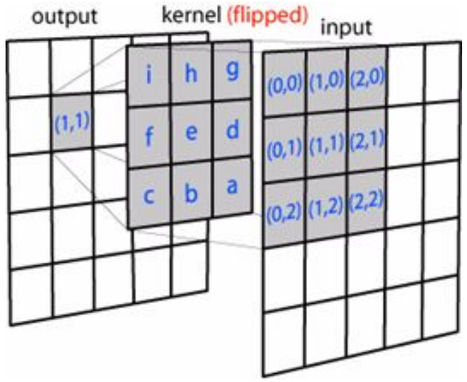
基本思想

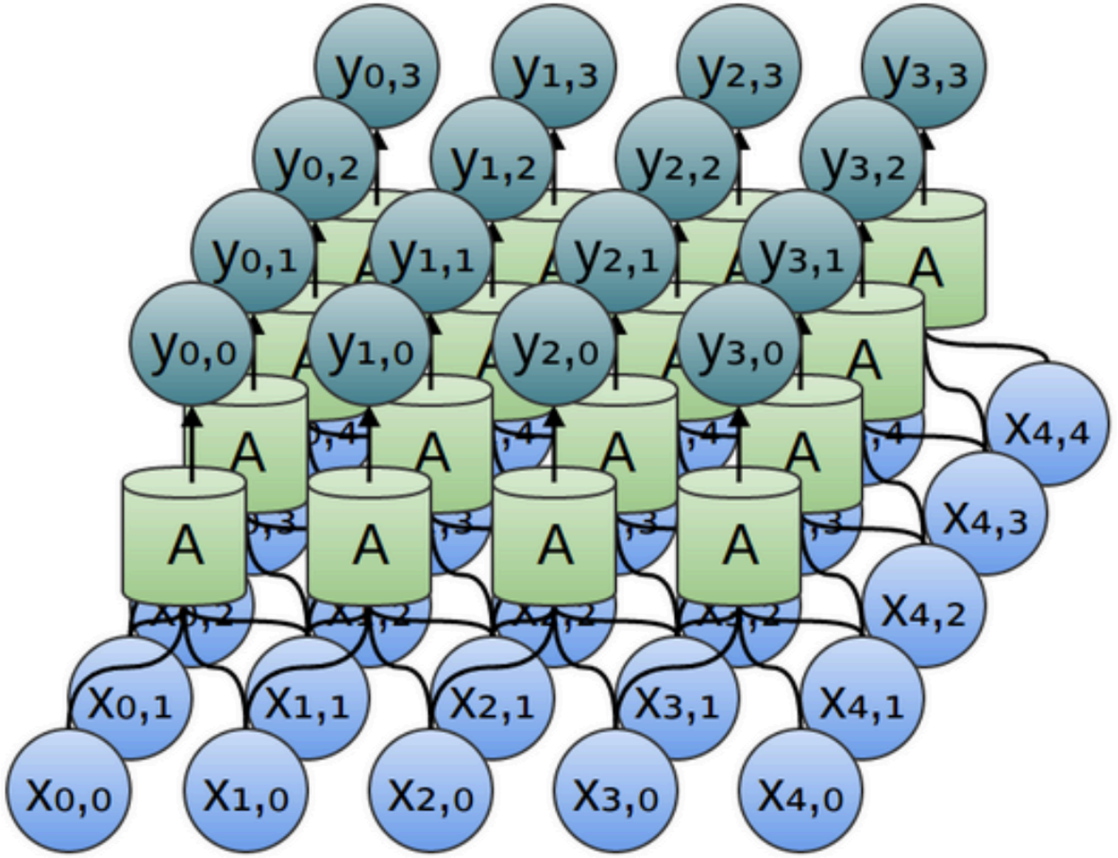
- Key idea 1: Weight sharing via convolutional layers
- Key idea 2: Pooling layers
- Key idea 3: Multiple feature maps



1-dimensional convolution plus pooling

2维卷积示例





2-dimensional convolution. From http://colah.github.io/

CNN 细节

- Shared weights means less parameters (than would be the case if fully connected).
- Pooling layers allow for local invariance.
- Multiple feature maps allow different kernels to act as specialized feature extractors.
- Training done through backpropagation.
- Errors are backpropagated through pooling modules.

简单的卷积操作示例

- Simple example of convolution operation

Input, e.g. an image

1	3	5	2	4
6	0	2	1	3
6	3	1	3	6
7	3	2	1	3
5	3	0	0	2

Filter (Kernel)

0.2	0.7
-0.5	0.7

简单的卷积操作示例

- Simple example of convolution operation

Input, e.g. an image

1	3	5	2	4
6	0	2	1	3
6	3	1	3	6
7	3	2	1	3
5	3	0	0	2

Filter (Kernel)

0.2	0.7
-0.5	0.7

$$c_1 = \mathrm{f}\,(0.2 * 1 + 0.7 * 3 - 0.5 * 6 + 0.7 * 0) = \mathrm{f}(-0.7)$$

Feature map

f(-0.7)

简单的卷积操作示例

- Simple example of convolution operation

Input, e.g. an image

1	3	5	2	4
6	0	2	1	3
6	3	1	3	6
7	3	2	1	3
5	3	0	0	2

Filter (Kernel)

0.2	0.7
-0.5	0.7

$c_1 = \mathrm{f}\,(0.2 * 1 + 0.7 * 3 - 0.5 * 6 + 0.7 * 0) = \mathrm{f}(-0.7)$

$c_2 = \mathrm{f}(0.2 * 3 + 0.7 * 5 - 0.5 * 0 + 0.7 * 2) = \mathrm{f}(5.5)$

Feature map

f(-0.7)	f(5.5)

Note: Bias terms omitted!

f represents some non-linear activation function

简单的卷积操作示例

- Simple example of convolution operation

Input, e.g. an image

1	3	5	2	4
6	0	2	1	3
6	3	1	3	6
7	3	2	1	3
5	3	0	0	2

Filter (Kernel)

0.2	0.7
-0.5	0.7

$c_1 = \mathrm{f}(0.2 * 1 + 0.7 * 3 - 0.5 * 6 + 0.7 * 0) = \mathrm{f}(-0.7)$

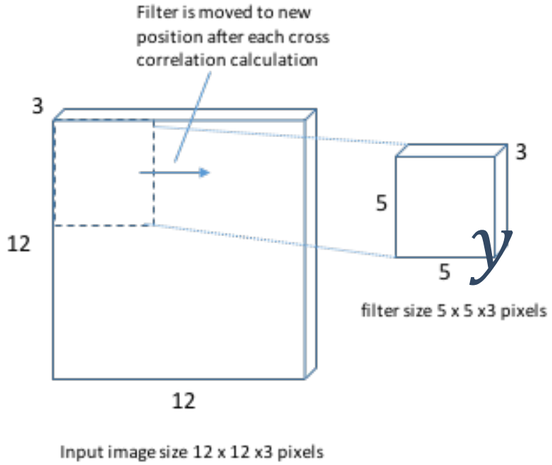
$c_2 = \mathrm{f}(0.2 * 3 + 0.7 * 5 - 0.5 * 0 + 0.7 * 2) = \mathrm{f}(5.5)$

$c_3 = \cdots$

Feature map

f(-0.7)	f(5.5)	…

简单的卷积操作示例



$$y = \sum_{k=1}^{3} \sum_{i=1}^{5} \sum_{j=1}^{5} x_{kij} \theta_{kij}$$

- After convolution, some other operations are performed such as applying the activation function (nonlinearity) and pooling
- During training, the values that are used in the filters are updated and gradually learned
- The parameter sharing concept brings invariance

中文情感分析 (Sentiment Analysis)

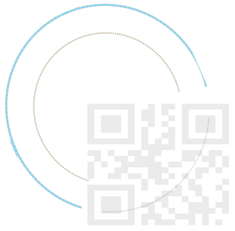
- Sentiment analysis is a collection of methods with the main intent to observe the opinion or attitude, for example, of a sentence expressed in natural language.

