Beyond Counting Words: Working with Word Embeddings

Damian Trilling

d.c.trilling@uva.nl
@damian0604
www.damiantrilling.net

12–13 April 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

This part: Keras

Neural networks

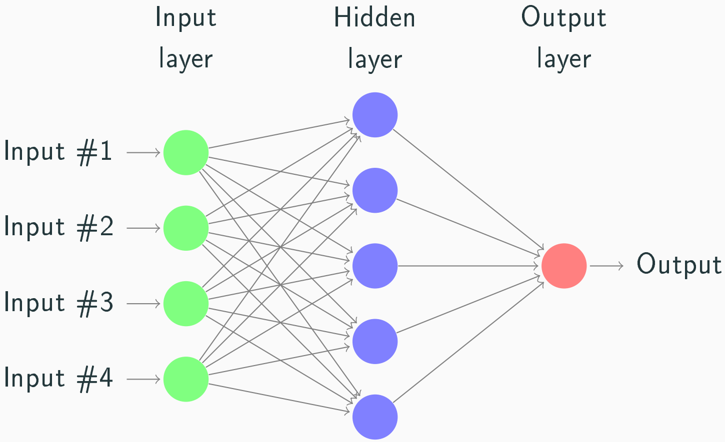
Using pretrained embeddings

Disclaimer: I cannot give a full overview of the whole topic of deep learning here – that's a whole (extensive) course in itself. But embeddings are closely related, that's why we at least will at least get out feet wet a bit.

Neural networks

Neural Networks

- In "classical" machine learning, we predict an outcome directly based on the input features
- In neural networks, we can have "hidden layers" that we predict
- These layers are not necessarily interpretable
- "Neurons" that "fire" based on an "activation function"



⇒ If we had multiple hidden layers in a row, we'd call it a *deep* network.

Why neural networks?

- learn hidden structures (e.g., embeddings (!))
- go beyond the idea that there is a direct relationship between occurrence of word X and label (or occurrence of pixel [R,G,B] and a label)
- images, machine translation — and more and more general NLP, sentiment analysis, etc.

Example of a comparatively easy introduction:
https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

Simple feed forward network

```
model.add(Dense(300, input_dim=input_dim, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Our first layer reduces the input features (e.g., the 10,000 features our CountVectorizer creates) to 300 neurons
- It does so using the relu function $f(x) = max(0, x)$ (as our counts cannot be negartive, just a linear function)
- The second layer reduces the 300 neurons to 1 output neuron using the sigmoid function (the S curve you know fron logistic regression)
- Of course, we can add multiple layers in between if we want to

Simple feed forward network

```
model.add(Dense(300, input_dim=input_dim, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Our first layer reduces the input features (e.g., the 10,000 features our CountVectorizer creates) to 300 neurons
- It does so using the relu function $f(x) = max(0, x)$ (as our counts cannot be negartive, just a linear function)
- The second layer reduces the 300 neurons to 1 output neuron using the sigmoid function (the S curve you know fron logistic regression)
- Of course, we can add multiple layers in between if we want to

Simple feed forward network

```
model.add(Dense(300, input_dim=input_dim, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Our first layer reduces the input features (e.g., the 10,000 features our CountVectorizer creates) to 300 neurons
- It does so using the relu function $f(x) = max(0, x)$ (as our counts cannot be negartive, just a linear function)
- The second layer reduces the 300 neurons to 1 output neuron using the sigmoid function (the S curve you know fron logistic regression)
- Of course, we can add multiple layers in between if we want to

Simple feed forward network

```
model.add(Dense(300, input_dim=input_dim, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- Our first layer reduces the input features (e.g., the 10,000 features our CountVectorizer creates) to 300 neurons
- It does so using the relu function $f(x) = max(0, x)$ (as our counts cannot be negartive, just a linear function)
- The second layer reduces the 300 neurons to 1 output neuron using the sigmoid function (the S curve you know fron logistic regression)
- Of course, we can add multiple layers in between if we want to

