<u>The "Deep Learning for NLP" Lecture Roadmap</u>

# Lecture 8: Transformers (2/2)

~~Lecture 5: Text Vectorization~~
~~and the Bag-of-Words Model~~
~~Lecture 6: Embeddings~~
~~Lecture 7: Transformers – (1/2)~~
Lectures 9-10: LLMs

MIT MANAGEMENT SLOAN SCHOOL

15.S04: Hands-on Deep Learning
Spring 2024
Farias, Ramakrishnan

# Review – Why Transformers?

We want to generate an output that has the same length as the input (so that we can classify each output element to the right slot type)

# Review – Why Transformers?

We want to generate an output that has the same length as the input (so that we can classify each output element to the right slot type)

In addition, we would like to

- Take the surrounding context of each word into account
- Take the order of the words into account

# Review –Transformer Architecture (1)

the train slowly left the station

# Review –Transformer Architecture (1)

the train slowly left the station

Input stand-alone embeddings

*initially random or pretrained (e.g., GloVe) weight vectors of embedding dimension D*

# Review –Transformer Architecture (1)

the train slowly left the station

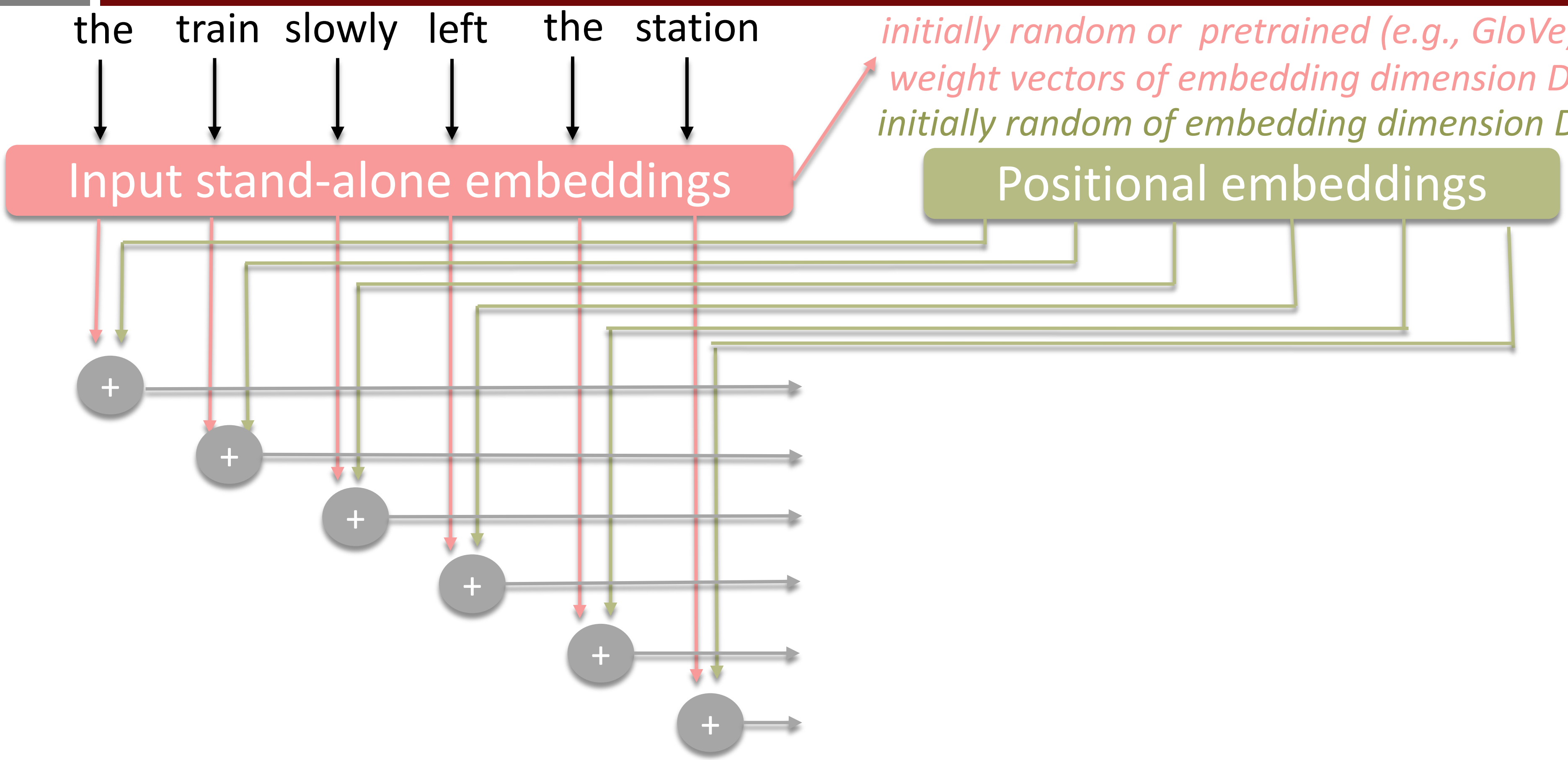
Input stand-alone embeddings

*initially random or pretrained (e.g., GloVe) weight vectors of embedding dimension D*

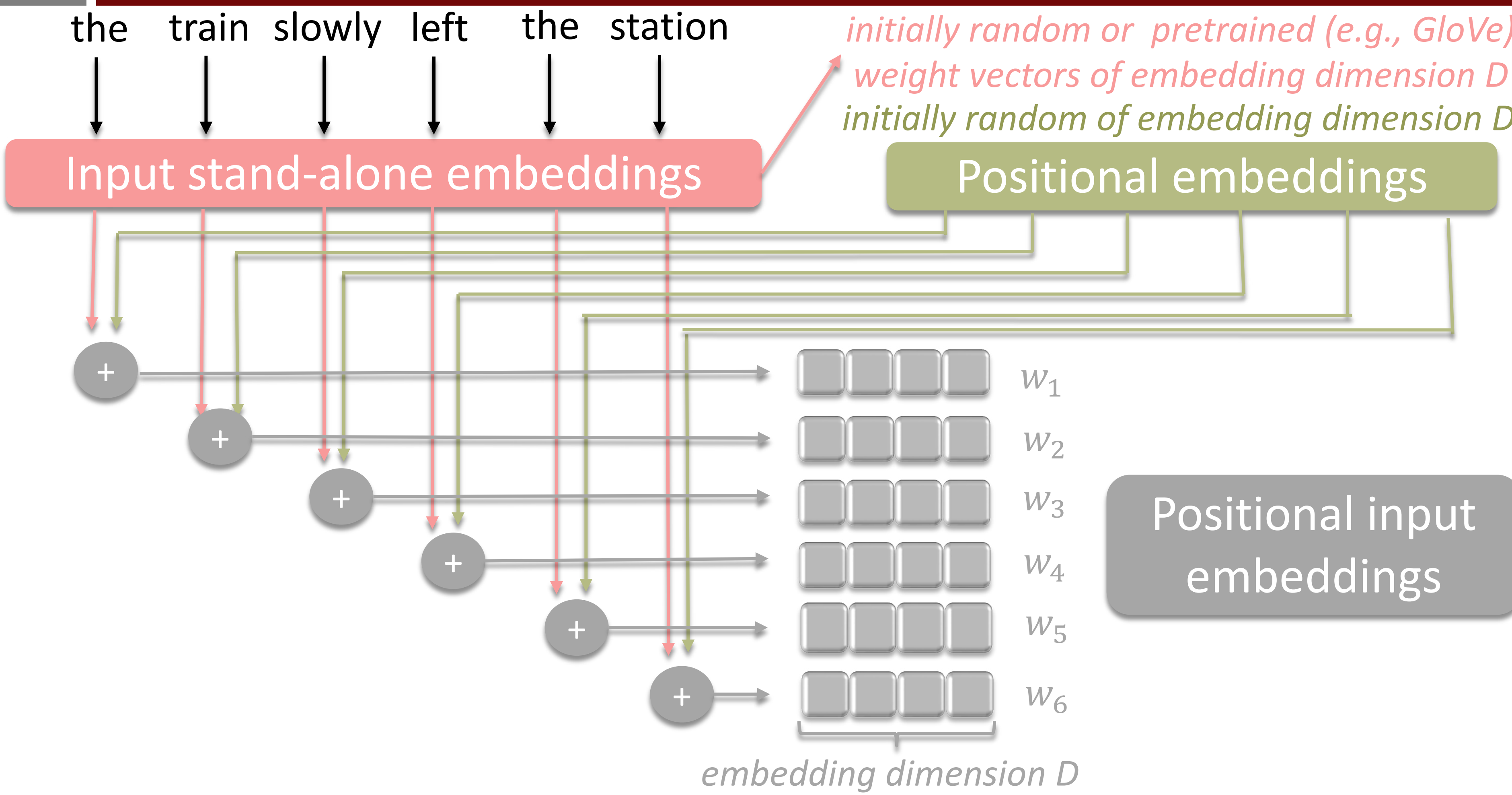
*initially random of embedding dimension D*

Positional embeddings

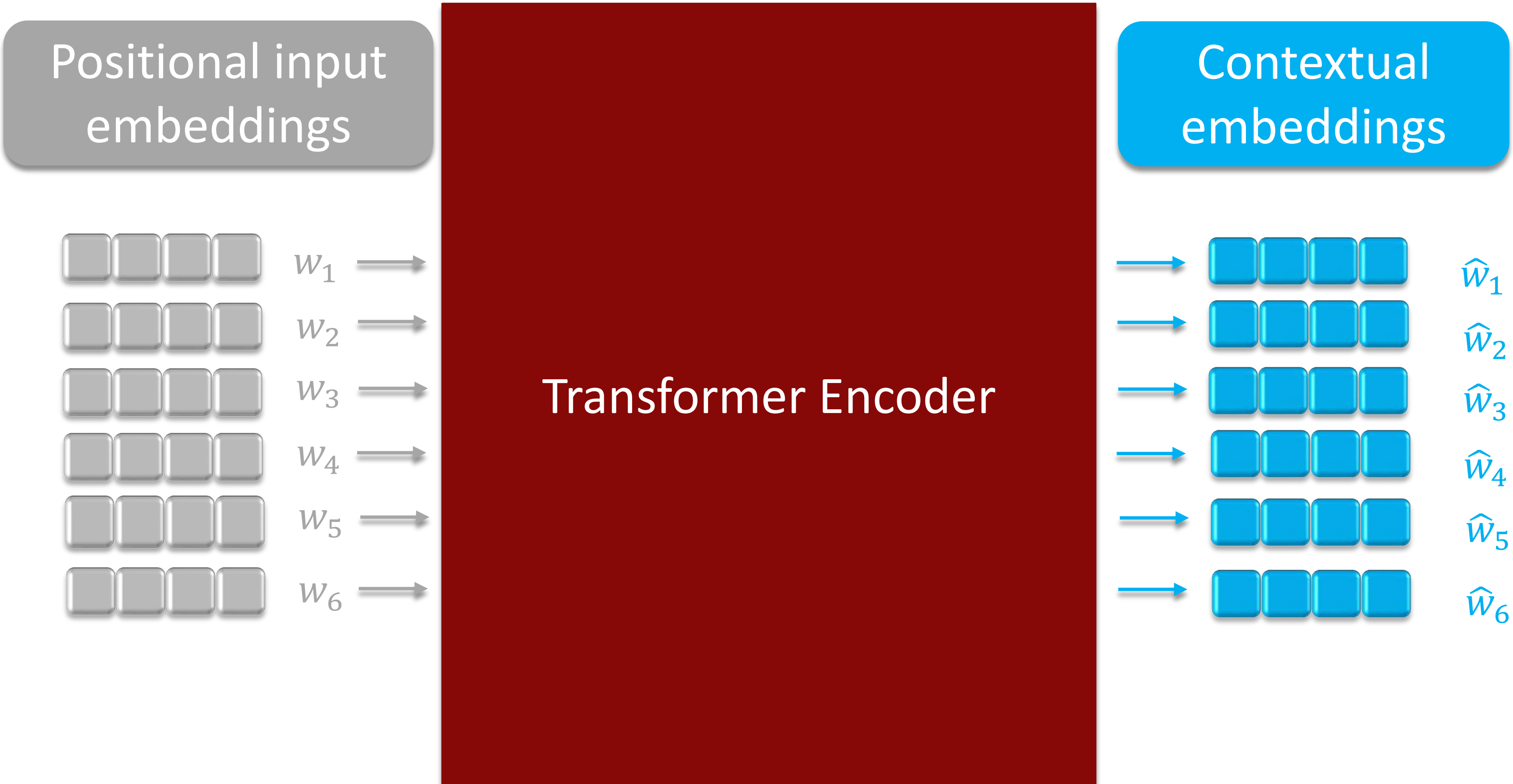
# Review –Transformer Architecture (1)



