Big Data and Automated Content Analysis (12EC)
Week 9: »Transformers«
Friday

Felicia Loecherbach
f.loecherbach@uva.nl

April 12, 2023

UvA RM Communication Science

Today

Neural Networks and Deep Learning

- Neural Networks in Keras

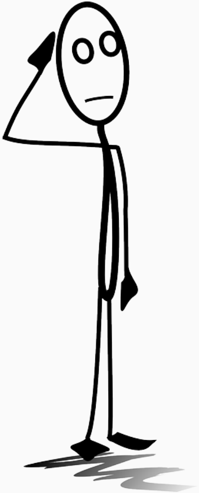
Transformers

- Some limitations of our approaches so far
- BERT, the game changer
- Critical voices
- Practical example

Next steps

Before we start: Questions from last week?

Today: From word embeddings via neural networks towards Transformers



Remember what we discussed about the move from BOW to word embeddings?

Neural Networks and Deep Learning

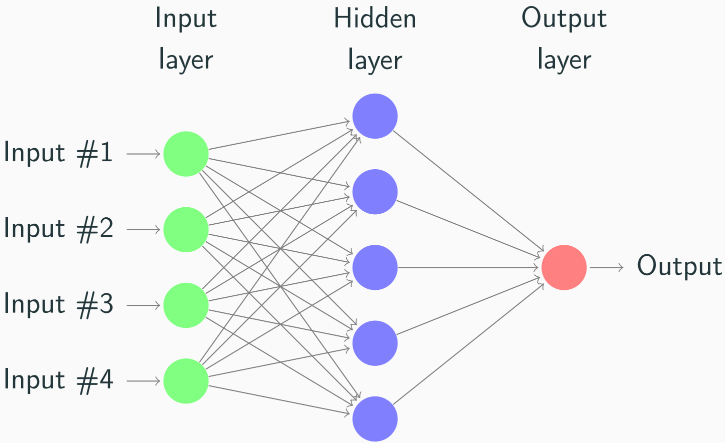
Neural Networks and Deep Learning

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”

Note that in our earlier example with our EmbeddingVectorizer, we essentially added a “layer” between the input and the output. Now, we generalize this idea.



⇒ 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 from 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 from 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 from 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 from logistic regression)
- Of course, we can add multiple layers in between if we want to

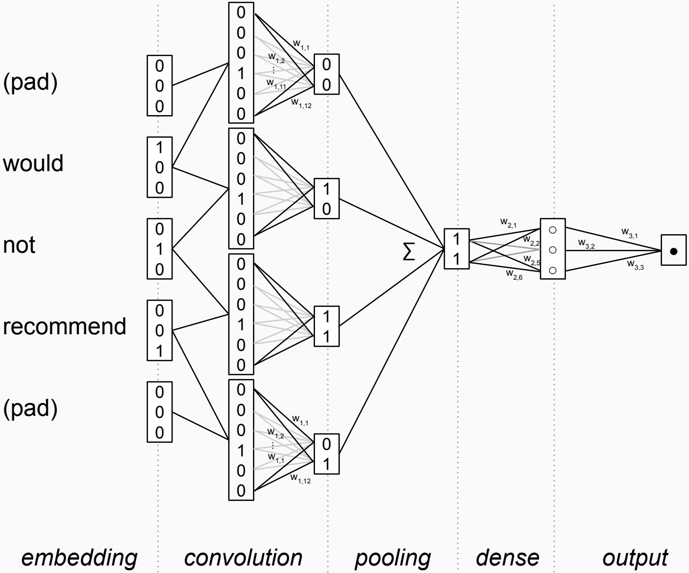
Convolutional networks

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

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!

There are example notebooks on github!

