Transformers and CNNs

Week 8 - CS6101 Deep Learning for NLP @ NUS SoC

Lim How Khang
Takanori Aoki

Attention is all you need

Key ideas of paper (https://arxiv.org/abs/1706.03762):

1. **Multi-head attention** (fully)
2. Batch with masking (related to above)
3. Sinusoidal positional encoding (jury is still out on this)

Implementation tricks:

Byte-Pair Encodings, Layer Normalization, Residual Connections, Adam (with warm start), Label Smoothing, Checkpoint Averaging, Decoder Beam Search and Length Penalty

Class Poll Results

Polly APP 2:16 PM

@howkhang has a poll for you!

Will attention eventually replace recurrence in deep learning for NLP?

1

2

1: Yes - it's already taking place.

29% (4)

@mohit, @ad, @garygsw, @zhangzn

2: No - recurrence is established.

71% (10)

@Louis Tran, @aceofspade, @Yang Chen, @Eric_Vader, @Nan, +5 more

Total votes

14

Resources

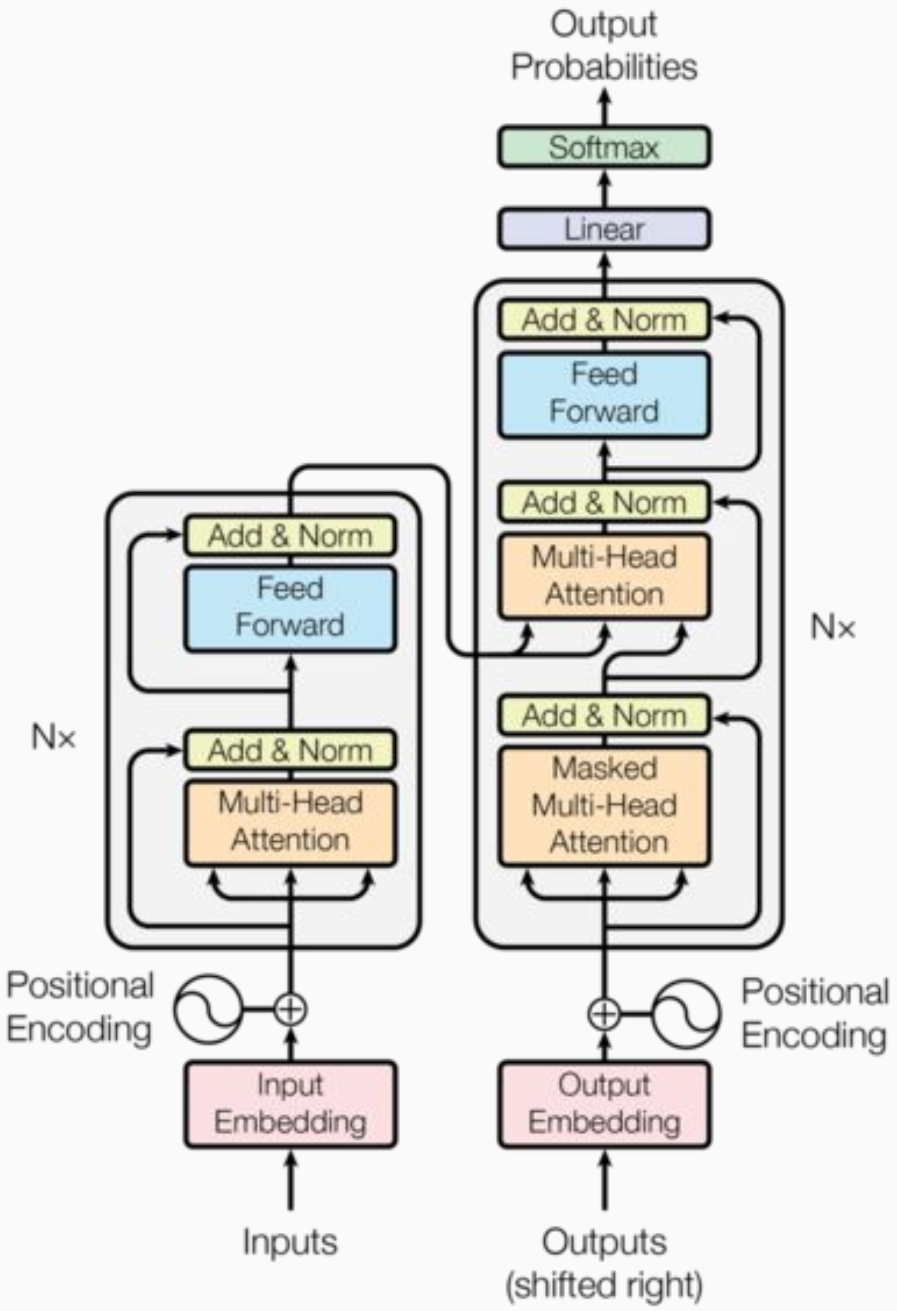
- Jay Alammar's Illustrated Transfomer
  http://jalammar.github.io/illustrated-transformer/
- Alexander Rush's Annotated Transformer
  http://nlp.seas.harvard.edu/2018/04/03/attention.html
- Michal Chromiak's post
  https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/
- Google AI Blog
  https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- Building the Mighty Transformer for Sequence Tagging with Pytorch
  https://medium.com/@kolloldas/building-the-mighty-transformer-for-sequence-tagging-in-pytorch-part-i-a1815655cd8

Encoder-Decoder Architecture

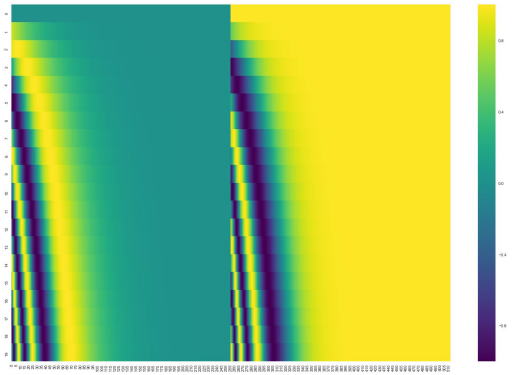
Most competitive neural sequence transduction models have an encoder-decoder structure (Vaswani et al, 2017).

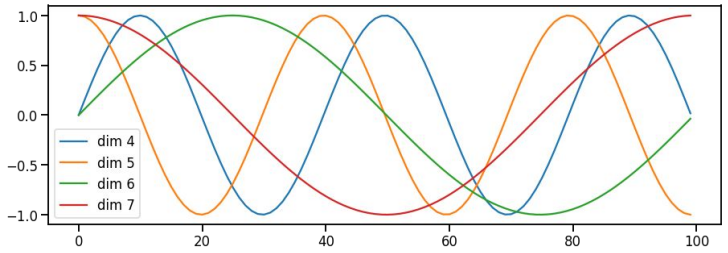
The encoder is composed of a stack of N=6 identical layers, each with two sub-layers: a multi-head self-attention mechanism, and a simple, position-wise fully connected feed-forward network. There is a residual connection around each of the two sub-layers, followed by layer normalization.

The decoder is also composed of a stack of N=6 identical layers. The decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. The self-attention sub-layer in the decoder stack has masking to prevent positions from attending to subsequent positions.



Positional Encoding



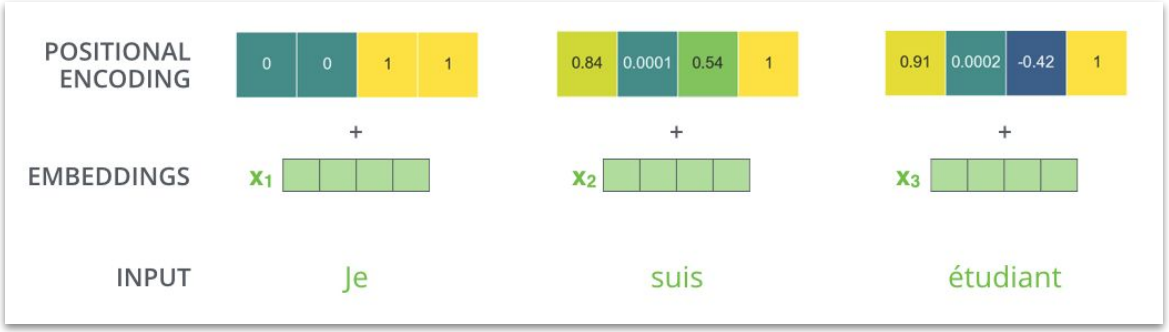


Credit: http://nlp.seas.harvard.edu/2018/04/03/attention.html

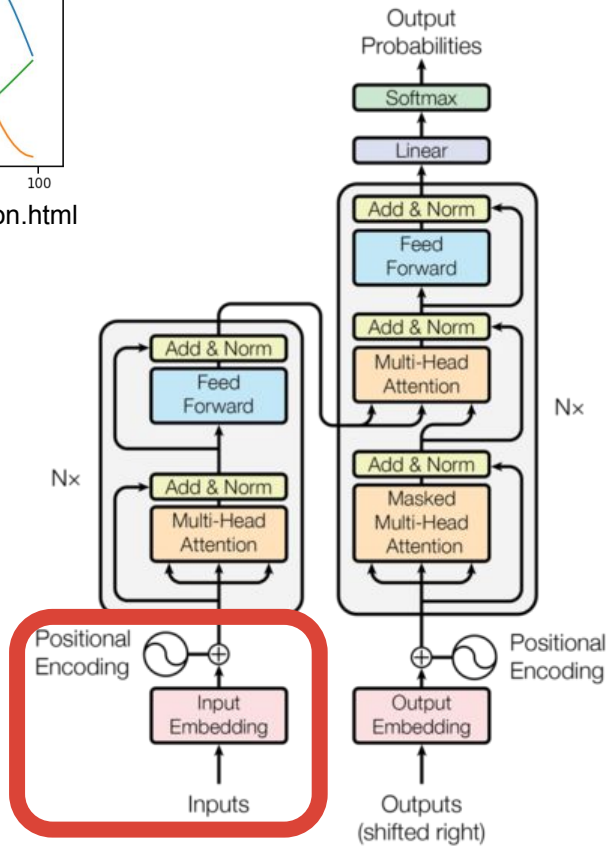
In this work, we use sine and cosine functions of different frequencies:

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

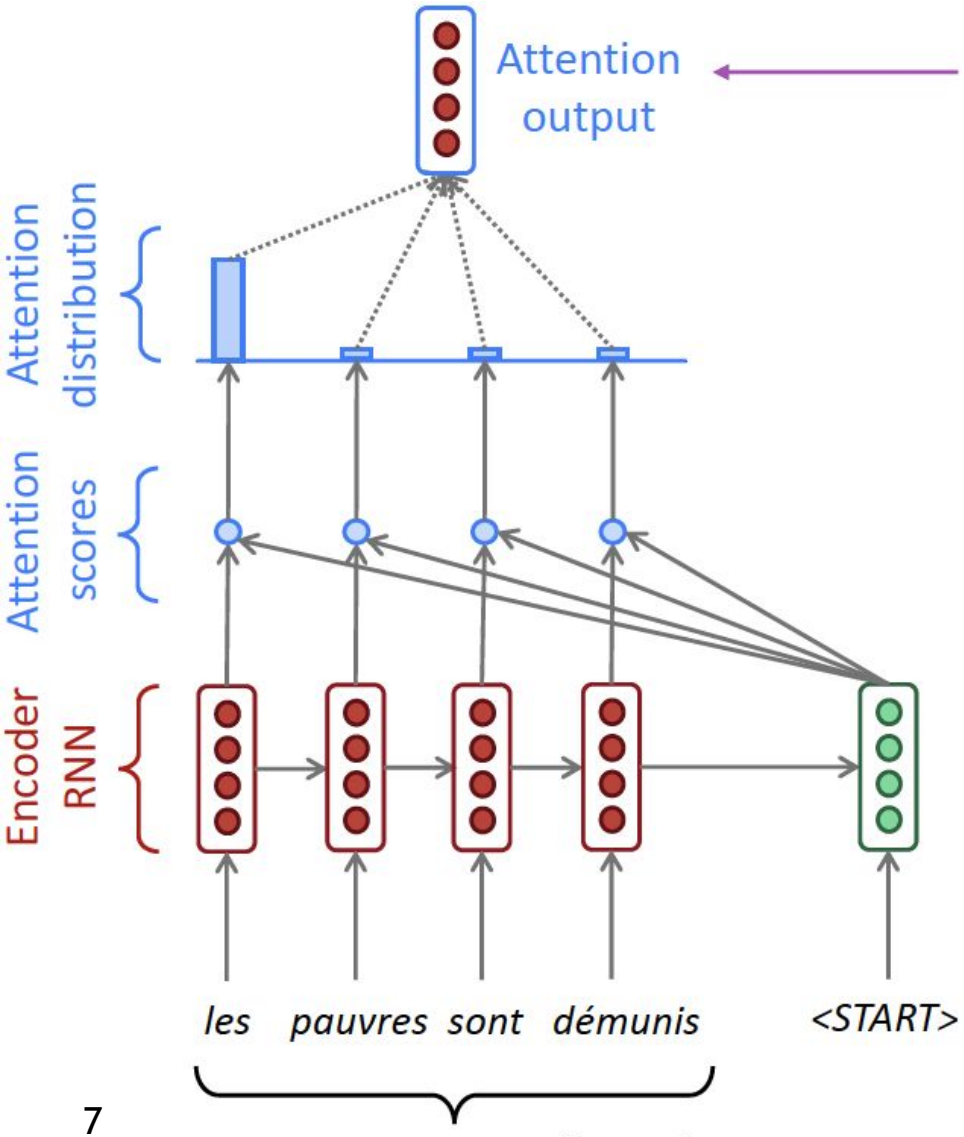
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$ where pos is the position and i is the dimension.



Credit: http://jalammar.github.io/illustrated-transformer/



Seq2Seq with Attention (Recap)



Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information the hidden states that received high attention.