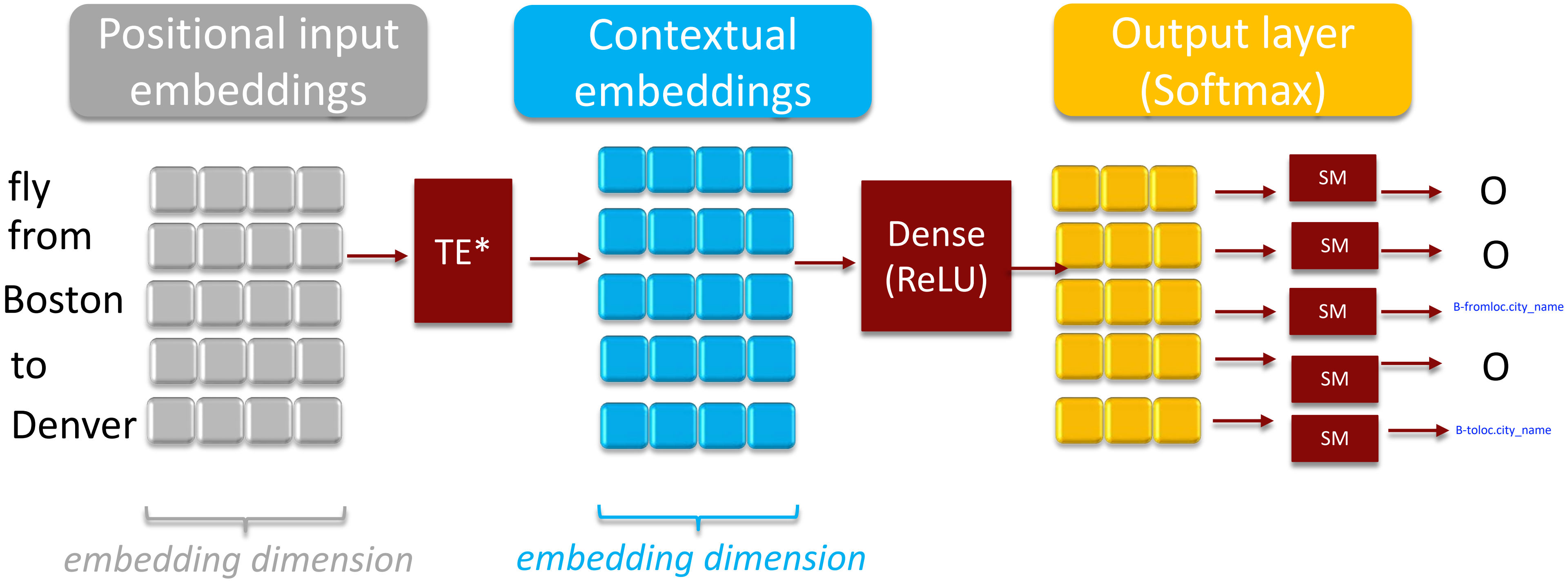
# Review –Transformer Architecture (1)



# Review –Transformer Architecture (2)



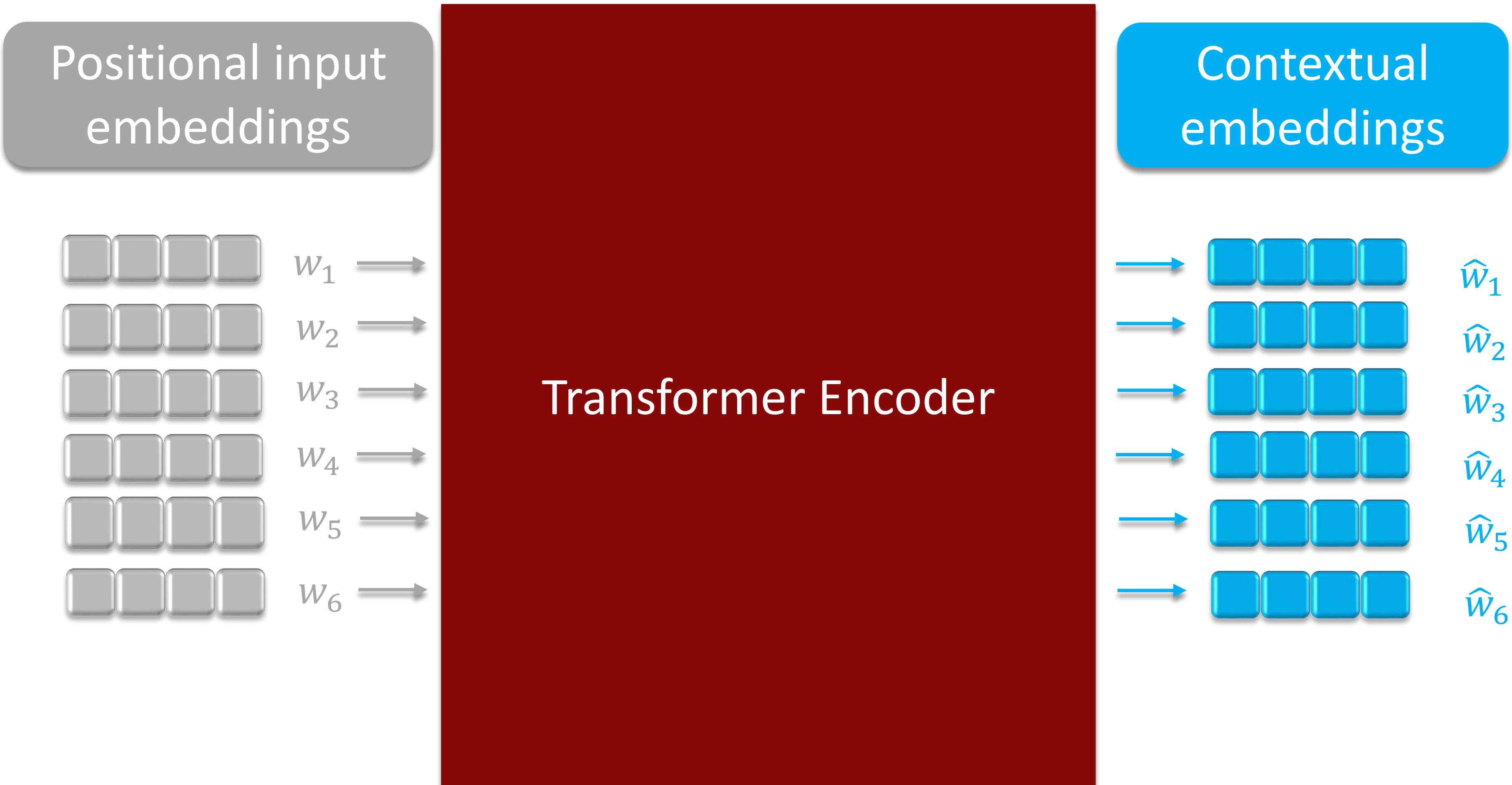
# Word-to-Slot Classification with Transformers (Revisited)




*Transformer Encoder

Indicate 4-element embedding vectors

# Transformer Encoder

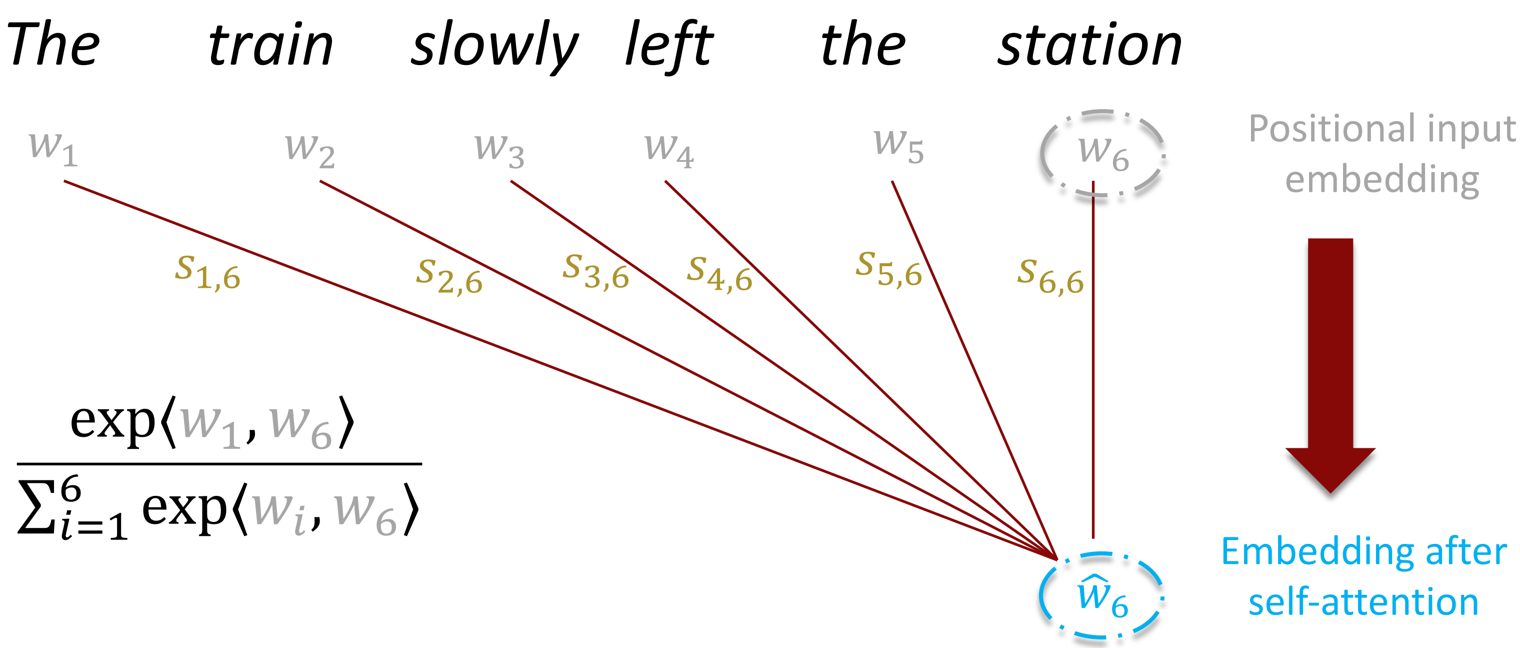


# We will now cover three (further) important elements of the Transformer Encoder*

- Making self-attention “tunable”
- Residual connections
- Layer normalization

* not covered in lecture 7

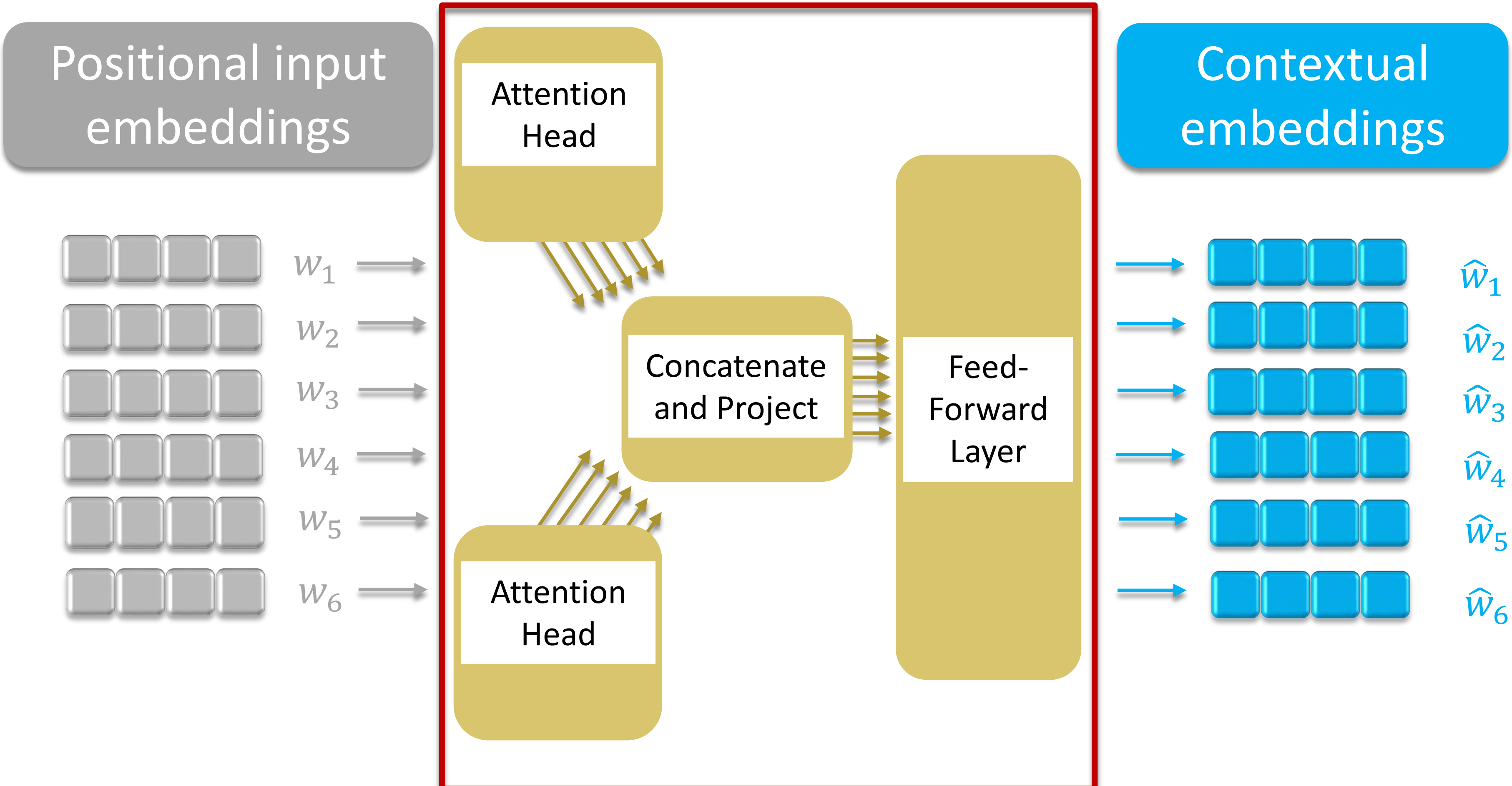
# Review: Self-Attention




$$s_{1,6} = \frac{\exp\langle w_1, w_6 \rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6 \rangle}$$

$$\boxed{\hat{w}_6 = s_{1,6} w_1 + s_{2,6} w_2 + s_{3,6} w_3 + s_{4,6} w_4 + s_{5,6} w_5 + s_{6,6} w_6}$$

As it stands, the self-attention heads don't have any internal weights so all the heads will produce the same output embeddings. We need to make each of them "tunable".



