

# Lecture 10. Neural Networks for text

**Handling text data**

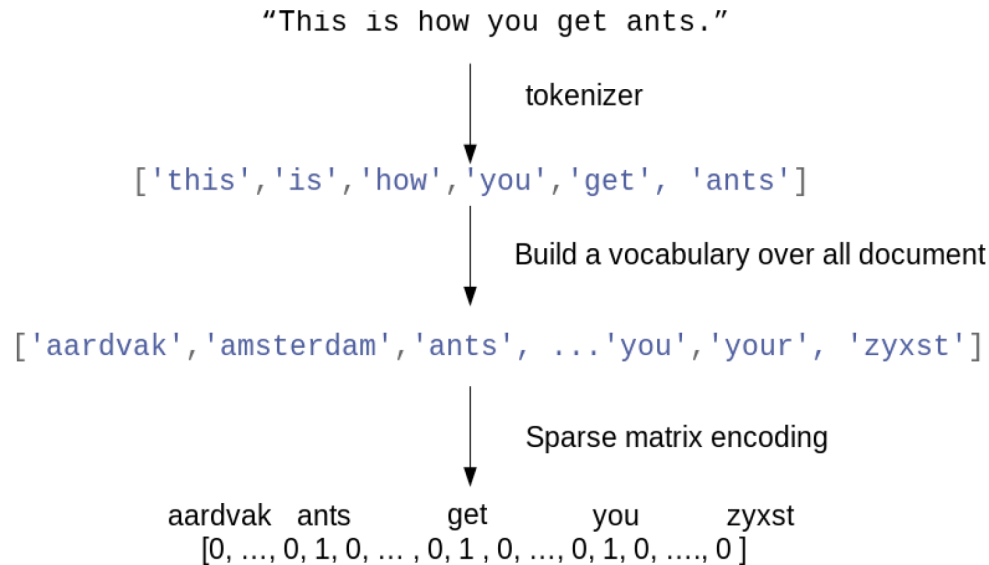
Joaquin Vanschoren

# Overview

- Bag of words representations
- Word embeddings
  - Learning word embeddings
  - Pretrained word embeddings
- 1D convolutional nets
- In practice

# Bag of word representation

- First, build a *vocabulary* of all occurring words. Maps every word to an index.
- Represent each document as an  $N$  dimensional vector (top- $N$  most frequent words)
  - One-hot (sparse) encoding: 1 if the word occurs in the document
- Destroys the order of the words in the text (hence, a 'bag' of words)



## Example: IMBD review database

- 50,000 reviews, labeled positive (1) or negative (0)
  - Every row (document) is one review, no other input features
  - Already tokenized. All markup, punctuation,... removed

Using TensorFlow backend.

Text contains 88584 unique words

Review 0: the this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert redford's is an amazing actor and now the same being director norman's father came from the same scottish island as myself so i loved

Review 5: the begins better than it ends funny that the russian submarine crew outperforms all other actors it's like those scenes where documentary shots br br spoiler part the message deciphered was contrary to the whole story it just does not mesh br br

Review 10: the french horror cinema has seen something of a revival over the last couple of years with great films such as inside and switchblade romance bursting on to the scene maléfique preceded the revival just slightly but stands head and shoulders over most modern horror titles and is surely one

## Bag of words with one-hot-encoding

- Encoded review: shows the list of word IDs. Words are sorted by frequency of occurrence.
  - Allows to easily remove the most common and least common words
- One-hot-encoded review: '1' if the word occurs.
  - Only the first 100 of 10000 values are shown

Review 3: the the scots excel at storytelling the traditional sort many years after the event i can still see in my mind's eye an elderly lady my friend's mother retelling the battle of culloden she makes the characters come alive her passion is that of an eye witness one to the events on the sodden heath a mile or so from where she lives br br of course it happened many years before she was born but you wouldn't guess from

Encoded review: [1, 1, 18606, 16082, 30, 2801, 1, 2037, 429, 108, 150, 100, 1, 1491, 10, 67, 128, 64, 8, 58, 15302, 741, 32, 3712, 758, 58, 5763, 449, 921, 1, 1, 982, 4, 64314, 56, 163, 1, 102, 213, 1236, 38, 1794, 6, 12, 4, 32, 741, 2410, 28, 5, 1, 684, 20, 1, 33926, 7336, 3, 3690, 39, 35, 36, 118, 56, 453, 7, 7, 4, 262, 9, 572, 108, 150, 156, 56, 13, 1444, 18, 22, 583, 479, 36]

```
One-hot-encoded review: [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
0. 1. 1. 1. 0. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 1.]
```

