Contextualised Meaning Representations and Transformers

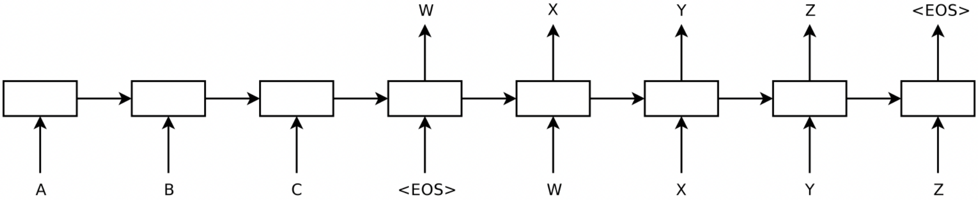
LT2213 V22: Computational Semantics

Nikolai Ilinykh
Department of Philosophy, Linguistics and Theory of Science
Centre for Linguistic Theory and Studies in Probability (CLASP)
University of Gothenburg, Sweden
{nikolai.ilinykh}@gu.se

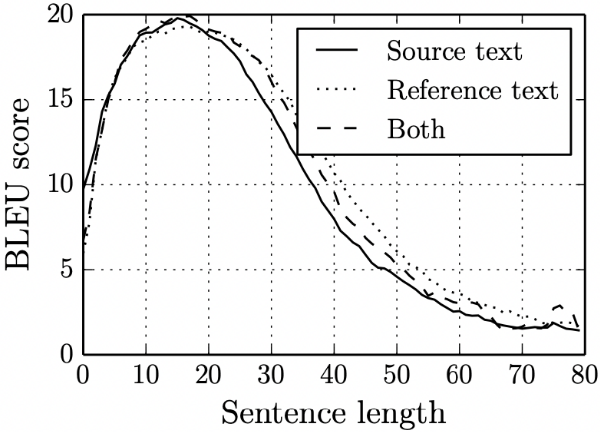
Presented at, May 9th, 2022

Where we are at

- Example task: language generation
- Questions for consideration:
  - How to generate a fluent and coherent text?
  - Data vs model trade-off: which model to choose? When is data good enough?
  - "He said: Teddy ___" - Teddy who or what? Bear? Roosevelt? We need context from the right side to make a correct prediction, e.g. Bi-LSTM (Schuster and Paliwal, 1997; Graves and Schmidhuber, 2005; Wu et al., 2016; Peters et al., 2018)
- Architecture: encoder-decoder RNN / LSTM (Sutskever et al., 2014)



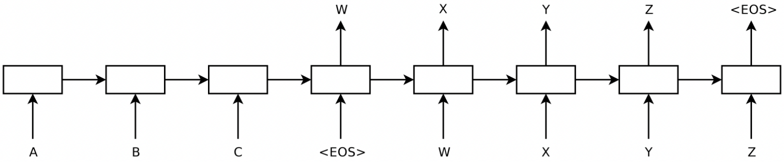
Problem: longer sentences (Cho et al., 2014)



GÖTEBORGS UNIVERSITET
CLASP centre for linguistic theory and studies in probability

Why longer sentences are hard?

GÖTEBORGS UNIVERSITET

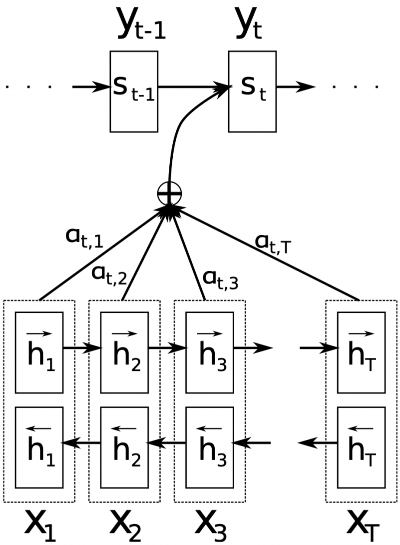


- Later hidden states in the decoder receive weaker learning signal, because encoded representation is blended with new information at each step during decoding process - last hidden states does not have a clear idea about encoded sentence.
- The model does not have knowledge of the word order and every time words are the same, but the order is different, the model will have hard time figuring out semantic differences between different sentences.