Convolutional networks

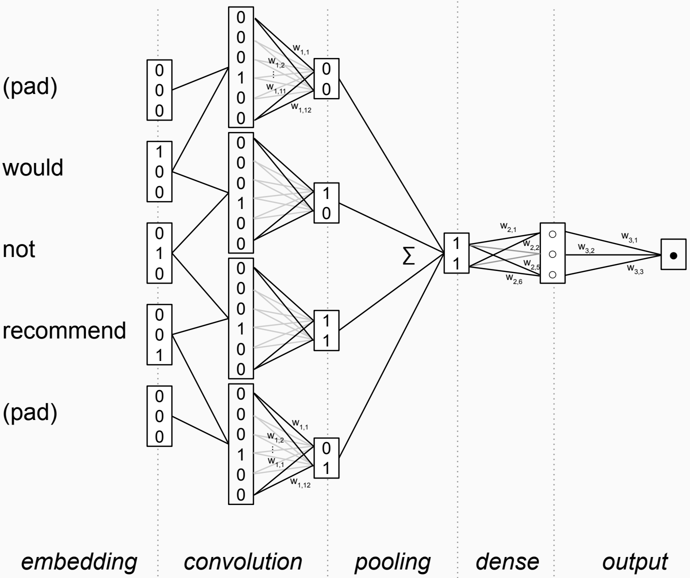
The problem with such a basic networks: just as with classic SML, we still loose all information about order (the “not good” problem).

Therefore,

- We concatenate the vectors of neighboring words
- We apply some filter (essentially, we detect patterns)
- and then pool the results (e.g., taking the maximum)

This means that we now excplitly take into acount *the temporal structure* of a sentence.

Convolutional networks



Convolutional networks

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
    input_length=maxlen))
model.add(Conv1D(embedding_dim, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The layers:

1. train an embedding model
2. apply the convolution with 5 “timestamps”
3. pool using the maximum
4. another layer with 300 dimensions
5. the final layer with 1 output neuron

Convolutional networks

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
    input_length=maxlen))
model.add(Conv1D(embedding_dim, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The layers:

1. train an embedding model
2. apply the convolution with 5 “timestamps”
3. pool using the maximum
4. another layer with 300 dimensions
5. the final layer with 1 output neuron

Convolutional networks

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
    input_length=maxlen))
model.add(Conv1D(embedding_dim, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The layers:

1. train an embedding model
2. apply the convolution with 5 “timestamps”
3. pool using the maximum
4. another layer with 300 dimensions
5. the final layer with 1 output neuron

Convolutional networks

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
    input_length=maxlen))
model.add(Conv1D(embedding_dim, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The layers:

1. train an embedding model
2. apply the convolution with 5 “timestamps”
3. pool using the maximum
4. another layer with 300 dimensions
5. the final layer with 1 output neuron

Convolutional networks

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim,
    input_length=maxlen))
model.add(Conv1D(embedding_dim, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(300, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

The layers:

1. train an embedding model
2. apply the convolution with 5 “timestamps”
3. pool using the maximum
4. another layer with 300 dimensions
5. the final layer with 1 output neuron

Convolutional networks

Note that the preprocessing differs!

- We do not take a word vector per document as input any more, but *a sequence of words*
- For concatenating, these sequences need to have equal length, which is why we *pad* then

LSTM (long short-term memory)

- Unlike “feed forward” neural networks, this is a “recurrent neural network” (RNN) – the training works in two directions
- Heavy in computation, very useful for predicting *sequences*
- Won’t cover today

Using pretrained embeddings

The embedding layer

- Often, the first layer is creating word embeddings
- Good embeddings need a lot of training data
- Training good embeddings needs time
- Therefore, we can replace that layer with a pre-trained embedding layer (!)
- We can even use a hybrid approach and allow the pre-trained embedding layer to be re-trained!