Sentiment analysis using CNNs

- Analysis based on Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. EMNLP. http://aclweb.org/anthology/D/D14/D14-1181.pdf

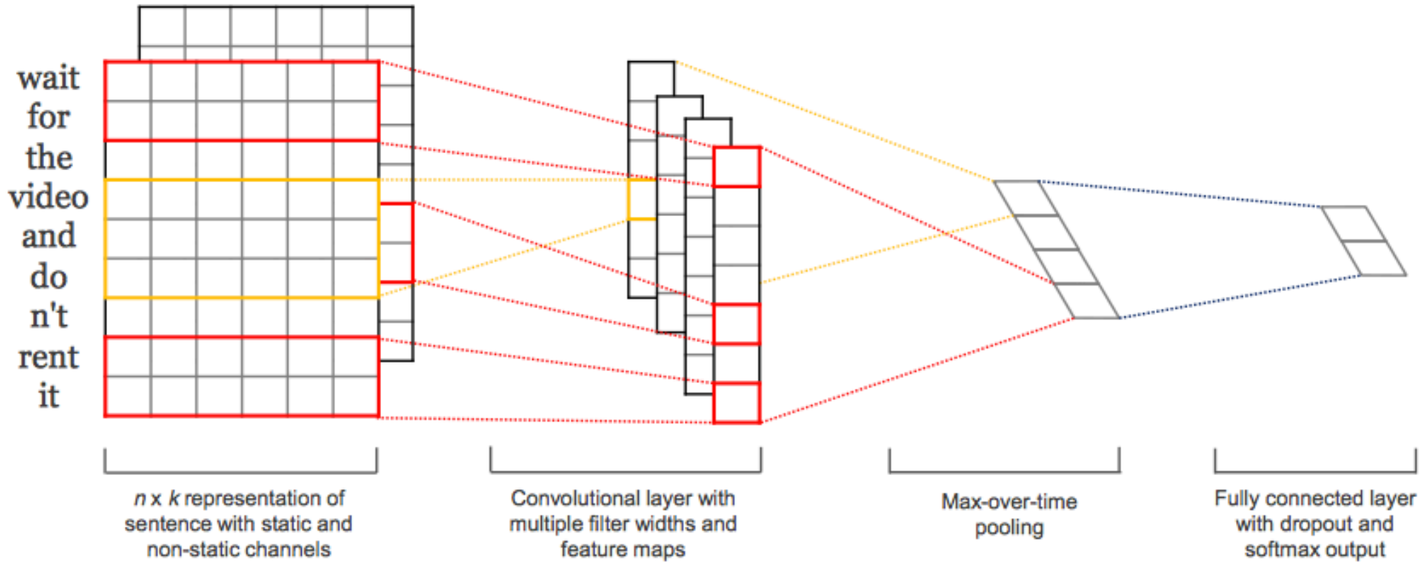
- A simple CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model.

- Good results are obtained by using pre-trained word vector. Results are still improved by further training the word vectors for specific tasks.



Sentiment analysis using CNNs

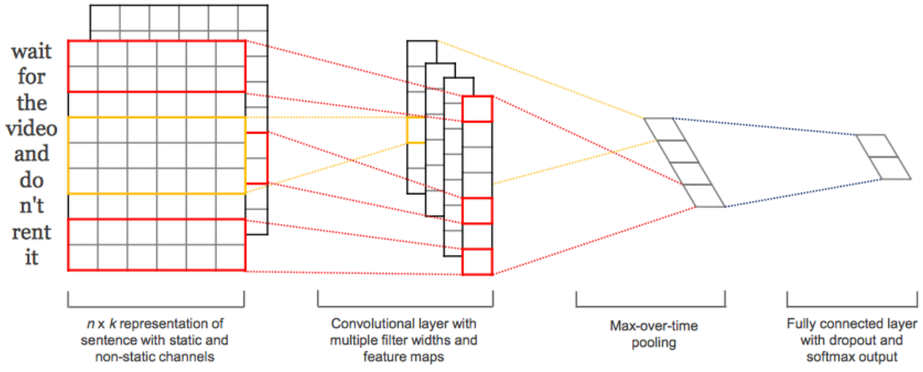
- Simple CNN model for sentiment analysis



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification.

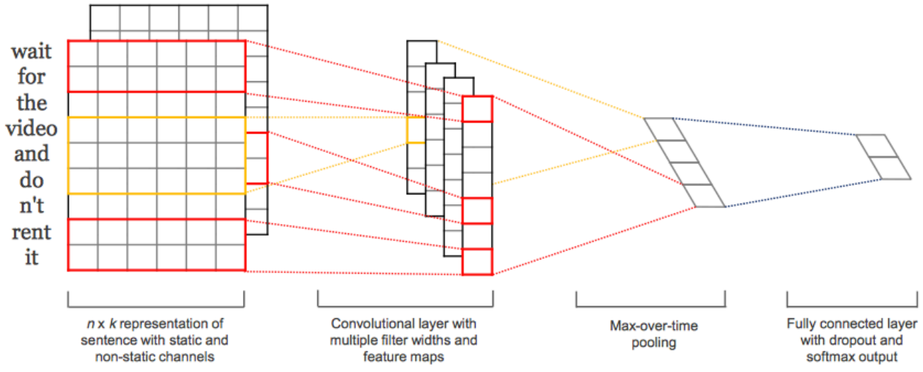
Sentiment analysis using CNNs

- Multiple filters sizes (3,4,5) to produce several feature maps (100 of each size) -> magnitude of 0,3 – 0,4 M parameters
- Max-over-time pooling used to select the most important feature
- Two input channels used, other with static word vectors and other with trainable vectors
- Fully connected softmax layer on top to produce probabilities for each class
- Dropout used in the fully connected layer for regularization, L2 norm gradient clipping for other weights. Early stopping used.
- Stochastic gradient descent update using Adadelta update rule
- Pre-trained word2vec used, trained with 100B words from Google news, 300 dimensions



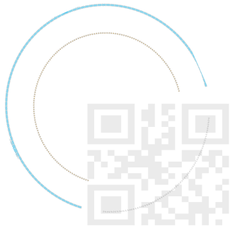
Sentiment analysis using CNNs

- CNN-rand; all words are initialized randomly and trained
- CNN-static; initialized with word2vec used (unknown initialized randomly) and kept static
- CNN-non-static; intialized with word2vec and trained further
- CNN-multichannel; Initialized with word2vec, one channel stays static and other channel is further trained



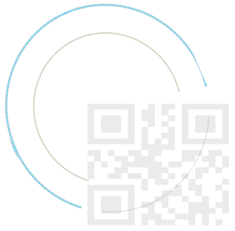
Sentiment analysis using CNNs

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–



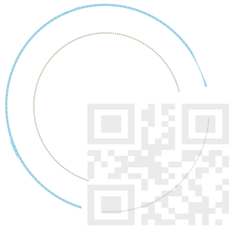
基于CNN的模型定义

```
class CNN_Text(nn.Module):

    def __init__(self, args):
        super(CNN_Text, self).__init__()
        self.args = args

        V = args.embed_num
        D = args.embed_dim
        C = args.class_num
        Ci = 1
        Co = args.kernel_num
        Ks = args.kernel_sizes

        self.embed = nn.Embedding(V, D)
        # self.convs1 = [nn.Conv2d(Ci, Co, (K, D)) for K in Ks]
        self.convs1 = nn.ModuleList([nn.Conv2d(Ci, Co, (K, D)) for K in Ks])
        '''
        self.conv13 = nn.Conv2d(Ci, Co, (3, D))
        self.conv14 = nn.Conv2d(Ci, Co, (4, D))
        self.conv15 = nn.Conv2d(Ci, Co, (5, D))
        '''
        self.dropout = nn.Dropout(args.dropout)
        self.fc1 = nn.Linear(len(Ks)*Co, C)
```



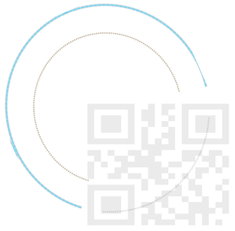
模型的前向传播过程

```
def forward(self, x):
    x = self.embed(x)  # (N, W, D)

    if self.args.static:
        x = Variable(x)

    x = x.unsqueeze(1)  # (N, Ci, W, D)

    x = [F.relu(conv(x)).squeeze(3) for conv in self.convs1]  # [(N, Co, W), ...]*len(Ks)

    x = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in x]  # [(N, Co), ...]*len(Ks)

    x = torch.cat(x, 1)

    '''
    x1 = self.conv_and_pool(x,self.conv13) #(N,Co)
    x2 = self.conv_and_pool(x,self.conv14) #(N,Co)
    x3 = self.conv_and_pool(x,self.conv15) #(N,Co)
    x = torch.cat((x1, x2, x3), 1) # (N,len(Ks)*Co)
    '''
    x = self.dropout(x)  # (N, len(Ks)*Co)
    logit = self.fc1(x)  # (N, C)
    return logit
```



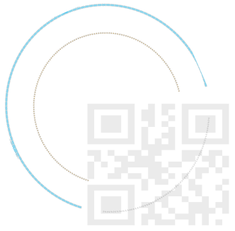
模型的优化策略与做法

```
optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)

feature, target = batch.text, batch.label
feature.data.t_(), target.data.sub_(1)  # batch first, index align
if args.cuda:
    feature, target = feature.cuda(), target.cuda()

optimizer.zero_grad()
logit = model(feature)

#print('logit vector', logit.size())
#print('target vector', target.size())
loss = F.cross_entropy(logit, target)
loss.backward()
optimizer.step()
```



