SFU SIMON FRASER UNIVERSITY
THINKING OF THE WORLD

Revisiting Sentiment Analysis from Neural Network Perspective

Jiaxi Tang @ Simon Fraser University

jiaxit@sfu.ca

Agenda

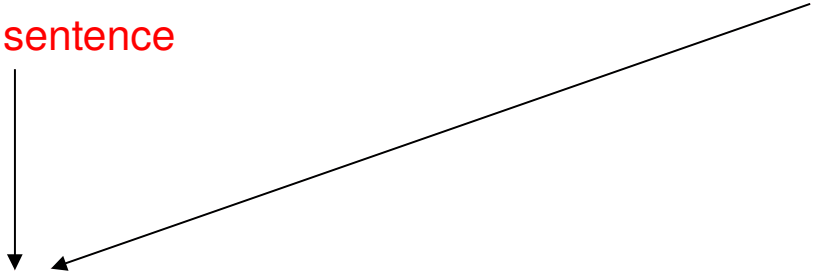
- Part 1: Sentiment Analysis from past（ 5min ）
  - Representation: Bag-of-words model
  - Method: KNN
  - Issues
- Part 2: Neural network(15min)
  - Representation: Word Embedding
  - Method: CNN
- Part 3: Practice(15min)
  - Code flow
  - Forward / Backward propagation
- Part 4: FAQ(15min)

Representation: Bag-of-words Model [1]

(1) John likes to watch movies. Mary likes movies too.

(2) John also likes to watch football games.

original sentence



(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]

(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

bag-of-words representation

["John",
"likes",
"to",
"watch",
"movies",
"also",
"football",
"games",
"Mary",
"too"]

vocabulary

[1] https://en.wikipedia.org/wiki/Bag-of-words_model

Representation: N-gram

(1) John likes to watch movies. Mary likes movies too.

(2) John also likes to watch football games.

Conceptually, we can view bag-of-word model as a special case of the n-gram model, with n=1

["John",
"likes",
"to",
"watch",
"movies",
"also",
"football",
"games",
"Mary",
"too"]

unigram feature

[“John likes”,
"likes to",
"to watch",
"watch
movies",
"Mary likes",
"likes movies",
"movies too",]

bigram feature

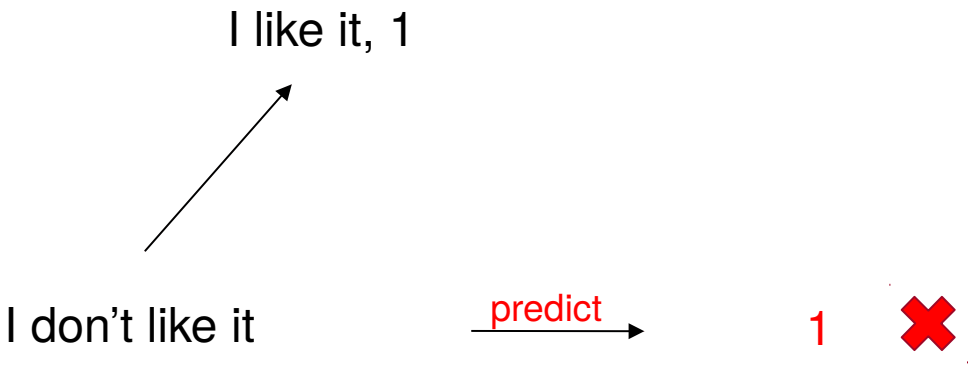
Method: K-Nearest Neighbor

Data:	Representation	Label:
(1) It is fantastic	[1,1,1,0,0,0,0,0]	1
(2) It is ultimately unsatisfying	[1,1,0,1,1,0,0,0]	0
(3) A mediocre film	[0,0,0,0,0,1,1,1]	0

It is ultimately mediocre film → predict → 0

[1,1,0,1,0,0,1,0]

Issue:

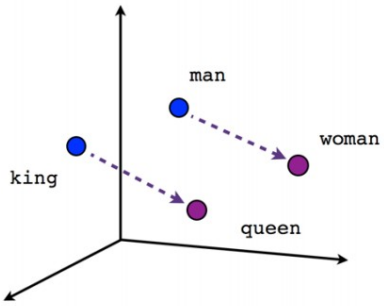


sparsity will lead to terrible performance!