CLASP centre for linguistic theory and studies in probability

Solution: attend over input words (Bahdanau et al., 2015)

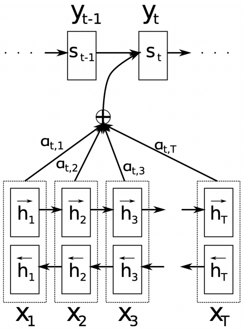
GÖTEBORGS UNIVERSITET



CLASP centre for linguistic theory and studies in probability

Solution: attend over input words (Bahdanau et al., 2015)

GÖTEBORGS UNIVERSITET



- Generation of the current word y_t depends not only on previously generated word y_{t-1}, but also on how important each word in the input X is for the current word.

CLASP centre for linguistic theory and studies in probability

Attention is an alignment function

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

- It is typically a linear layer that transforms hidden state and input sequence representations into attention scores over the input sequence.[1]

[1] https://lilianweng.github.io/posts/2018-06-24-attention/

Attention in sequence-to-sequence network [2]

```python
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.gru(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

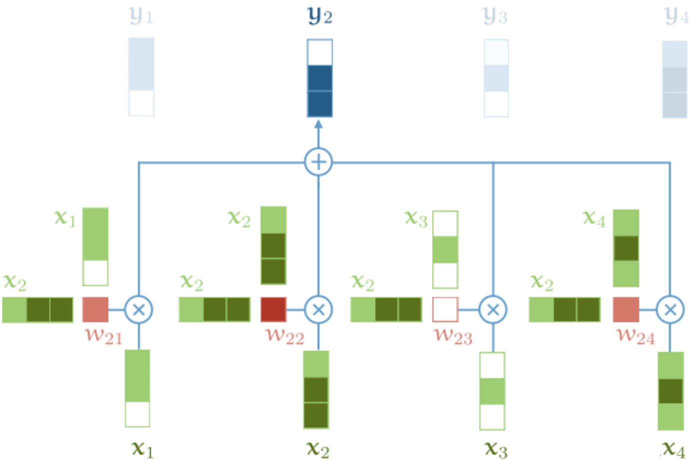
[2] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Self-Attention

GÖTEBORGS UNIVERSITET

- Self-attention is a sequence-to-sequence-operation.
  - It takes the sequence of vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t$ and outputs the sequence of vectors $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$, all vectors have same dimension k.
  - Each output vector $\mathbf{y}_i$ is a weighted average over all the input vectors:
    $\mathbf{y}_i = \sum_j \mathbf{w}_{ij} \mathbf{x}_j$.
  - The weight $\mathbf{w}_{ij}$ is **not** a model parameter (not yet!), but it is a function over x_i and x_j, typically **a dot product** function:
    $\mathbf{w}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$.
  - Dot product gives us values between negative and positive infinity, so we apply $\mathrm{softmax}$ to map the values in a range between 0 and 1:
    $\mathbf{w}_{ij} = \frac{exp \mathbf{w}_{ij}}{\sum_j exp \mathbf{w}_{ij}}$.

Self-Attention in a nutshell



Now, why does does self-attention work?

GÖTEBORGS UNIVERSITET

- When you multiply two feature vectors, you get a score for how well one feature corresponds to another feature.
- If both features match with each other, the resulting dot product gets a positive term; otherwise - negative term.
- Moreover, the **magnitude** of the features indicates contribution of each feature to the total score.
- Gathering features is impractical. Therefore, we make input features $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_t$ and output features $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_t$ <u>parameters of the model Θ</u>.
- Self-attention works because parameters of the model (feature embeddings) are updates based on alignment function (or function of your choice).

Applying self-attention

- First, assign each word $\mathbf{t}$ in your vocabulary an embedding vector $\mathbf{v}_t$. Embeddings will be learned automatically.
- Our input sequence is then turned into the sequence of vectors: $\mathbf{v}_{the}, \mathbf{v}_{person}, \mathbf{v}_{walks}, \mathbf{v}_{towards}, \mathbf{v}_{the}, \mathbf{v}_{station}$.
- Once self-attention is applied, the output will be another sequence of vectors $\mathbf{y}_{the}, \mathbf{y}_{person}, \mathbf{y}_{walks}, \mathbf{y}_{towards}, \mathbf{y}_{the}, \mathbf{y}_{station}$, where $\mathbf{y}_{person}$ is a weighted sum over all input embedding vectors, weighted by their dot-product with $\mathbf{v}_{person}$.

Self-Attention: specifics

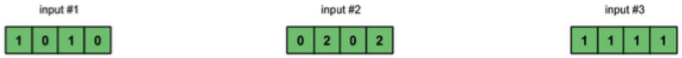
- The relatedness that we learn in $\mathbf{v}_t$ is determined by the task.
- So far, no parameters in self-attention, it is entirely determined by the way one creates feature representations (embeddings). Later we will see what type of parameters are added to self-attention in language transformers.
- Self-attention sees its input as a *set*, not a sequence.

Self-attention in Transformers

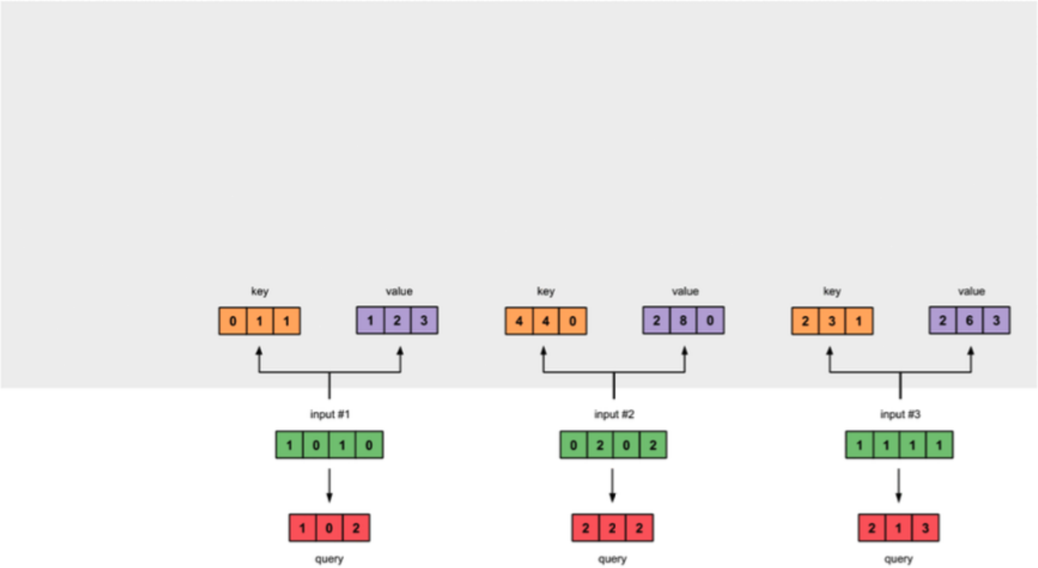
Transformers (Vaswani et al., 2017) are big language models which are successfully used across all NLP tasks. Their main power is in how they utilise self-attention. Each input vector $\mathbf{x}_i$ is used in three different ways by self-attention:

- It is compared to every other input vector to establish the weights for its own output $\mathbf{y}_i$: **query**.
- It is compared to every other input vector to establish the weights for other output vectors $\mathbf{y}_j$: **key**.
- It is used as part of the weighted sum to compute each output vector once the weights have been established: **value**.
- In the basic self-attention, each input vector plays all three roles. Each role is learned separately by creating three linear layers which are learned.

Self-Attention: prepare inputs



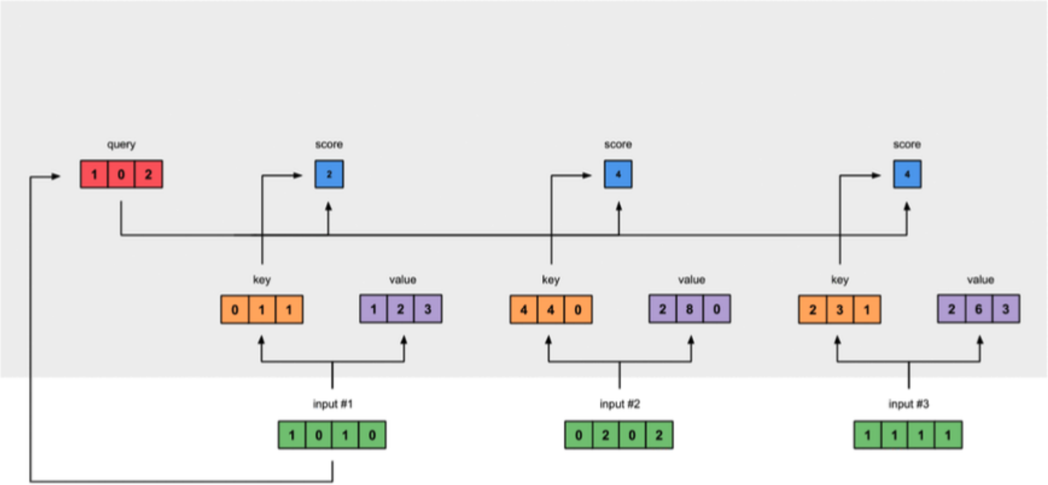
Self-Attention: compute keys, queries and values



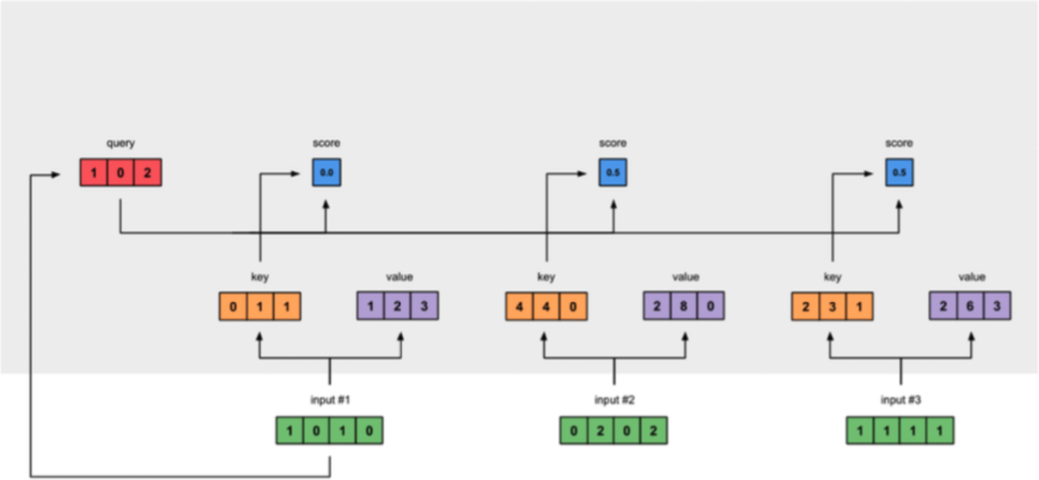
GÖTEBORGS UNIVERSITET

CLASP linguistic theory and studies in probability

Self-Attention: calculate attention scores for each input



Self-Attention: transform attention scores into probabilities

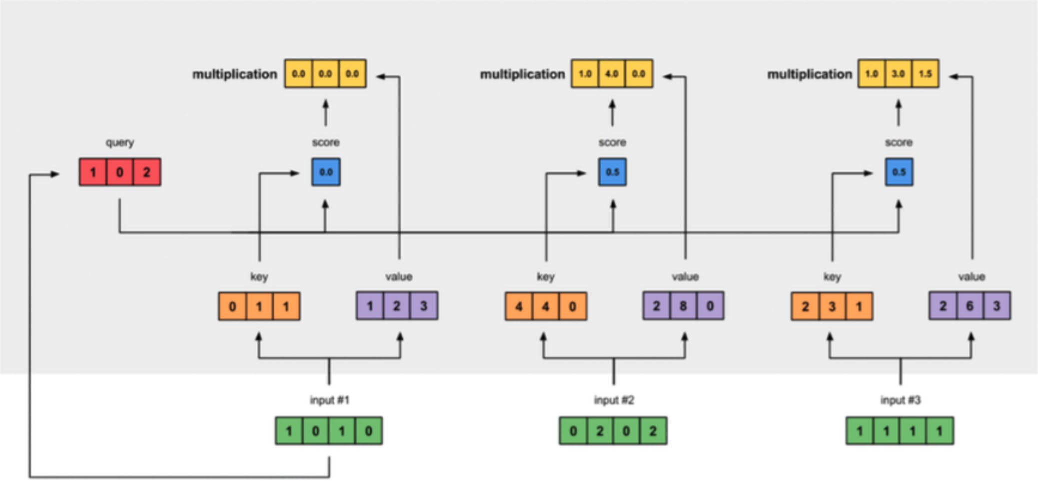


GÖTEBORGS UNIVERSITET

CLASP centre for linguistic theory and studies in probability

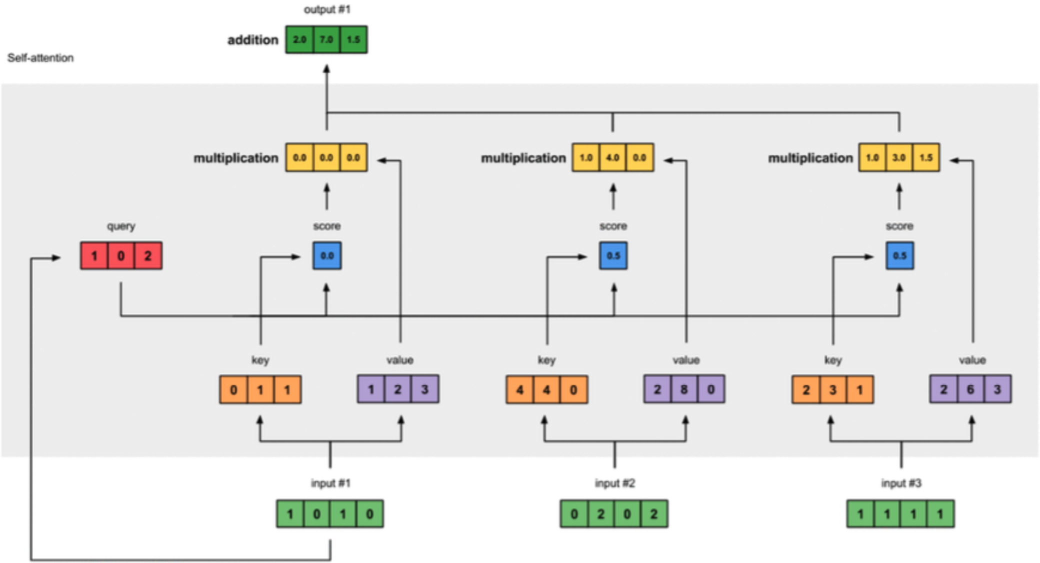
Self-Attention: multiply the result with values

GÖTEBORGS UNIVERSITET

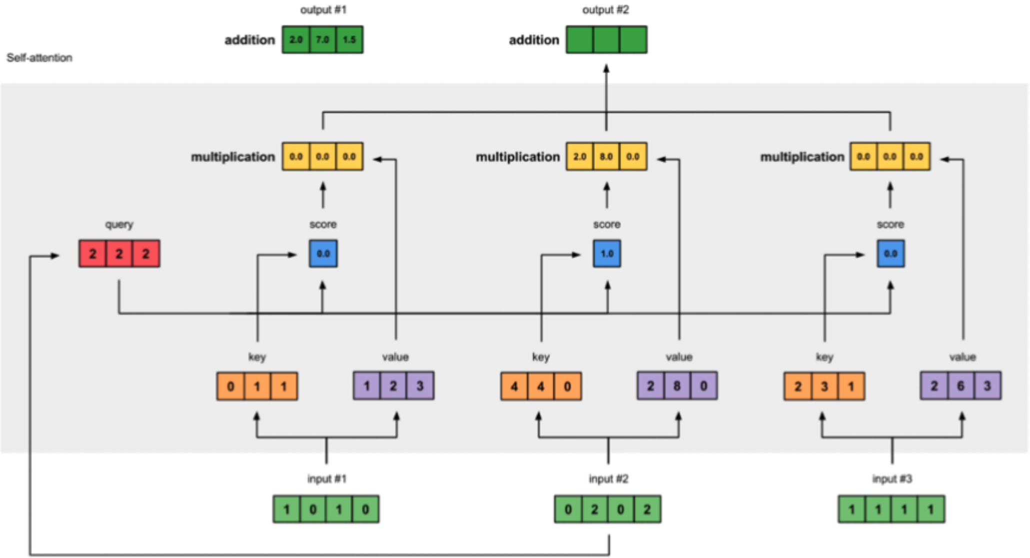


CLASP centre for linguistic theory and studies in probability

Self-Attention: sum all weight values to compute current output

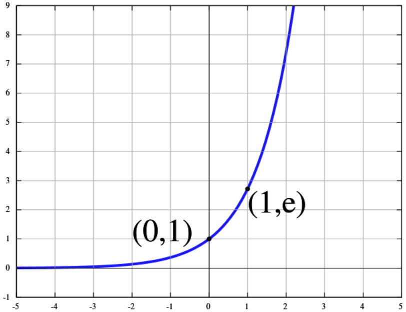


