Modern NN Architectures and Transfer Learning

Vsevolod Domkin

prj-nlp-2020

Outline

* Multi-Task Learning
* Transfer Learning
* UlMFit
* ElMo
* Attention Mechanism
* Transformer
* BERT
* Beyond BERT

Transfer Learning

Conserving knowledge gained while solving one problem and applying it to a different but related problem

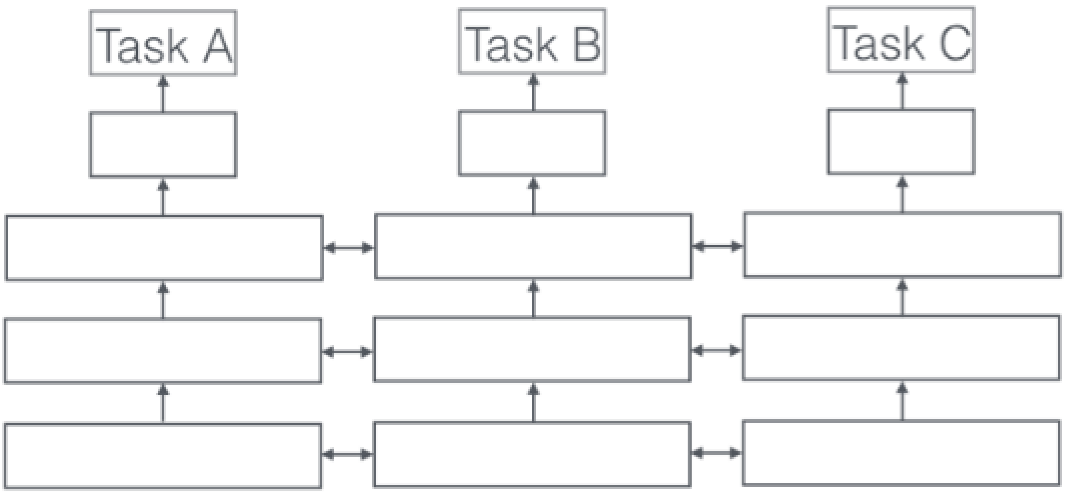
Multi-Task Learning

Training on several datasets simultaneously, information/parameters sharing between similar problems.

Generalization is improved by using training data for similar or related tasks.

Origins:
* Rich Caruana, Multi-task Leaning: A Knowledge-Based Source of Inductive Bias. Proceedings of ICML, 1993.
* Collobert, Ronan, et al. Natural language processing (almost) from scratch, Journal of Machine Learning Research 12, 2011.
* Augenstein, Isabelle, and Anders Søgaard. "Multi-Task Learning of Keyphrase Boundary Classification." arXiv preprint arXiv:1704.00514, 2017.

Multi-Task Learning

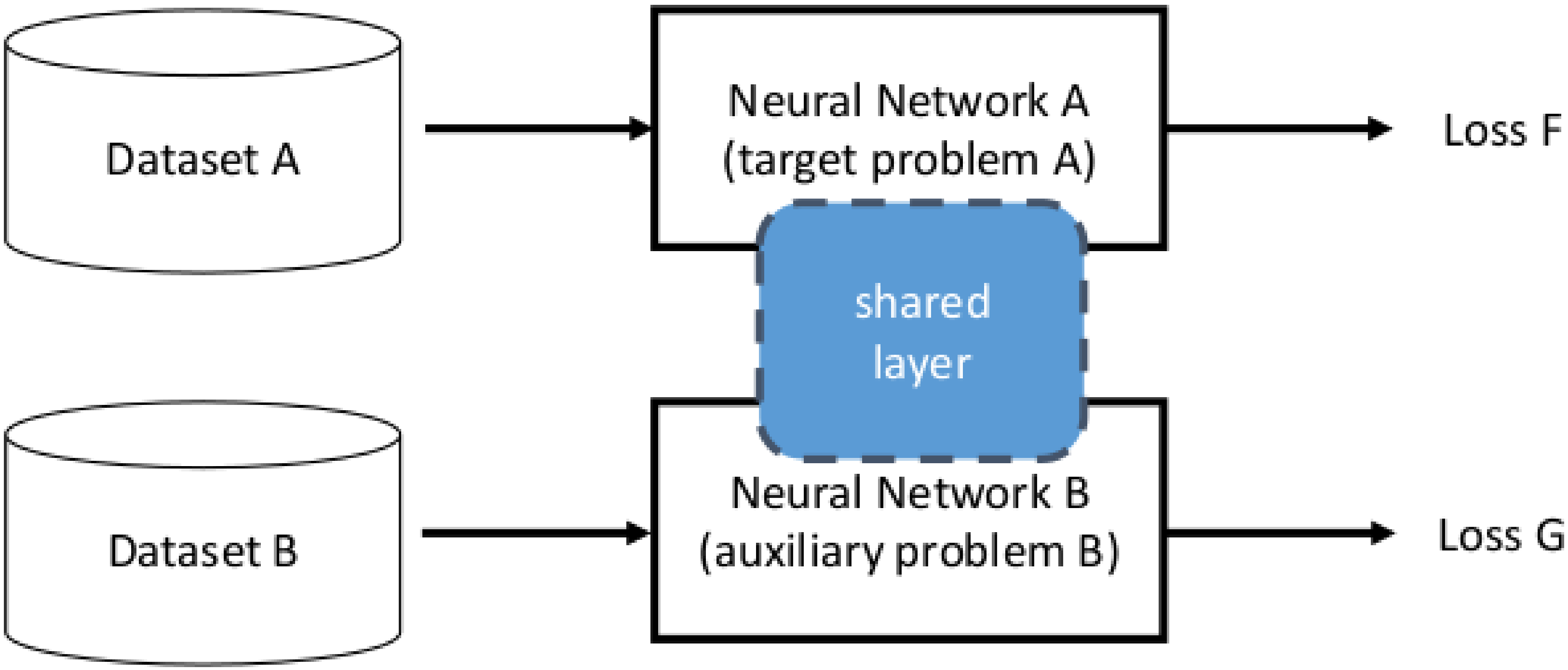


- Auxiliary objectives from the same datasets (language modelling, predict data statistics, learning the inverse)
- Joint training on similar NLP tasks (machine translation, semantic parsing, chunking, speech recogntion)

