EE-559 – Deep learning

13.2. Transformer Networks

François Fleuret

https://fleuret.org/ee559/

Feb 2, 2020

The most powerful language models (as of 03.07.2019) take the form of a sequence of attention-based revisions of the representation.

The original sequence structure of the signal is secondary and provided indirectly to the processing through additional inputs.

Vaswani et al. (2017) use the terminology of Graves et al. (2014): attention is an averaging of **values** associated to **keys** matching a **query**.

With Q the tensor of row queries, K the keys, and V the values,

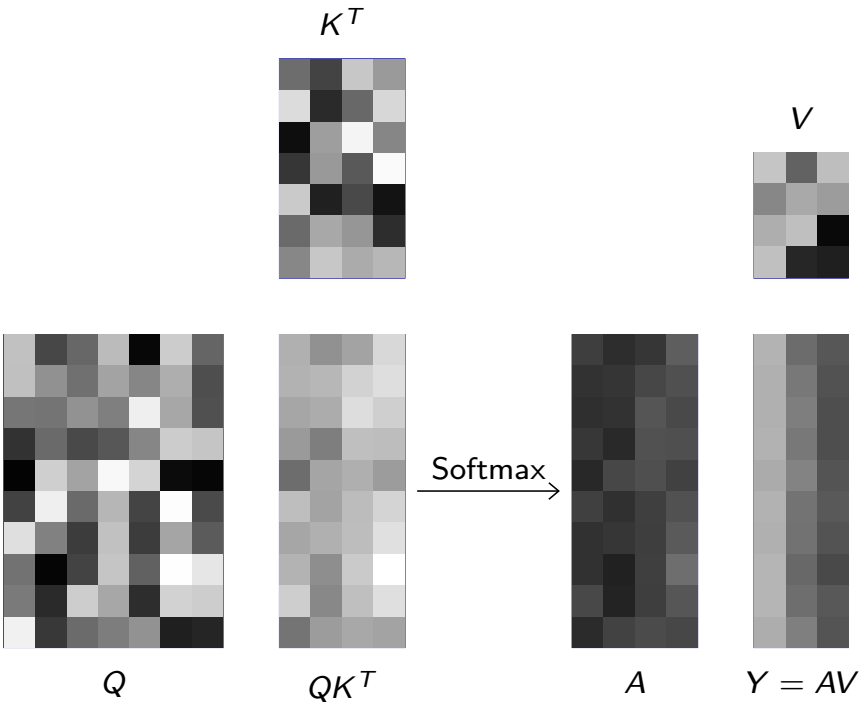
$$Y_j = \sum_i \text{softmax}_i(Q_j, K_i^T) V_i$$

or

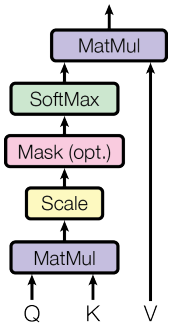
$$Y = \text{softmax}\left(\frac{Q\,K^T}{\sqrt{d}}\right) V \in \mathbb{R}^{a \times d},$$

where

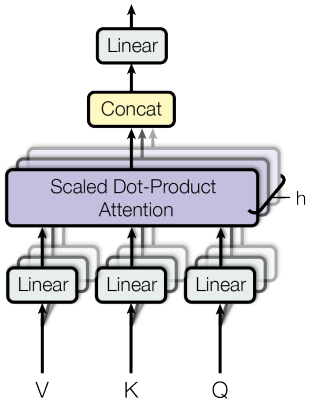
$$Q \in \mathbb{R}^{a \times b}, K \in \mathbb{R}^{c \times b}, V \in \mathbb{R}^{c \times d}.$$