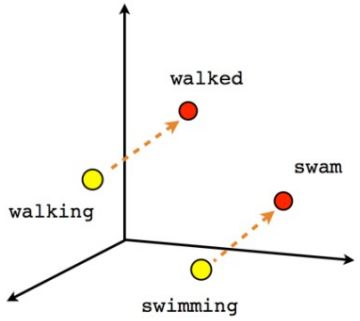
Representation: Word Embedding

Each word is associated with a vector(embedding) of latent values, e.g.

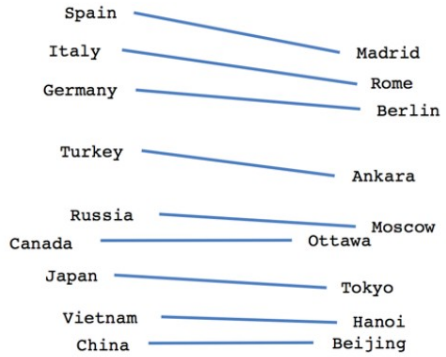
“like” = [0.7, 0.1, -0.5, 0.4, 1.2]



Male-Female



Verb tense



Country-Capital

Bag-of-words vs. Word Embedding

Bag-of-words	Word embedding
Each value has explicit meaning	Values are latent, don't know the meaning
Very sparse	Dense real values
Usually don't have contextual information	Have contextual information

Representation: Word Embedding

two ways to get word embedding:

(1) Initialize word embedding with random value, train them on the given dataset, and learn their optimal value.

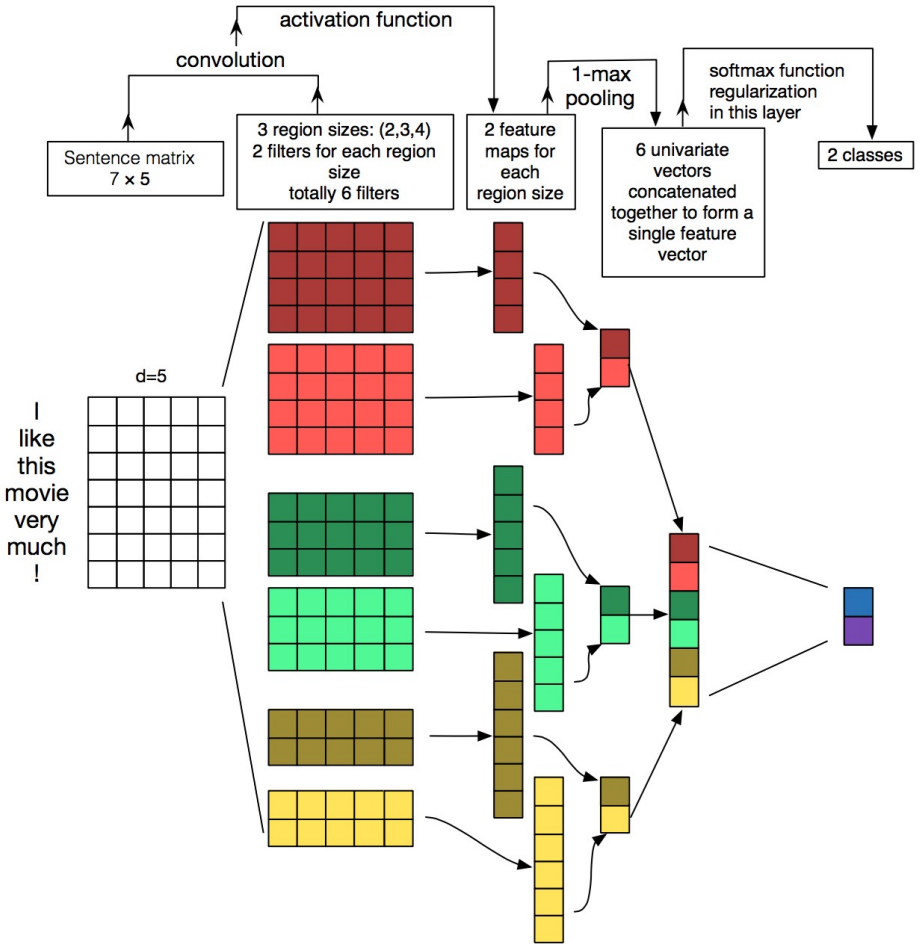
(2) Use existing word embedding trained from other datasets.