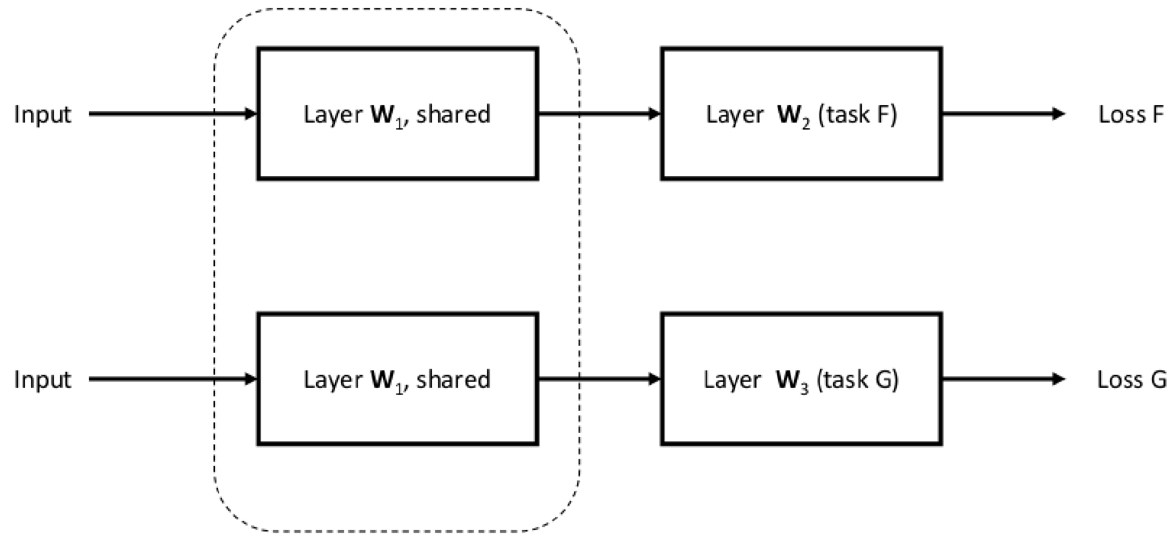
Source: https://ruder.io/multi-task/

Multi-Objective Loss

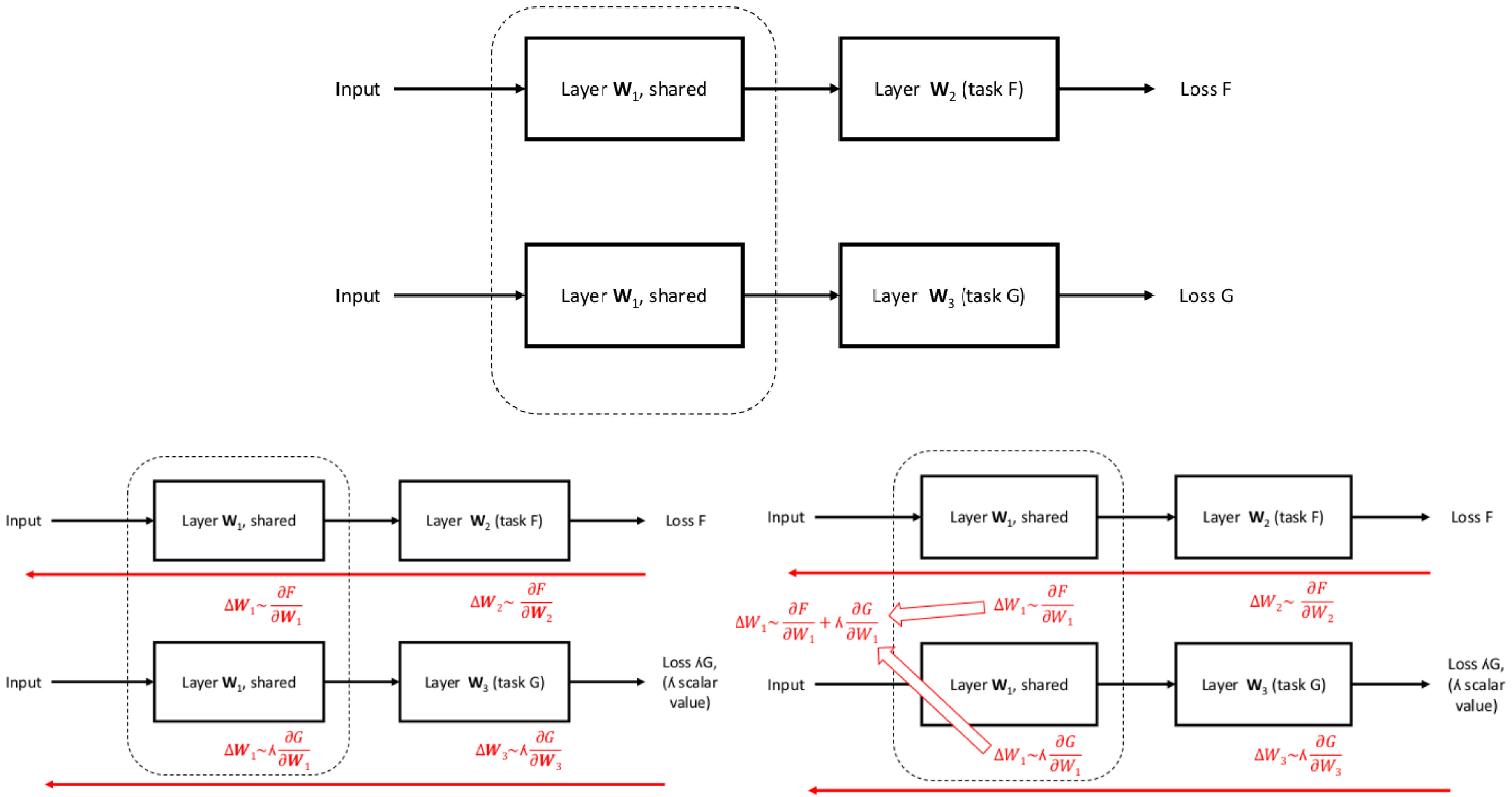
$$H(w) = F(w) + \lambda G(w)$$



Parameter Sharing

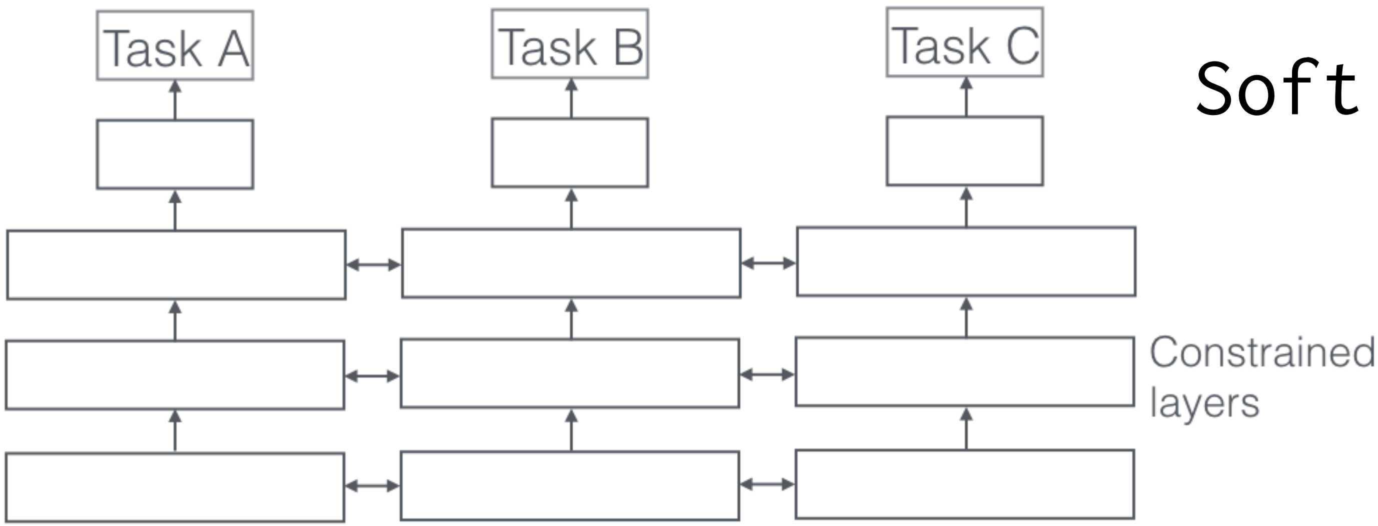


Parameter Sharing

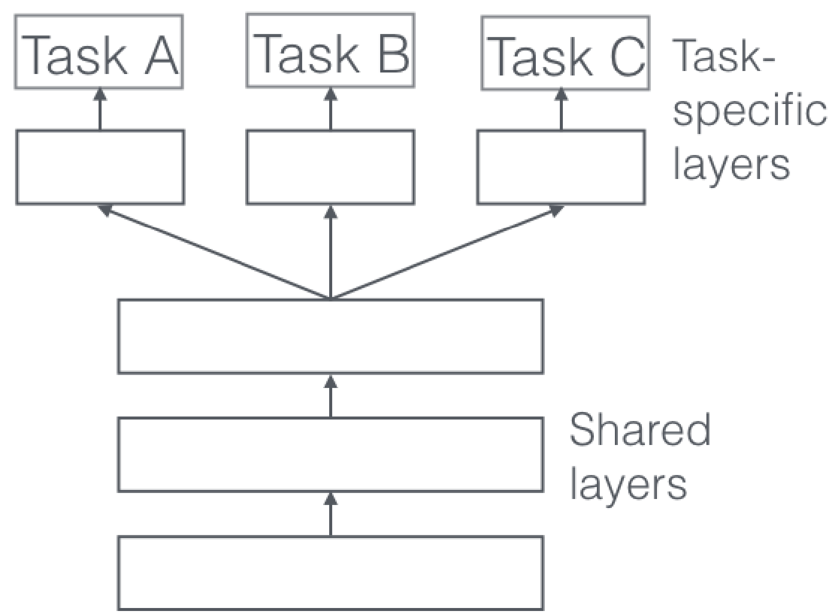


Parameter Sharing

Soft



Hard:



Why does MTL work?

* Implicit data augmentation
* Attention focusing
* Eavesdropping
* Representation bias
* Regularization

Multi-Task Datasets

* GLUE & SuperGLUE
  https://gluebenchmark.com/
* NLP Decathlon https://decanlp.com/

DecaNLP

Leaderboard

Rank	Model	decaScore	Breakdown by Task			
1 June 20, 2018	MQAN *Salesforce Research*	590.5	**SQuAD**	74.4	**QA-SRL**	78.4
			IWSLT	18.6	**QA-ZRE**	37.6
			CNN/DM	24.3	**WOZ**	84.8
			MNLI	71.5	**WikiSQL**	64.8
			SST	87.4	**MWSC**	48.7
2 May 18, 2018	Sequence-to-sequence baseline *Salesforce Research*	513.6	**SQuAD**	47.5	**QA-SRL**	68.7
			IWSLT	14.2	**QA-ZRE**	28.5
			CNN/DM	25.7	**WOZ**	84.0
			MNLI	60.9	**WikiSQL**	45.8
			SST	85.9	**MWSC**	52.4

Back to Transfer Learning

MTL: a common model trained for many problems.

TL: use a model trained on one task and adapt it to other tasks.

TL example: word embeddings transfer their knowledge to task-specific Neural Nets.

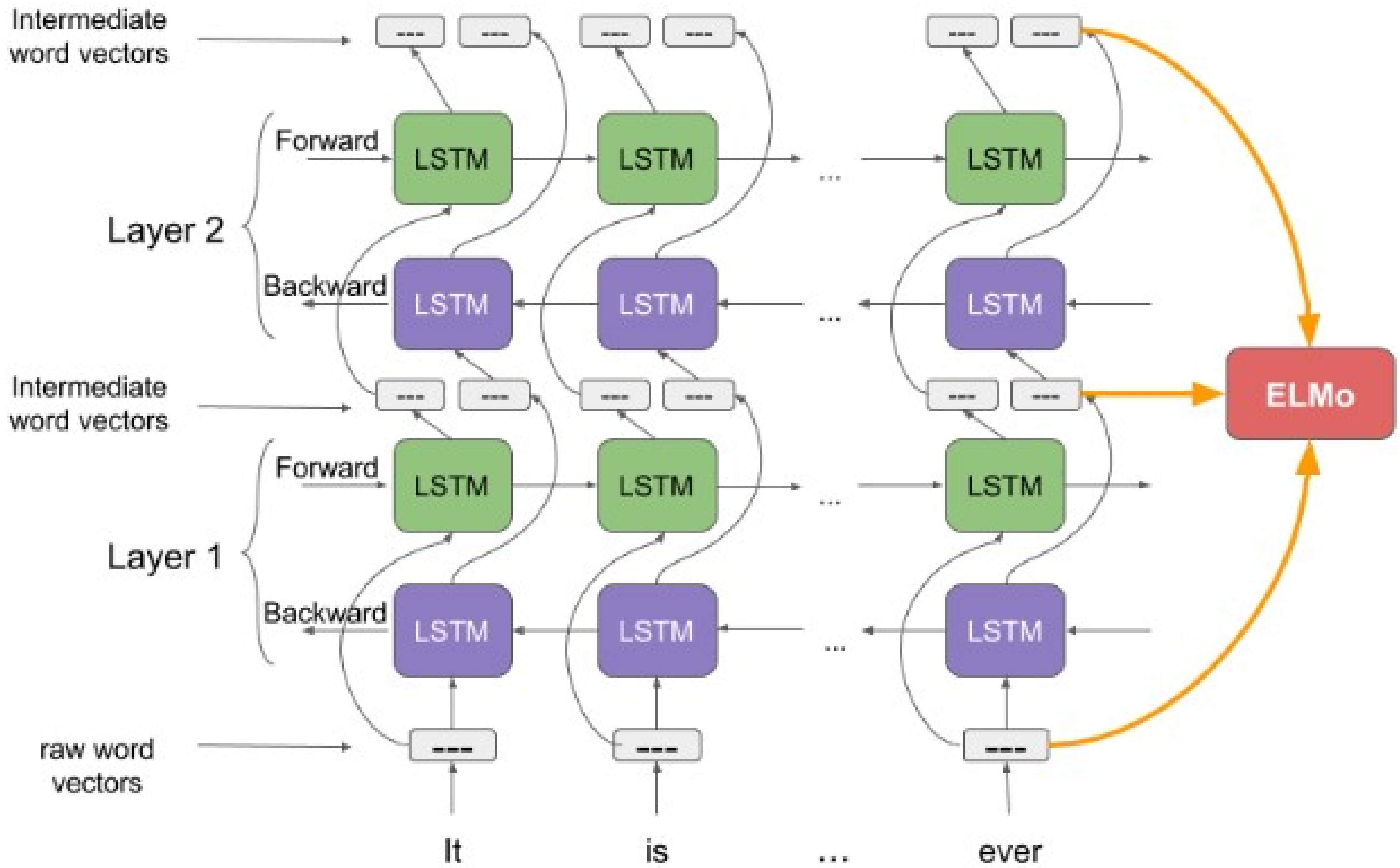
Good task for TL: LM.