Self-Attention: repeat for all other outputs



Self-attention trick: scaling the dot product

GÖTEBORGS UNIVERSITET



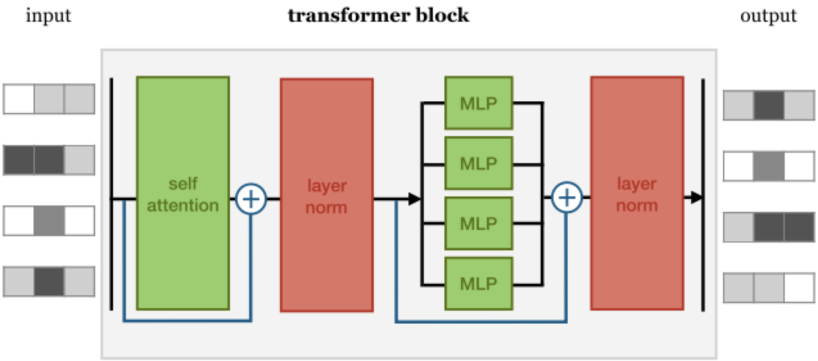
- Softmax is based on exponential function, so it is sensitive to very large input values (logits).
- These would slow down learning and kill the gradient.
- The dot product is scaled by the square root of the dimension size **k**.

Multi-head attention

- A word can mean different things to different neighbours.
- Therefore, we give additional power of discrimination to self-attention by combining **several** self-attention mechanisms each with different queries, keys and values. We call them **attention heads**.
- For each input $\mathbf{x}_i$ each attention head will produce a different output vector $\mathbf{y}_i^h$. They are concatenated and passed through a linear layer to reduce the dimensions back to $\mathbf{k}$.

Building Transformers

- A transformer is an architecture where the only direct interaction between units is through self-attention.
- First, wrap up the self-attention into a block that will be repeated.



BERT

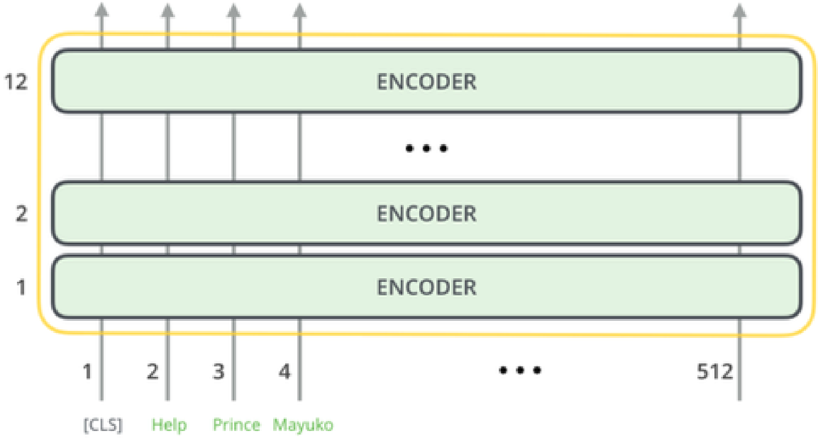
- BERT is a transformer encoder model, consisting of multiple encoder stacks.
- **Input**: embeddings of the sequence of tokens, including some special tokens (CLS, SEP, MASK).

GÖTEBORGS UNIVERSITET

centre for linguistic theory and studies in probability

BERT

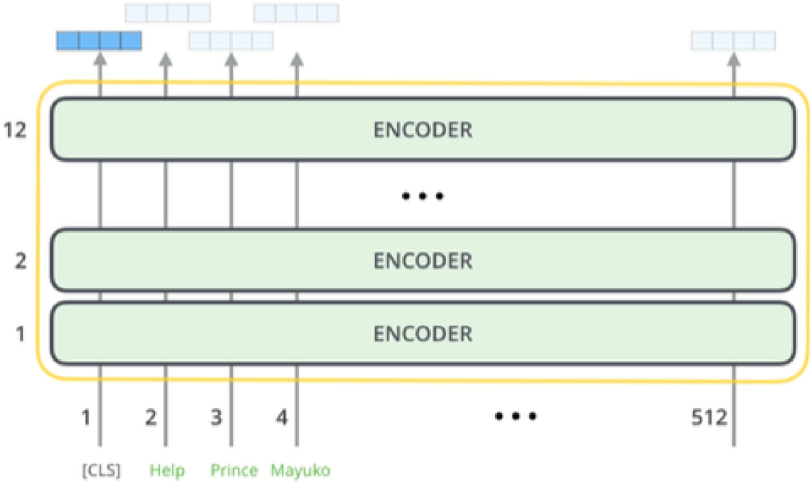