模型的评估

```
def eval(data_iter, model, args):
    model.eval()
    corrects, avg_loss = 0, 0
    for batch in data_iter:
        feature, target = batch.text, batch.label
        feature.data.t_(), target.data.sub_(1)  # batch first, index align
        if args.cuda:
            feature, target = feature.cuda(), target.cuda()

        logit = model(feature)
        loss = F.cross_entropy(logit, target, size_average=False)

        avg_loss += loss.data[0]
        corrects += (torch.max(logit, 1)
                     [1].view(target.size()).data == target.data).sum()

    size = len(data_iter.dataset)
    avg_loss /= size
    accuracy = 100.0 * corrects/size
    print('\nEvaluation - loss: {:.6f}  acc: {:.4f}%({}/{}) \n'.format(avg_loss,
                                                                       accuracy,
                                                                       corrects,
                                                                       size))
    return accuracy
```

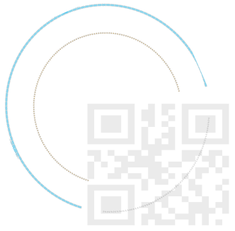


当然，还有模型的测试

```python
def predict(text, model, text_field, label_feild, cuda_flag):
    assert isinstance(text, str)
    model.eval()
    # text = text_field.tokenize(text)
    text = text_field.preprocess(text)
    text = [[text_field.vocab.stoi[x] for x in text]]
    x = text_field.tensor_type(text)
    x = autograd.Variable(x, volatile=True)
    if cuda_flag:
        x = x.cuda()
    print(x)
    output = model(x)
    _, predicted = torch.max(output, 1)
    #return label_feild.vocab.itos[predicted.data[0][0]+1]
    return label_feild.vocab.itos[predicted.data[0]+1]
```

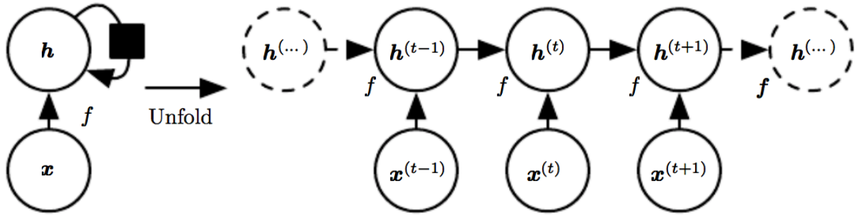
```
jxgu@jxgu:~/github/NLP_Practice.PyTorch/text_classification/cnn_based$ ./main.py -predi

Loading data...

Parameters:
        BATCH_SIZE=64
        CLASS_NUM=2
        CUDA=True
        DEVICE=-1
        DROPOUT=0.5
        EARLY_STOP=1000
        EMBED_DIM=128
        EMBED_NUM=21114
        EPOCHS=256
        KERNEL_NUM=100
        KERNEL_SIZES=[3, 4, 5]
        LOG_INTERVAL=1
        LR=0.001
        MAX_NORM=3.0
        PREDICT=Hello my dear , I love you so much .
        SAVE_BEST=True
        SAVE_DIR=snapshot/2018-04-07_21-28-00
        SAVE_INTERVAL=500
        SHUFFLE=False
        SNAPSHOT=./snapshot/2018-04-06_17-26-28/best_steps_19100.pt
        STATIC=False
        TEST=False
        TEST_INTERVAL=100

Loading model from ./snapshot/2018-04-06_17-26-28/best_steps_19100.pt...
Variable containing:
    0   186  4702     4    44    90    25    40    53     2
[torch.cuda.LongTensor of size 1x10 (GPU 0)]


[Text]  Hello my dear , I love you so much .
[Label] positive

jxgu@jxgu:~/github/NLP_Practice.PyTorch/text_classification/cnn_based$
```

Sentiment analysis using RNNs

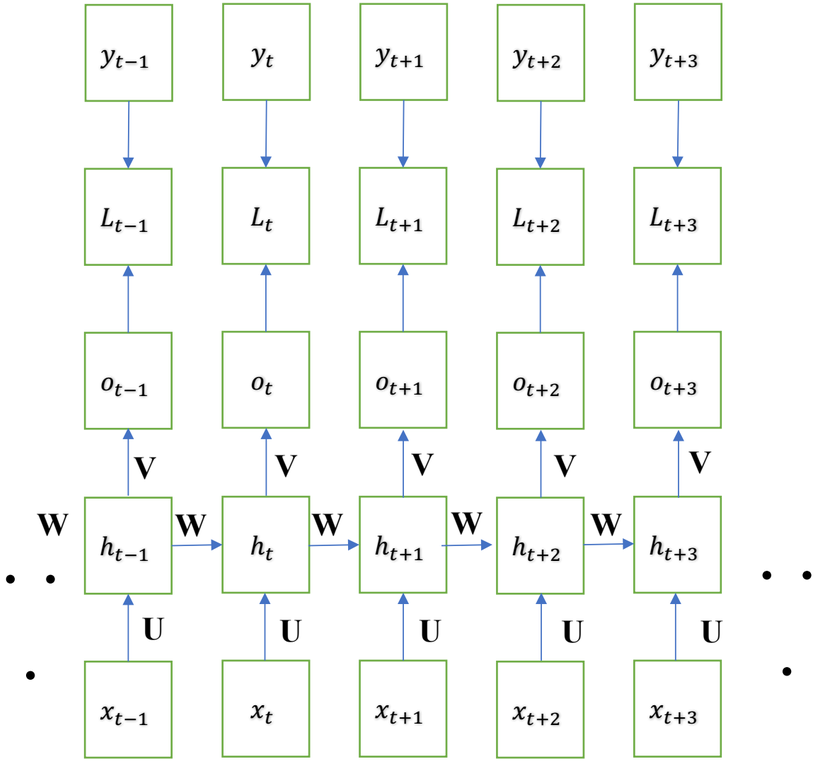


Recurrent Neural Network (RNN)

- Shallow RNN:

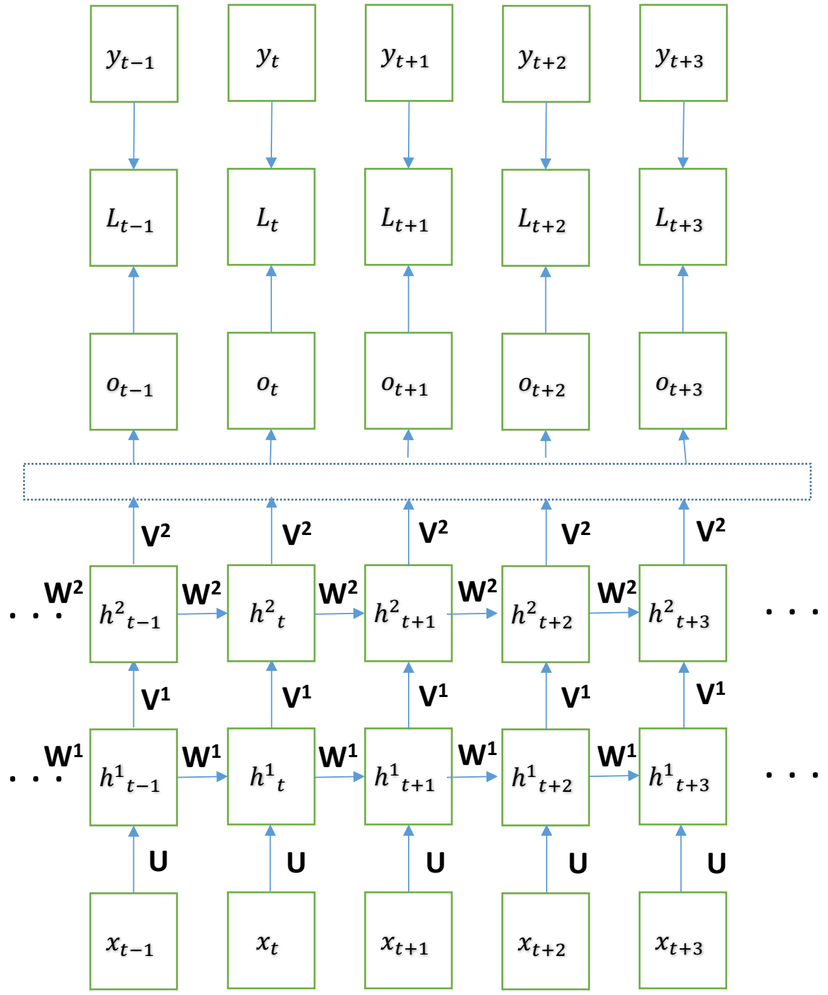


Source: Goodfellow, I., Bengio, Y., Courville, A., Deep Learning,



Recurrent Neural Network (RNN)

- Deep RNN example:

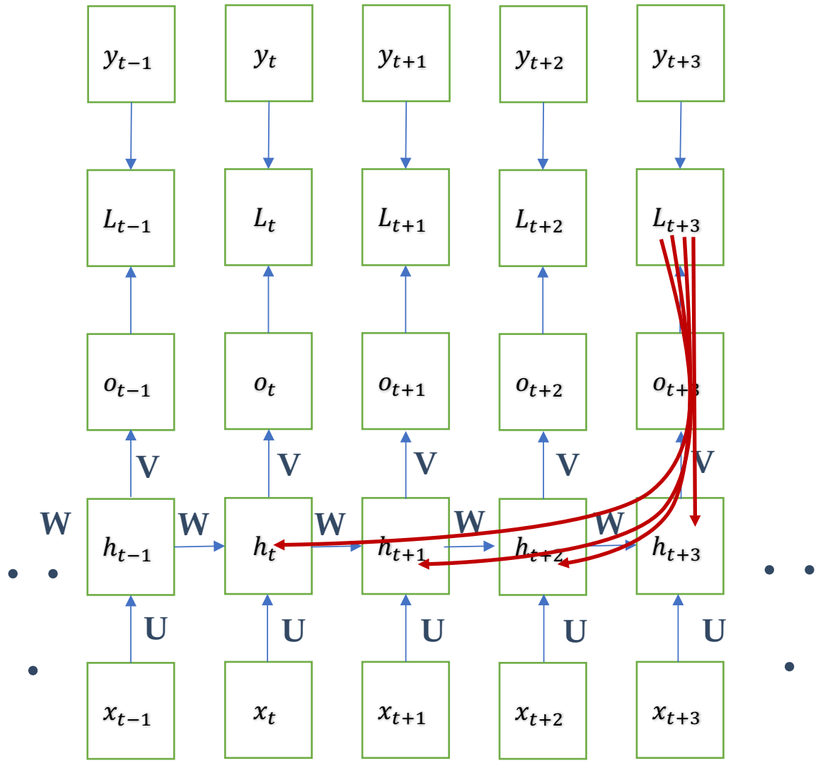


Vanishing gradient problem and LSTM

- Problem:
  gradients propagate over many stages,
  and involves several multiplications of the
  weight matrix.

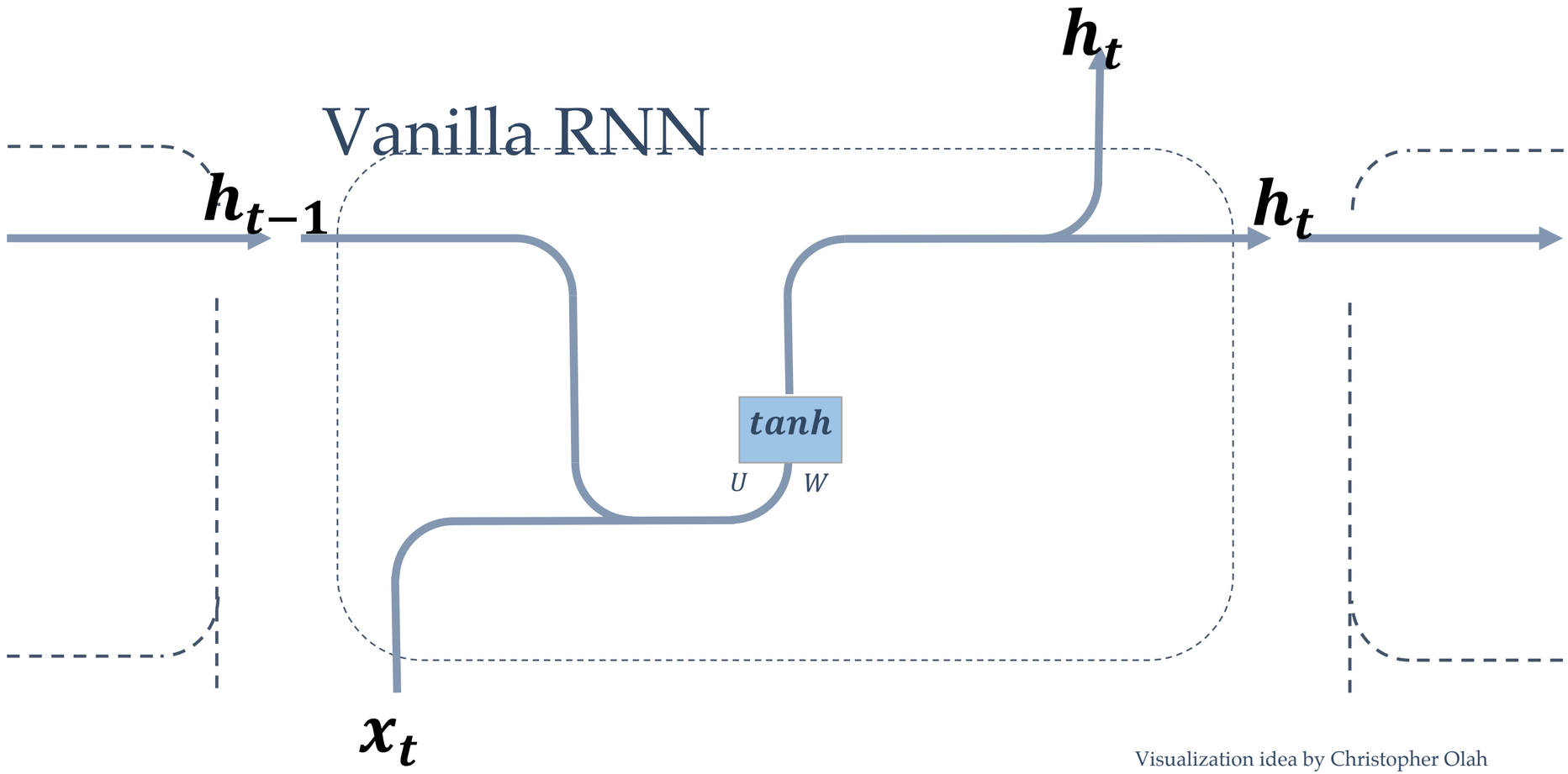
  ->

  vanishing or exploding gradients



Standard RNN cell

$$h_t = tanh\left(U^{(c)}\, x_t + W^{(c)}\, h_{t-1} + b_c\right)$$



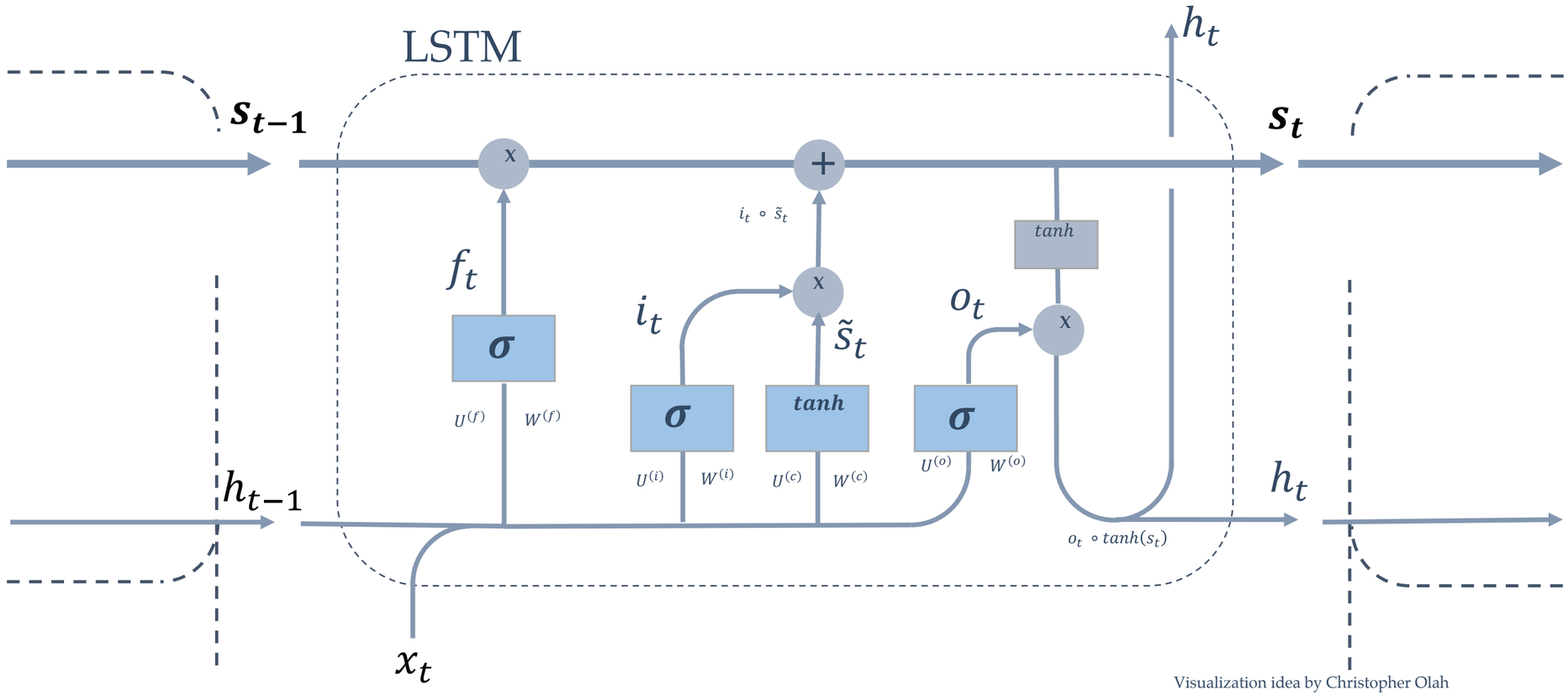
Visualization idea by Christopher Olah

Standard RNN cell

$$s_t = f_t \circ s_{t-1} + i_t \circ \tilde{s}_t \qquad \tilde{s}_t = tanh\left(U^{(c)} x_t + W^{(c)} h_{t-1} + b_c\right)$$