Scaled Dot-Product Attention



Multi-Head Attention



(Vaswani et al., 2017)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\,K^T}{\sqrt{d}}\right) V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}\,(H_1, \ldots, H_h)\, W^O$$

$$H_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right),\ i = 1, \ldots, h$$

with

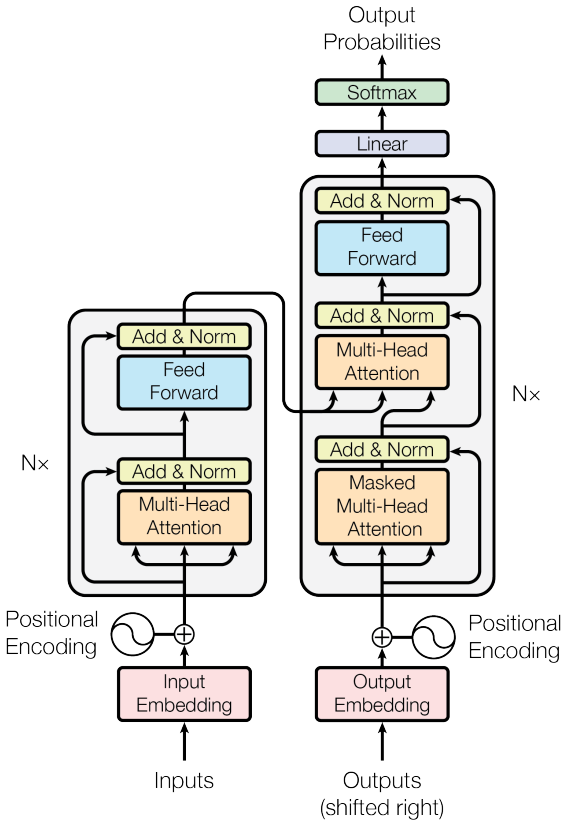
$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k},\ W_i^K \in \mathbb{R}^{d_{model} \times d_k},\ W_i^V \in \mathbb{R}^{d_{model} \times d_v},\ W^O \in \mathbb{R}^{hd_v \times d_{model}}$$

Attention disregards positioning in the sequence, Vaswani et al. provide this information through a **positional encoding**, added to the original input.

It has the same dimension as d_{model} and is of the form

$$PE_{t,2i} = \sin\left(\frac{t}{10,000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{t,2i+1} = \cos\left(\frac{t}{10,000^{\frac{2i+1}{d_{model}}}}\right)$$



(Vaswani et al., 2017)

The Universal Transformer (Dehghani et al., 2018) is a similar model where all the blocks are identical, resulting in **a recurrent model that iterates over consecutive revisions of the representation instead of positions.**

The positional embedding is expended with the block index $1 \leq t \leq T$

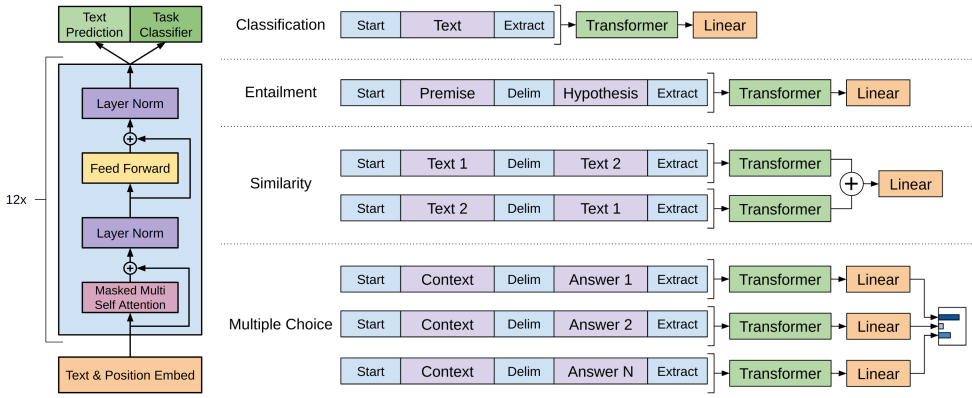
$$P^t_{i,2j} = \sin\left(\frac{i}{10,000^{\frac{2j}{d_{model}}}}\right) + \sin\left(\frac{t}{10,000^{\frac{2j}{d_{model}}}}\right)$$

$$P^t_{i,2j+1} = \cos\left(\frac{i}{10,000^{\frac{2j}{d_{model}}}}\right) + \cos\left(\frac{t}{10,000^{\frac{2j}{d_{model}}}}\right)$$

Additionally the number of steps is modulated per position dynamically.