Google word2vec: https://code.google.com/archive/p/word2vec/

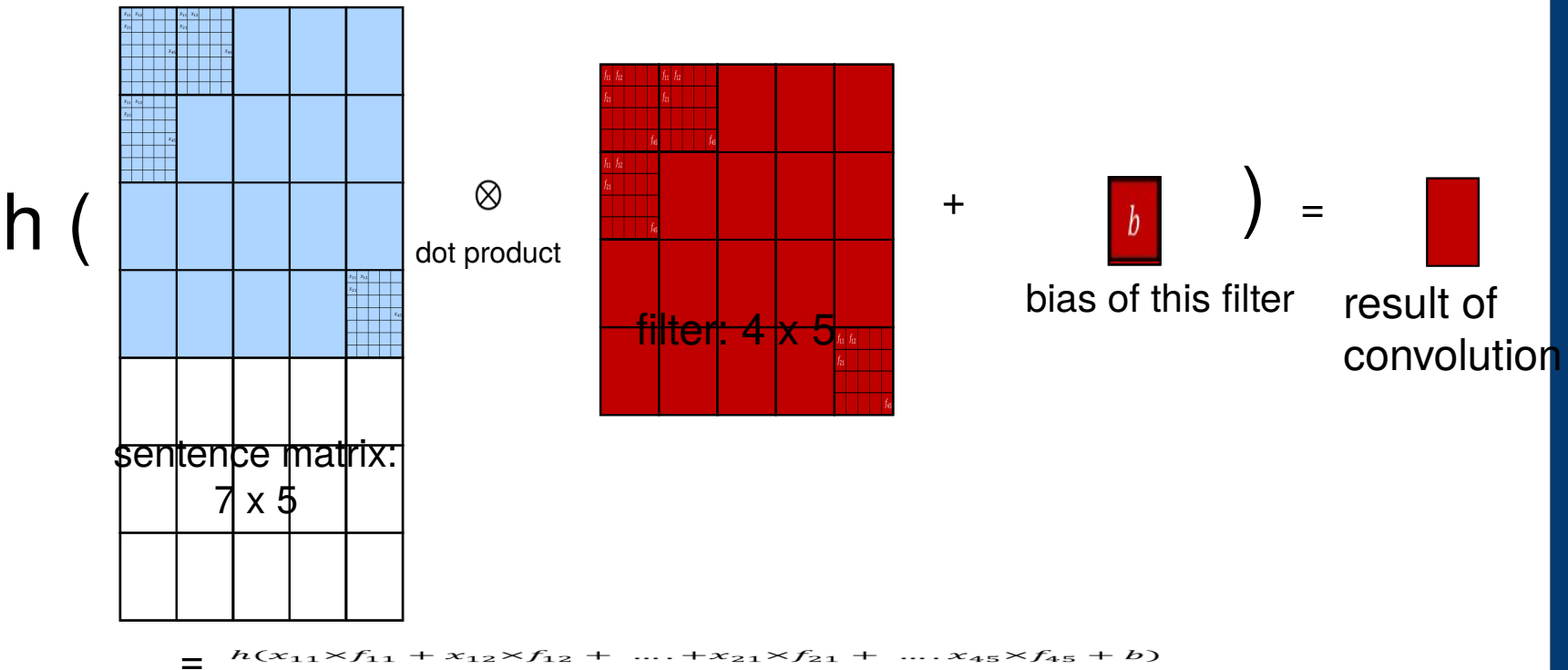
GloVe: https://github.com/stanfordnlp/GloVe

Try (1) first, then use (2)