## Word counts

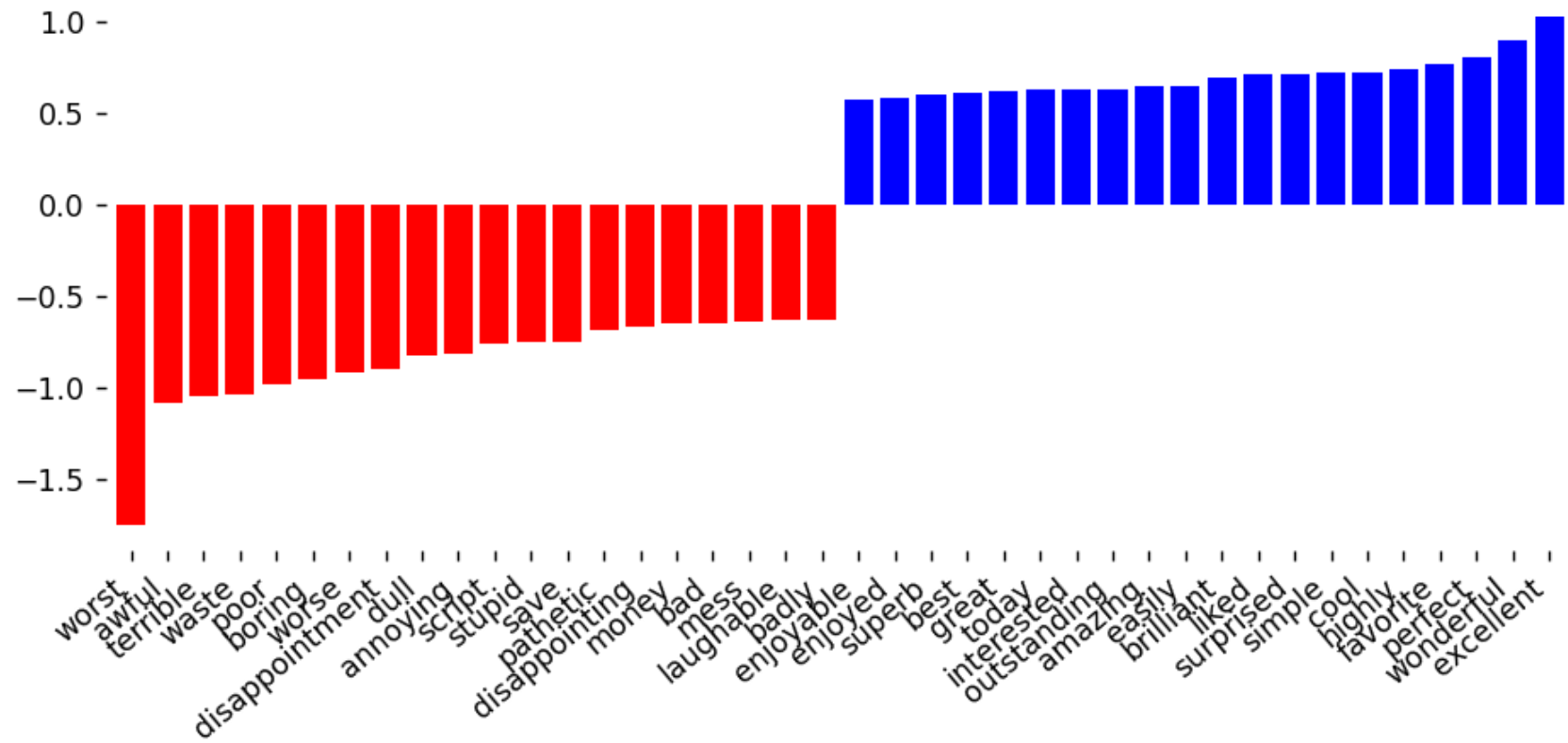
- Count the number of times each word appears in the document
- Example using sklearn CountVectorizer
  - Here, we fit the Countvectorizer on just the first 2 reviews

```
Vocabulary (feature names) after fit: ['actor', 'amazing', 'an', 'and', 'are',  
'as', 'bad', 'be', 'being', 'best', 'big', 'boobs', 'brilliant', 'but', 'cam  
e', 'casting', 'cheesy', 'could', 'describe', 'direction', 'director', 'ever',  
'everyone', 'father', 'film', 'from', 'giant', 'got', 'had', 'hair', 'horror',  
'hundreds', 'imagine', 'is', 'island', 'just', 'location', 'love', 'loved', 'm  
ade', 'movie', 'movies', 'music', 'myself', 'norman', 'now', 'of', 'on', 'pape  
r', 'part', 'pin', 'played', 'plot', 'really', 'redford', 'ridiculous', 'rober  
t', 'safety', 'same', 'scenery', 'scottish', 'seen', 'so', 'story', 'suited',  
'terrible', 'the', 'there', 'these', 'they', 'thin', 'this', 'to', 've', 'wa  
s', 'words', 'worst', 'you']  
Count encoding doc 1: [1 1 1 2 0 1 0 0 2 0 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 0 0  
0 0 0 0 1 1 1 2 1  
0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 2 1 1 0 1 1 1 0 4 1 0 1 0 1 0 0  
1 0 0 1]  
Count encoding doc 2: [0 0 0 3 1 0 1 1 0 1 2 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 1 1  
1 1 1 1 0 1 0 0 0  
1 0 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 4 0 1 0 1 2 2 1  
0 1 1 0]
```

# Classification

- With this tabular representation, we can fit any model (e.g. Logistic regression)
- Visualize coefficients: which words are indicative for positive/negative reviews?

Logistic regression accuracy: 0.8538



# Preprocessing

- Tokenization: how to you split text into words? On spaces only? Also -, ` ?
- Stemming: naive reduction to word stems. E.g. 'the meeting' to 'the meet'
  - Lemmatization: smarter reduction (NLP-based). E.g. distinguishes between nouns and verbs
- Discard stop words ('the', 'an',...)
- Only use  $N$  (e.g. 10000) most frequent words
  - Or, use a hash function (risks collisions)
- n-grams: Use combinations of  $n$  adjacent words next to individual words
  - e.g. 2-grams: "awesome movie", "movie with", "with creative", ...
- Character n-grams: combinations of  $n$  adjacent letters: 'awe', 'wes', 'eso',...
- Useful libraries: nlTK (<https://www.nltk.org/>), spaCy (<https://spacy.io/>), gensim (<https://radimrehurek.com/gensim/>),...



# Scaling

- L2 Normalization (vector norm): sum of squares of all word values equals 1

- Normalized Euclidean distance is equivalent to cosine distance
- Works better for distance-based models (e.g. kNN, SVM,...)

$$t_i = \frac{t_i}{\|t\|_2}$$

- Term Frequency - Inverted Document Frequency (TF-IDF)

- Scales value of words by how frequently they occur across all  $N$  documents
- Words that only occur in few documents get higher weight, and vice versa

$$t_i = \frac{t_i}{\log\left(\frac{N}{|\{d \in D: t_i \in d\}|}\right)}$$

- Usually done in preprocessing, e.g. sklearn `Normalizer` or `TfidfTransformer`

- L2 normalization can also be done in a `Lambda` layer

```
model.add(Lambda(lambda x: tf.linalg.normalize(x,axis=1)))
```

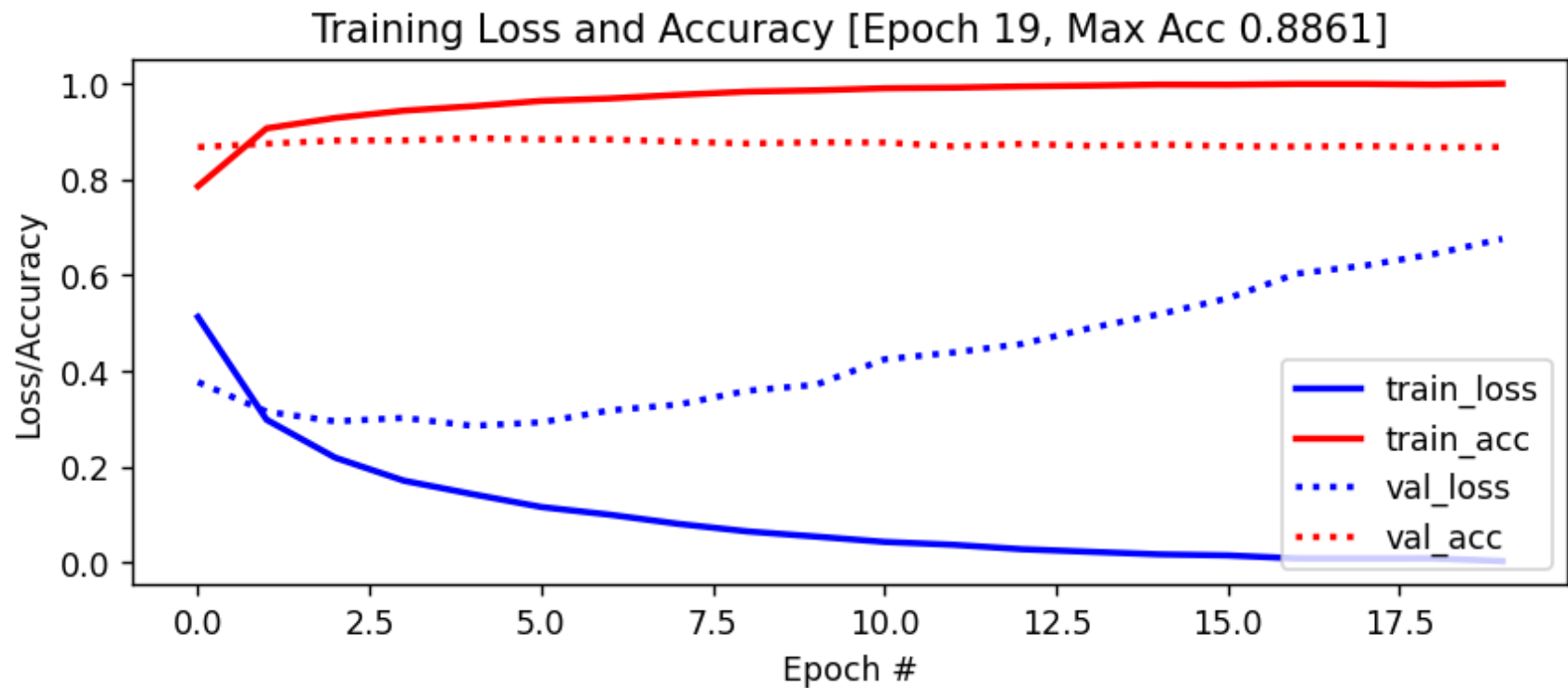
# Neural networks on bag of words representations

