Lecture 10. Neural Networks for text

Turning text into numbers

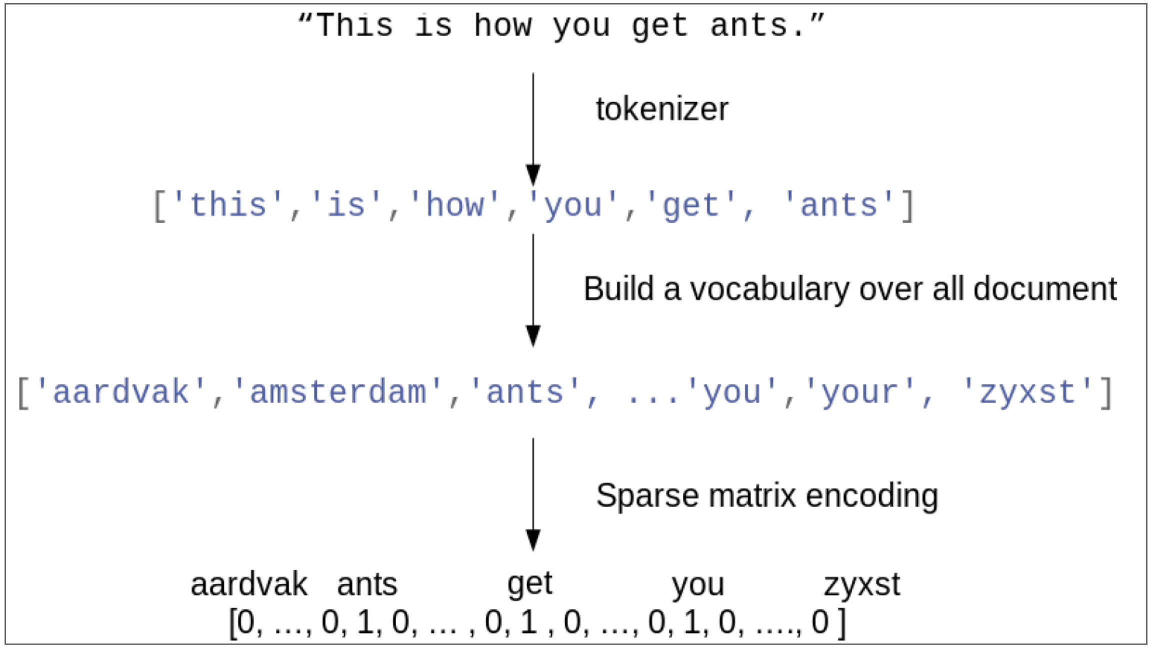
Joaquin Vanschoren

Overview

- Word embeddings
    - Word2Vec, FastText, GloVe
    - Neural networks on word embeddings
- Sequence-to-sequence models
    - Self-attention
    - Transformer models

Bag of word representation

- First, build a *vocabulary* of all occuring 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

```
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 a
n amazing actor and now the same being director norman's father came from the same scottish isl
and 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 dech
ifered 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 yea
rs with great films such as inside and switchblade romance bursting on to the scene maléfique p
receded the revival just slightly but stands head and shoulders over most modern horror titles
and is surely one
```

```
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 cullo
den 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 be
fore 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, 1
28, 64, 8, 58, 15302, 741, 32, 3712, 758, 58, 5763, 449, 9211, 1, 982, 4, 64314, 56, 163, 1, 10
2, 213, 1236, 38, 1794, 6, 12, 4, 32, 741, 2410, 28, 5, 1, 684, 20, 1, 33926, 7336, 3, 3690, 3
9, 35, 36, 118, 56, 453, 7, 7, 4, 262, 9, 572, 108, 150, 156, 56, 13, 1444, 18, 22, 583, 479, 3
6]

One-hot-encoded review:  [0. 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

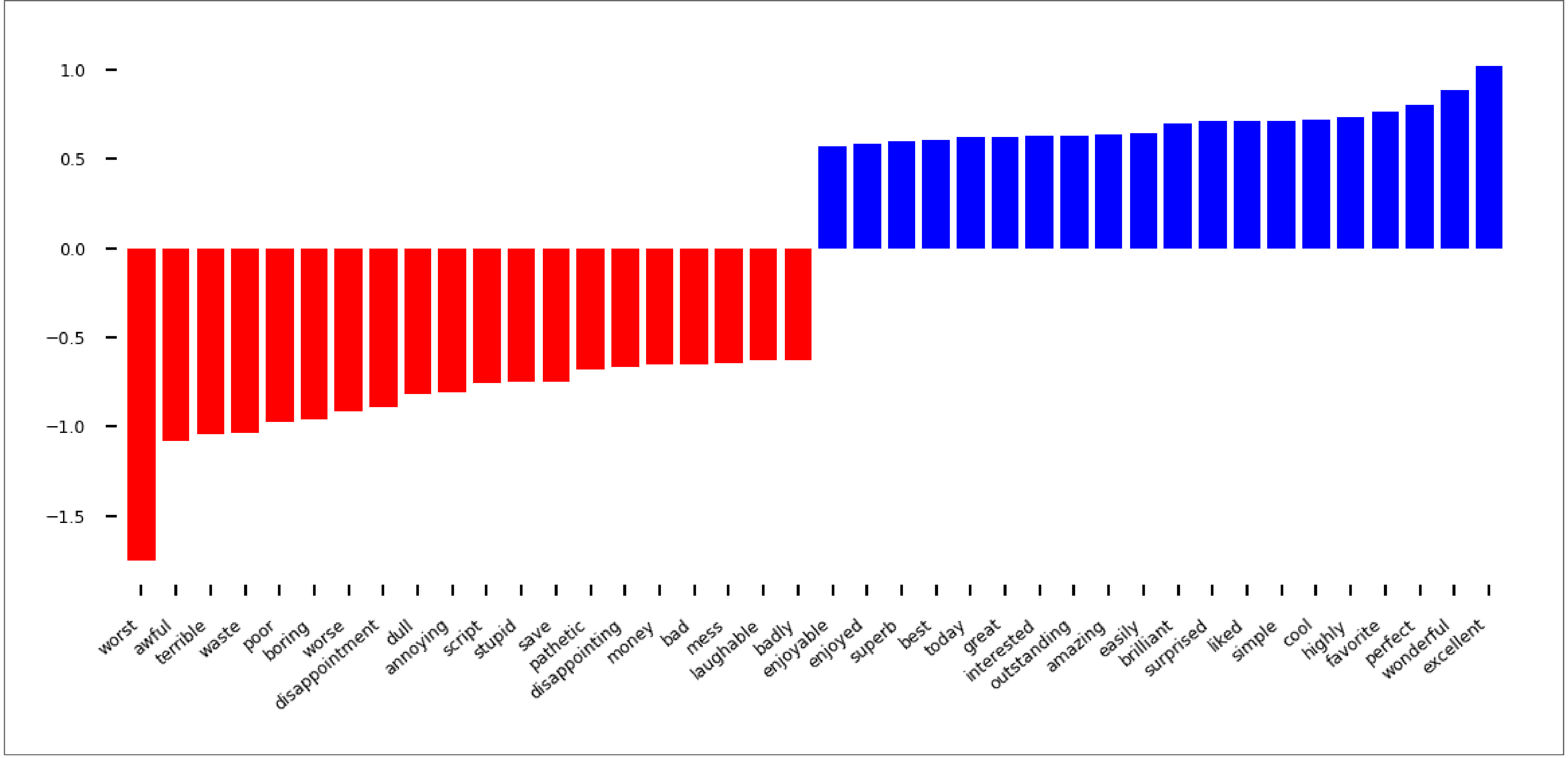
- Count the number of times each word appears in the document
- Example using sklearn `CountVectorizer` (on 2 documents)
- In practice, we also:
    - Preprocess the text (tokenization, stemming, remove stopwords, ...)
    - Use *n-grams* ("not terrible", "terrible acting",...), *character n-grams* ('ter', 'err', 'eri',...)
    - Scale the word-counts (e.g. L2 normalization or TF-IDF)