$$f_t = \sigma\left(U^{(f)} x_t + W^{(f)} h_{t-1} + b_f\right) \qquad h_t = o_t \circ tanh(s_t)$$

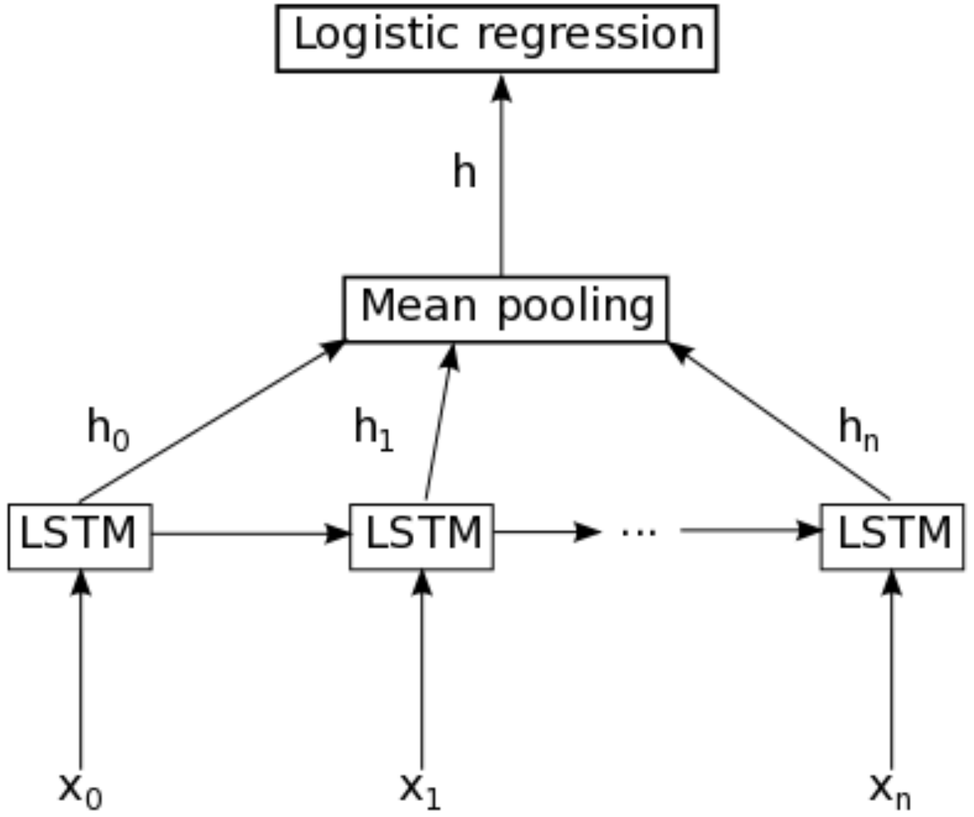
$$i_t = \sigma\left(U^{(i)} x_t + W^{(i)} h_{t-1} + b_i\right) \qquad o_t = \sigma\left(U^{(o)} x_t + W^{(o)} h_{t-1} + b_o\right)$$



Visualization idea by Christopher Olah

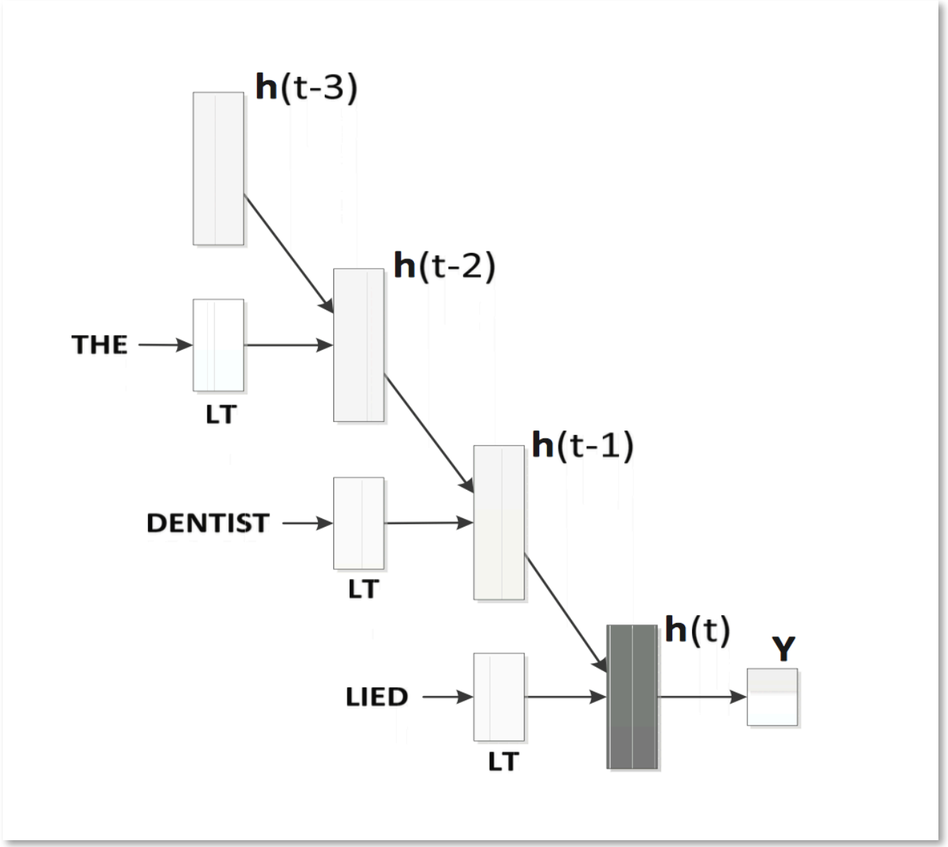
Sentiment analysis / text classification

- A quick example, to see the idea.
- Given text collections and their labels. Predict labels for unseen texts.



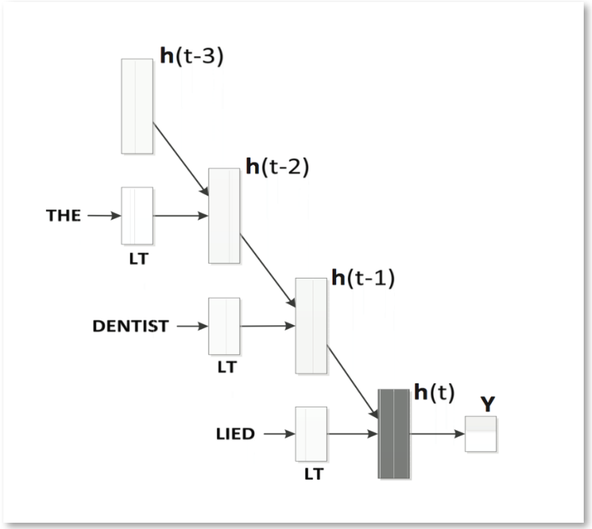
Sentiment analysis using RNNs

- Analysis based on [Wan2015]: Wang, X., Liu, Y., Sun, C., Wang, B., & Wang, X. (2015). Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory.
- Twitter sentiment prediction, using simple RNN or LSTM recurrent network.