- We could build simple neural networks on bag-of-word vectors
  - E.g. One-hot-encoding, 10000 most frequent words, drop top-3 stopwords
- Simple model with 2 dense layers and ReLU activation
  - Binary classification: single output node: convert 0/1 label to float
  - Sigmoid activation for output node and binary cross-entropy loss

```
model = models.Sequential()  
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))  
model.add(layers.Dense(16, activation='relu'))  
model.add(layers.Dense(1, activation='sigmoid'))  
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['a  
ccuracy'])
```

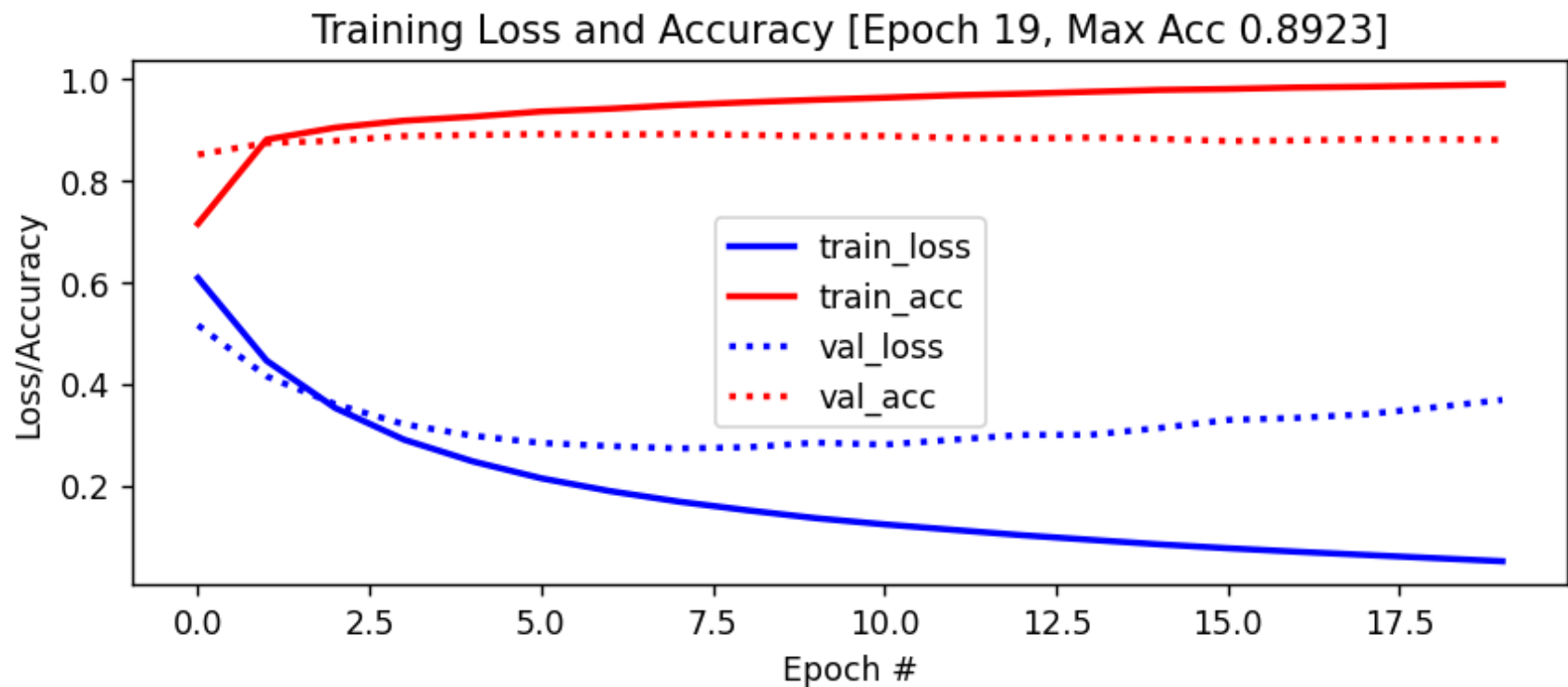
# Model selection

- Take a validation set of 10,000 samples from the training set
- The validation loss peaks after a few epochs, after which the model starts to overfit
  - Performance is better than Logistic regression (obviously)



## Regularization: smaller networks

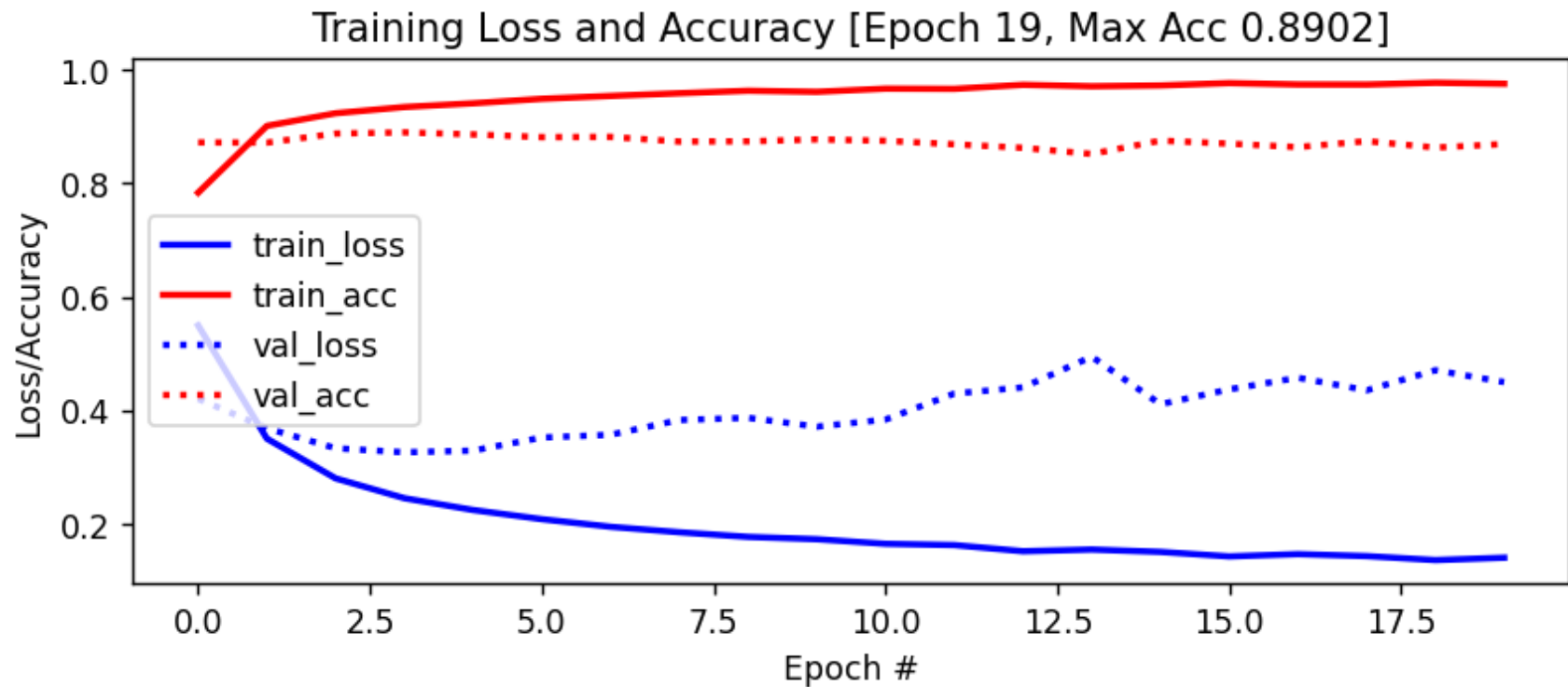
- The easiest way to avoid overfitting is to use a simpler model
  - E.g. use only 4 hidden nodes
- Less overfitting, but validation accuracy about the same