Decoder RNN

Encoder-Decoder Attention (Recap)

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

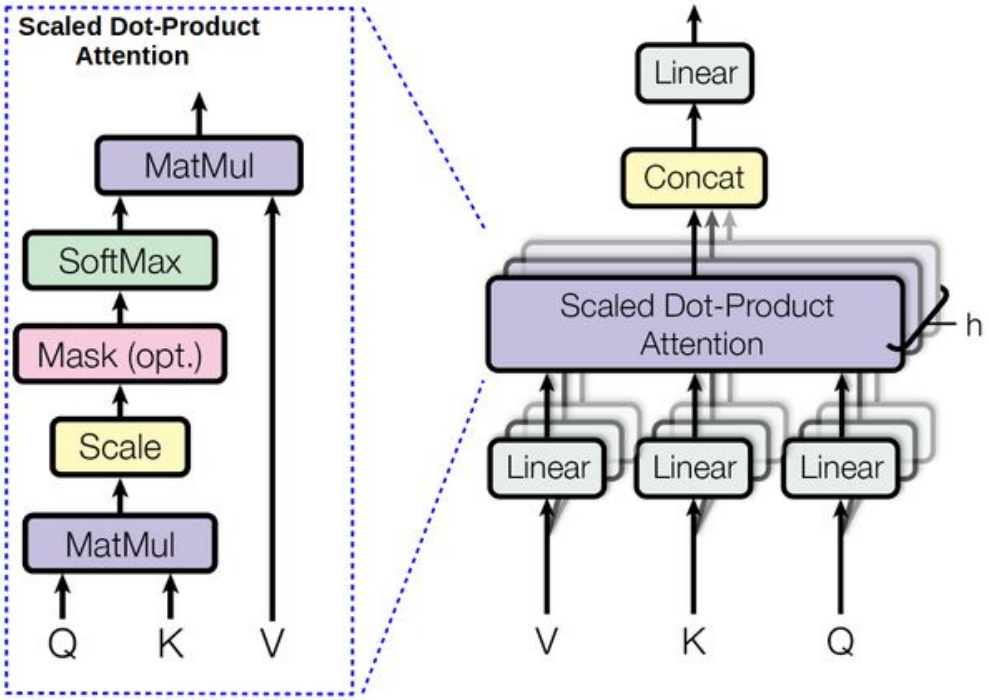
$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

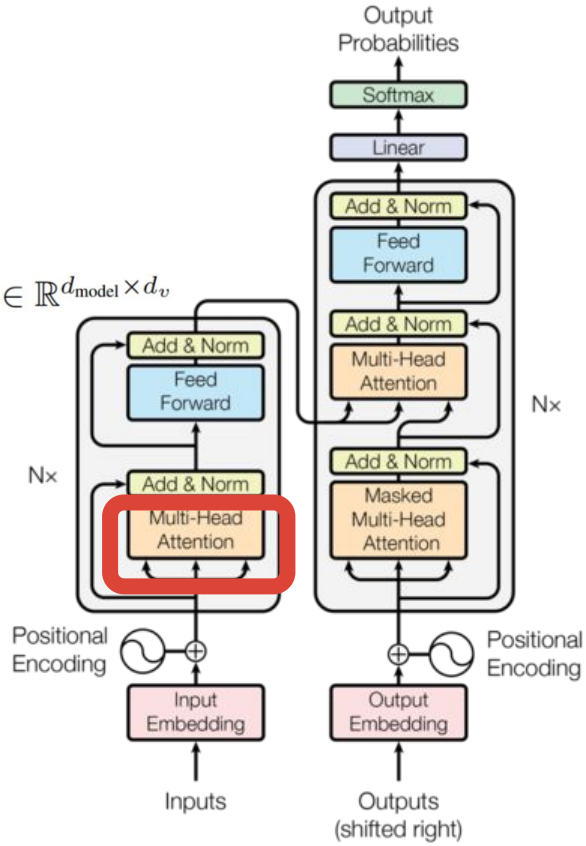
Multi-Head Attention

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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.





Credit: https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/

Multi-Head Attention

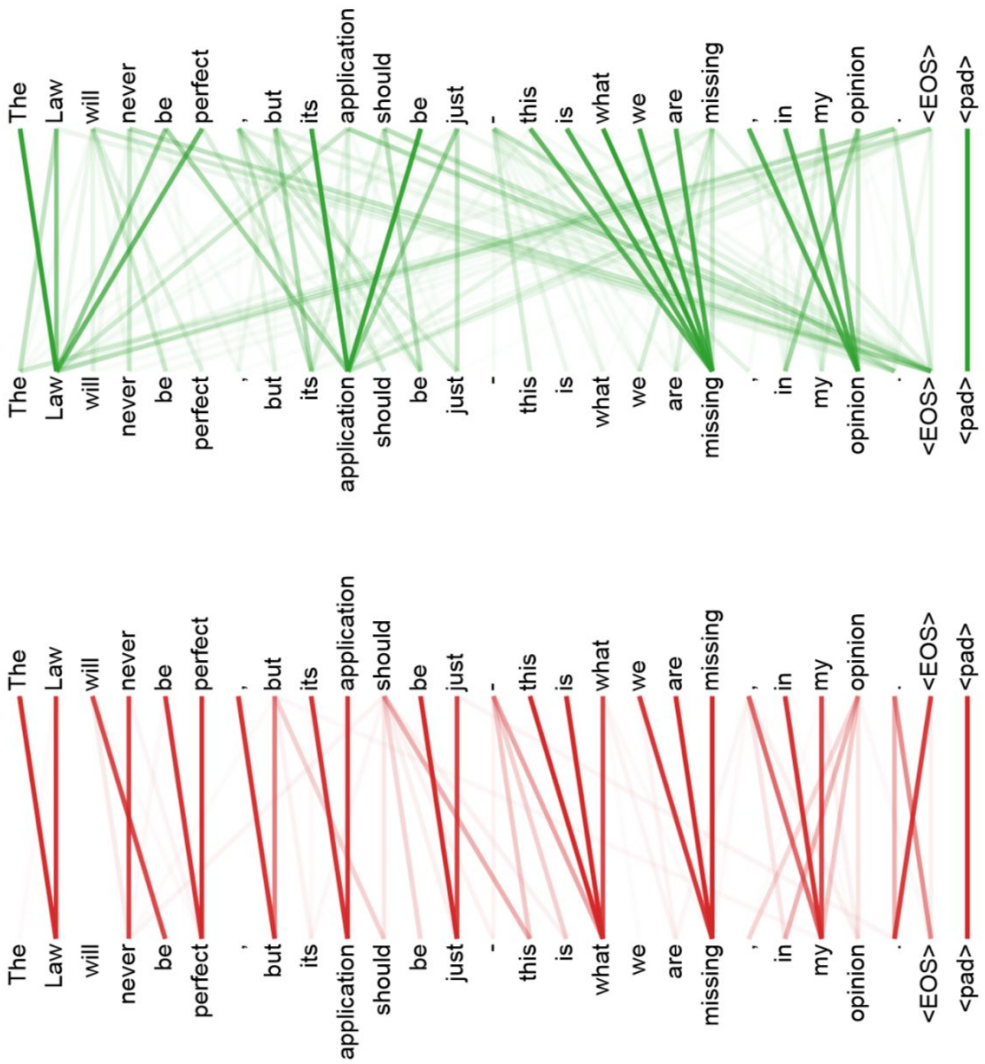
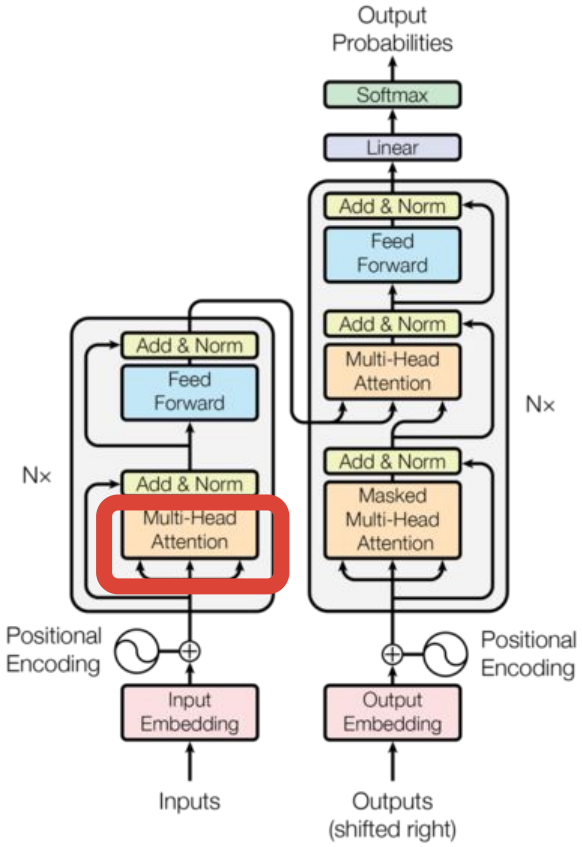
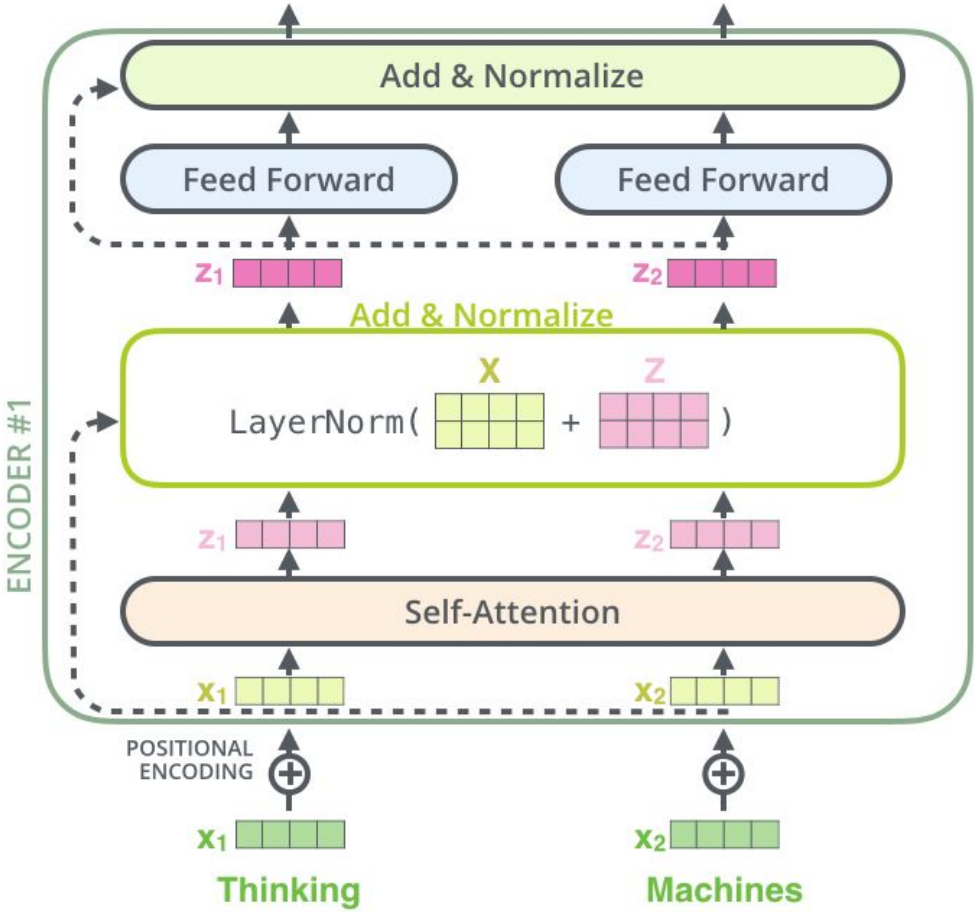


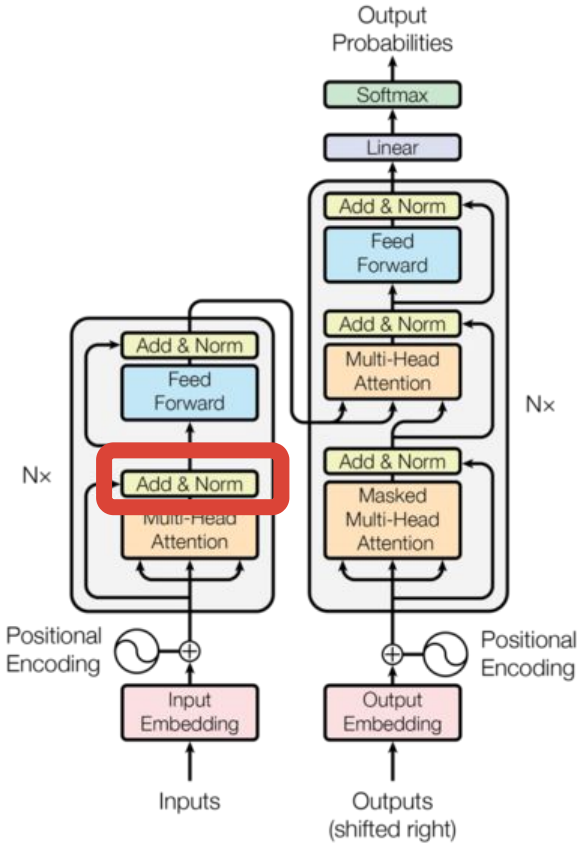
Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.



Layer Normalization + Residual Connections

$$\tilde{z}_{n,j} = \gamma \frac{z_{n,j} - \mathbb{E}[z_n]}{\sqrt{\frac{1}{|L(j)|}(z_{n,j} - \mathbb{E}[z_n])^2}} + \beta \qquad \mathbb{E}[z_n] = \frac{1}{|L(j)|} \sum_{k \in L(j)} z_{n,k}$$





Layer Normalization, Ba et al (2016) https://arxiv.org/pdf/1607.06450.pdf

Credit: http://jalammar.github.io/illustrated-transformer/

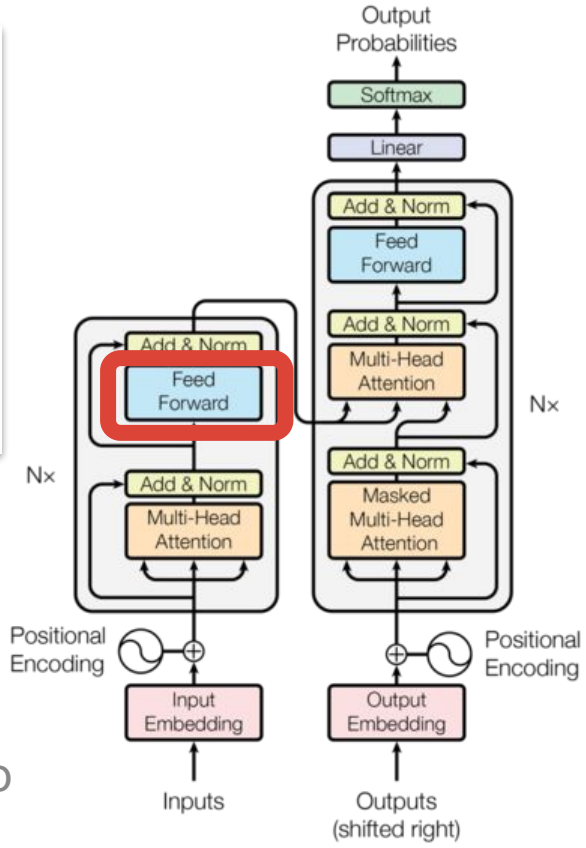
Position-wise Feed Forward Network

3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2)$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

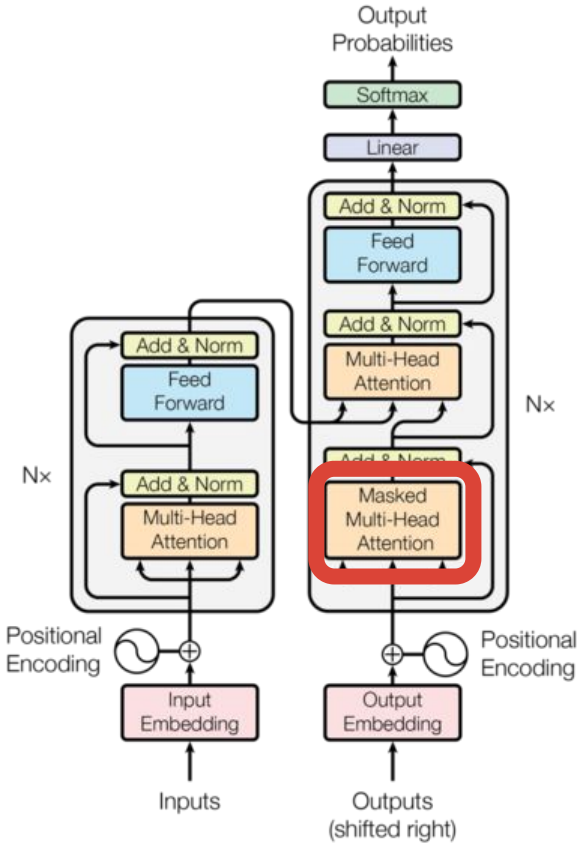


Why do we need a FFN when we already have attention?

How is this position-wise?

How is the FFN described in formula (2) the same as two convolutions with kernel size 1?

Decoder Masked Attention



Credit: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

More Recent Works

Constituency Parsing with a Self-Attentive Encoder (Kitaev and Klein, 2018)

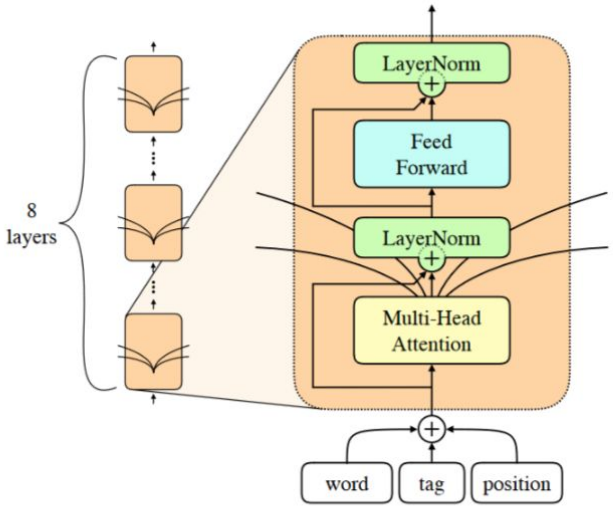


Figure 2: An overview of our encoder, which produces a context-aware summary vector for each word in the sentence. The multi-headed attention mechanism is the only means by which information may propagate between different positions in the sentence.

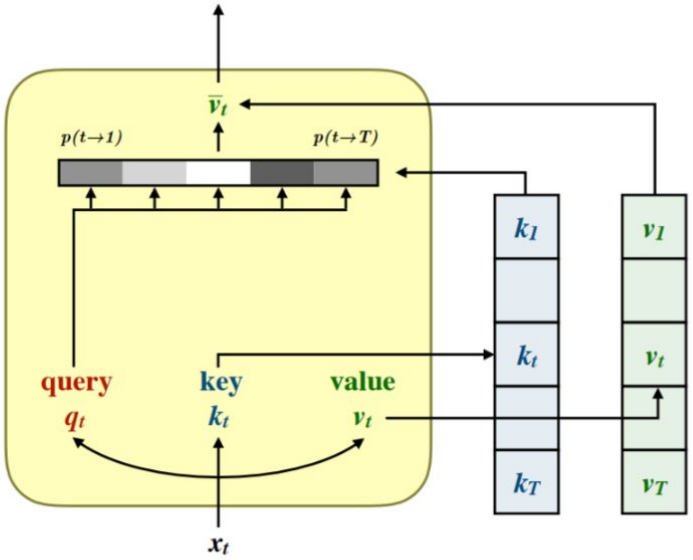


Figure 3: A single attention head. An input x_t is split into three vectors that participate in the attention mechanism: a query q_t, a key k_t, and a value v_t. The query q_t is compared with all keys to form a probability distribution $p(t \to \cdot)$, which is then used to retrieve an average value $\bar{v}_t$.

https://arxiv.org/abs/1805.01052

More Recent Works

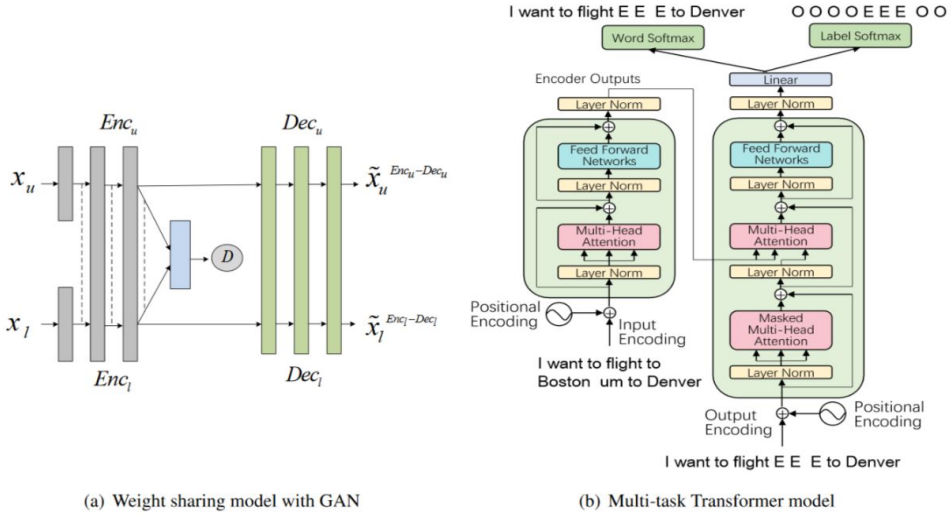
Semi-Supervised Disfluency Detection (Wang et al. 2018)

1 Introduction

A characteristic of spontaneous speech is different from written text, since it's usually accompanied by disfluencies. Identifying and removing these non-fluent factors would help to improve the spontaneous speech quality. It often plays a significant role in understanding the semantics of these sentences and it's vital for the downstream NLP tasks, such as question answering, machine translation, and information extraction.

I want to flight [to Boston + { um } + to Denver]
RM IM RP

Figure 1: Example of disfluency annotation style in Switchboard corpus.



(a) Weight sharing model with GAN (b) Multi-task Transformer model

Figure 2: The framework of the proposed model. (a) is the whole architecture of our model, which contains two independent encoders with some weight sharing and the fully-shared decoder. (b) is the specific architecture of the proposed model which extends the Transformer into multi-task learning setting.

https://arxiv.org/abs/1806.06957

More Recent Works

Improving Language Understanding by Generative Pre-Training (OpenAI)

Model specifications Our model largely follows the original transformer work [62]. We trained a 12-layer decoder-only transformer with masked self-attention heads (768 dimensional states and 12 attention heads). For the position-wise feed-forward networks, we used 3072 dimensional inner states. We used the Adam optimization scheme [27] with a max learning rate of 2.5e-4. The learning rate was increased linearly from zero over the first 2000 updates and annealed to 0 using a cosine schedule. We train for 100 epochs on minibatches of 64 randomly sampled, contiguous sequences of 512 tokens. Since layernorm [2] is used extensively throughout the model, a simple weight initialization of $N(0, 0.02)$ was sufficient. We used a bytepair encoding (BPE) vocabulary with 40,000 merges [53] and residual, embedding, and attention dropouts with a rate of 0.1 for regularization. We also employed a modified version of L2 regularization proposed in [37], with $w = 0.01$ on all non bias or gain weights. For the activation function, we used the Gaussian Error Linear Unit (GELU) [18]. We used learned position embeddings instead of the sinusoidal version proposed in the original work. We use the *ftfy* library[2] to clean the raw text in BooksCorpus, standardize some punctuation and whitespace, and use the *spaCy* tokenizer.[3]

https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

More Recent Works

A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation (Lakew et al. 2018)

Summary and Conclusions

In this work, we showed how bilingual, multilingual, and zero-shot models perform in terms of overall translation quality, as well as the errors types produced by each system. Our analysis compared Recurrent models with the recently introduced Transformer architecture. Furthermore, we explored the impact of grouping related languages for a zero-shot translation task. In order to make the overall evaluation more sound, BLEU and TER scores were complemented with mTER and lmmTER, leveraging multiple professional post-edits. Our investigation on the translation quality and the results of the fine-grained analysis shows that:

- Multilingual models consistently outperform bilingual models with respect to all considered error types, i.e., lexical, morphological, and reordering.
- The Transformer approach delivers the best performing multilingual models, with a larger gain over corresponding bilingual models than observed with RNNs.
- Multilingual models between related languages achieve the best performance scores and relative gains over corresponding bilingual models.
- When comparing zero-shot and bilingual models, relatedness of the source and target languages does not play a crucial role.
- The Transformer model delivers the best quality in all considered zero-shot condition and translation directions.

https://arxiv.org/abs/1806.06957

More Recent Works

How Much Attention Do You Need?
A Granular Analysis of Neural Machine Translation Architectures (Domhan 2018)

http://aclweb.org/anthology/P18-1167

Table 5: Different variations of the encoder and decoder self-attention layer.

In addition to that, we try a combination where the first and fourth block use self-attention, the second and fifth an RNN, the third and sixth a CNN (*combined*).

Replacing the self-attention on both the encoder and the decoder side with an RNN or CNN results in a degradation of performance. In most settings, such as WMT'17 EN→DE for both variations and WMT'17 LV→EN for the RNN, the performance is comparable when replacing the decoder side self-attention. For the encoder however, except for IWSLT, we see a drop in performance of up to 1.5 BLEU points when not using self-attention. Therefore, self-attention seems to be more important on the encoder side than on the decoder side. Despite the disadvantage of having a limited context window, the CNN performs as well as self-attention on the decoder side on IWLT and WMT'17 EN→DE in terms of BLEU and only slightly worse in terms of METEOR. The combination of the three mechanisms (*combined*) on the decoder side performs almost identical to the full Transformer model, except for IWSLT where it is slightly worse.

It is surprising how well the model works with-

a granular level. Using this language we explored how specific aspects of the Transformer architecture can successfully be applied to RNNs and CNNs. We performed an extensive evaluation on IWSLT EN→DE, WMT'17 EN→DE and LV→EN, reporting both BLEU and METEOR over multiple runs in each setting.

We found that RNN based models benefit from multiple source attention mechanisms and residual feed-forward blocks. CNN based models on the other hand can be improved through layer normalization and also feed-forward blocks. These variations bring the RNN and CNN based models close to the Transformer. Furthermore, we showed that one can successfully combine architectures. We found that self-attention is much more important on the encoder side than it is on the decoder side, where even a model without self-attention performed surprisingly well. For the data sets we evaluated on, models with self-attention on the encoder side and either an RNN or CNN on the decoder side performed competitively to the Transformer model in most cases.

We make our implementation available so that it can be used for exploring novel architecture varia-

More Recent Works

Universal Transformers (Dehghani et al. 2018)

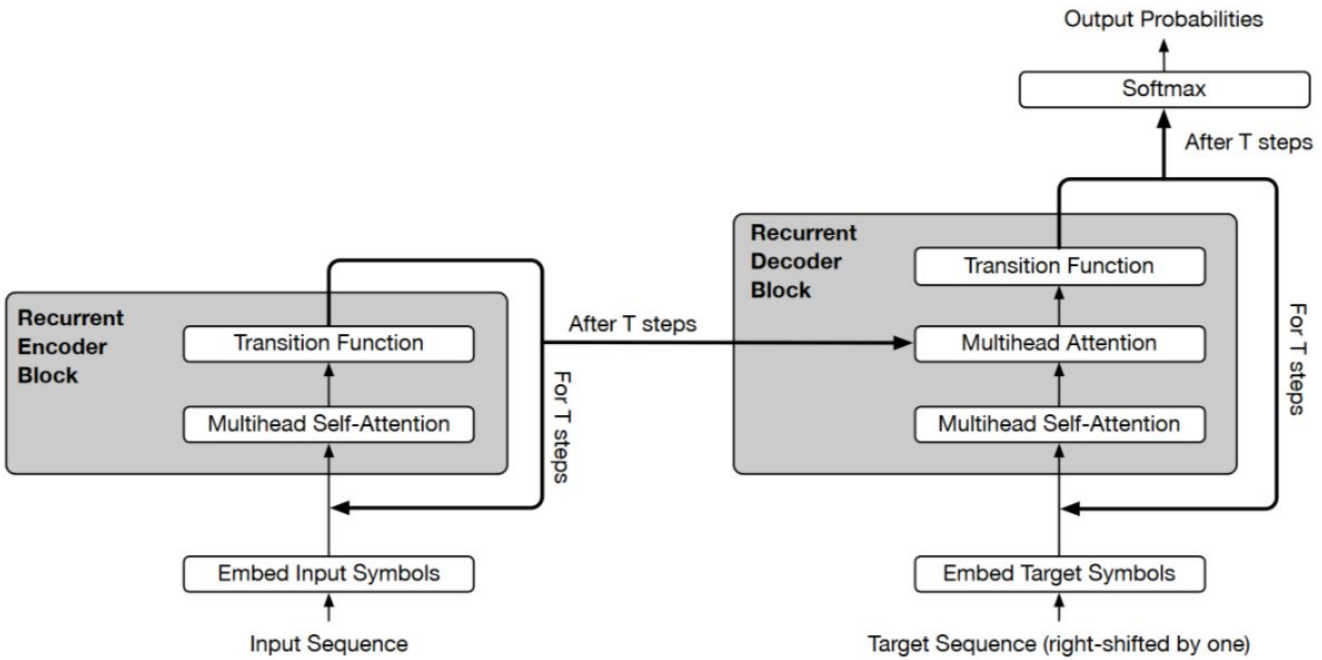


Figure 2: The recurrent blocks of the Universal Transformer encoder and decoder. This diagram omits position and time-step encodings as well as dropout, residual connections and layer normalization. A complete version can be found in the appendix. The Adaptive Universal Transformer dynamically determines the number of steps T for each position using ACT.

https://arxiv.org/abs/1807.03819

End of Part 1

Convolutional Neural Networks for NLP

Takanori Aoki

Overview

- Why CNN for NLP ?
- Comparison between RNNs and CNNs
- What is “convolution” ?
- Convolutional Neural Networks for Sentence Classification (Kim, 2014)
- Character-level CNN

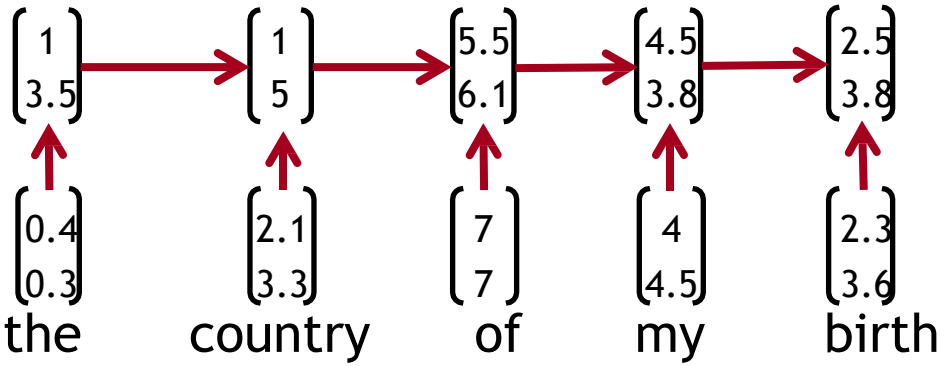
Why CNN for NLP ?

- Application
  - Text classification (e.g. sentiment analysis)
  - Translation
    - Recurrent Continuous Translation Models (Kalchbrenner and Blunsom , 2013)
    - Using CNN for encoding and RNN for decoding

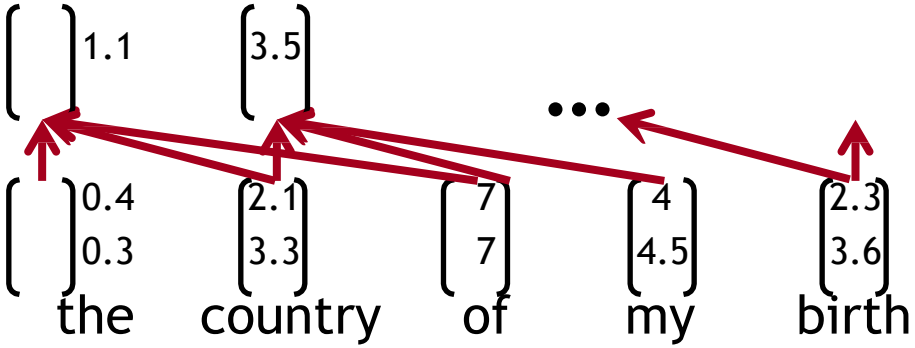
- Motivation
  - CNN is faster since the computations can be more parallelized compared to RNN
    - RNN needs to be processed sequentially because subsequent steps depend on previous ones
  - CNN is good at extracting “strong signal” from a document no matter where important features are.
    - In RNN, words in the center of a longer document might get lost

Comparison between RNNs and CNNs

- RNN processes input sequentially



- CNN can compute vectors for every possible phrase
  - Example: “the country of my birth”
    - “the country", "country of", "of my", "my birth", "the country of", "country of my", "of my birth", …



What is “convolution” ?

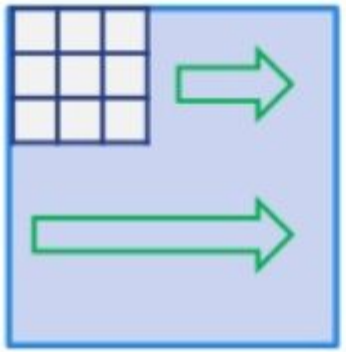
- Convolution is good at extracting features from input data !
- Elementwise multiplication and sum of a filter and the input

- 1-d convolution: $(f * g)[n] = \sum f[n-m]g[m]$
  - Useful for text classification
  - ‘f’ is words,
  - ‘g’ is value of filter

1D-conv.

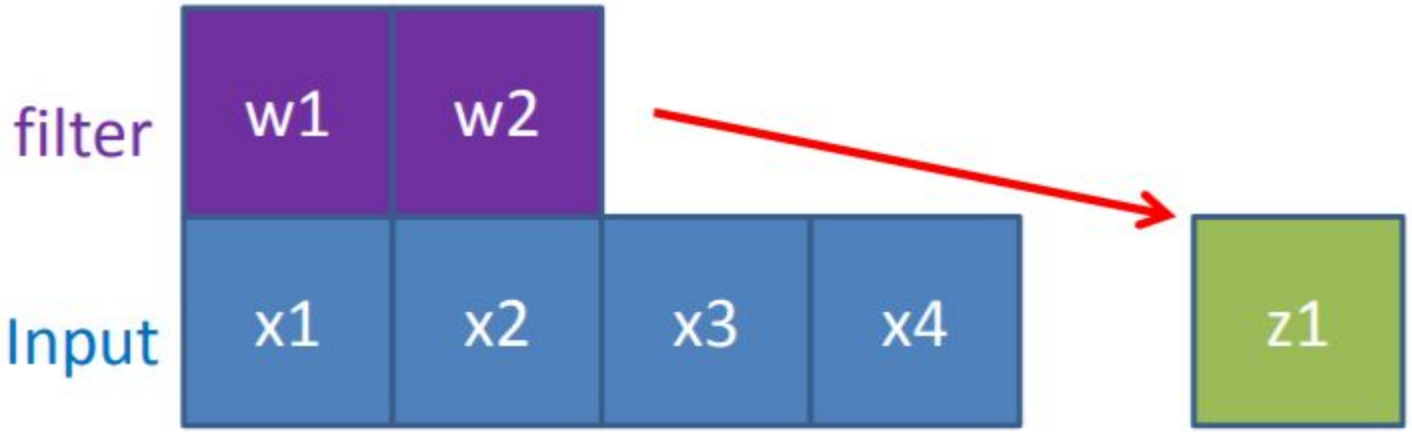
- 2-d convolution $f[x,y] * g[x,y] = \sum \sum f[n_1,n_2] \cdot g[x-n_1, y-n_2]$

2D-conv.



  - Useful for image recognition
  - ‘f’ is input pixels of image
  - ‘g’ is value of filter

1-d convolution processing flow



$$z_1 = w_1 x_1 + w_2 x_2 + b$$

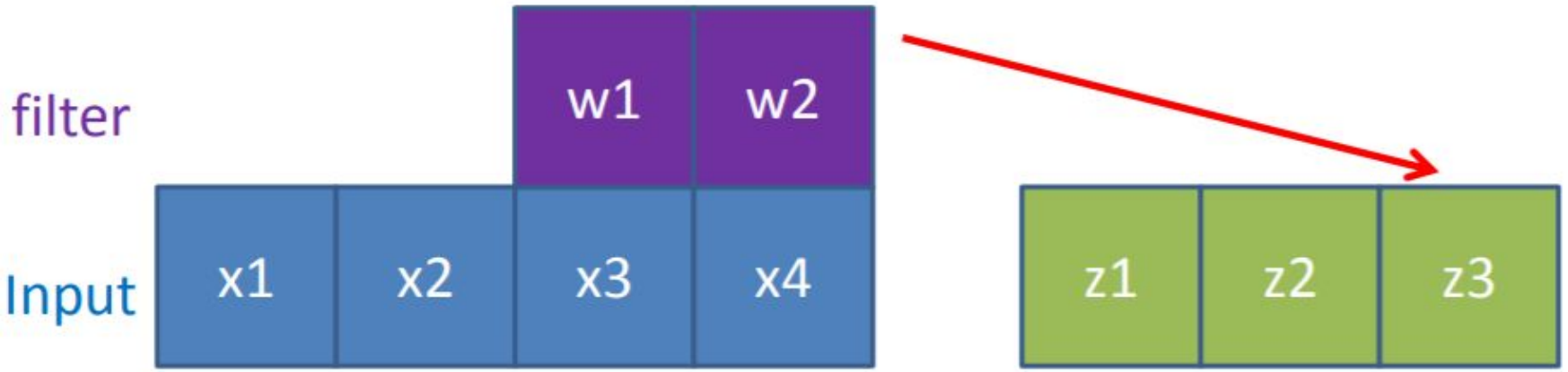
ref: https://pdfs.semanticscholar.org/2af2/2f144da59836946a58fe6ff993be59bccd24.pdf

1-d convolution processing flow

$$z_2 = w_1 x_2 + w_2 x_3 + b$$

ref: https://pdfs.semanticscholar.org/2af2/2f144da59836946a58fe6ff993be59bccd24.pdf

1-d convolution processing flow



$$z_3 = w_1 x_3 + w_2 x_4 + b$$

Featur map $\boldsymbol{z_i}$ measures the dot product similarity to the local inputs. Our goal is to train the parameters $\boldsymbol{w_i}$. The hope is we can capture patterns occurred in the input.

ref: https://pdfs.semanticscholar.org/2af2/2f144da59836946a58fe6ff993be59bccd24.pdf

Convolutional Neural Networks for Sentence Classification (Kim, 2014)

- Following 7 text classifications were attempted by using CNN

MR: Movie reviews with one sentence per review. Classification involves detecting positive / negative reviews

SST-1: Stanford Sentiment Treebank—an extension of MR but with train/dev/test splits provided and fine-grained labels (very positive, positive, neutral, negative, very negative)

SST-2: Same as SST-1 but with neutral reviews removed and binary labels.

Subj: Subjectivity dataset where the task is to classify a sentence as being subjective or objective

TREC: TREC question dataset—task involves classifying a question into 6 question types (whether the question is about person, location, numeric information, etc.)

CR: Customer reviews of various products (cameras, MP3s etc.). Task is to predict positive / negative reviews

MPQA: Opinion polarity detection subtask of the MPQA dataset

Convolutional Neural Networks for Sentence Classification (Kim, 2014)

- Highlights
  - Convolution layer
    - To extract feature (i.e. generate feature map) from sentence
  - Max-over-time pooling layer
    - To capture “strongest signal” from feature map
  - Multi-channel
    - To prevent overfitting
  - Fully connected layer with softmax
    - To classify sentence
  - Dropout and L2 regularization
    - To make a model robust

CNN architecture of Kim's paper (2014)

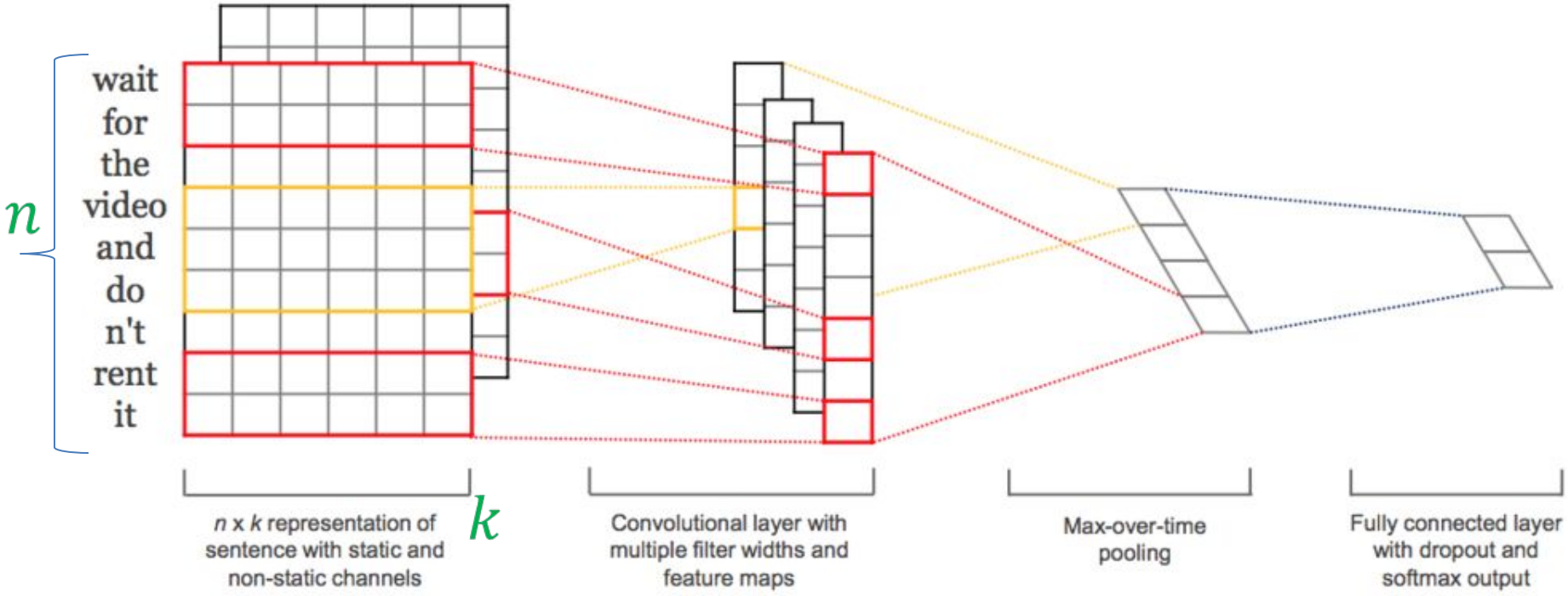


Figure 1: Model architecture with two channels for an example sentence.

n-words (possibly zero padded) and each word vector has k-dimensions

Convolutional layer

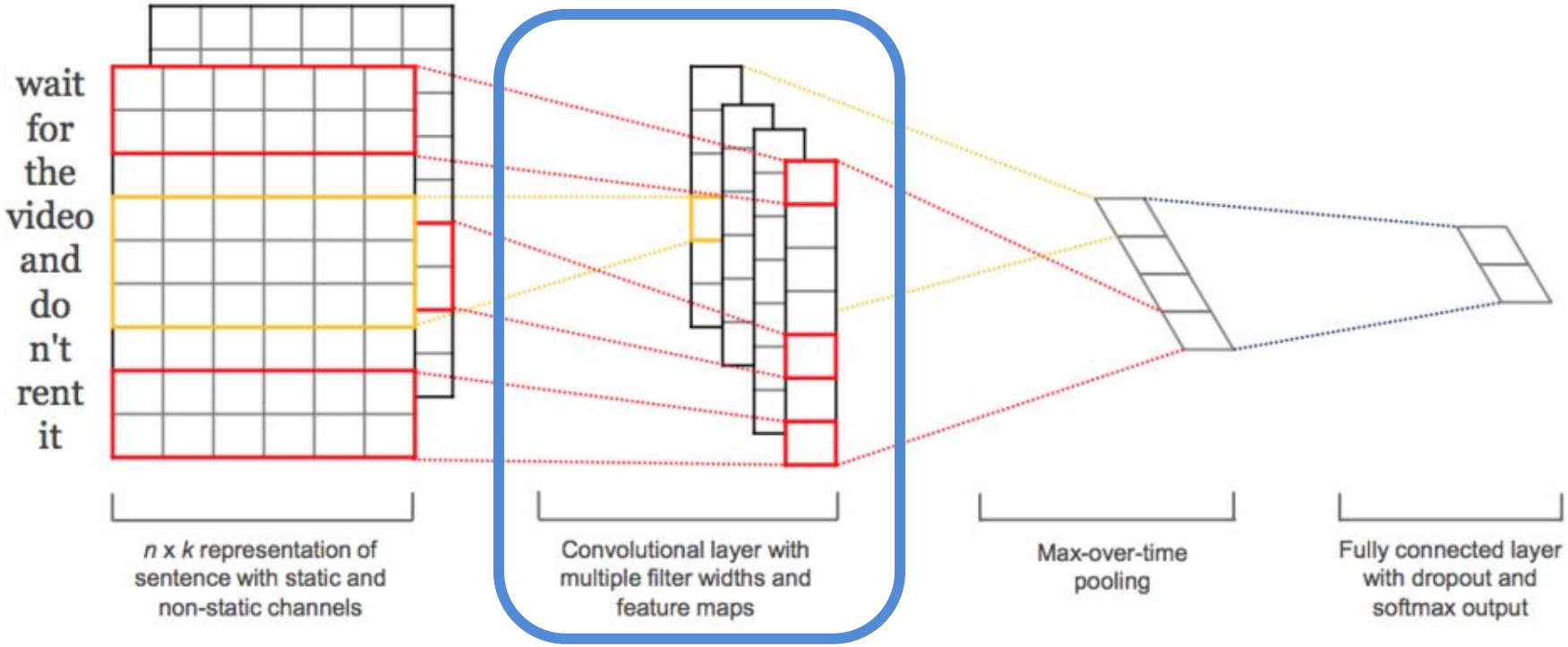


Figure 1: Model architecture with two channels for an example sentence.

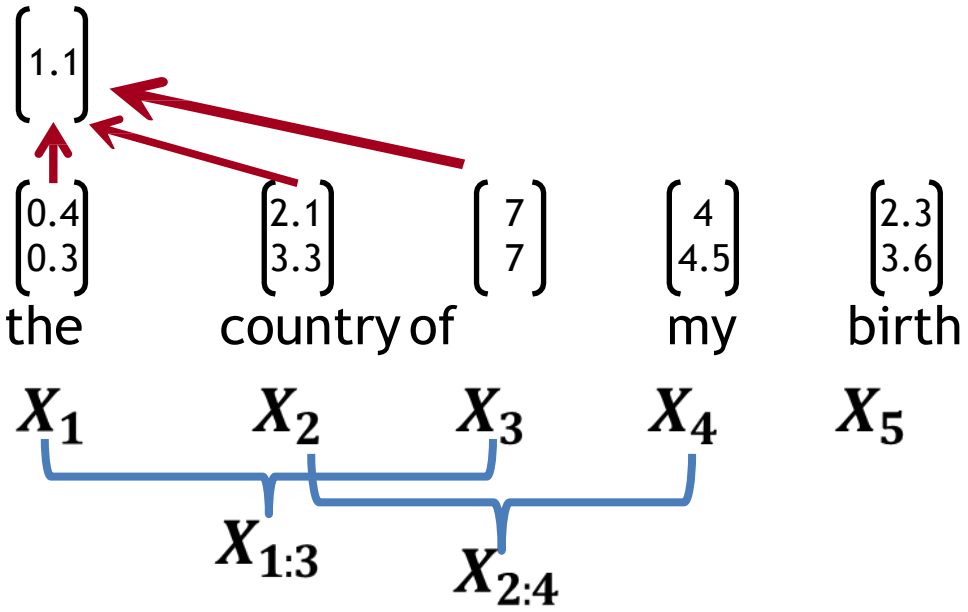
Convolutional layer:
To extract feature (i.e. generate feature map) from sentence

How Convolutional Layer works – generate feature map

- Word vectors: $X_i \in R^k$

  X_i is k-dimensional word vector corresponding to the i-th word in the sentence
- Sentence: $X_{1:n} = X_1 \oplus X_2 \oplus \cdots \oplus X_n$ (concatenated vectors)

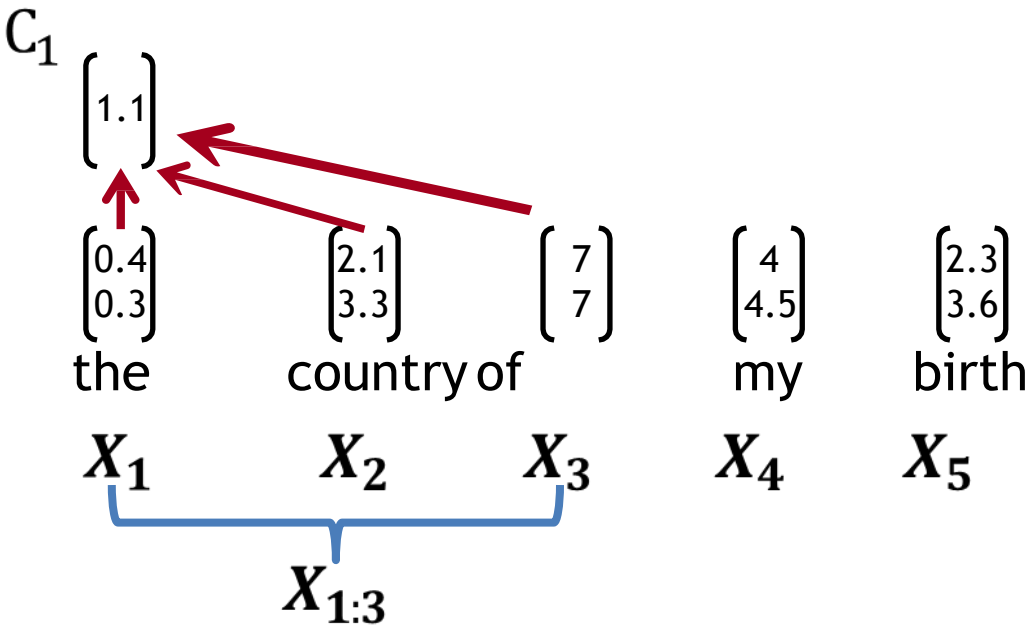
  $\oplus$ is the concatenation operator
- Concatenation of words in range: $X_{i:i+j}$
- Convolutional filter: $W \in R^{hk}$ (h: window size)
  - E.g. 3-gram (i.e. window size=3) convolution with 2-dimensional word vectors



How Convolutional Layer works – generate feature map

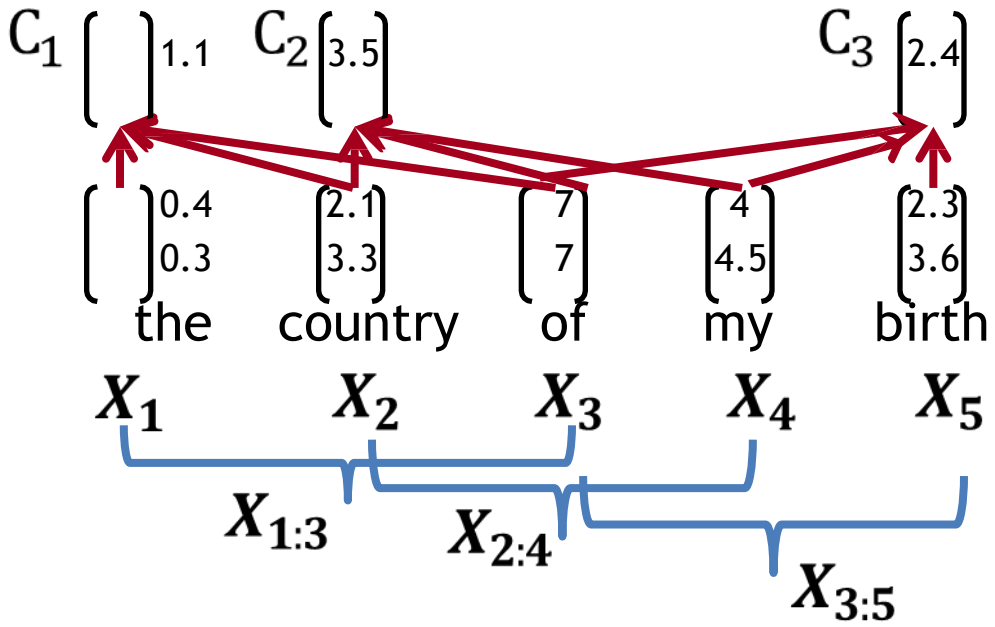
- Convolutional filter: $\boldsymbol{W} \in \boldsymbol{R}^{\boldsymbol{hk}}$ (goes over window of $\boldsymbol{h}$ words)
  - Note: filter is a vector!
- Window size h could be 2 or higher
- Formula to compute feature: $c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$

$\boldsymbol{f}$: activation function $\boldsymbol{b}$: bias term



How Convolutional Layer works – generate feature map

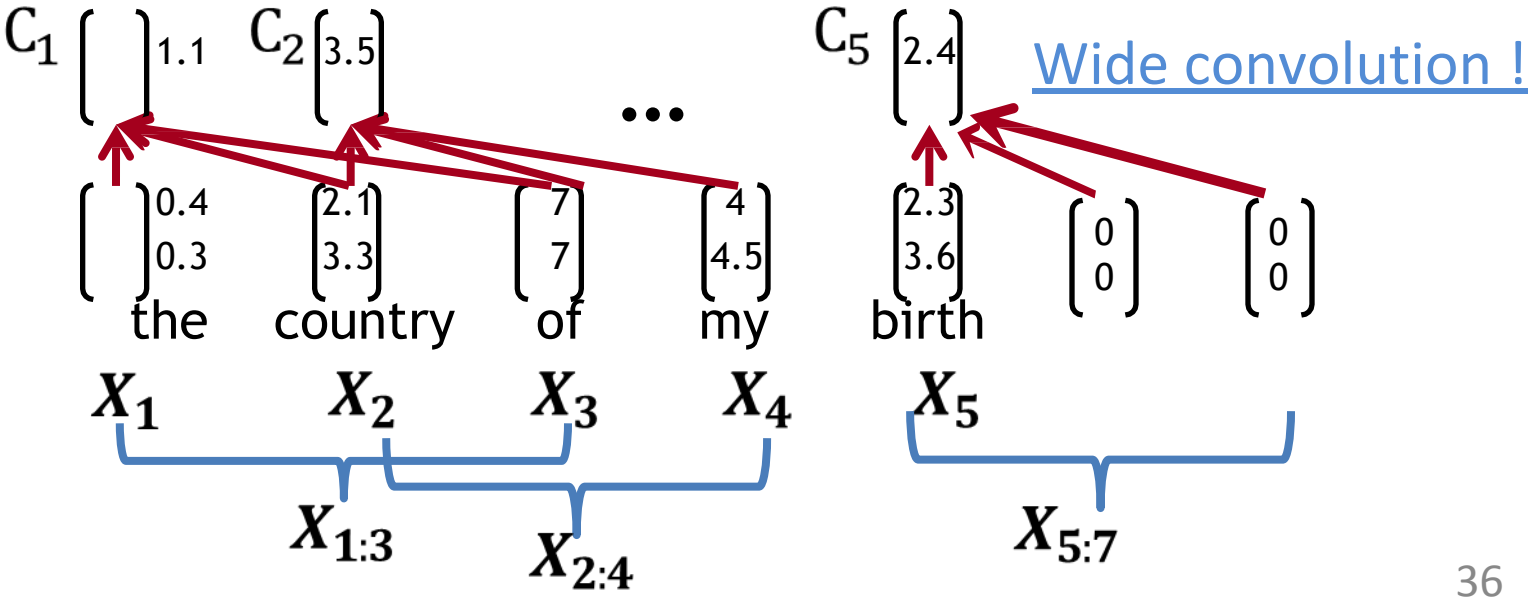
- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\boldsymbol{X_{1:n} = X_1 \oplus X_2 \oplus \cdots \oplus X_n}$
- All possible windows of length h: $\{\boldsymbol{X_{1:h}, X_{2:h+1}, X_{3:h+2}, \ldots, X_{n-h+1:n}}\}$
- Result is a feature map: $\boldsymbol{C} = [C_1, C_2, C_3, \ldots,, C_{n-h+1}] \in \boldsymbol{R^{n-h+1}}$



Narrow convolution !

How Convolutional Layer works – generate feature map

- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\boldsymbol{X_{1:n} = X_1 \oplus X_2 \oplus \cdots \oplus X_n}$
- All possible windows of length h: $\{\boldsymbol{X_{1:h}, X_{2:h+1}, X_{3:h+2}, \ldots, X_{n-h+1:n}}\}$
- Result is a feature map: $\boldsymbol{C} = [C_1, C_2, C_3, \ldots,, C_{n-h+1}] \in \boldsymbol{R^{n-h+1}}$



Max-over-time pooling layer

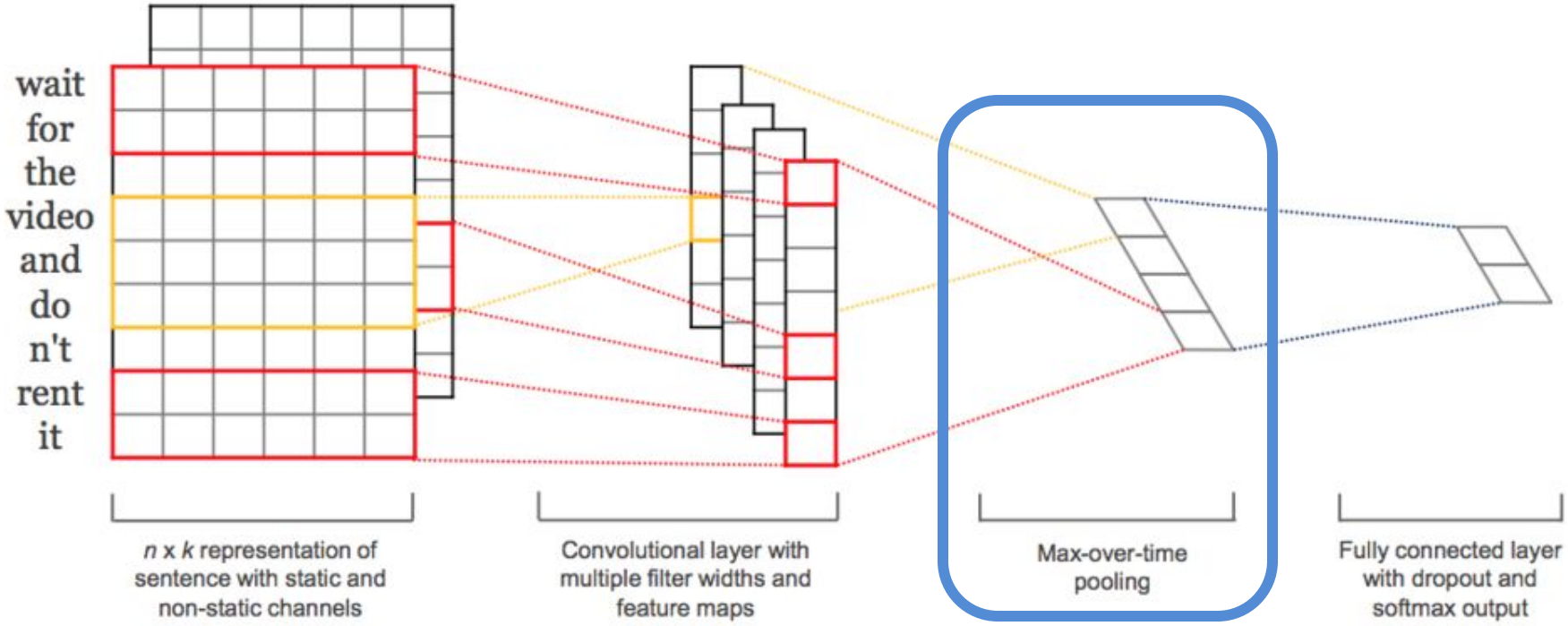
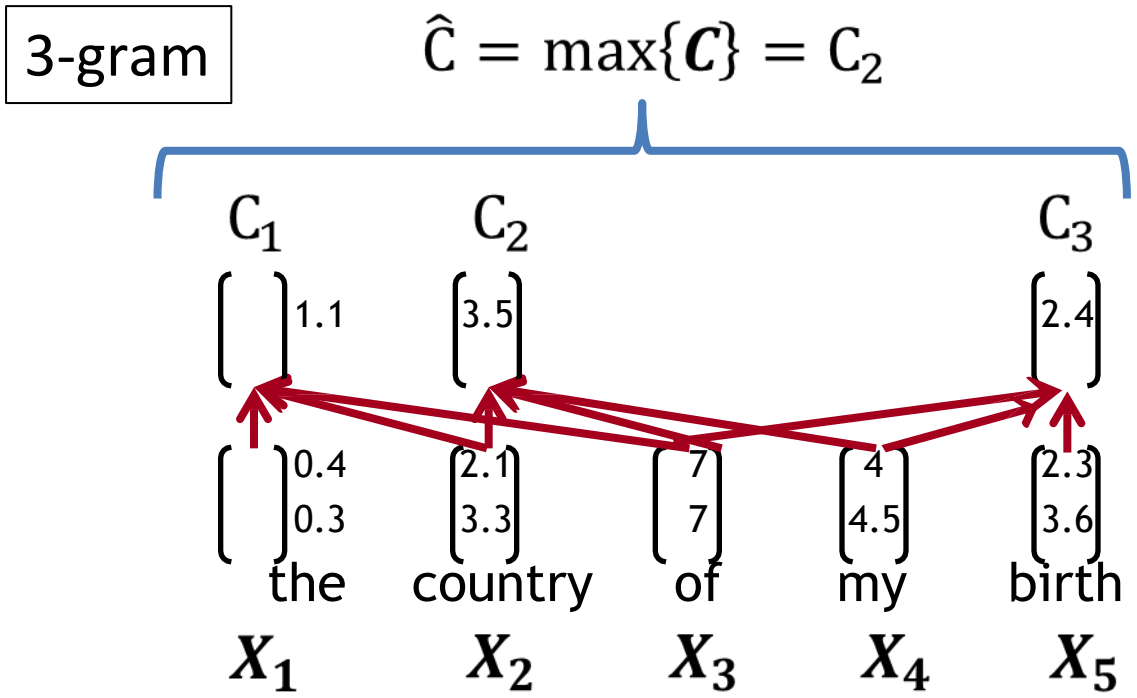


Figure 1: Model architecture with two channels for an example sentence.

Max-over-time pooling layer:
To capture "strongest signal" from feature map

How Pooling Layer works - capture the strongest signal

- Max-over-time pooling layer
  - Capture most important activation (maximum over time)
- From feature map : $\boldsymbol{C} = [C_1, C_2, C_3, \ldots,, C_{n-h+1}] \in \boldsymbol{R^{n-h+1}}$
- Pooled single number: $\hat{C} = \max\{\boldsymbol{C}\}$

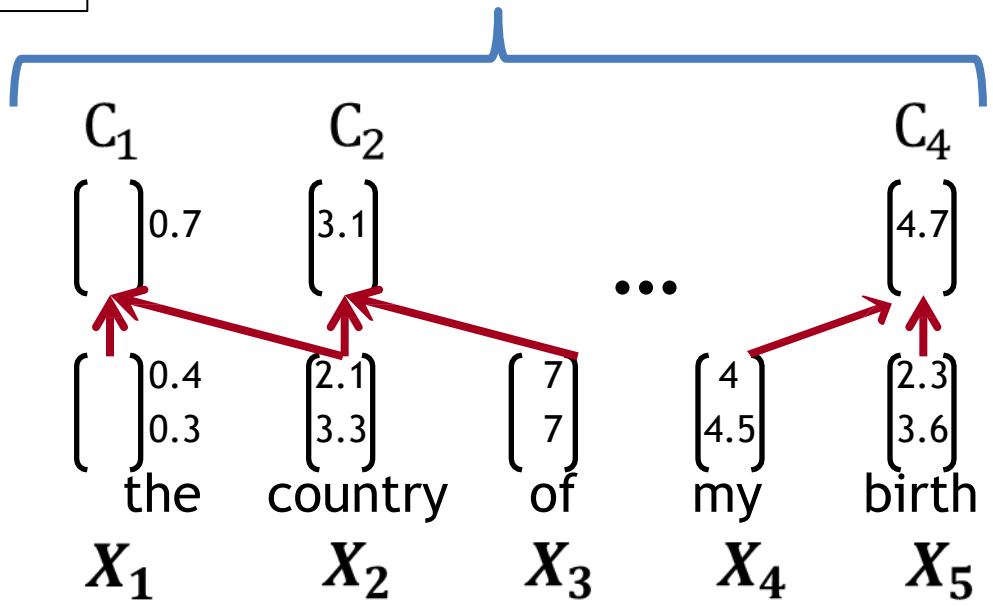


How Pooling Layer works - capture the strongest signal

- We can use multiple filters which can be applied for different word set by using different window sizes h
  - To make a model better, we need more features !
- Because of max pooling $\hat{C}$ =max{$\boldsymbol{C}$}, we don't need to worry about length of $\boldsymbol{C}$
- $\boldsymbol{C} = [C_1, C_2, C_3, \ldots, C_{n-h+1}] \in \boldsymbol{R}^{n-h+1}$
- Hence, we can have some filters that look at uni-gram, 2-grams, etc.

2-gram

$$\hat{C} = \max\{\boldsymbol{C}\} = C_4$$



Multi-channel approach

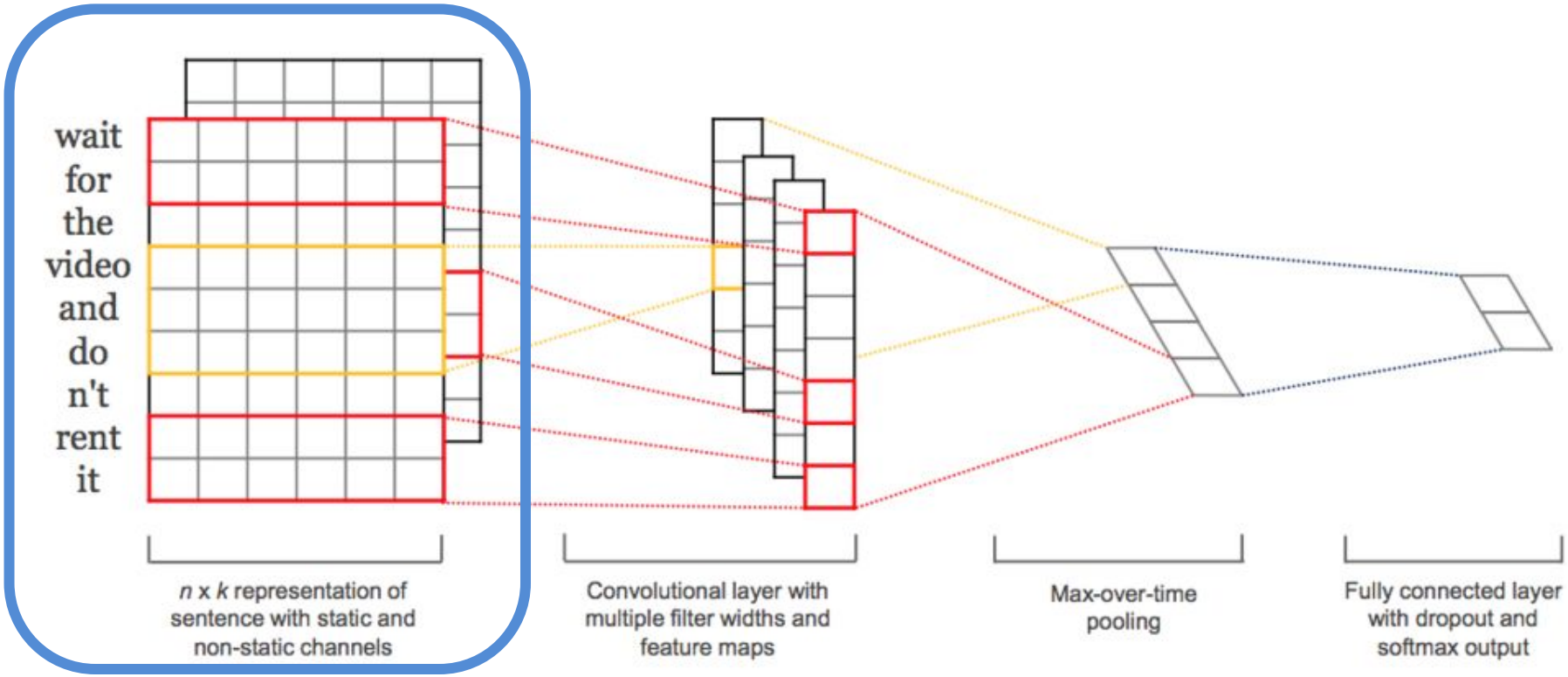
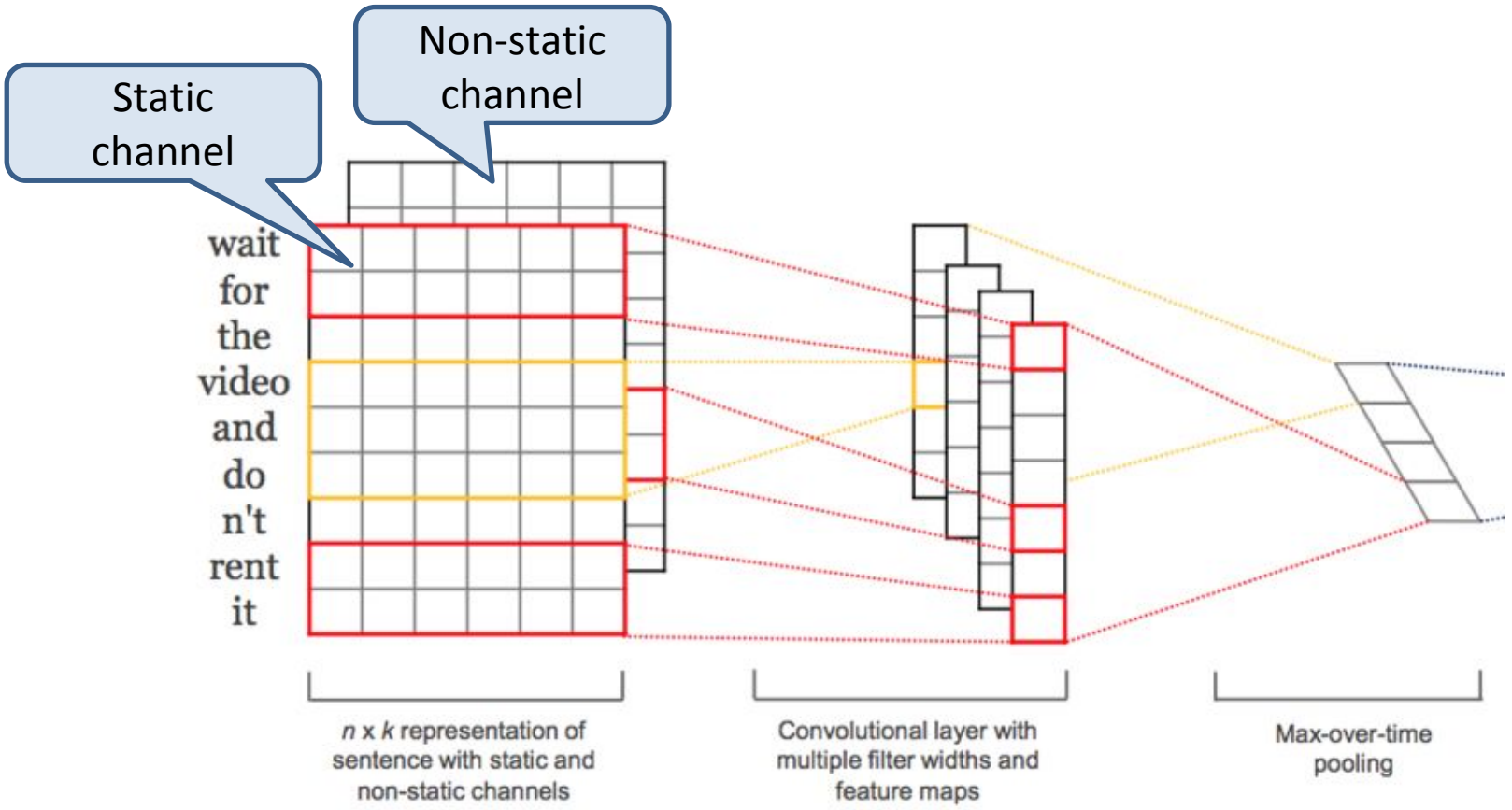


Figure 1: Model architecture with two channels for an example sentence.

Multi-channel:
To prevent overfitting

How Multi-channel works

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies, Static channel and Non-static channel.
- Backprop into only one set, keep other "static"
- Both channels are added to C_i before max-pooling layer



Fully connected layer with softmax

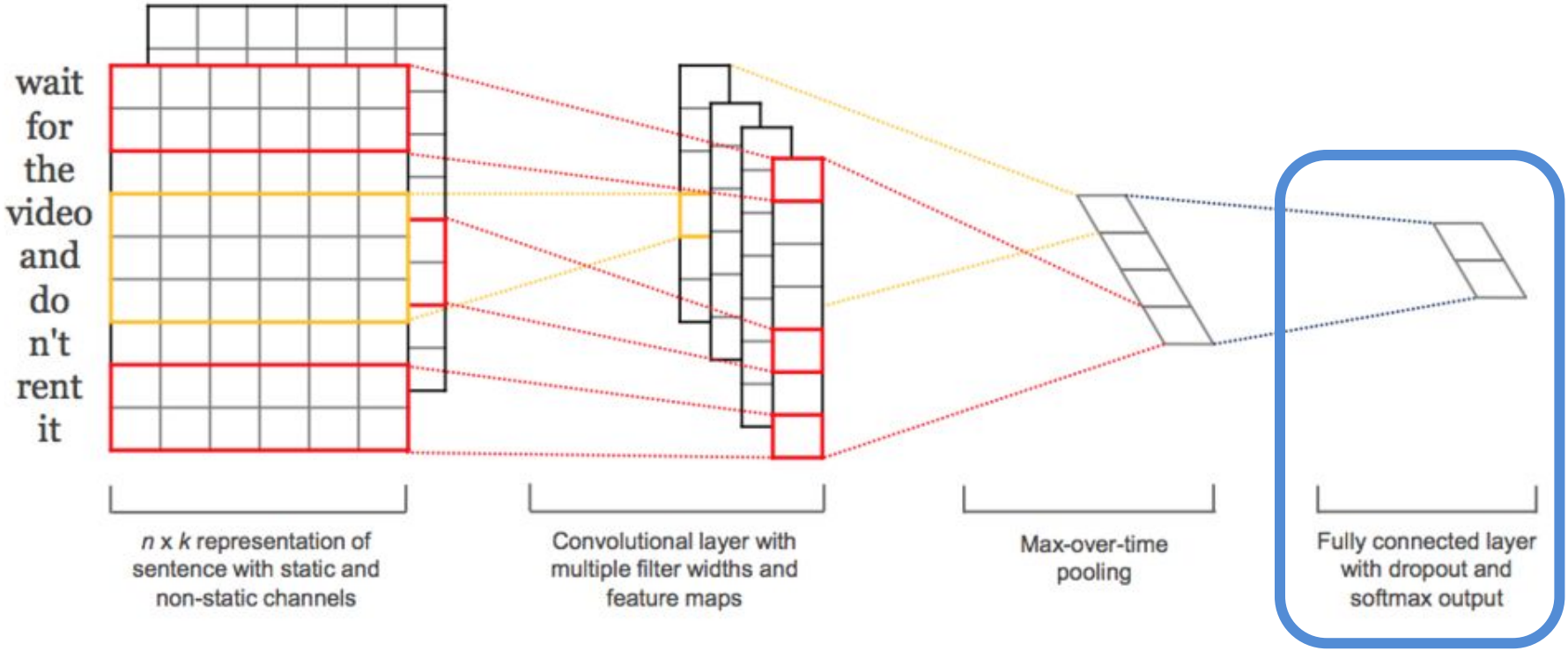


Figure 1: Model architecture with two channels for an example sentence.

Fully connected layer with softmax:
To classify sentence
Dropout and L2 regularization:
To make a model robust

Fully connected layer with softmax and dropout

- Final feature vector: $\boldsymbol{z} = [\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_m]$
  m: the number of outputs of Max-pooling layer

- Simple final softmax layer:
$$y = softmax\left(W^{(S)} z + b\right)$$

- In addition, dropout is introduced to above formula
  - Randomly mask/dropout/set to 0 some of the feature weights z
- Create masking vector ($r \in \boldsymbol{R}^{\boldsymbol{m}}$) of Bernoulli random variables with probability p of being 1 (p is a hyperparameter)
$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$
  - $\circ$ is element-wise multiplication operator

Fully connected layer with softmax and dropout

$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$

- During model training, gradients are backpropagated ONLY through those elements of z vector for which $r_i = 1$

- During inference, there is NO dropout, so value of feature vector z get larger !
- Hence, scaling final vector by Bernoulli probability p was introduced

$$\hat{W}^{(S)} = pW^{(S)}$$

- Kim (2014) reported 2 – 4% accuracy improvement and ability to use very large networks without overfitting

To make a model robust: L2 regularization

- Constrain L_2-norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number s (s is a hyperparameter)

- Whenever If $\|W_{c\cdot}^{(S)}\| > s$ after a gradient descent step then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$

All hyperparameters of Kim's paper (2014)

Following hyperparameter were chosen as a result of Grid Search on the SST-2 dev set (*1).

- Activation function: Rectified Linear Unit (ReLU)
- Window filter sizes: h = 3, 4, and 5 with 100 feature maps each
- Dropout p = 0.5
- L2 constraints for rows of softmax = 3
- Mini batch size for SGD training: 50
- Word vectors: pre-trained (*2) with word2vec, dimension k = 300

*1 For datasets without a standard dev set Kim randomly select 10% of the training data as the dev set. Training is done through stochastic gradient descent over shuffled mini-batches with the Adadelta update rule (Zeiler, 2012).

*2 Publicly available word2vec vectors that were trained on 100 billion words from Google News. Words not present in the set of pre-trained words are initialized randomly

Experiments

- These 4 models were compared

CNN-rand: Baseline model where all words are randomly initialized and then modified during training.

CNN-static: A model with pre-trained vectors from word2vec. All words— including the unknown ones that are randomly initialized—are kept static and only the other parameters of the model are learned.

CNN-non-static: Same as above but the pretrained vectors are fine-tuned for each task.

CNN-multichannel: A model with two sets of word vectors. Each set of vectors is treated as a “channel” and each filter is applied to both channels, but gradients are backpropagated only through one of the channels. Hence the model is able to fine-tune one set of vectors while keeping the other static. Both channels are initialized with word2vec.

Experimental results (Kim, 2014)

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Note: Problem with comparison?

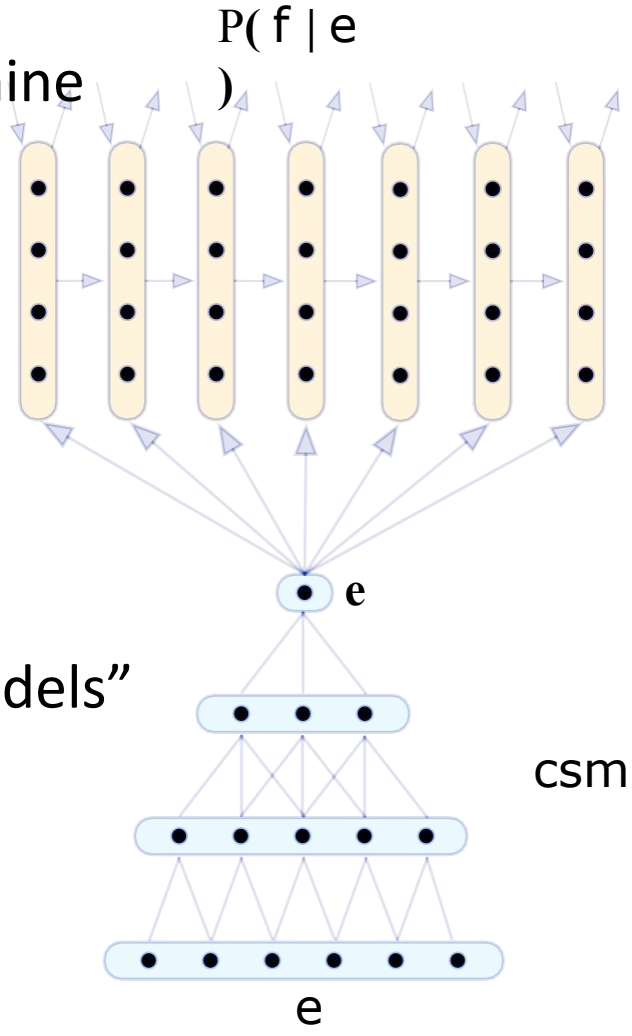
- Dropout gives 2 – 4 % accuracy improvement. However, several baselines didn't use dropout.
  - Still remarkable results and simple architecture !

- Difference to window and RNN architectures we described in previous lectures: pooling, many filters and dropout

- Ideas can be used in RNN's too

CNN application: Translation

- One of the first successful neural machine translation efforts
- Uses CNN for encoding and RNN for decoding
- Kalchbrenner and Blunsom (2013) "Recurrent Continuous Translation Models"



Character-level CNN

Why Character-level CNN (CLCNN) ?

- Advantage
  - Model is robust against typos and misspelling
    - Useful to classify human written sentences such as Amazon review and WhatsApp chat
  - CLCNN can be applicable for non-document strings such as URL and source code
  - Raw input can be used for a model without some pre-processing
    - In some languages such as Japanese and Chinese, word segmentation is required because there is no whitespace between words

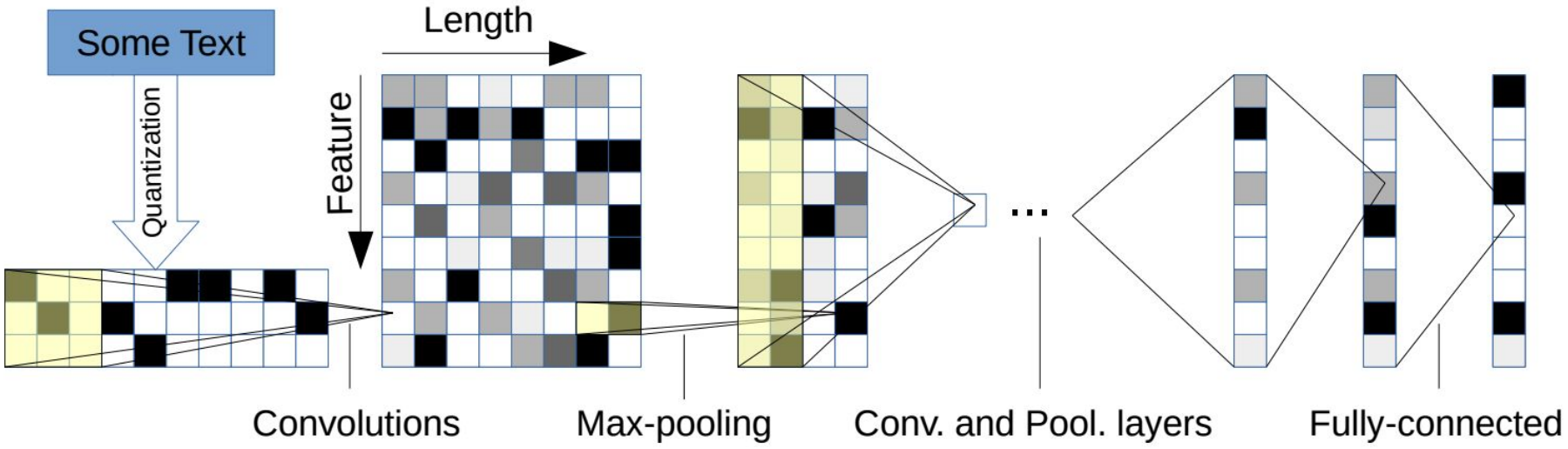
I study Deep Learning for NLP.

私は自然言語処理のための深層学習を勉強します。

- Disadvantage
  - Model training time is longer

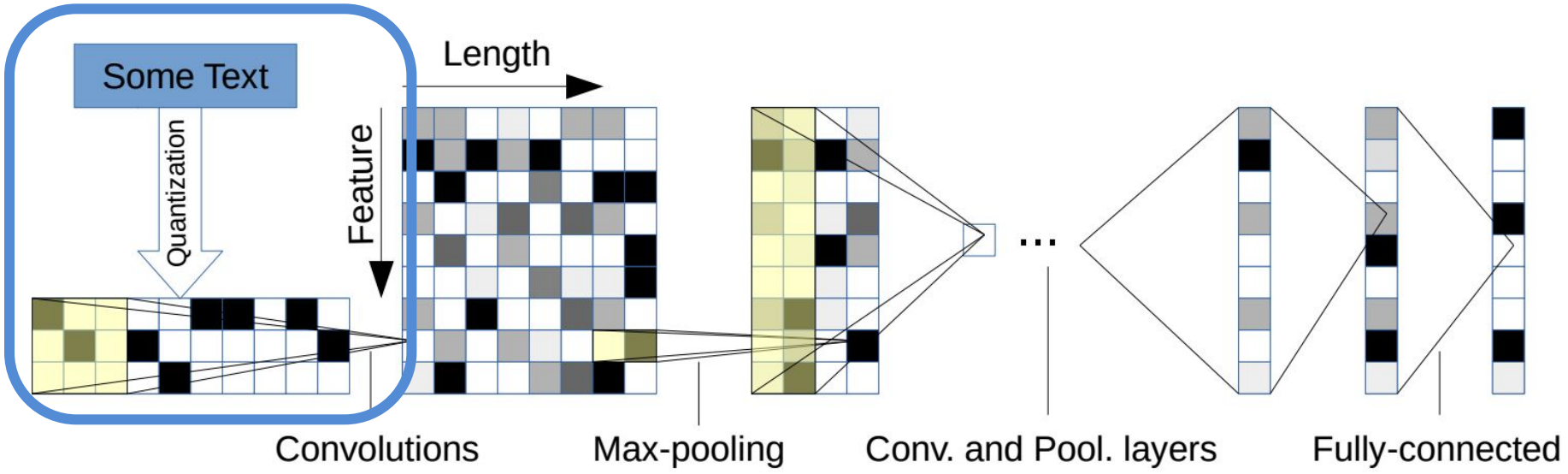
Character-level Convolutional Networks for Text Classification (Zhang, X., et al., 2015)

- Text classifications were attempted by using Character-level CNN
  - Architectur of Character-level CNN is as follows



  - Data Augmentation using Thesaurus
    - All replaceable words were extracted from the given text and randomly choose r of them to be replaced by synonyms

Quantization (Zhang, X., et al., 2015)

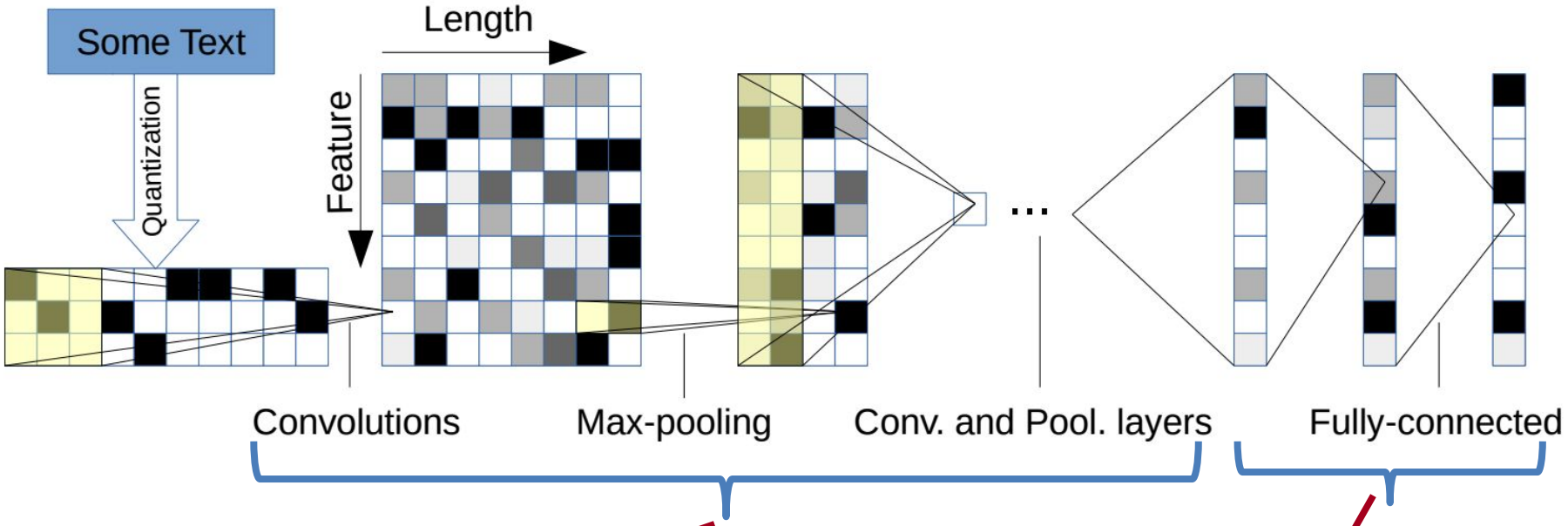


- Encode following 70 target characters as one-hot vectors

```
abcdefghijklmnopqrstuvwxyz0123456789
-,;.!?:'''/\|_@#$%^&*~`+-=<>()[]{}
```

(e.g.) a = (1 0 0 … 0),
b = (0 1 0 … 0),
…
} = (0 0 0 … 1)

Conv, Pooling, and Fully-connected layers (Zhang, X., et al., 2015)



Layer	Large Feature	Small Feature	Kernel	Pool
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	N/A
4	1024	256	3	N/A
5	1024	256	3	N/A
6	1024	256	3	3

Layer	Output Units Large	Output Units Small
7	2048	1024
8	2048	1024
9	Depends on the problem	

Dataset for evaluation (Zhang, X., et al., 2015)

- Following 8 text classifications were attempted by using CLCNN

Table 3: Statistics of our large-scale datasets. Epoch size is the number of minibatches in one epoch

Dataset	Classes	Train Samples	Test Samples	Epoch Size
AG's News	4	120,000	7,600	5,000
Sogou News (*1)	5	450,000	60,000	5,000
DBPedia	14	560,000	70,000	5,000
Yelp Review Polarity	2	560,000	38,000	5,000
Yelp Review Full	5	650,000	50,000	5,000
Yahoo! Answers	10	1,400,000	60,000	10,000
Amazon Review Full	5	3,000,000	650,000	30,000
Amazon Review Polarity	2	3,600,000	400,000	30,000

*1 Although this is a dataset in Chinese, we used pypinyin package combined with jieba Chinese segmentation system to produce Pinyin – a phonetic romanization of Chinese. The models for English can then be applied to this dataset without change. The fields used are title and content.

Experimental results (Zhang, X., et al., 2015)

Best model in the dataset
Worst model in the dataset

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Table 4: Testing errors of all the models. Numbers are in percentage. "Lg" stands for "large" and "Sm" stands for "small". "w2v" is an abbreviation for "word2vec", and "Lk" for "lookup table". "Th" stands for thesaurus. ConvNets labeled "Full" are those that distinguish between lower and upper letters

Best model in the dataset
Worst model in the dataset

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-								.39
LST								10
Lg.								88
Sm.								00
Lg.								80
Sm.								63
Lg.								84
Sm.								85
Lg.								52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

- Larger datasets tend to perform better in CLCNN
  - More than several millions samples, CLCNN starts to do better
- CLCNN may work well for user-generated data
  - Further analysis is needed to validate the hypothesis
- There is no free lunch

A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys (Saxe, X. & Berlin, K., 2017)

- Malicious URLs, File Paths and Registry Keys detections were attempted by using Character-level CNN
  - Targets are not documents but just strings!

Malicious URLs

```
http:\\0fx8o.841240.cc\201610\18\content_23312\svchost.exe
http:\\31.14.136.202\secure.apple.id.login\Apple\login.php
http:\\1stopmoney.com\paypal-login-secure\websc.php
```

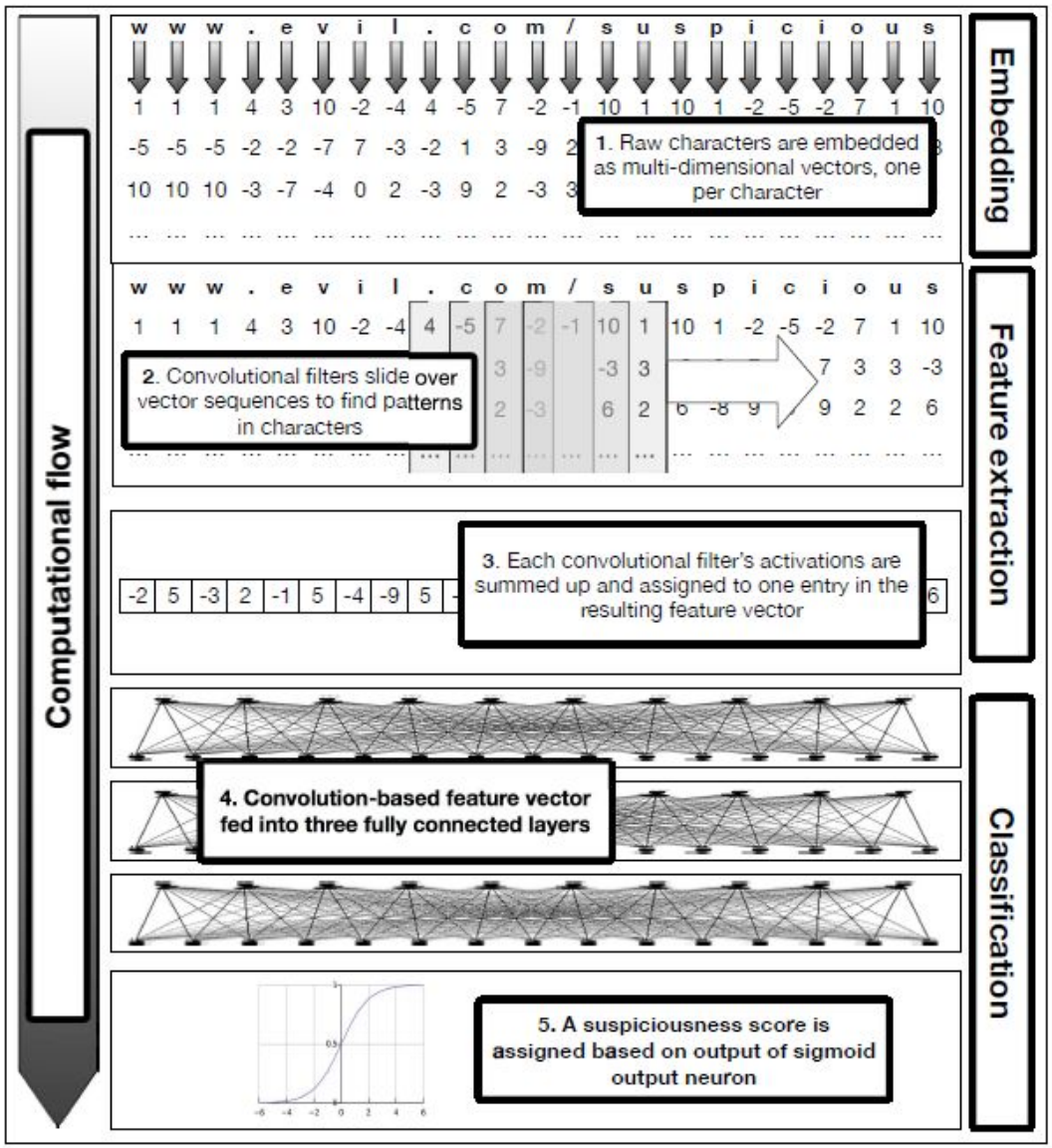
Malicious File Paths

```
C:\Temp\702D97503A79B0EC69\JUEGOS/Call of Duty 4+Keygen
C:\Temp\svchost.vbs
C:\DOCUME~1\BASANT~1\LOCALS~1\Temp\WzEC.tmp\fax.doc.exe
```

Malicious Registry Keys

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run Alpha Antivirus
HKCR\Applications\WEBCAM HACKER 1.0.0.4.EXE
HKCR\AppID\bccicabecccag.exe
```

Overview of CNN architecture (Saxe, X. & Berlin, K., 2017)

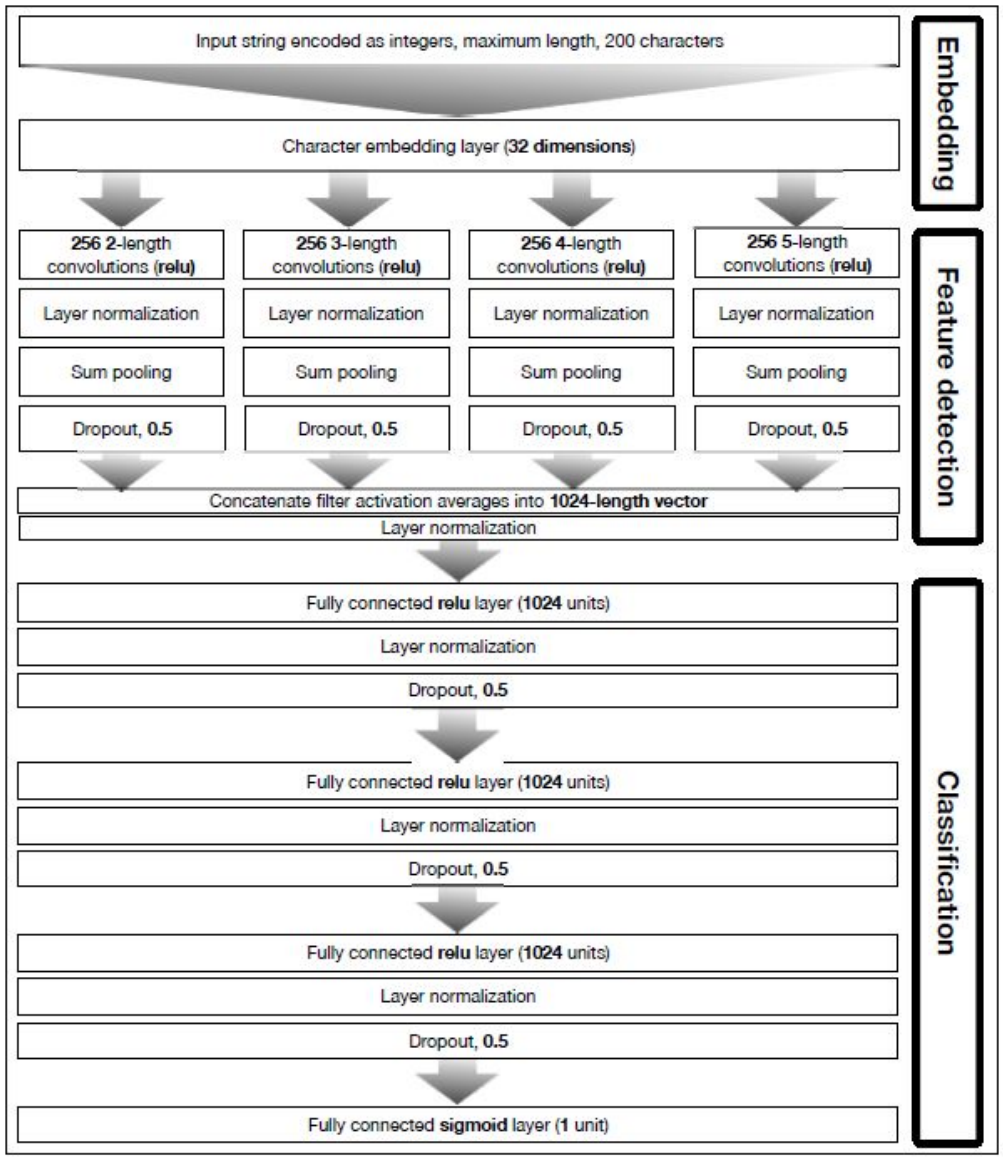


Encoding input string into embedded vector by each character

Feature extraction by Convolution and Pooling layers

Classification by Fully-connected layers

Detail of CNN architecture (Saxe, X. & Berlin, K., 2017)



Experimental results
(Saxe, X. & Berlin, K., 2017)

Task		TPR (at various FPRs) 10^{-4}	10^{-3}	10^{-2}	AUC
URLs	Convnet	**0.77**	**0.84**	**0.92**	**0.993**
	N-gram	0.76	0.78	0.84	0.985
	Expert	0.74	0.78	0.84	0.985
File Paths	Convnet	0.16	**0.43**	**0.68**	**0.978**
	N-gram	**0.18**	0.33	0.65	0.972
Registry Keys	Convnet	**0.51**	**0.62**	**0.86**	**0.992**
	N-gram	0.11	0.49	0.72	0.988

Other interesting attempts for Japanese

- Japanese Text Classification by Character-level Deep ConvNets and Transfer Learning (Sato, M., et al., 2017)

  http://www.scitepress.org/Papers/2017/61934/61934.pdf

  - Compared model performance between Japanese text romanization and character-level embeddings
    - One-hot encoding is not good way because there are 2,000+ daily use characters in Japanese
  - Transfer learning can be helpful to build robust model given small training dataset
- Document Classification through Image-Based Character Embedding and Wildcard Training (Shimada, D., 2016)

  http://ucrel.lancs.ac.uk/bignlp2016/Shimada.pdf

  - Using text as image data to build CNN

Bibliography

Word-level CNN

- Convolutional Neural Networks for Sentence Classification (Kim, 2014)
  https://arxiv.org/pdf/1408.5882.pdf

Character-level CNN

- Character-level Convolutional Networks for Text Classification (Zhang, X., et al., 2015)
  https://arxiv.org/pdf/1509.01626.pdf
- A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys (Saxe, X. & Berlin, K., 2017)
  https://arxiv.org/pdf/1702.08568.pdf
- Japanese Text Classification by Character-level Deep ConvNets and Transfer Learning (Sato, M., et al., 2017)
  http://www.scitepress.org/Papers/2017/61934/61934.pdf
- Document Classification through Image-Based Character Embedding and Wildcard Training (Shimada, D., 2016)
  http://ucrel.lancs.ac.uk/bignlp2016/Shimada.pdf