Sentiment analysis using RNNs

- RNN-FLT (Recurrent Neural Network with Fixed Lookup-Table): a simple implementation of the recurrent sentiment classifier



Forward pass:

$$b_h^t = f\left(a_h^t\right)$$

$$a_h^t = \sum_i^E w_{ih} e_i^t + \sum_{h'}^H w_{h'h} b_{h'}^{t-1}$$

$$y = f\left(\sum_i^H b_i^T v_i\right)$$

Backpropagation:

$$\delta_i^t = \frac{\partial O}{\partial a_i^t}$$

$$\delta_i^T = f'\left(a_i^T\right)\frac{\partial O}{\partial y} v_i$$

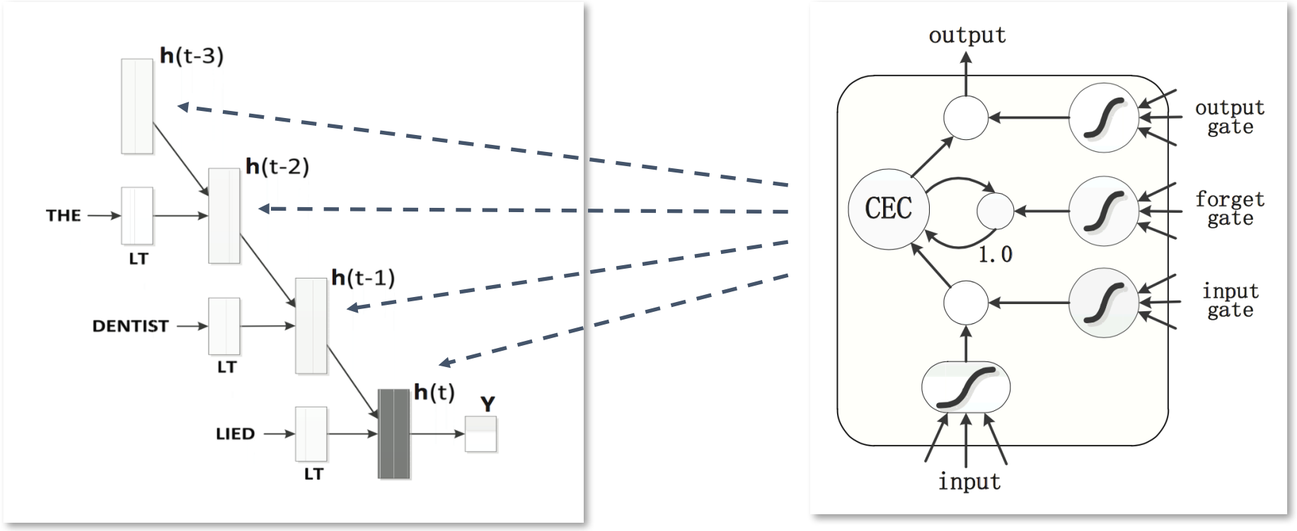
$$\delta_h^t = f'\left(a_h^t\right)\sum_{h'}^H \delta_{h'}^{t+1} w_{hh'}$$

f represents sigmoid function, w are the weights, e are the word embeddings, v includes hidden-output weights, t is the time step, T is the last time step.

The loss O is calculated by using cross-entropy loss, and training is conducted by stochastic gradient descent (SGD)

Sentiment analysis using RNNs

- RNN-TLT (Recurrent Neural Network with Trainable Lookup-Table) and LSTM-TLT: implementations that include further training of the pre-trained word vectors. In LSTM-TLT the classifier uses LSTM blocks instead of regular RNN blocks.



- Each regular RNN block is replaced by an LSTM block -> much more complicated functionality -> helps to combat the vanishing gradient problem

Sentiment analysis using RNNs

- Experiments are run by using Stanford Twitter Sentiment corpus (STS); 800 000 positive and 800 000 negative tweets. Manually labeled test set includes 177 negative and 182 positive tweets. Training set sentiment labels are retrieved from emoticons.
- 25 dimensional word vectors that were trained with 1.56M tweets from training set using word2vec.
- Hidden layer size 60.

Classifier	Accuracy(%)
SVM	81.6
MNB	82.7
MAXENT	83.0
MAX-TDNN	78.8
NBoW	80.9
DCNN	**87.4**
RAE	77.6
RNN-FLT	80.2
RNN-TLT	86.4
LSTM-TLT	**87.2**

Non-neural classifiers
(Naive Bayes, Maximum Entropy, Support Vector Machine)

Neural Bag-of-Words, summation of word vectors as input

Dynamic Convolutional Neural Network

Recursive Autoencoder

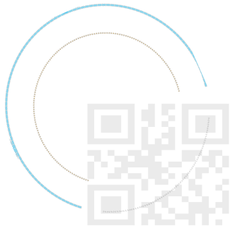
Models presented in this paper

Sentiment analysis using RNNs

- Additional experiment are run using human-labeled dataset SemEval 2013. The dataset has training set of 4099, development set of 735 and test set of 1742 tweets.

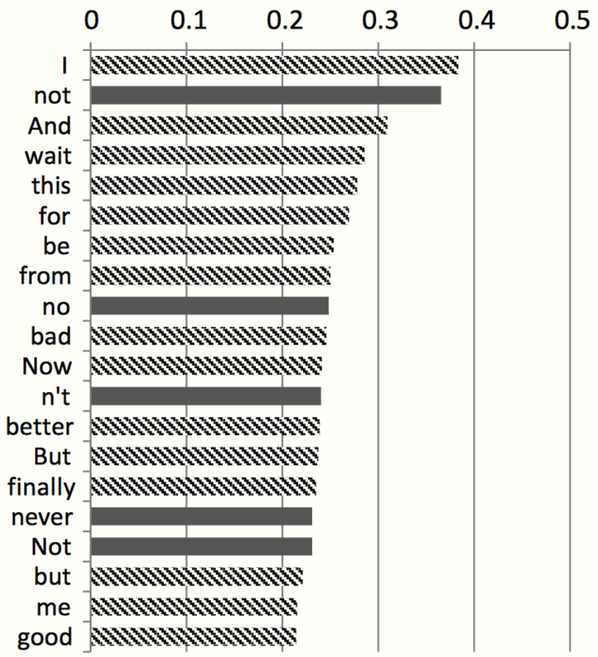
- Fixed word vectors, pre-trained with word2vec using STS dataset, this time 300-dimensions

Method	Accuracy(%)
SVM	74.5
RAE	75.4
RNN-FLT	83.0
LSTM-FLT	**84.0**



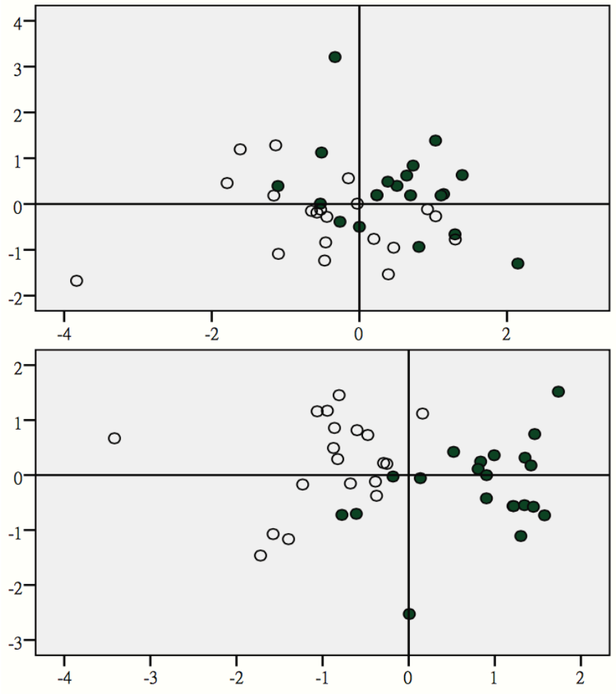
Sentiment analysis using RNNs

- Which words change most when training the pre-trained word2vec vectors?