## Weight regularization (L2)

- L2 regularized model (on 16 hidden nodes) is equally/more resistant to overfitting, even though both have the same number of parameters
- Validation accuracy doesn't improve

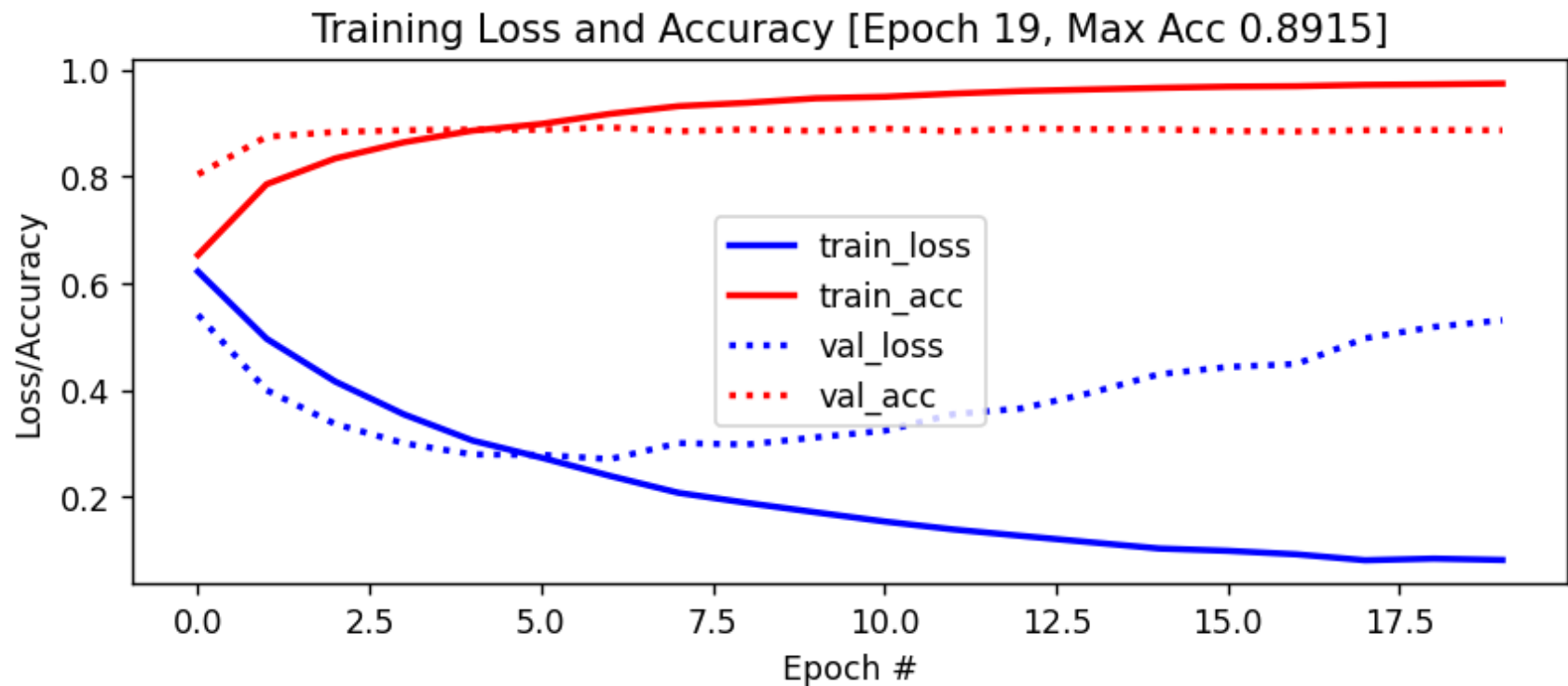
```
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),  
                           activation='relu'))
```



## Dropout Regularization

- Overfits less than the original model, but more than L2-regularization

```
dpt_model.add(layers.Dense(16, activation='relu'))  
dpt_model.add(layers.Dropout(0.5))
```



## Tuning

- We can vary the remaining hyperparameters
  - Number of layers or hidden units, activation functions, optimizers, batch size, learning rates,...
- Quick grid search (using early stopping to select number of epochs):
  - We're not getting much better results using this representation

Out[15]:

		mean_test_score	mean_train_score
param_hidden_size	param_dropout_rate		
8	0.10	0.89	0.96
	0.25	0.89	0.96
	0.50	0.89	0.96
16	0.10	0.89	0.96
	0.25	0.88	0.96
	0.50	0.89	0.96
64	0.10	0.88	0.96
	0.25	0.88	0.96
	0.50	0.89	0.96

## Predictions

Let's look at a few predictions:

Review 0: ? please give this one a miss br br ? ? and the rest of the cast rendered terrible performances the show is flat flat flat br br i don't know how michael madison could have allowed this one on his plate he almost seemed to know this wasn't going to work out and his performance was quite ? so all you madison fans give this a miss

Predicted positiveness: [0.068]

Review 16: ? from 1996 first i watched this movie i feel never reach the end of my satisfaction i feel that i want to watch more and more until now my god i don't believe it was ten years ago and i can believe that i almost remember every word of the dialogues i love this movie and i love this novel absolutely perfection i love willem ? he has a strange voice to spell the words black night and i always say it for many times never being bored i love the music of it's so much made me come into another world deep in my heart anyone can feel what i feel and anyone could make the movie like this i don't believe so thanks thanks

Predicted positiveness: [0.661]



# Multi-class classification (topic classification)

- Reuters dataset: 11,000 news stories, 46 topics
- Each row is one news story
  - We again use the 10,000 most frequent words, and drop the top-3 stop words
- Each word is replaced by a *word index* (word ID)