```
Vocabulary (feature names) after fit: ['actor' 'amazing' 'an' 'and' 'are' 'as' 'bad' 'be' 'bein
g' 'best' 'big'
 'boobs' 'brilliant' 'but' 'came' 'casting' 'cheesy' 'could' 'describe'
 'direction' 'director' 'ever' 'everyone' 'father' 'film' 'from' 'giant'
 'got' 'had' 'hair' 'horror' 'hundreds' 'imagine' 'is' 'island' 'just'
 'location' 'love' 'loved' 'made' 'movie' 'movies' 'music' 'myself'
 'norman' 'now' 'of' 'on' 'paper' 'part' 'pin' 'played' 'plot' 'really'
 'redford' 'ridiculous' 'robert' 'safety' 'same' 'scenery' 'scottish'
 'seen' 'so' 'story' 'suited' 'terrible' 'the' 'there' 'these' 'they'
 'thin' 'this' 'to' 've' 'was' '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 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 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
```



1.0
0.5
0.0
−0.5
−1.0
−1.5
worst
awful
terrible
waste
poor
boring
worse
disappointment
dull
annoying
script
stupid
save
pathetic
disappointing
money
bad
mess
laughable
badly
enjoyable
enjoyed
superb
best
today
great
interested
outstanding
amazing
easily
brilliant
surprised
liked
simple
cool
highly
favorite
perfect
wonderful
excellent

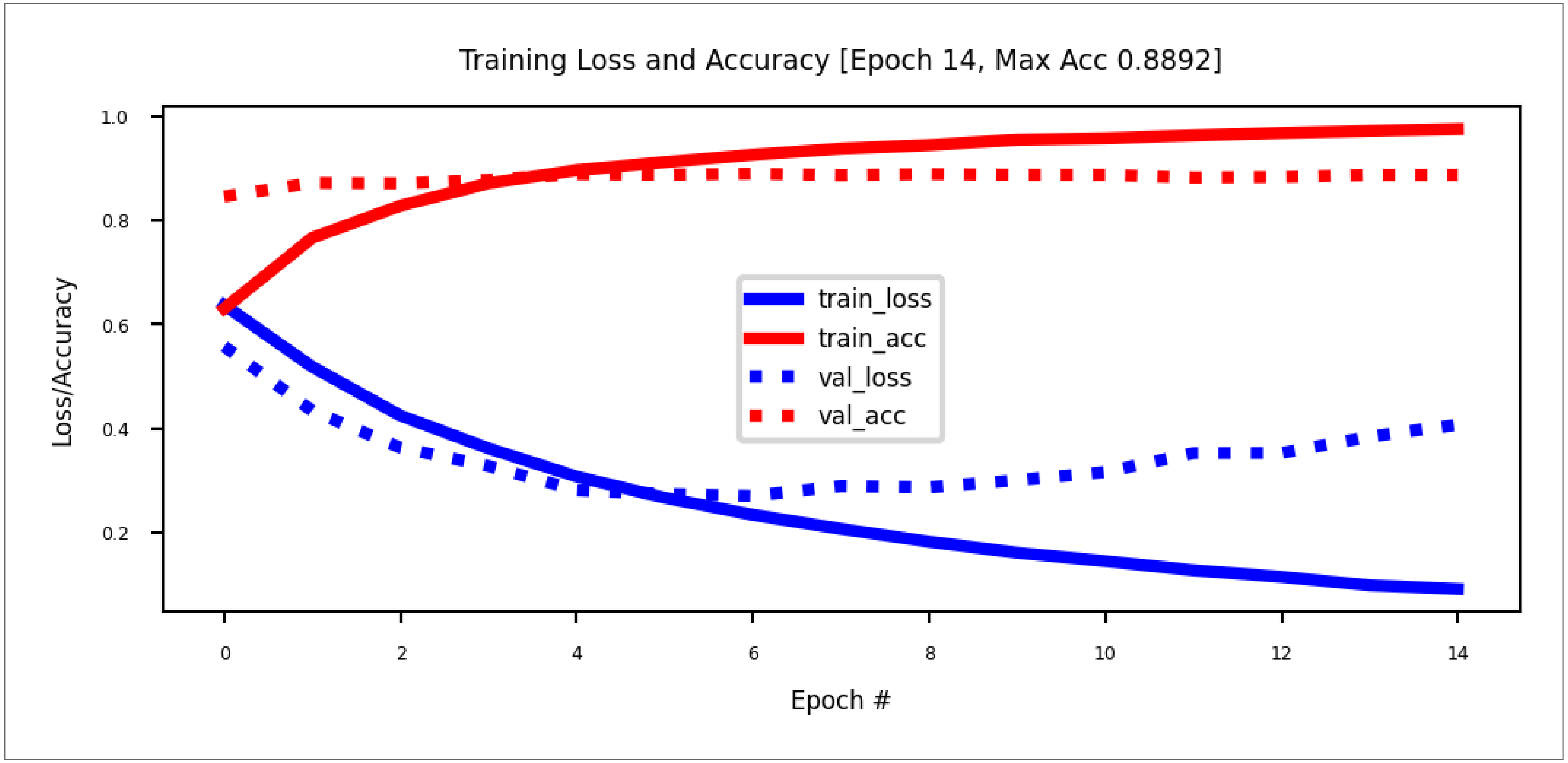
Neural networks on bag of words

- We can also build neural networks on bag-of-word vectors
    - E.g. Use one-hot-encoding with 10000 most frequent words
- Simple model with 2 dense layers, ReLU activation, dropout
    - Binary classification: single output node, sigmoid activation, binary cross-entropy loss

```
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

Evaluation

- 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



Predictions

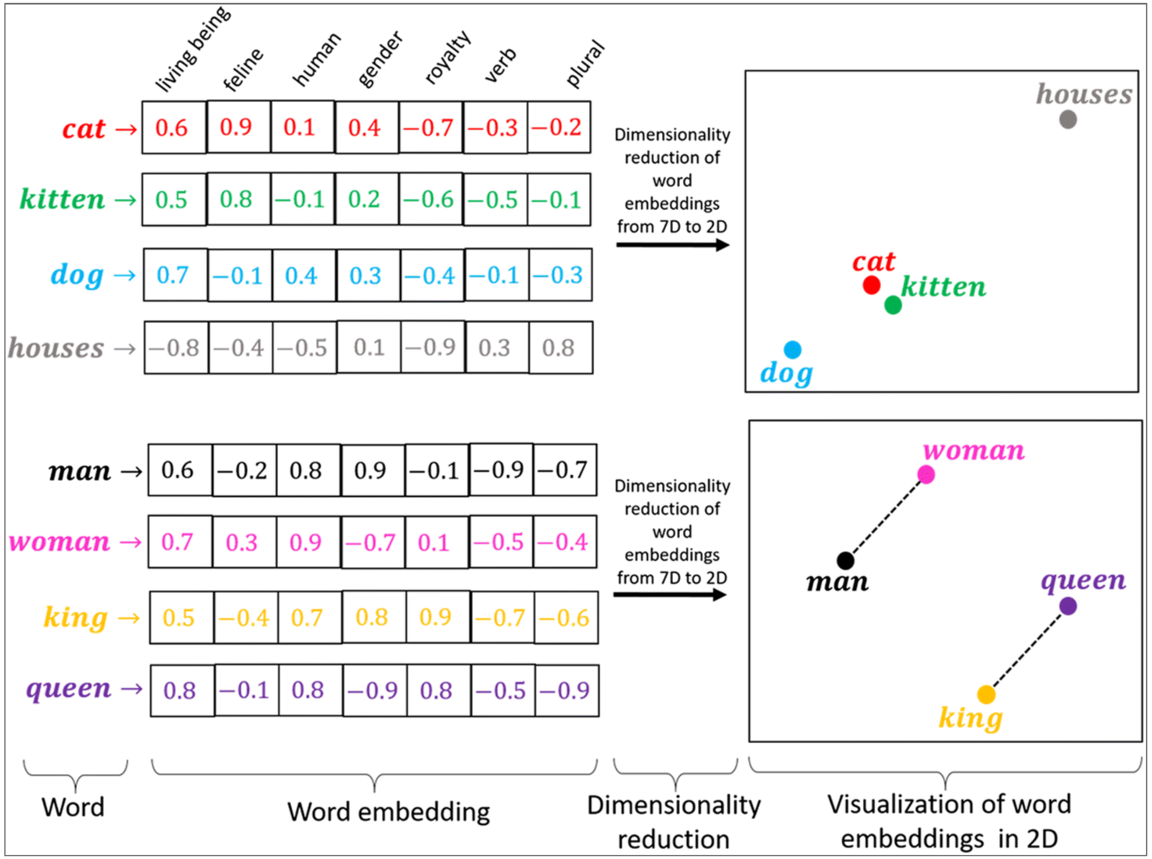
Let's look at a few predictions:

```
782/782 [==============================] - 2s 2ms/step
Review 0:  ? please give this one a miss br br ? ? and the rest of the cast rendered terrible p
erformances the show is flat flat flat br br i don't know how michael madison could have allowe
d this one on his plate he almost seemed to know this wasn't going to work out and his performa
nce was quite ? so all you madison fans give this a miss
Predicted positiveness:  [0.046]

Review 16:  ? from 1996 first i watched this movie i feel never reach the end of my satisfactio
n i feel that i want to watch more and more until now my god i don't believe it was ten years a
go 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 mu
ch made me come into another world deep in my heart anyone can feel what i feel and anyone coul
d make the movie like this i don't believe so thanks thanks
Predicted positiveness:  [0.956]
```