Method: CNN model

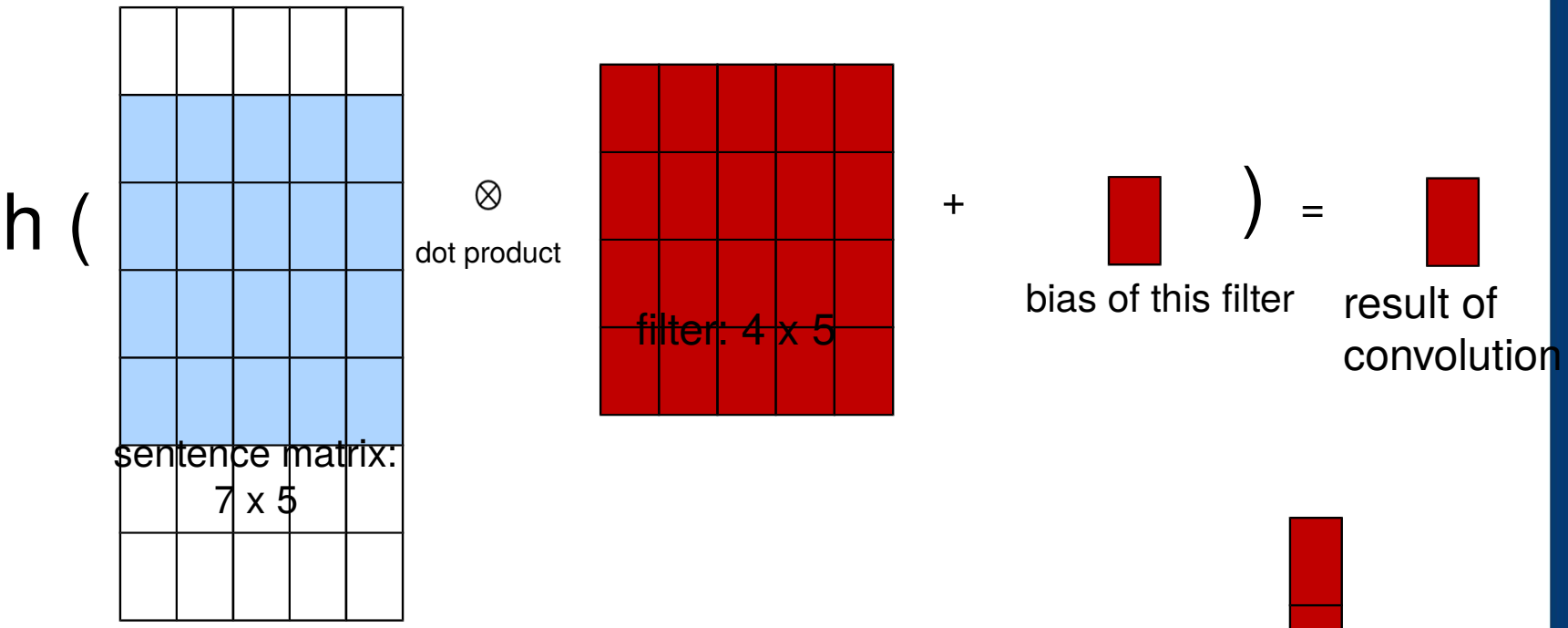


Method: CNN model – convolution

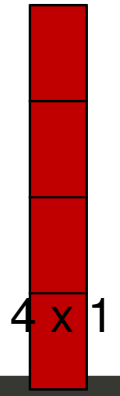


$$= h(x_{11} \times f_{11} + x_{12} \times f_{12} + \ldots + x_{21} \times f_{21} + \ldots x_{45} \times f_{45} + b)$$

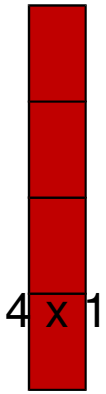
Method: CNN model – convolution



After convolutional operation with this filter, we get 4 values:

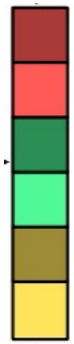


Method: CNN model - pooling



pick largest one (1-max pooling)

For the total 6 filters, we get 6 values, then we concatenate them together:



Method: CNN model

a filter with size = n

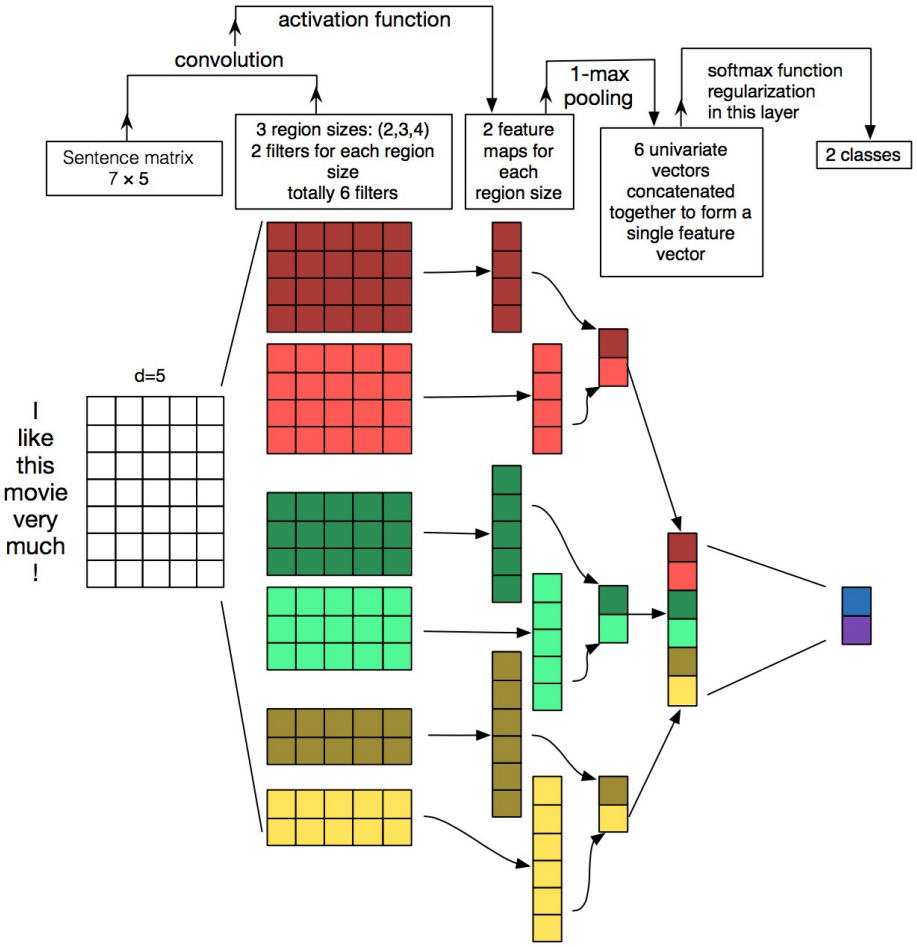
↓

detect a specific n-gram feature

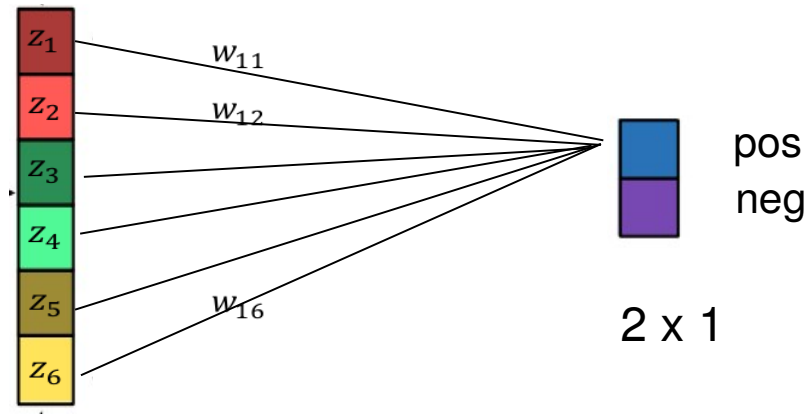
feature vector

↓

embedding of this sentence



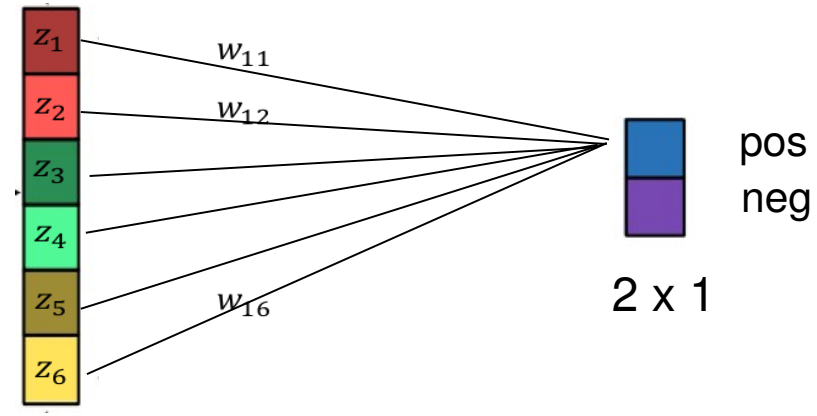
Method: CNN model – output layer, softmax loss



$$pos = z_1 \times w_{11} + z_2 \times w_{12} + \ldots + z_6 \times w_{16} + b_{pos}$$

$$neg = z_1 \times w_{21} + z_2 \times w_{22} + \ldots + z_6 \times w_{26} + b_{neg}$$