News wire: ? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln dlrs from 1 2 5 mln dlrs it said cash flow per share this year should be 2 50 to three dlr s reuter 3

Encoded: [1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 3095, 111, 16, 369, 186, 90, 67, 7]

Topic: 3

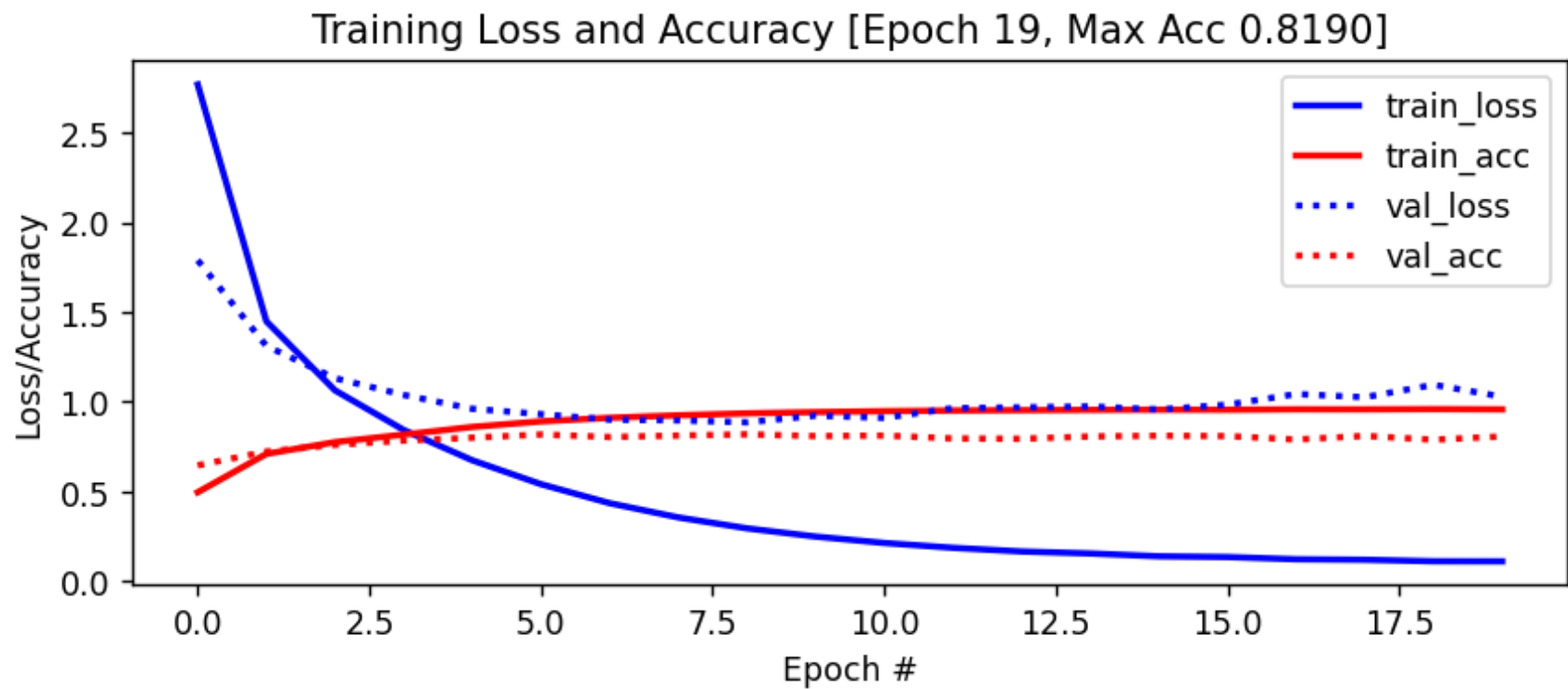
## Building the network

- 16 hidden units may be too limited to learn 46 topics, hence we use 64
- The output layer now needs 46 units, one for each topic
  - We use `softmax` activation for the output to get probabilities
  - Loss function is now `categorical_crossentropy`
    - Hence, output labels need to be one-hot-encoded

```
model = models.Sequential()  
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))  
model.add(layers.Dense(64, activation='relu'))  
model.add(layers.Dense(46, activation='softmax'))  
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

## Result

- Some overfitting after about 8 epochs
- Regularization may help, but let's try different representations



# Word Embeddings

- An embedding maps each word to a point in a much smaller  $m$ -dimensional space (e.g. 300 values)
- 2 main approaches:
  - Learn the embedding jointly with your main task.
    - Add an *embedding layer* with  $m$  hidden nodes to map word IDs to an  $m$ -dimensional vector
    - Add your hidden and output layers, learn weights end-to-end with SGD
  - Use a pre-trained embedding
    - Usually trained on another, much bigger task (e.g. Wikipedia, Google News,...)
    - Freeze embedding weights to produce simple word embeddings, or finetune to a new tasks

# Training Embedding layers

- Input layer uses fixed length documents. E.g. 100 nodes for 100 word IDs
  - Pad with 0's if document is shorter. Input is a 2D tensor of shape (samples, max\_length)
- Add an *embedding layer* to learn the embedding
  - First represents every word as an  $n$ -dimensional one-hot encoded bag of words.
    - Reshapes 2D tensor to 3D tensor of shape (samples, max\_length,  $n$ )
  - To learn an  $m$ -dimensional embedding, use  $m$  hidden nodes
    - Learn a weight matrix  $W^{n \times m}$  that maps any one-hot-encoded word to its embedding
  - Uses a linear activation function:  $\mathbf{X}_{embed} = W\mathbf{X}_{orig}$ 
    - Outputs a 3D tensor of shape (samples, max\_length,  $m$ )
- Add layers to map word embeddings to the desired output (see later)
- Learn all weights from the labeled data.

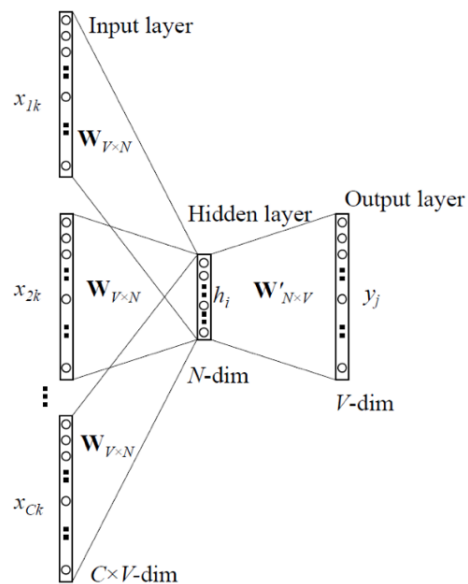
# Pre-trained embeddings

- With more data we can build better embeddings, but we also need more labels