Transformers trained on NLP tasks

GPT is a transformer trained for generating.



(Radford, 2018)

"GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data."

(Radford et al., 2019)

BERT (Devlin et al., 2018) is a transformer pre-trained with:

- Masked Language Model (MLM), that consists in predicting [15% of] words which have been replaced with a "MASK" token.
- Next Sentence Prediction (NSP), which consists in predicting if a certain sentence follows the current one.

It is then fine-tuned on multiple NLP tasks. (Conneau et al., 2019)

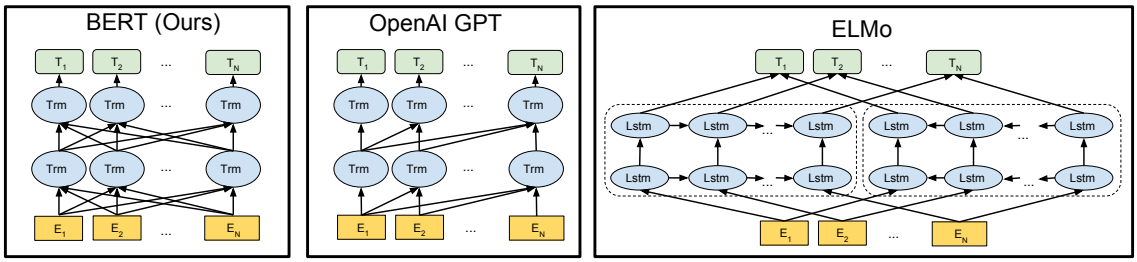


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

(Devlin et al., 2018)

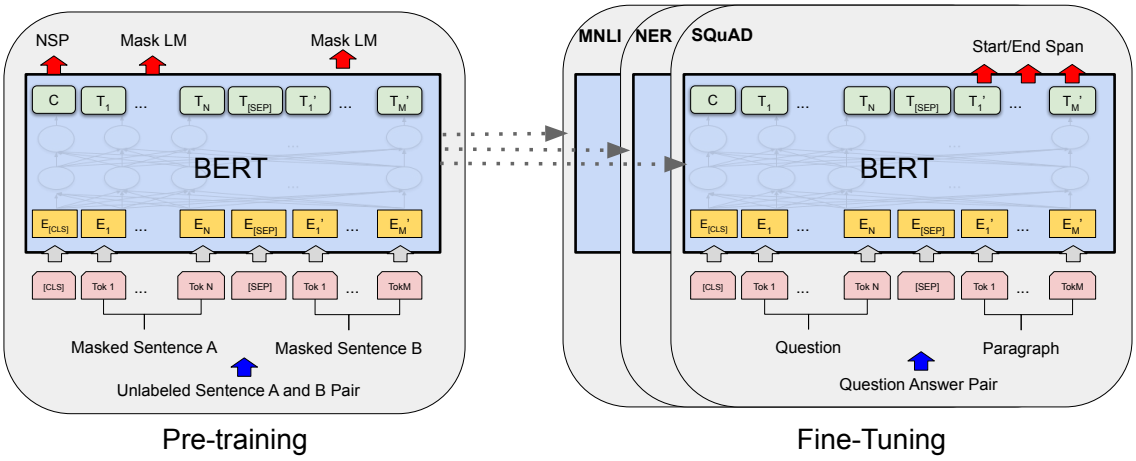
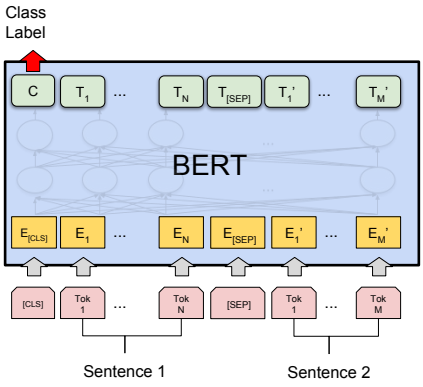
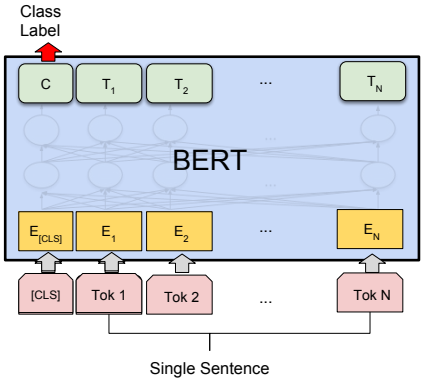


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are 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).