Method: CNN model – softmax loss



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

scores = unnormalized log probabilities of the classes

pos = -0.2 (ground truth)
neg = 1.3

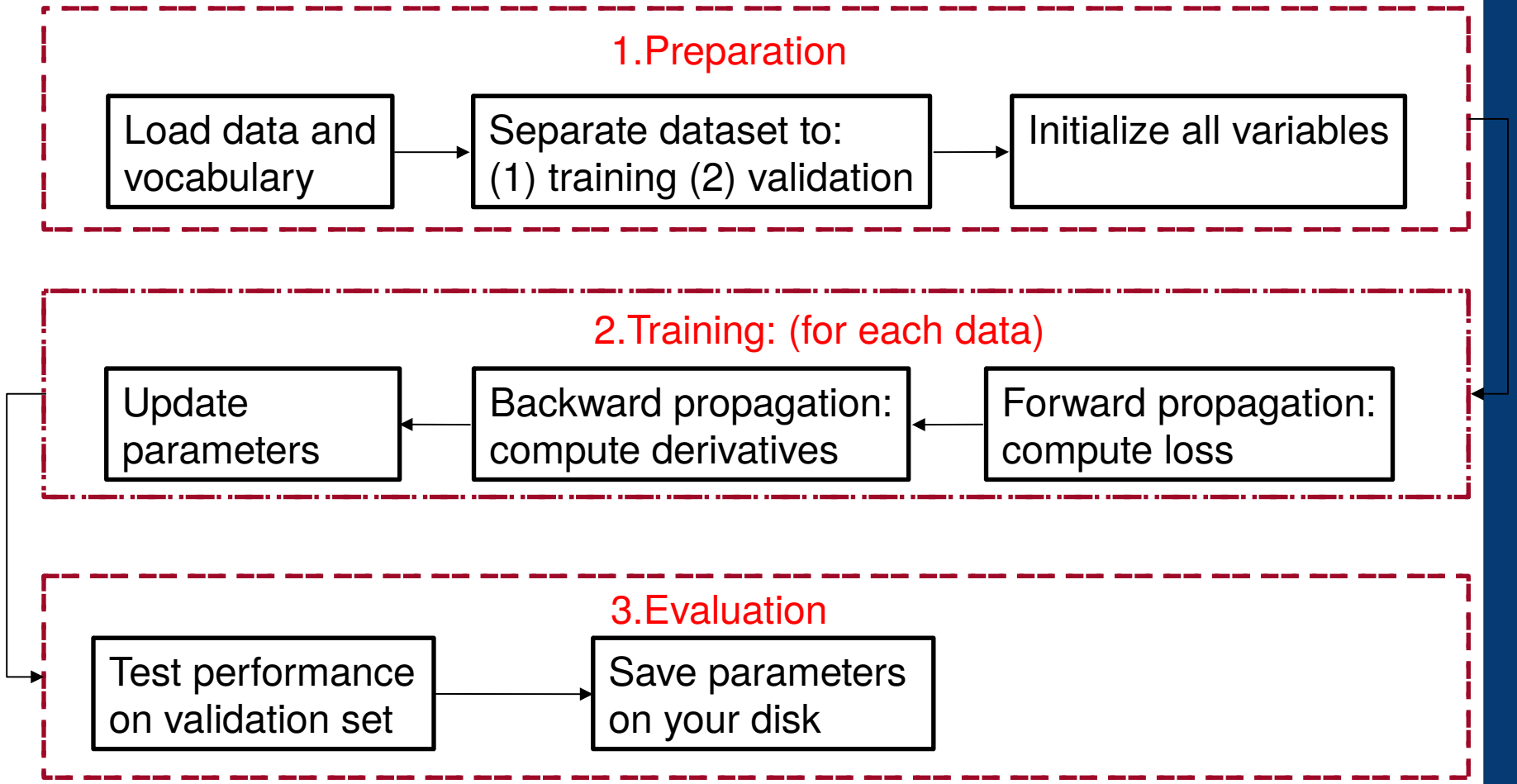
exp → 0.8187, 3.6693

normalize → 0.1824, 0.8176

probabilities

$$L_i = -\log(0.1824) = 1.7016$$

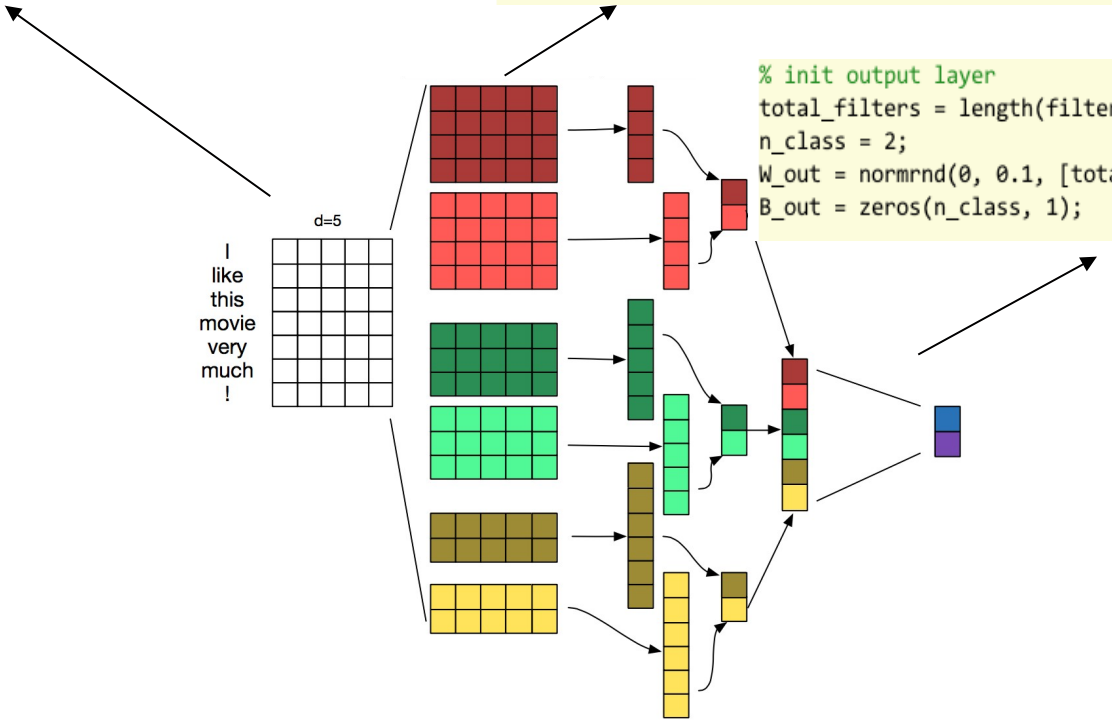
Practice: Code flow



Practice: Initialization

```
% init embedding
d = 5;
total_words = length(wordMap);
% random sample from normal distribution
% with mean=0, variance=0.1
T = normrnd(0, 0.1, [total_words, d]);
```

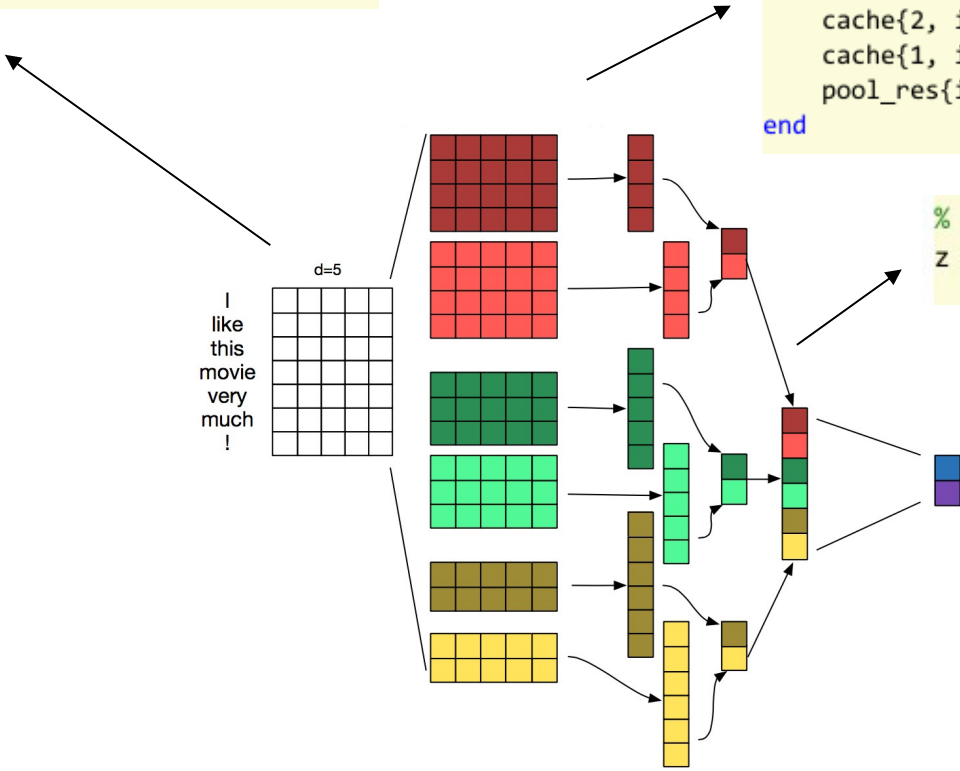
```
% init filters
filter_size = [2,3,4];
n_filter = 2;

W_conv = cell(length(filter_size), 1);
B_conv = cell(length(filter_size), 1);

for i = 1: length(filter_size)
    % get filter size
    f = filter_size(i);
    % init W with: FW x FH x FC x K
    W_conv{i} = normrnd(0, 0.1, [f, d, 1, n_filter]);
    B_conv{i} = zeros(n_filter, 1);
end
```




```
% init output layer
total_filters = length(filter_size) * n_filter;  % will be 6
n_class = 2;
W_out = normrnd(0, 0.1, [total_filters, n_class]);
B_out = zeros(n_class, 1);
```