- Solution: learn embedding on auxiliary task that doesn't require labels
  - E.g. given a word, predict the surrounding words.
  - Nowadays, this is called self-supervised learning. Supervision is provided by data itself.
- Most common approaches:
  - Word2Vec: Learn neural embedding for a word based on surrounding words in text
    - Encoding is learned using a 1-layer neural net
  - GloVe (Global Vector): Count co-occurrences of words in a matrix
    - Use a low-rank approximation to get a condensed vector representation
  - FastText: learns embedding for character n-grams
    - Can also produce embeddings for new, unseen words
  - Language models (BERT, ELMO, GPT3,...): learn a context-dependent embedding
    - Words get different embeddings based on the sentence they appear in

## Word2Vec

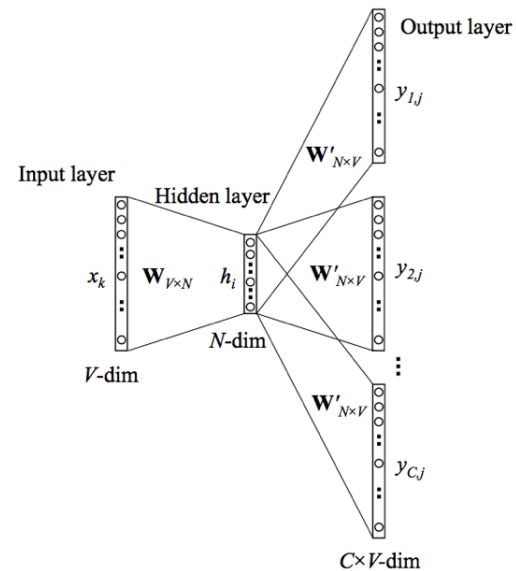
- Move a window over text to get  $C$  context words ( $V$ -dim one-hot)
- Add hidden layer with  $N$  linear nodes, average pooling, and softmax layer(s)
- CBOW: predict word given context, use weights of last layer as embedding
- Skip-Gram: predict context given word, use weights of first layer as embedding (better for large corpora)

I like playing football with my friends



### CBOW

predict word given context

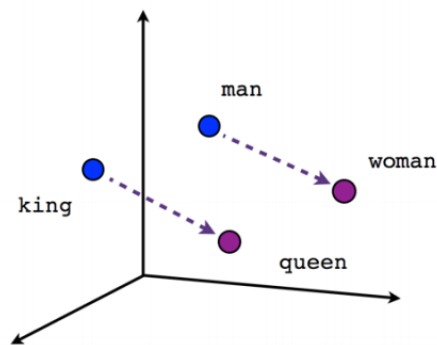


### Skip-gram

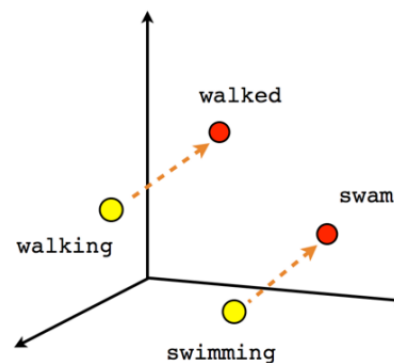
predict context given word

## Word2Vec properties

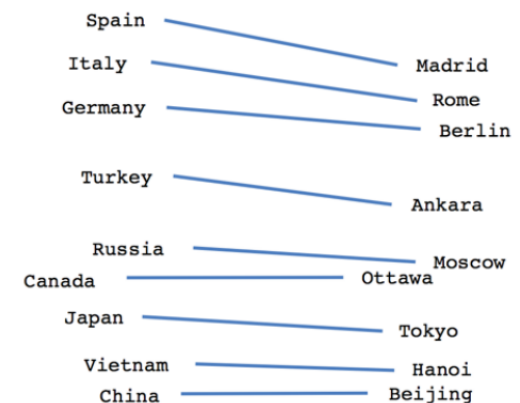
- Word2Vec happens to learn interesting relationships (<https://www.aclweb.org/anthology/N13-1090/>) between words
  - Simple vector arithmetic can map words to plurals, conjugations, gender analogies,...
  - e.g. Gender relationships:  $vec_{king} - vec_{man} + vec_{woman} \sim vec_{queen}$
  - PCA applied to embeddings shows Country - Capital relationship
- Careful: embeddings can capture gender and other biases (<https://arxiv.org/abs/1607.06520>), present in the data.
  - Important unsolved problem!



Male-Female



Verb tense



Country-Capital



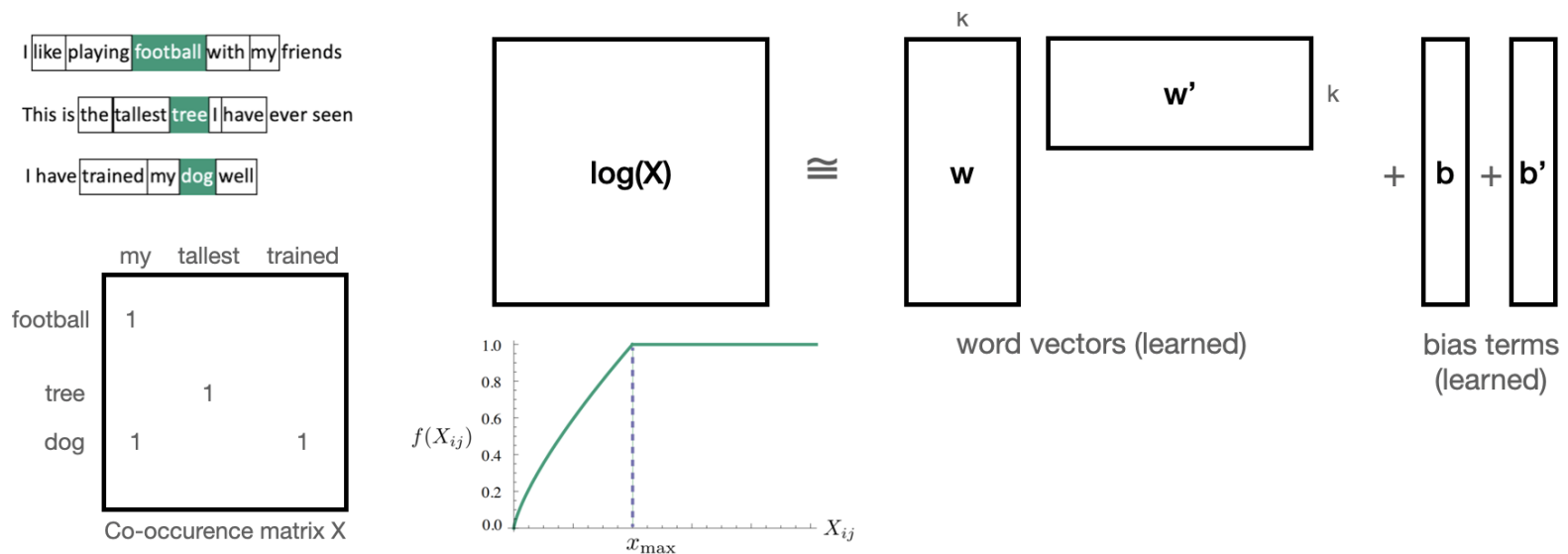
## FastText

- Limitations of Word2Vec:
  - Cannot represent new (out-of-vocabulary) words
  - Words like 'meet' and 'meeting' are learned independently: less efficient (no parameter sharing)
- FastText addresses this by using character n-grams
  - Basic model can be Skip-gram or CBOW
  - Words are represented by all character n-grams of length 3 to 6
    - "football" 3-grams: <fo, foo, oot, otb, tba, bal, all, ll>
    - Because there are so many n-grams, they are hashed