Sentiment analysis using RNNs

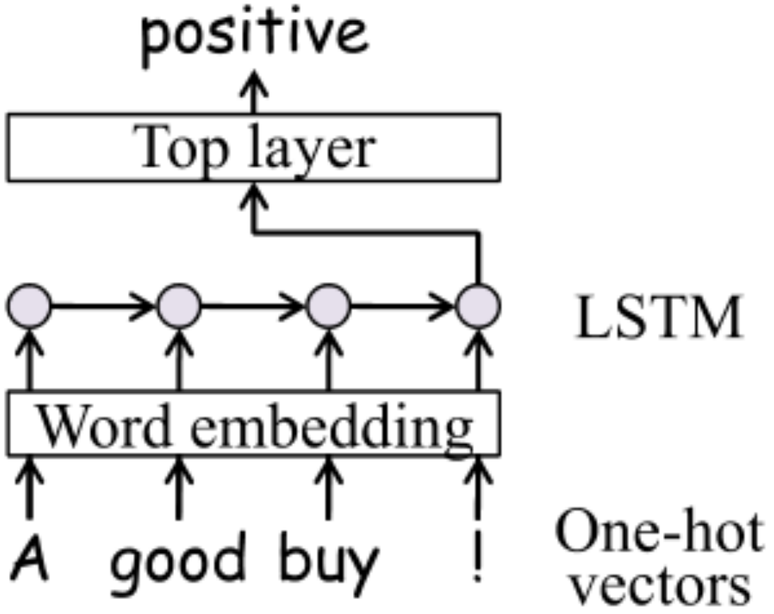
- How the sentiment words are moved during training in 2-d space?
- 20 most negative and 20 most positive words were tracked during training



Before tuning

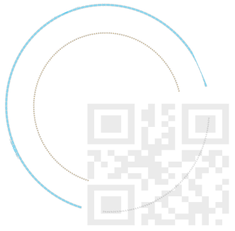
After tuning

Model definition



模型定义

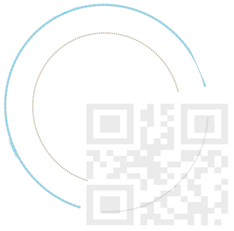
```
class LSTMClassifier(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, label_size, batch_size, use_gpu):
        super(LSTMClassifier, self).__init__()
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.use_gpu = use_gpu

        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)
        self.hidden2label = nn.Linear(hidden_dim, label_size)
        self.hidden = self.init_hidden()
```



前向传播+反向传播

```
def forward(self, sentence):
    embeds = self.word_embeddings(sentence)
    x = embeds.view(len(sentence), self.batch_size, -1)
    lstm_out, self.hidden = self.lstm(x, self.hidden)
    y  = self.hidden2label(lstm_out[-1])
    return y
```

```
train_inputs, train_labels = traindata
train_labels = torch.squeeze(train_labels)

if use_gpu:
    train_inputs, train_labels = Variable(train_inputs.cuda()), train_labels.cuda()
else: train_inputs = Variable(train_inputs)

model.zero_grad()
model.batch_size = len(train_labels)
model.hidden = model.init_hidden()
output = model(train_inputs.t())

loss = loss_function(output, Variable(train_labels))
loss.backward()
optimizer.step()
```



实验数据

1. Details of file fold:

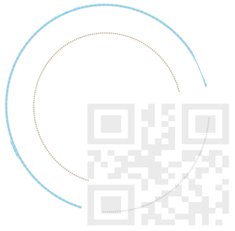
- data/
- data/train_txt/*.txt
- data/train_txt.txt
- data/train_label.txt
- data/test_txt/*.txt
- data/test_txt.txt
- data/test_label.txt

2. File description:

file	description
data/train_txt/	training text fold
data/test_txt/	testing text fold
data/train_txt.txt	file name list for training text
data/test_txt.txt	file name list for testing text
data/train_label.txt	label list for training text
data/test_label.txt	label list for testing text

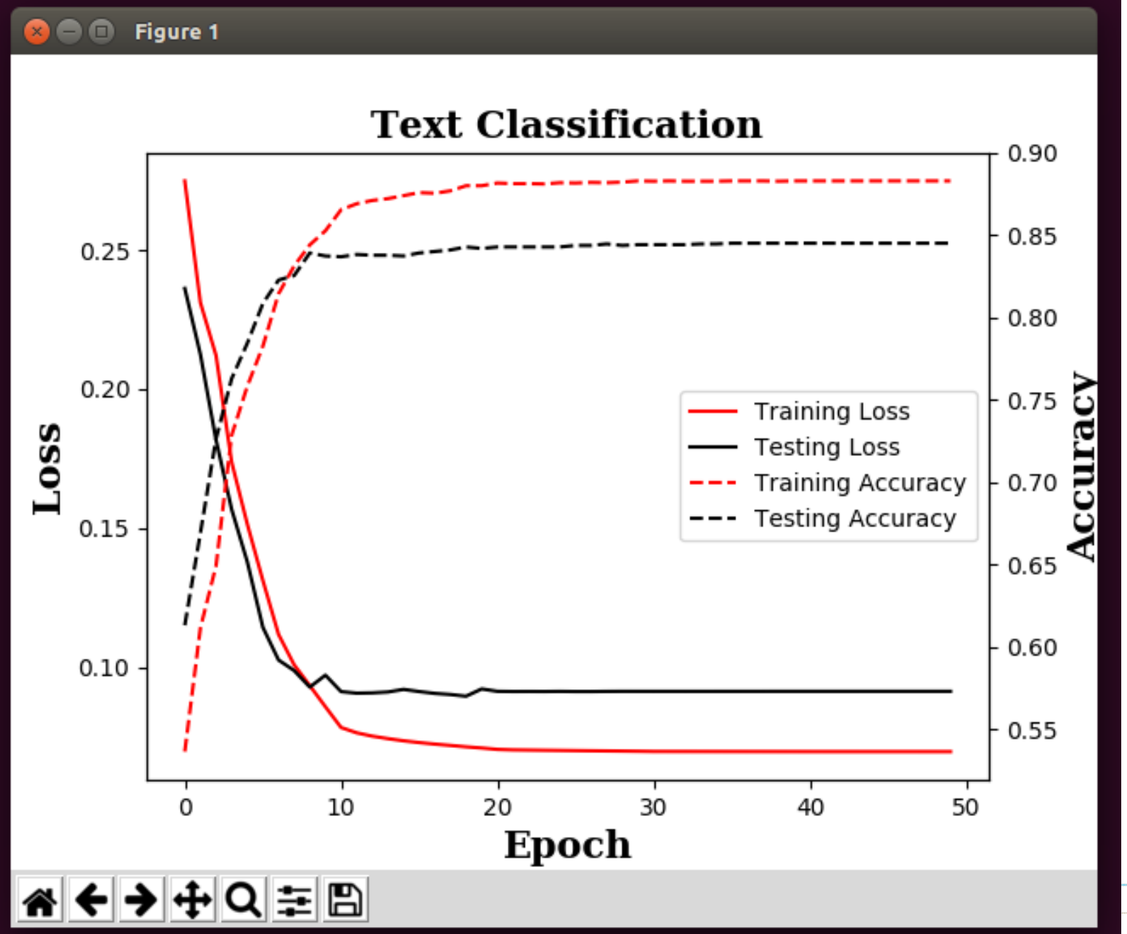
预处理

```
class Corpus(object):
    def __init__(self, DATA_DIR, filenames):
        self.dictionary = Dictionary()
        self.data = self.tokenize(DATA_DIR, filenames)

    def tokenize(self, DATA_DIR, filenames):
        for filename in filenames:
            path = os.path.join(DATA_DIR, filename)
            with open(path, 'r') as f:
                tokens = 0
                for line in f:
                    words = line.split() + ['<eos>']
                    tokens += len(words)
                    for word in words:
                        self.dictionary.add_word(word)

            # Tokenize file content
            with open(path, 'r') as f:
                ids = torch.LongTensor(tokens)
                token = 0
                for line in f:
                    words = line.split() + ['<eos>']
                    for word in words:
                        ids[token] = self.dictionary.word2idx[word]
                        token += 1

        return ids
```