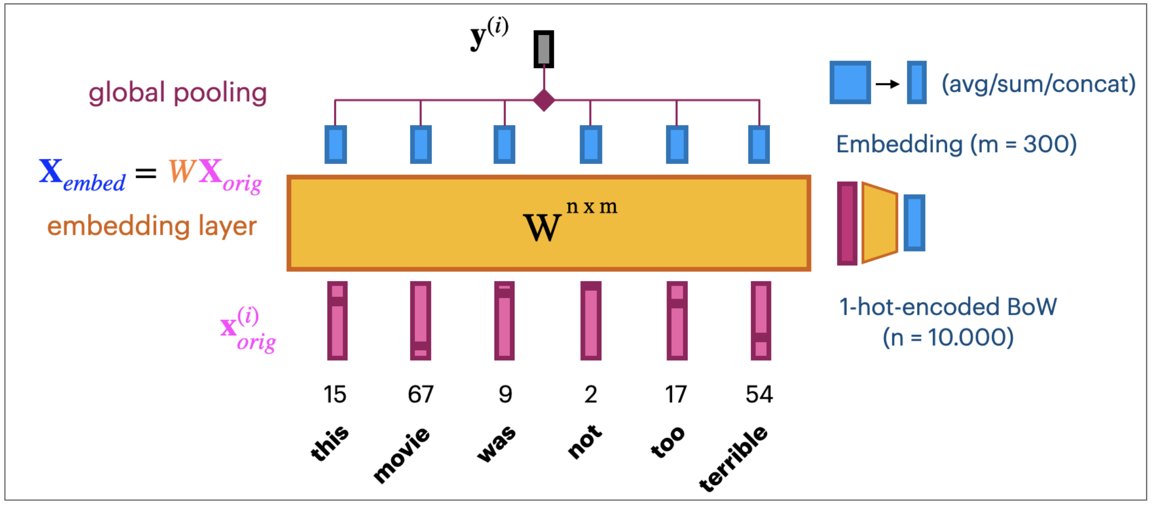
Word Embeddings

- A word embedding is a numeric vector representation of a word
  - Can be manual or *learned* from an existing representation (e.g. one-hot)



Learning embeddings from scratch

- Input layer uses fixed length documents (with 0-padding). 2D tensor (samples, max_length)
- Add an *embedding layer* to learn the embedding
    - Create n-dimensional one-hot encoding. Yields a 3D tensor (samples, max_length, n)
    - To learn an m-dimensional embedding, use m hidden nodes. Weight matrix W^{nxm}
    - Linear activation function: $\mathbf{X}_{embed} = W\mathbf{X}_{orig}$. 3D tensor (samples, max_length, m)
- Combine all word embeddings into a document embedding (e.g. global pooling).
- Add (optional) layers to map word embeddings to the output. Learn embedding weights from data.

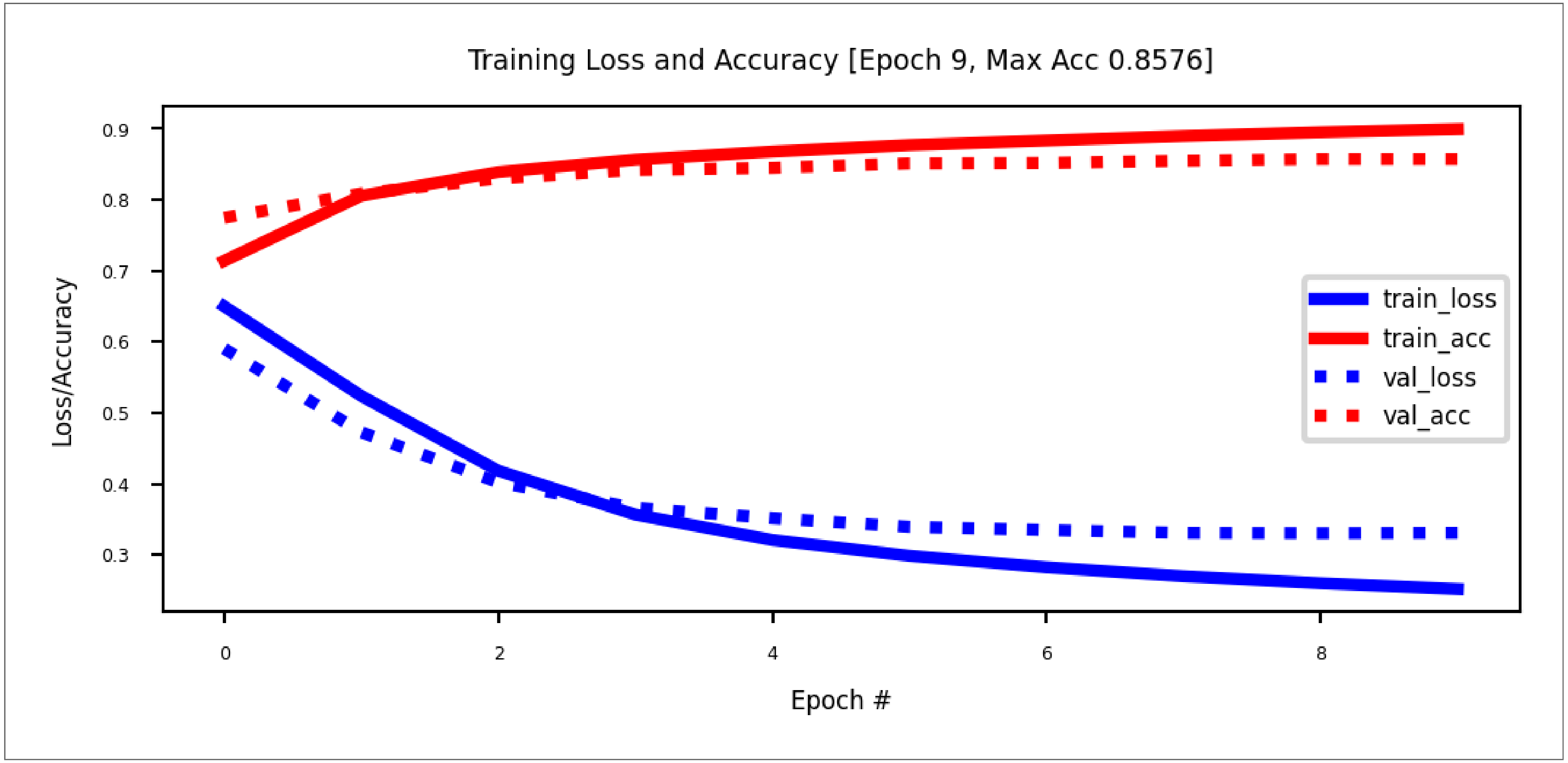


Let's try this:

```
max_length = 100 # pad documents to a maximum number of words
vocab_size = 10000 # vocabulary size
embedding_length = 20 # embedding length (more would be better)

model = models.Sequential()
model.add(layers.Embedding(vocab_size, embedding_length, input_length=max_length))
model.add(layers.GlobalAveragePooling1D())
model.add(layers.Dense(1, activation='sigmoid'))
```

- Training on the IMDB dataset: slightly worse than using bag-of-words?
  - Embedding of dim 20 is very small, should be closer to 100 (or 300)
  - We don't have enough data to learn a really good embedding from scratch

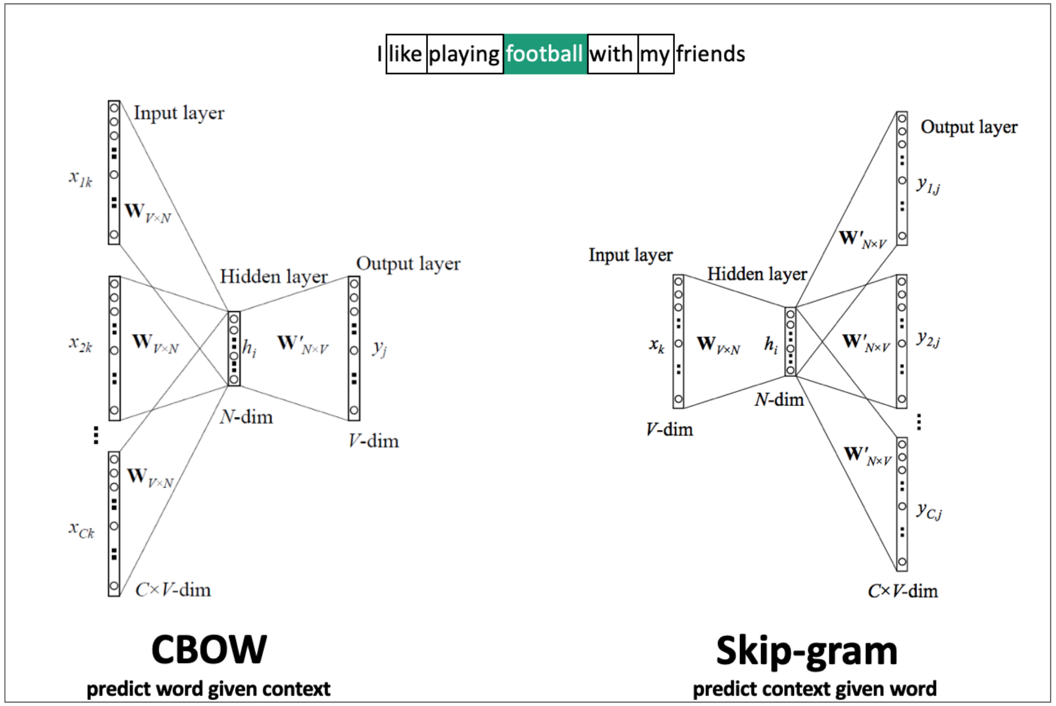


Pre-trained embeddings

- 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.
    - Also called self-supervised learning. Supervision is provided by data itself
- Freeze embedding weights to produce simple word embeddings, or finetune to a new tasks
- Most common approaches:
    - Word2Vec: Learn neural embedding for a word based on surrounding words
    - FastText: learns embedding for character n-grams
        - Can also produce embeddings for new, unseen words
    - GloVe (Global Vector): Count co-occurrences of words in a matrix
        - Use a low-rank approximation to get a latent vector representation