Practice: Forward propagation

```
% get sentence matrix
% words_indexs = [wordMap('i'), wordMap('like'),
% .., wordMap('!')]
X = T(word_indexs, :);
```



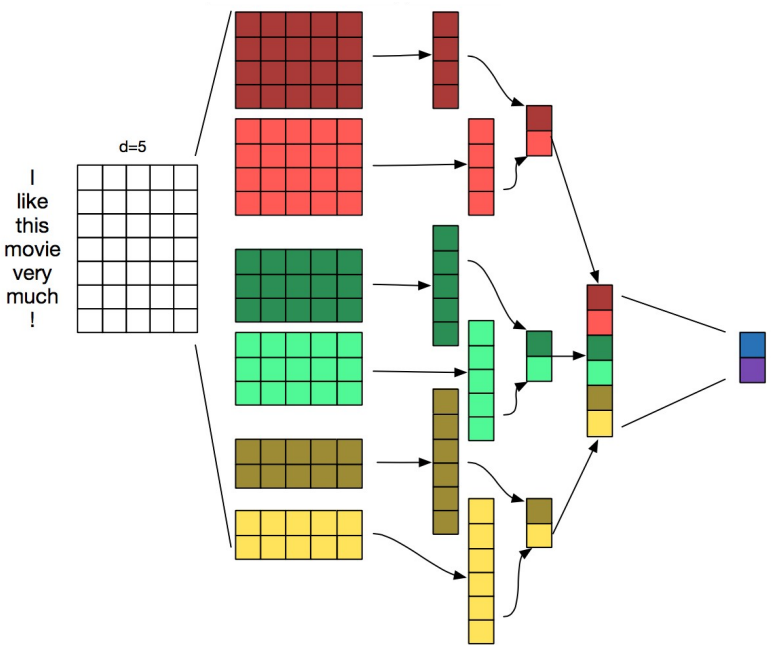

```
pool_res = cell(1, length(filter_size));
cache = cell(2, length(filter_size));
for i = 1: length(filter_size)
    % convolutional operation
    conv = vl_nnconv(X, W_conv{i}, B_conv{i});

    % apply activation function: relu
    relu = vl_nnrelu(conv);

    % 1-max pooling operation
    sizes = size(conv);
    pool = vl_nnpool(relu, [sizes(1), 1]);

    % important: keep these values for back-prop
    cache{2, i} = relu;
    cache{1, i} = conv;
    pool_res{i} = pool;
end
```

```
% concatenate
z = vl_nnconcat(pool_res, 3);
```

```
% conpute loss
% o: value of output layer
% y: ground truth label (1 or 2)
loss = vl_nnloss(o, y);
```

Practice: Backward propagation

Part2 of http://www.robots.ox.ac.uk/~vgg/practicals/cnn/



Challenge:

1. split by ratio ✉ cross-validation
2. use existing word2vec to initialize word embedding
3. regularization (l2 norm, dropout)
4. SGD ✉ RMSProp or Adam
5. model ensembles
5. use mini-batch to accelerate
6. …

FAQ

SFU SIMON FRASER UNIVERSITY
THINKING OF THE WORLD

Thanks