Let’s look into some code

https://github.com/uvacw/teaching-bdaca/blob/main/12ec-course/week08/exercises/06downstreamkeras.ipynb

Transformers

Transformers

Some limitations of our approaches so far

Some problems to address

Just replacing BOW with word embeddings is not enough:

Context

- “bank” (money), “bank” (of a river), . . . → all the same word embedding
- Understanding references within a sentence (“ who is ‘she’?”) → can actually be done with, for instance, LSTM/RNN

Some problems to address

Tha amount of training data

- Thousands of annotations needed in traditional BOW
- Especially be problematic when we have many and/or small categories
- We already argued that pre-trained embeddings can partly mitigate this
- Yet, a human doesn't need hundreds of examples but just a few to learn the difference between, say, two animal species (few-shot learning)

"Attention is all you need"

- Title of an extremely influential paper (Vaswani et al., 2017)
- Paradigm shift:

  "The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely."

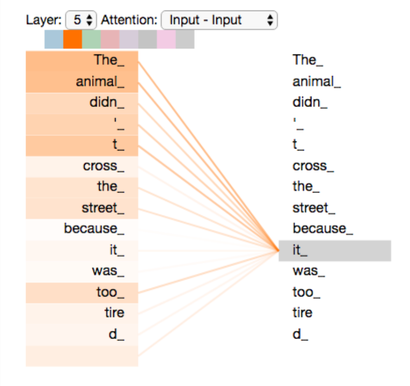
"Attention is all you need"

- Title of an extremely influential paper (Vaswani et al., 2017)
- Paradigm shift:

  "The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely."

OK, that sounds complicated.

“Attention is all you need”



Layer: 5 Attention: Input - Input

Figure 1: The idea of “attention”.
http://jalammar.github.io/illustrated-transformer/

The technical details

We will not go into all the math behind it – it's beyond the scope of this class. But there is a nice hands-on explanation of the Vaswani et al. (2017)-paper including a commented Python implementation available at http://nlp.seas.harvard.edu/annotated-transformer/.

(Further reading if you consider to continue working on this)

Transformers

BERT, the game changer

Bidirectional Encoder Representations from Transformers

The famous BERT (Devlin et al., 2018) model

> *" The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to- right language model pre-training, the MLM ob- jective enables the representation to fuse the left and the right context, which allows us to pre- train a deep bidirectional Transformer. In addi- tion to the masked language model, we also use a "next sentence prediction" task that jointly pre- trains text-pair representations."*

The idea of finetuning

- BERT (or GPT-4, . . .) are Large Language Models (LLMs)
- Training them is *really* expensive. Training BERT is said to have cost 7,000 USD, GPT-3 even 4,600,000 USD (just for the computing!)
- So, no, you don't do that yourself (nor do universities, normally).
- Solution: Separating (unsupervised) pre-training from fine-tuning for downstream tasks!

Pre-training vs fine-tuning

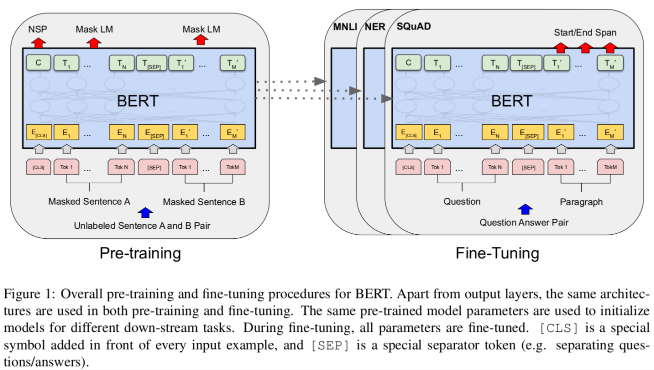


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

Figure 2: Finetuning BERT (Devlin et al., 2018).

How to fine-tune

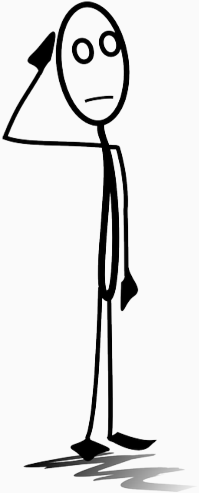
General idea: you download a pre-trained model, than finetune it

- https://www.huggingface.co is your starting point
- We also made an example notebook: https://github.com/uvacw/teaching-bdaca/blob/main/modules/machinelearning-text-exercises/transformers_bert_classification.ipynb
- You probably want to use a GPU (e.g., on Google Colab)

How to fine-tune

Note that you can fine-tune for very differnet tasks:

- classification (→ SML)
- question answering
- translation



Do you see how this relates to public-facing applications like DALL-E and ChatGPT? Do you see how these relate to encoding, decoding, and transformers?