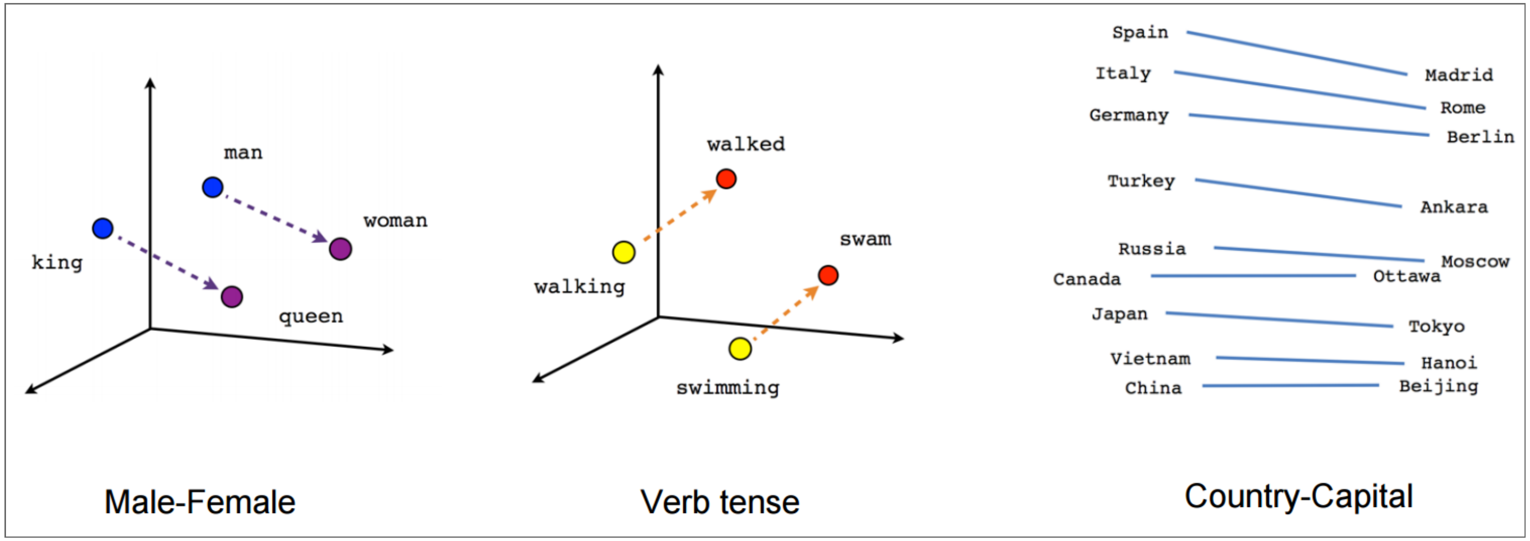
WORD2VEC

- Move a window over text to get C context words (V-dim one-hot encoded)
- Add embedding layer with N linear nodes, global average pooling, and softmax layer(s)
- CBOW: predict word given context, use weights of last layer W'_{NxV} as embedding
- Skip-Gram: predict context given word, use weights of first layer W^T_{VxN} as embedding
    - Scales to larger text corpora, learns relationships between words better



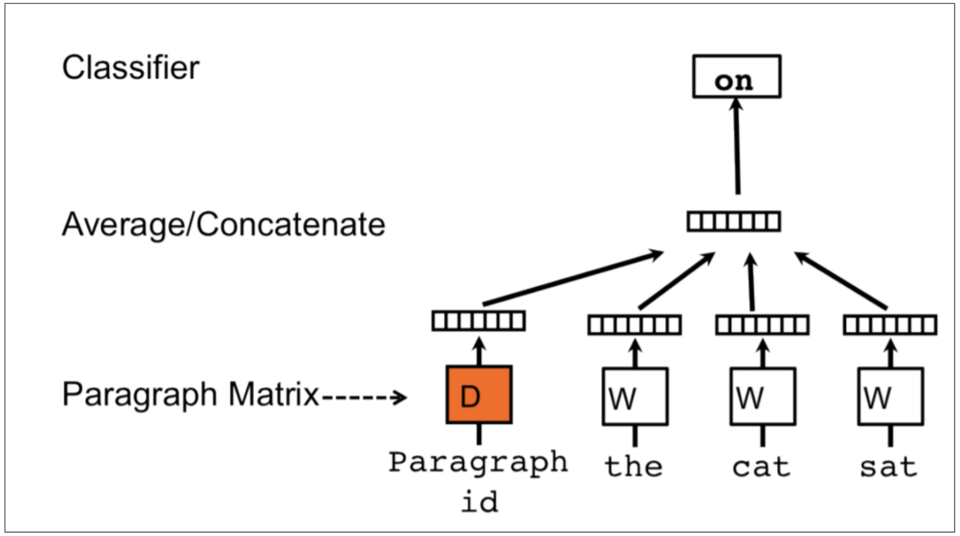
Word2Vec properties

- Word2Vec happens to learn **interesting relationships** 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** present in the data.
  - Important unsolved problem!



Doc2Vec

- Alternative way to combine word embeddings (instead of global pooling)
- Adds a paragraph (or document) embedding: learns how paragraphs (or docs) relate to each other
    - Captures document-level semantics: context and meaning of entire document
- Can be used to determine semantic similarity between documents.



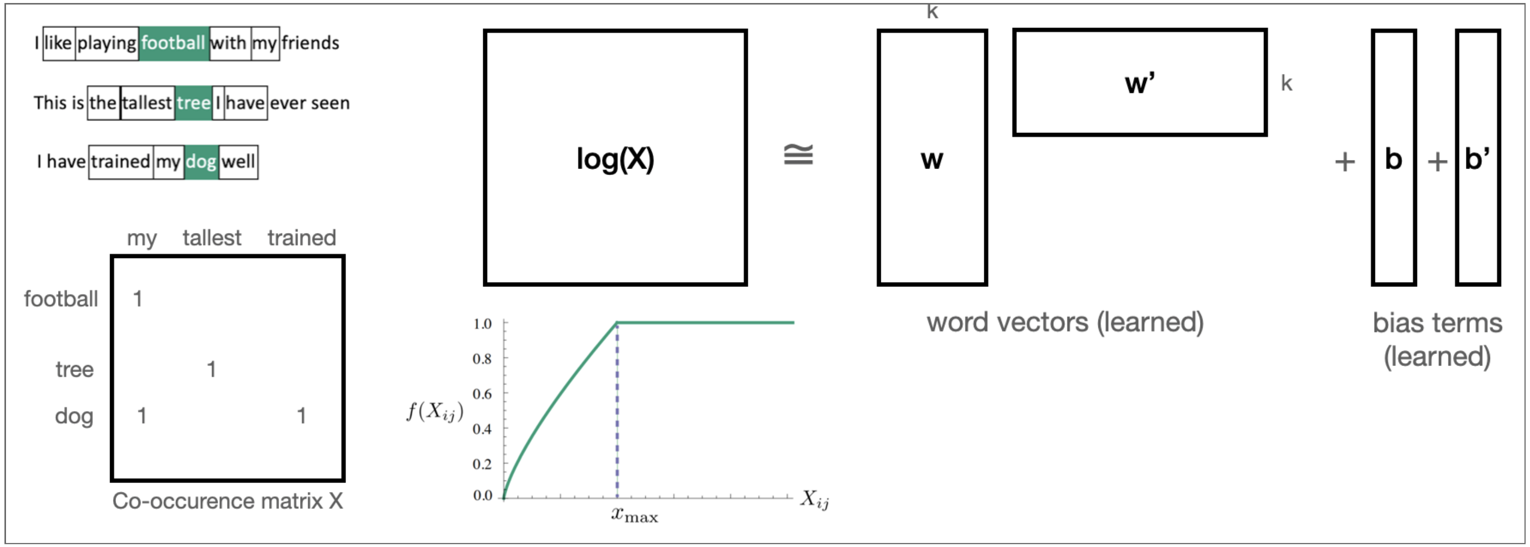
FastText

- Limitations of Word2Vec:
  - Cannot represent new (out-of-vocabulary) words
  - Similar words are learned independently: less efficient (no parameter sharing)
    - E.g. 'meet' and 'meeting'
- FastText: same model, but uses *character n-grams*
  - 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 = bin size)
  - Representation of word "football" is sum of its n-gram embeddings
- Negative sampling: also trains on random negative examples (out-of-context words)
  - Weights are updated so that they are *less* likely to be predicted