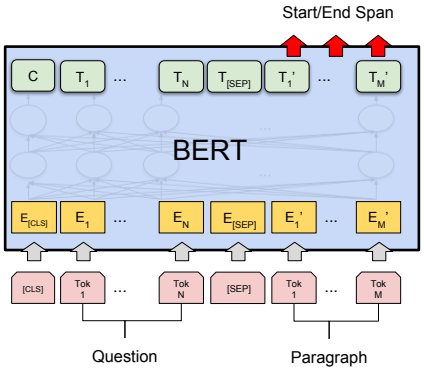
(Devlin et al., 2018)



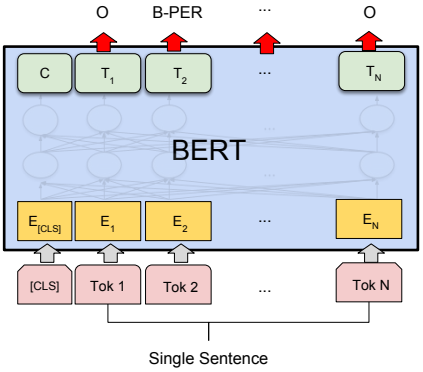
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

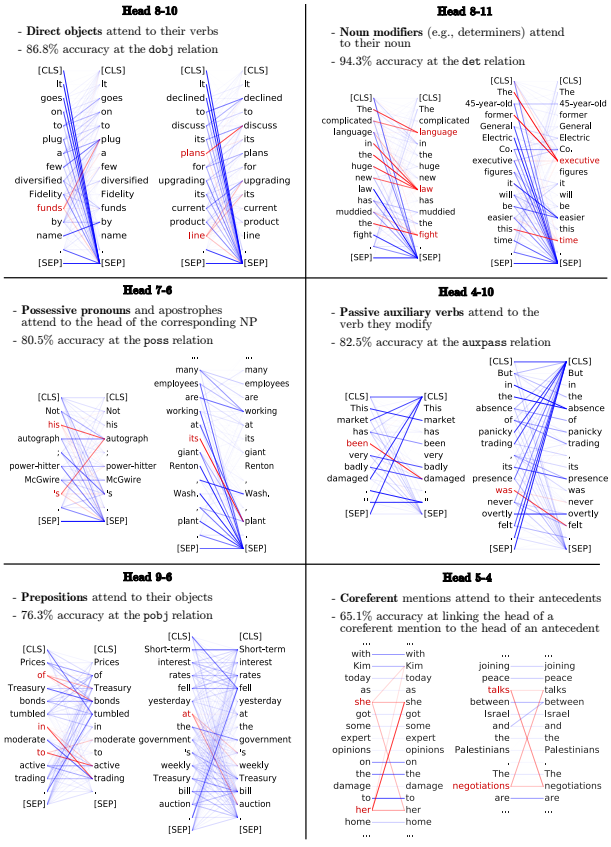


(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

(Devlin et al., 2018)



(Clark et al., 2019)

Attention in computer vision

Wang et al. (2018) proposed an attention mechanism for images, following the model from Vaswani et al. (2017).

$$y = \text{softmax}\left(\left(W_\theta x\right)^T \left(W_\Phi x\right)\right) W_g x.$$

Wang et al. insert "non-local blocks" in residual architectures and get improvements on both video and images classification.