Transformers

Critical voices

Stochastic parrots?

Really read the paper by (Bender et al., 2021)!

Think about:

- environmental and financial costs
- bias
- ethical issues
- “amplification of a hegemonic worldview”
- ...

Transformers

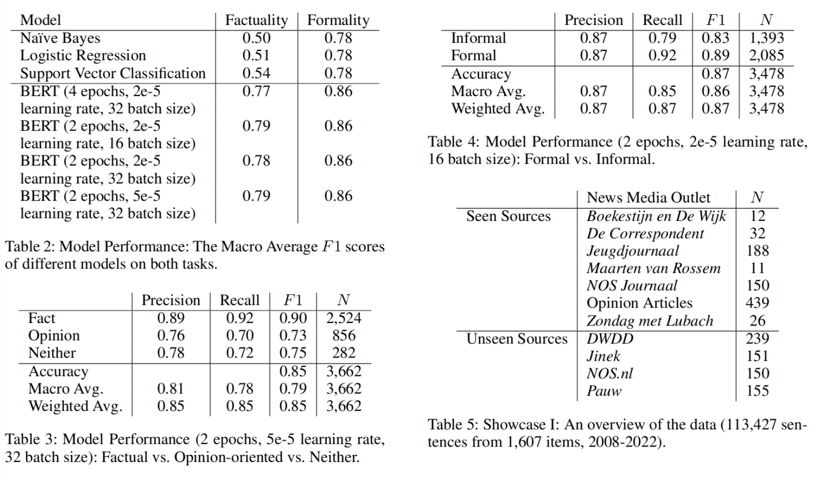
Practical example

Lin et al. (2023)

Can we train a model to rate news items *from very different sources such as newspaper articles, videos, podcasts, blogs, . . .* on two scales

1. informal — formal
2. factual — opinionated

to identify/describe their genre?



Model	Factuality	Formality
Naïve Bayes	0.50	0.78
Logistic Regression	0.51	0.78
Support Vector Classification	0.54	0.78
BERT (4 epochs, 2e-5 learning rate, 32 batch size)	0.77	0.86
BERT (2 epochs, 2e-5 learning rate, 16 batch size)	0.79	0.86
BERT (2 epochs, 2e-5 learning rate, 32 batch size)	0.78	0.86
BERT (2 epochs, 5e-5 learning rate, 32 batch size)	0.79	0.86

Table 2: Model Performance: The Macro Average $F1$ scores of different models on both tasks.

	Precision	Recall	$F1$	N
Fact	0.89	0.92	0.90	2,524
Opinion	0.76	0.70	0.73	856
Neither	0.78	0.72	0.75	282
Accuracy			0.85	3,662
Macro Avg.	0.81	0.78	0.79	3,662
Weighted Avg.	0.85	0.85	0.85	3,662

Table 3: Model Performance (2 epochs, 5e-5 learning rate, 32 batch size): Factual vs. Opinion-oriented vs. Neither.

	Precision	Recall	$F1$	N
Informal	0.87	0.79	0.83	1,393
Formal	0.87	0.92	0.89	2,085
Accuracy			0.87	3,478
Macro Avg.	0.87	0.85	0.86	3,478
Weighted Avg.	0.87	0.87	0.87	3,478

Table 4: Model Performance (2 epochs, 2e-5 learning rate, 16 batch size): Formal vs. Informal.

	News Media Outlet	N
Seen Sources	*Boekestijn en De Wijk*	12
	De Correspondent	32
	Jeugdjournaal	188
	Maarten van Rossem	11
	NOS Journaal	150
	Opinion Articles	439
	Zondag met Lubach	26
Unseen Sources	*DWDD*	239
	Jinek	151
	NOS.nl	150
	Pauw	155

Table 5: Showcase I: An overview of the data (113,427 sentences from 1,607 items, 2008-2022).