Global Vector model (GloVe)

- Builds a co-occurence matrix $\mathbf{X}$: counts how often 2 words occur in the same context
- 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

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



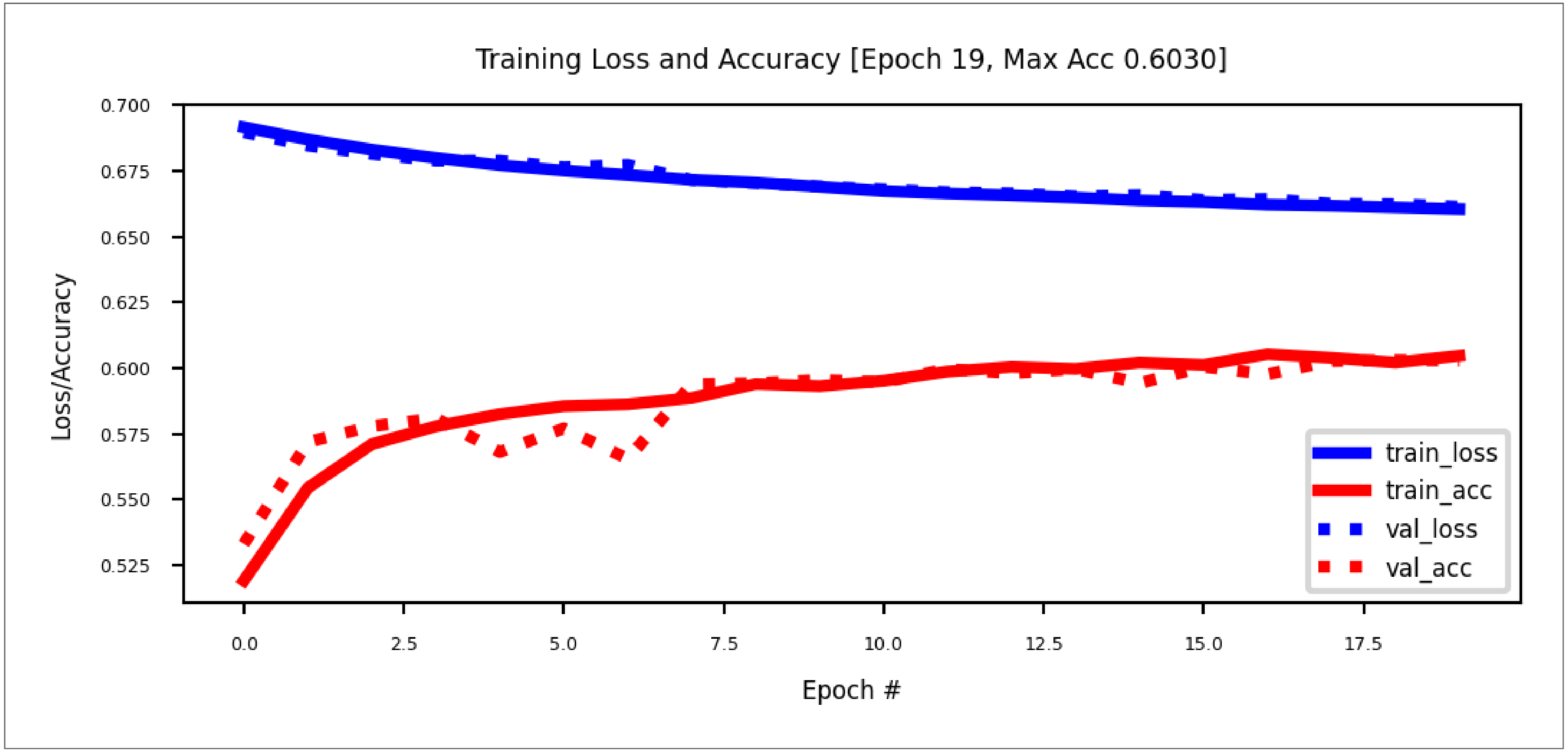
Let's try this

- Download the **GloVe embeddings trained on Wikipedia**
- We can now get embeddings for 400,000 English words
- E.g. 'queen' (in 100-dim):

```
array([-0.5  , -0.708,  0.554,  0.673,  0.225,  0.603, -0.262,  0.739,
       -0.654, -0.216, -0.338,  0.245, -0.515,  0.857, -0.372, -0.588,
        0.306, -0.307, -0.219,  0.784, -0.619, -0.549,  0.431, -0.027,
        0.976,  0.462,  0.115, -0.998,  1.066, -0.208,  0.532,  0.409,
        1.041,  0.249,  0.187,  0.415, -0.954,  0.368, -0.379, -0.68 ,
       -0.146, -0.201,  0.171, -0.557,  0.719,  0.07 , -0.236,  0.495,
        1.158, -0.051,  0.257, -0.091,  1.266,  1.105, -0.516, -2.003,
       -0.648,  0.164,  0.329,  0.048,  0.19 ,  0.661,  0.081,  0.336,
        0.228,  0.146, -0.51 ,  0.638,  0.473, -0.328,  0.084, -0.785,
        0.099,  0.039,  0.279,  0.117,  0.579,  0.044, -0.16 , -0.353,
       -0.049, -0.325,  1.498,  0.581, -1.132, -0.607, -0.375, -1.181,
        0.801, -0.5  , -0.166, -0.706,  0.43 ,  0.511, -0.803, -0.666,
       -0.637, -0.36 ,  0.133, -0.561], dtype=float32)
```

Same simple model, but with frozen GloVe embeddings: much worse!

```
embedding_layer = layers.Embedding(input_dim=10000, output_dim=100,
                                   input_length=max_length, trainable=False)
embedding_layer.set_weights([weights]) # set pre-trained weigths
model = models.Sequential([
    embedding_layer, layers.GlobalAveragePooling1D(),
    layers.Dense(1, activation='sigmoid')]
```

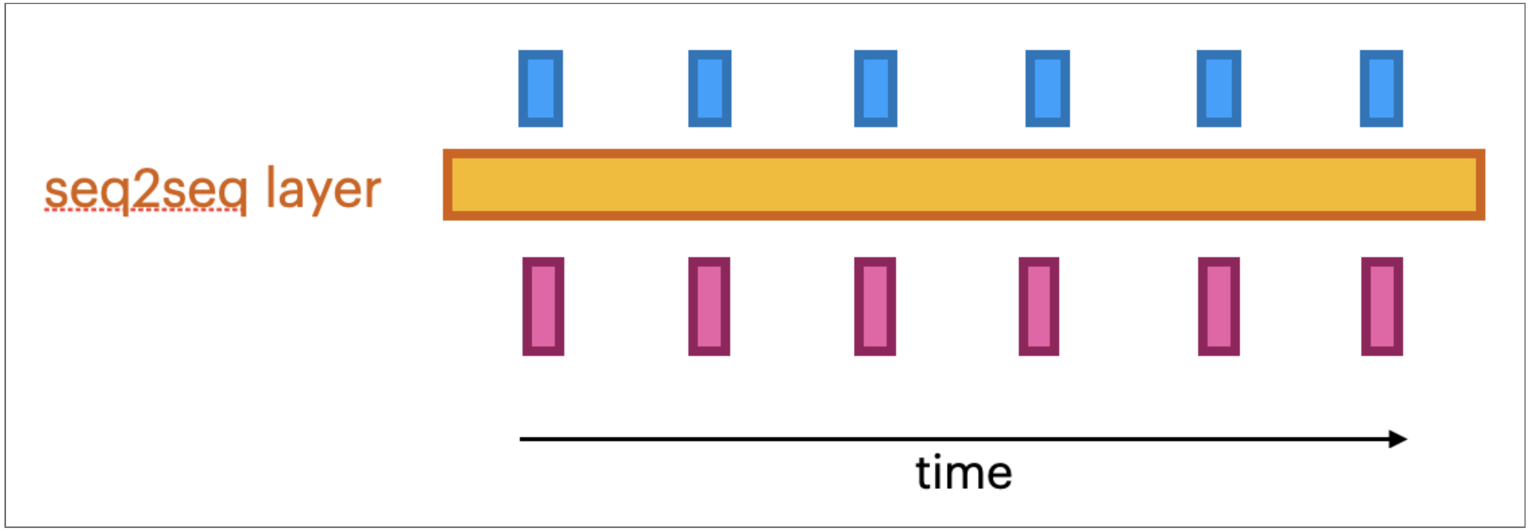


Sequence-to-sequence (seq2seq) models

- Global average pooling or flattening destroys the word order
- We need to model sequences explictly, e.g.:
    - 1D convolutional models: run a 1D filter over the input data
        - Fast, but can only look at small part of the sentence
    - Recurrent neural networks (RNNs)
        - Can look back at the entire previous sequence
        - Much slower to train, have limited memory in practice
    - Attention-based networks (Transformers)
        - Best of both worlds: fast and very long memory

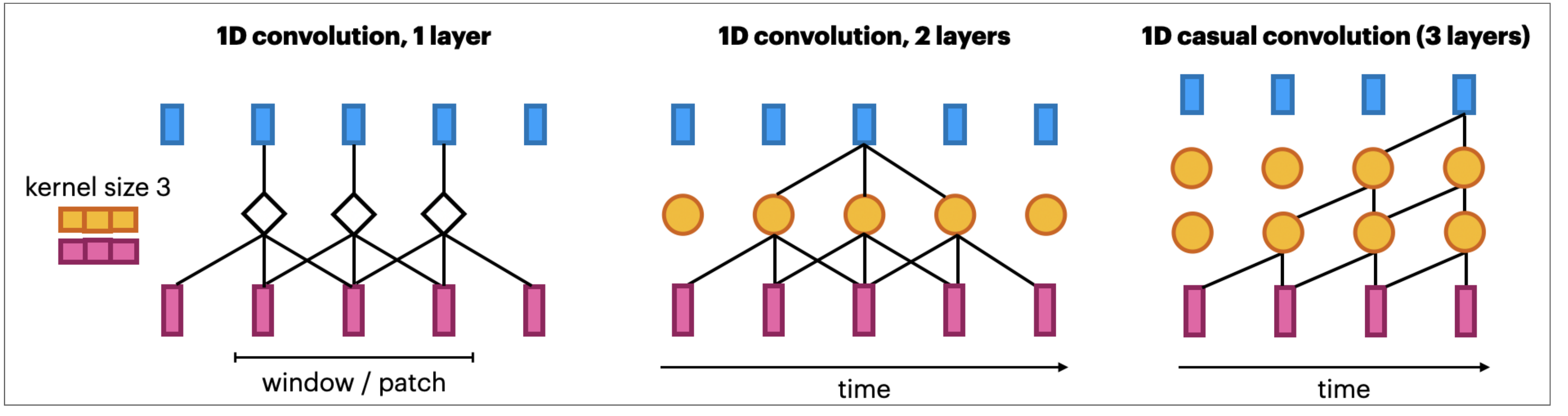
seq2seq models

- Produce a series of output given a series of inputs over time
- Can handle sequences of different lengths
    - Label-to-sequence, Sequence-to-label, seq2seq,...
    - Autoregressive models (e.g. predict the next character, unsupervised)



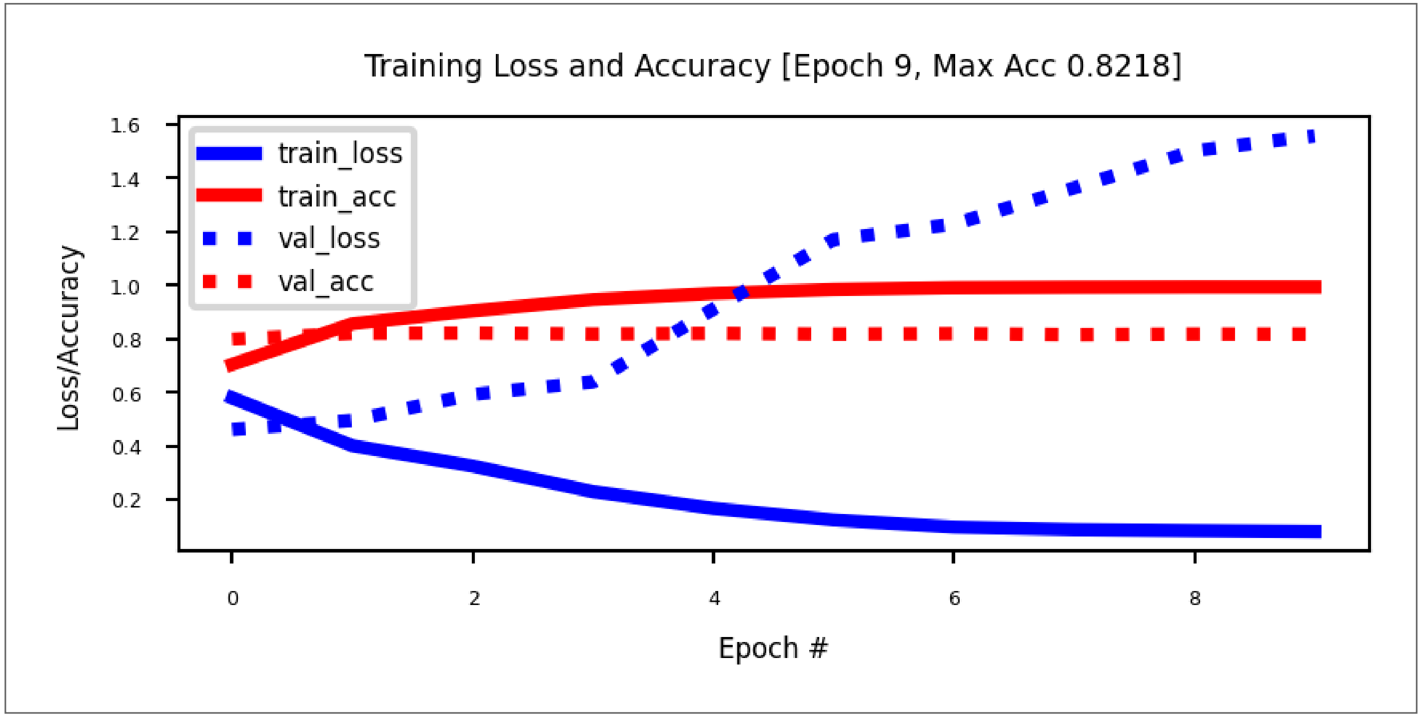
1D convolutional networks

- Similar to 2D convnets, but moves only in 1 direction (time)
    - Extract local 1D patch, apply filter (kernel) to every patch
    - Pattern learned can later be recognized elsewhere (translation invariance)
- Limited memory: only sees a small part of the sequence (receptive field)
    - You can use multiple layers, dilations,... but becomes expensive
- Looks at 'future' parts of the series, but can be made to look only at the past
    - Known as 'causal' models (not related to causality)



- Same embedding, but add 2 `Conv1D` layers and `MaxPooling1D`. Better!

```
model = models.Sequential([
        embedding_layer,
        layers.Conv1D(32, 7, activation='relu'),
        layers.MaxPooling1D(5),
        layers.Conv1D(32, 7, activation='relu'),
        layers.GlobalAveragePooling1D(),
        layers.Dense(1, activation='sigmoid')]
```



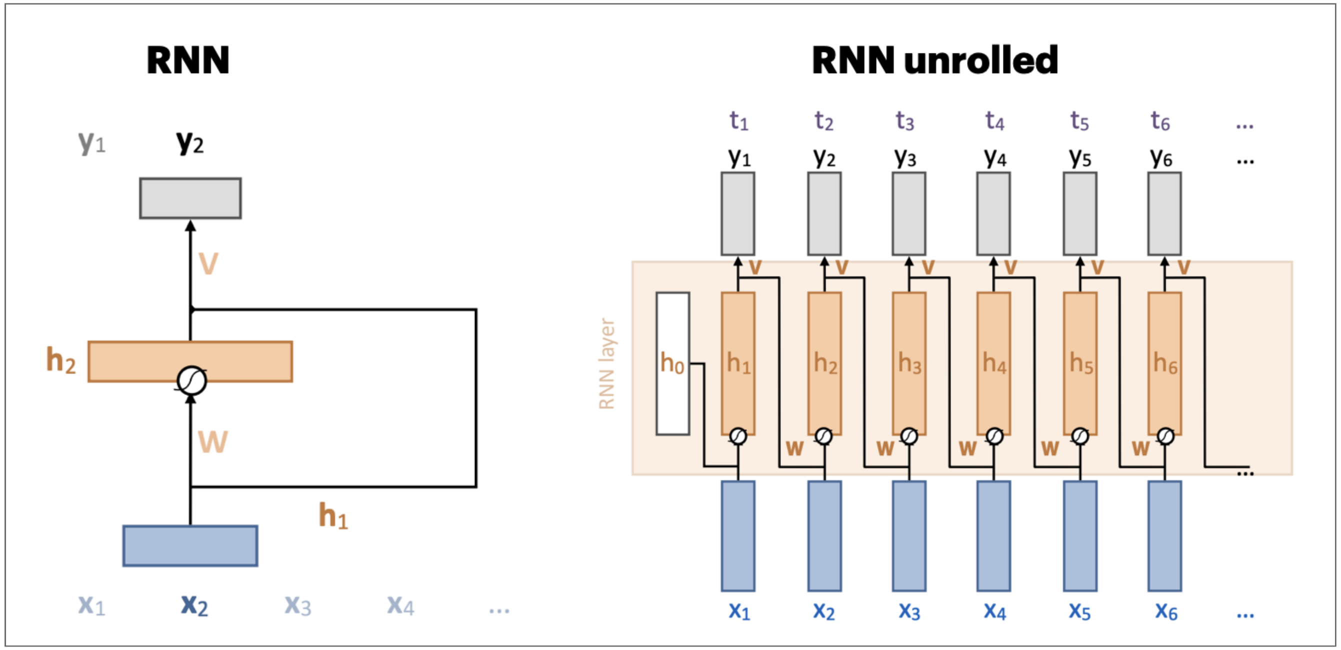

```
157/157 [==============================] - 3s 18ms/step - loss: 0.0807 - accuracy: 0.9932 - val
_loss: 1.5570 - val_accuracy: 0.8172
```