- In terms of structure, the model consists of 12 encoders each equipped with self-attention, 8 attention heads in each.



BERT

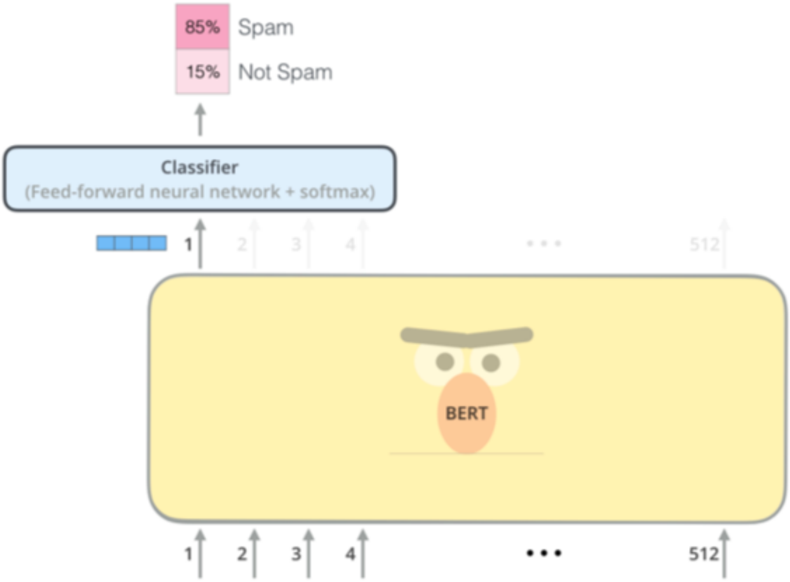
BERT

- Output is a sequence of vectors for each input term, *but* empirically it has been shown that it is relatively ok to use the CLS token for our task. Why? Because it encompasses information about the whole sequence (think of it as a general topic of the sentence).

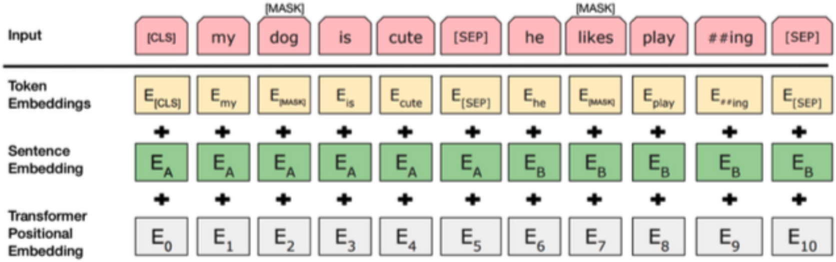


BERT

- Task: for example, spam classification, e.g. BERT is a classification model, it is very hard to use it for text generation.



BERT: input



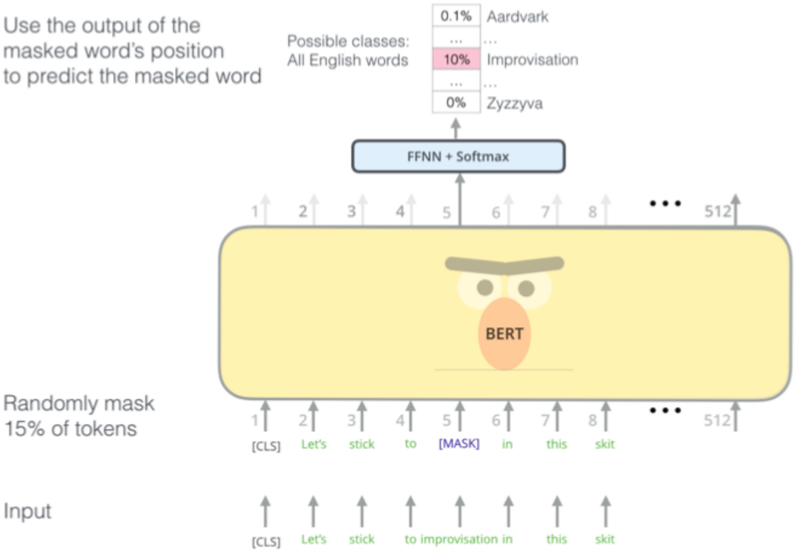
Input
[CLS]
my
[MASK]
dog
is
cute
[SEP]
he
[MASK]
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} $E_{[MASK]}$ E_{is} E_{cute} $E_{[SEP]}$ E_{he} $E_{[MASK]}$ E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Sentence Embedding
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Transformer Positional Embedding
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Properties of BERT

GÖTEBORGS UNIVERSITET

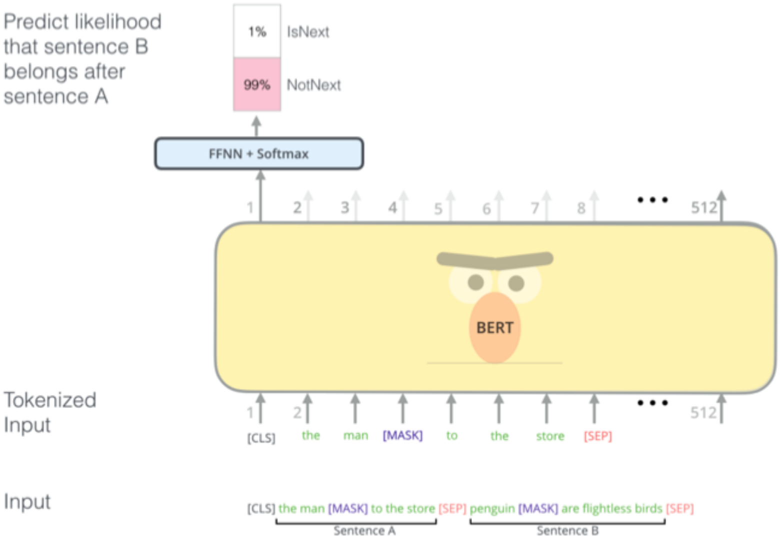
- BERT reads the entire sequence at once: it can be considered a bidirectional model, but it's more correct to say that it is a non-direction architecture.
- BERT is trained with two strategies: masked language modelling and next sentence prediction
  - MLM: 15% of the words in the input sequence are replaced with a MASK token. The task is to predict original values of the masked words, based on the context provided by non-masked words. We learn very deep representations based on this objective.
  - NSP: put two sentences in a single sequence, separate them by the SEP token and learn to predict whether the second sentence in the pair is likely to be the sentence that follows from the first one. We learn to better handle relations between multiple sentences, e.g. a random sentence would be disconnected from the first sentence.

BERT: masked language modelling



BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

BERT: next sentence prediction



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

BERT: fine-tuning

GÖTEBORGS UNIVERSITET

- We have discussed how to train BERT, but in fact it was pre-training.
- We need to fine-tune BERT on downstream tasks. Why? During pre-training the model is learning from some sort of fundamental tasks / a lot of data to capture general patterns, then it is fine-tuned for a specific task on a specific data. Model's parameters are slightly changed (not all of them, sometimes) in order to fit specific data better.
- We take pre-trained BERT and unfreeze its parameters for some more training.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610. IJCNN 2005.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.