      - Dimensionality  $V = \text{bin size}$
      - Representation of word "football" is sum of its n-gram embeddings
- Training includes both positive examples (in-context words) and random negative examples

## Global Vector model (GloVe)

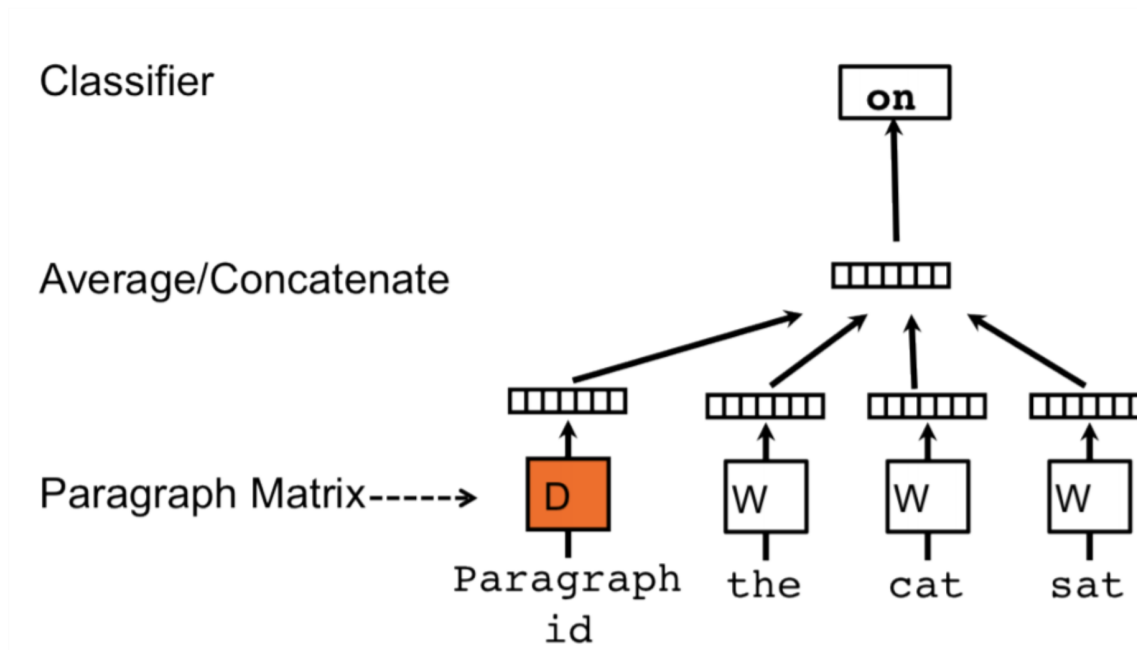
- Builds a co-occurrence matrix  $\mathbf{X}$ 
  - Counts how often 2 words occur in the same context (and how close)
- Learns a  $k$ -dimensional embedding  $w$  through matrix factorization with rank  $k$ 
  - Actually learns 2 embeddings  $w$  and  $w'$  (differ in random initialization)
- Minimizes loss  $\mathcal{L}$ , where  $b_i$  and  $b'_i$  are bias terms and  $f$  is a weighting function

$$\mathcal{L} = \sum_{i,j=1}^V f(\mathbf{X}_{ij})(\mathbf{w}_i \mathbf{w}'_j + b_i + b'_j - \log(\mathbf{X}_{ij}))^2$$



# Document/paragraph embeddings

- Simplest approach to represent a document: sum or average of all word vectors
- Doc2Vec
  - Next to word embeddings, also learn an embedding for the document/paragraph itself
  - Acts as a memory that remembers what is missing from the current context (the topic of the paragraph)
  - Can be used to determine semantic similarity between documents.
  - Can be tricky to train.

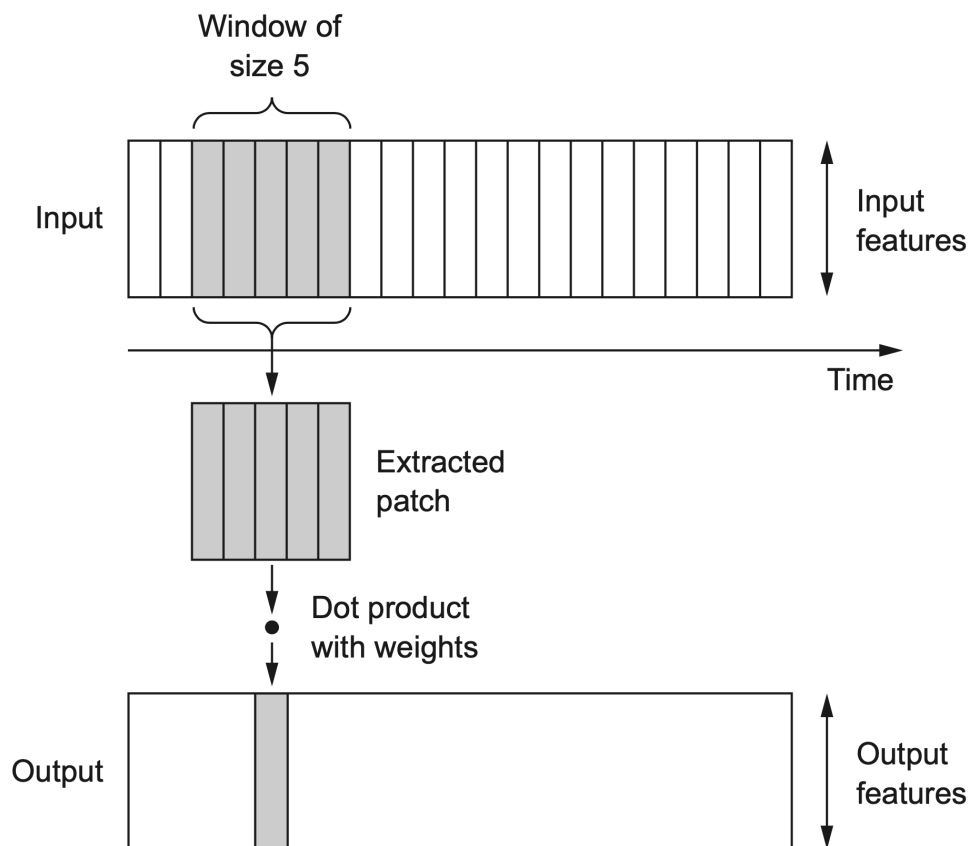


# Using word embeddings

- Say that we have  $m$ -dimensional word embeddings and documents of size  $l$
- Our embedding layer will produce a 3D tensor of shape (samples,  $l$ ,  $m$ )
- We now need to map this to the desired output
  - Simply flattening the tensor learns a direct mapping. This destroys the word order.
  - A recurrent neural network (RNN) could leverage the exact sequence of the words
    - Can be slow to train
  - A 1D convolutional network can also leverage locality (learn local patterns)
    - Often competitive with RNNs, and much faster/cheaper
    - Works well for simple tasks like classification and forecasting

# 1D convolutional networks

- Similar to 2D convnets, 1D convnets extract local 1D patches from image tensors
  - Apply identical transformation (filter) to every patch
- Pattern learned can later be recognized elsewhere (translation invariance)
- 1D Pooling: extracts 1D patches and outputs the max value (max pooling) or average



# Neural network for text in practice

## Training embedding layers

- Simple model with an embedding layer and direct mapping to binary class

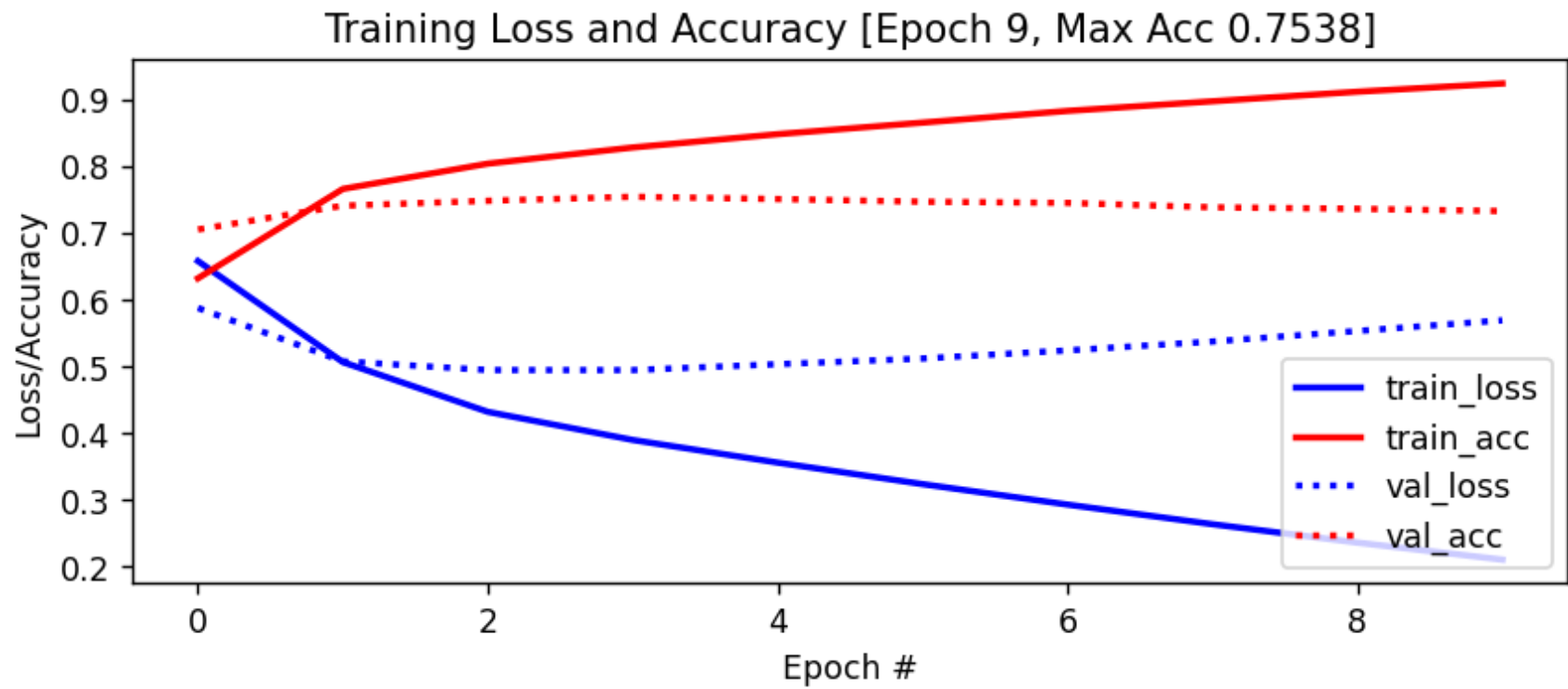
```
max_length = 20 # pad documents to a maximum number of words
vocab_size = 10000 # vocabulary size
embedding_length = 8 # embedding length (more would be better)