Recurrent neural networks (RNNs)

- Adds a recurrent connection that concats previous output to next input
  - Hidden layer

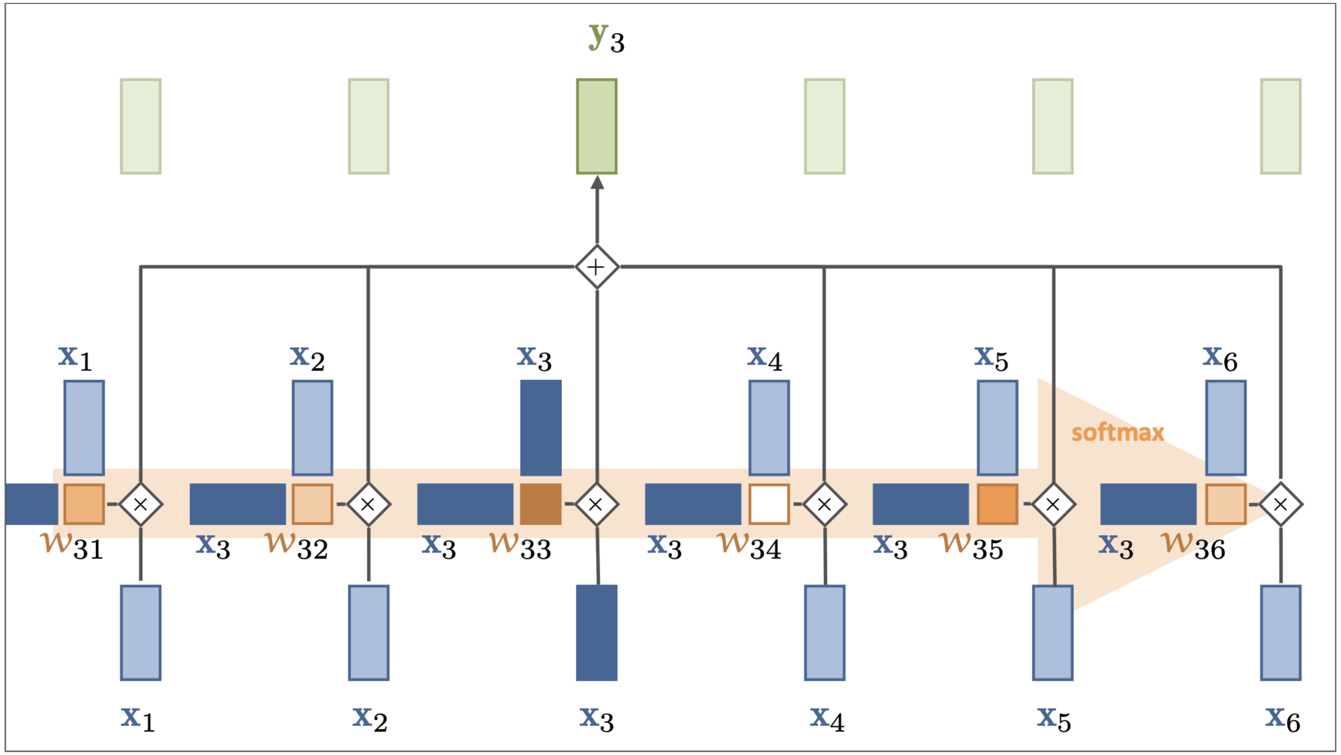
$$h_t = \sigma\left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b\right)$$

- Unbounded memory, but training requires *backpropagation through time*
  - Requires storing previous network states (slow + lots of memory)
  - Vanishing gradients strongly limit practical memory
- Improved with *gating*: learn what to input, forget, output (LSTMs, GRUs,...)



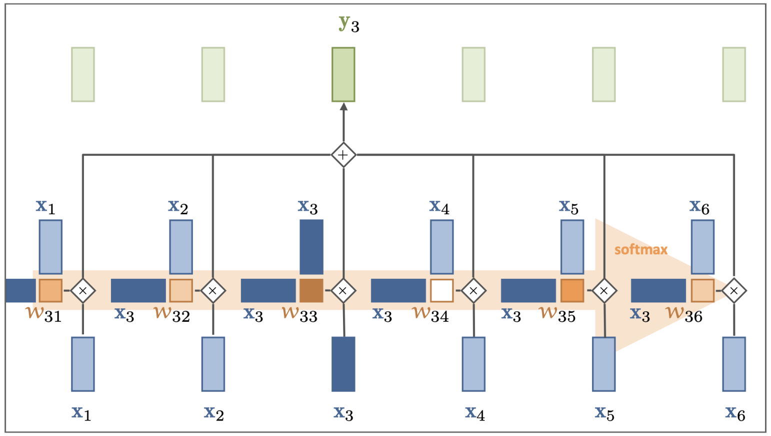
Simple self-attention

- Compute dot product of input vector x_i with every x_j (including itself): w_{ij}
- Compute softmax over all these weights (positive, sum to 1)
- Multiply by each input vector, and sum everything up
- Can be easily vectorized: $Y^T = WX^T$, $W = \text{softmax}(X^T X)$



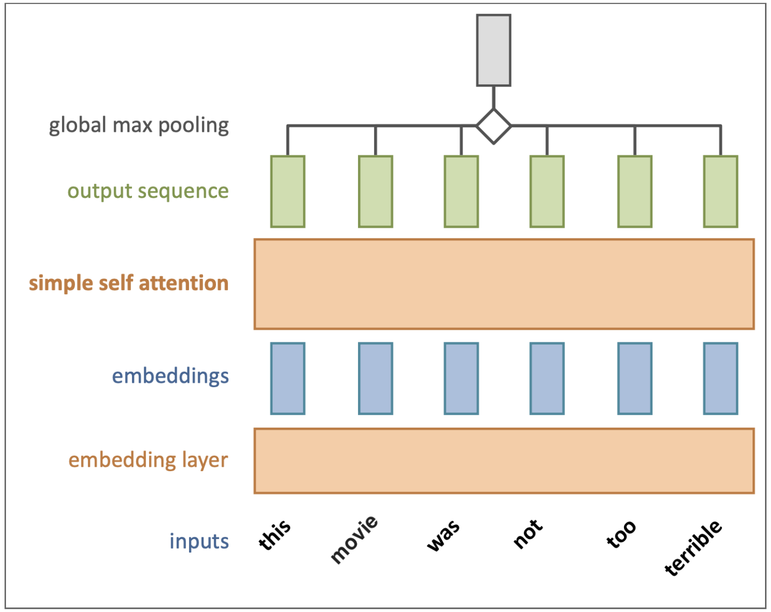
Simple self-attention (2)

- Output is mostly influenced by the current input (w_{ii} is largest)
    - Mixes in information from other inputs according to how *similar* they are
- Doesn't learn (no parameters), the embedding of X defines self-attention
- $Y^T = WX^T$ is *linear*, vanishing gradients only through softmax
- Has no problem looking *very* far back in the sequence
- Operates on *sets* (permutation invariant): allows img-to-set, set-to-set,... tasks
    - No access to sequence. For seq tasks we encode sequence in embedding



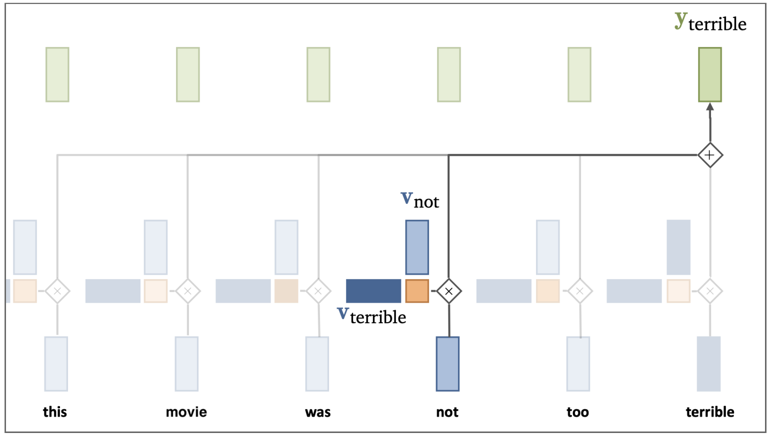
Simple self-attention layer

- Let's add a simple self-attention layer to our movie sentiment model
- Without self-attention, every word would contribute independently from others
    - Exactly as in a bag-of-words model
    - The word *terrible* will likely result in a negative prediction



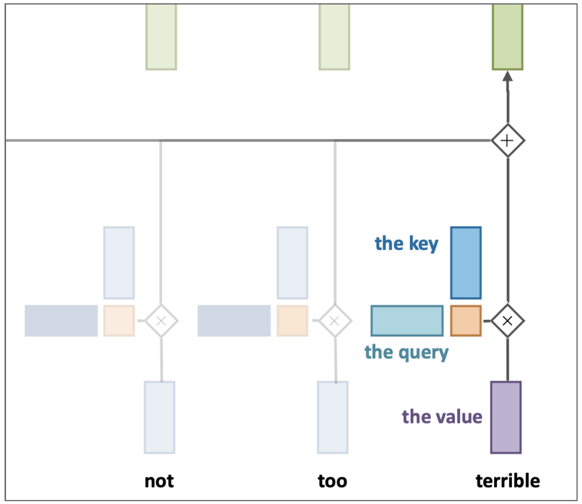
Simple self-attention layer

- Self-attention can learn that the meaning on the word *terrible* is inverted by the presence of the word *not*, even if it is further away in the sequence.
- In general, each self attention layer can learn specific relationships between words (e.g. inversion). We'll need many of them.