ELMo

ELMo word vectors are computed on top of a two-layer biLM.

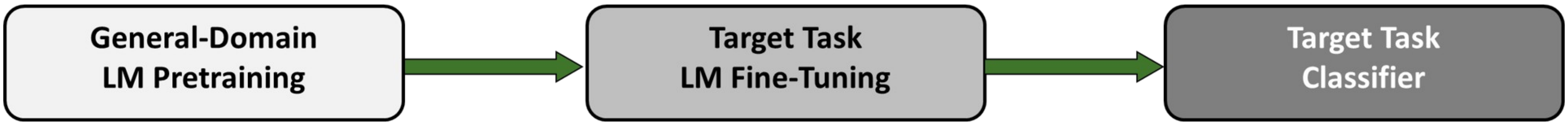
* The architecture uses a charCNN to represent words of a text string into raw word vectors
* The raw word vectors act as inputs to the first layer of biLM
* The intermediate word vectors are fed into the next layer of biLM
* The final representation is the weighted sum of the raw word vectors and the intermediate word vectors

ELMo



Intermediate word vectors
Forward
Layer 2
Backward
Intermediate word vectors
Forward
Layer 1
Backward
raw word vectors
LSTM
ELMo
It
is
...
ever

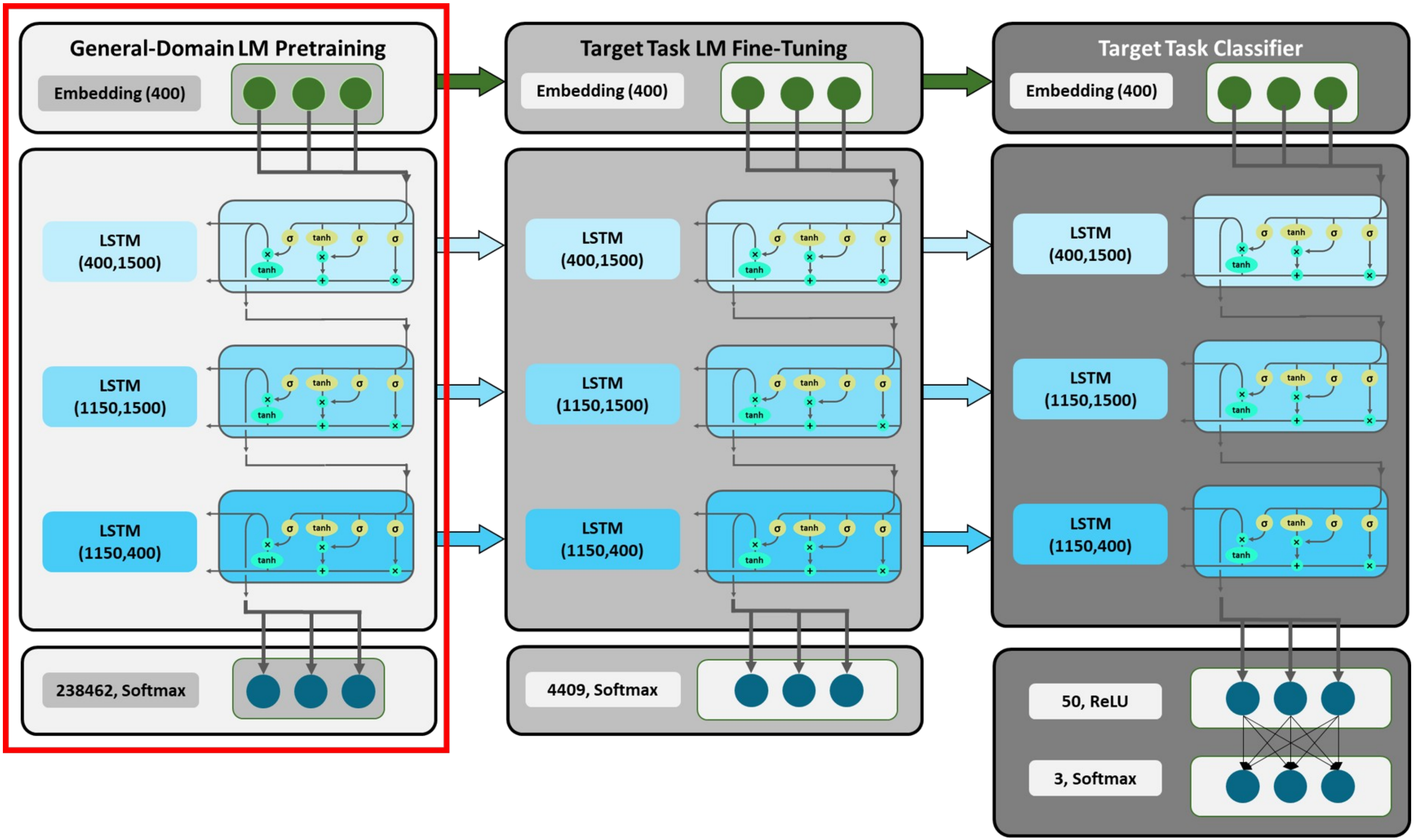
ULMFiT



Universal Language Model Fine-Tuning

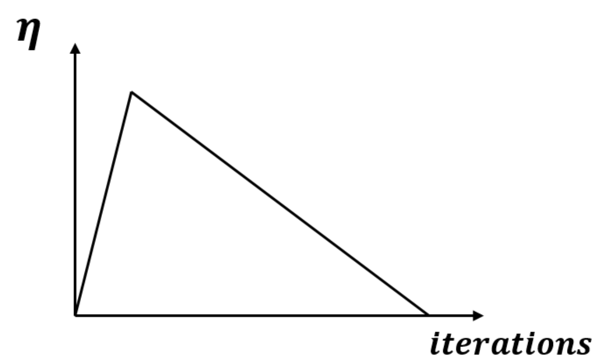
* General-Domain LM Pretraining: an LM is pretrained on a large general-domain corpus (WikiText-103). Figuratively speaking, at this stage the model learns the general features of the language, e.g. that the typical sentence structure of the English language is subject-verb-object.
* Target Task LM Fine-Tuning: train on the target task dataset (i.e. Sentiment analysis). The LM is consequently fine-tuned on the data of the target task.
* Target Task Classifier: since ultimately, in our case, we do not want our model to predict the next word in a sequence but to provide a classification, in a third step the pretrained LM is expanded by two linear blocks so that the final output is a probability distribution over the labels

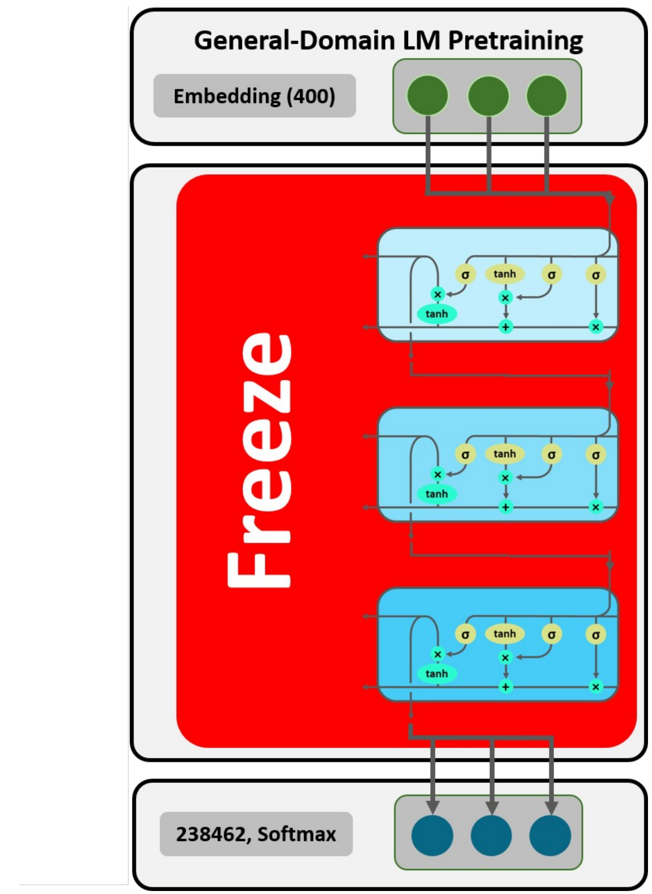
ULMFiT Example



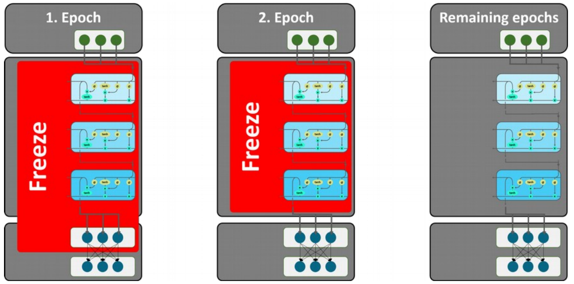
ULMFiT Tricks

* Freezing
* Slanted triangular learning rate schedule





* Discriminative fine-tuning
* Concat pooling (max+mean)
* Gradual unfreezing



seq2seq issues

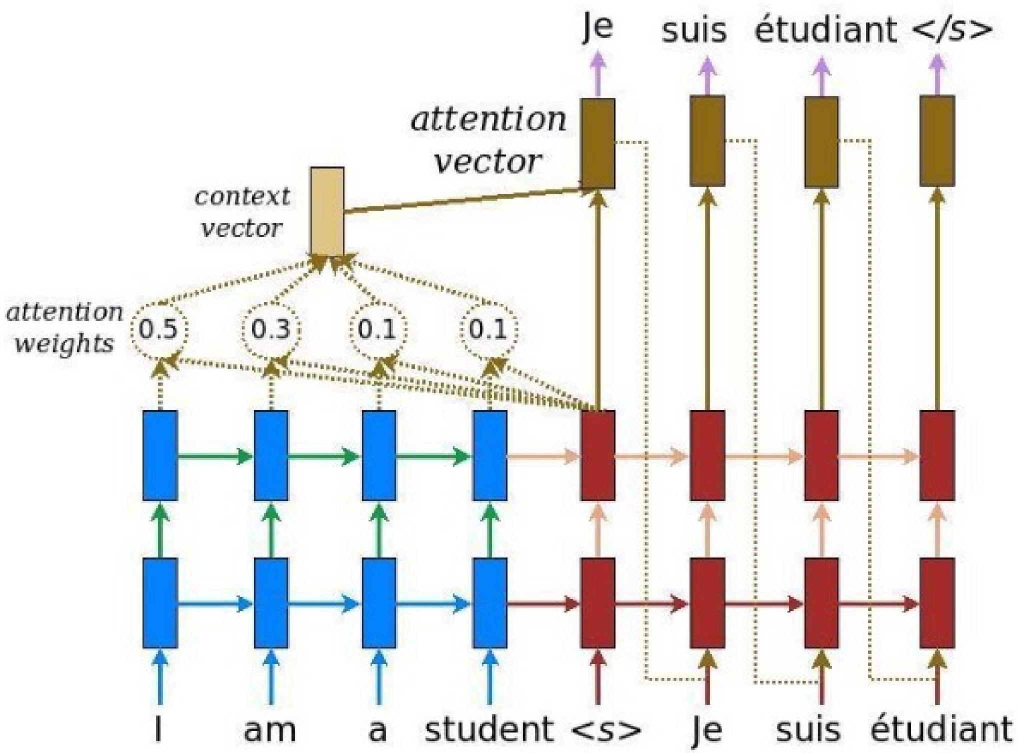
* input and output sequences may have different length
* order of words in sequence may differ
* complex relationships between input and output sequences: not necessarily one-to-one, may be many-to one, one-to-many, many-to-many

Another seq2seq issue

Numerical Complexity

Model	FLOPs
RNN	$O(length \cdot dim^2)$
1D ConvNet	$O(length \cdot dim^2 \cdot K)$

Solution: attention mechanism



- We introduce attention mechanism, at each time step k we have a set of "importance weights" $\mathbf{a}(k)$ for the whole sequence at encoder.
- Decoder uses weighted sum of all hidden states of encoder at each time step instead the only last.

$$A(q, \{(k, v)\}) \xrightarrow[\text{output}]{\text{maps as}} \sum_{i=1}^{k} \overbrace{f_c(q, k_i)}^{\theta_i} v_i, q \in Q, k \in K, v \in V$$

$$Q, K, V - vectorspace, f_c - compatibilityfunction$$

Attention algorithm

1. We have source sequence $\boldsymbol{x} = [x_1, \dots, x_n]$ and target output sequence $\boldsymbol{y} = [y_1, \dots, y_m]$.
2. The decoder network has hidden state

$$\boldsymbol{s_t} = f(\boldsymbol{s_{t-1}}, y_{t-1}, \boldsymbol{c_t})$$

3. We calculate:

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

Context vector

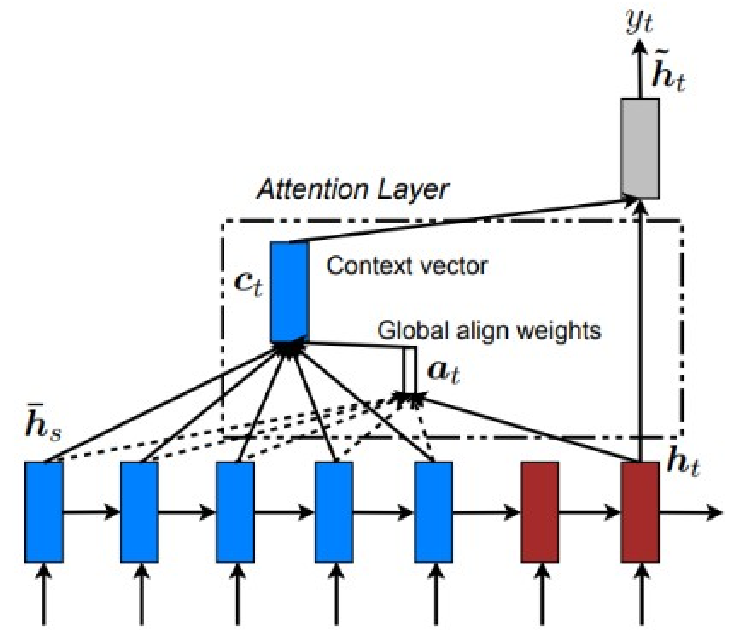
$$\alpha_{t,i} = \text{align}(y_t, x_i)$$

Alignment score

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

Trainable alignment model



Attention Mechanisms

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

Attention is more efficient

Numerical Complexity

Model	FLOPs
RNN	$O(length \cdot dim^2)$
1D ConvNet	$O(length \cdot dim^2 \cdot K)$
Self-attention	$O(length^2 \cdot dim)$

Attention algorithm – Query, Key, Value

source sequence $\boldsymbol{x} = [x_1, \dots, x_n]$ ~ **<Key, Value>**

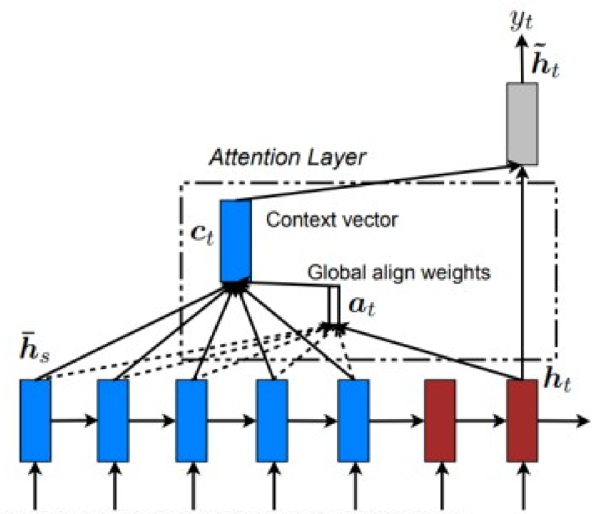
target sequence $\boldsymbol{y} = [y_1, \dots, y_m]$ ~ **<Query>**

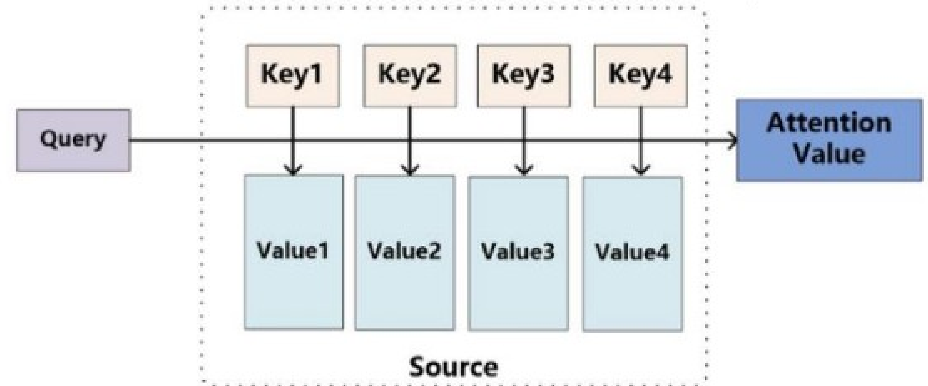
$\boldsymbol{h}_i$ - encoder's hidden state.

$$\boldsymbol{c}_t = \sum_{i=1}^{n} \alpha_{t,i}\, \boldsymbol{h}_i$$

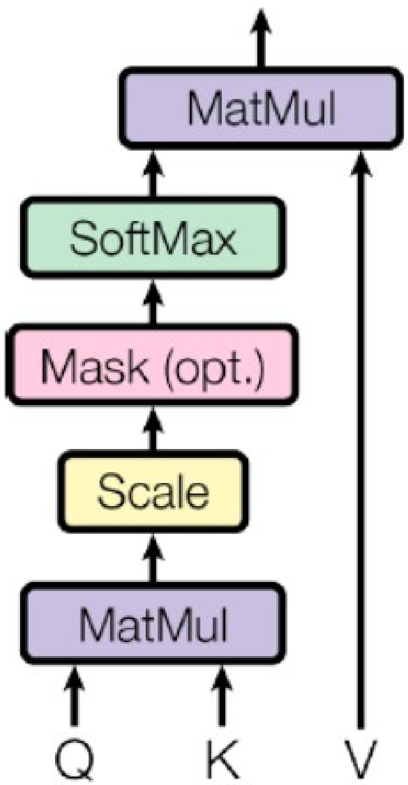
Context vector

$$Attention(Query_t, Source) = \sum_{i=1}^{n} \alpha_{t,i}\, Value_i$$





Scaled Dot-Product Attention

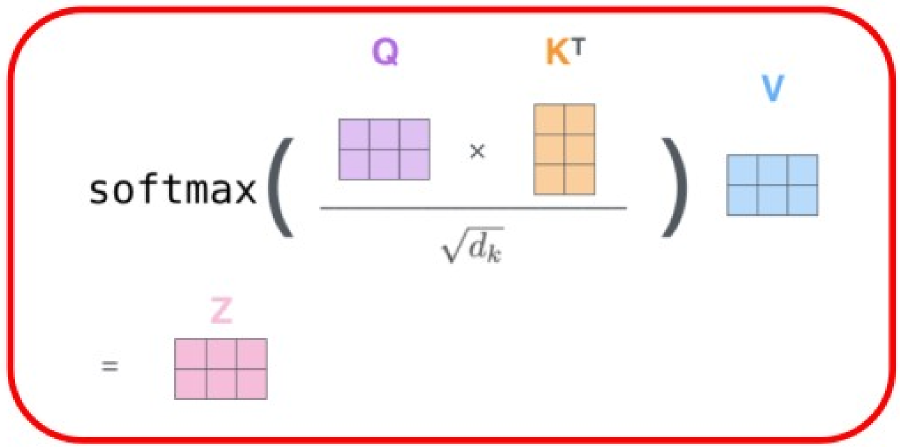


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

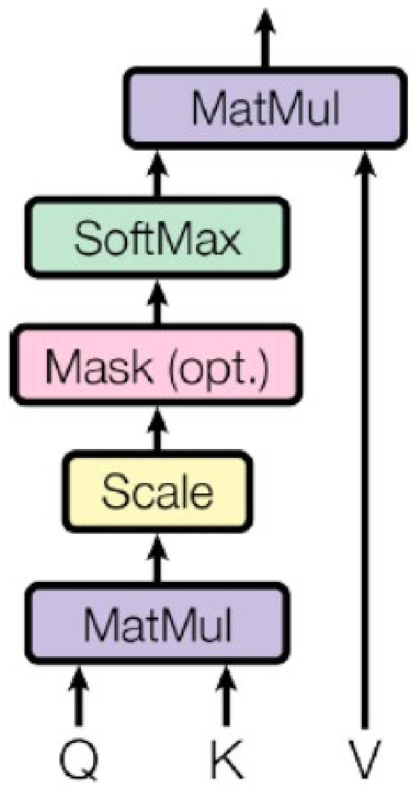
Query

Key = Value

In matrix form:



Scaled Dot-Product Attention

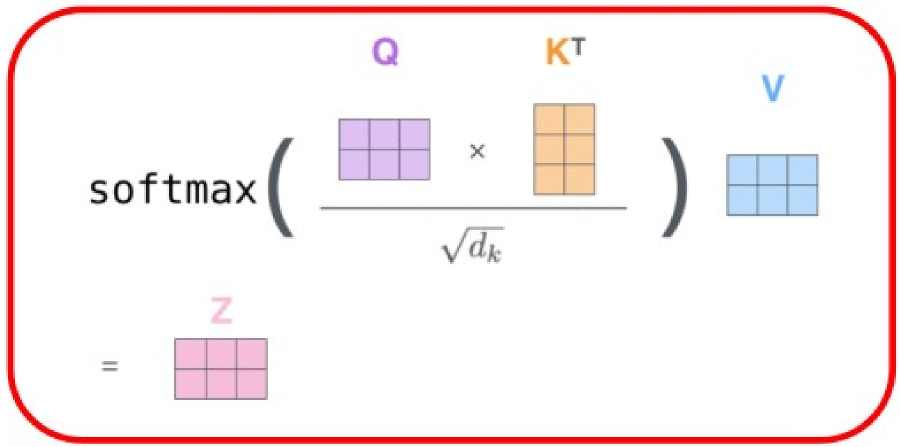


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Query

Key = Value

In matrix form:



Self-attention

Self-attention is a seq2seq operation over the input vectors $\mathbf{x}1, \mathbf{x}_2, \ldots \mathbf{x}_t$ and the corresponding output vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t$. The vectors all have dimension k.