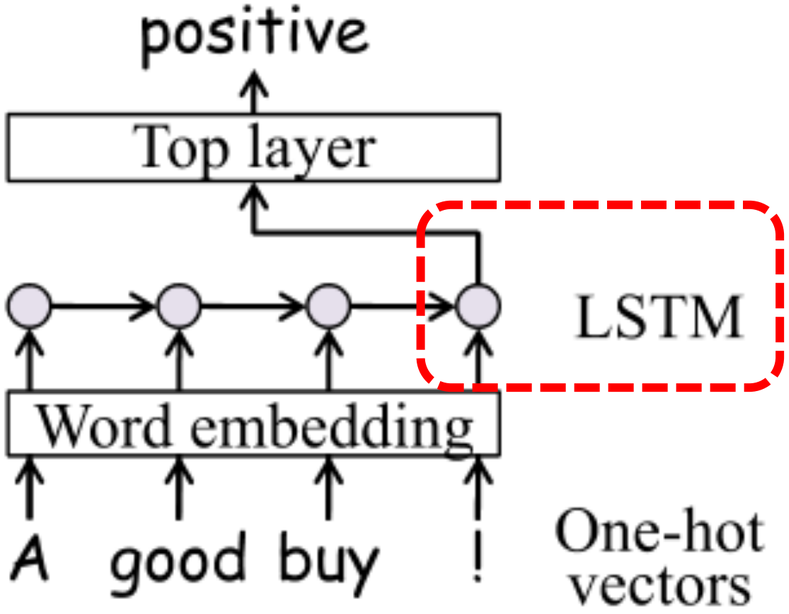


训练过程

```
xgu@jxgu:~/github/NLP_Practice.Pytorch/text_classification/rnn_based$ python main.py
Epoch:   0/ 50] Training Loss: 0.275, Testing Loss: 0.236, Training Acc: 0.536, Testing Acc: 0.613
Epoch:   1/ 50] Training Loss: 0.231, Testing Loss: 0.212, Training Acc: 0.611, Testing Acc: 0.669
Epoch:   2/ 50] Training Loss: 0.212, Testing Loss: 0.182, Training Acc: 0.649, Testing Acc: 0.726
Epoch:   3/ 50] Training Loss: 0.174, Testing Loss: 0.157, Training Acc: 0.729, Testing Acc: 0.763
Epoch:   4/ 50] Training Loss: 0.151, Testing Loss: 0.138, Training Acc: 0.758, Testing Acc: 0.784
Epoch:   5/ 50] Training Loss: 0.131, Testing Loss: 0.114, Training Acc: 0.783, Testing Acc: 0.809
Epoch:   6/ 50] Training Loss: 0.112, Testing Loss: 0.103, Training Acc: 0.814, Testing Acc: 0.823
Epoch:   7/ 50] Training Loss: 0.101, Testing Loss: 0.099, Training Acc: 0.831, Testing Acc: 0.825
Epoch:   8/ 50] Training Loss: 0.093, Testing Loss: 0.093, Training Acc: 0.844, Testing Acc: 0.839
Epoch:   9/ 50] Training Loss: 0.086, Testing Loss: 0.097, Training Acc: 0.853, Testing Acc: 0.837
Epoch:  10/ 50] Training Loss: 0.078, Testing Loss: 0.091, Training Acc: 0.865, Testing Acc: 0.837
Epoch:  11/ 50] Training Loss: 0.076, Testing Loss: 0.091, Training Acc: 0.869, Testing Acc: 0.838
Epoch:  12/ 50] Training Loss: 0.075, Testing Loss: 0.091, Training Acc: 0.871, Testing Acc: 0.838
Epoch:  13/ 50] Training Loss: 0.074, Testing Loss: 0.091, Training Acc: 0.872, Testing Acc: 0.838
Epoch:  14/ 50] Training Loss: 0.073, Testing Loss: 0.092, Training Acc: 0.874, Testing Acc: 0.837
Epoch:  15/ 50] Training Loss: 0.073, Testing Loss: 0.091, Training Acc: 0.876, Testing Acc: 0.839
Epoch:  16/ 50] Training Loss: 0.072, Testing Loss: 0.091, Training Acc: 0.875, Testing Acc: 0.840
Epoch:  17/ 50] Training Loss: 0.072, Testing Loss: 0.090, Training Acc: 0.877, Testing Acc: 0.841
Epoch:  18/ 50] Training Loss: 0.071, Testing Loss: 0.090, Training Acc: 0.880, Testing Acc: 0.843
Epoch:  19/ 50] Training Loss: 0.071, Testing Loss: 0.092, Training Acc: 0.880, Testing Acc: 0.842
Epoch:  20/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.843
Epoch:  21/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.881, Testing Acc: 0.843
Epoch:  22/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.881, Testing Acc: 0.843
Epoch:  23/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.881, Testing Acc: 0.843
Epoch:  24/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.843
Epoch:  25/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.844
Epoch:  26/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.844
Epoch:  27/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.845
Epoch:  28/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.882, Testing Acc: 0.844
Epoch:  29/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.844
Epoch:  30/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.844
Epoch:  31/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.844
Epoch:  32/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.844
Epoch:  33/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  34/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  35/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  36/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  37/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  38/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  39/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  40/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  41/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  42/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  43/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  44/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  45/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  46/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  47/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  48/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
Epoch:  49/ 50] Training Loss: 0.070, Testing Loss: 0.091, Training Acc: 0.883, Testing Acc: 0.845
```



问题？


positive
Top layer
LSTM
Word embedding
A
good
buy
!
One-hot
vectors

模型改进（ATTENTION）

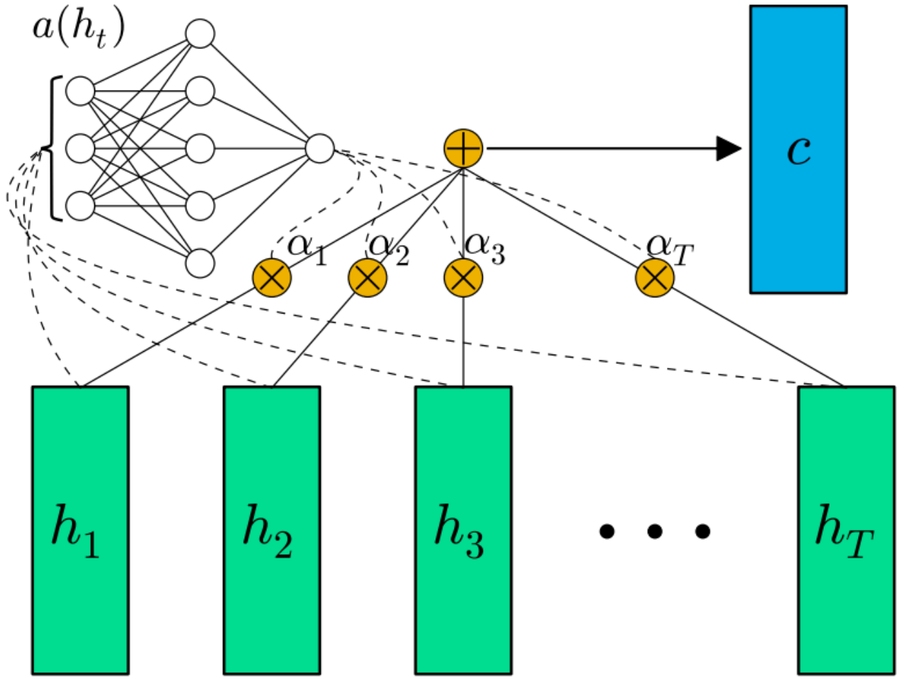
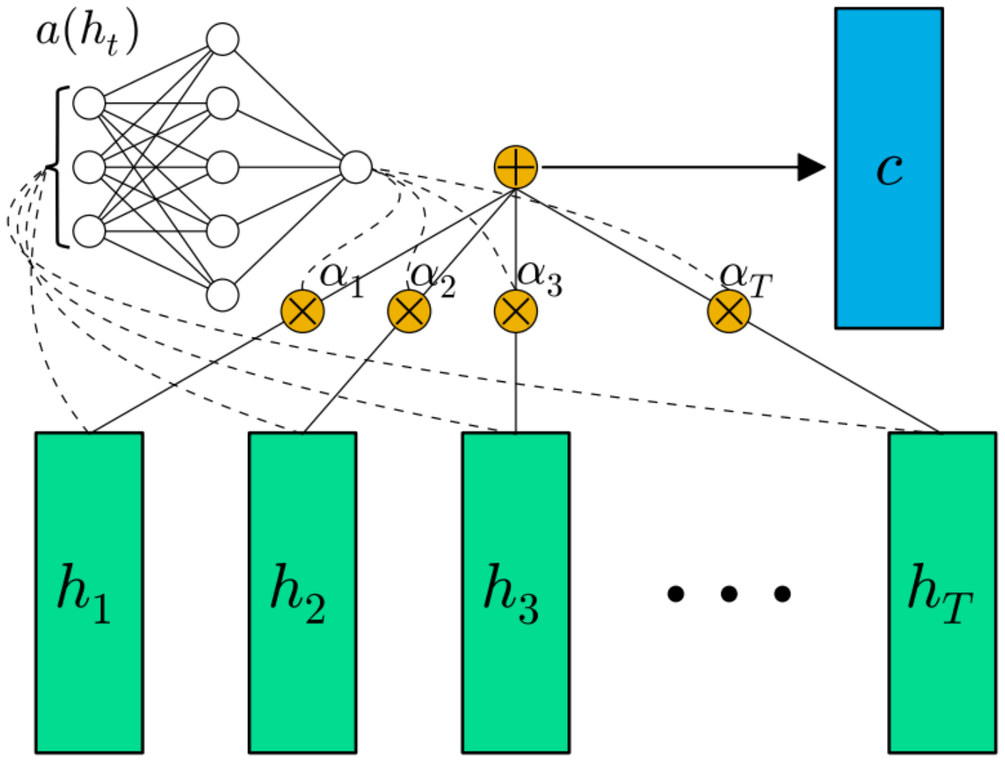


Figure 1: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

Raffel, Colin, and Daniel PW Ellis. "Feed-forward networks with attention can solve some long-term memory problems." arXiv preprint arXiv:1512.08756 (2015).

https://github.com/richliao/textClassifier/blob/master/textClassifierRNN.py

模型改进（ATTENTION）



$$c = \frac{1}{T}\sum_{t=1}^{T} h_t$$

$$e_t = a(h_t), \alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^{T}\exp(e_k)}, c = \sum_{t=1}^{T}\alpha_t h_t$$

Figure 1: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

END