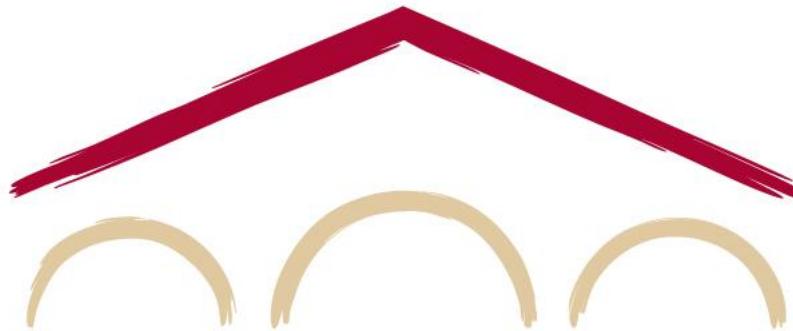


Assign/bakeoff 1 overview



Christopher Potts
CS224u: Natural Language Understanding

Homework and bakeoff: Multi-domain sentiment

```
__author__ = "Christopher Potts"  
__version__ = "CS224u, Stanford, Spring 2023"
```

 Open in Colab

 Open

Studio Lab

Background resources

<https://web.stanford.edu/class/cs224u/background.html>

Basic tools

1. Notebook: [Jupyter notebook tutorial](#)
2. Notebook: [NumPy tutorial](#)
3. Notebook: [PyTorch tutorial](#)
4. Notebook: [Using and extending the course PyTorch models](#)

Supervised learning

1. Tutorial videos on supervised learning
2. Stanford AI Lab Deep Learning Tutorial
3. Video: Overview of supervised sentiment analysis [\[slides\]](#)
4. Video: General practical tips [\[slides\]](#)
5. Video: Stanford Sentiment Treebank [\[slides\]](#)
6. Notebook: [Overview of the Stanford Sentiment Treebank](#)
7. Video: DynaSent [\[slides\]](#)
8. Video: sst.py [\[slides\]](#)
9. Video: Hyperparameter search and classifier comparison [\[slides\]](#)
10. Video: Feature representation [\[slides\]](#)
11. Notebook: [Hand-built feature functions](#)
12. Video: RNN classifiers [\[slides\]](#)
13. Notebook: [Dense feature representations and neural networks](#)
14. Video: Practical fine-tuning [\[slides\]](#)
15. Notebook: [Fine-tuning large language models](#)

Task setting

1. Ternary sentiment: positive/negative/neutral
2. Train/dev data:
 - a. DynaSent Round 1
 - b. DynaSent Round 2
 - c. Stanford Sentiment Treebank (SST)
3. Bakeoff test:
 - a. Mixture of the above
 - b. Mystery examples

Important methodological note

1. The DynaSent and SST test sets are public.
2. We are counting on people not to cheat by developing their models on these test sets.
3. Evaluate exactly once on the test set and turn in the results, with no further system tuning or additional runs.
4. *Much of the scientific integrity of our field depends on people adhering to this honor code.*

Data loading

```
dynasent_r1 = load_dataset("dynabench/dynasent", 'dynabench.dynasent.r1.all')
```

```
dynasent_r2 = load_dataset("dynabench/dynasent", 'dynabench.dynasent.r2.all')
```

```
sst = load_dataset("SetFit/sst5")
```

```
print_label_dist(dynasent_r1)
```

train

negative: 14021
neutral: 45076
positive: 21391

validation

negative: 1200
neutral: 1200
positive: 1200

```
print_label_dist(dynasent_r2)
```

train

negative: 4579
neutral: 2448
positive: 6038

validation

negative: 240
neutral: 240
positive: 240

```
print_label_dist(sst)
```

train

negative: 3310
neutral: 1624
positive: 3610

validation

negative: 428
neutral: 229
positive: 444

Question 1: Linear classifiers

Our first set of experiments will use simple linear classifiers with sparse representations derived from counting unigrams. These experiments will introduce some useful techniques and provide a baseline for original systems.

Question 1: Linear classifiers

Background: Feature functions

Background: Feature space
vectorization

Background: scikit-learn models

Background: Classifier assessment

Task 1: Feature functions [1 point]

Task 2: Model training [1 point]

Task 3: Model assessment [1 point]

Question 1, Task 1: Feature functions

```
def unigrams_phi(s):
    """The basis for a unigrams feature function.

    Downcases all tokens.

    Parameters
    -----
    s : str
        The example to represent

    Returns
    -----
    Counter
        A map from tokens (str) to their counts in `text`

    """
    return Counter(s.lower().split())
```

Question 1, Task 1: Feature functions

```
def unigrams_phi(s):
    """The basis for a unigrams feature function.

    Downcase
    Parameters
    -----
    s : str
        The text to process.

    Returns
    -----
    Counter
        A map from tokens to their counts in `text`.

    """
    return Counter(s.lower().split())

def tweetgrams_phi(s, **kwargs):
    """The basis for a feature function using `TweetTokenizer`.

    Parameters
    -----
    s : str
    kwargs : dict
        Passed to `TweetTokenizer`

    Returns
    -----
    Counter
        A map from tokens to their counts in `text`.

    """
    pass
##### YOUR CODE HERE
```

Unit tests!

```
test_tweetgrams_phi(tweetgrams_phi)
```

Question 1, Task 2: Model training

Question 1: Linear classifiers

Background: Feature functions

Background: Feature space vectorization

Background: scikit-learn models

Background: Classifier assessment

Task 1: Feature functions [1 point]

Task 2: Model training [1 point]

Task 3: Model assessment [1 point]

```
def train_linear_model(model, featfunc, train_dataset):
    """Train an sklearn classifier.

    Parameters
    -----
    model : sklearn classifier model
    featfunc : func
        Maps strings to Counter instances
    train_dataset: dict
        Must have a key "sentence" containing strings that `featfunc` will process, and a key "gold_label" giving labels

    Returns
    -----
    tuple
        * A trained version of `model`
        * A fitted `vectorizer` for the train set
    """

    pass
    # Step 1: Featurize all the examples in `train_dataset['sentence']`
    ##### YOUR CODE HERE

    # Step 2: Instantiate and use a `DictVectorizer`:
    ##### YOUR CODE HERE

    # Step 3: Train the model on the feature matrix and
    # train_dataset['gold_label']:
    ##### YOUR CODE HERE

    # Step 4: Return (model, vectorizer):
    ##### YOUR CODE HERE
```

Question 1, Task 3: Model assessment

Question 1: Linear classifiers

Background: Feature functions

Background: Feature space
vectorization

Background: scikit-learn models

Background: Classifier assessment

Task 1: Feature functions [1 point]

Task 2: Model training [1 point]

Task 3: Model assessment [1 point]

```
def assess_linear_model(model, featfunc, vectorizer, assess_dataset):
    """Assess a trained sklearn model.

    Parameters
    -----
    model: trained sklearn model
    featfunc : func
        Maps strings to count dicts
    vectorizer : fitted DictVectorizer
    assess_dataset: dict
        Must have a key "sentence" containing strings that `featfunc`  

        will process, and a key "gold_label" giving labels

    Returns
    -----
    A classification report (multiline string)
    """
    pass
    # Step 1: Featurize the assessment data:
    ##### YOUR CODE HERE

    # Step 2: Vectorize the assessment data features:
    ##### YOUR CODE HERE

    # Step 3: Make predictions:
    ##### YOUR CODE HERE

    # Step 4: Return a classification report (str):
    ##### YOUR CODE HERE
```

Question 2: Transformer fine-tuning

We're now going to move into a more modern mode: fine-tuning pretrained components.

We'll use BERT-mini (originally from [the BERT repo](#)) for the homework so that we can rapidly develop prototypes. You can then consider scaling up to larger models.

Question 2: Transformer fine-tuning

Background: Tokenization

Background: Representation

Background: Masking

Task 1: Batch tokenization [1 point]

Task 2: Contextual representations [1 point]

Task 3: Fine-tuning module [1 point]

Optional use: Classifier interface

Question 2, Task 1: Batch tokenization

Question 2: Transformer fine-tuning

Background: Tokenization

Background: Representation

Background: Masking

Task 1: Batch tokenization [1 point]

Task 2: Contextual representations [1 point]

Task 3: Fine-tuning module [1 point]

Optional use: Classifier interface

```
def get_batch_token_ids(batch, tokenizer):
    """Map `batch` to a tensor of ids. The return
    value should meet the following specification:

    1. The max length should be 512.
    2. Examples longer than the max length should be truncated
    3. Examples should be padded to the max length for the batch.
    4. The special [CLS] should be added to the start and the special
       token [SEP] should be added to the end.
    5. The attention mask should be returned
    6. The return value of each component should be a tensor.

    Parameters
    -----
    batch: list of str
    tokenizer: Hugging Face tokenizer

    Returns
    -----
    dict with at least "input_ids" and "attention_mask" as keys,
    each with Tensor values

    """
pass
##### YOUR CODE HERE
```

Question 2, Task 2: Representation

Question 2: Transformer fine-tuning

Background: Tokenization

Background: Representation

Background: Masking

Task 1: Batch tokenization [1 point]

Task 2: Contextual representations [1 point]

Task 3: Fine-tuning module [1 point]

Optional use: Classifier interface

```
def get_reps(dataset, model, tokenizer, batchsize=20):
    """Represent each example in `dataset` with the final hidden state
    above the [CLS] token.

    Parameters
    -----
    dataset : list of str
    model : BertModel
    tokenizer : BertTokenizerFast
    batchsize : int

    Returns
    -----
    torch.Tensor with shape `(n_examples, dim)` where `dim` is the
    dimensionality of the representations for `model`

    .....
    data = []
    with torch.no_grad():
        pass
        # Iterate over `dataset` in batches:
        ##### YOUR CODE HERE

        # Encode the batch with `get_batch_token_ids`:
        ##### YOUR CODE HERE

        # Get the representations from the model, making
        # sure to pay attention to masking:
        ##### YOUR CODE HERE

    # Return a single tensor:
    ##### YOUR CODE HERE
```

Question 2, Task 3: Fine-tuning module

Question 2: Transformer fine-tuning

Background: Tokenization

Background: Representation

Background: Masking

Task 1: Batch tokenization [1 point]

Task 2: Contextual representations [1 point]

Task 3: Fine-tuning module [1 point]

Optional use: Classifier interface

```
import torch.nn as nn

class BertClassifierModule(nn.Module):
    def __init__(self,
                 n_classes,
                 hidden_activation,
                 weights_name="prajjwal1/bert-mini"):
        """This module loads a Transformer based on `weights_name`, puts it in train mode, add a dense layer with activation function give by `hidden_activation`, and puts a classifier layer on top of that as the final output. The output of the dense layer should have the same dimensionality as the model input.

    Parameters
    -----
    n_classes : int
        Number of classes for the output layer
    hidden_activation : torch activation function
        e.g., nn.Tanh()
    weights_name : str
        Name of pretrained model to load from Hugging Face

    """
        super().__init__()
        self.n_classes = n_classes
        self.weights_name = weights_name
        self.bert = AutoModel.from_pretrained(self.weights_name)
        self.bert.train()
        self.hidden_activation = hidden_activation
        self.hidden_dim = self.bert.embeddings.word_embeddings.embedding_dim
        # Add the new parameters here using `nn.Sequential`.
        # We can define this layer as
        #
        # h = f(cW1 + b_h)
        # y = hW2 + b_y
        #
        # where c is the final hidden state above the [CLS] token,
        # W1 has dimensionality (self.hidden_dim, self.hidden_dim),
        # W2 has dimensionality (self.hidden_dim, self.n_classes),
        # f is the hidden activation function, and we rely on the
        # PyTorch loss function to add apply a softmax to y.
        self.classifier_layer = None
##### YOUR CODE HERE
```

```
def forward(self, indices, mask):
    """Process `indices` with `mask` by feeding these arguments to `self.bert` and then feeding the initial hidden state in `last_hidden_state` to `self.classifier_layer`.

    Parameters
    -----
    indices : tensor.LongTensor of shape (n_batch, k)
        Indices into the `self.bert` embedding layer. `n_batch` is the number of examples and `k` is the sequence length for this batch
    mask : tensor.LongTensor of shape (n_batch, d)
        Binary vector indicating which values should be masked.
        `n_batch` is the number of examples and `k` is the sequence length for this batch

    Returns
    -----
    tensor.FloatTensor
        Predicted values, shape `(n_batch, self.n_classes)`

    """
    pass
##### YOUR CODE HERE
```

Optional use: Classifier interface

The above module doesn't have functionality for processing data and fitting models. Our course code includes some general purpose code for adding these features. Here is an example that should work well with the module you wrote above. For more details on the design of these interfaces, see [tutorial_pytorch_models.ipynb](#).

Question 3: Original systems

Your task is to develop an original ternary sentiment classifier model. There are many options. The only rule:

You cannot make any use of the test sets for DynaSent-R1, DynaSent-R2, or SST-3, at any time during the course of development.

This needs to be an original system. It doesn't suffice to download code from the Web, retrain, and submit. You can build on others' code, but you have to do something new and meaningful with it.

Original system formatting instructions

```
# PLEASE MAKE SURE TO INCLUDE THE FOLLOWING BETWEEN THE START AND STOP COMMENTS:  
#   1) Textual description of your system.  
#   2) The code for your original system.  
# PLEASE MAKE SURE NOT TO DELETE OR EDIT THE START AND STOP COMMENTS  
  
# START COMMENT: Enter your system description in this cell.  
  
# STOP COMMENT: Please do not remove this comment.
```

Question 4: Bakeoff entry

```
bakeoff_df = pd.read_csv(  
    os.path.join("data", "sentiment", "cs224u-sentiment-test-unlabeled.csv"))  
  
bakeoff_df.head()
```

To enter the bakeoff, you simply need to use your original system to:

1. Add a column named 'prediction' to `cs224u-sentiment-test-unlabeled.csv` with your model predictions (which are strings in { `positive` , `negative` , `neutral` }). The existing columns should remain.
2. Save the file as `cs224u-sentiment-bakeoff-entry.csv`.

Submit the following files to Gradescope:

- `hw_sentiment.ipynb` (this notebook)
- `cs224u-sentiment-bakeoff-entry.csv` (bake-off output)

Please make sure you use these filenames. The autograder looks for files with these names.

Contextual word representations

Christopher Potts

Stanford Linguistics

CS224u: Natural language understanding



Guiding ideas

Static vector representations of words

1. Feature-based (sparse): Classical lexical representations
2. Count-based methods (sparse): PMI, TF-IDF, etc.
3. Classical dimensionality reduction (dense): PCA, SVD, LDA, etc.
4. Learned dimensionality reduction (dense): autoencoders, word2vec, GloVe, etc.

Hands-on review:

<https://web.stanford.edu/class/cs224u/background.html>

Word representations and context

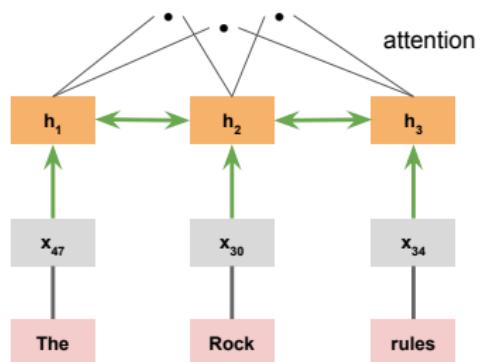
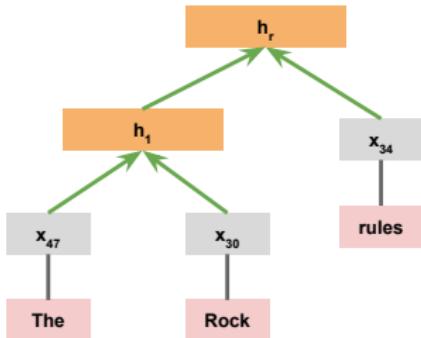
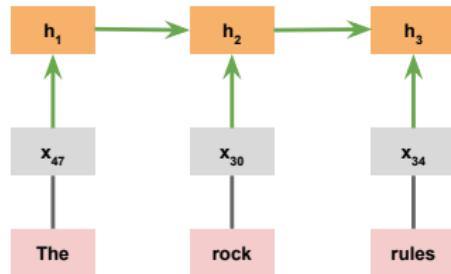
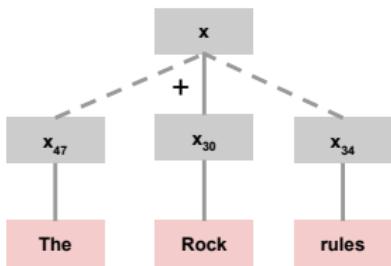
1. a. The vase broke.
b. Dawn broke.
c. The news broke.
d. Sandy broke the world record.
e. Sandy broke the law.
f. The burglar broke into the house.
g. The newscaster broke into the movie broadcast.
h. We broke even.
 2. a. flat tire/beer/note/surface
b. throw a party/fight/ball/fit
 3. a. A crane caught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



A brief history of contextual representation

1. November 2015: [Dai and Le \(2015\)](#) showed the value of LM-style pretraining for downstream tasks.
2. August 2017: [McCann et al. \(2017\)](#) (CoVe) pretrained bi-LSTMs for machine translation and showed that this was a useful start-state for downstream tasks.
3. February 2018: [Peters et al. \(2018\)](#) (ELMo) first showed how very large-scale pretraining of bidirectional LSTMs can lead to rich multipurpose representations.
4. June 2018: [Radford et al. \(2018\)](#) introduced GPT.
5. October 2018: [Devlin et al. \(2019\)](#) introduced BERT.

Model structure and linguistic structure



Attention

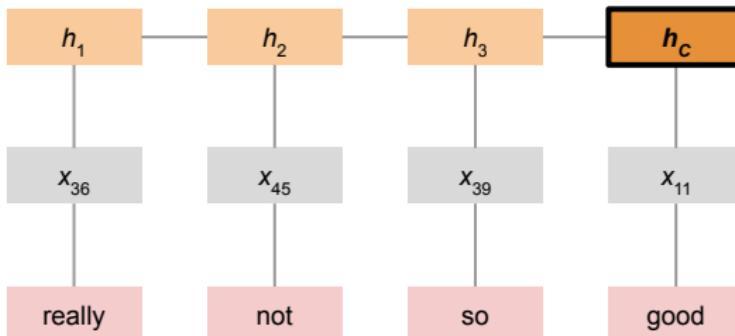
classifier $y = \text{softmax}(\tilde{h}W + b)$

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

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

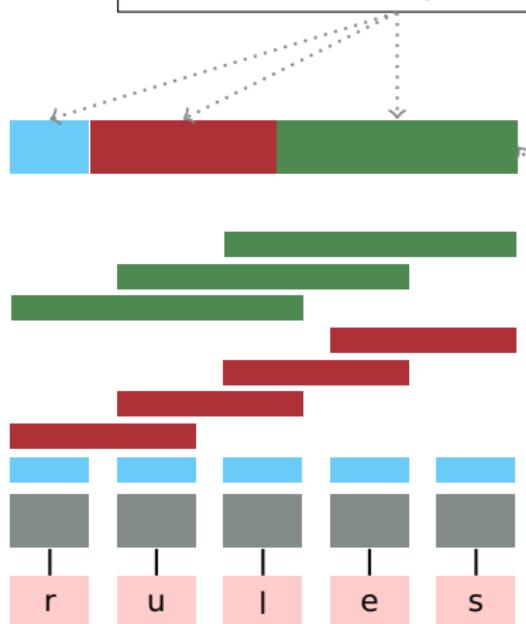
attention weights $\alpha = \text{softmax}(\tilde{\alpha})$

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



Subword modeling in ELMo

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:

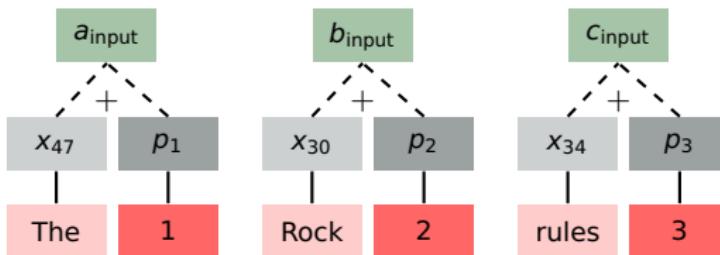
4	2	6	1
1	7	8	2
1	3	9	3
4	7	9	3

Guiding idea: Word pieces

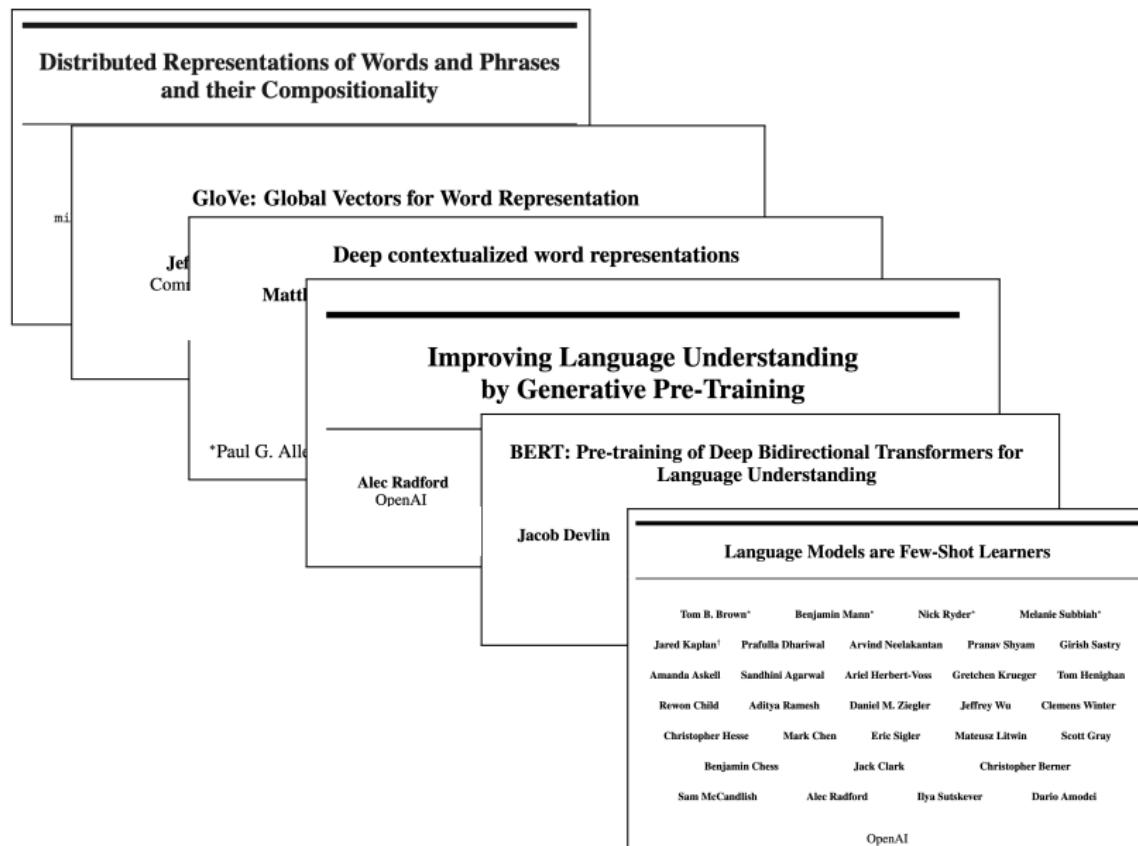
```
[1]: from transformers import BertTokenizer  
  
[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')  
  
[3]: tokenizer.tokenize("This isn't too surprising.")  
  
[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']  
  
[4]: tokenizer.tokenize("Encode me!")  
  
[4]: ['En', '#code', 'me', '!']  
  
[5]: tokenizer.tokenize("Snuffleupagus?")  
  
[5]: ['S', '#nu', '#ffle', '#up', '#agu', '#s', '?']  
  
[6]: tokenizer.vocab_size  
  
[6]: 28996
```



Guiding idea: Positional encoding



Guiding idea: Massive scale pretraining



Guiding idea: Fine-tuning

1. 2016–2018: Static word representation as RNN embeddings
2. 2018–

```
[31]: class HfBertClassifierModel(nn.Module):  
    def __init__(self, n_classes, weights_name='bert-base-cased'):  
        super().__init__()  
        self.n_classes = n_classes  
        self.weights_name = weights_name  
        self.bert = BertModel.from_pretrained(self.weights_name)  
        self.bert.train()  
        self.hidden_dim = self.bert.embeddings.word_embeddings.embedding_dim  
        # The only new parameters -- the classifier:  
        self.classifier_layer = nn.Linear(  
            self.hidden_dim, self.n_classes)  
  
    def forward(self, indices, mask):  
        reps = self.bert(  
            indices, attention_mask=mask)  
        return self.classifier_layer(reps.pooler_output)
```

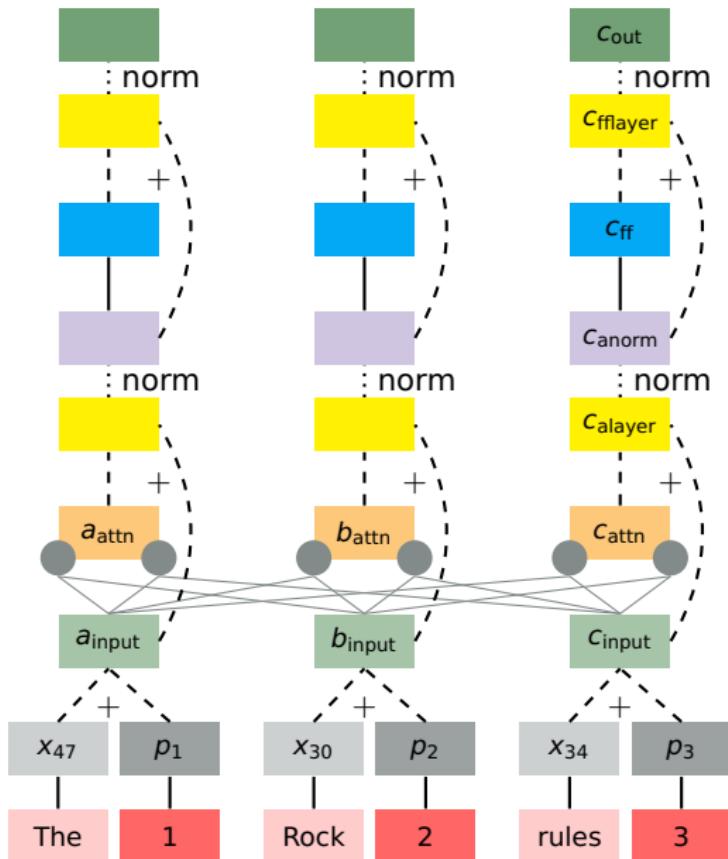
3. 2021–

```
openai api fine_tunes.create -t <TRAIN_FILE_ID_OR_PATH> -m <BASE_MODEL>
```



The Transformer

Core model structure



Computing the attention representations

Calculation as previously given

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

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

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

Matrix format

$$\mathbf{softmax} \left(\frac{c_{\text{input}} \begin{bmatrix} a_{\text{input}} \\ b_{\text{input}} \end{bmatrix}^T}{\sqrt{d_k}} \right) \begin{bmatrix} a_{\text{input}} \\ b_{\text{input}} \end{bmatrix}$$

Computing the attention representations

```
[1]: import numpy as np  
  
[2]: seq_length = 3  
d_k = 4  
  
[3]: inputs = np.random.uniform(size=(seq_length, d_k))  
inputs  
  
[3]: array([[0.31436922, 0.66969307, 0.270804 , 0.72023504],  
           [0.87180132, 0.27637445, 0.43091867, 0.34138704],  
           [0.20292054, 0.6345131 , 0.01058343, 0.22846636]])  
  
[4]: a_input = inputs[0]  
b_input = inputs[1]  
c_input = inputs[2]
```

Computing the attention representations

```
[5]: def softmax(X):
    z = np.exp(X)
    return (z / z.sum(axis=0)).T
```

```
[6]: c_alpha = softmax([
    (c_input.dot(a_input) / np.sqrt(d_k)),
    (c_input.dot(b_input) / np.sqrt(d_k))])
```

```
[7]: c_attn = sum([c_alpha[0]*a_input, c_alpha[1]*b_input])
c_attn
```

```
[7]: array([0.57768027, 0.48390338, 0.34643646, 0.54128076])
```

```
[8]: ab = inputs[:-1]
```

```
[9]: softmax(c_input.dot(ab.T) / np.sqrt(d_k)).dot(ab)
```

```
[9]: array([0.57768027, 0.48390338, 0.34643646, 0.54128076])
```

```
[10]: # If we allow every input to attend to itself:
softmax(inputs.dot(inputs.T) / np.sqrt(d_k)).dot(inputs)
```

```
[10]: array([[0.4614388 , 0.53204444, 0.2451212 , 0.45136127],
           [0.50173123, 0.50618272, 0.26184404, 0.43678288],
           [0.45493467, 0.5332328 , 0.23643403, 0.4388242 ]])
```

Multi-headed attention

$$c_{\text{attn}}^3 = \text{sum}([\alpha_1(a_{\text{input}} W_3^V), \alpha_2(b_{\text{input}} W_3^V)])$$

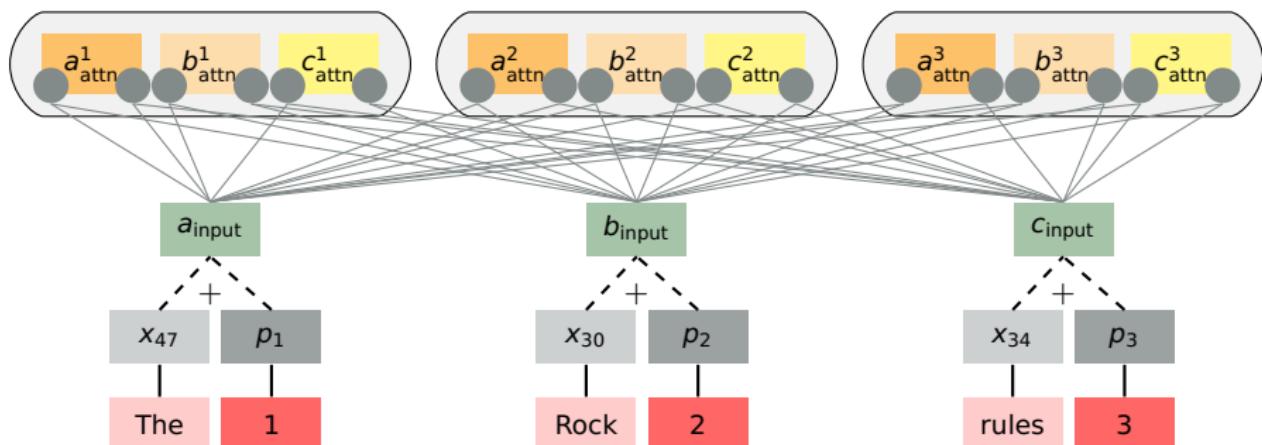
$$\alpha = \text{softmax}(\tilde{\alpha})$$

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

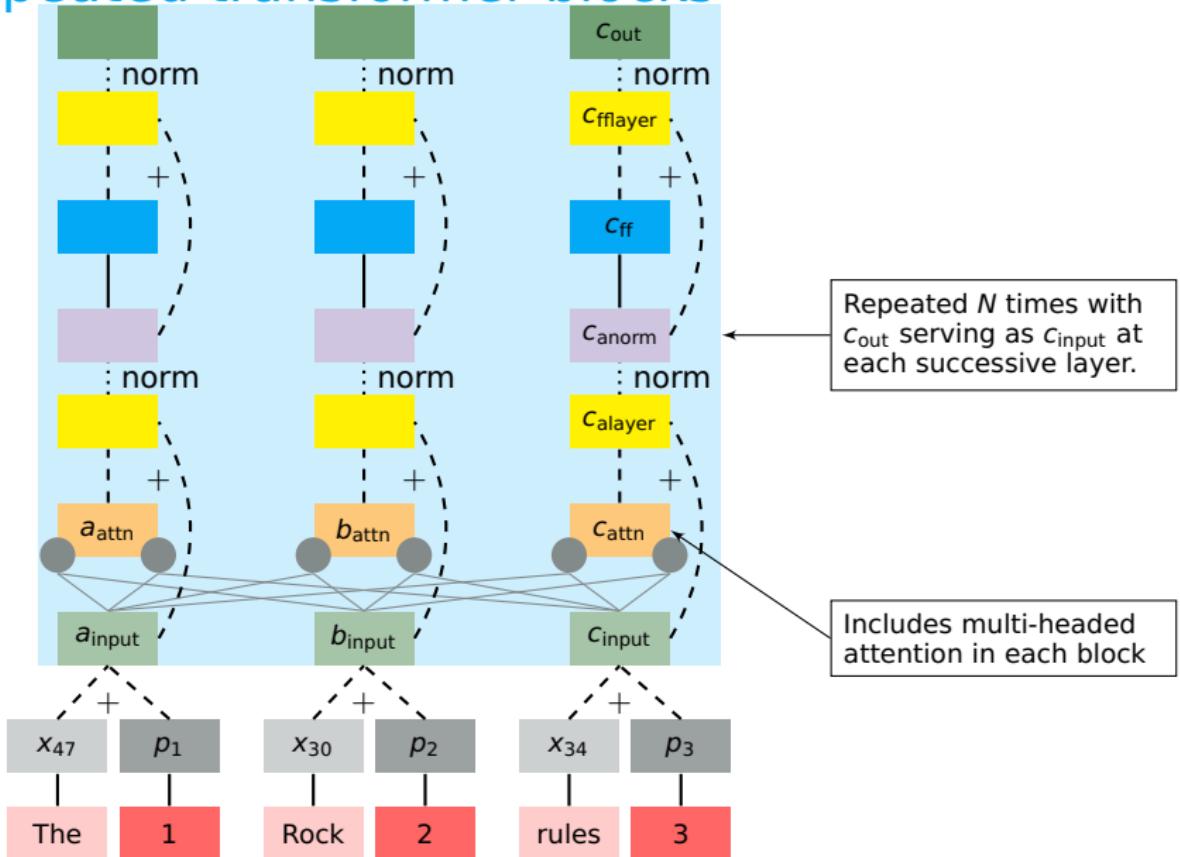
a_{attn}^1 a_{attn}^2 a_{attn}^3

b_{attn}^1 b_{attn}^2 b_{attn}^3

c_{attn}^1 c_{attn}^2 c_{attn}^3



Repeated transformer blocks



The architecture diagram

A view from PyTorch

```
from transformers import AutoModel

model = AutoModel.from_pretrained('bert-base-cased')

model

BertModel(
    (embeddings): BertEmbeddings(
        (word_embeddings): Embedding(28996, 768, padding_idx=0)
        (position_embeddings): Embedding(512, 768)
        (token_type_embeddings): Embedding(2, 768)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
        (layer): ModuleList(
            (0): BertLayer(
                (attention): BertAttention(
                    (self): BertSelfAttention(
                        (query): Linear(in_features=768, out_features=768, bias=True)
                        (key): Linear(in_features=768, out_features=768, bias=True)
                        (value): Linear(in_features=768, out_features=768, bias=True)
                        (dropout): Dropout(p=0.1, inplace=False)
                    )
                    (output): BertSelfOutput(
                        (dense): Linear(in_features=768, out_features=768, bias=True)
                        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                        (dropout): Dropout(p=0.1, inplace=False)
                    )
                )
                (intermediate): BertIntermediate(
                    (dense): Linear(in_features=768, out_features=3072, bias=True)
                    (intermediate_act_fn): GELUActivation()
                )
                (output): BertOutput(
                    (dense): Linear(in_features=3072, out_features=768, bias=True)
                    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                    (dropout): Dropout(p=0.1, inplace=False)
                )
            )
        )
    (1): BertLayer(
```

Positional encoding

The role of positional encoding

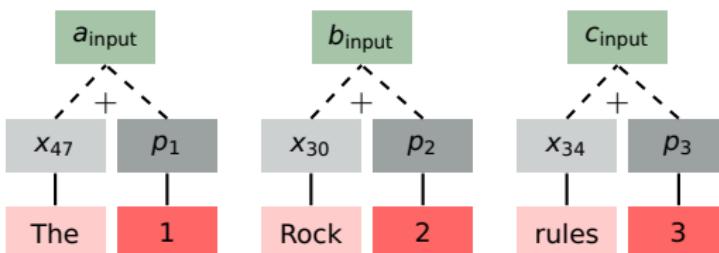
- The Transformer has a very limited capacity to keep track of word order:
 - ▶ the attention connections are not directional, and
 - ▶ there are no other interactions between the columns.
- Positional encodings ensure differences between A B C and C B A.
- Positional encodings have also been used to keep track of hierarchical notions of position like premise/hypothesis in Natural Language Inference.

Evaluating positional encoding schemes

1. Does the set of positions need to be decided ahead of time?
2. Does the scheme hinder generalization to new positions?

Models will tend to impose a max length on the sequences they can process for reasons relating to their learned weights. We will ask whether different positional encoding schemes are imposing anything about length generalization *separate from this*.

Absolute positional encoding

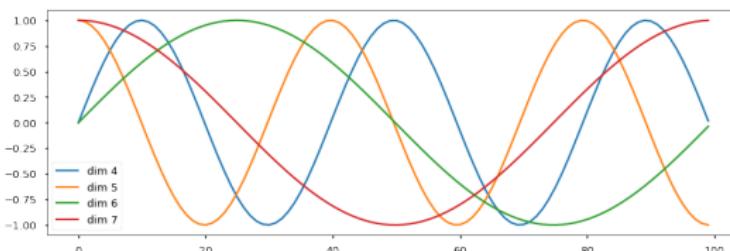


Limitations

1. Set of position needs to be decided ahead of time.
2. May hinder generalization to new positions, even for familiar phenomena:



Frequency-based positional encoding



```
import numpy as np
def posenc(pos, d_model):
    div_term = np.exp(
        np.arange(0, d_model, 2) * -(np.log(10000.0) / d_model))
    rep = np.zeros(d_model)
    rep[0::2] = np.sin(pos * div_term)
    rep[1::2] = np.cos(pos * div_term)
    return rep
```

Limitations

1. Set of position needs to be decided ahead of time.
2. May hinder generalization to new positions, even for familiar phenomena.

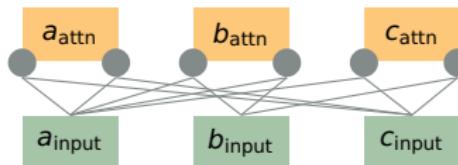
'The Annotated Transformer'

Relative positional encoding: Basics

Previously

$$c_{\text{attn}} = \mathbf{sum}([\alpha_1^c a_{\text{input}}, \alpha_2^c b_{\text{input}}])$$

$$\alpha^c = \mathbf{softmax}\left(\left[\frac{c_{\text{input}}^T a_{\text{input}}}{\sqrt{d_k}}, \frac{c_{\text{input}}^T b_{\text{input}}}{\sqrt{d_k}}\right]\right)$$



Relative encoding

$$c_{\text{attn}} = \mathbf{sum}([\alpha_1^c a_{\text{input}} + a_{3,1}^V, \alpha_2^c b_{\text{input}} + a_{3,2}^V])$$

$$\alpha^c = \mathbf{softmax}\left(\left[\frac{c_{\text{input}}^T (a_{\text{input}} + a_{3,1}^K)}{\sqrt{d_k}} \frac{c_{\text{input}}^T (b_{\text{input}} + a_{3,2}^K)}{\sqrt{d_k}}\right]\right)$$

Shaw et al. 2018

Relative positional encoding: Windows

$$c_{\text{attn}} = \mathbf{sum}([a_1^c a_{\text{input}} + a_{3,1}^V, a_2^c b_{\text{input}} + a_{3,2}^V])$$

$$\alpha^c = \mathbf{softmax} \left(\begin{bmatrix} \frac{c_{\text{input}}^T (a_{\text{input}} + a_{3,1}^K)}{\sqrt{d_k}} & \frac{c_{\text{input}}^T (b_{\text{input}} + a_{3,2}^K)}{\sqrt{d_k}} \end{bmatrix} \right)$$

With window size $d = 2$:

1	2	3	4	5	6	7
a_{input}	b_{input}	c_{input}	d_{input}	e_{input}	f_{input}	g_{input}
$a_{4,1}^K = w_{-2}^K$	$a_{4,2}^K = w_{-2}^K$	$a_{4,3}^K = w_{-1}^K$	$a_{4,4}^K = w_0^K$	$a_{4,5}^K = w_1^K$	$a_{4,6}^K = w_2^K$	$a_{4,7}^K = w_2^K$
$a_{3,1}^K = w_{-2}^K$	$a_{3,2}^K = w_{-1}^K$	$a_{3,3}^K = w_0^K$				

Same logic for the representations a_{ij}^V .

Shaw et al. 2018

Relative positional encoding: Full definition

With learned attention parameters:

$$\text{attn}_i = \sum_{j=1}^n \alpha_{ij} (x_j W^V + a_{jj}^V)$$

$$\alpha_{ij} = \text{softmax} \left(\frac{(x_i W^Q)^T (x_j W^K + a_{ij}^K)}{\sqrt{d_k}} \right)$$

Limitations

- ~~Set of position needs to be decided ahead of time.~~
- ~~May hinder generalization to new positions, even for familiar phenomena.~~



Shaw et al. 2018

GPT

GPT: Autoregressive loss function

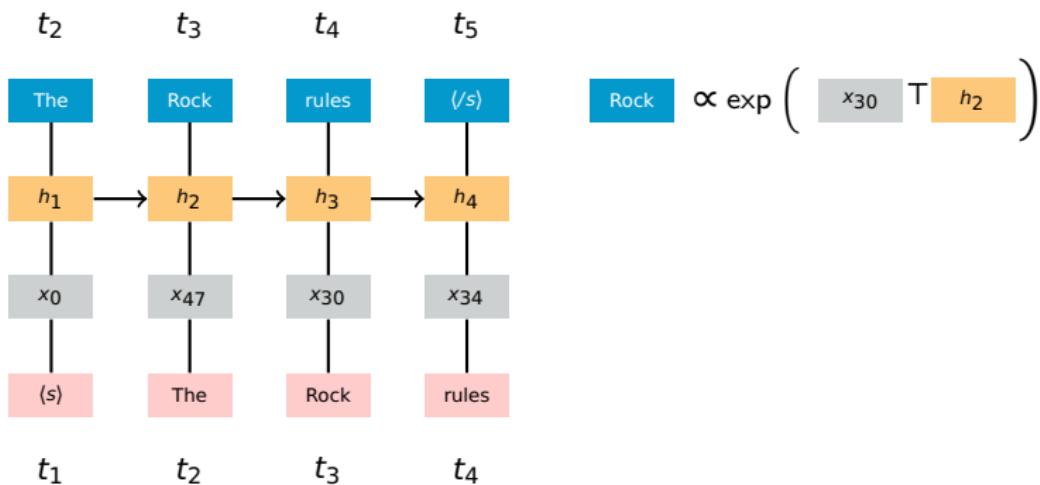
For vocabulary \mathcal{V} , sequence $\mathbf{x} = [x_1, \dots, x_T]$, and word-level embedding e :

$$\max_{\theta} \sum_{t=1}^T \log \frac{\exp(e(x_t)^T h_{\theta}(\mathbf{x}_{1:t-1}))}{\sum_{x' \in \mathcal{V}} \exp(e(x')^T h_{\theta}(\mathbf{x}_{1:t-1}))}$$

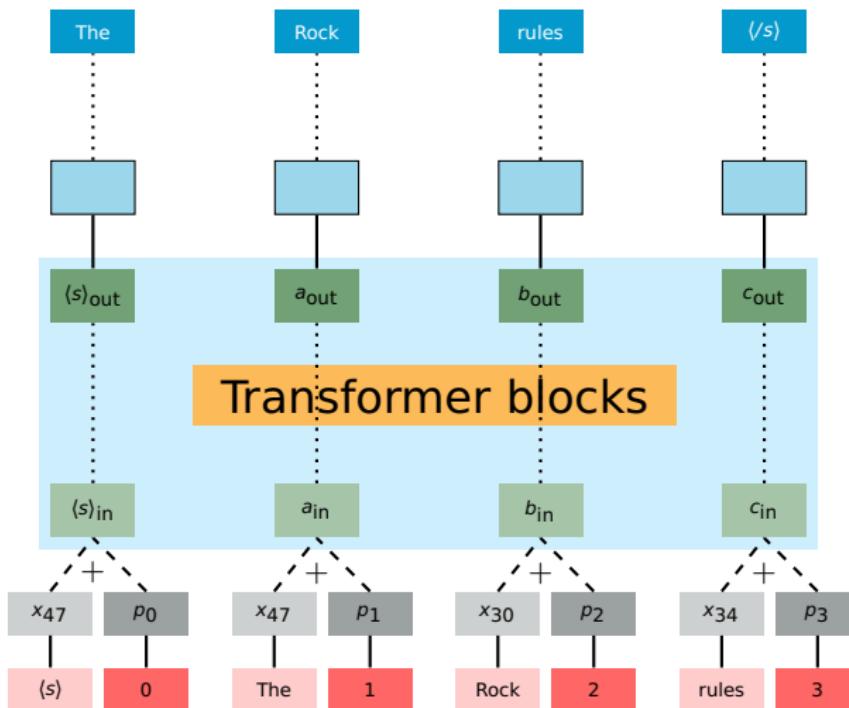
for model parameters h_{θ} .

Conditional language modeling

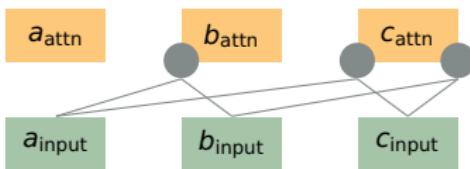
$$\mathbf{x} = [\langle s \rangle, \text{The}, \text{Rock}, \text{rules}, \langle /s \rangle]$$



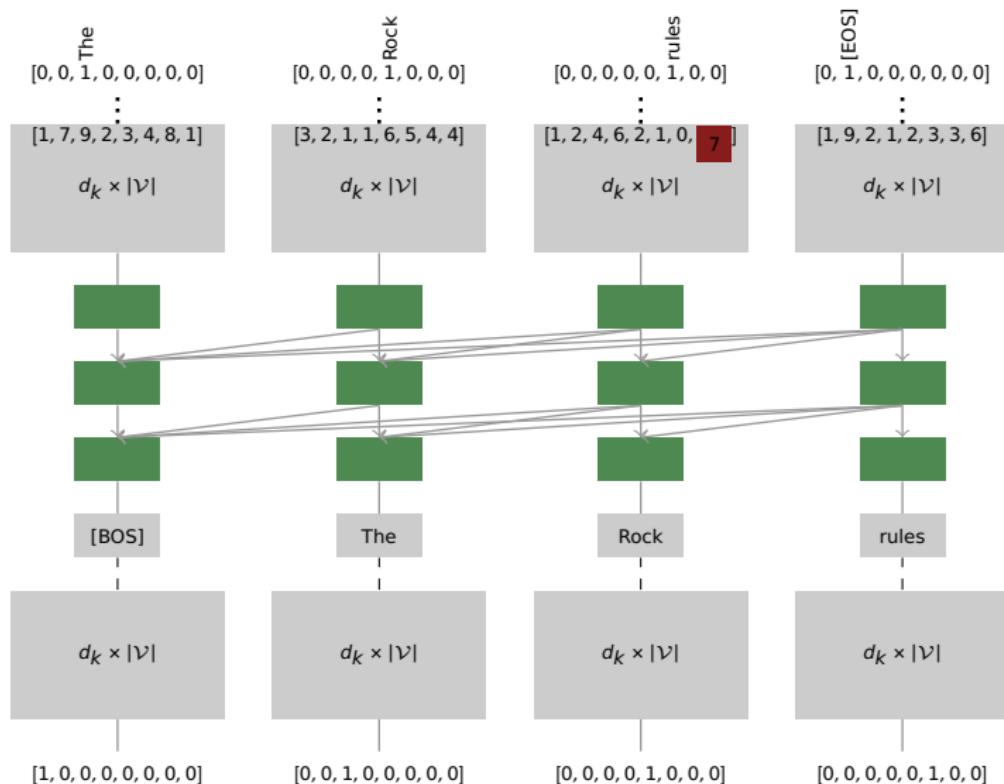
GPT



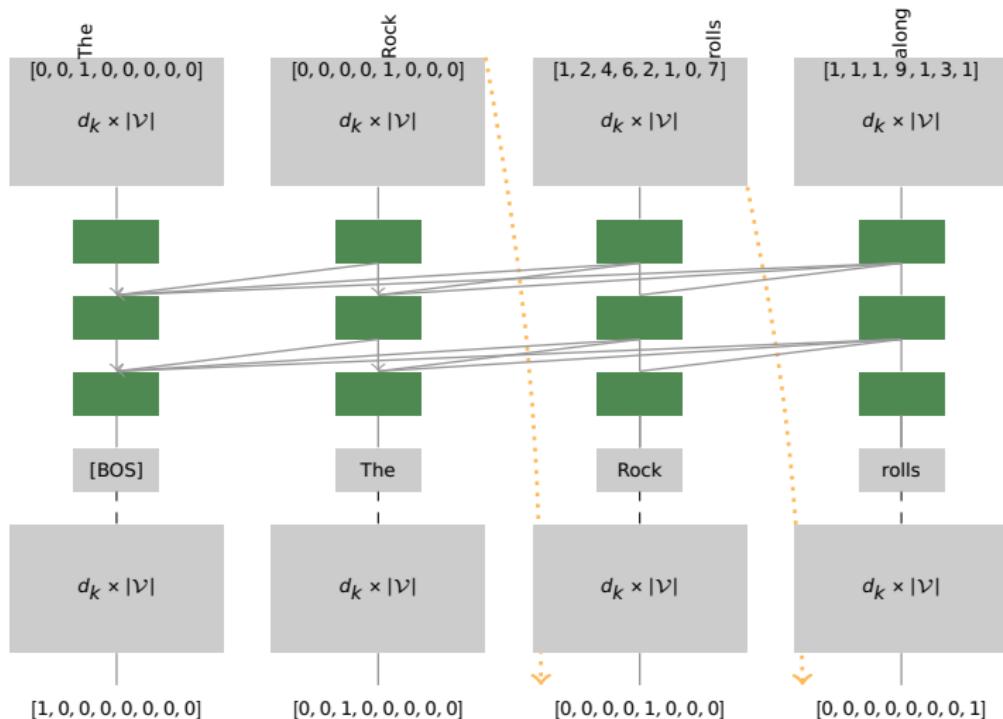
GPT: Attention masking



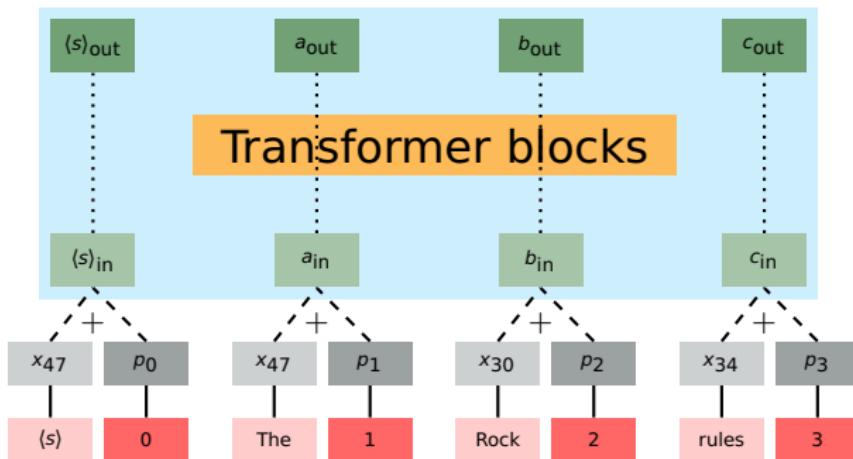
GPT: Training with teacher forcing



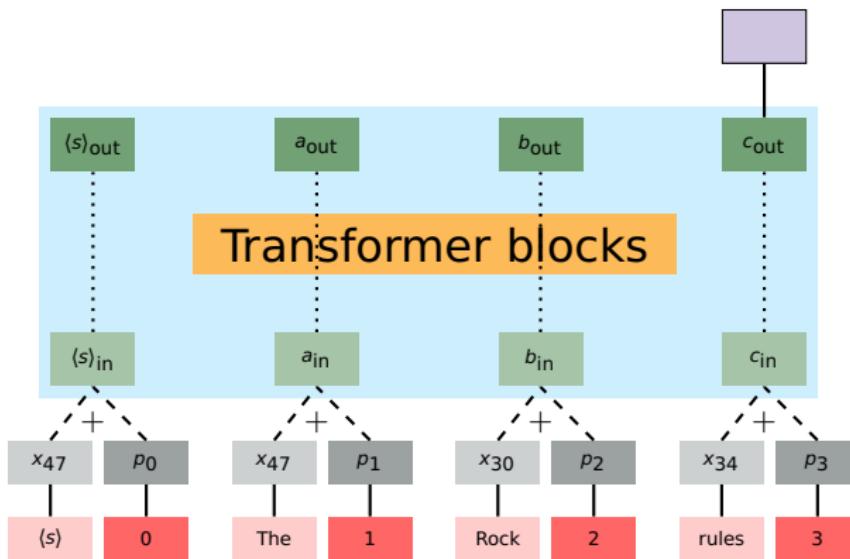
Generation



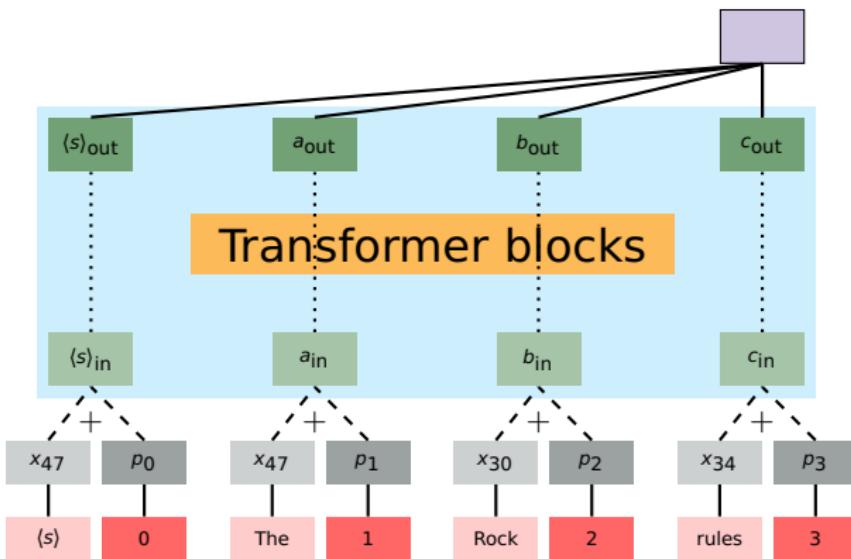
GPT: Fine-tuning



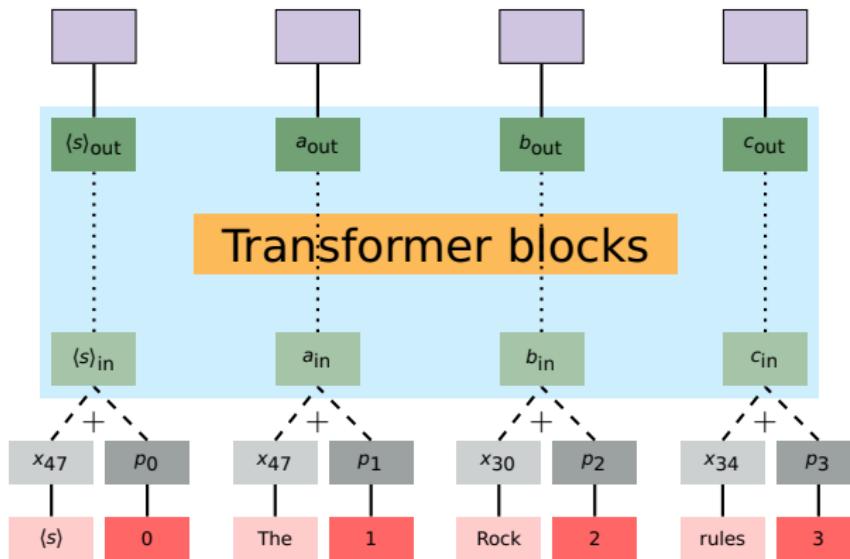
GPT: Fine-tuning



GPT: Fine-tuning



GPT: Fine-tuning



GPT: Scaling up from OpenAI

	Layers	d_k	d_{ff}	Parameters
GPT (Radford et al. 2018)	12	768	3,072	117M
GPT-2 (Radford et al. 2019)	48	1,600	1,600	1,542M
GPT-3 (Brown et al. 2020)	96	12,288	?	175,000M

The GPT-3 paper also reports on models ranging in size from 125M to 13B (Table 2.1).

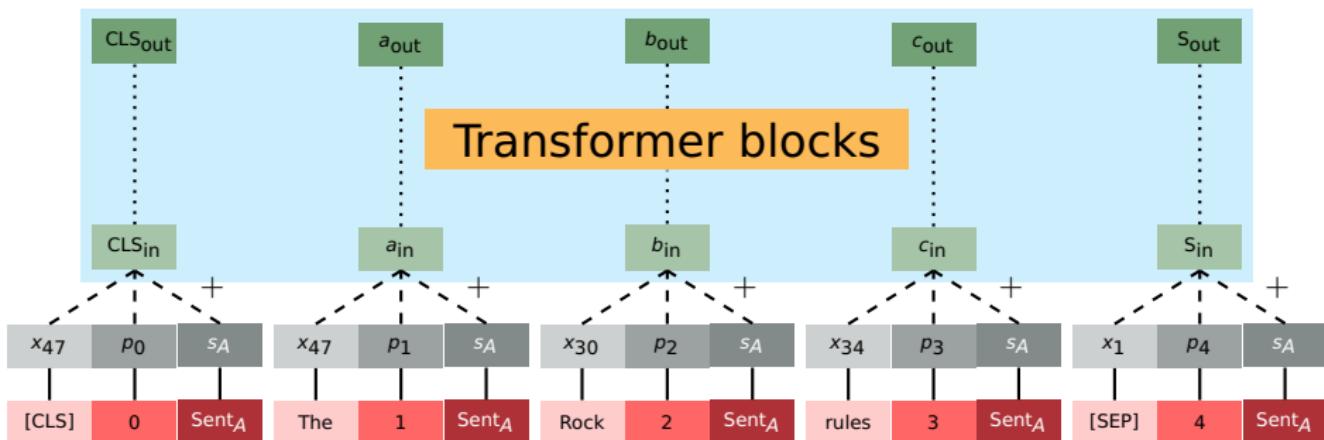
GPT: Scaling up truly open models

	Layers	d_k	d_{ff}	Parameters
GPT-Neo (Eleuther)	24	2,048	2,048	2,700M
GPT-J (Eleuther)	28	4,096	16,384	6,000M
GPT-NeoX (Eleuther)	44	6,144	6,144	20,000M
OPT-66B (Zhang et al. 2022)	64	9,216	9,216	66,000M
BLOOM (Scao et al. 2022)	70	14,336	14,336	176,247M

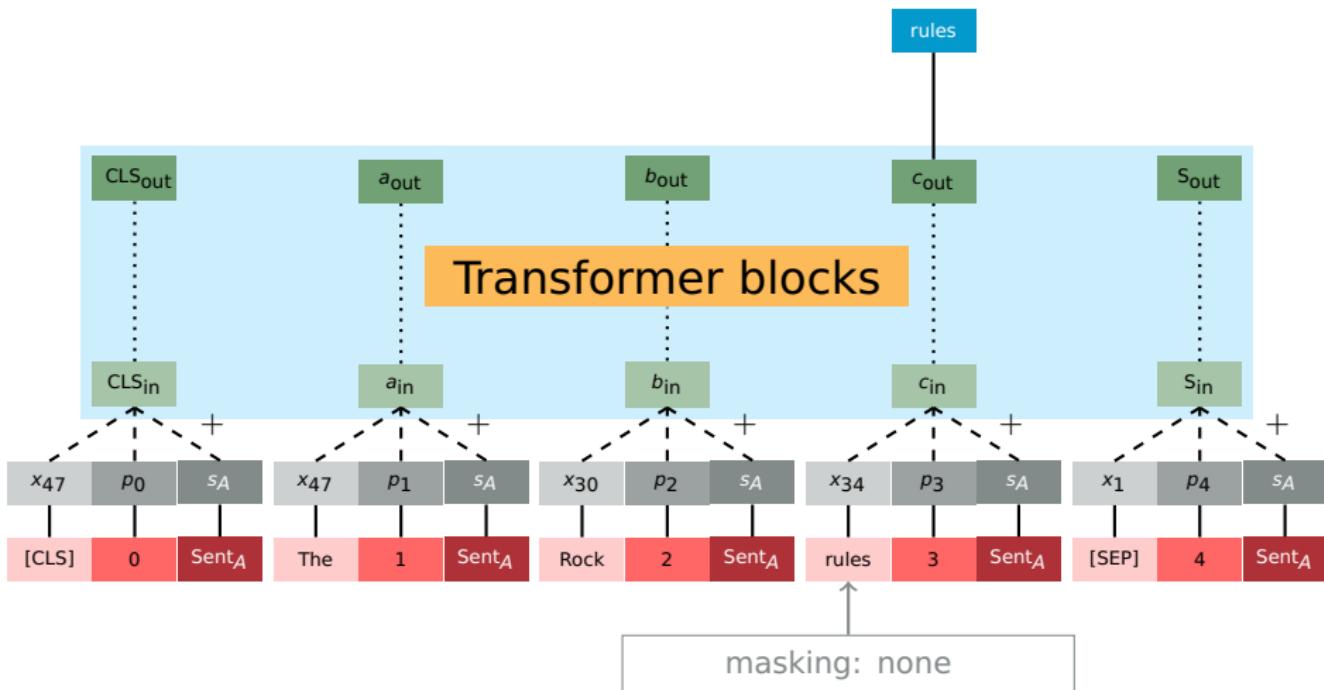
This table will be out of date by the time anyone reads it, if not before!

BERT

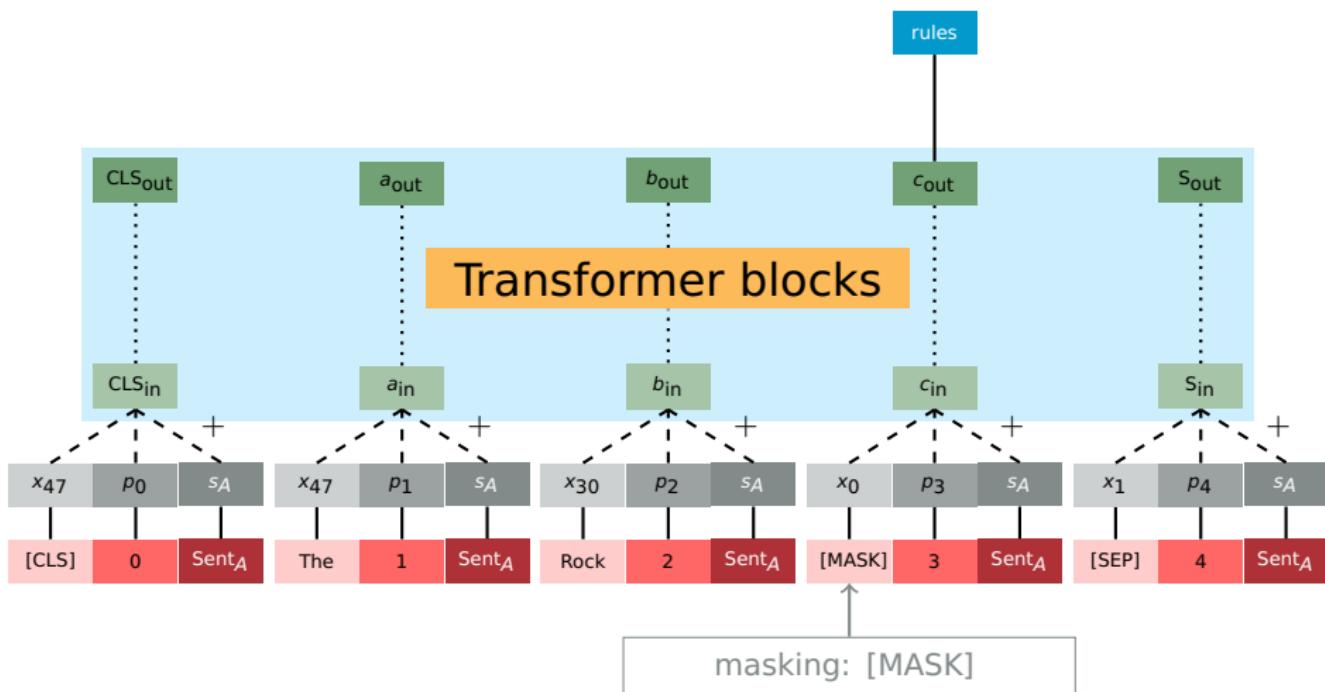
BERT: Core model structure



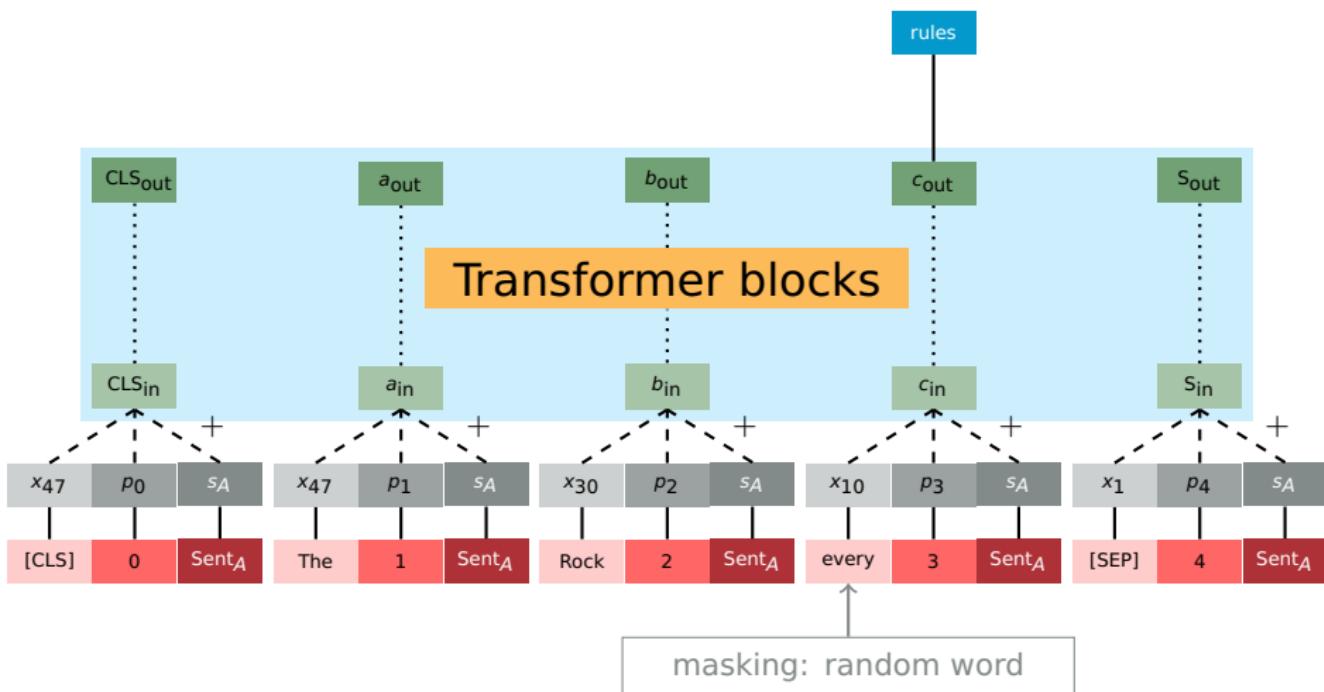
Masked Language Modeling (MLM)



Masked Language Modeling (MLM)



Masked Language Modeling (MLM)



BERT: MLM loss function

For Transformer parameters H_θ and sequence $\mathbf{x} = [x_1, \dots, x_T]$ with masked version $\hat{\mathbf{x}}$:

$$\max_{\theta} \sum_{t=1}^T m_t \log \frac{\exp(e(x_t)^\top H_\theta(\hat{\mathbf{x}})_t)}{\sum_{x' \in \mathcal{V}} \exp(e(x')^\top H_\theta(\hat{\mathbf{x}})_t)}$$

where \mathcal{V} is the vocabulary, x_t is the actual token at step t , $m_t = 1$ if token t was masked, else 0, and $e(x)$ is the embedding for x .

Binary next sentence prediction pretraining

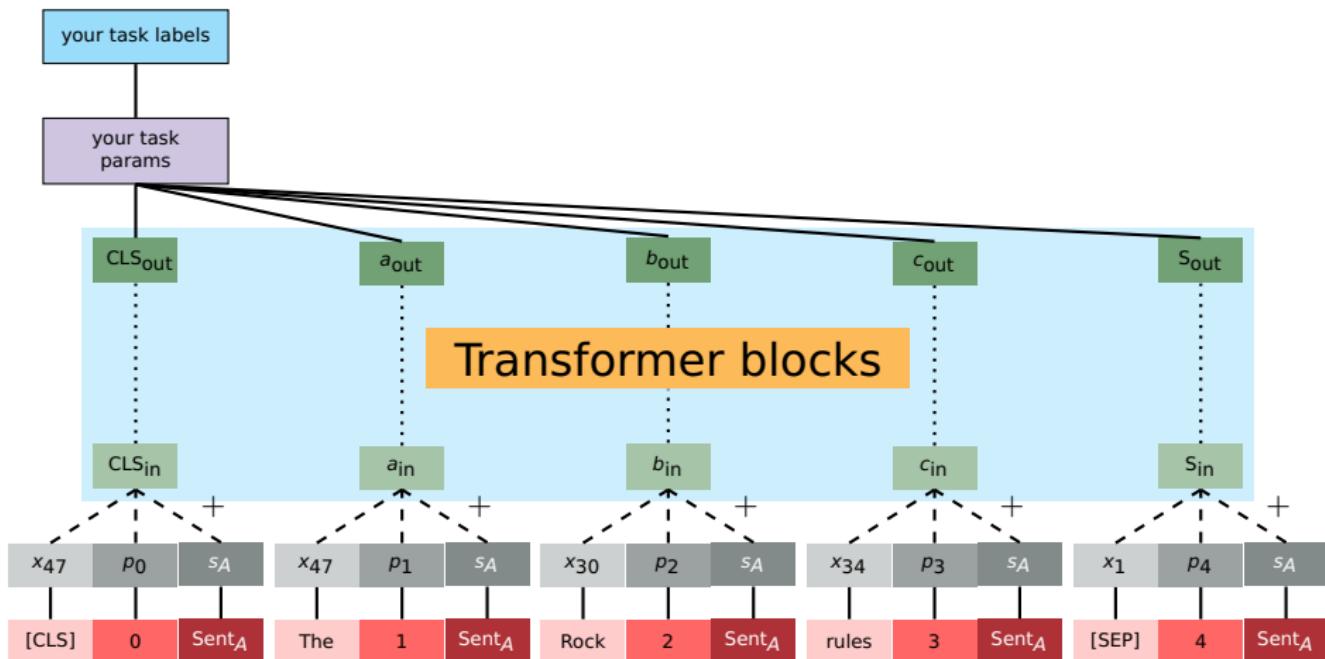
Positive: Actual sentence sequences

- [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

BERT: Transfer learning and fine-tuning



Tokenization and the BERT embedding space

```
[1]: from transformers import BertTokenizer  
  
[2]: tokenizer = BertTokenizer.from_pretrained('bert-base-cased')  
  
[3]: tokenizer.tokenize("This isn't too surprising. ")  
  
[3]: ['This', 'isn', "'", 't', 'too', 'surprising', '.']  
  
[4]: tokenizer.tokenize("Encode me!")  
  
[4]: ['En', '#code', 'me', '!']  
  
[5]: tokenizer.tokenize("Snuffleupagus?")  
  
[5]: ['S', '#nu', '#ffle', '#up', '#agu', '#s', '?']  
  
[6]: tokenizer.vocab_size  
  
[6]: 28996
```

BERT: Core model releases

	Layers	d_k	d_{ff}	Parameters
BERT-tiny	2	128	512	4M
BERT-mini	4	246	1,024	11M
BERT-small	4	512	2,048	29M
BERT-medium	8	512	2,048	41M
BERT-base	12	768	3,072	110M
BERT-large	24	1,024	4,096	340M

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

Many new releases at the [project site](#) and on [Hugging Face](#), including BERT variants for different languages.

BERT: Known limitations

1. Devlin et al. (2019): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): “The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning.”
3. Devlin et al. (2019): “The second downside of using an MLM is that only 15% of tokens are predicted in each batch”
4. Yang et al. (2019): “BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language”

RoBERTa

Addressing the known limitations with BERT

1. Devlin et al. (2019): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): “The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning.”
3. Devlin et al. (2019): “The second downside of using an MLM is that only 15% of tokens are predicted in each batch”
4. Yang et al. (2019): “BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language”

Robustly optimized BERT approach

BERT	RoBERTa
Static masking/substitution	Dynamic masking/substitution
Inputs are two concatenated document segments	Inputs are sentence sequences that may span document boundaries
Next Sentence Prediction (NSP)	No NSP
Training batches of 256 examples	Training batches of 2,000 examples
Word-piece tokenization	Character-level byte-pair encoding
Pretraining on BooksCorpus and English Wikipedia	Pretraining on BooksCorpus, Wikipedia, CC-News, OpenWebText, Stories
Train for 1M steps	Train for up to 500K steps
Train on short sequences first	Train only on full-length sequences

Additional differences in the optimizer and data presentation (sec 3.1).

RoBERTa results informing final system design

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
<i>Our reimplementation:</i>			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for BERT_{BASE}. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from [Yang et al. \(2019\)](#).

RoBERTa results informing final system design

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
<i>Our reimplementations (with NSP loss):</i>				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
<i>Our reimplementations (without NSP loss):</i>				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
BERT _{BASE}	88.5/76.3	84.3	92.8	64.3
XLNet _{BASE} (K = 7)	-/81.3	85.8	92.7	66.1
XLNet _{BASE} (K = 6)	-/81.0	85.6	93.4	66.7

RoBERTa choice
for efficient
batching, and
comparisons with
related work.

Table 2: Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. We report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over five random initializations (seeds). Results for BERT_{BASE} and XLNet_{BASE} are from Yang et al. (2019).

RoBERTa results informing final system design

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	3.68	85.2	92.9
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

RoBERTa results informing final system design

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4
BERT_{LARGE}						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
XLNet_{LARGE}						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of BERT_{LARGE}. Results for BERT_{LARGE} and XLNet_{LARGE} are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

RoBERTa: Core model releases

	Layers	d_k	d_{ff}	Parameters
RobERTa-base	12	768	3,072	125M
RobERTa-large	24	1,024	4,096	355M

Related work

A Primer in BERTology: What we know about how BERT works

Anna Rogers, Olga Kovaleva, Anna Rumshisky

Department of Computer Science, University of Massachusetts Lowell
Lowell, MA 01854

ELECTRA

Addressing the known limitations with BERT

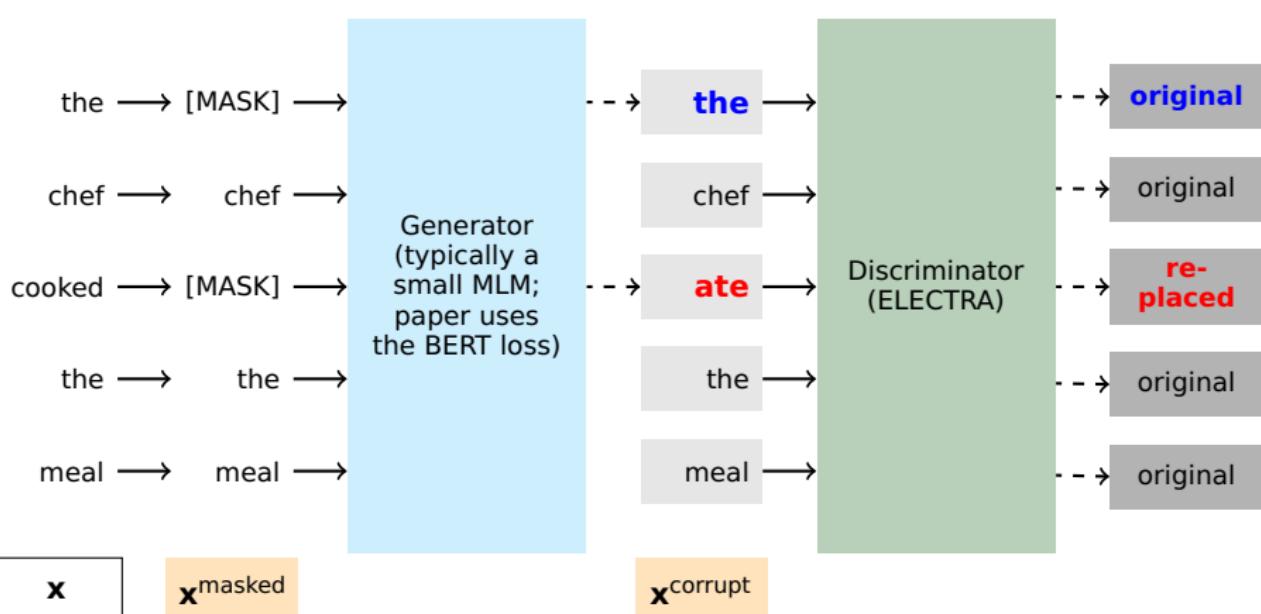
1. Devlin et al. (2019): admirably detailed but still partial ablation studies and optimization studies.
2. Devlin et al. (2019): “The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning.”
3. Devlin et al. (2019): “The second downside of using an MLM is that only 15% of tokens are predicted in each batch”
4. Yang et al. (2019): “BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language”

Core model structure (Clark et al. 2019)

Random sample of
~15% of tokens masked

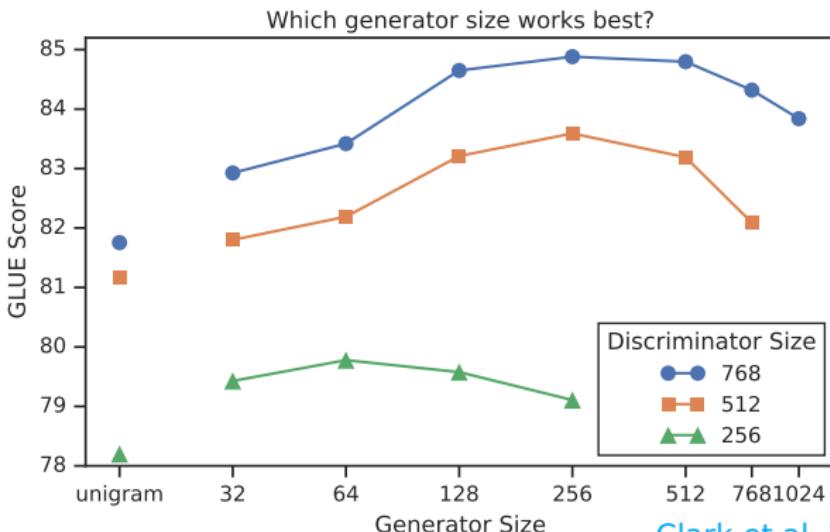
Masked tokens replaced
proportional to Generator
probabilities

Loss:
Generator +
 λ ELECTRA

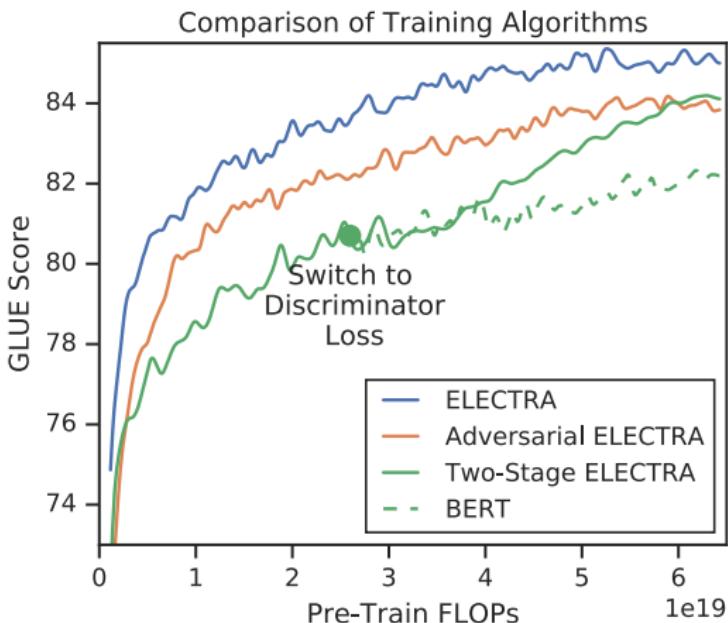


Generator/Discriminator relationships

Where Generator and Discriminator are the same size, they can share Transformer parameters, and more sharing is better. However, the best results come from having a Generator that is small compared to the Discriminator:



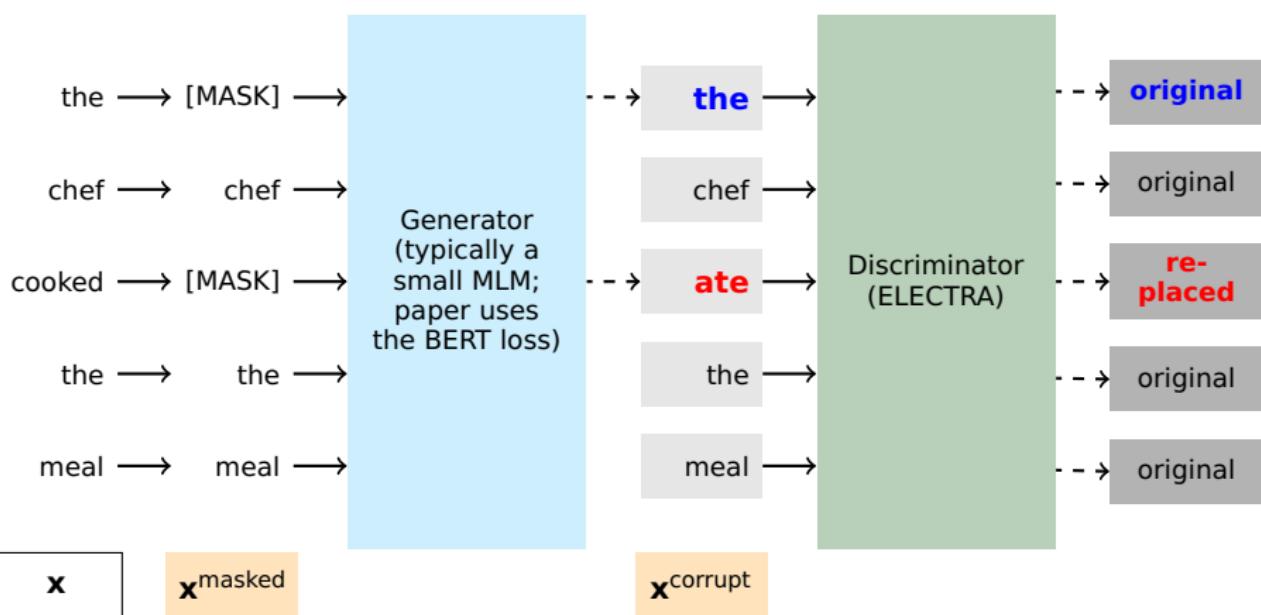
Efficiency



Clark et al. 2019, Figure 3

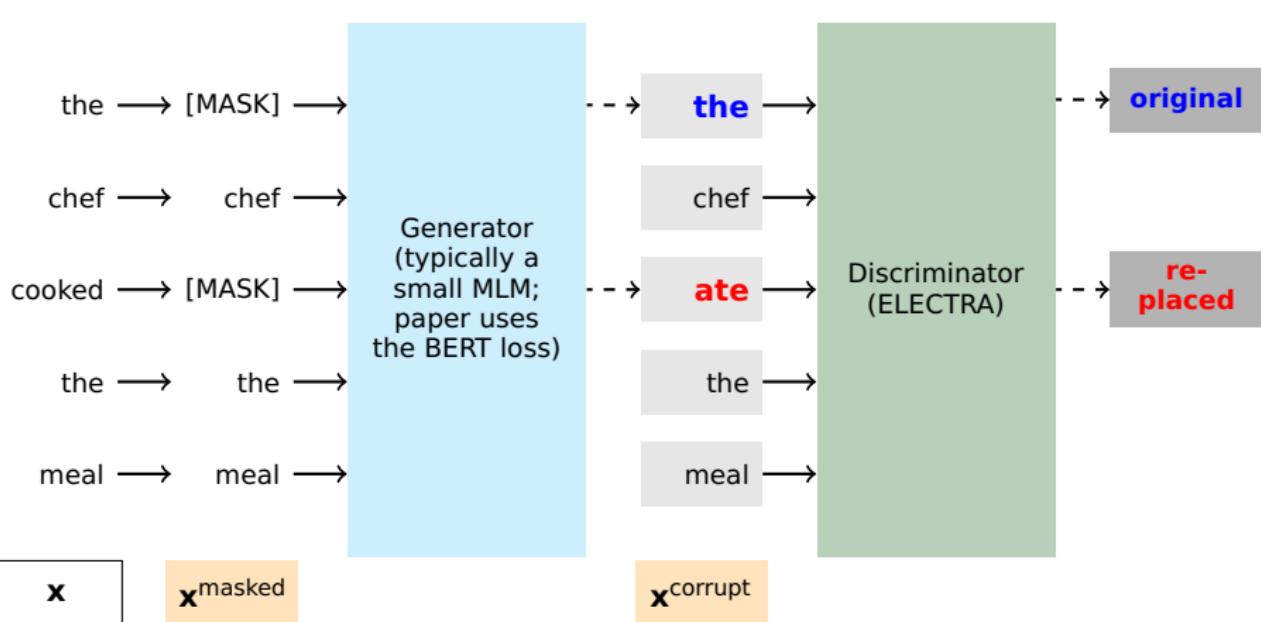
ELECTRA efficiency analyses

Full ELECTRA



ELECTRA efficiency analyses

ELECTRA 15%



ELECTRA efficiency analyses

Replace MLM

the → dog → - - → the

chef → chef → - - → chef

cooked → run → - - → cooked

the → the → - - → the

meal → meal → - - → meal

Generator
(typically a
small MLM;
paper uses
the BERT loss)

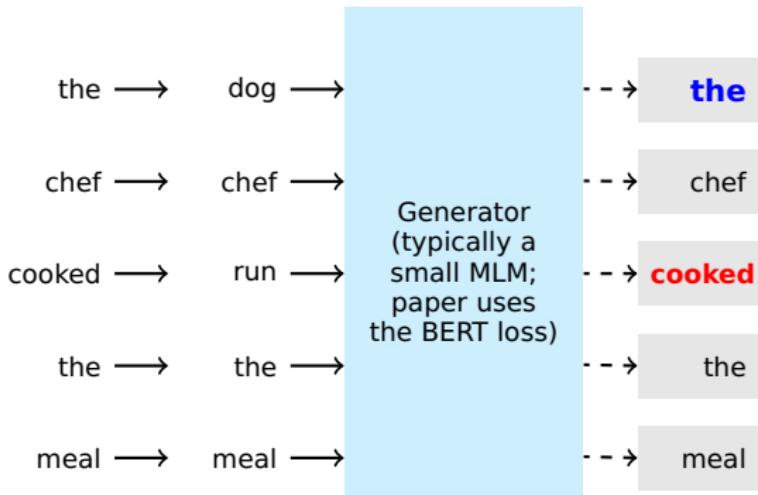
x

x^{replaced}

x^{corrupt}

ELECTRA efficiency analyses

All-tokens MLM



x

x^{replaced}

x^{corrupt}

ELECTRA efficiency analyses

Model	GLUE score
ELECTRA	85.0
All-tokens MLM	84.3
Replace MLM	82.4
ELECTRA 15%	82.4
BERT	82.2

ELECTRA model releases

Available from the [project site](#):

Model	Layers	Hidden Size	Params	GLUE test
Small	12	256	14M	77.4
Base	12	768	110M	82.7
Large	24	1024	335M	85.2

‘Small’ is the model designed to be “quickly trained on a single GPU”.

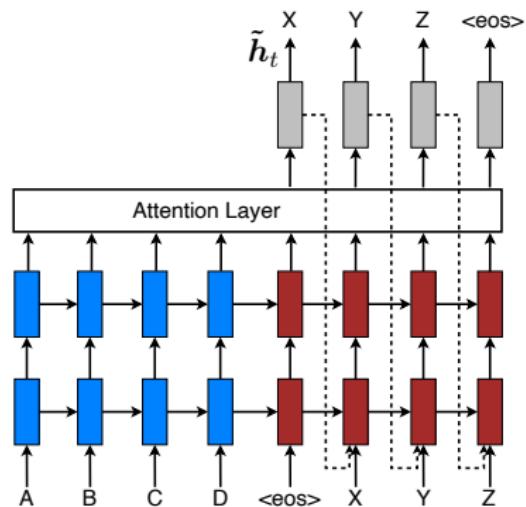
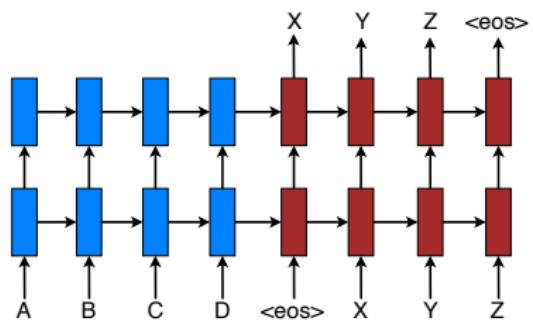
seq2seq architectures

Some tasks with natural seq2seq structure

1. Machine translation (language to language)
2. Summarization (text to shorter text)
3. Free-form question answering (question to answer)
4. Dialogue (utterance to utterance)
5. Semantic parsing (sentence to logical form)
6. Code generation (sentence to program)
7. ...

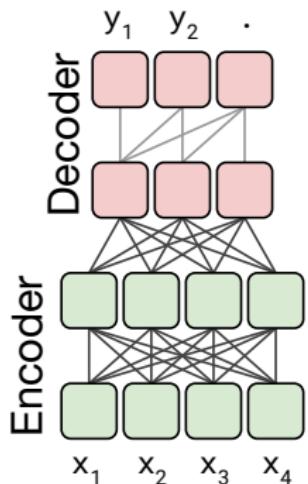
More general class: encoder-decoder (agnostic about whether the encoding and decoding involve sequences).

From the RNN era

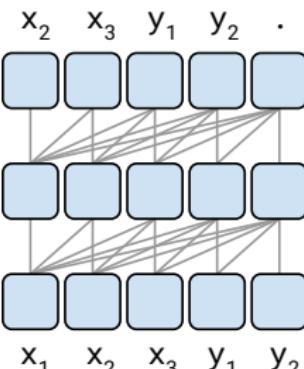


From Luong et al. 2015, Figures 1 and 4

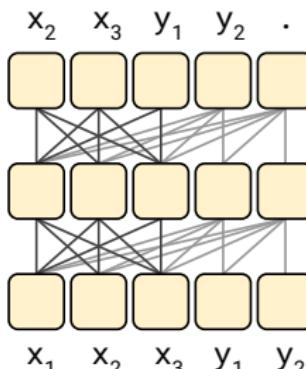
Transformer-based options



Language model



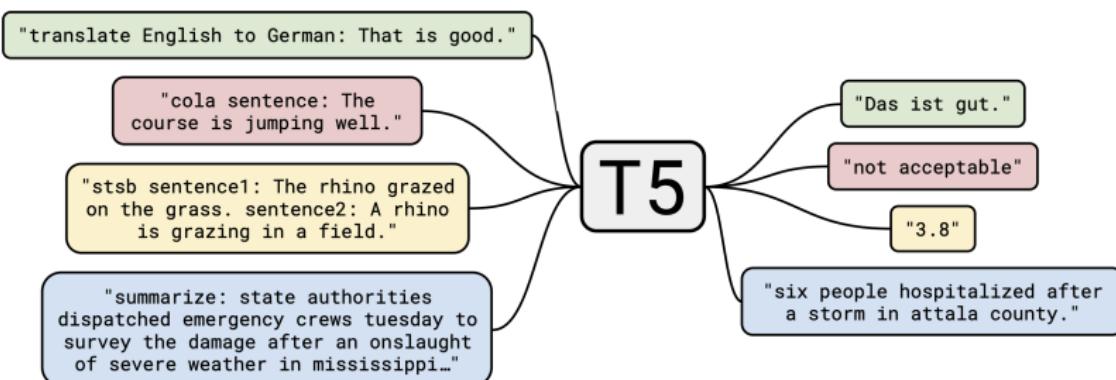
Prefix LM



From Raffel et al. 2019, Figure 4

T5

Encoder–Decoder variant with extensive multi-task supervised and unsupervised training. Input task-prefixes guide model behavior. **Lots of pretrained versions.**



From Raffel et al. 2019, Figure 1

T5 model releases

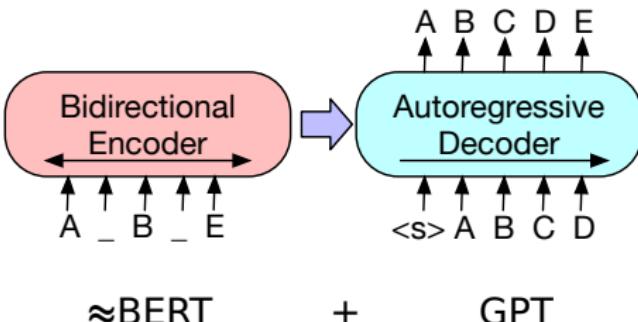
See https://huggingface.co/docs/transformers/model_doc/t5

	Layers	d_k	d_{ff}	Parameters
T5-small	6	512	2,048	60M
T5-base	12	768	3,072	220M
T5-large	24	1,024	4,096	770M
T5-3B	24	1,024	16,384	3,000M
T5-11B	24	1,024	65,536	11,000M

Also: FLAN-T5 models (instruction-tuned) of [Chung et al. 2022](#)

BART

- Text infilling
 - Sentence shuffling
 - Token masking
 - Token deletion
 - Document rotation



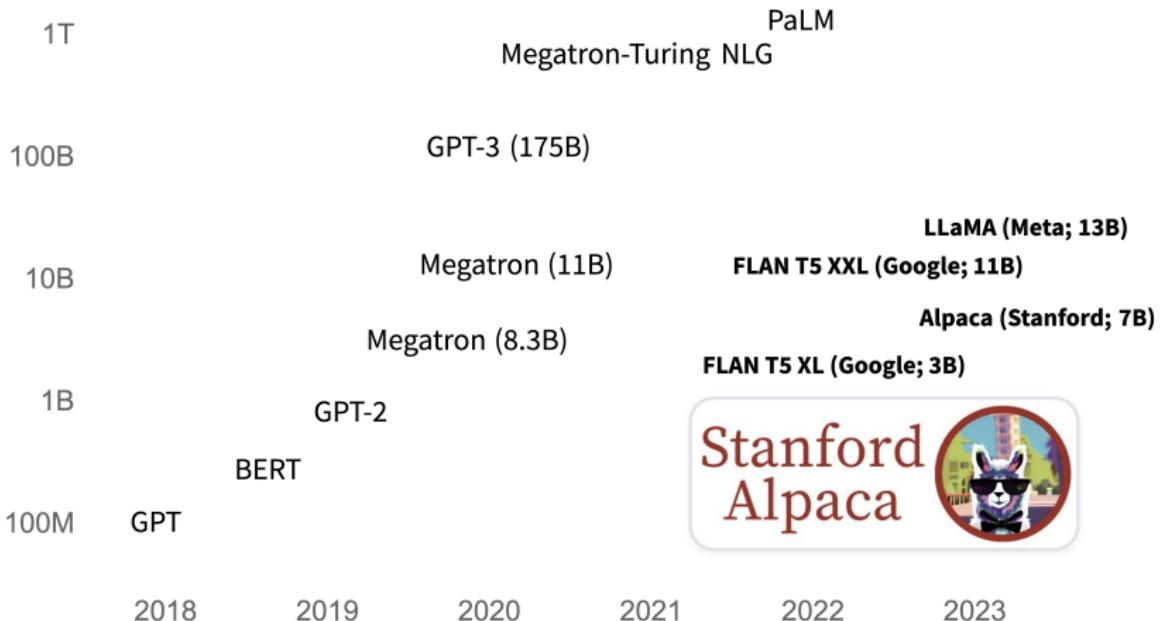
Fine-tuning

- **Classification**: Uncorrupted copies of the input are fed to the encoder and the decoder. The final decoder state is usually the basis for classification.
 - **seq2seq**: Standard encoder-decoder usage.

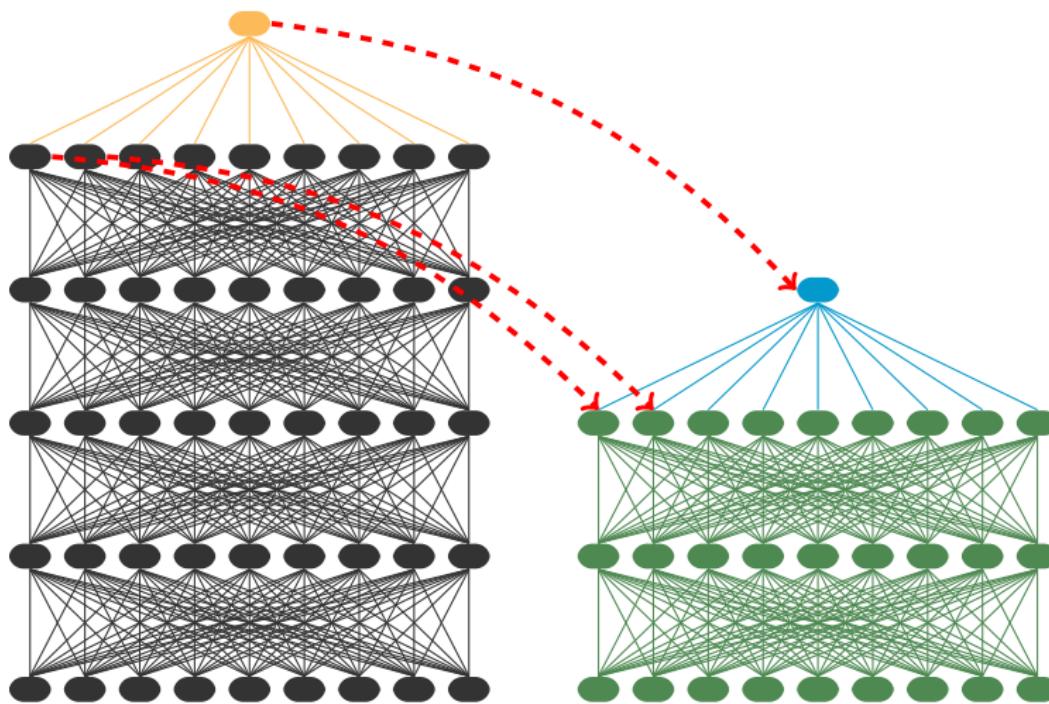
Lewis et al. 2019

Distillation

Trends in model size



Teachers and students



Distillation objectives

From least to most heavy duty; weighted averages of these are common:

0. Gold data for the task.
1. Teacher's output labels.
2. Teacher's output scores ([Hinton et al. 2015](#)).
3. Teacher's final output states (cosine loss; [Sanh et al. 2019](#)).
4. Other teacher hidden states and embeddings ([Romero et al. 2015](#))
5. Student is trained to mimic the counterfactual behavior of the teacher under interventions ([Wu et al. 2022](#)).

For more: [Gou et al. 2021](#)

Distillation objectives

These can be used in combination to some extent:

1. **Standard**: One teacher with frozen parameters; only student parameters are updated.
2. **Multi-teacher**: An ensemble for the same task or potentially multi-task.
3. **Co-distillation**: The teacher and student are trained jointly. Also called ‘online distillation’ ([Anil et al. 2018](#)).
4. **Self-distillation**: The objective includes terms that seek to make some model components align with others ([Zhang et al. 2019](#)).

For more: [Gou et al. 2021](#)

Distillation performance

Distillation has been applied in many domains. The following point to a core result for GLUE ([Wang et al. 2018](#)): via distillation, we can increase efficiency with almost no loss in performance.

1. [Sanh et al. \(2019\)](#): Distill 12-layer BERT-base into 6 layers retaining 97% of GLUE performance.
2. [Sun et al. \(2019\)](#): Distill BERT-base into 3-layer and 6-layer variants, also maintaining good performance on GLUE.
3. [Jiao et al. \(2020\)](#): Distill BERT-base into 4-layers with similarly strong GLUE results.

Wrap-up

Other noteworthy architectures

- Transformer XL ([Dai et al. 2019](#)): Long contexts via recurrent connections to previous (frozen) states.
- XLNet ([Yang et al. 2019](#)): Bidirectional context with an autoregressive loss, via sampling of different sequence orders.
- DeBERTa ([He et al. 2021](#)): Separate representations for word and positions, with distinct attention connections.

BERT: Known limitations

1. Devlin et al. (2019): admirably detailed but still partial ablation studies and optimization studies.
RoBERTa
2. Devlin et al. (2019): “The first [downside] is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning.” ELECTRA
3. Devlin et al. (2019): “The second downside of using an MLM is that only 15% of tokens are predicted in each batch” ELECTRA
4. Yang et al. (2019): “BERT assumes the predicted tokens are independent of each other given the unmasked tokens, which is oversimplified as high-order, long-range dependency is prevalent in natural language” XLNet!

Pretraining data

1. OpenBookCorpus ([Bandy and Vincent 2021](https://huggingface.co/datasets/bookcorpusopen)):
<https://huggingface.co/datasets/bookcorpusopen>
2. The Pile ([Gao et al. 2020](https://pile.eleuther.ai)):
<https://pile.eleuther.ai>
3. Big Science Data ([Laurençon et al. 2022](https://huggingface.co/bigscience-data)):
<https://huggingface.co/bigscience-data>
4. Wikipedia processing:
<https://github.com/attardi/wikiextractor>
5. Pushshift Reddit Data ([Baumgartner et al. 2020](https://files.pushshift.io/reddit/)):
<https://files.pushshift.io/reddit/>

Current trends

1. Autoregressive architectures seem to have taken over, possibly just because the field is focused on generation.
2. Bidirectional models may still have the edge when it comes to representation.
3. seq2seq is still a dominant choice for tasks with that structure.
4. People are still obsessed with scaling up, but we are seeing a counter movement towards “smaller” models (still $\approx 10B$ parameters).



References I

- Rohan Anil, Gabriel Pereyra, Alexandre Passos, Róbert Ormándi, George E. Dahl, and Geoffrey E. Hinton. 2018. Large scale distributed neural network training through online distillation. *ArXiv*, abs/1804.03235.
- Jack Bandy and Nicholas Vincent. 2021. Addressing" documentation debt" in machine learning research: A retrospective datasheet for bookcorpus. *arXiv preprint arXiv:2105.05241*.
- Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. The PushShift Reddit dataset. In *Proceedings of the international AAAI conference on web and social media*, volume 14, pages 830–839.
- T. Brown, B. Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, P. Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, R. Child, Aditya Ramesh, D. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, E. Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, J. Clark, Christopher Berner, Sam McCandlish, A. Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.
- Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.
- Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2019. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.
- Andrew M Dai and Quoc V Le. 2015. *Semi-supervised sequence learning*. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.
- Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. *ArXiv:1901.02860*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. *BERT: Pre-training of deep bidirectional transformers for language understanding*. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.
- Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819.
- Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. *DeBERTa: Decoding-enhanced BERT with disentangled attention*. In *Proceedings of the International Conference on Learning Representations*.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. *Distilling the knowledge in a neural network*. *ArXiv:1503.02531*.
- Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. *TinyBERT: Distilling BERT for natural language understanding*. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.



References II

- Hugo Laurençon, Lucile Saulnier, Thomas Wang, Christopher Akiki, Albert Villanova del Moral, Teven Le Scao, Leandro Von Werra, Chenghao Mou, Eduardo González Ponferrada, Huu Nguyen, et al. 2022. The bigscience roots corpus: A 1.6 tb composite multilingual dataset. *Advances in Neural Information Processing Systems*, 35:31809–31826.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. ArXiv:1910.13461.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. *RoBERTa: A robustly optimized BERT pretraining approach*. ArXiv:1907.11692.
- Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. *Effective approaches to attention-based neural machine translation*. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.
- Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. *Learned in translation: Contextualized word vectors*. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6294–6305. Curran Associates, Inc.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. *Distributed representations of words and phrases and their compositionality*. In Christopher J. C. Burges, Leon Bottou, Max Welling, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. *GloVe: Global vectors for word representation*. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.
- Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. *Deep contextualized word representations*. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. *Improving language understanding by generative pre-training*. Ms, OpenAI.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.
- Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. ArXiv:2002.12327.



References III

- Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2015. [FitNets: Hints for thin deep nets](#). In *International Conference on Learning Representations*.
- Alexander Rush. 2018. [The annotated Transformer](#). In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60, Melbourne, Australia. Association for Computational Linguistics.
- Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. [arXiv:1910.01108](#).
- Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Lucchini, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. [arXiv preprint arXiv:2211.05100](#).
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. [Neural machine translation of rare words with subword units](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.
- Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. [Self-attention with relative position representations](#). [ArXiv:1803.02155](#).
- Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. [Patient knowledge distillation for BERT model compression](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. [GLUE: A multi-task benchmark and analysis platform for natural language understanding](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.
- Zhengxuan Wu, Atticus Geiger, Josh Rozner, Elisa Kreiss, Hanson Lu, Thomas Icard, Christopher Potts, and Noah D. Goodman. 2022. [Causal distillation for language models](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4295, Seattle, United States. Association for Computational Linguistics.
- Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. [XLNet: Generalized autoregressive pretraining for language understanding](#). In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc.



References IV

- Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. 2019. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3712–3721.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuhui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained Transformer language models. *arXiv preprint arXiv:2205.01068*.

Fantastic Language Models and How to Build Them

Guest Lecture — CS 224U: Natural Language Understanding
Stanford || Zoom || Folks 2x-ing the Recording
April 12, 2023



**Siddharth
Karamcheti**

On the Importance of “Building”

Today – a *practical* take on large-scale language models (LLMs).

Whirlwind tour of the full pipeline:

- **Model Architecture** – Evolution of the Transformer
- **Training at Scale** – From 124M to 1T+ Parameters
- **Efficient Finetuning & Inference** – Tips & Tricks

Punchline: From “folk knowledge” → insight / intuition / (re-)discovery!

Please ask lots of questions! Why is this information useful to <YOU>?

Part I: Evolution of the Transformer

“Experiment is the mother of knowledge.”
— Madeline L’Engle, *A Wrinkle in Time*

Recipe for a Good™ Language Model

Massive amounts of cheap, easy to acquire data...



... a simple, high-throughput way to *consume* it!

Natural to scale with data.
Composable and “general”.

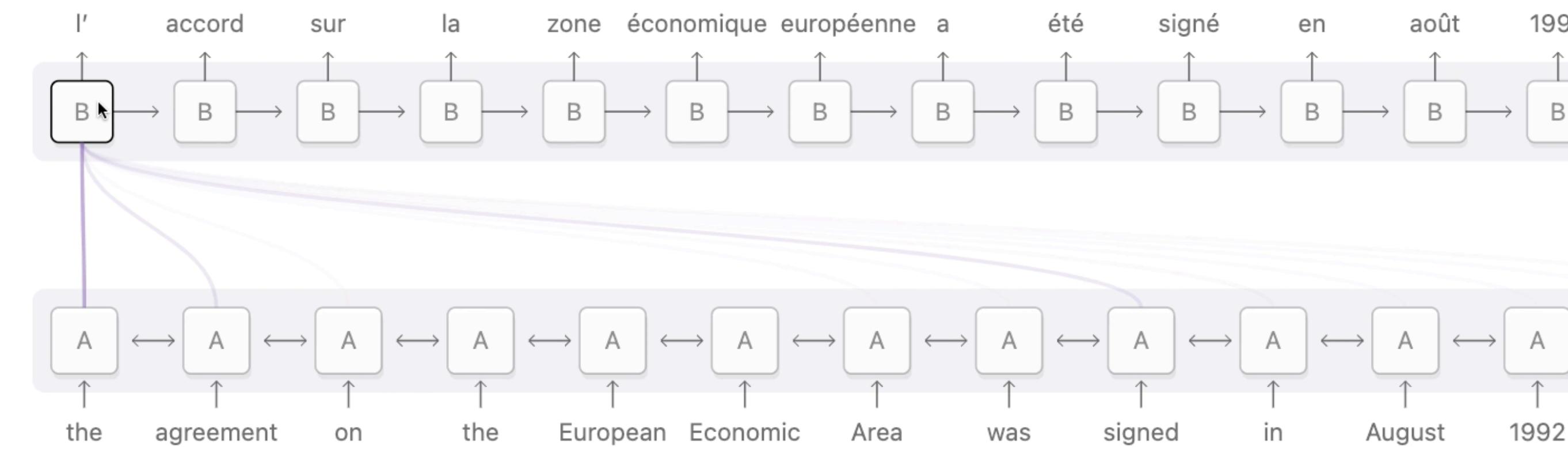
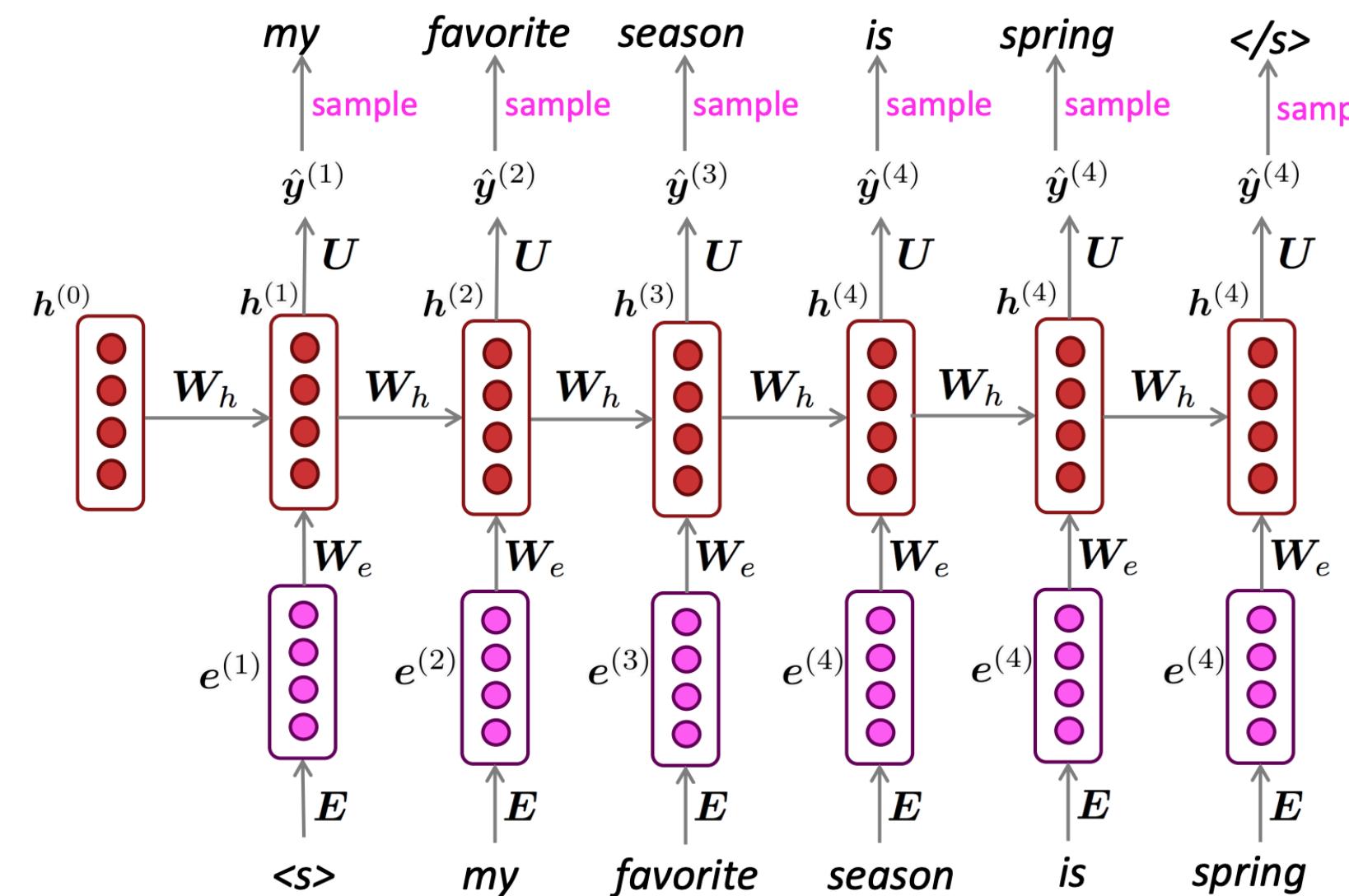
Fast & parallelizable training.
High hardware utilization.

Minimal “assumptions” on
relationships between data?

<Story Time>

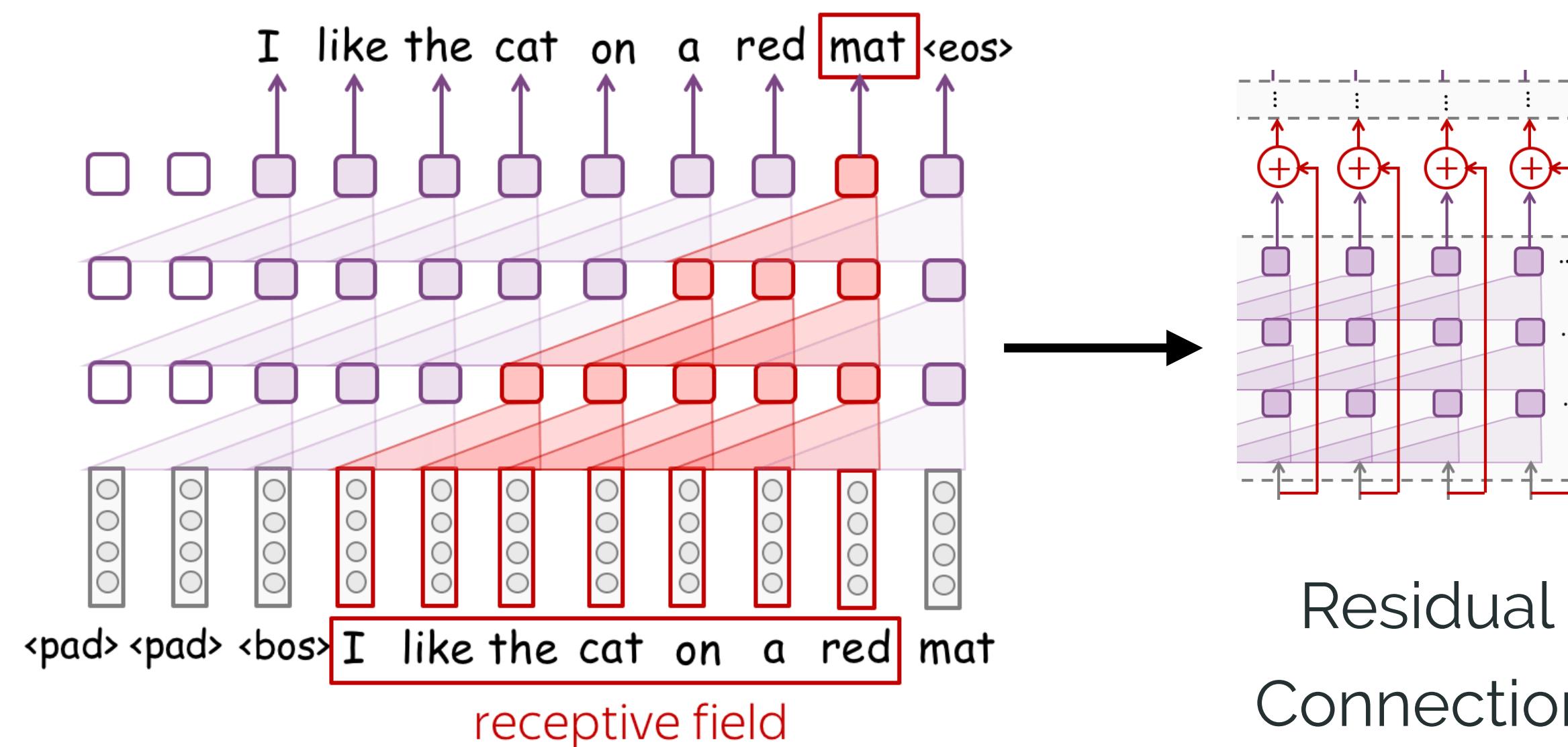
Pre-2017 – Historical Context

RNNs



RNN Key Ideas: Long Context, **Attention**

CNNs



Residual
Connection

CNN Key Ideas:

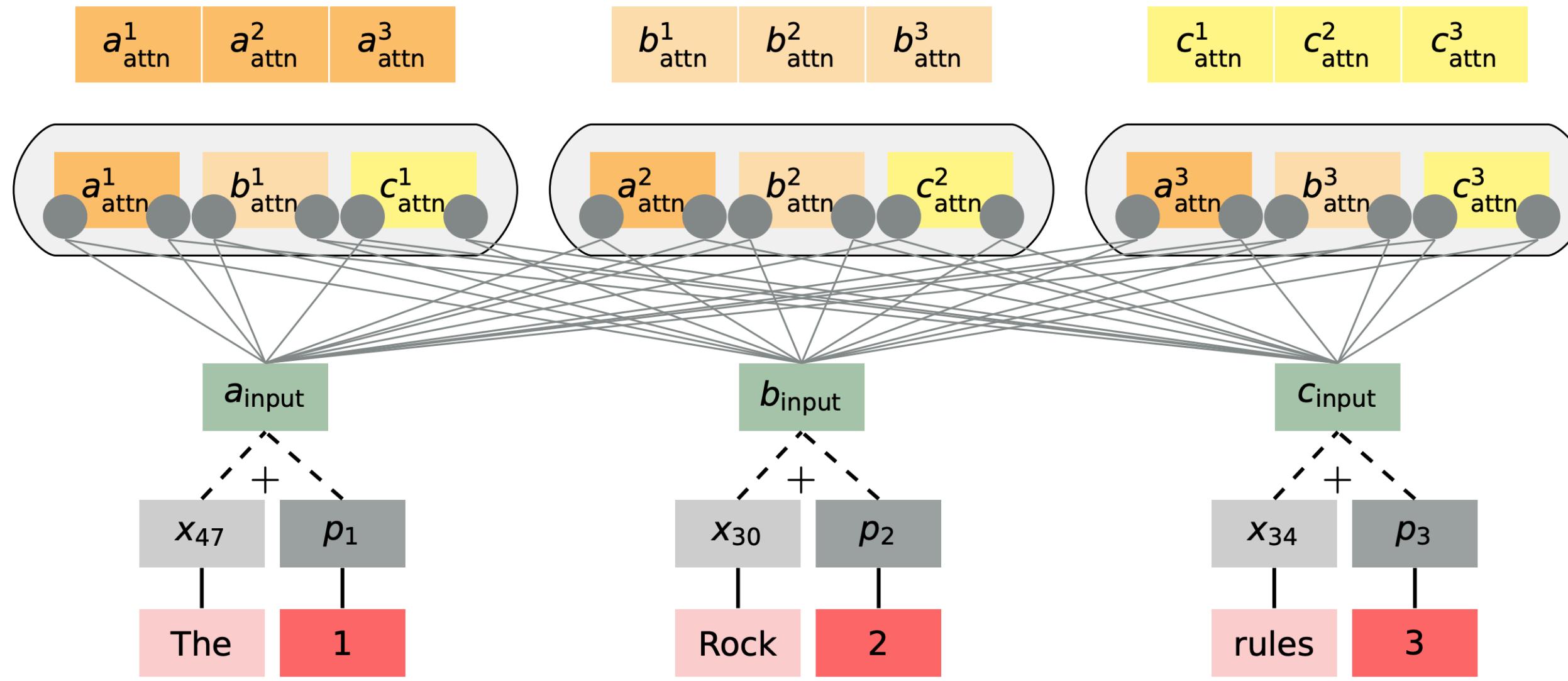
- Layer: Multiple “Filters” (Views)
- **Scaling Depth** w/ Residuals
- **Parallelizable!**

< How do I do better? >

Reference: “Attention and Augmented Recurrent Neural Networks,” Chris Olah and Shan Carter. *Distill*, 2016.

Reference: “Convolutional Neural Networks for Text,” Lena Voita. ML for NLP @ YSDA

Formulating the Self-Attention Block



```
class Attention(nn.Module):
    def __init__(self, embed_dim: int, n_heads: int):
        super().__init__()
        self.n_heads, self.dk = n_heads, (embed_dim // n_heads)
        self.qkv = nn.Linear(embed_dim, 3 * embed_dim)
        self.proj = nn.Linear(embed_dim, embed_dim)

    def forward(self, x: Tensor[bsz, seq, embed_dim]):
        q, k, v = rearrange(
            self.qkv(x),
            "bsz seq (qkv nh dk) -> qkv bsz nh seq dk",
            qkv=3,
            nh=self.n_heads, # Different "views" (like CNN filters)!
            dk=self.dk,
        ).unbind(0)
```

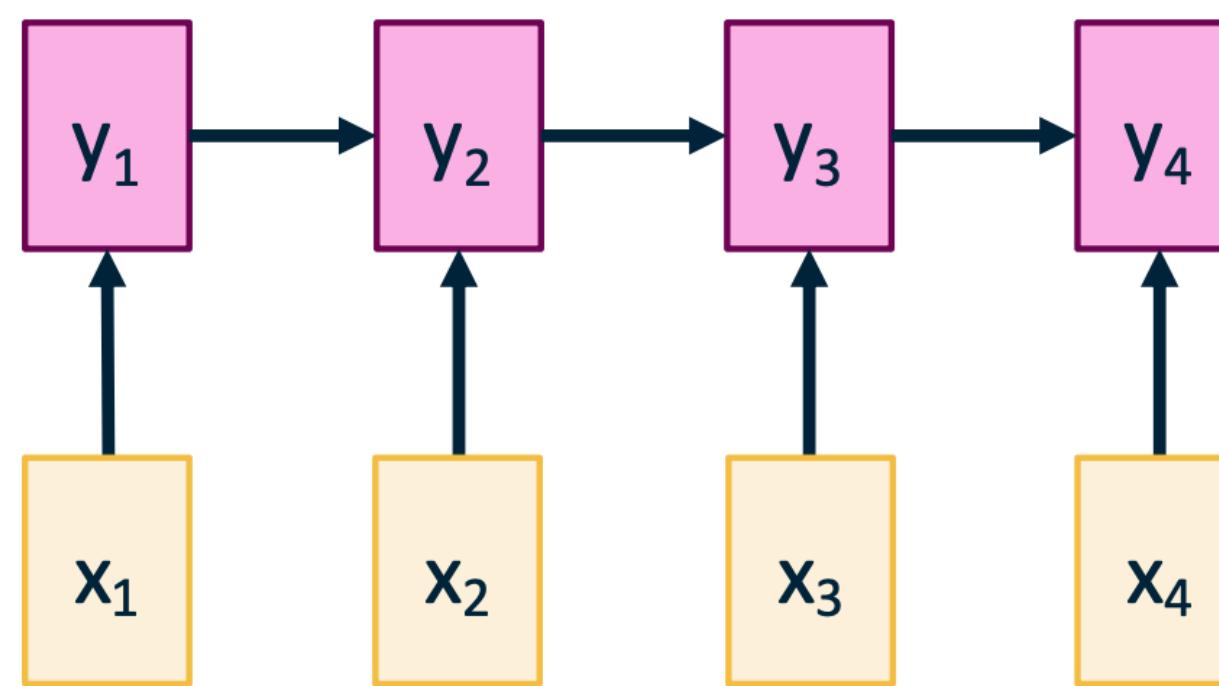
Self-Attention: “The” → query, key, & value

Multi-Headed: Different “views” per layer

< Is this actually better? >

Aside – Self-Attention & Parallelization

Recurrent Neural Network

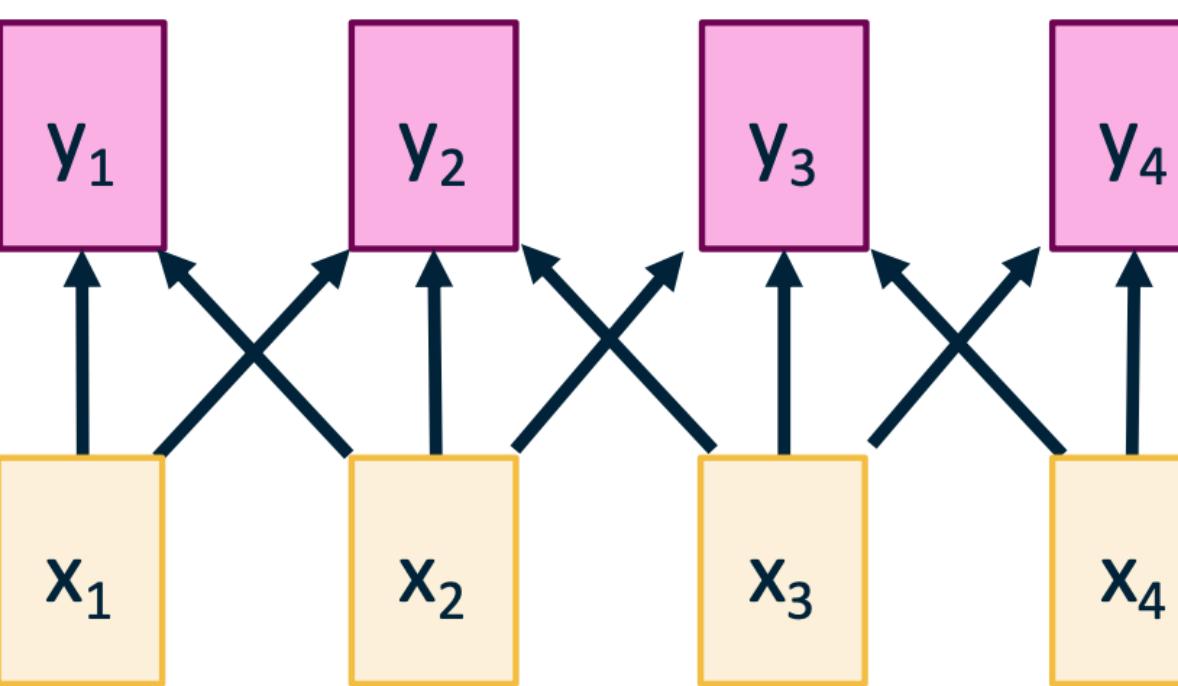


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

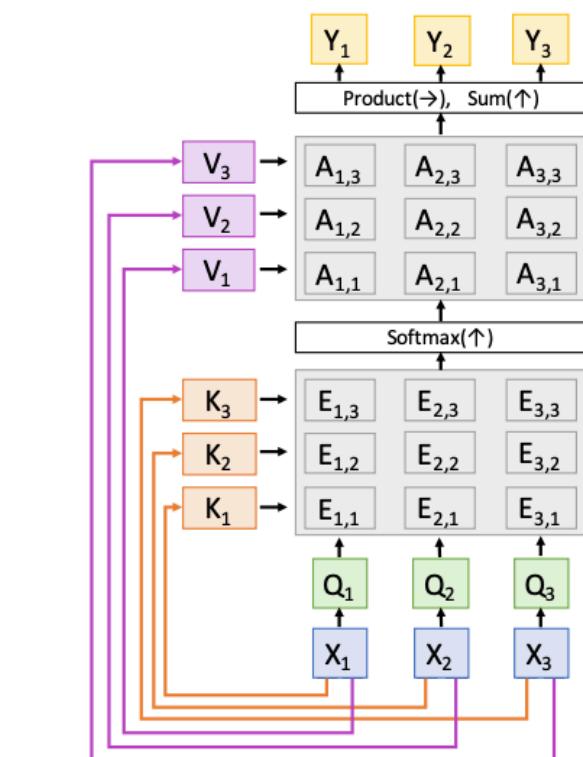


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(+) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

< Great! But... what am I missing? >

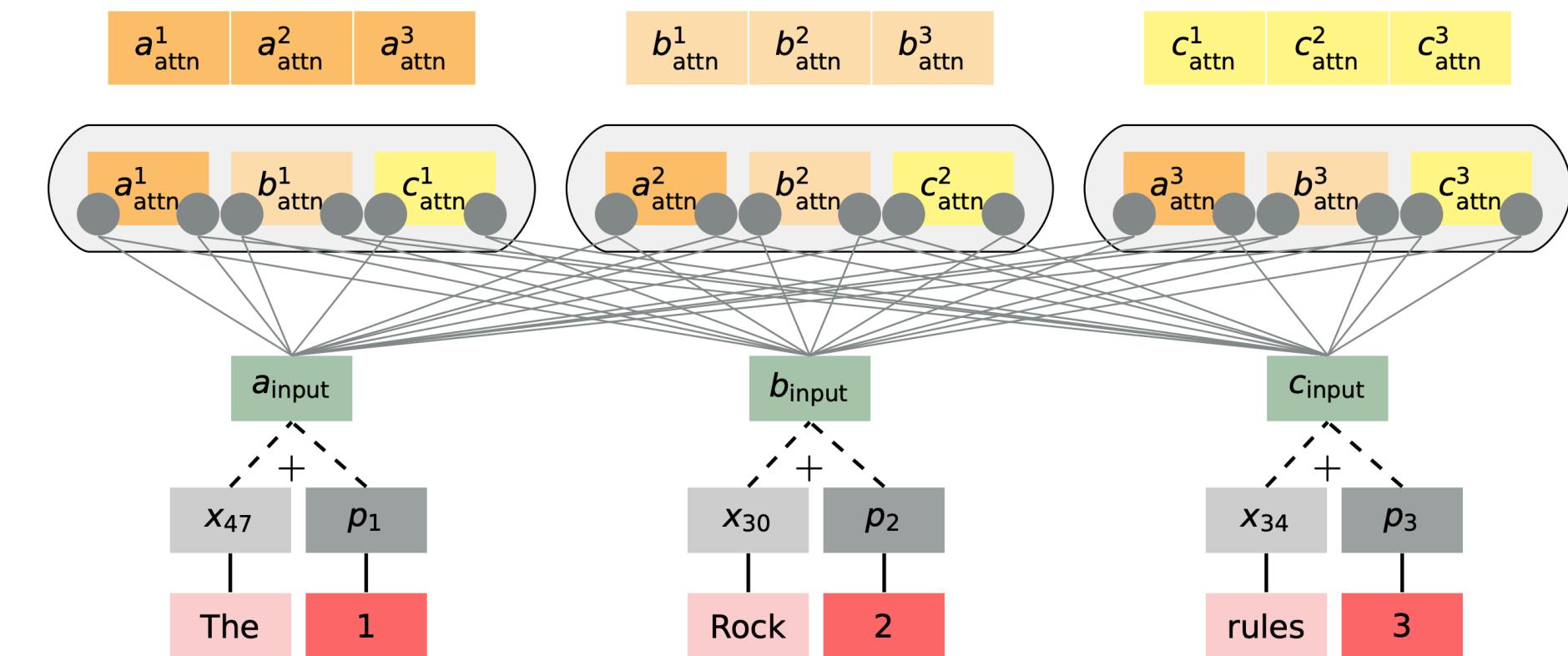
Formulating the Self-Attention Block



```
class Attention(nn.Module):
    def __init__(self, embed_dim: int, n_heads: int):
        super().__init__()
        self.n_heads, self.dk = n_heads, (embed_dim // n_heads)
        self.qkv = nn.Linear(embed_dim, 3 * embed_dim)
        self.proj = nn.Linear(embed_dim, embed_dim)

    def forward(self, x: Tensor[bsz, seq, embed_dim]):
        q, k, v = rearrange(
            self.qkv(x),
            "bsz seq (qkv nh dk) -> qkv bsz nh seq dk",
            qkv=3,
            nh=self.n_heads, # Different "views" (like CNN filters)!
            dk=self.dk,
        ).unbind(0)

# RNN Attention --> *for each view*
        scores = torch.softmax(
            q @ (k.transpose(-2, -1)),
            dim=-1
        )
        return self.proj(
            rearrange(scores @ v, "b nh seq dk -> b seq (nh dk)")
        )
```



< Where's my nonlinearity? >

Expressivity & Nonlinearity

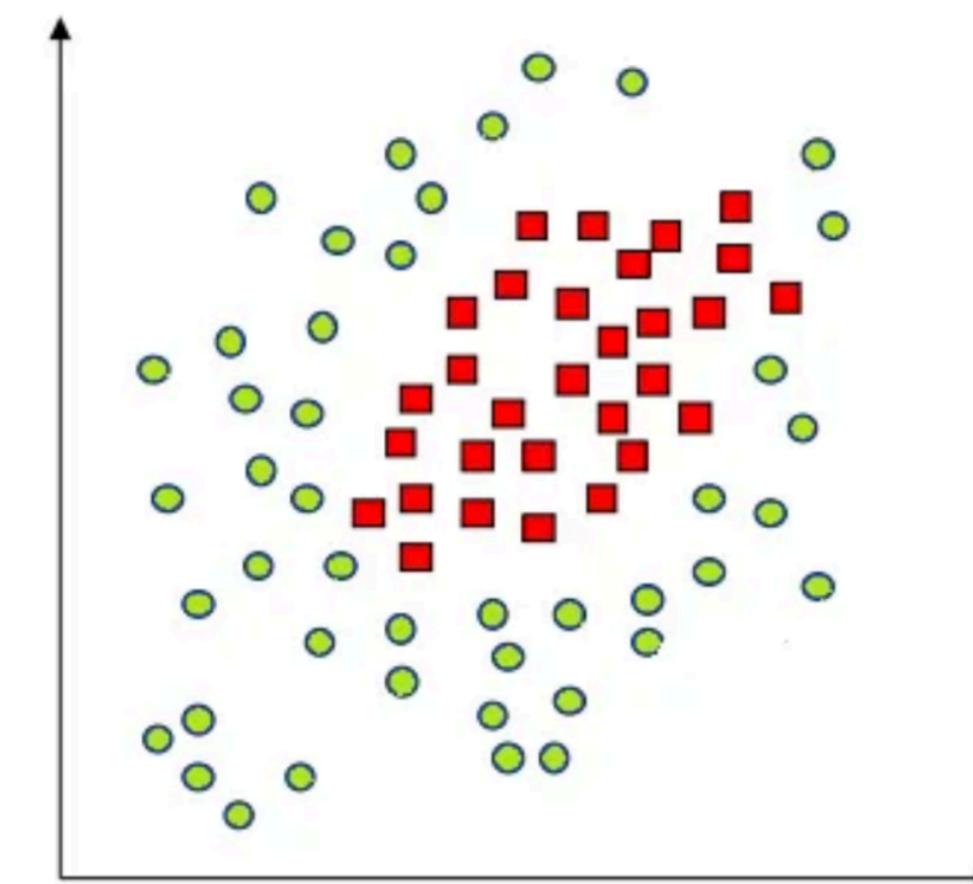


```
class ExpressiveTransformerBlock(nn.Module):
    def __init__(self, embed_dim: int, n_heads: int, up: int = 4):
        super().__init__()
        self.attn = Attention(embed_dim, n_heads)

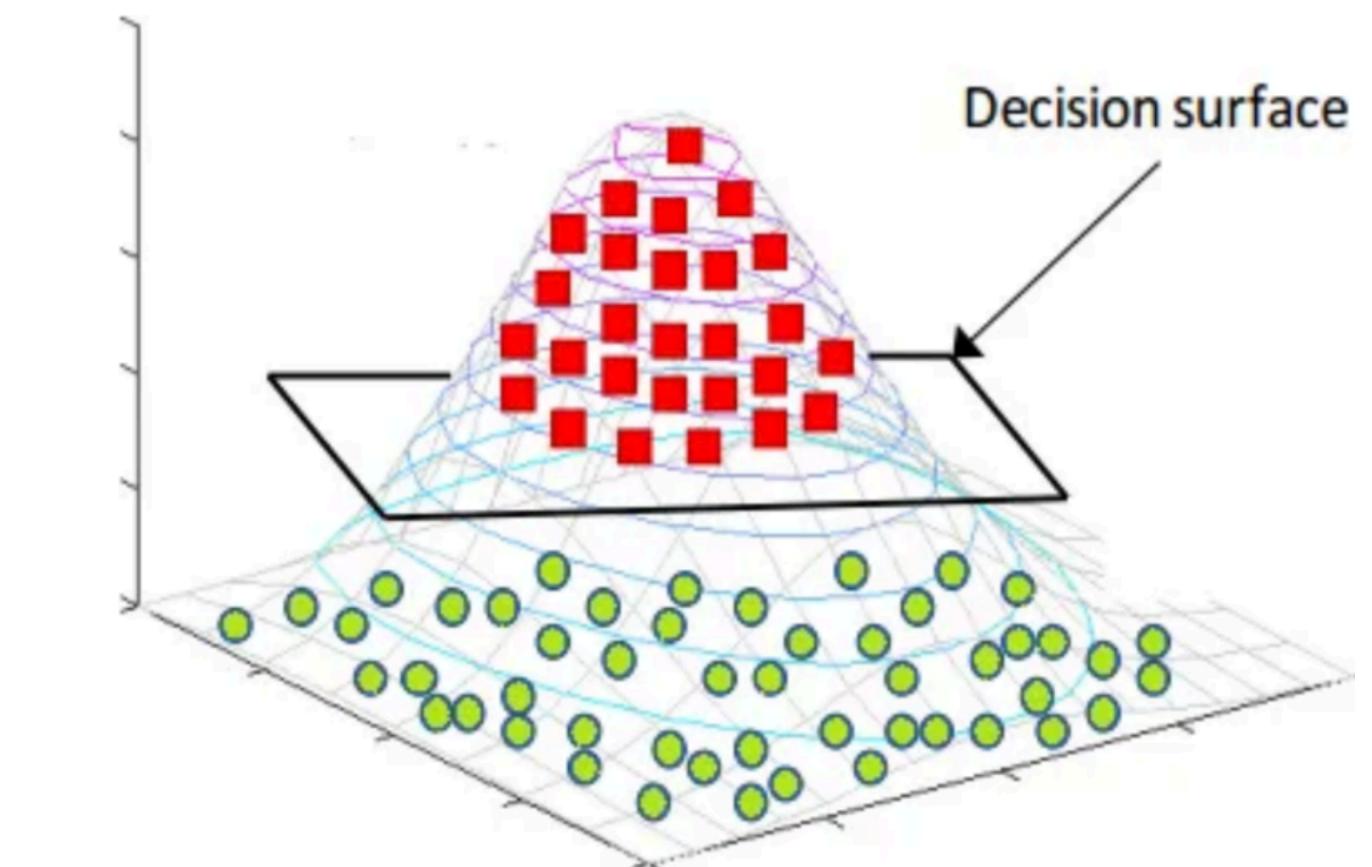
    # Project *up* to high-dimension, nonlinear, compress!
    self.mlp = nn.Sequential(
        nn.Linear(embed_dim, up * embed_dim),
        nn.ReLU(),
        nn.Linear(up * embed_dim, embed_dim)
    )

    def forward(self, x: T[bsz, seq, embed_dim]):
        x = x + self.attn(x)
        x = x + self.mlp(x)
        return x
```

Residual + MLP → “Sharpen” + “Forget”



CS 229 → SVMs & “Implicit Lifting”



< New Problem – Activations Blow Up! >

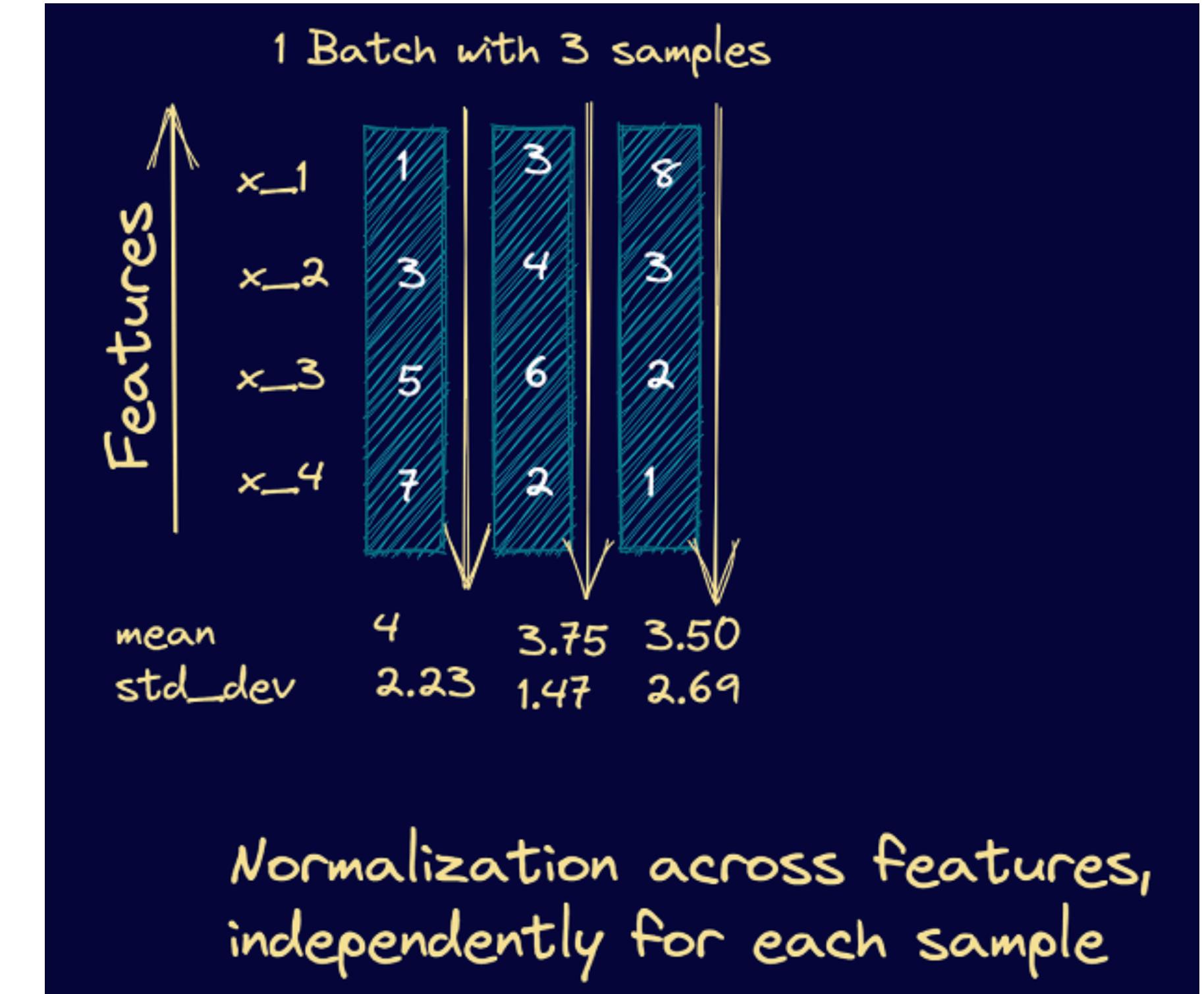
Going Deeper → Activation Instability



```
class NormalizedTransformerBlock(nn.Module):
    def __init__(self, embed_dim: int, n_heads: int, up: int = 4):
        super().__init__()
        self.attn = Attention(embed_dim, n_heads)
        self.mlp = nn.Sequential(
            nn.Linear(embed_dim, up * embed_dim),
            nn.ReLU(),
            nn.Linear(up * embed_dim, embed_dim)
        )

        # Add Normalization Layers
        self.attn_norm = nn.LayerNorm(embed_dim)
        self.mlp_norm = nn.LayerNorm(embed_dim)

    def forward(self, x: T[bsz, seq, embed_dim]):
        x = self.attn_norm(x + self.attn(x))
        x = self.mlp_norm(x + self.mlp(x))
        return x
```



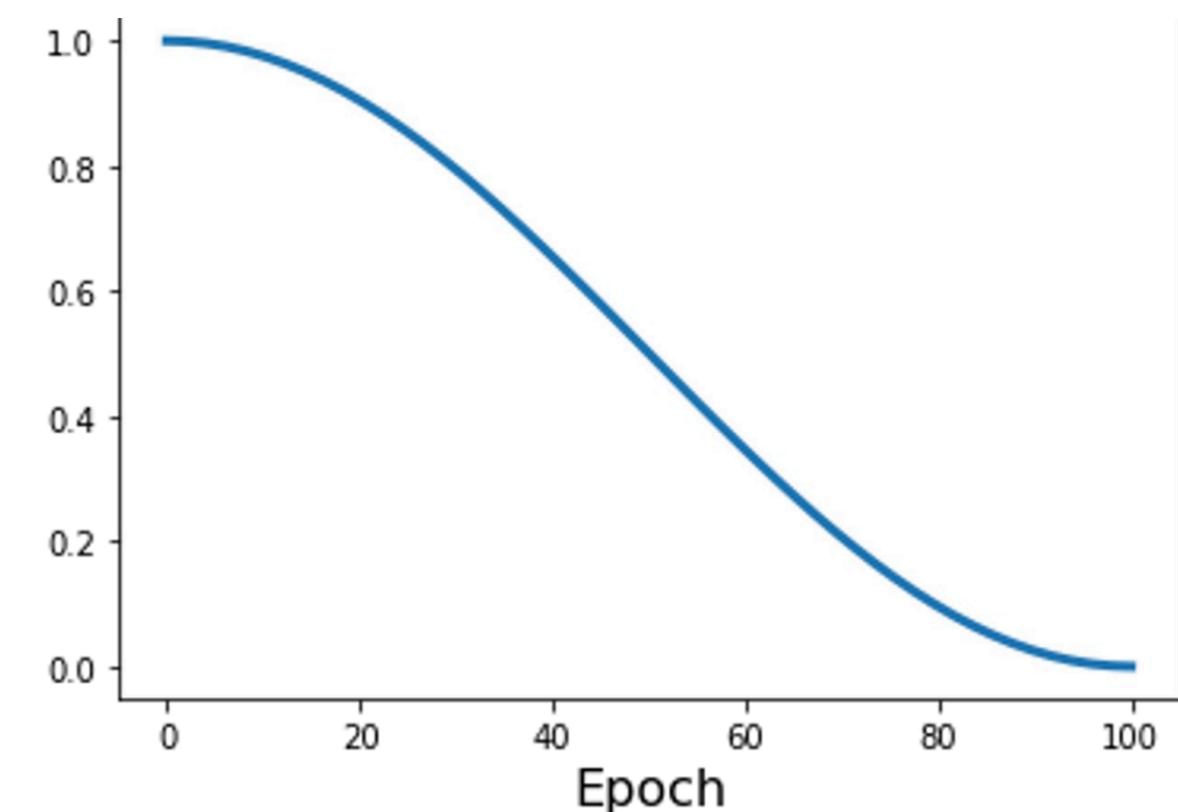
Layer Normalization

< And... we're done? >

Well, Shucks → Emergent Optimization Problems

Typical LR Decay

[CS 221, CS 229]

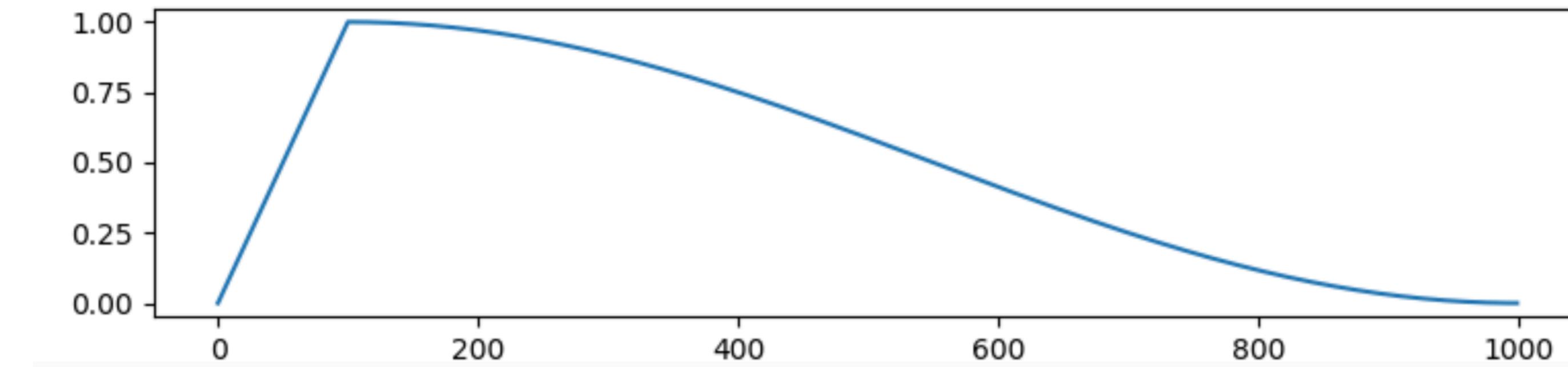


Normalized
Transformer



Transformer Pretraining LR Schedule

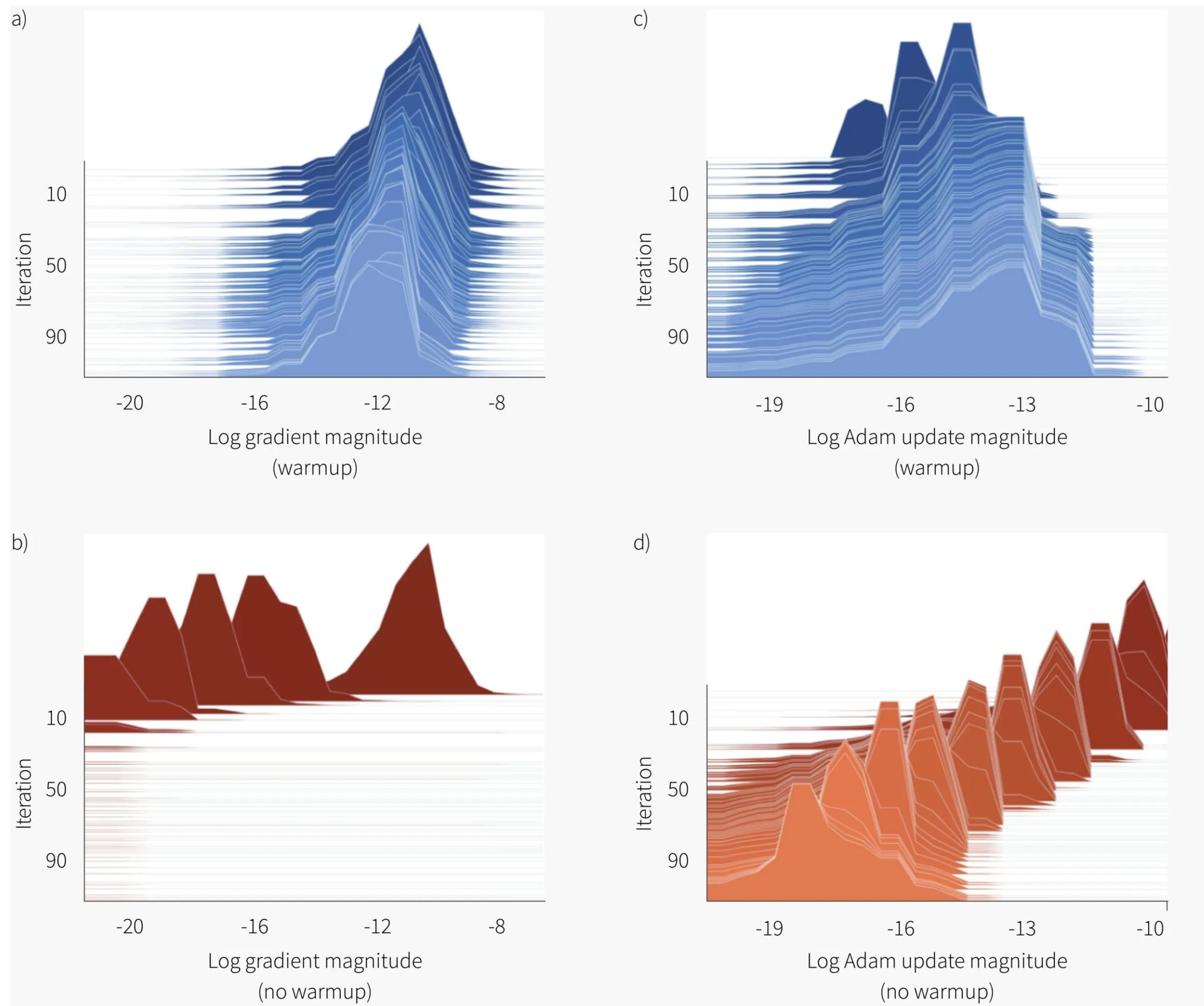
Linear Warmup (5% of Training) then Decay...



Learning Rate Warmup → Breaks conventional machine learning wisdom?

< Ok but... why? >

3 Years Later...



3.1. Problem in Transformer Optimization

In this section we demonstrate that the requirement for warmup comes from a combined effect of high variance in the Adam optimizer and backpropagation through layer normalization. Liu et al. (2020) showed that at the begin-

of the input. Specifically, the gradient has the following property:

$$\left\| \frac{\partial \text{LN}(\mathbf{x})}{\partial \mathbf{x}} \right\| = O\left(\frac{\sqrt{d}}{\|\mathbf{x}\|} \right) \quad (1)$$

where \mathbf{x} is the input to layer normalization and d is the embedding dimension. If input norm $\|\mathbf{x}\|$ is larger than \sqrt{d} then backpropagation through layer normalization has a down scaling effect that reduces gradient magnitude for lower layers. Compounding across multiple layers this can quickly lead to gradient vanishing.

< Ok, now we're done...? >

The Modern Transformer (March 2023)



```
class ModernTransformerBlock(nn.Module):
    def __init__(self, embed_dim: int, n_heads: int, up: int = 4):
        super().__init__()
        self.attn = Attention(embed_dim, n_heads, qk_bias=False)
        self.mlp = nn.Sequential(
            SwishGLU(embed_dim, up * embed_dim),
            nn.Linear(up * embed_dim, embed_dim)
        )

    # Post-Norm --> *Pre-Norm*
    self.pre_attn_norm = RMSNorm(embed_dim)
    self.pre_mlp_norm = RMSNorm(embed_dim)

    def forward(self, x: T[bsz, seq, embed_dim]):
        x = x + self.attn(self.pre_attn_norm(x))
        x = x + self.mlp(self.pre_mlp_norm(x))
        return x
```



```
# SwishGLU -- A Gated Linear Unit (GLU) with Swish Activation
class SwishGLU(nn.Module):
    def __init__(self, in_dim: int, out_dim: int):
        super().__init__()
        self.swish = nn.SiLU()
        self.project = nn.Linear(in_dim, 2 * out_dim)

    def forward(self, x: T[bsz, seq, embed_dim]):
        projected, gate = self.project(x).tensor_split(2, dim=-1)
        return projected * self.swish(gate)

# RMSNorm -- Simple Alternative to LayerNorm
class RMSNorm(nn.Module):
    def __init__(self, dim: int, eps: float = 1e-8):
        super().__init__()
        self.scale, self.eps = dim**-0.5, eps
        self.g = nn.Parameter(torch.ones(dim))

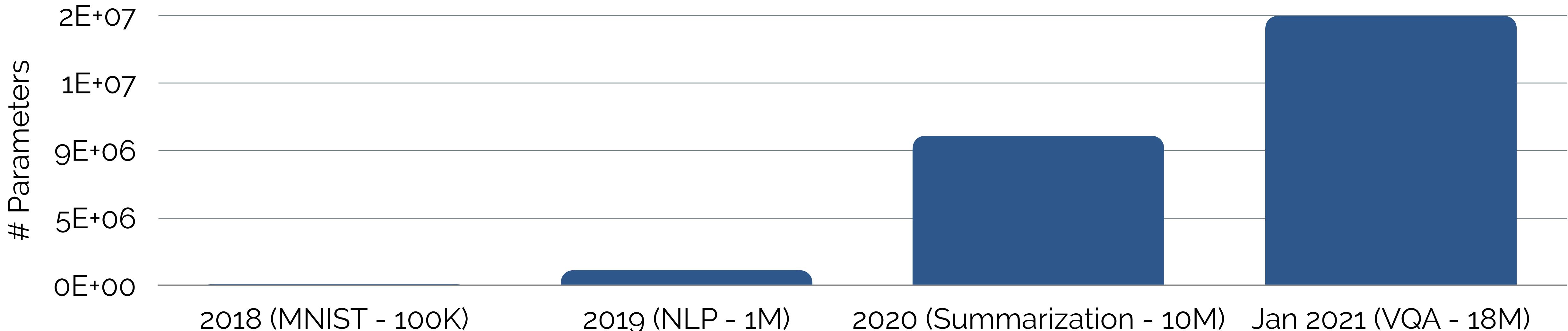
    def forward(self, x: T[bsz, seq, embed_dim]):
        norm = torch.norm(x, dim=-1, keepdim=True) * self.scale
        return x / norm.clamp(min=self.eps) * self.g
```

< Fin />

Part II: Training at Scale

“Nothing in life is to be feared. It is only to be understood.”
— Marie Curie

Short Story – My Deep Learning Trajectory



- “Standard Pipeline”: Train on 1 GPU (e.g., on Colab) —> **~max of a few hours.**
- Let's train a GPT-2 Small (124M)!
 - **Problem:** Batch > 4 goes OOM on a decent GPU = > **12 GB** of GPU RAM
 - Simple Trick —> *Gradient Accumulation!*
 - But... **99.63 Days** to train on Single GPU (400K Steps)

GPT-2 Training Clock



Shortening the Clock → The Scaling Toolbox

GPT-2 Training Clock

99.63 D

Goal: 100 Days on 1 GPU → ~**4 Days on 16 GPUs**

- **Data Parallelism** – Scaling across GPUs & Nodes
- **Mixed Precision** – Bits, Bytes, and TensorCores
- **ZeRO Redundancy** – Minimizing Memory Footprint

Later... Model Parallelism – Hardware Limitations – Software Optimization

*Even if you're not training big models... **understanding breeds innovation!***

Data Parallelism – A Toy Example

GPT-2 Training Clock

99.63 D

```
● ● ●  
BATCH_SIZE = 128  
  
class MLP(nn.Module):  
    def __init__(  
        self, n_classes: int = 10, mnist_dim: int = 784, hidden: int = 128  
    ):  
        super().__init__()  
        self.mlp = nn.Sequential(  
            nn.Linear(mnist_dim, hidden),  
            nn.ReLU(),  
            nn.Linear(hidden, hidden),  
            nn.ReLU(),  
            nn.Linear(hidden, n_classes)  
    )  
  
    def forward(self, x: T[bsz, mnist_dim]):  
        return self.mlp(x)  
  
# Main Code  
dataloader = DataLoader(dataset=torchvision.datasets(...), batch_size=BATCH_SIZE)  
model = MLP()  
  
# Train Loop  
criterion, opt = nn.CrossEntropyLoss(), optim.AdamW(model.parameters())  
for (inputs, labels) in dataloader:  
    loss = criterion(model(inputs), labels)  
    loss.backward(); opt.step(); opt.zero_grad()
```

Idea → Parallelize?

SIMD

Single Instruction, Multiple Data



SPMD

Single Program, Multiple Data

< Seems hard? >

(Distributed) Data Parallelism – Implementation

GPT-2 Training Clock

99.63 D

7.2 D – 16 GPUs w/ Data Parallelism (DDP)

```
● ● ●  
from torch.nn.parallel import DistributedDataParallel as DDP  
from torch.utils.data.distributed import DistributedSampler  
  
BATCH_SIZE, WORLD_SIZE = 128, 8 # World Size == # of GPUs  
class MLP(nn.Module):  
    def __init__(  
        self, n_classes: int = 10, mnist_dim: int = 784, hidden: int = 128  
    ):  
        super().__init__()  
        self.mlp = nn.Sequential(  
            nn.Linear(mnist_dim, hidden),  
            nn.ReLU(),  
            nn.Linear(hidden, hidden),  
            nn.ReLU(),  
            nn.Linear(hidden, n_classes)  
    )  
  
    def forward(self, x: T[bsz, mnist_dim]):  
        return self.mlp(x)  
  
# Main Code  
train_set = torchvision.dataset(...)  
dist_sampler = DistributedSampler(dataset=train_set)  
dataloader = DataLoader(  
    train_set, sampler=dist_sampler, batch_size=BATCH_SIZE // WORLD_SIZE  
)  
  
model = DDP(  
    MLP(),  
    device_ids=[os.environ["LOCAL_RANK"]],  
    output_device=os.environ["LOCAL_RANK"]  
)
```

Auto-Partitions Data across Processes

Simple Wrapper around nn.Module()

```
# Train Loop  
criterion, opt = nn.CrossEntropyLoss(), optim.AdamW(model.parameters())  
for (inputs, labels) in dataloader:  
    loss = criterion(model(inputs), labels)  
    loss.backward(); opt.step(); opt.zero_grad()  
  
# Run: `torchrun --nnode 1 --nproc_per_node=8 main.py`
```

Nifty Utility → Spawns Processes

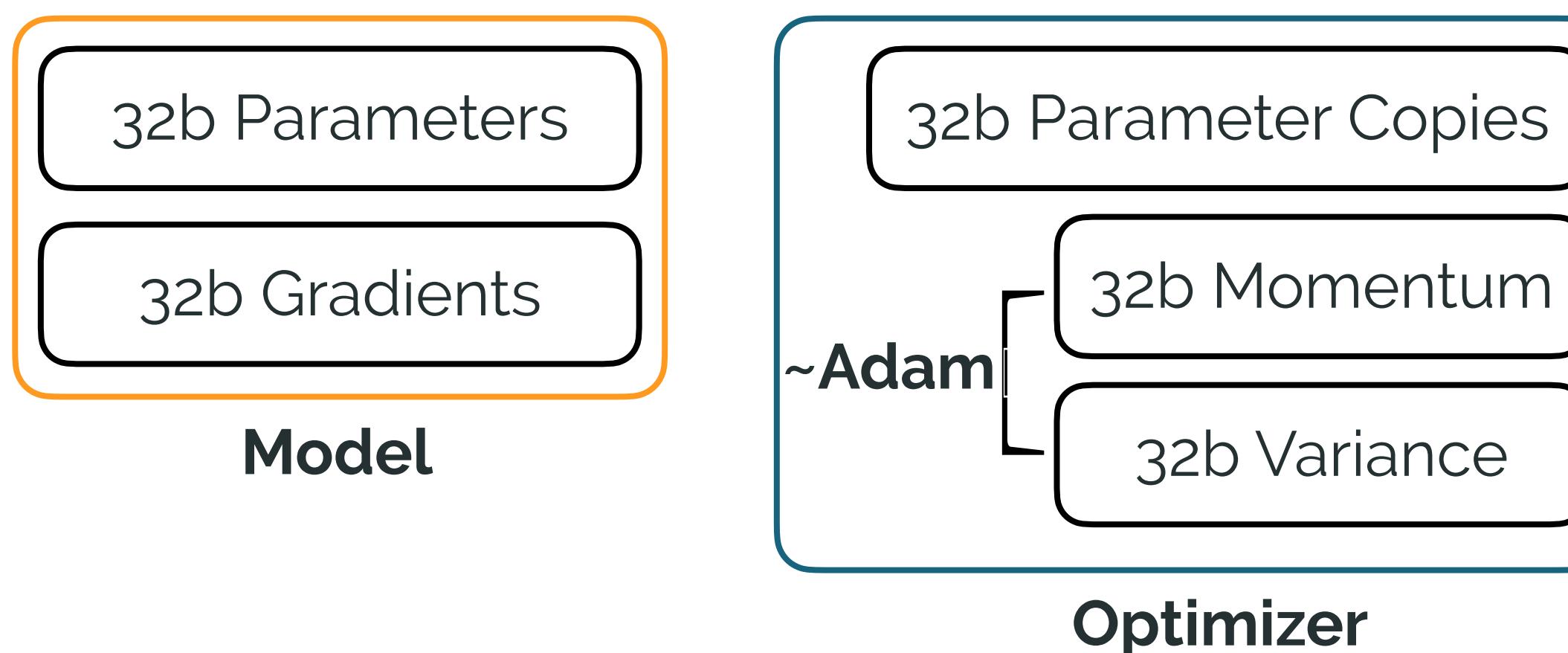
Important – Memory Footprint of Training?

GPT-2 Training Clock



Standard (Float 32) Memory Footprint

[Excludes Activations + Temporary Buffers]



Lower Bound on “Static” Memory (w/ Adam):
= **# Parameters * 20 Bytes**

Activation Memory >> Static Memory

Training Implications

- 1B Parameters → 18 GB (~**31 GB w/ BSZ = 1**)
- 175B Parameters → **3 TB (w/o activations!)**

Facts about Floating Points

- Float32 – Standard defined in IEEE-754
 - Sign (1) – Exponent (8) – Significand (23)
 - Wide Range → up to 1e38

< **Do we need *all* 32 bits? >**

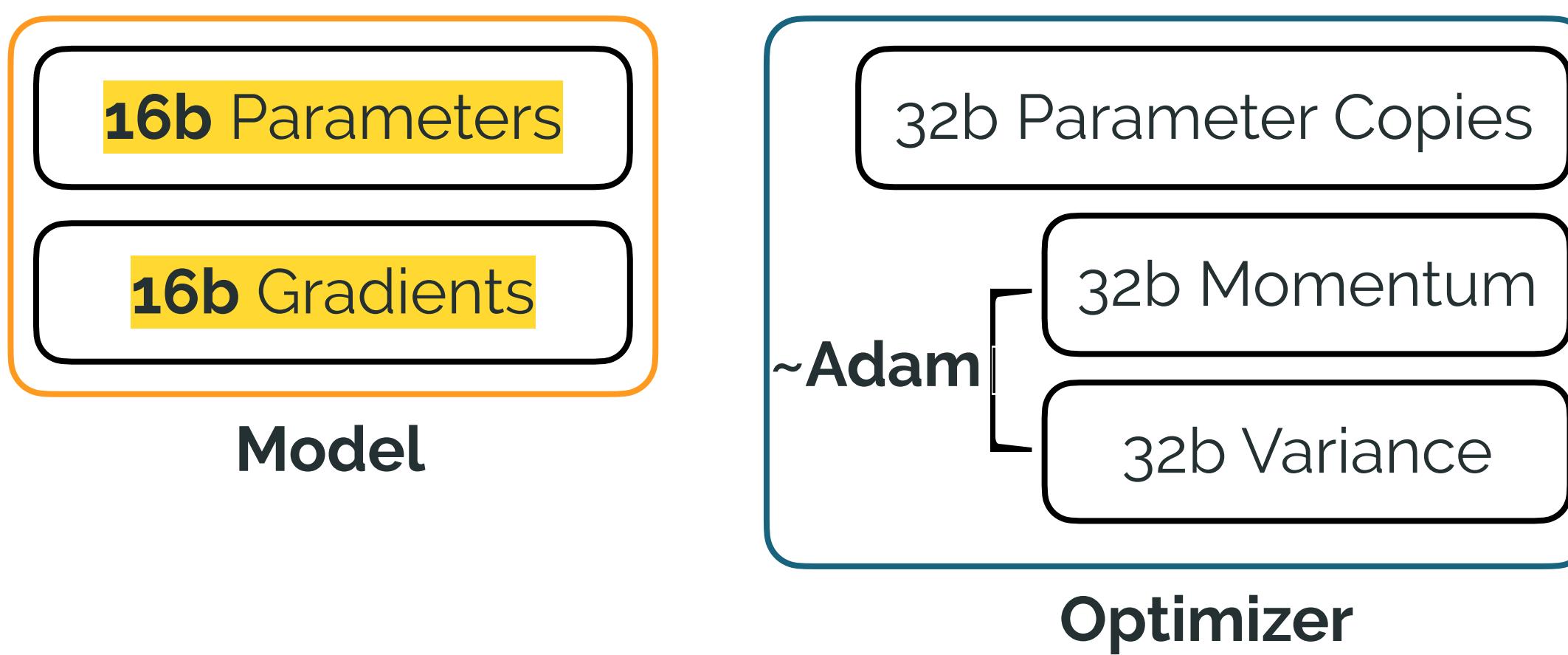
Mixed Precision Training

GPT-2 Training Clock



Mixed Precision (FP16) Memory Footprint

[Excludes Activations + Temporary Buffers]



Lower Bound on "Static" Memory (w/ Adam):

$$= \# \text{ Parameters} * \textbf{16 Bytes}$$

Activation Memory → **halved!**

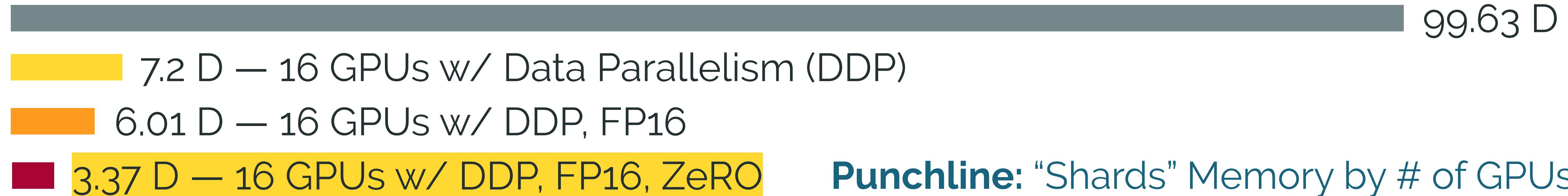
Hmm... Optimizer Memory?

FP16 **does not mean *everything*** is FP16.

Real Gain: NVIDIA Tensor Core Speedup!

Eliminate Redundancies → ZeRO

GPT-2 Training Clock



Punchline: “Shards” Memory by # of GPUs!

Standard Data Parallelism

“Replicate everything but the data!”

2Ψ Bytes

Model

2Ψ Bytes
[+ Buffers]

Gradients
[Entire Model]

12Ψ Bytes

Optimizer States
[Entire Model]

GPU 1

Model

Gradients
[Entire Model]

Optimizer States
[Entire Model]

GPU 2

Ψ = # of Parameters

ZeRO Data Parallelism

“Replicate only what you need”

2Ψ Bytes

Model

$(2\Psi / W)$ Bytes
[+ Buffers]

Model

Gradients
[Layers 1-6]

Opt. States
[Layers 1-6]

Model

Gradients
[Layers 7-12]

Opt. States
[Layers 7-12]

GPU 1

GPU 2

$(12\Psi / W)$ Bytes

W = # of GPUs

Alas – Hitting a (Communication) Wall

Problem – At some point, communication cost between nodes is too much!

Answers:

Exploit Matrix Multiplication...

$$\begin{array}{c} \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ \hline & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \times \begin{matrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{matrix} = \begin{matrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{matrix} \\ x_0 \text{ (2, 5)} \quad w \text{ (5, 8)} \quad out_0 \text{ (2, 8)} \end{array}$$

$$\begin{array}{c} \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ \hline & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix} \times \begin{matrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{matrix} = \begin{matrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{matrix} \\ x_1 \text{ (2, 5)} \quad w \text{ (5, 8)} \quad out_1 \text{ (2, 8)} \end{array}$$

$$\begin{array}{c} \begin{matrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{matrix} \\ out \text{ (4, 8)} \end{array}$$

Schedule Backwards Pass Wisely...



< Harder to implement, model-specific... still miles to go! >