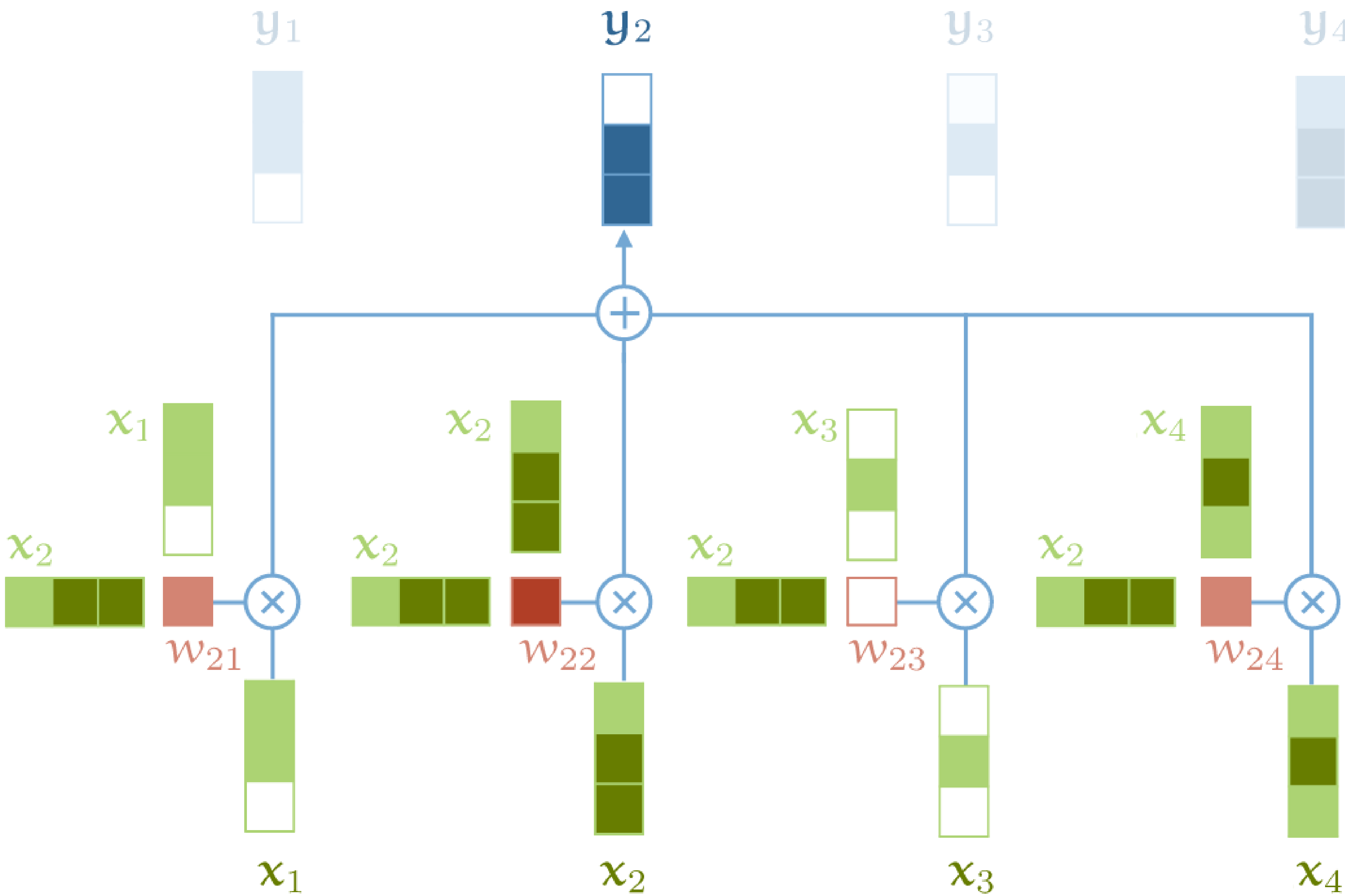
To produce output vector $\mathbf{y}_i$, the self attention operation simply takes a weighted average over all the input vectors:

$$\mathbf{y}i = \sum j w i j \mathbf{x} j$$

Where j indexes over the whole sequence and the weights sum to one over all j. The weight w_{ij} is not a parameter, as in a normal neural net, but it is derived from a function over $\mathbf{x}_i$ and $\mathbf{x}_j$. The simplest option for this function is the dot product: $w'_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$

Note that $\mathbf{x}_i$ is the input vector at the same position as the current output vector $\mathbf{y}_i$. For the next output vector, we get an entirely new series of dot products, and a different weighted sum (plus softmax to normalize).

Self-attention

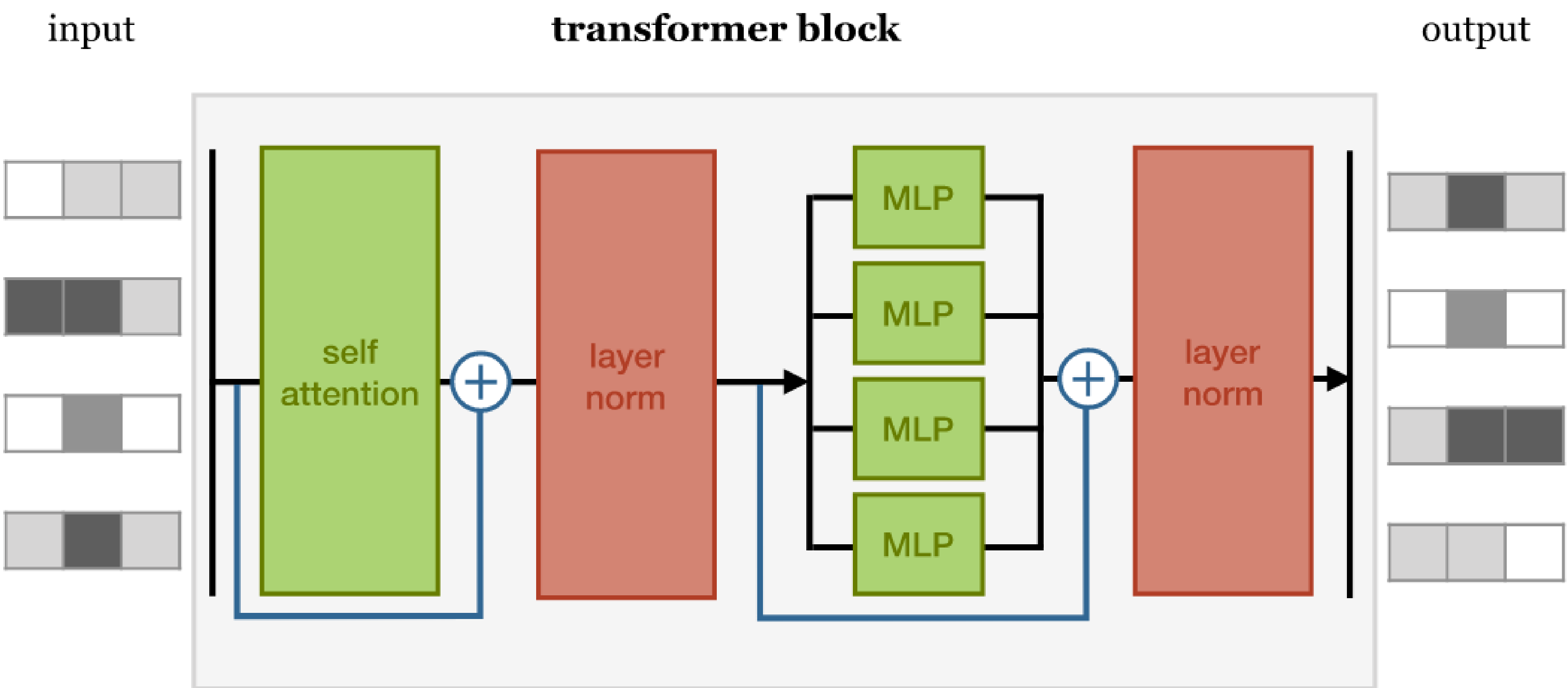


Narrow & wide self-attention

There are two ways to apply multi-head self-attention. The standard option is to cut the embedding vector into chunks: if the embedding vector has 256 dimensions, and we have 8 attention heads, we cut it into 8 chunks of 32 dimensions. For each chunk, we generate keys, values and queries of 32 dimensions each. This means that the matrices $\mathbf{W}^r_q$, $\mathbf{W}^r_k$, $\mathbf{W}^r_v$ are all 32×32.

We can also make the matrices 256×256, and apply each head to the whole size 256 vector. The first is faster, and more memory efficient but all else being equal, the second does give better results (at the cost of more memory and time).