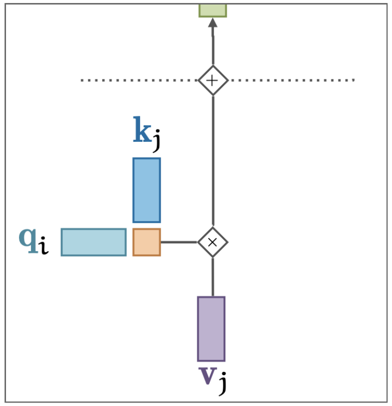
Standard self-attention

- Inputs occur in one of 3 positions in the self-attention layer:
  - Value v: input vector that provides the output, weighted by:
  - Query q: the vector that corresponds to the wanted *output*
  - Key k: The vector that the query is matched aganinst
- Works as a soft version of a dictionary, in which:
  - Every key matches the query to some extent (w.r.t. its dot product)
  - A weighted mixture of all values (normalized by softmax)



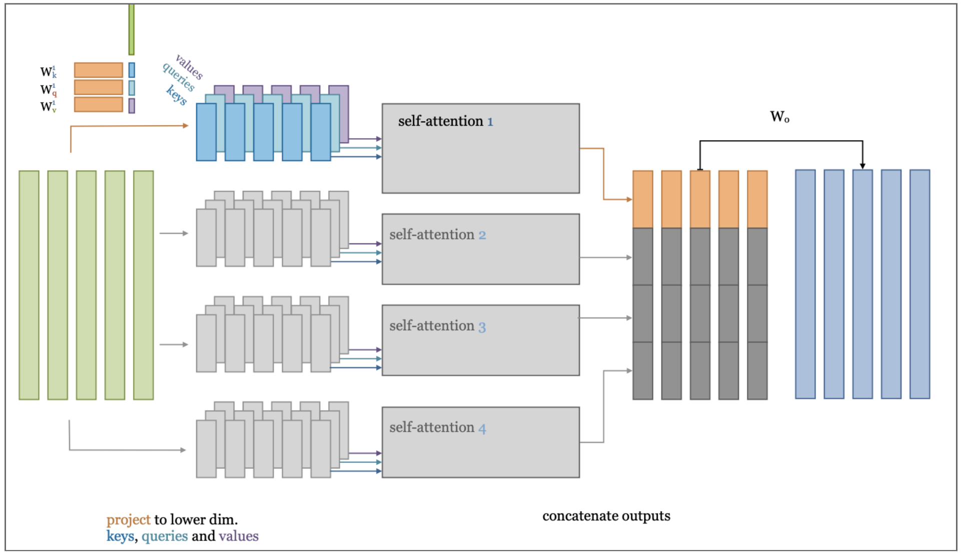
Standard self-attention

- We want to *learn* how each of these interact by adding learned transformations
  - $k_i = Kx_i + b_k$
  - $q_i = Qx_i + b_q$
  - $v_i = Vx_i + b_v$
- Makes self-attention more flexible, learnable
- Learn what to pay attention to in the input (e.g. sequence, image,...)



Standard self-attention (2)

- Inputs can have multiple relationships with each other (e.g. negation, strengtening,...)
- To learn these in parallel, we can split the self-attention in multiple heads
- Input vector is embedded (linearly) into lower dimensionality, multiple times

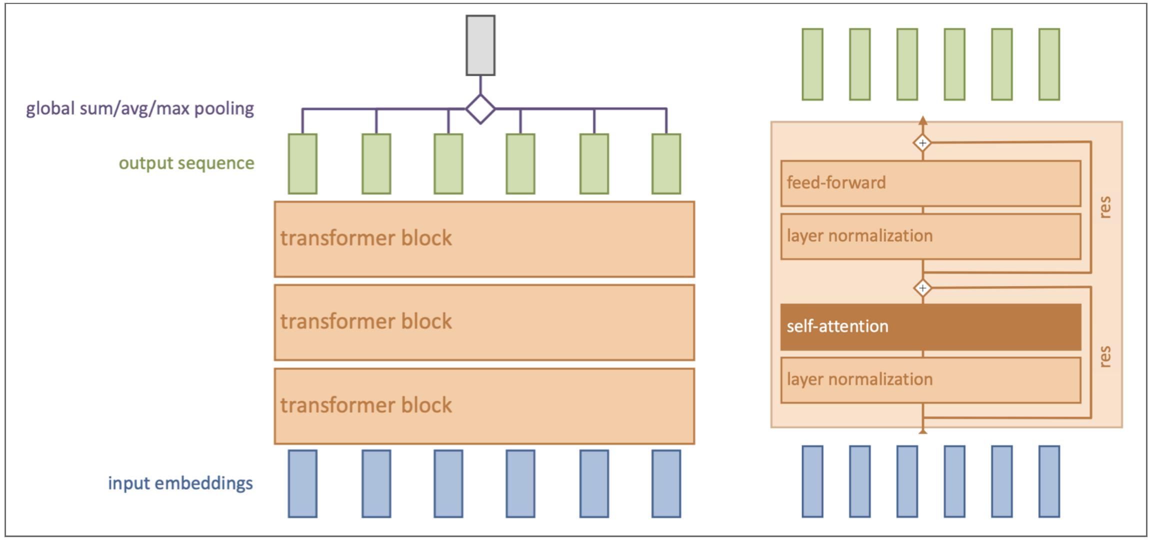


Standard self-attention (3)

- The softmax operation can still lead to vanishing gradients (unless values are small)
    - We can scale the dot product by the input dimension k: $w^{'}_{ij} = \frac{{x_i}^T x_j}{\sqrt{k}}$

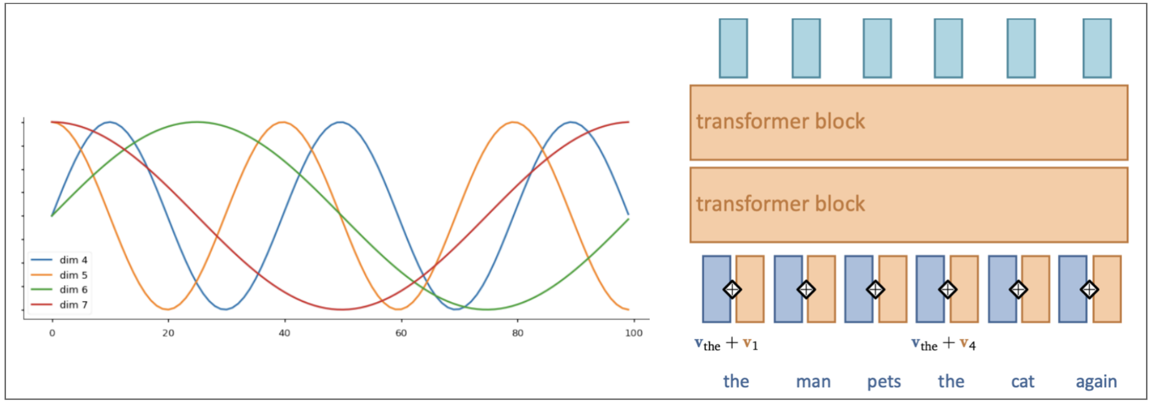
Transformer model

- Repeat self-attention multiple times in controlled fashion
- Works for sequences, images, graphs,... (learn how sets of objects interact)
- Models consist of multiple transformer blocks, usually:
    - Layer normalization (every input is normalized independently)
    - Self-attention layer (learn interactions)
    - Residual connections (preserve gradients in deep networks)
    - Feed-forward layer (learn mappings)



Positional encoding

- We need some way to tell the self-attention layer about position in sequencet
- Represent position by vectors, using some easy-to-learn predictable pattern
  - Add these encodings to vector embeddings
  - Give information on how far one input is fron the others



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
- Sequence-to-sequence models
    - 1D convolutional nets (fast, limited memory)
    - RNNs (slow, also quite limited memory)
    - Self-attention (allows very large memory)
- Transformers

Acknowledgement
Several figures came from the excellent **VU Deep Learning course**