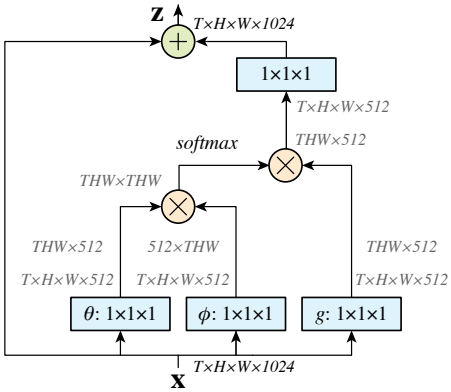


Figure 2. A spacetime **non-local block**. The feature maps are shown as the shape of their tensors, *e.g.*, $T\times H\times W\times 1024$ for 1024 channels (proper reshaping is performed when noted). "$\otimes$" denotes matrix multiplication, and "$\oplus$" denotes element-wise sum. The softmax operation is performed on each row. The blue boxes denote $1\times1\times1$ convolutions. Here we show the embedded Gaussian version, with a bottleneck of 512 channels. The vanilla Gaussian version can be done by removing θ and ϕ, and the dot-product version can be done by replacing softmax with scaling by $1/N$.

(Wang et al., 2018)

Figure 3. Examples of the behavior of a non-local block in res_3 computed by a 5-block non-local model trained on Kinetics. These example are from held-out validation videos. The starting point of arrows represents one $\mathbf{x}_i$, and the ending points represent $\mathbf{x}_j$. The 20 highes weighted arrows for each $\mathbf{x}_i$ are visualized. The 4 frames are from a 32-frame input, shown with a stride of 8 frames. These visualization show how the model finds related clues to support its prediction.

(Wang et al., 2018)

Ramachandran et al. (2019) replaced convolutions with local attention.

$$y_{i,j} = \sum_{(a,b)\in\mathcal{N}(i,j)} W_{i-a,j-b} x_{a,b} \qquad \text{(Convolution)}$$

$$y_{i,j} = \sum_{(a,b)\in\mathcal{N}(i,j)} \text{softmax}_{a,b}\left(\left(W_Q x_{i,j}\right)^T \left(W_K x_{a,b}\right)\right) v_{a,b} \qquad \text{(Local attention)}$$

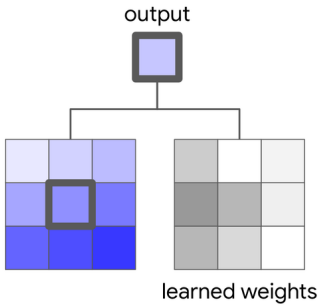


Figure 2: An example of a 3×3 convolution. The output is the inner product between the local window and the learned weights.

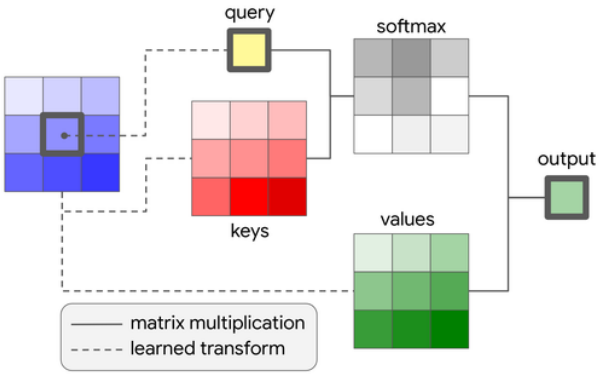


Figure 3: An example of a local attention layer over spatial extent of $k = 3$.

(Ramachandran et al., 2019)

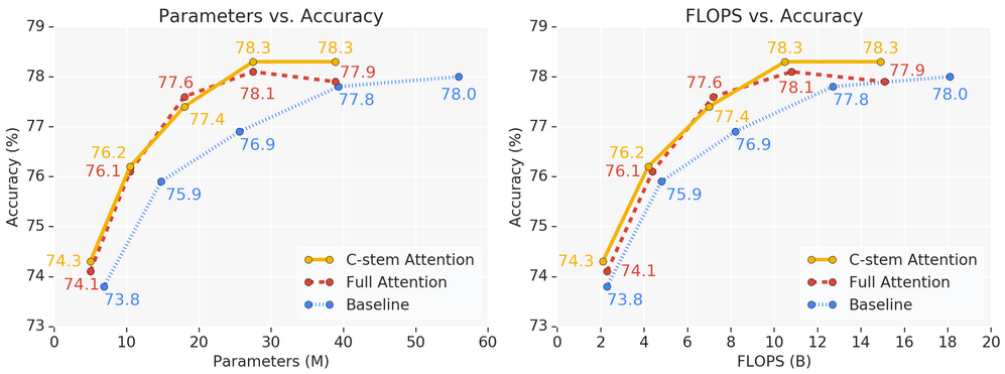


Figure 5: Comparing parameters and FLOPS against accuracy on ImageNet classification across a range of network widths for ResNet-50. Attention models have fewer parameters and FLOPS while improving upon the accuracy of the baseline.