model = models.Sequential()
model.add(layers.Embedding(vocab_size, embedding_length, input_length=max_
length))
model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))
```

Model: "sequential\_6"

Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, 20, 16)	160000
=====		
flatten (Flatten)	(None, 320)	0
=====		
dense_18 (Dense)	(None, 1)	321
=====		
Total params: 160,321		
Trainable params: 160,321		
Non-trainable params: 0		

- Training on the IMDB dataset



# Using pretrained embeddings

- Download the raw 20 newsgroups dataset  
(<http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20>).
  - 20,000 message board messages belonging to 20 different topic categories
- Preprocess the data by appending all files and vectorize it using Keras' `TextVectorization` layer
  - Lowercasing, punctuation removal, tokenization, indexing, integer encoding (or TFIDF)
- Download the GloVe embeddings trained on Wikipedia  
(<https://nlp.stanford.edu/projects/glove>) (800MB!)
- Build an embedding matrix of shape (max\_words, embedding\_dim) that we can load into an embedding layer

```
embedding_layer = layers.Embedding(  
    num_tokens, # 20000  
    embedding_dim, # 100  
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),  
    # GloVe embedding  
    trainable=False, # Freeze the pretrained weights  
)
```



# Conv1D model

- First layer produces the word embeddings
- Three 1D convolutional layers with MaxPooling
- One dense layer with dropout
- Output later (20 classes) with softmax activation

```
model = models.Sequential()  
model.add(embedding_layer)  
model.add(layers.Conv1D(128, 5, activation='relu'))  
model.add(layers.MaxPooling1D(5))  
model.add(layers.Conv1D(128, 5, activation='relu'))  
model.add(layers.MaxPooling1D(5))  
model.add(layers.Conv1D(128, 5, activation='relu'))  
model.add(layers.GlobalMaxPooling1D())  
model.add(layers.Dense(128, activation='relu'))  
model.add(layers.Dropout(0.5))  
model.add(layers.Dense(len(class_names), activation="softmax"))
```

## Training

## Using 1D convnets on IMDB data

- Let's try to learn a self-learned embedding with 1D convolutional layers
- A lot better than the previous self-trained embeddings. Worse than using the pretrained embeddings.
- Start overfitting after a few epochs.
- Try using the pretrained embeddings together with 1D convolutional layers :)

```
model = Sequential()  
model.add(layers.Embedding(10000, 128, input_length=100))  
model.add(layers.Conv1D(32, 7, activation='relu'))  
model.add(layers.MaxPooling1D(5))  
model.add(layers.Conv1D(32, 7, activation='relu'))  
model.add(layers.GlobalMaxPooling1D())  
model.add(layers.Dense(1))
```

## Results

# Summary

- Bag of words representations
  - Useful, but limited, since they destroy the order of the words in text
- Word embeddings
  - Learning word embeddings from labeled data is hard, you may need a lot of data
  - Pretrained word embeddings
    - Word2Vec: learns good embeddings and interesting relationships
    - FastText: can also compute embeddings for entirely new words
    - GloVe: also takes the global context of words into account
    - Language models: state-of-the-art, but expensive
- 1D convolutional nets
  - Allow us to use the sequence of the words in text, cheaper than RNNs
- In practice
  - Using pretrained embeddings gives great performance, even with little data
    - Embedding weights can be frozen or finetuned to the task at hand
  - 1D convnets can be used to learn decent embeddings for a given task