Put self-attention in an NN



Transformer — neural network of stacked attention layers

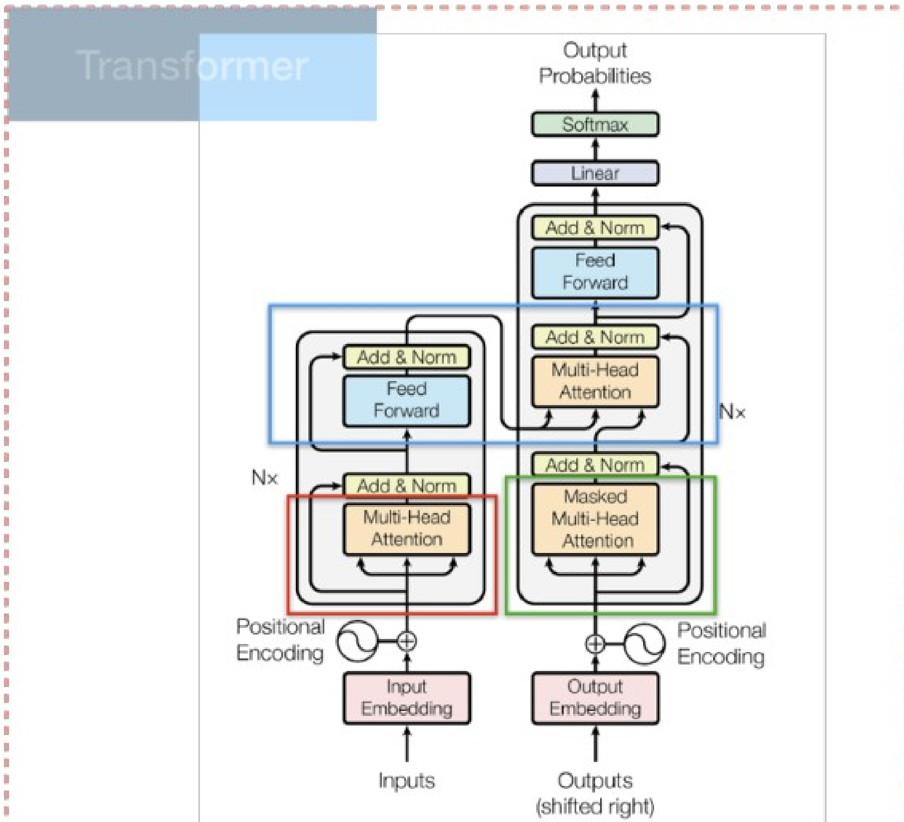


Figure 1: The Transformer - model architecture.

encoder self attention

1. Multi-head Attention
2. **Q**uery=**K**ey=**V**alue

decoder self attention

1. **Masked** Multi-head Attention
2. **Q**uery=**K**ey=**V**alue

encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=**K**ey=**V**alue
3. Decoder Self attention=**Query**

But, why do we need the Transformer?

RNN

Pros: are popular and successful for variable-length representations such as sequences (e.g. languages), images, etc. RNNs are considered the core of seq2seq (with attention). The gating models such as LSTM or GRU are for long-range error propagation
Cons: the sequentiality prohibits parallelization within instances. Long-range dependencies still tricky, despite gating. Sequence-aligned states in RNN are wasteful. Hard to model hierarchical-alike domains such as languages

CNN

Pros: Trivial to parallelize (per layer) and fit intuition that most dependencies are local.
Cons: Path length between positions can be logarithmic when using dilated convolutions, left-padding for text

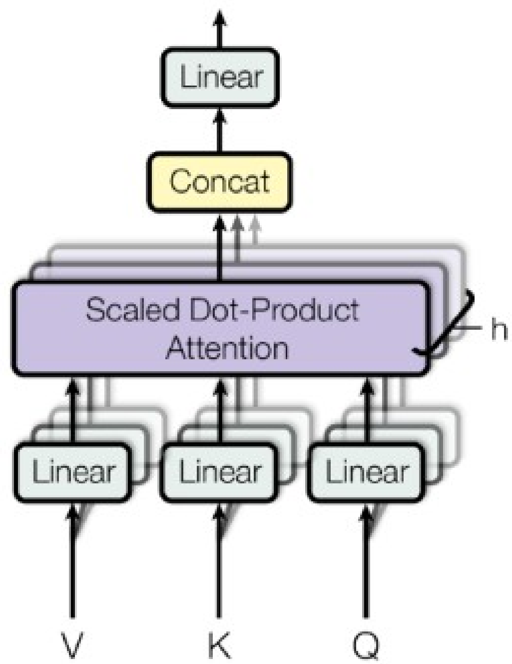
But, why do we need the Transformer?

Transformer achieves:

* Parallelization of seq2seq: RNN/CNN handle sequences word-by-word sequentially which is an obstacle to parallelize. Transformer achieves parallelization by replacing recurrence with attention and encoding the symbol position in the sequence. This, in turn, leads to significantly shorter training time.

* Reduce sequential computation: Constant O(1) number of operations to learn dependency between two symbols independently of their position distance in sequence.

Multi-Head Attention



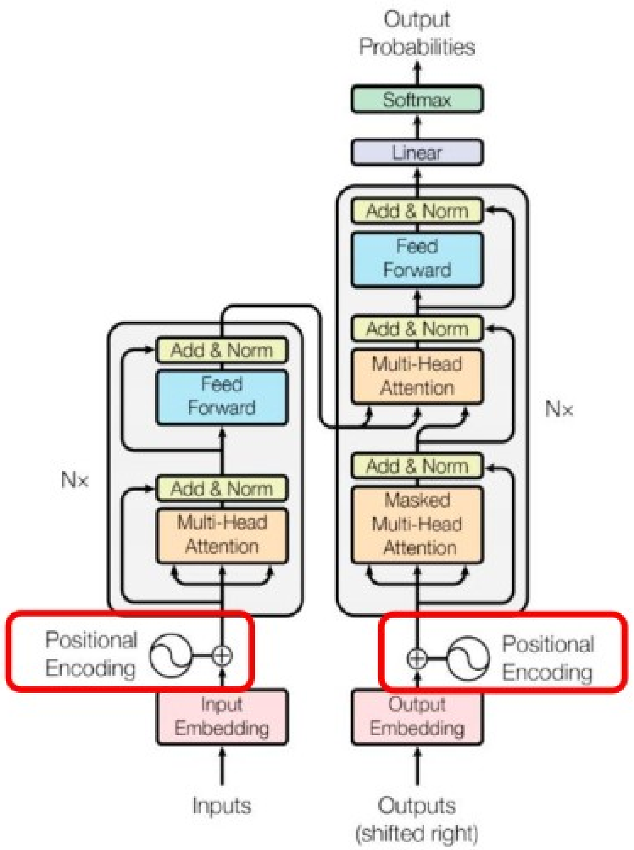
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Essentially, the Multi-Head Attention is just several attention layers stacked in parallel, with different linear transformations of the same input.

Positional Encodings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

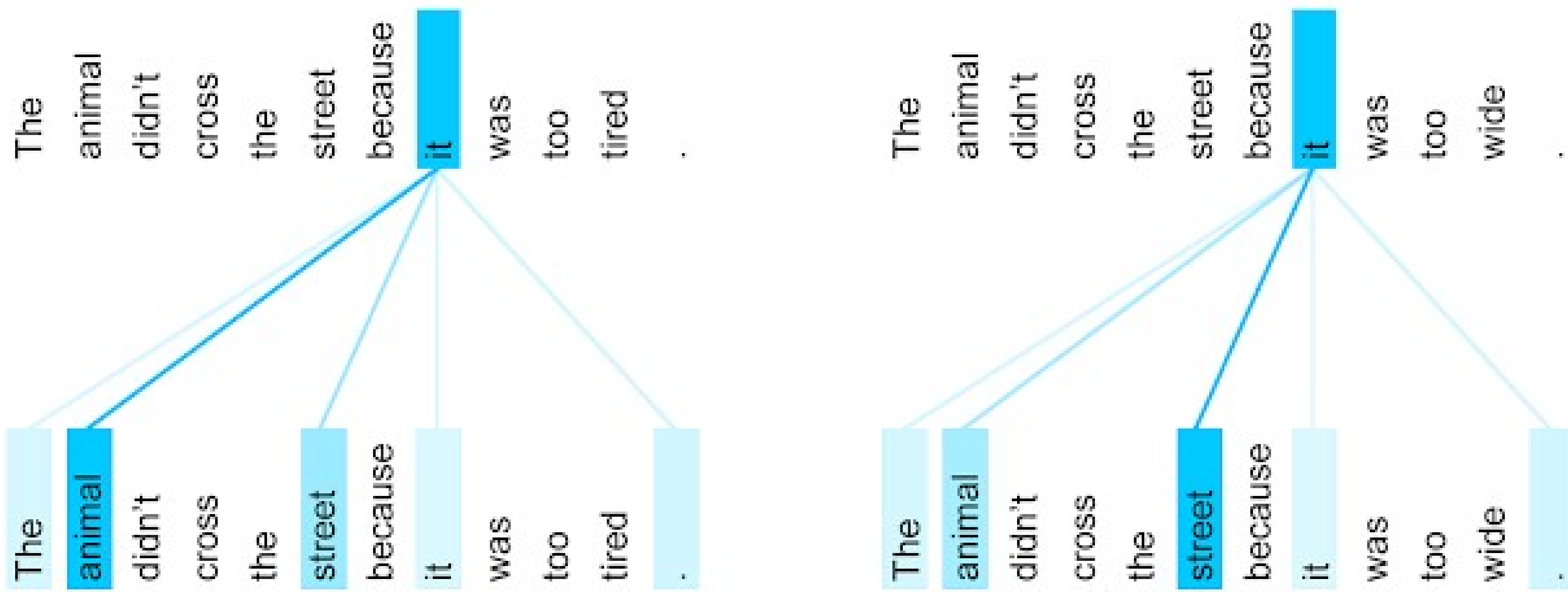
Goal: add to input embeddings information about the place and order of inputs



The wavelengths form a geometric progression from 2π to 10000·2π. They chose this function because they hypothesized it would allow the model to easily learn to attend by relative positions, since, for any fixed offset k, PE(pos+k) can be represented as a linear function of PE(pos).

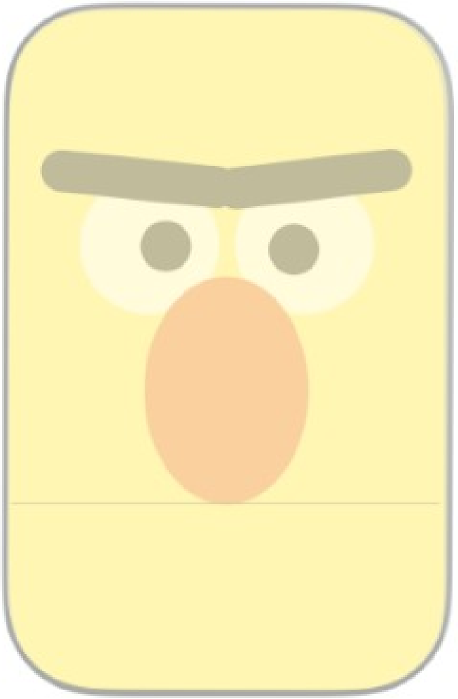
Important: residual connections!

BERT self-attention in action



The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

BERT



- BERT is designed to pre-train <u>deep bidirectional representations</u> by jointly conditioning on both left and right context in all layers
- pre-trained BERT representations can be fine-tuned with just <u>one additional output layer</u> to create SOTA models for a wide range of tasks

A small anecdote

The OpenAI transformer (GPT) gave us a fine-tunable pre-trained model based on the Transformer. But something went missing in this transition from LSTMs to Transformers. ELMo’s language model was bi-directional, but the openAI transformer only trains a forward language model.

Could we build a transformer-based model whose language model looks both forward and backwards (in the technical jargon — “is conditioned on both left and right context”)?

“Hold my beer”, said R-rated BERT. “We’ll use transformer encoders”.

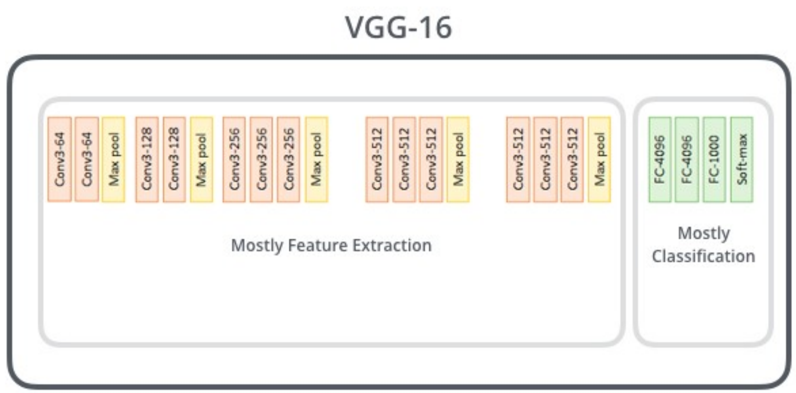
“This is madness”, replied Ernie, “Everybody knows bidirectional conditioning would allow each word to indirectly see itself in a multi-layered context.”

“We’ll use masks”, said BERT confidently.

Inspiration from CV

Input

Features



Output

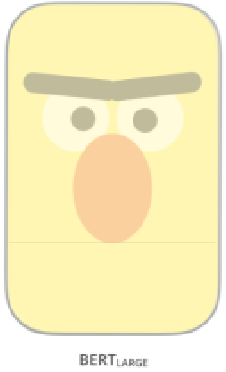
Prediction

0.2%	Kit fox
0.1%	English setter
95%	Egyptian cat
1%	Great Dane
	…
0%	Hotdog

Setup details

- BERT’s model architecture is a multi-layer bidirectional Transformer encoder.
- Parameters: the number of layers (i.e., Transformer blocks) as **L**, the hidden size as **H**, and the number of self-attention heads as **A.**

BERT_BASE: L=12, H=768, A=12
Total Parameters=110M



BERT_LARGE: L=24, H=1024, A=16
Total Parameters=340M

Pre-training task: Masked LM

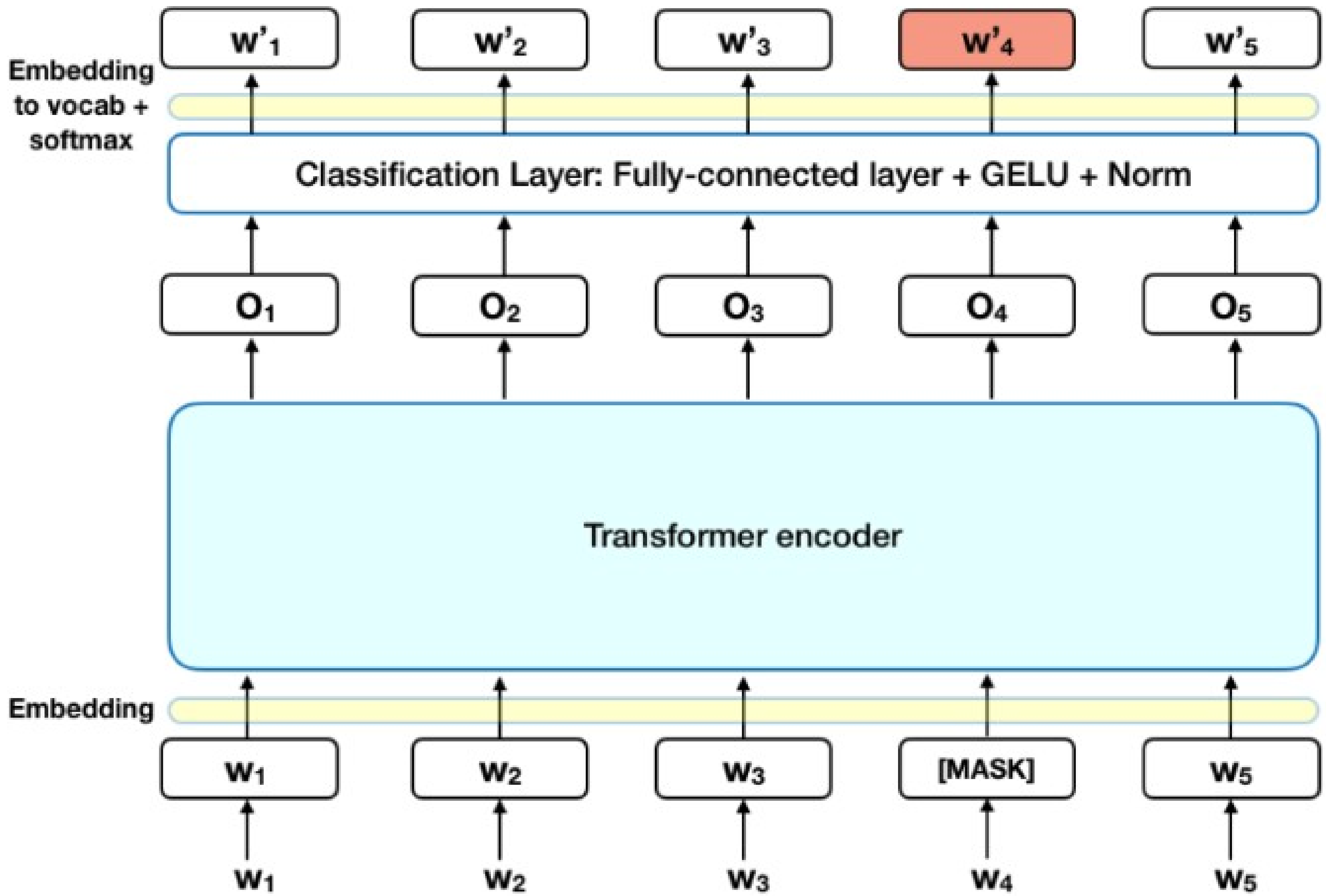
- Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token.

- 80% of the time: Replace the word with the [MASK] token,
  e.g., my dog is hairy → my dog is [MASK]

- 10% of the time: Replace the word with a random word,
  e.g., my dog is hairy → my dog is apple

- 10% of the time: Keep the word unchanged,
  e.g., my dog is hairy → my dog is hairy

Pre-training task: Masked LM



Pre-training task: Next Sentence Prediction

Input = `[CLS] the man went to [MASK] store [SEP]`

`he bought a gallon [MASK] milk [SEP]`

Label = `IsNext`

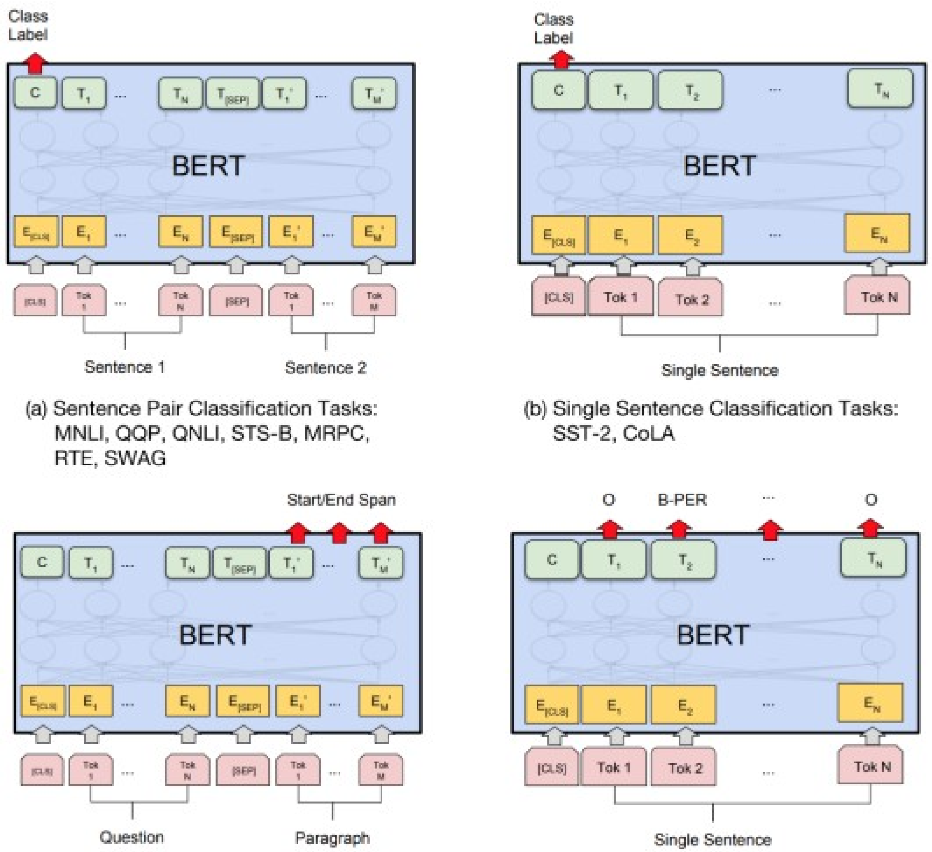
Input = `[CLS] the man [MASK] to the store [SEP]`

`penguin [MASK] are flight ##less birds [SEP]`

Label = `NotNext`

Model achieves 97%-98% accuracy on this task

Fine-tuning



(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

- Use the final hidden state (which corresponds to [CLS]) as sentence representation

- Batch size: 16, 32
- Learning rate (Adam): 5e-5, 3e-5, 2e-5
- Number of epochs: 3, 4

General Language Understanding Evaluation (GLUE)

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from `https://gluebenchmark.com/leaderboard` and `https://blog.openai.com/language-unsupervised/`.