(Ramachandran et al., 2019)

Implementation example

PyTorch allows to operate with matrices or linear modules over tensors of dimensions greater than 2.

```
>>> a = torch.empty(100, 2, 3, 4)
>>> m = nn.Linear(4, 11)
>>> m(a).size()
torch.Size([100, 2, 3, 11])
```

matmul allows structured batch matrix products.

```
>>> a = torch.empty(11, 9, 2, 3).normal_()
>>> b = torch.empty(11, 9, 3, 4).normal_()
>>>
>>> m = a.matmul(b)
>>>
>>> m.size()
torch.Size([11, 9, 2, 4])
>>>
>>> m[7, 1]
tensor([[-0.0234,  0.9557, -0.8338,  2.2662],
        [-0.1781,  0.2189, -0.5915, -0.1888]])
>>> a[7, 1].mm(b[7, 1])
tensor([[-0.0234,  0.9557, -0.8338,  2.2662],
        [-0.1781,  0.2189, -0.5915, -0.1888]])
>>>
>>> m[3, 0]
tensor([[-0.7566, -0.1191,  0.9691,  0.6992],
        [ 0.7131, -0.7072, -0.4597, -1.6044]])
>>> a[3, 0].mm(b[3, 0])
tensor([[-0.7566, -0.1191,  0.9691,  0.6992],
        [ 0.7131, -0.7072, -0.4597, -1.6044]])
```



```
class AttentionLayer2d(nn.Module):
    def __init__(self, dim_x, dim_k, dim_v):
        super(AttentionLayer2d, self).__init__()
        self.dim_k = dim_k
        self.W_q = nn.Linear(dim_x, dim_k, bias = None)
        self.W_k = nn.Linear(dim_x, dim_k, bias = None)
        self.W_v = nn.Linear(dim_x, dim_v, bias = None)

    def forward(self, x):
        # NCHW -> NKC
        u = x.permute(0, 2, 3, 1)
        u = u.view(u.size(0), -1, u.size(3))

        q = self.W_q(u)
        k = self.W_k(u)
        v = self.W_v(u)

        a = q.matmul(k.transpose(1, 2)) / math.sqrt(self.dim_k)
        # a will operate on the right, it should be row-normalized
        a = torch.softmax(a, 2)
        y = a.matmul(v)

        # NKC -> NCHW
        y = y.reshape(x.size(0), x.size(2), x.size(3), -1)
        y = y.permute(0, 3, 1, 2)

        return y
```

References

K. Clark, U. Khandelwal, O. Levy, and C. Manning. **What does BERT look at? an analysis of BERTś attention**. CoRR, abs/1906.04341, 2019.

A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. **Unsupervised cross-lingual representation learning at scale**. CoRR, abs/1911.02116, 2019.

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. **Universal transformers**. CoRR, abs/1807.03819, 2018.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. **BERT: Pre-training of deep bidirectional transformers for language understanding**. CoRR, abs/1810.04805, 2018.

A. Graves, G. Wayne, and I. Danihelka. **Neural turing machines**. CoRR, abs/1410.5401, 2014.

A. Radford. **Improving language understanding with unsupervised learning**. web, June 2018. `https://openai.com/blog/language-unsupervised/`.

A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage, and I. Sutskever. **Better language models and their implications**. web, February 2019. `https://blog.openai.com/better-language-models/`.

P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens. **Stand-alone self-attention in vision models**. CoRR, abs/1906.05909, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. **Attention is all you need**. CoRR, abs/1706.03762, 2017.

X. Wang, R. Girshick, A. Gupta, and K. He. **Non-local neural networks**. In Conference on Computer Vision and Pattern Recognition (CVPR), 2018.