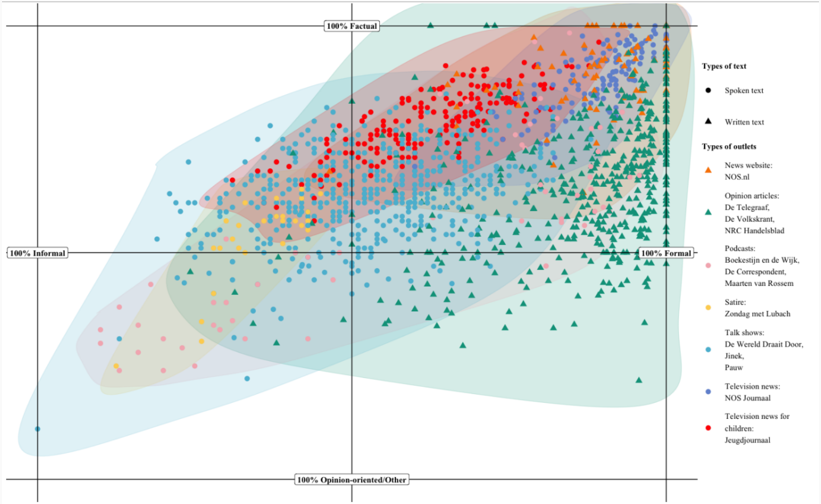


Figure 1: A two-dimensional grid with Dutch news items.

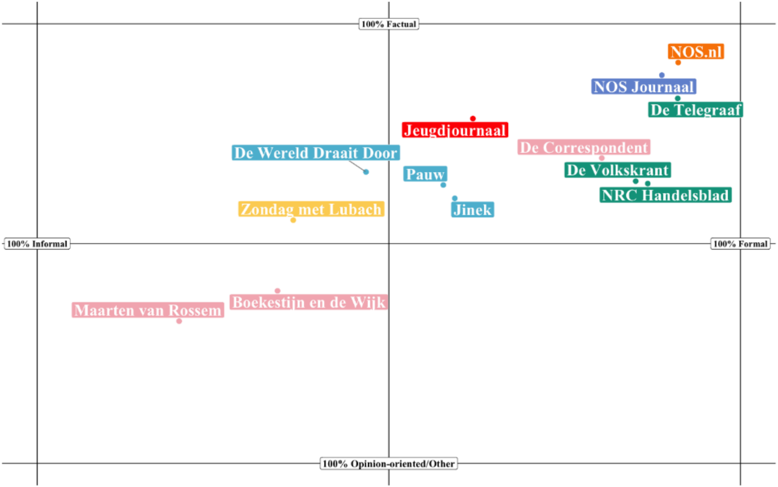


Figure 2: A two-dimensional grid with Dutch news items aggregated on the outlet level.

Lin et al. (2023)

- BOW approaches cannot achieve this well enough, but finetuning a transformer can
- They can generalize across *very* different formats

Finally: Some links

- https://nlp.seas.harvard.edu/2018/04/03/attention.html
- http://jalammar.github.io/illustrated-transformer/

Let’s look into Google Colab!

Next steps

Try it yourself! https://github.com/uvacw/teaching-bdaca/blob/main/12ec-course/week09/exercises/README.md

References

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). ***On the dangers of stochastic parrots: Can language models be too big?* (Vol. 1).** Association for Computing Machinery. https://doi.org/10.1145/3442188.3445922

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). **Bert: Pre-training of deep bidirectional transformers for language understanding.** *arXiv preprint arXiv:1810.04805*.

Lin, Z., Welbers, K., Vermeer, S., & Trilling, D. (2023). **Beyond discrete genres: Mapping news items onto a multidimensional framework of genre cues [https://arxiv.org/abs/2212.04185].** *International Conference on the Web and Social Media (ICWSM)*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). **Attention is all you need.** In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf