# Contextual word representations

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding

# Overview

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

# Associated materials

1. Notebook: `contextualreps.ipynb`
2. Smith 2019
3. CS224n lecture: slides and YouTube version
4. ELMo:
   - Peters et al. 2018
   - Project site: `https://allennlp.org/elmo`
5. Transformer
   - Vaswani et al. 2017
   - Alexander Rush: The Annotated Transformer [link]
6. BERT
   - Devlin et al. 2019
   - Project site: `https://github.com/google-research/bert`
   - bert-as-service [link]

# Word representations and context

# Word representations and context

1. a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.

# Word representations and context

1. a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.
2. a. flat tire/beer/note/surface
   b. throw a party/fight/ball/fit

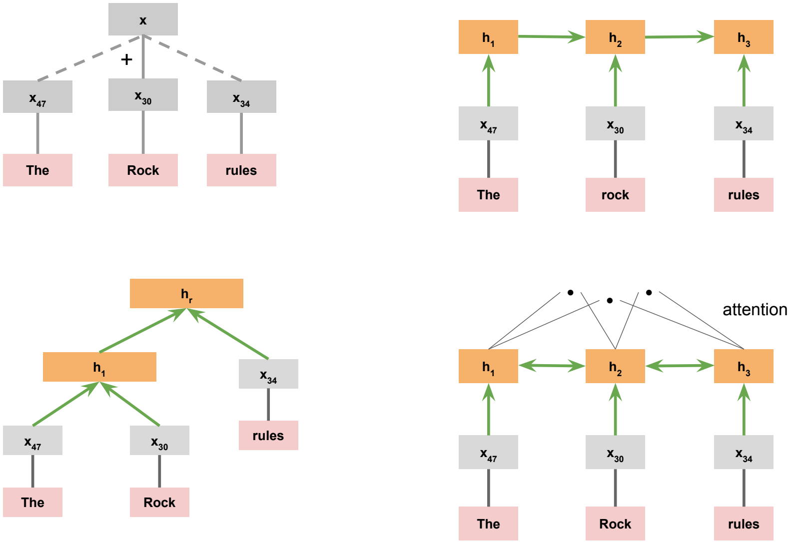
# Word representations and context

1.
   a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.
2.
   a. flat tire/beer/note/surface
   b. throw a party/fight/ball/fit
3.
   a. A crane caught a fish.
   b. A crane picked up the steel beam.
   c. I saw a crane.

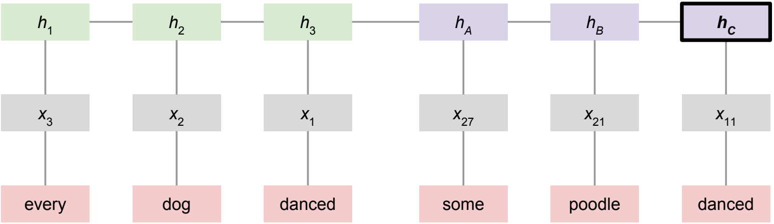
# Word representations and context

1. a. The vase broke.
   b. Dawn broke.
   c. The news broke.
   d. Sandy broke the world record.
   e. Sandy broke the law.
   f. The burgler broke into the house.
   g. The newscaster broke into the movie broadcast.
   h. We broke even.
2. a. flat tire/beer/note/surface
   b. throw a party/fight/ball/fit
3. a. A crane caught a fish.
   b. A crane picked up the steel beam.
   c. I saw a crane.
4. a. Are there typos? I didn't see any.
   b. Are there bookstores downtown? I didn't see any.

# Model structure and linguistic structure

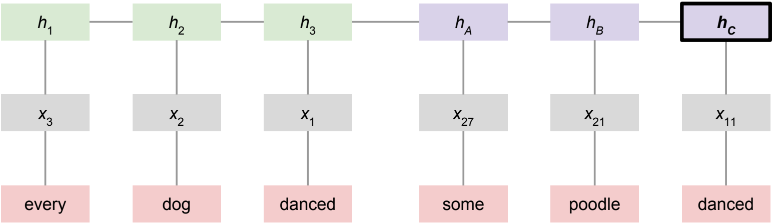


# Guiding idea: Attention (from the NLI slides)



# Guiding idea: Attention (from the NLI slides)

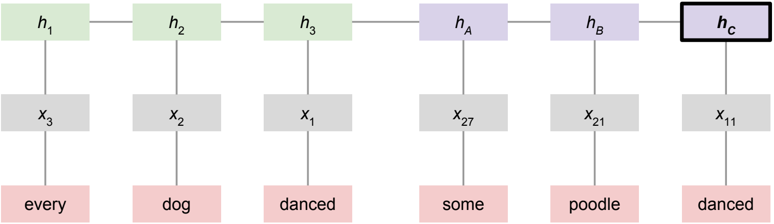
scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



# Guiding idea: Attention (from the NLI slides)

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

scores $\tilde{\alpha} = \left[\; h_C^\top h_1 \quad h_C^\top h_2 \quad h_C^\top h_3 \;\right]$

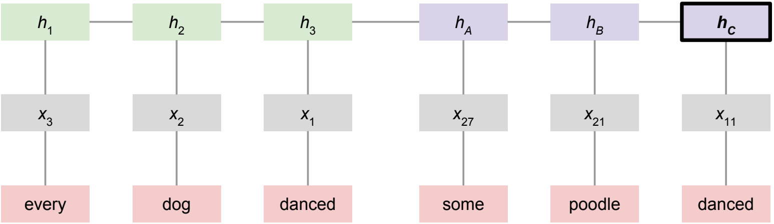


# Guiding idea: Attention (from the NLI slides)

context $\kappa = \textbf{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3)$

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



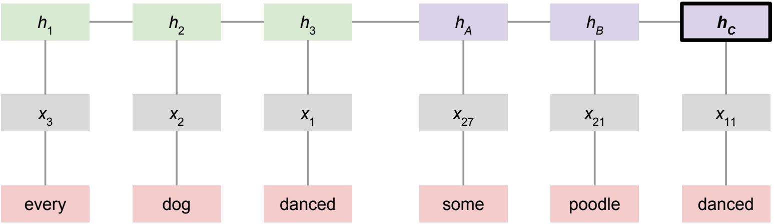
# Guiding idea: Attention (from the NLI slides)

attention combo $\tilde{h} = \tanh([\kappa; h_C]W_\kappa)$

context $\kappa = \textbf{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3)$

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



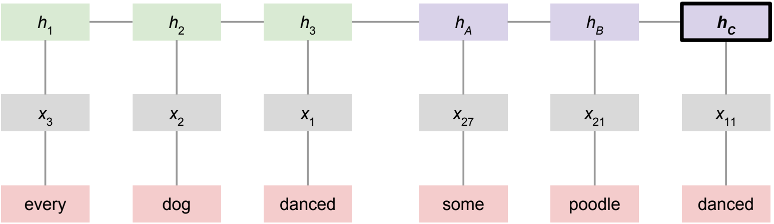
# Guiding idea: Attention (from the NLI slides)

attention combo $\tilde{h} = \tanh([\kappa; h_C]W_\kappa)$ or $\tilde{h} = \tanh(\kappa W_\kappa + h_C W_h)$

context $\kappa = \textbf{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3)$

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



# Guiding idea: Attention (from the NLI slides)

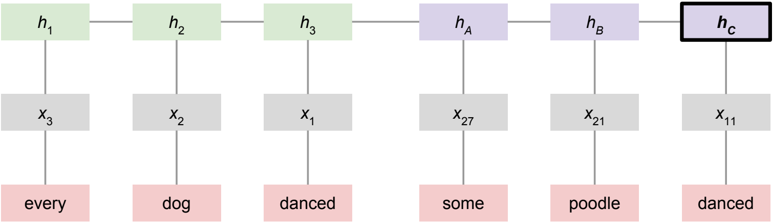
classifier $y = \textbf{softmax}(\tilde{h}W + b)$

attention combo $\tilde{h} = \tanh([\kappa; h_C]W_\kappa)$

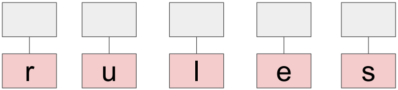
context $\kappa = \textbf{mean}(\alpha_1 h_1, \alpha_2 h_2, \alpha_3 h_3)$

attention weights $\alpha = \textbf{softmax}(\tilde{\alpha})$

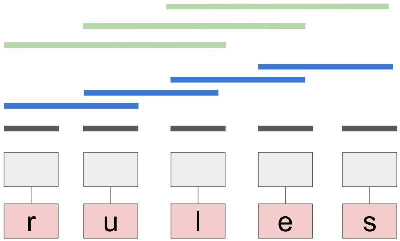
scores $\tilde{\alpha} = \begin{bmatrix} h_C^\top h_1 & h_C^\top h_2 & h_C^\top h_3 \end{bmatrix}$



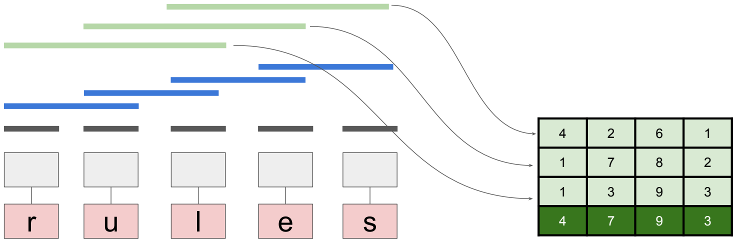
# Guiding idea: Subword modeling



# Guiding idea: Subword modeling

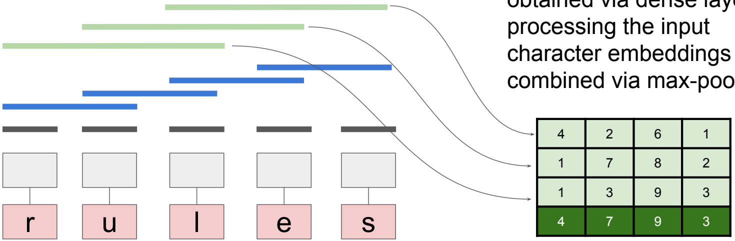


# Guiding idea: Subword modeling

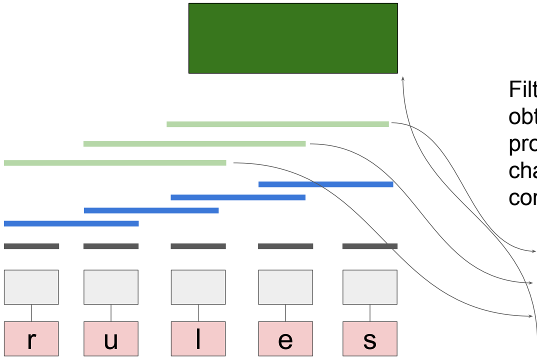


# Guiding idea: Subword modeling

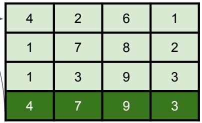
Filters of different length, obtained via dense layers processing the input character embeddings and combined via max-pooling:



# Guiding idea: Subword modeling



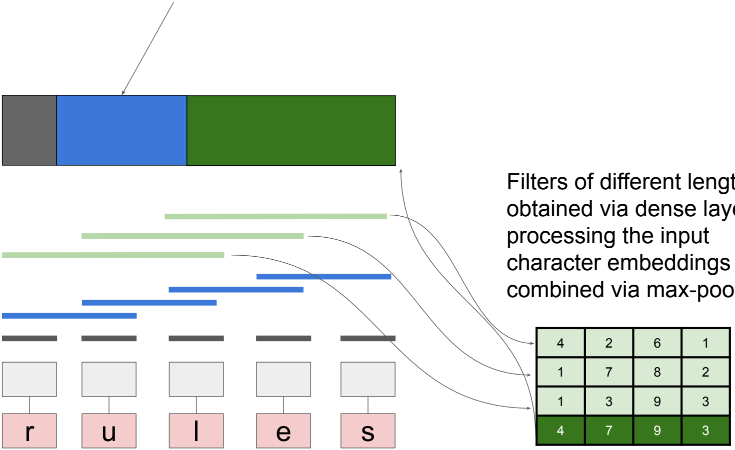

Filters of different length, obtained via dense layers processing the input character embeddings and combined via max-pooling:



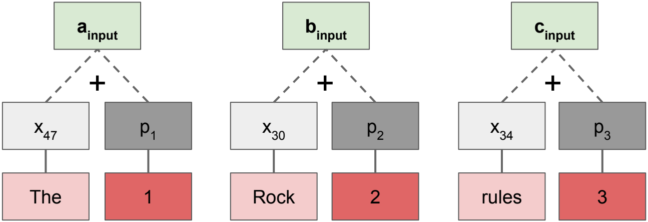
# Guiding idea: Subword modeling

Max-pooling layers concatenated to form the word representation.

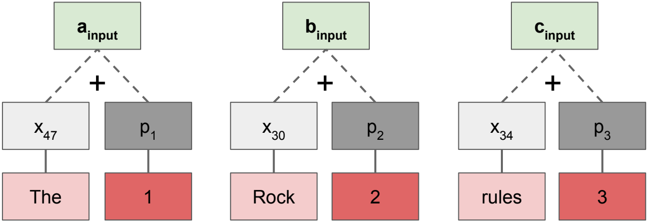
Filters of different length, obtained via dense layers processing the input character embeddings and combined via max-pooling:

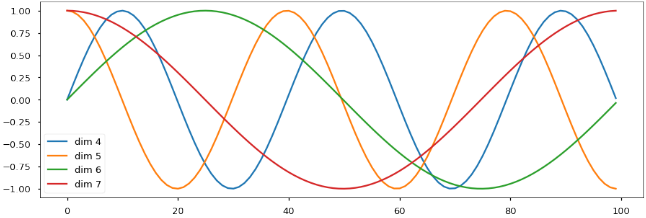


# Guiding idea: Positional encoding



# Guiding idea: Positional encoding



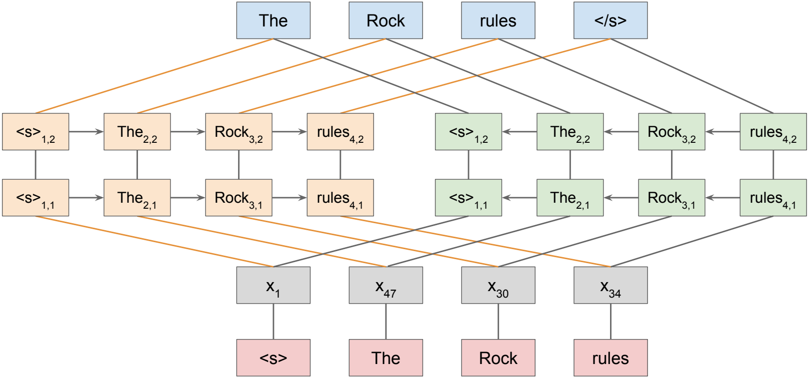




From 'The Annotated Transformer'

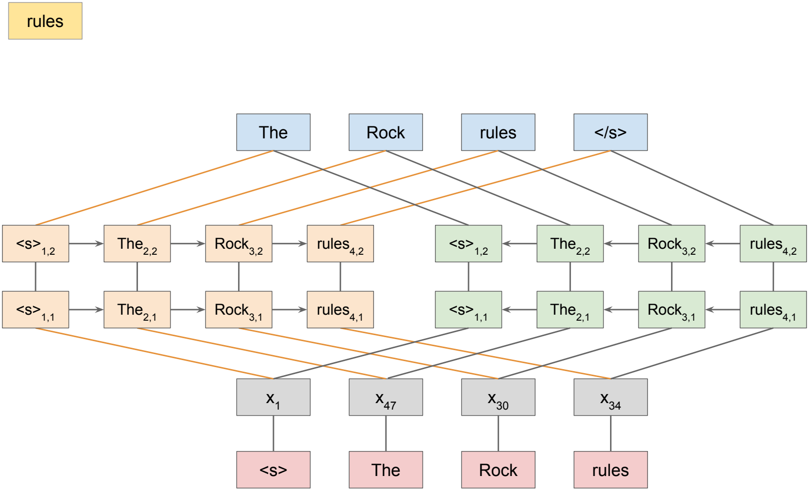
# ELMo

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

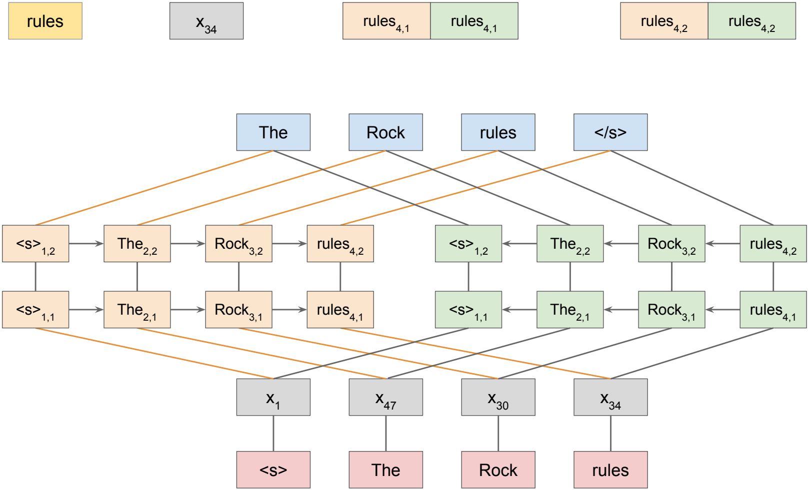
# Core model structure



# Core model structure

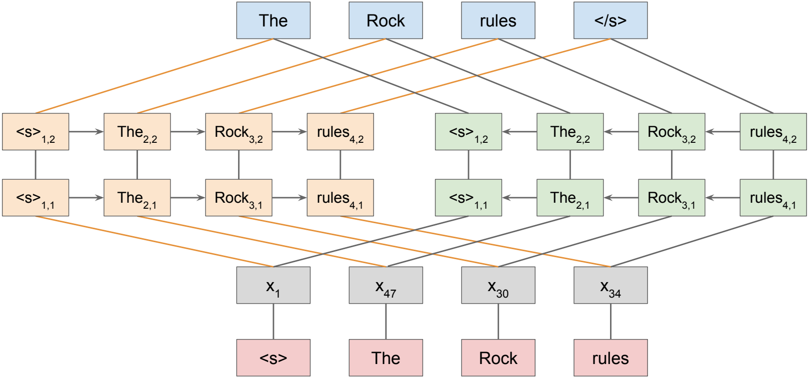


# Core model structure

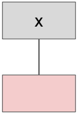


# Core model structure

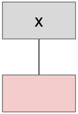
$$\boxed{\text{rules}} = s_0^{\text{task}} \cdot \boxed{x_{34}} + s_1^{\text{task}} \cdot \boxed{\text{rules}_{4,1}\ \text{rules}_{4,1}} + s_2^{\text{task}} \cdot \boxed{\text{rules}_{4,2}\ \text{rules}_{4,2}}$$

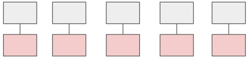


# Word embeddings

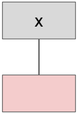


# Word embeddings

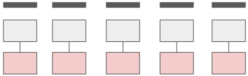


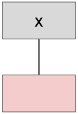
# Word embeddings

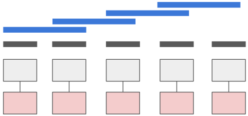



A series of convolutional filters with max pooling, concatenated to form the initial representation



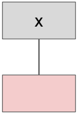
# Word embeddings



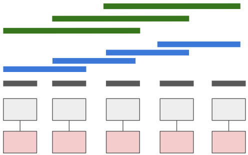



A series of convolutional filters with max pooling, concatenated to form the initial representation

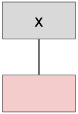
# Word embeddings

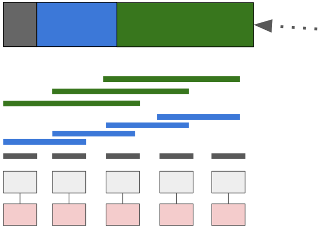



A series of convolutional filters with max pooling, concatenated to form the initial representation



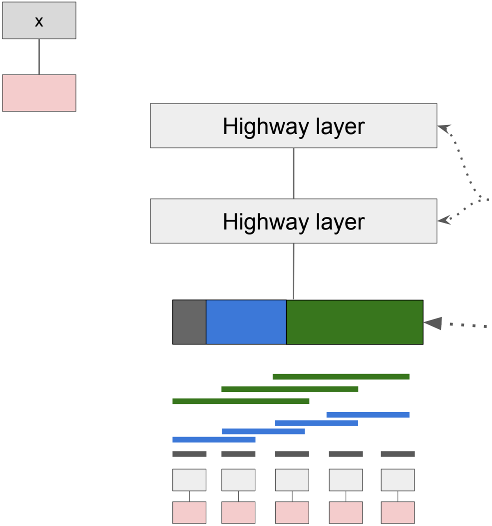
# Word embeddings



A series of convolutional filters with max pooling, concatenated to form the initial representation

# Word embeddings

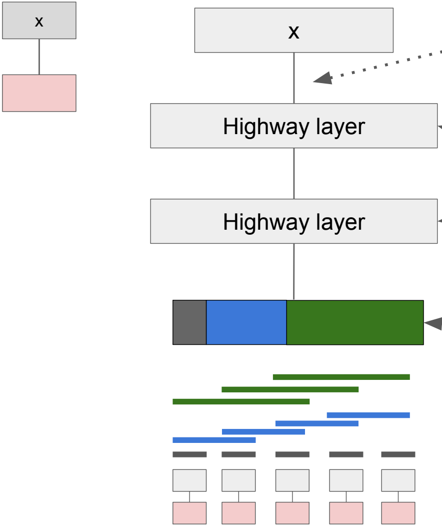



Highway layers introduce gating information between layers:

$$(cW + b)(cW_T + b_T) + c(1 - (cW_T + b_T))$$

A series of convolutional filters with max pooling, concatenated to form the initial representation

# Word embeddings



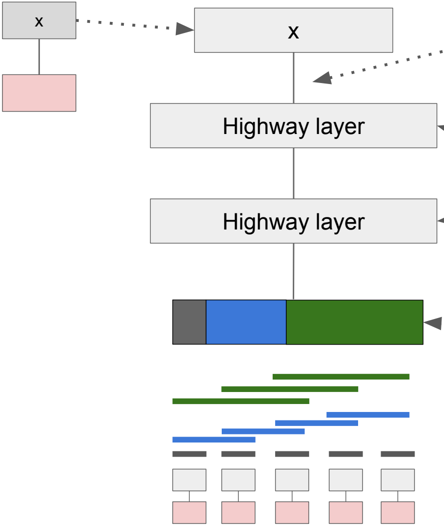

Linear projection to the embedding dimensionality, which must be double the RNN hidden dimensionality

Highway layers introduce gating information between layers:

$$(cW + b)(cW_T + b_T) + c(1 - (cW_T + b_T))$$

A series of convolutional filters with max pooling, concatenated to form the initial representation

# Word embeddings




Linear projection to the embedding dimensionality, which must be double the RNN hidden dimensionality

Highway layers introduce gating information between layers:

$$(cW + b)(cW_T + b_T) + c(1 - (cW_T + b_T))$$

A series of convolutional filters with max pooling, concatenated to form the initial representation

# ELMo model releases

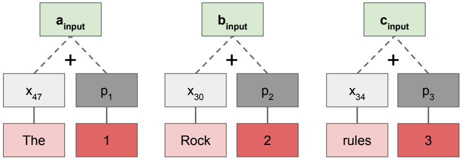
| | | LSTM | | |
|---|---|---|---|---|
| **Model** | **Parameters** | **Hidden size** | **Output size** | **Highway layers** |
| Small | 13.6M | 1024 | 128 | 1 |
| Medium | 28.0M | 2048 | 256 | 1 |
| Original | 93.6M | 4096 | 512 | 2 |
| Original (5.5B) | 93.6M | 4096 | 512 | 2 |

Additional details at `https://allennlp.org/elmo`; the options files reveal additional information about the subword convolutional filters, activation functions, thresholds, and layer dimensions.

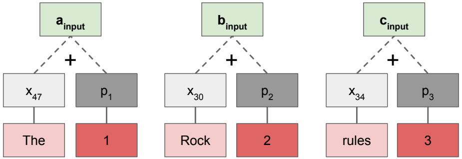
# Transformers

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

# Core model structure

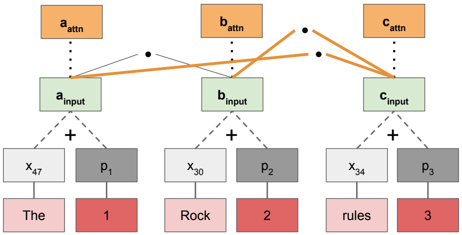


# Core model structure




$$c_{\text{input}} = x_{34} + p_3$$

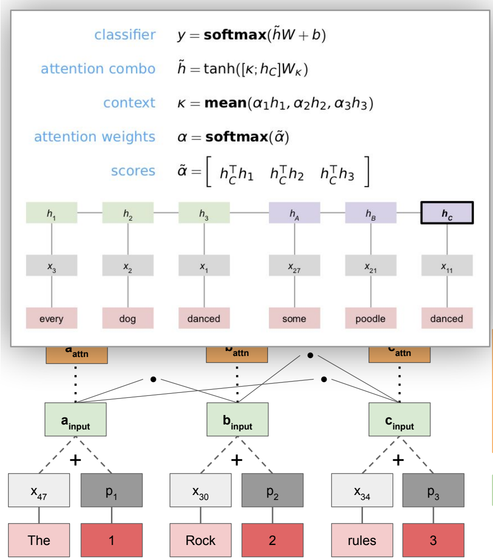
# Core model structure




$$\mathrm{c}_{\mathrm{attn}} = \mathbf{sum}\left([\alpha_1 \mathrm{a}_{\mathrm{input}}, \alpha_2 \mathrm{b}_{\mathrm{input}}]\right)$$
$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$
$$\tilde{\alpha} = \left[\frac{{\mathrm{c}_{\mathrm{input}}}^{\top}\mathrm{a}_{\mathrm{input}}}{\sqrt{d_k}}, \frac{{\mathrm{c}_{\mathrm{input}}}^{\top}\mathrm{b}_{\mathrm{input}}}{\sqrt{d_k}}\right]$$

$$\mathrm{c}_{\mathrm{input}} = x_{34} + p_3$$

# Core model structure



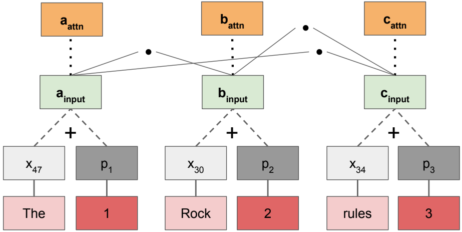

$$\mathbf{c}_{\text{attn}} = \mathbf{sum}\left([\alpha_1 \mathbf{a}_{\text{input}}, \alpha_2 \mathbf{b}_{\text{input}}]\right)$$

$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{\mathbf{c}_{\text{input}}{}^\top \mathbf{a}_{\text{input}}}{\sqrt{d_k}}, \frac{\mathbf{c}_{\text{input}}{}^\top \mathbf{b}_{\text{input}}}{\sqrt{d_k}}\right]$$

$$\mathbf{c}_{\text{input}} = x_{34} + p_3$$

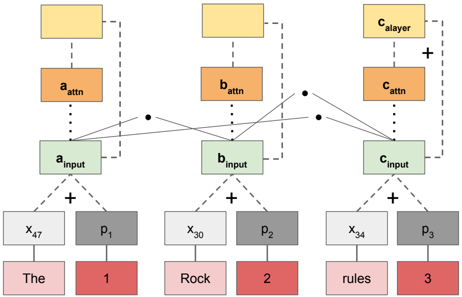
# Core model structure




$$
\begin{aligned}
\mathrm{c}_{\text{attn}} &= \textbf{sum}\left([\alpha_1 \mathrm{a}_{\text{input}}, \alpha_2 \mathrm{b}_{\text{input}}]\right) \\
\alpha &= \textbf{softmax}(\tilde{\alpha}) \\
\tilde{\alpha} &= \left[\frac{{\mathrm{c}_{\text{input}}}^{\top} \mathrm{a}_{\text{input}}}{\sqrt{d_k}}, \frac{{\mathrm{c}_{\text{input}}}^{\top} \mathrm{b}_{\text{input}}}{\sqrt{d_k}}\right]
\end{aligned}
$$

$$\mathrm{c}_{\text{input}} = x_{34} + p_3$$

# Core model structure




$$c_{\text{alayer}} = c_{\text{attn}} + \textbf{Dropout}(c_{\text{input}})$$

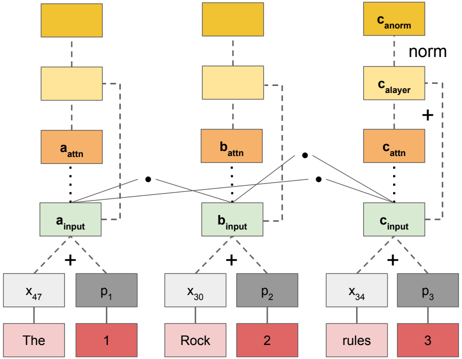
$$c_{\text{attn}} = \textbf{sum}\left([\alpha_1 a_{\text{input}}, \alpha_2 b_{\text{input}}]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{{c_{\text{input}}}^{\top} a_{\text{input}}}{\sqrt{d_k}}, \frac{{c_{\text{input}}}^{\top} b_{\text{input}}}{\sqrt{d_k}}\right]$$

$$c_{\text{input}} = x_{34} + p_3$$

# Core model structure



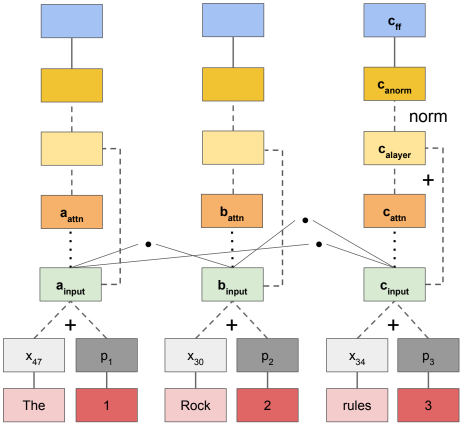

$$c_{\text{anorm}} = \frac{c_{\text{alayer}} - \textbf{mean}(c_{\text{alayer}})}{\textbf{std}(c_{\text{alayer}}) + \varepsilon}$$

$$c_{\text{alayer}} = c_{\text{attn}} + \textbf{Dropout}(c_{\text{input}})$$

$$c_{\text{attn}} = \textbf{sum}\left([\alpha_1 a_{\text{input}}, \alpha_2 b_{\text{input}}]\right)$$
$$\alpha = \textbf{softmax}(\tilde{\alpha})$$
$$\tilde{\alpha} = \left[\frac{{c_{\text{input}}}^\top a_{\text{input}}}{\sqrt{d_k}}, \frac{{c_{\text{input}}}^\top b_{\text{input}}}{\sqrt{d_k}}\right]$$

$$c_{\text{input}} = x_{34} + p_3$$

# Core model structure




$$\mathrm{c}_{\mathrm{ff}} = \mathbf{ReLU}(\mathrm{c}_{\mathrm{anorm}} W_1 + b_1) W_2 + b_2$$

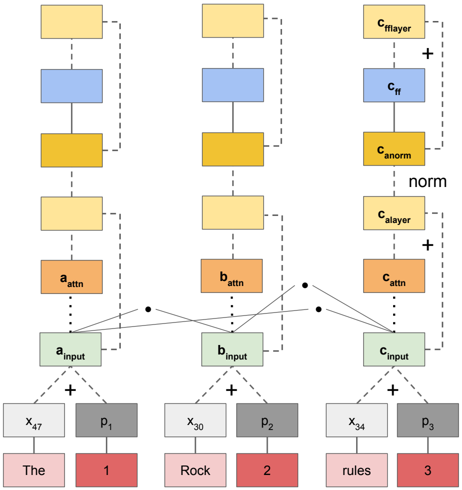
$$\mathrm{c}_{\mathrm{anorm}} = \frac{\mathrm{c}_{\mathrm{alayer}} - \mathbf{mean}(\mathrm{c}_{\mathrm{alayer}})}{\mathbf{std}(\mathrm{c}_{\mathrm{alayer}}) + \varepsilon}$$

$$\mathrm{c}_{\mathrm{alayer}} = \mathrm{c}_{\mathrm{attn}} + \mathbf{Dropout}(\mathrm{c}_{\mathrm{input}})$$

$$\mathrm{c}_{\mathrm{attn}} = \mathbf{sum}\left([\alpha_1 \mathrm{a}_{\mathrm{input}}, \alpha_2 \mathrm{b}_{\mathrm{input}}]\right)$$
$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$
$$\tilde{\alpha} = \left[\frac{{\mathrm{c}_{\mathrm{input}}}^{\top} \mathrm{a}_{\mathrm{input}}}{\sqrt{d_k}}, \frac{{\mathrm{c}_{\mathrm{input}}}^{\top} \mathrm{b}_{\mathrm{input}}}{\sqrt{d_k}}\right]$$

$$\mathrm{c}_{\mathrm{input}} = x_{34} + p_3$$

# Core model structure




$$c_{\text{fflayer}} = c_{\text{anorm}} + \textbf{Dropout}(c_{\text{ff}})$$

$$c_{\text{ff}} = \textbf{ReLU}(c_{\text{anorm}} W_1 + b_1) W_2 + b_2$$

$$c_{\text{anorm}} = \frac{c_{\text{alayer}} - \textbf{mean}(c_{\text{alayer}})}{\textbf{std}(c_{\text{alayer}}) + \varepsilon}$$

$$c_{\text{alayer}} = c_{\text{attn}} + \textbf{Dropout}(c_{\text{input}})$$

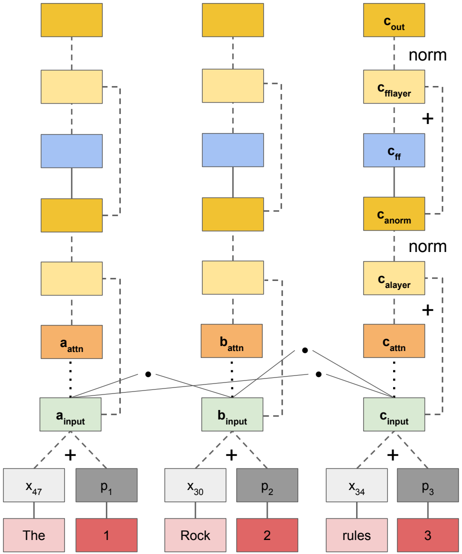
$$c_{\text{attn}} = \textbf{sum}\left([\alpha_1 a_{\text{input}}, \alpha_2 b_{\text{input}}]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{{c_{\text{input}}}^{\top} a_{\text{input}}}{\sqrt{d_k}}, \frac{{c_{\text{input}}}^{\top} b_{\text{input}}}{\sqrt{d_k}}\right]$$

$$c_{\text{input}} = x_{34} + p_3$$

# Core model structure




$$\mathrm{c}_{\text{out}} = \frac{\mathrm{c}_{\text{fflayer}} - \textbf{mean}(\mathrm{c}_{\text{fflayer}})}{\textbf{std}(\mathrm{c}_{\text{fflayer}}) + \varepsilon}$$

$$\mathrm{c}_{\text{fflayer}} = \mathrm{c}_{\text{anorm}} + \textbf{Dropout}(\mathrm{c}_{\text{ff}})$$

$$\mathrm{c}_{\text{ff}} = \textbf{ReLU}(\mathrm{c}_{\text{anorm}} W_1 + b_1) W_2 + b_2$$

$$\mathrm{c}_{\text{anorm}} = \frac{\mathrm{c}_{\text{alayer}} - \textbf{mean}(\mathrm{c}_{\text{alayer}})}{\textbf{std}(\mathrm{c}_{\text{alayer}}) + \varepsilon}$$

$$\mathrm{c}_{\text{alayer}} = \mathrm{c}_{\text{attn}} + \textbf{Dropout}(\mathrm{c}_{\text{input}})$$

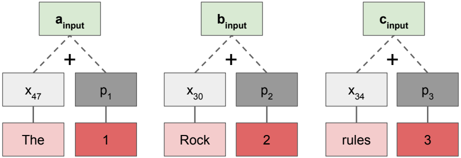
$$\mathrm{c}_{\text{attn}} = \textbf{sum}\left([\alpha_1 \mathrm{a}_{\text{input}}, \alpha_2 \mathrm{b}_{\text{input}}]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

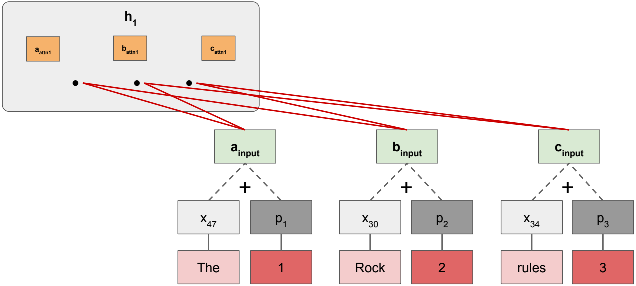
$$\tilde{\alpha} = \left[\frac{{\mathrm{c}_{\text{input}}}^{\top} \mathrm{a}_{\text{input}}}{\sqrt{d_k}}, \frac{{\mathrm{c}_{\text{input}}}^{\top} \mathrm{b}_{\text{input}}}{\sqrt{d_k}}\right]$$

$$\mathrm{c}_{\text{input}} = x_{34} + p_3$$

# Multi-headed attention



# Multi-headed attention

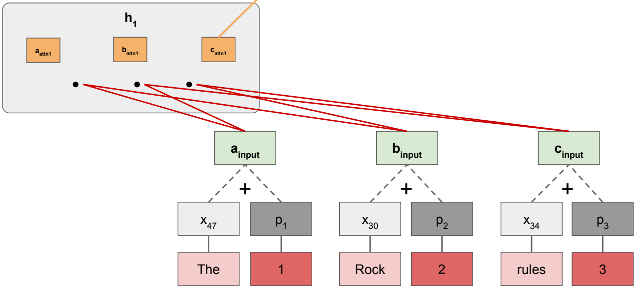


# Multi-headed attention

$$\mathrm{c}_{\mathrm{attn1}} = \mathbf{sum}\left(\left[\alpha_1(\mathrm{a}_{\mathrm{input}}W_1^V), \alpha_2(\mathrm{b}_{\mathrm{input}}W_1^V)\right]\right)$$

$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(\mathrm{c}_{\mathrm{input}}W_1^Q)^\top(\mathrm{a}_{\mathrm{input}}W_1^K)}{\sqrt{d_k}}, \frac{(\mathrm{c}_{\mathrm{input}}W_1^Q)^\top(\mathrm{b}_{\mathrm{input}}W_1^K)}{\sqrt{d_k}}\right]$$

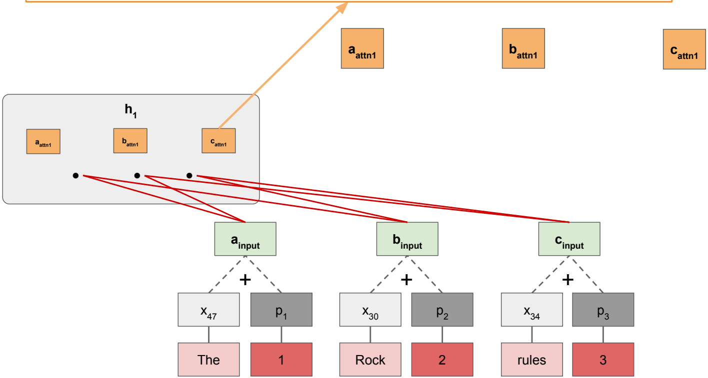


# Multi-headed attention

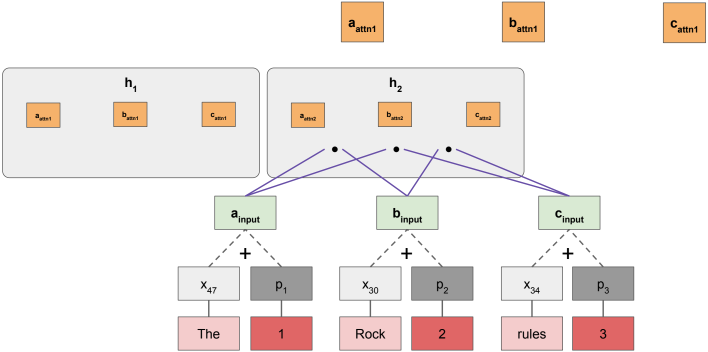
$$\mathrm{c}_{\mathrm{attn1}} = \mathbf{sum}\left(\left[\alpha_1(\mathrm{a}_{\mathrm{input}} W_1^V), \alpha_2(\mathrm{b}_{\mathrm{input}} W_1^V)\right]\right)$$

$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(\mathrm{c}_{\mathrm{input}} W_1^Q)^\top (\mathrm{a}_{\mathrm{input}} W_1^K)}{\sqrt{d_k}}, \frac{(\mathrm{c}_{\mathrm{input}} W_1^Q)^\top (\mathrm{b}_{\mathrm{input}} W_1^K)}{\sqrt{d_k}}\right]$$



# Multi-headed attention

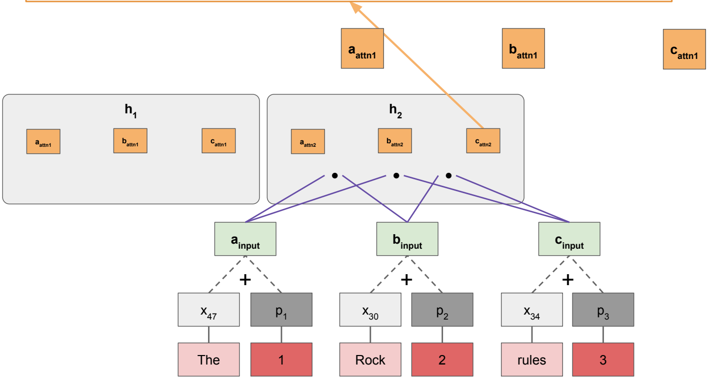


# Multi-headed attention

$$c_{\text{attn2}} = \textbf{sum}\left(\left[\alpha_1(a_{\text{input}}W_2^V), \alpha_2(b_{\text{input}}W_2^V)\right]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(c_{\text{input}}W_2^Q)^\top(a_{\text{input}}W_2^K)}{\sqrt{d_k}}, \frac{(c_{\text{input}}W_2^Q)^\top(b_{\text{input}}W_2^K)}{\sqrt{d_k}}\right]$$

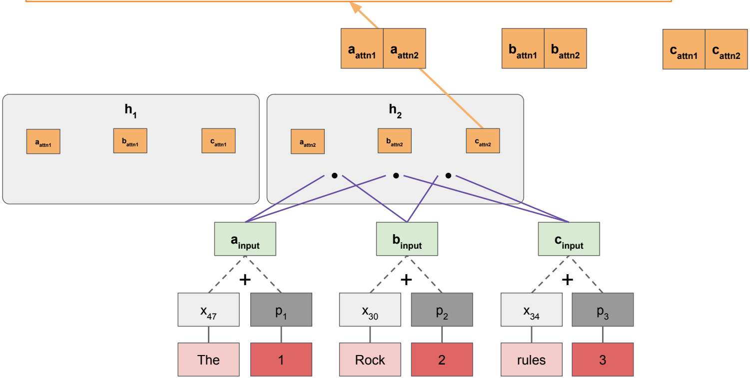


# Multi-headed attention

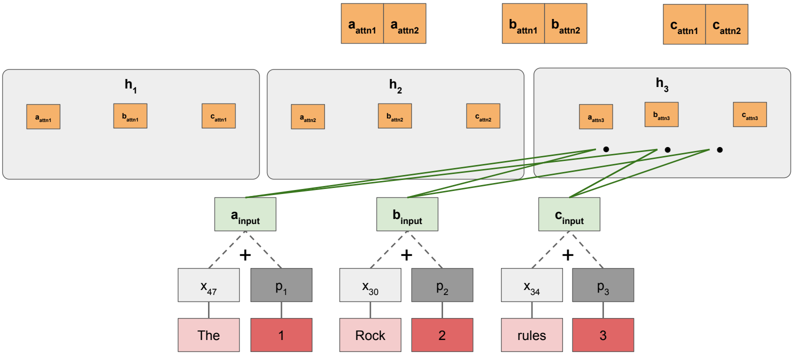
$$\mathrm{c}_{\mathrm{attn2}} = \mathbf{sum}\left(\left[\alpha_1(\mathrm{a}_{\mathrm{input}} W_2^V), \alpha_2(\mathrm{b}_{\mathrm{input}} W_2^V)\right]\right)$$

$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(\mathrm{c}_{\mathrm{input}} W_2^Q)^\top(\mathrm{a}_{\mathrm{input}} W_2^K)}{\sqrt{d_k}}, \frac{(\mathrm{c}_{\mathrm{input}} W_2^Q)^\top(\mathrm{b}_{\mathrm{input}} W_2^K)}{\sqrt{d_k}}\right]$$



# Multi-headed attention

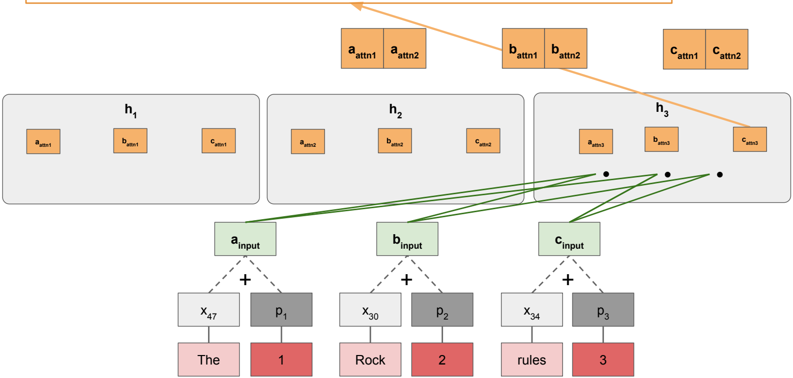


# Multi-headed attention

$$c_{\text{attn3}} = \textbf{sum}\left(\left[\alpha_1(a_{\text{input}} W_3^V), \alpha_2(b_{\text{input}} W_3^V)\right]\right)$$

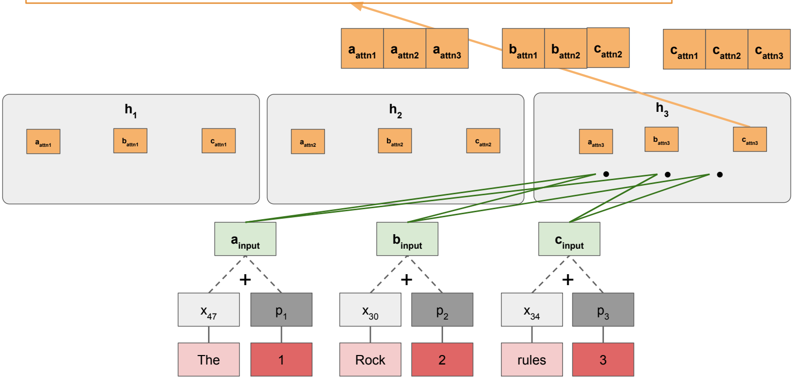
$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(c_{\text{input}} W_3^Q)^\top (a_{\text{input}} W_3^K)}{\sqrt{d_k}}, \frac{(c_{\text{input}} W_3^Q)^\top (b_{\text{input}} W_3^K)}{\sqrt{d_k}}\right]$$



# Multi-headed attention

$$\mathrm{c_{attn3}} = \mathbf{sum}\left(\left[\alpha_1(\mathrm{a_{input}} W_3^V), \alpha_2(\mathrm{b_{input}} W_3^V)\right]\right)$$
$$\alpha = \mathbf{softmax}(\tilde{\alpha})$$
$$\tilde{\alpha} = \left[\frac{(\mathrm{c_{input}} W_3^Q)^\top (\mathrm{a_{input}} W_3^K)}{\sqrt{d_k}}, \frac{(\mathrm{c_{input}} W_3^Q)^\top (\mathrm{b_{input}} W_3^K)}{\sqrt{d_k}}\right]$$

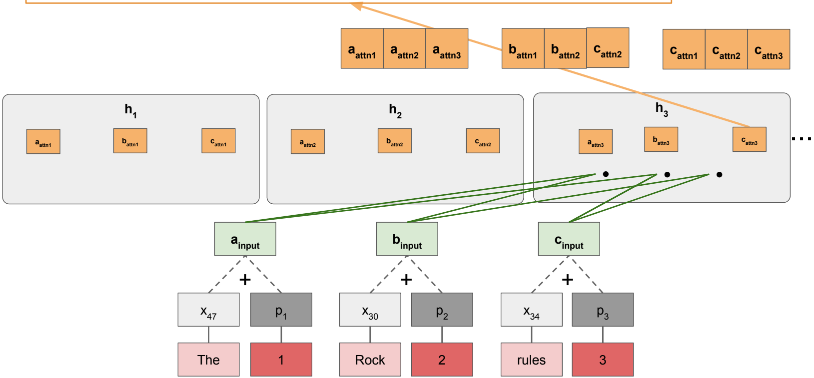


# Multi-headed attention

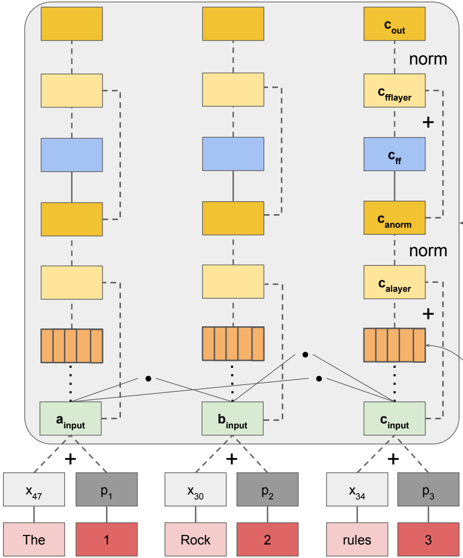
$$c_{\text{attn3}} = \textbf{sum}\left(\left[\alpha_1(a_{\text{input}} W_3^V), \alpha_2(b_{\text{input}} W_3^V)\right]\right)$$

$$\alpha = \textbf{softmax}(\tilde{\alpha})$$

$$\tilde{\alpha} = \left[\frac{(c_{\text{input}} W_3^Q)^\top (a_{\text{input}} W_3^K)}{\sqrt{d_k}}, \frac{(c_{\text{input}} W_3^Q)^\top (b_{\text{input}} W_3^K)}{\sqrt{d_k}}\right]$$



# Repeated transformer blocks




Repeated 6 times, with $c_{out}$ serving as $c_{input}$ to each successive layer

Reminder that we also do multi-headed attention in each layer

# The architecture diagram

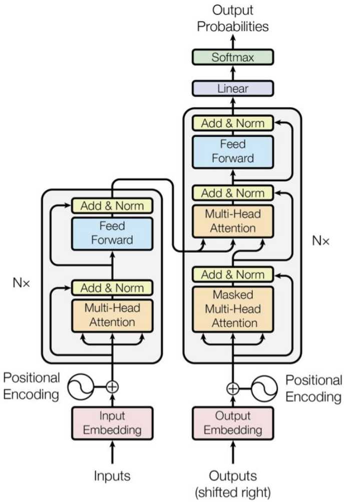



Figure 1: The Transformer - model architecture.

# The architecture diagram

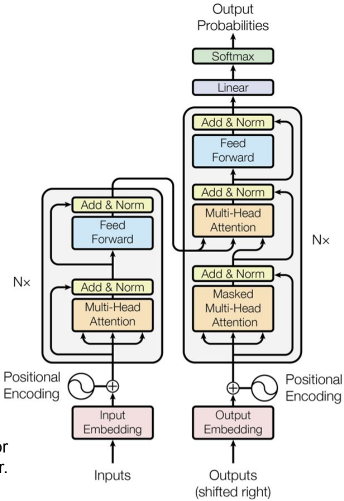



Figure 1: The Transformer - model architecture.

The left side is repeated for every state in the encoder.

# The architecture diagram

Each decoder state self-attends
with all of its fellow decoder states
and with all the encoder states.

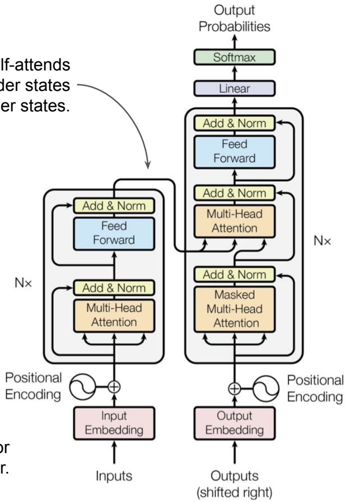
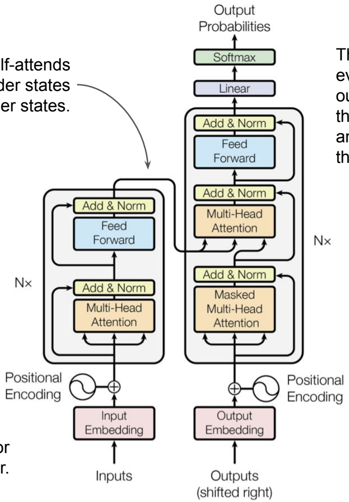



Figure 1: The Transformer - model architecture.

The left side is repeated for
every state in the encoder.

# The architecture diagram

Each decoder state self-attends with all of its fellow decoder states and with all the encoder states.




The right side is repeated for every decoder state, with outputs for each state that has them (all of them for dialogue and machine translation, only the final one for NLI).

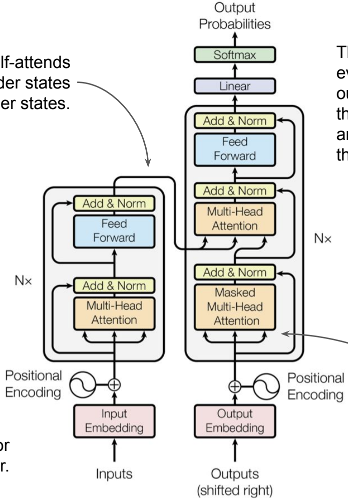
The left side is repeated for every state in the encoder.

Figure 1: The Transformer - model architecture.

# The architecture diagram

Each decoder state self-attends with all of its fellow decoder states and with all the encoder states.

The right side is repeated for every decoder state, with outputs for each state that has them (all of them for dialogue and machine translation, only the final one for NLI).



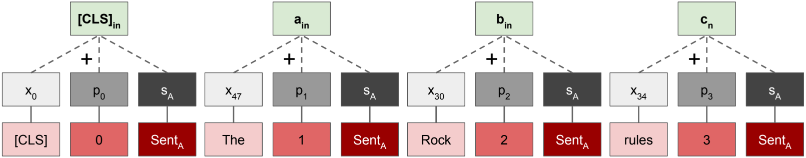

The left side is repeated for every state in the encoder.

In the decoder, self-attention is limited to preceding words.

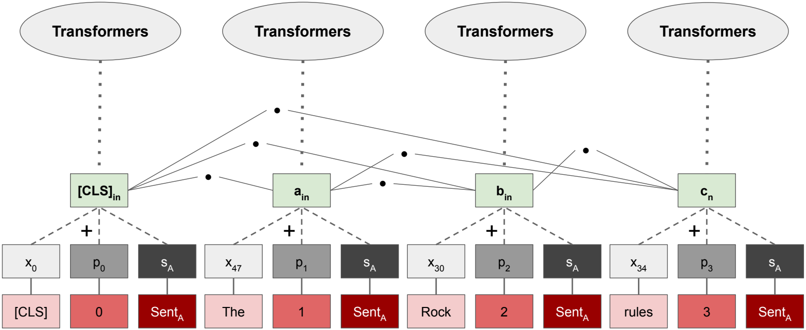
Figure 1: The Transformer - model architecture.

# BERT

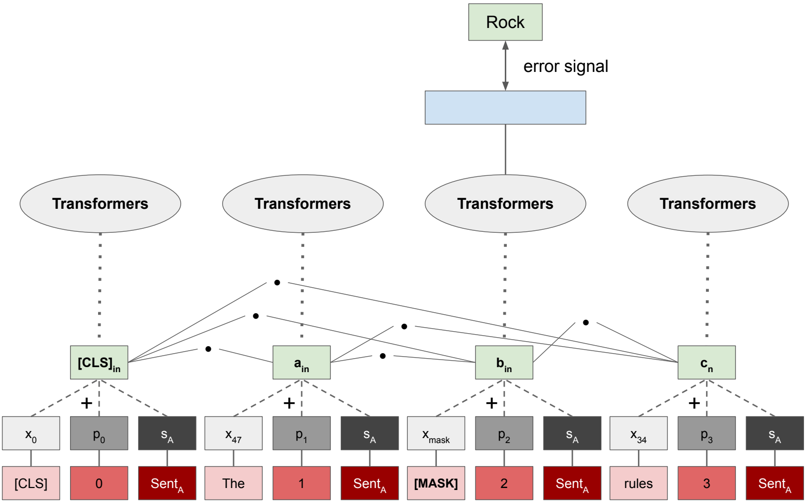
# Core model structure



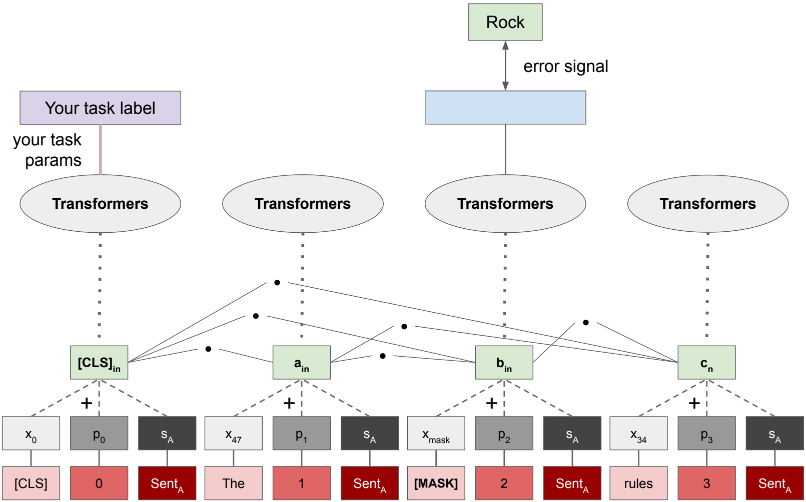
# Core model structure



# Masked Language Modeling (MLM)



# Transfer learning and fine-tuning



# Binary sentence prediction pretraining

## Positive: Actual sentence sequences

- [CLS] the man went to [MASK] store [SEP]
- he bought a gallon [MASK] milk [SEP]
- Label: `IsNext`

## Negative: Randomly chosen second sentence

- [CLS] the man went to [MASK] store [SEP]
- penguin [MASK] are flight ##less birds [SEP]
- Label: `NotNext`

# Tokenization and the BERT embedding space

```
In [1]: import random
        # In the code from https://github.com/google-research/bert
        from tokenization import FullTokenizer

In [2]: vocab_filename = "uncased_L-12_H-768_A-12/vocab.txt"

In [3]: with open(vocab_filename) as f:
            vocab = f.read().splitlines()

In [4]: len(vocab)

Out[4]: 30522

In [5]: random.sample(vocab, 5)

Out[5]: ['folder', '##gged', 'principles', 'moving', '##ceae']

In [6]: tokenizer = FullTokenizer(vocab_file=vocab_filename, do_lower_case=True)

In [7]: tokenizer.tokenize("This isn't too surprising!")

Out[7]: ['this', 'isn', "'", 't', 'too', 'surprising', '!']

In [8]: tokenizer.tokenize("Does BERT know Snuffleupagus?")

Out[8]: ['does', 'bert', 'know', 's', '##nu', '##ffle', '##up', '##ag', '##us', '?']
```

# BERT model releases

## Base

- Transformer layers: 12
- Hidden representations: 768 dimensions
- Attention heads: 12
- Total parameters: 110M

## Large

- Transformer layers: 24
- Hidden representations: 1024 dimensions
- Attention heads: 16
- Total parameters: 340M

Limited to sequences of 512 tokens due to dimensionality of the positional embeddings.

# contextualreps.ipynb

1. Overview: Resources and guiding insights
2. ELMo: **E**mbeddings from Language **Mo**dels
3. Transformers
4. BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
5. `contextualreps.ipynb`: Easy ways to bring ELMo and BERT into your project

# Guiding idea

- Your existing architecture can benefit from contextual representations.
- `contextualreps.ipynb` shows you how to bring in ELMo and BERT representations.
- You don't get the benefits of fine-tuning (for that, you need to integrate more fully with ELMo and BERT code), but you still get a reliable boost!

# Standard RNN dataset preparation

| | Embedding | | |
|---|---|---|---|
| 1 | −0.42 | 0.10 | 0.12 |
| 2 | −0.16 | −0.21 | 0.29 |
| 3 | −0.26 | 0.31 | 0.37 |

**Examples**

[a, b, a]
[b, c]

$\Downarrow$

**Indices**

[1, 2, 1]
[2, 3]

$\Downarrow$

**Vectors**

$\Big[[-0.42\ 0.10\ 0.12], [-0.16\ -0.21\ 0.29], [-0.42\ 0.10\ 0.12]\Big]$

$\Big[[-0.16\ -0.21\ 0.29], [-0.26\ 0.31\ 0.37]\Big]$

# RNN contextual representation inputs

**Examples**

[a, b, a]
[b, c]

$$\Downarrow$$

**Vectors**

$$\Big[[-0.41\ -0.08\ 0.27], [0.17\ -0.22\ 0.78][-0.46\ 0.24\ 0.12]\Big]$$
$$\Big[[-0.02\ -0.56\ 0.11][-0.45\ 0.43\ 0.32]\Big]$$

# Code snippet: ELMo RNN inputs

```
In [1]: from allennlp.commands.elmo import ElmoEmbedder
        import os
        import sst
        from torch_rnn_classifier import TorchRNNClassifier

In [2]: SST_HOME = os.path.join("data", "trees")

In [3]: elmo = ElmoEmbedder()

In [4]: def elmo_phi(tree):
            vecs = elmo.embed_sentence(tree.leaves())
            return vecs.mean(axis=0)

In [5]: def fit_rnn(X, y):
            mod = TorchRNNClassifier(vocab=[], max_iter=50, use_embedding=False)
            mod.fit(X, y)
            return mod
```

# Code snippet: ELMo RNN inputs

```
In [6]: elmo_experiment = sst.experiment(
            SST_HOME,
            elmo_phi,
            fit_rnn,
            train_reader=sst.train_reader,
            assess_reader=sst.dev_reader,
            vectorize=False)

Finished epoch 50 of 50; error is 0.07357715629041195

              precision    recall  f1-score   support


    negative      0.700     0.687     0.693       428
     neutral      0.353     0.284     0.315       229
    positive      0.710     0.795     0.750       444

   micro avg      0.647     0.647     0.647      1101
   macro avg      0.588     0.589     0.586      1101
weighted avg      0.632     0.647     0.638      1101
```

# Code snippet: BERT RNN inputs

```
In [1]: # bert-serving-start -model_dir data/bert/uncased_L-12_H-768_A-12/ \
        # -pooling_strategy NONE -max_seq_len NONE -show_tokens_to_client
        from bert_serving.client import BertClient
        import os
        import sst
        from torch_rnn_classifier import TorchRNNClassifier

In [2]: SST_HOME = os.path.join("data", "trees")

In [3]: # Load the train and dev sets as strings, to let BERT tokenize:
        sst_train = [(" ".join(t.leaves()), label) for t, label in sst.train_reader(SST_HOME)]
        sst_dev = [(" ".join(t.leaves()), label) for t, label in sst.dev_reader(SST_HOME)]

In [4]: X_str_train, y_train = zip(*sst_train)
        X_str_dev, y_dev = zip(*sst_dev)

In [5]: X_str_dev, y_dev = zip(*sst_dev)
```

# Code snippet: BERT RNN inputs

```
In [6]: bc = BertClient(check_length=False)

In [7]: # Prefetch all the BERT representations:
        X_bert_train = bc.encode(list(X_str_train), show_tokens=False)
        X_bert_dev = bc.encode(list(X_str_dev), show_tokens=False)

In [8]: # Create a look-up for fast featurization:
        BERT_LOOKUP = {}
        for sents, reps in ((X_str_train, X_bert_train), (X_str_dev, X_bert_dev)):
            assert len(sents) == len(reps)
            for s, rep in zip(sents, reps):
                BERT_LOOKUP[s] = rep
```

# Code snippet: BERT RNN inputs

```
In [9]: def bert_phi(tree):
            s = " ".join(tree.leaves())
            return BERT_LOOKUP[s]

In [10]: def fit_rnn(X, y):
             mod = TorchRNNClassifier(vocab=[], max_iter=50, use_embedding=False)
             mod.fit(X, y)
             return mod

In [11]: bert_rnn_experiment = sst.experiment(
             SST_HOME,
             bert_phi,
             fit_rnn,
             train_reader=sst.train_reader,
             assess_reader=sst.dev_reader,
             vectorize=False)

Finished epoch 50 of 50; error is 2.6541710644960403

              precision    recall  f1-score   support

    negative      0.767     0.668     0.714       428
     neutral      0.322     0.323     0.322       229
    positive      0.737     0.827     0.779       444

   micro avg      0.660     0.660     0.660      1101
   macro avg      0.608     0.606     0.605      1101
weighted avg      0.662     0.660     0.659      1101
```

# References I

Devlin, Jacob, Ming-Wei Chang, Kenton Lee & Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the north american association of computational linguistics*, Stroudsburg, PA: Association for Computational Linguistics.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee & 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)*, 2227–2237. Association for Computational Linguistics. http://aclweb.org/anthology/N18-1202.

Smith, Noah A. 2019. Contextual word representations: A contextual introduction. ArXiv:1902.06006v2.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser & Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (eds.), *Advances in neural information processing systems 30*, 5998–6008. Curran Associates, Inc. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.