Beyond BERT: RoBERTa

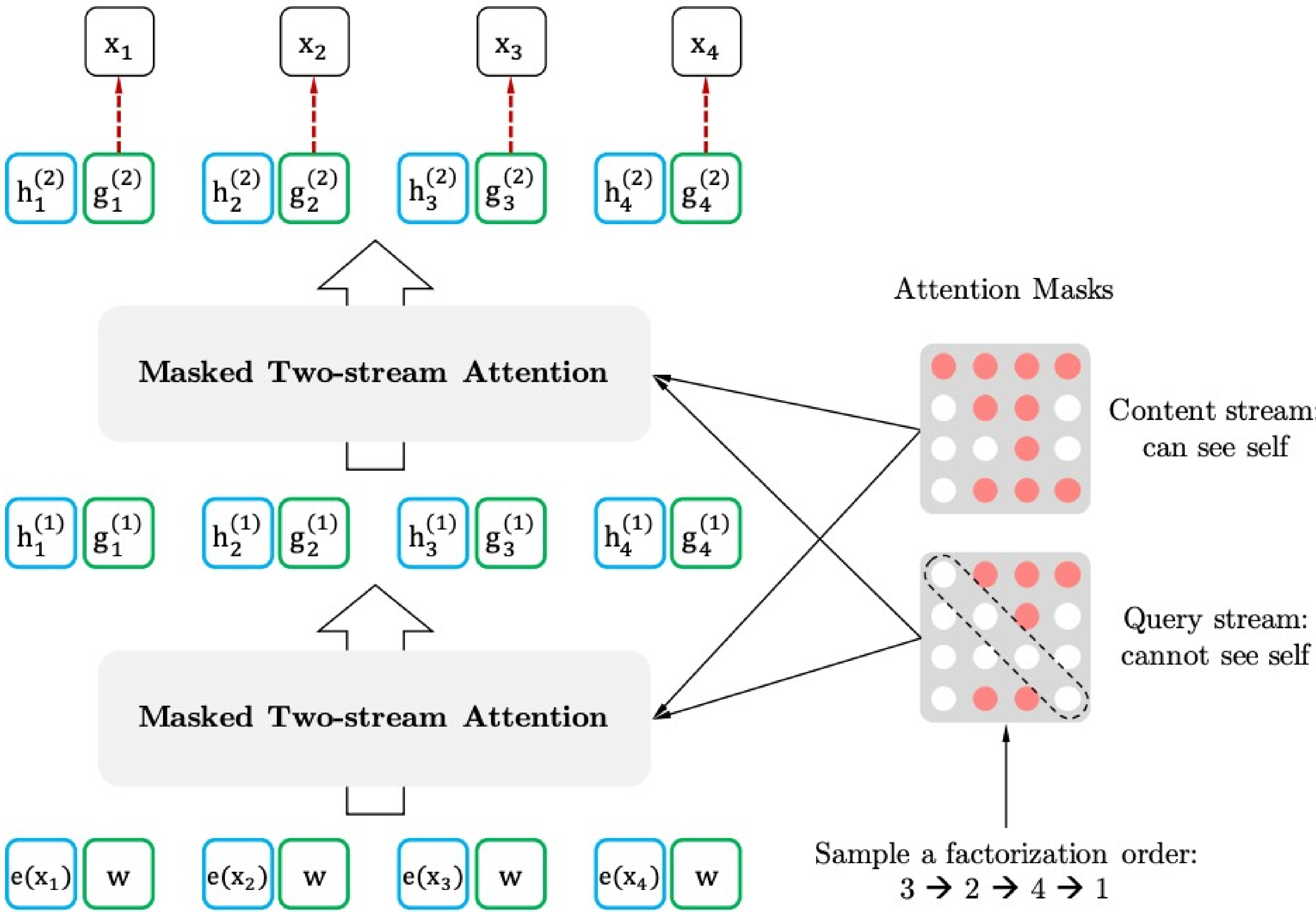
RoBERTa builds on BERT's language masking strategy, wherein the system learns to predict intentionally hidden sections of text within otherwise unannotated language examples. RoBERTa, which was implemented in PyTorch, modifies key hyperparameters in BERT, including removing BERT's next-sentence pretraining objective, and training with much larger mini-batches and learning rates. This allows RoBERTa to improve on the masked language modeling objective compared with BERT and leads to better downstream task performance. We also explore training RoBERTa on an order of magnitude more data than BERT, for a longer amount of time. We used existing unannotated NLP data sets as well as CC-News, a novel set drawn from public news articles.

Beyond BERT: XLNet

BERT is an auto-encoder (autoencoder, AE):
* Each hidden word is predicted individually. We lose information about the possible relationships between masked words (example: ”New York”)
* Inconsistency between the phases of training the BERT model and the use of the pre-trained BERT model([MASK] tokens)

XLNet is an autoregressive language modeling (AR LM). It is trying to predict the next token from the sequence of the previous ones. In classic autoregressive models, this contextual sequence is taken independently from two directions of the original string. XLNet generalizes this method and forms context from different places in the source sequence by taking all (in theory) possible permutations of the original sequence

XLNet



x_1
x_2
x_3
x_4
$h_1^{(2)}$ $g_1^{(2)}$ $h_2^{(2)}$ $g_2^{(2)}$ $h_3^{(2)}$ $g_3^{(2)}$ $h_4^{(2)}$ $g_4^{(2)}$
Attention Masks
Masked Two-stream Attention
Content stream:
can see self
$h_1^{(1)}$ $g_1^{(1)}$ $h_2^{(1)}$ $g_2^{(1)}$ $h_3^{(1)}$ $g_3^{(1)}$ $h_4^{(1)}$ $g_4^{(1)}$
Query stream:
cannot see self
Masked Two-stream Attention
$e(x_1)$ w $e(x_2)$ w $e(x_3)$ w $e(x_4)$ w
Sample a factorization order:
$3 \rightarrow 2 \rightarrow 4 \rightarrow 1$

Beyond BERT: Reformer

“The Efficient Transformer”
2 techniques to improve the efficiency of Transformers:

* replace dot-product attention by one that uses locality-sensitive hashing, changing its complexity from O(L^2) to O(L logL) (L - input length)

* use reversible residual layers instead of the standard residuals, which allows storing activations only once in the training process instead of N times (N - number of layers)

Recap

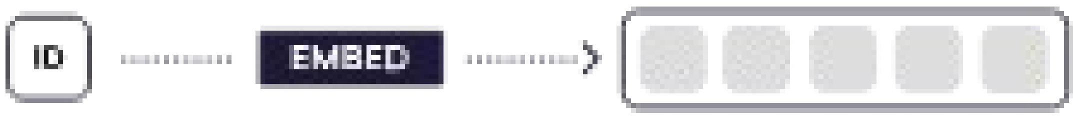
* transfer learning more “technological” than multi-task
* approaches to TL:
  - contextual embeddings
  - RNN pretraining
  - transformer pretraining
* attention is the solution to seq2seq issues
* transformer - stacked attention

The “DL Formula”

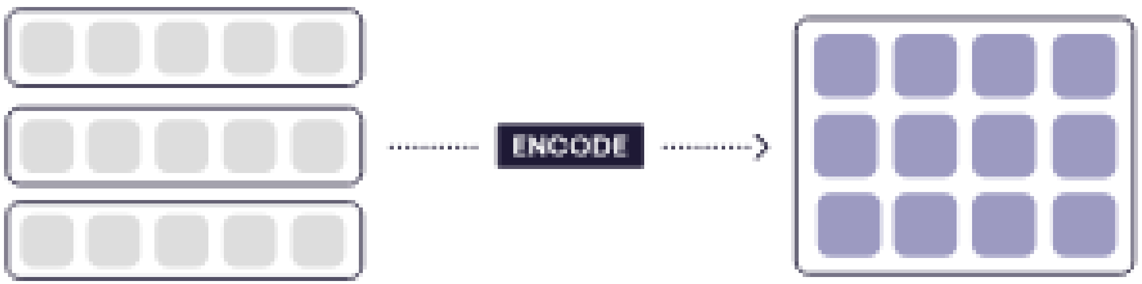
https://explosion.ai/blog/deep-learning-formula-nlp

Embed, encode, attend, predict

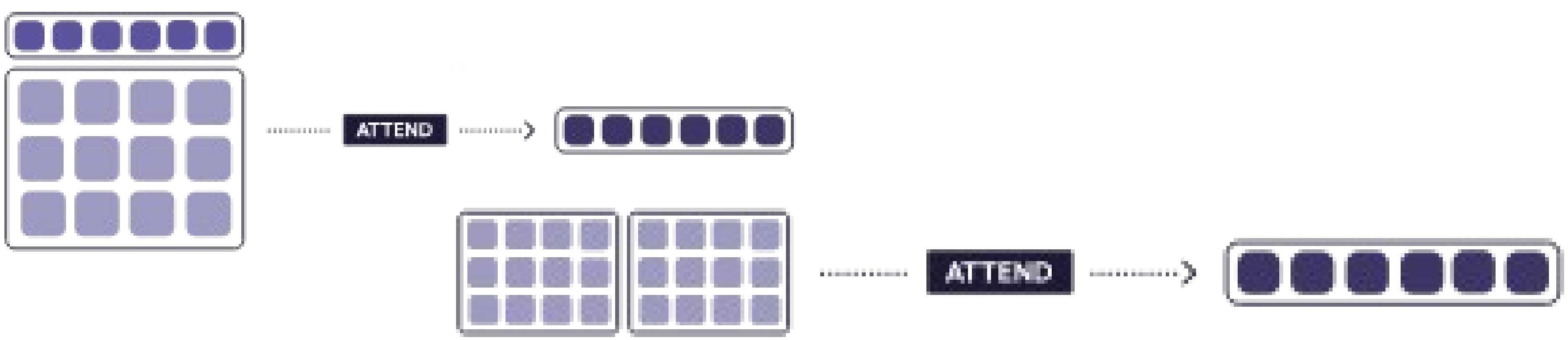
1.



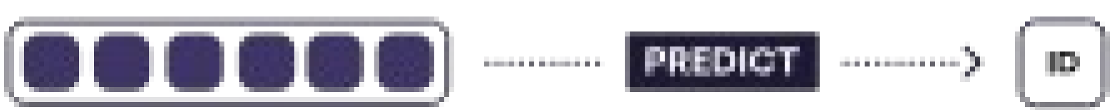
2.



3.



4.



Read More

ELMo:

https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

ULMFiT:

https://humboldt-wi.github.io/blog/research/information_systems_1819/group4_ulmfit/

https://medium.com/mlreview/understanding-building-blocks-of-ulmfit-818d3775325b

Attention & Transformers:

http://nlp.seas.harvard.edu/2018/04/03/attention.html

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

https://medium.com/@adityathiruvengadam/transformer-architecture-attention-is-all-you-need-aeccd9f50d09

http://www.peterbloem.nl/blog/transformers

BERT et al:

http://jalammar.github.io/illustrated-bert/

https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1