# How can we make this representation more "tunable"?

Please see

*HODL-SP24-Section-A-The Self-Attention Layer.pdf*

# Making Self-Attention Tunable

$$s_{1,6} = \frac{\exp\langle w_1, w_6 \rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6 \rangle} \Rightarrow s_{1,6} = \frac{\exp\langle \boldsymbol{A} w_1, \boldsymbol{A} w_6 \rangle}{\sum_{i=1}^{6} \exp\langle \boldsymbol{A} w_i, \boldsymbol{A} w_6 \rangle}$$

- We can multiply the positional input embeddings by a matrix $\boldsymbol{A}$ before their dot-product is computed.
  - Multiplying by a matrix $\boldsymbol{A}$ = a dense layer with a linear activation

# Making Self-Attention Tunable

$$s_{1,6} = \frac{\exp\langle w_1, w_6 \rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6 \rangle} \quad \Rightarrow \quad s_{1,6} = \frac{\exp\langle \boldsymbol{A} w_1, \boldsymbol{A} w_6 \rangle}{\sum_{i=1}^{6} \exp\langle \boldsymbol{A} w_i, \boldsymbol{A} w_6 \rangle}$$

- We can multiply the positional input embeddings by a matrix $\boldsymbol{A}$ before their dot-product is computed.
  - Multiplying by a matrix $\boldsymbol{A}$ = a dense layer with a linear activation

- <u>The key point</u>: The numbers in the matrix $\boldsymbol{A}$ are "learnable" weights i.e., weights that we will optimize with backprop. This is what we mean by "tunable"

# Making Self-Attention Tunable

$$s_{1,6} = \frac{\exp\langle \boldsymbol{A}w_1, \boldsymbol{A}w_6\rangle}{\sum_{i=1}^{6}\exp\langle \boldsymbol{A}w_i, \boldsymbol{A}w_6\rangle} \Rightarrow s_{1,6} = \frac{\exp\langle \boldsymbol{A}^{\boldsymbol{K}}w_1, \boldsymbol{A}^{\boldsymbol{Q}}w_6\rangle}{\sum_{i=1}^{6}\exp\langle \boldsymbol{A}^{\boldsymbol{K}}w_i, \boldsymbol{A}^{\boldsymbol{Q}}w_6\rangle}$$

- We use two different matrices (called *a key matrix and a query matrix)* before computing the similarities.
- We now have two matrices $\boldsymbol{A}^{\boldsymbol{K}}$ and $\boldsymbol{A}^{\boldsymbol{Q}}$ of "learnable" weights -> twice as tunable as before!

# Tweak: Making Self-Attention Tunable

In the final step, we apply a (third) matrix $\boldsymbol{A}^{\boldsymbol{V}}$ of learnable weights and *then* compute the contextual embedding!

$$\widehat{w}_6 = s_{1,6}\boldsymbol{A}^{\boldsymbol{V}} w_1 + s_{2,6}\boldsymbol{A}^{\boldsymbol{V}} w_2 + s_{3,6}\boldsymbol{A}^{\boldsymbol{V}} w_3 + s_{4,6}\boldsymbol{A}^{\boldsymbol{V}} w_4 + s_{5,6}\boldsymbol{A}^{\boldsymbol{V}} w_5 + s_{6,6}\boldsymbol{A}^{\boldsymbol{V}} w_6$$

instead of

$$\widehat{w}_6 = s_{1,6}w_1 + s_{2,6}w_2 + s_{3,6}w_3 + s_{4,6}w_4 + s_{5,6}w_5 + s_{6,6}w_6$$

# Summary: Making Self-Attention Tunable

$$s_{1,6} = \frac{\exp\langle w_1, w_6 \rangle}{\sum_{i=1}^{6} \exp\langle w_i, w_6 \rangle} \Rightarrow s_{1,6} = \frac{\exp\langle \boldsymbol{A}^{\boldsymbol{K}} w_1, \boldsymbol{A}^{\boldsymbol{Q}} w_6 \rangle}{\sum_{i=1}^{6} \exp\langle \boldsymbol{A}^{\boldsymbol{K}} w_i, \boldsymbol{A}^{\boldsymbol{Q}} w_6 \rangle}$$

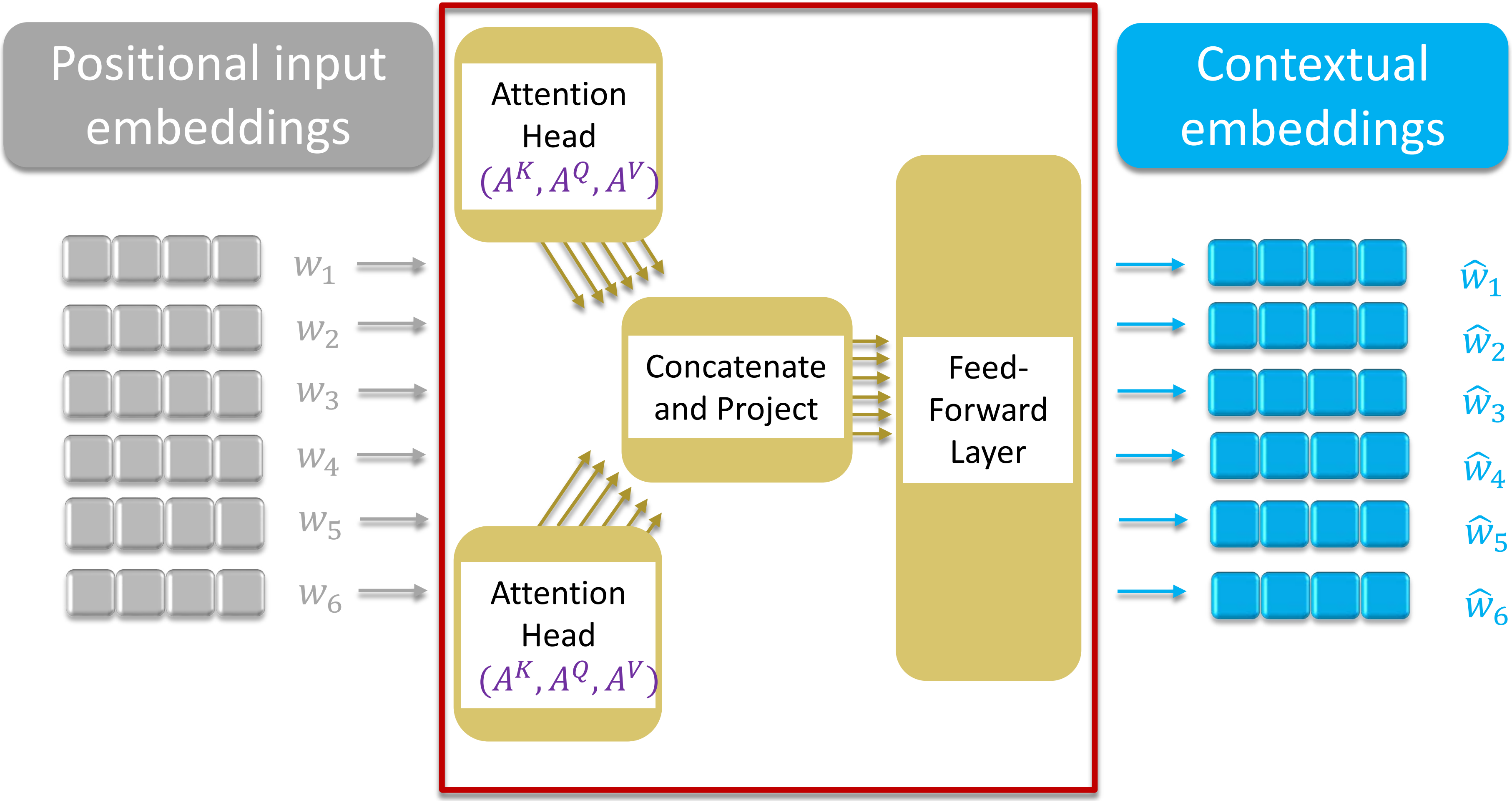
$$\boxed{\widehat{w}_6 = s_{1,6} \boldsymbol{A}^{\boldsymbol{V}} w_1 + s_{2,6} \boldsymbol{A}^{\boldsymbol{V}} w_2 + s_{3,6} \boldsymbol{A}^{\boldsymbol{V}} w_3 + s_{4,6} \boldsymbol{A}^{\boldsymbol{V}} w_4 + s_{5,6} \boldsymbol{A}^{\boldsymbol{V}} w_5 + s_{6,6} \boldsymbol{A}^{\boldsymbol{V}} w_6}$$

The values in matrices $\boldsymbol{A}^{\boldsymbol{K}}$, $\boldsymbol{A}^{\boldsymbol{Q}}$, $\boldsymbol{A}^{\boldsymbol{V}}$ are weights learned through optimization (SGD). This makes them "tunable" and (as we will see shortly) enable the attention "heads" to learn different patterns in the input

This entire operation can be written compactly in this matrix equation (https://arxiv.org/pdf/1706.03762.pdf):

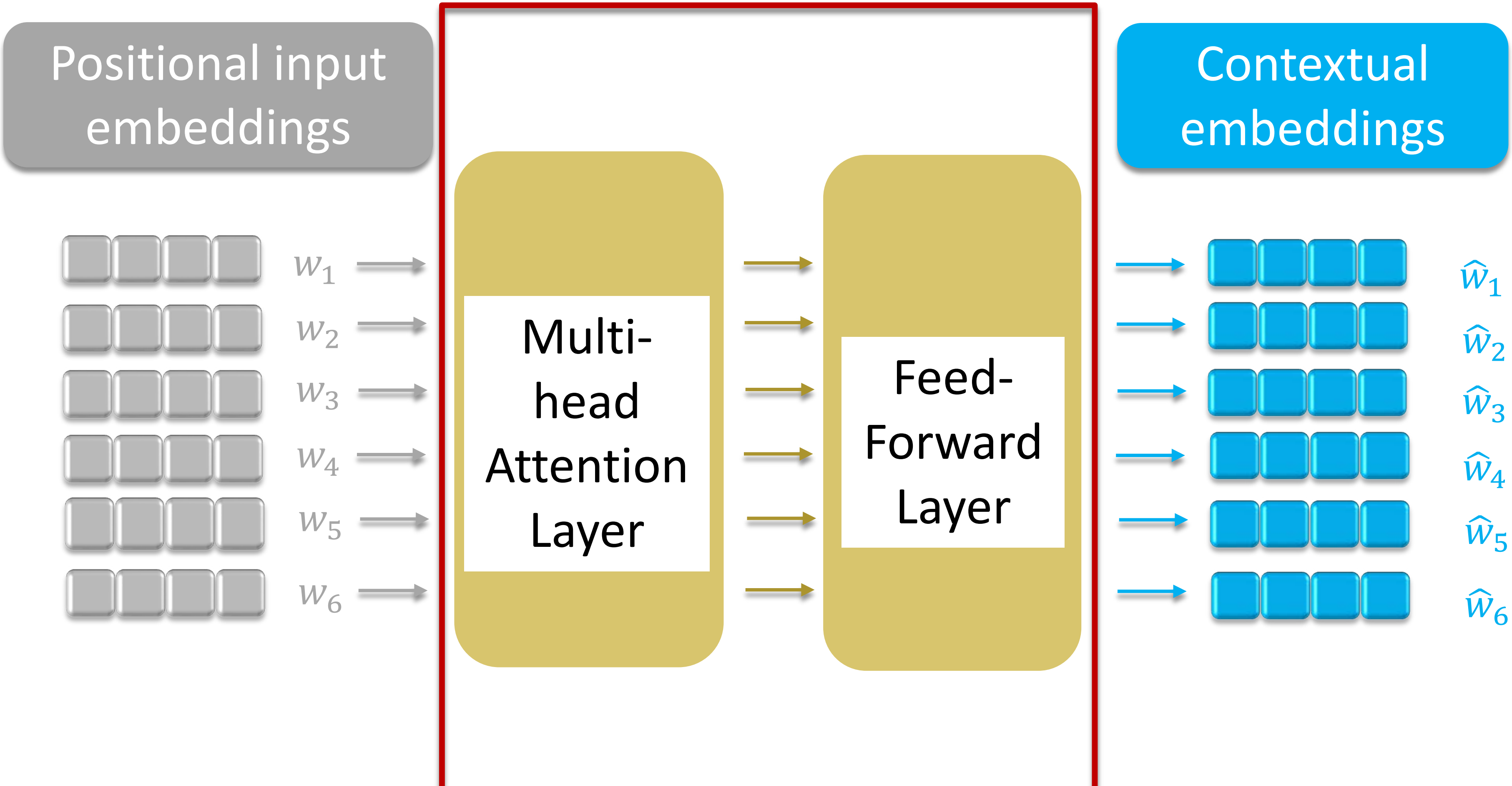
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

# Summary: Multi-Head Attention



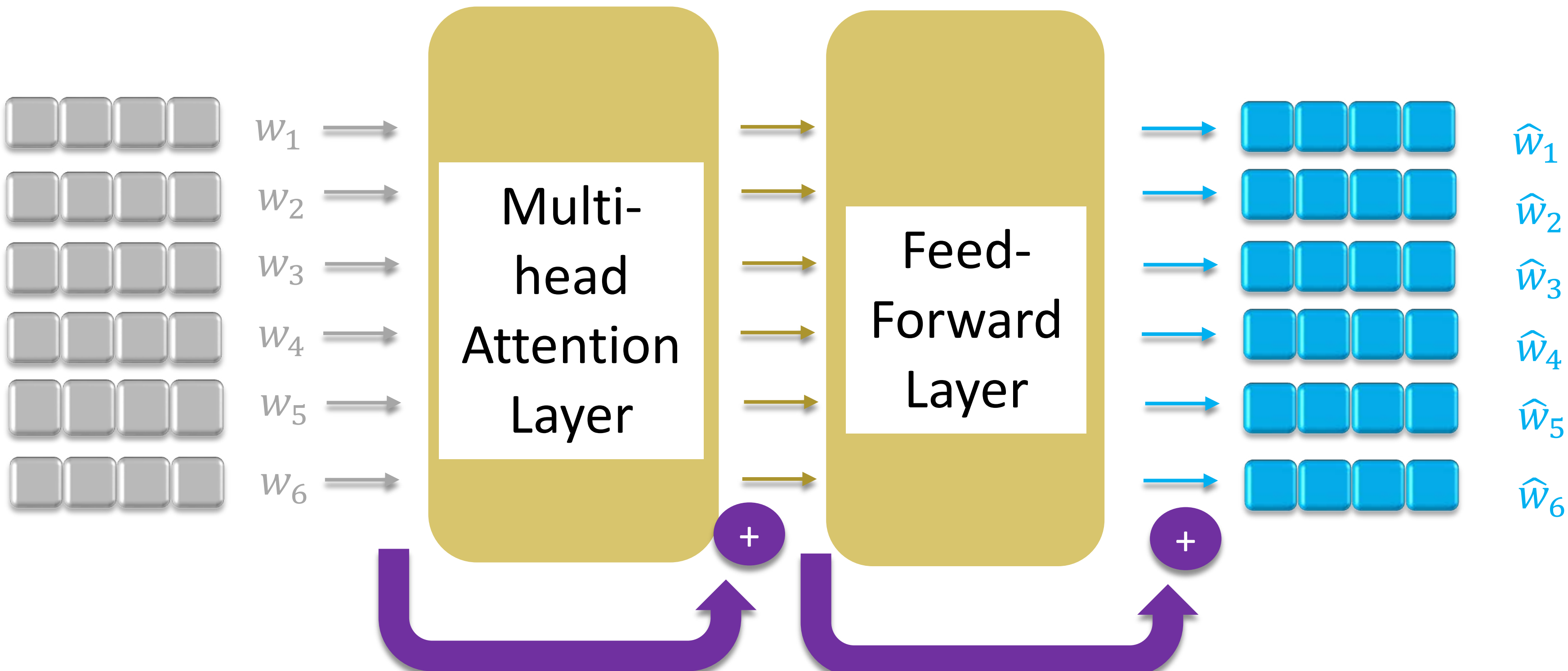

*Important: Each attention head will have its <u>own</u> $A^K, A^Q, A^V$

# Transformer Encoder



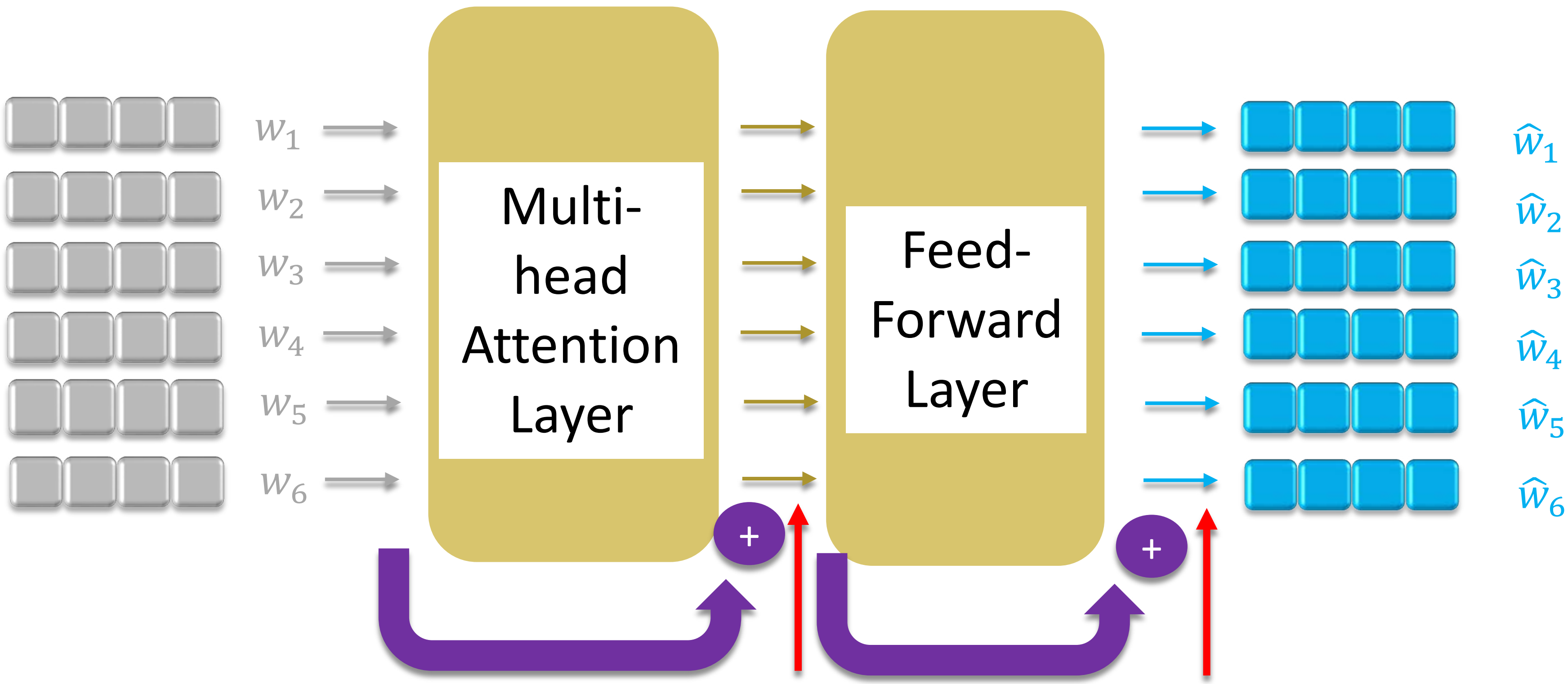
# Another tweak: Residual Connection

We sum the input embedding to the output embedding of the Attention / Feed-Forward Layers. This helps gradients flow better during backpropagation.

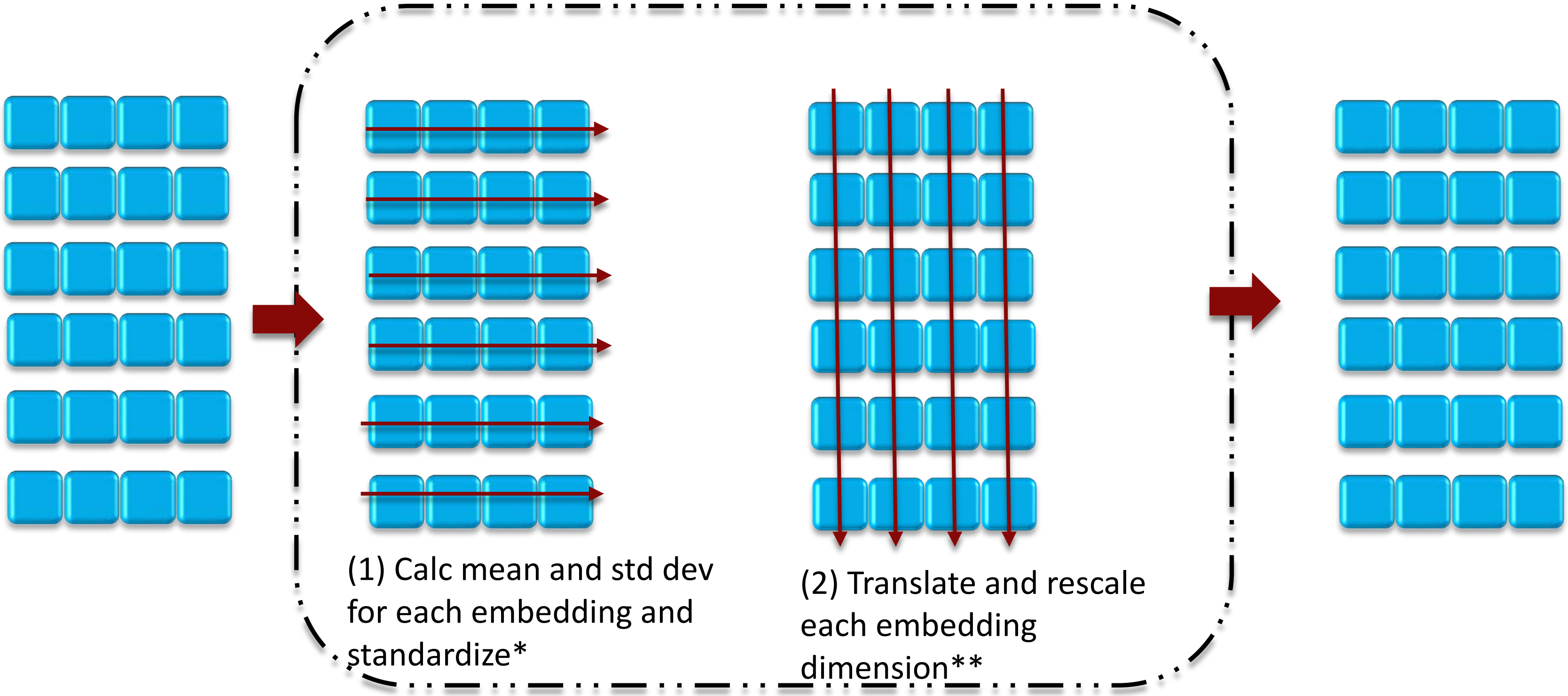


# A final tweak: Layer Normalization

After the Attention / Feed-Forward Layers, we standardize (i.e., subtract mean and divide by std) each embedding. This ensures that the weights stay small.



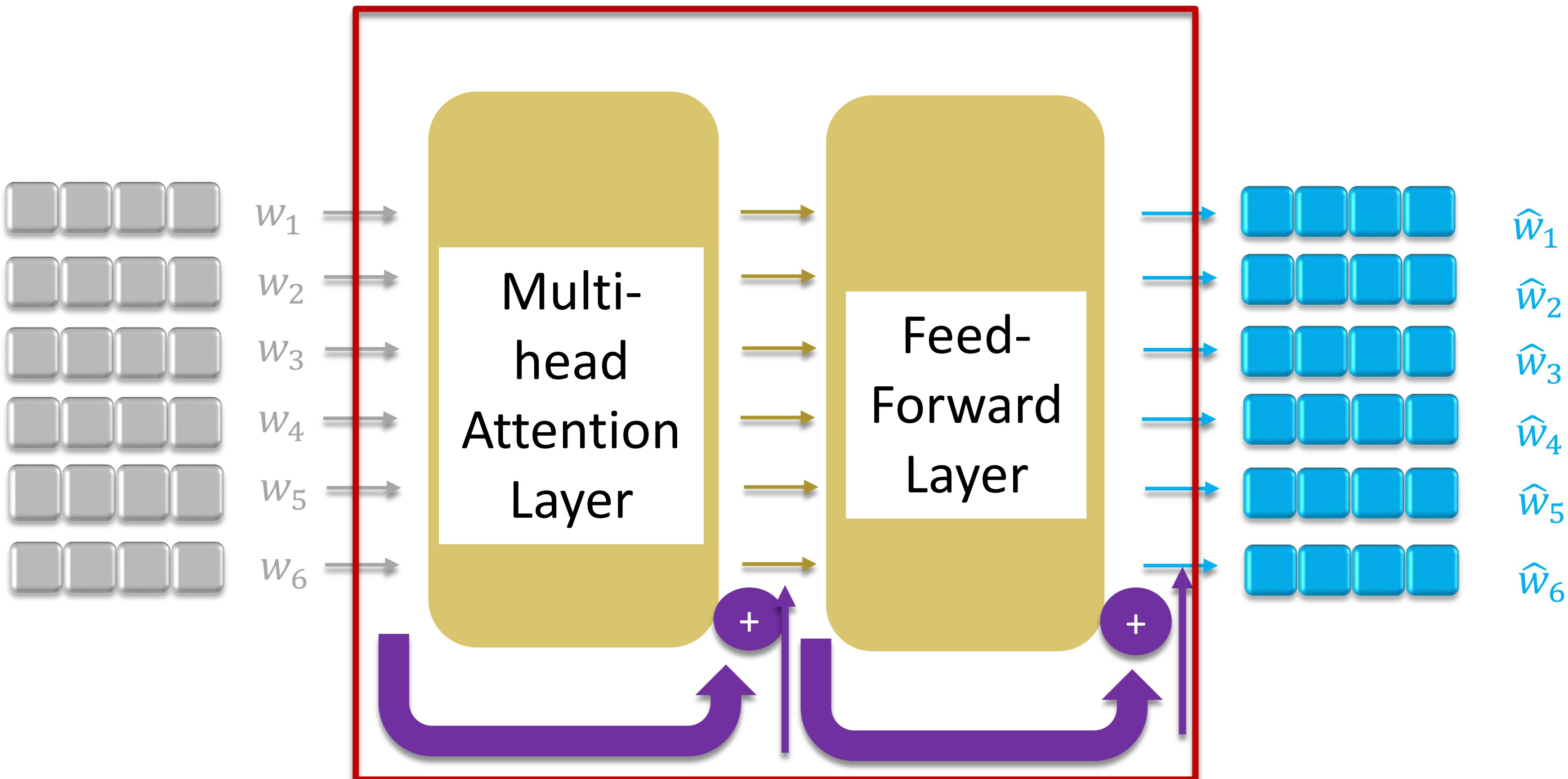
# Layer Normalization



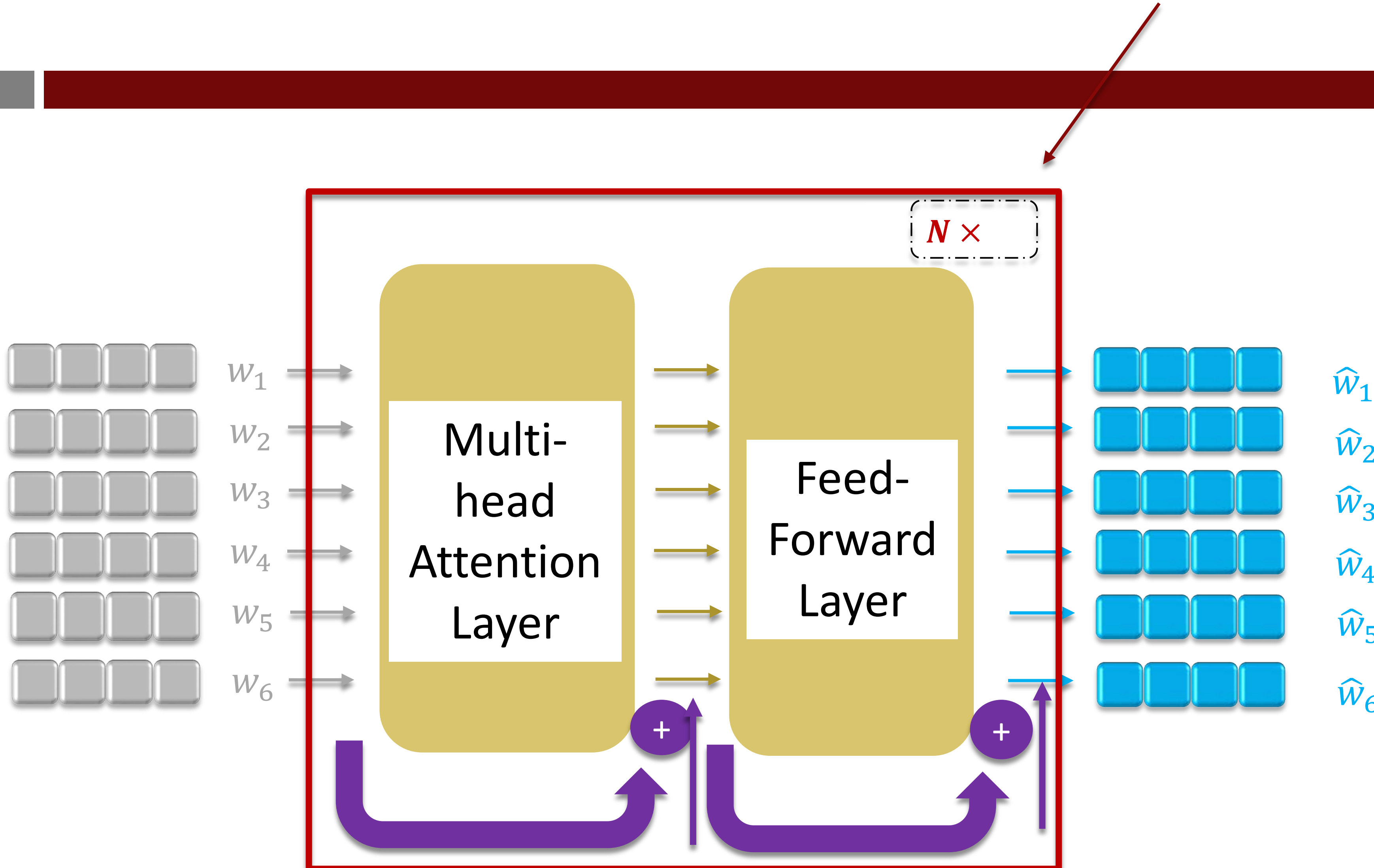

Layer Normalization

*subtract mean and divide by standard deviation
** see https://keras.io/api/layers/normalization_layers/layer_normalization/ for details

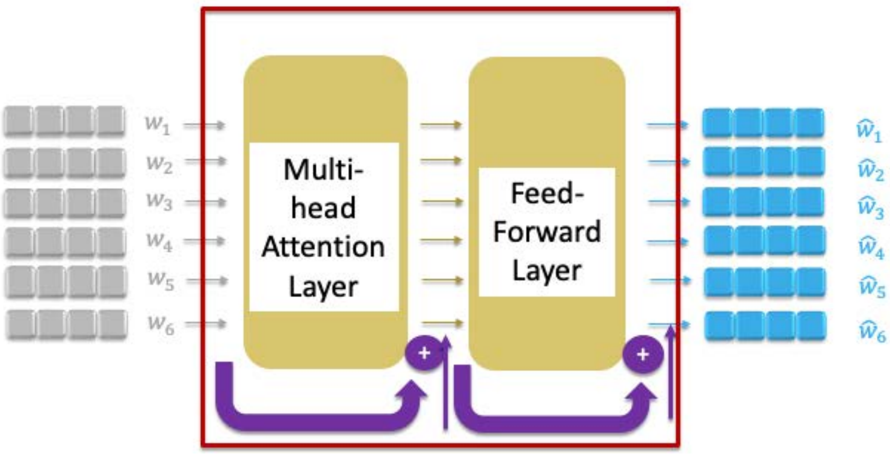
# Transformer Encoder

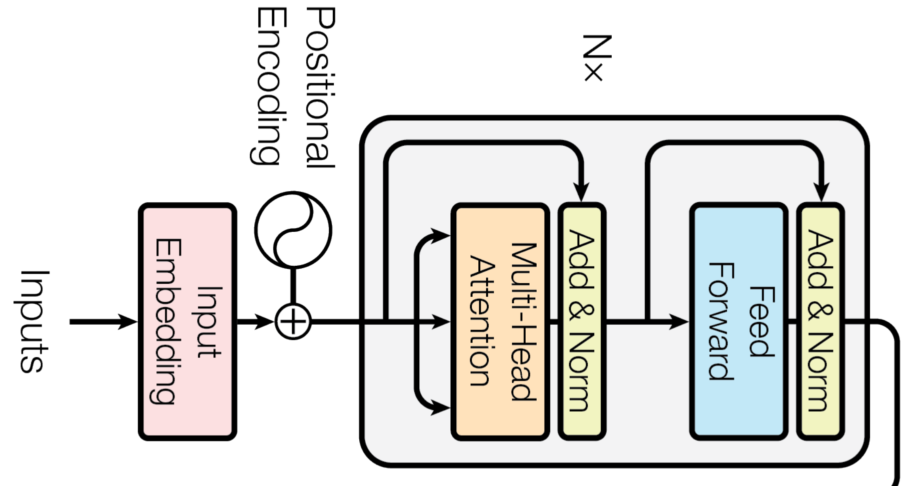


# Transformer Encoders are stackable!



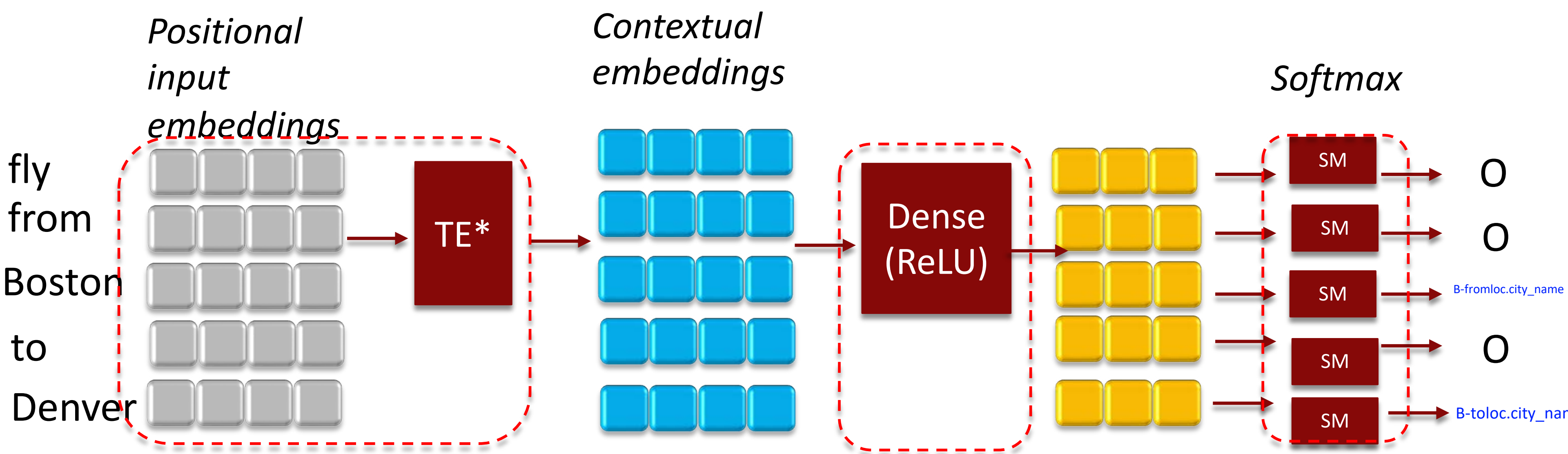
# The Transformer Encoder




https://arxiv.org/abs/1706.03762

# Review: What is Optimized?




*Weights optimized by backprop*

1. *Positional embeddings*
2. *Stand-alone embeddings (unless pretrained and Trainable=False)*
3. *Matrices $A^K, A^Q, A^V$ for each attention head (inside TE)*
4. *Layer norm scale and bias parameters (inside TE)*
5. *Weights in Feed-Forward layers (inside TE)*
6. *Weights in Dense layers outside TE*
7. *Weights in final Softmax layer*

*Transformer Encoder

Indicate 4-element embedding vectors

# Applying the Transformer – Common Use-Cases

## Sequence classification

"I loved the movie" → Transformer Encoder → Positive

## Sequence labeling

"fly from Boston to Denver" → Transformer Encoder →

| Token | Label |
|---|---|
| fly | O |
| from | O |
| Boston | B-fromloc.city_name |
| to | O |
| Denver | B-toloc.city_name |

## Sequence generation

"I loved the movie" → Transformer Causal Encoder* → "I loved the movie, especially the cinematography and the background score"

*covered in Lecture 9

# We saw how to do Sequence Labeling

## Sequence classification

"I loved the movie" → Transformer Encoder → Positive

## Sequence labeling

"fly from Boston to Denver" → Transformer Encoder →

| Token | Label |
|---|---|
| fly | O |
| from | O |
| Boston | B-fromloc.city_name |
| to | O |
| Denver | B-toloc.city_name |

## Sequence generation

"I loved the movie" → Transformer Causal Encoder → "I loved the movie, especially the cinematography and the background score"

# How can we do this?

## Sequence classification

"I loved the movie" → Transformer Encoder → Positive

## Sequence labeling

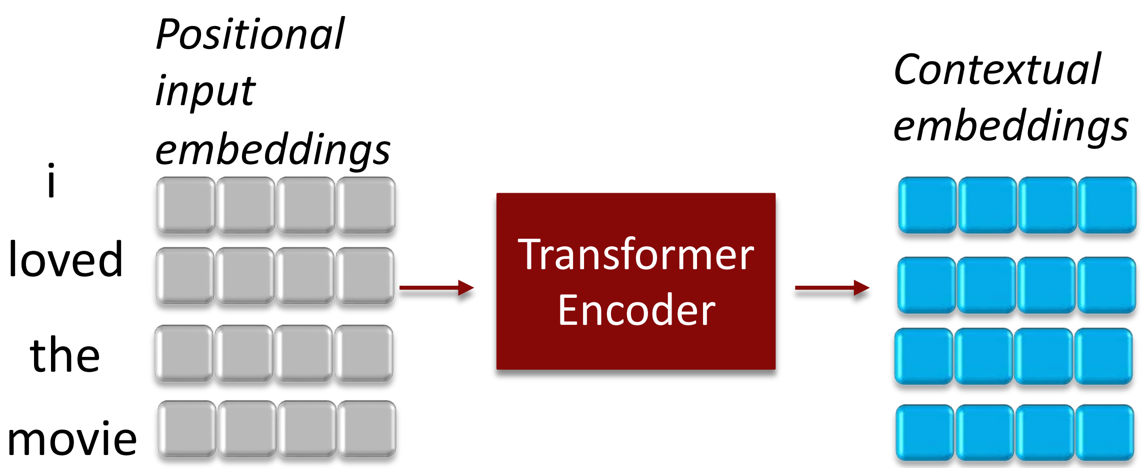
"fly from Boston to Denver" → Transformer Encoder →

| Token | Label |
|---|---|
| fly | O |
| from | O |
| Boston | B-fromloc.city_name |
| to | O |
| Denver | B-toloc.city_name |

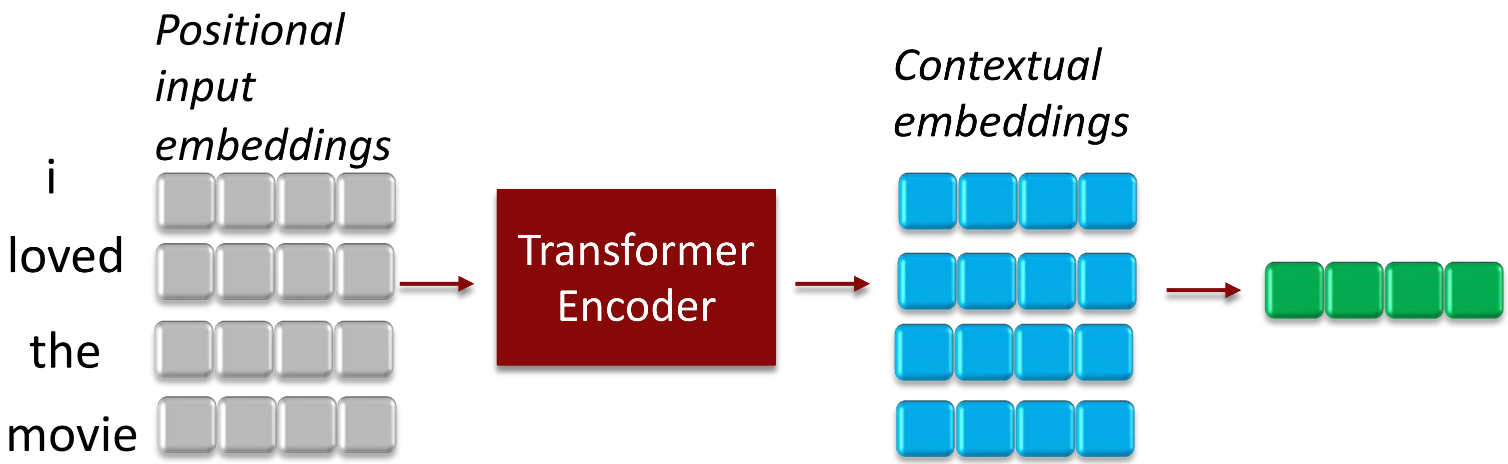
## Sequence generation

"I loved the movie" → Transformer Causal Encoder → "I loved the movie, especially the cinematography and the background score"

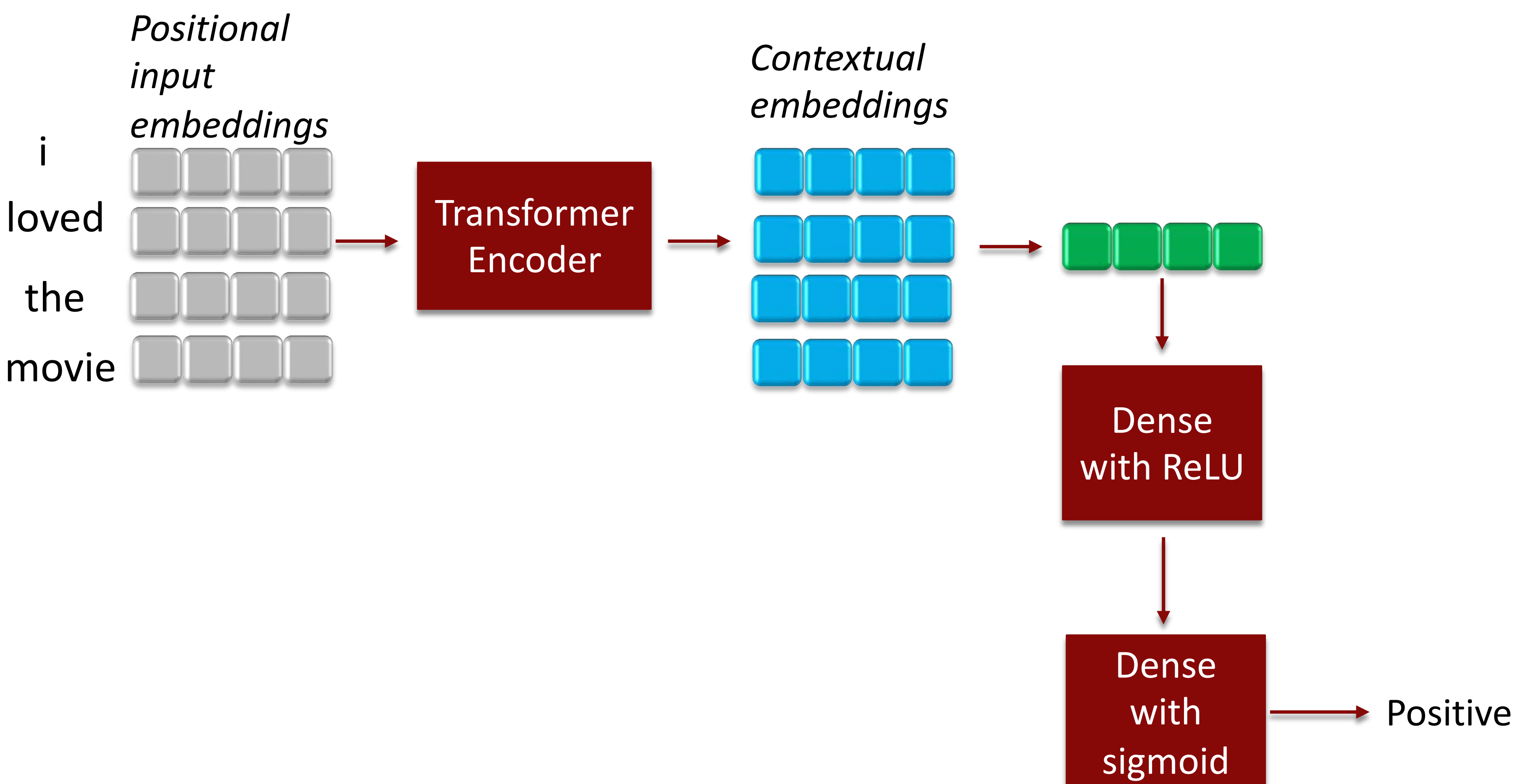
# Recall: The Transformer Encoder produces a contextual embedding for each token in the input



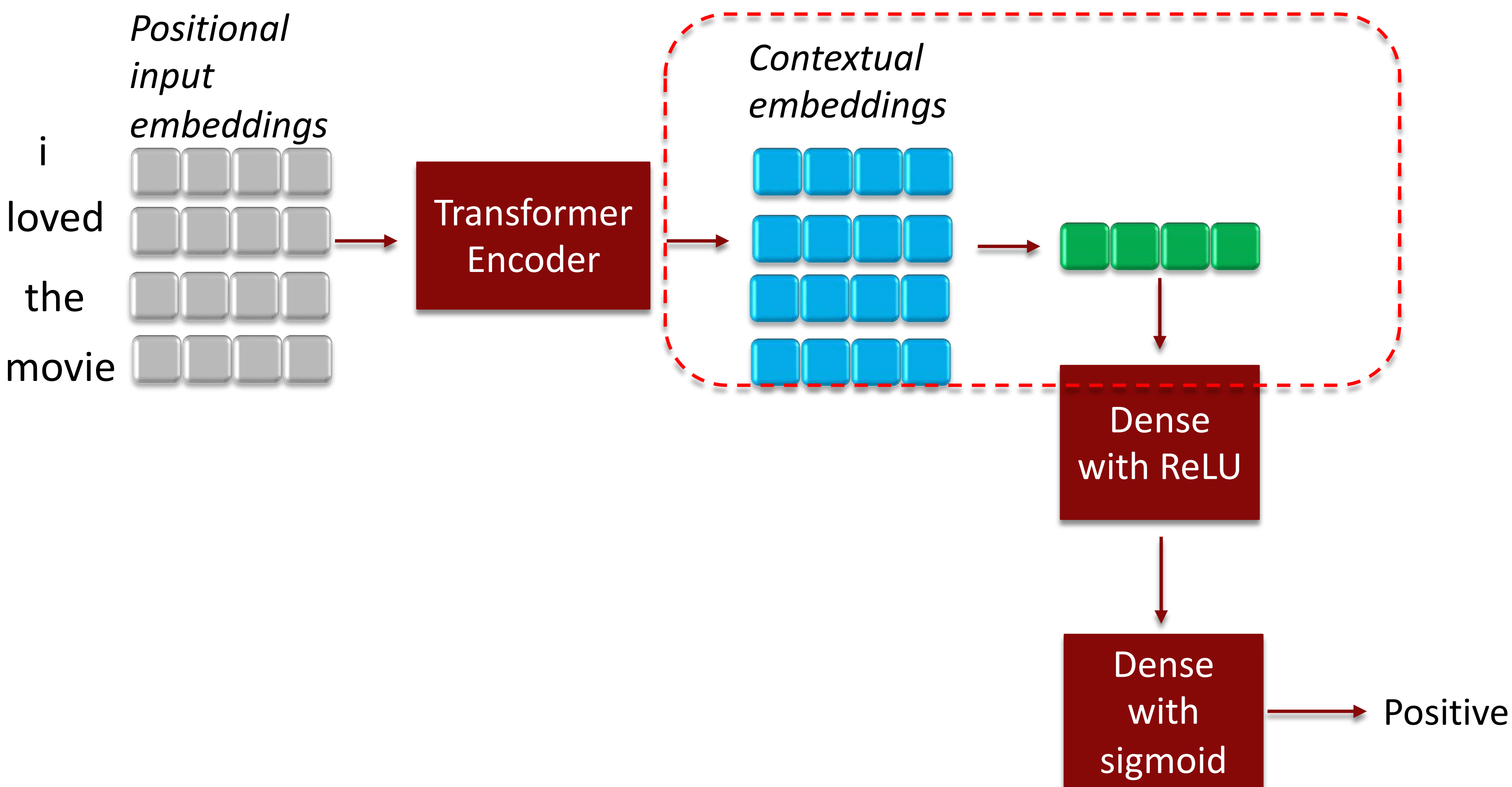
# If we could "summarize" the multiple contextual embeddings into a single embedding that represents the whole sentence …



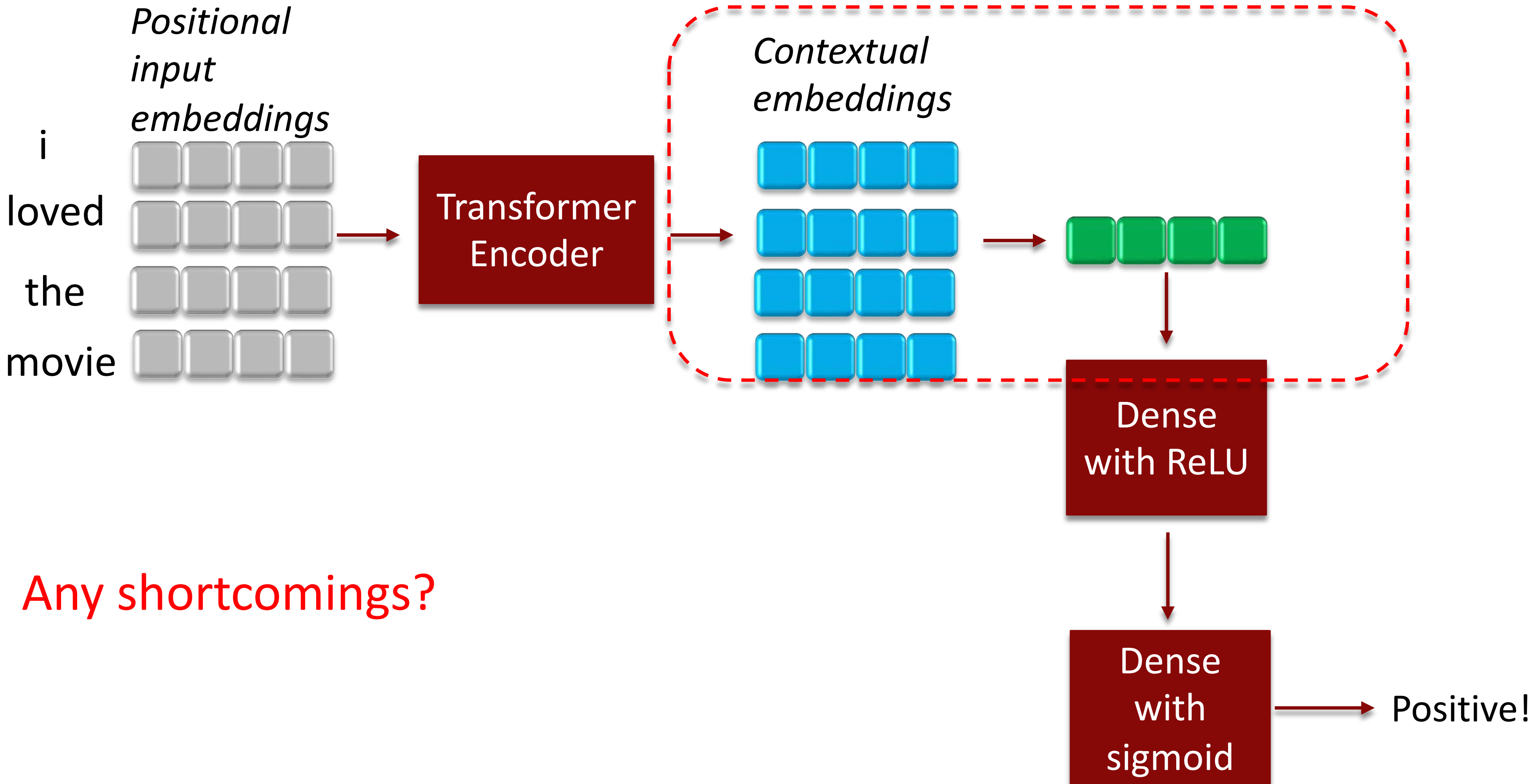
# … we can feed the sentence embedding into a dense ReLU layer, followed by a sigmoid (or softmax)



# How can we do this?

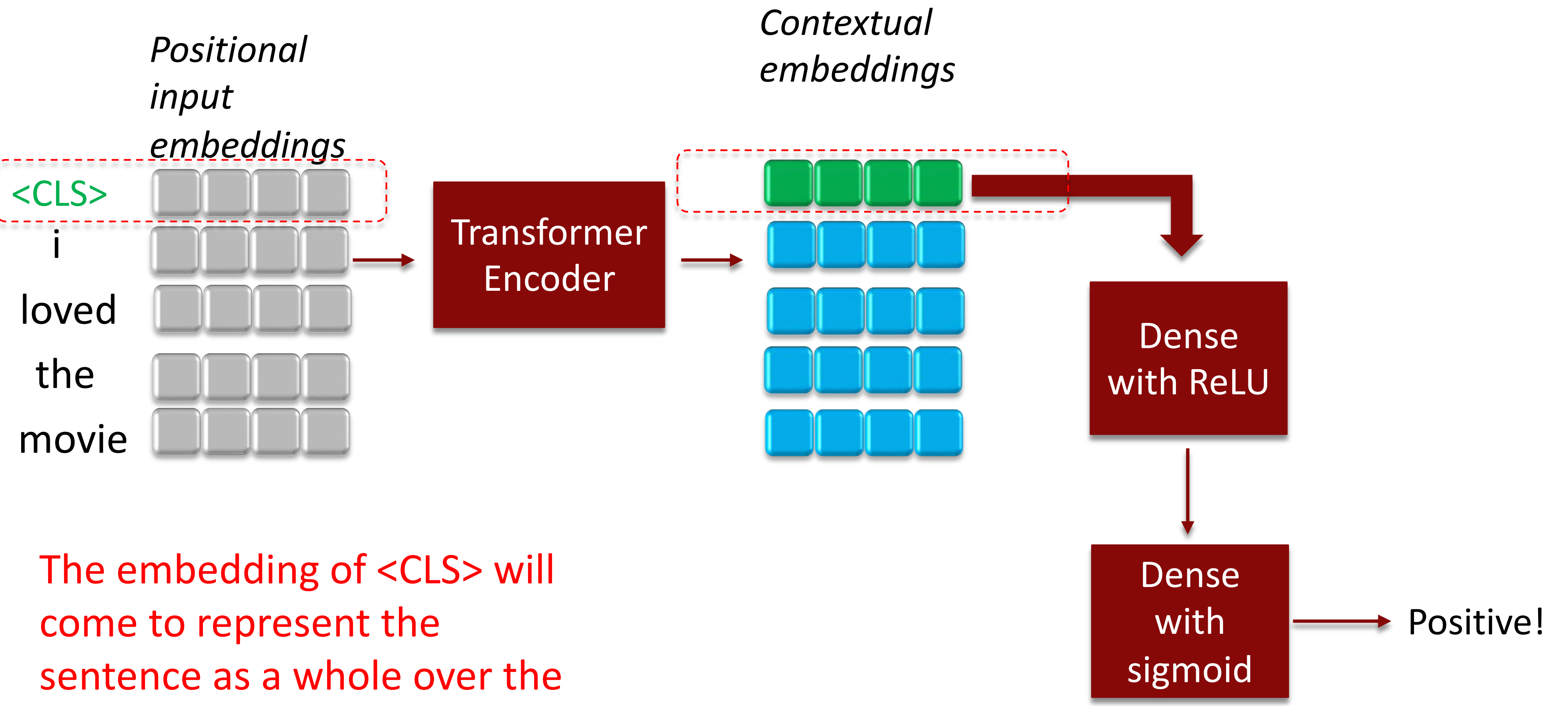


# We *can* average the four  to get 




Any shortcomings?

# A better approach: Add a special token at the beginning of each sentence and just use its output embedding as the sentence-embedding




The embedding of <CLS> will come to represent the sentence as a whole over the course of training

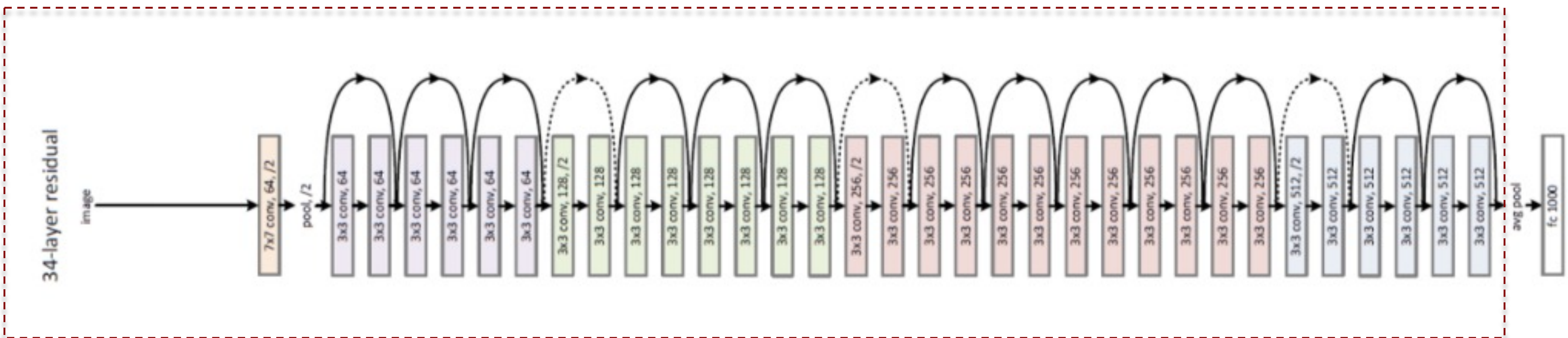
If the input data for a task is natural language text, we don’t have to restrict ourselves to just the text we have.

Wouldn’t it be great to learn from “all the text that’s out there”?

# Self-Supervised Learning

# Recall the Transfer Learning example from Lecture 4

## ResNet 34*

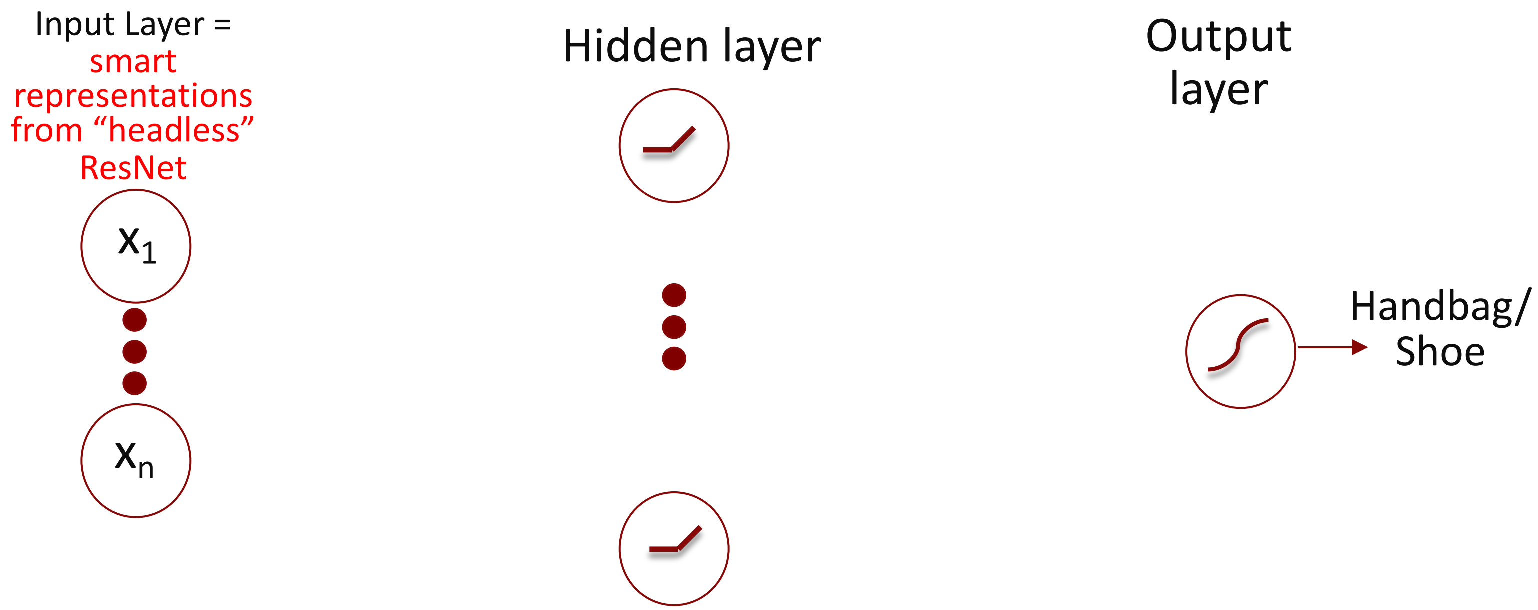



We fed the output of "headless Resnet" to a small NN

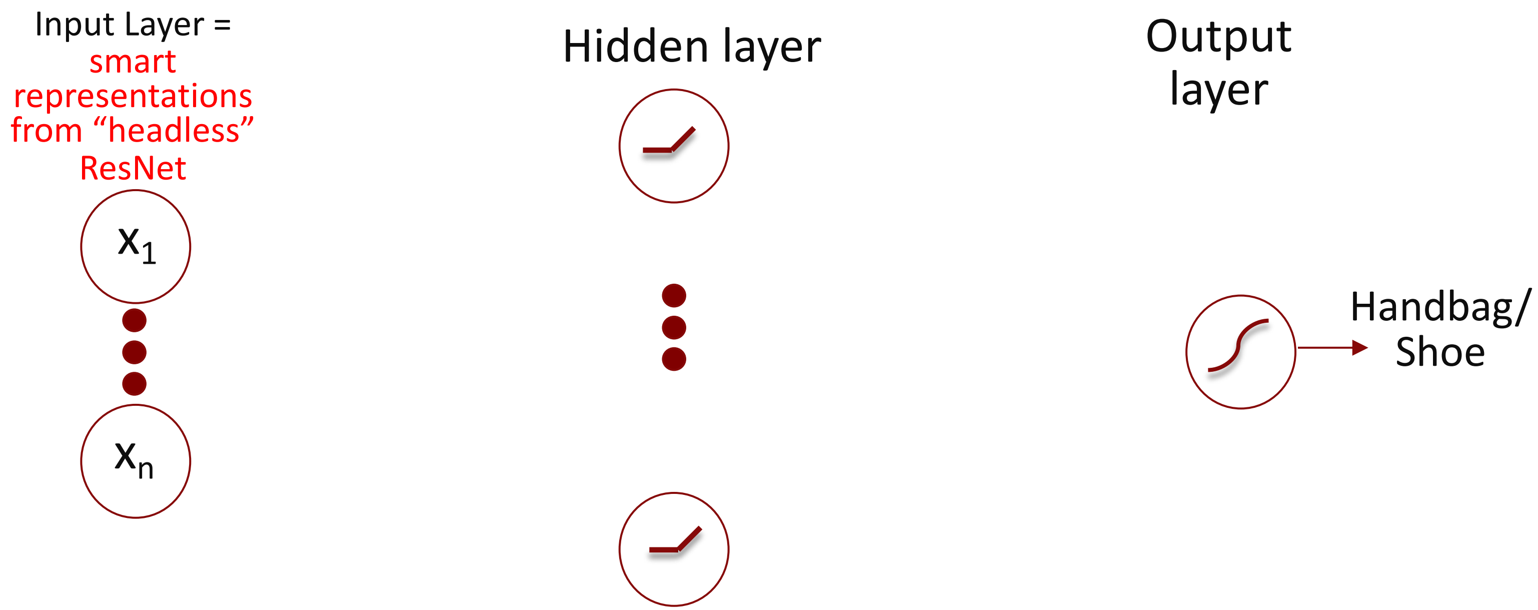


*https://arxiv.org/abs/1512.03385

# We built a very accurate handbags/shoes classifier with only 100 examples



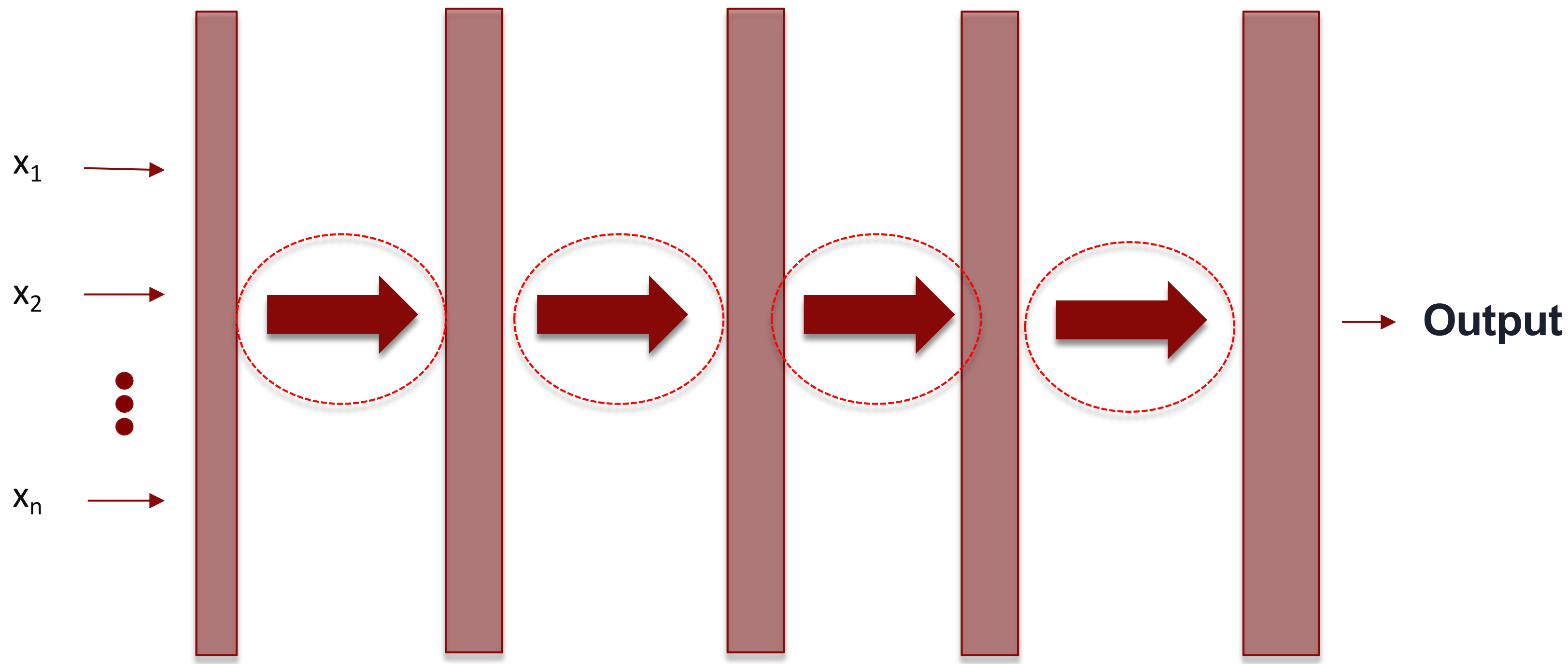
# We built a very accurate handbags/shoes classifier with only 100 examples



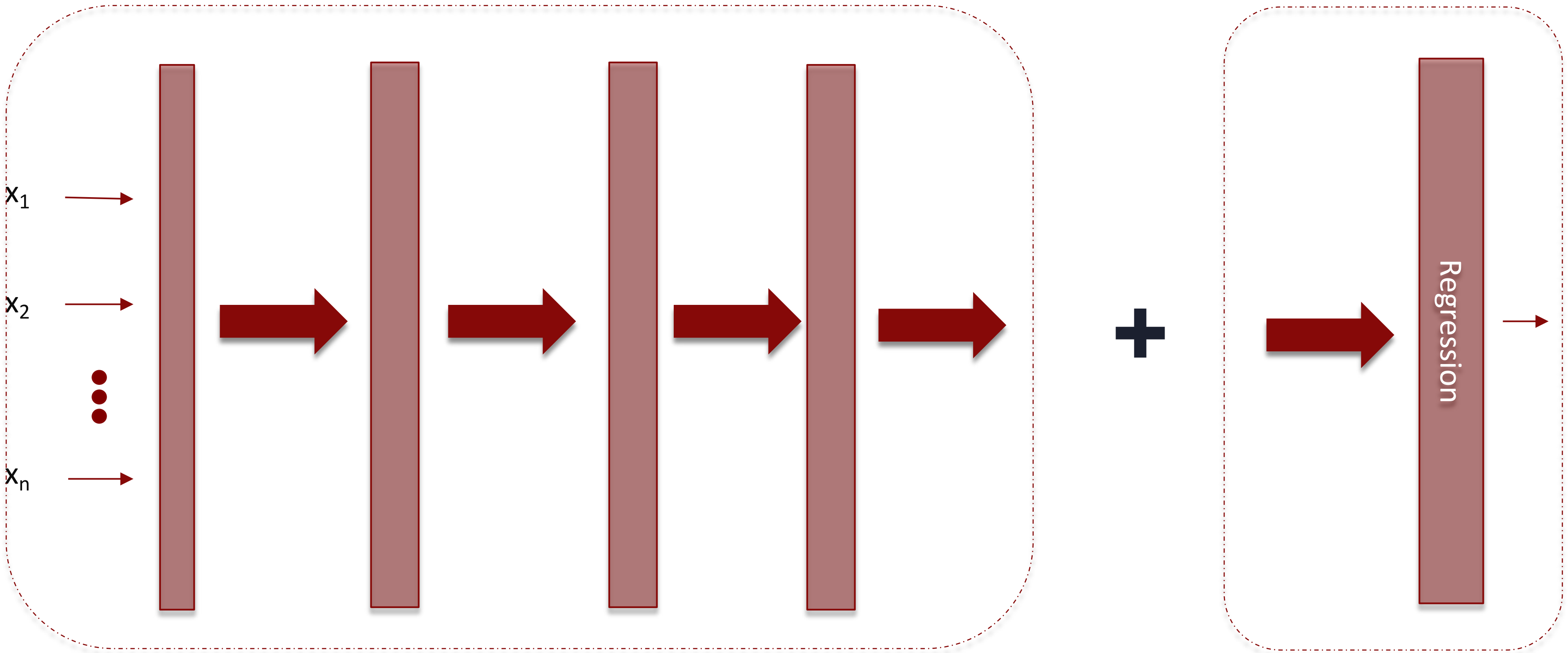

Why was this so effective?

# Neural networks are Representation learners

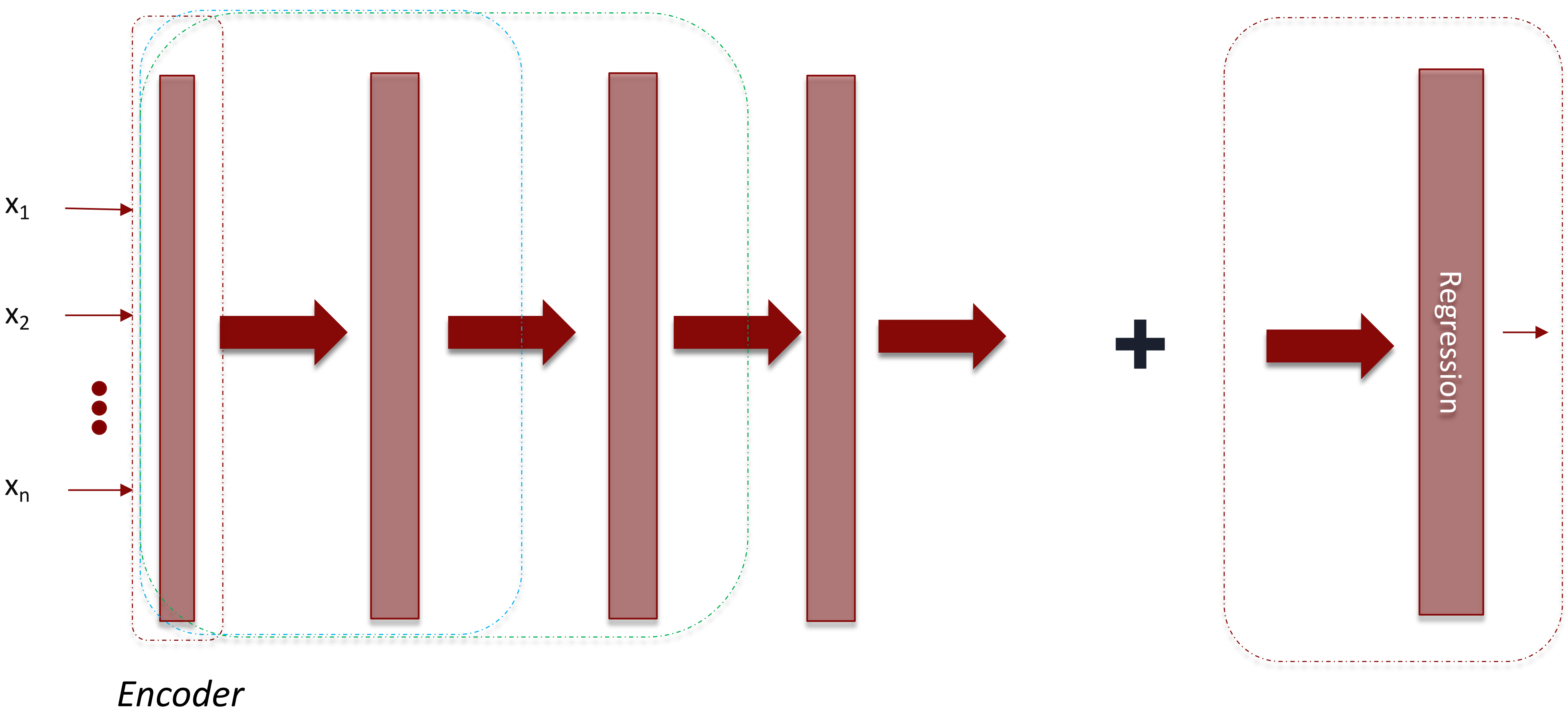
# The output of every layer in a DNN can be thought of as a transformed version of the ”raw” input. These transformed versions of the input are called representations



# From this perspective, a deep NN trained with Supervised Learning learns many representations <u>and</u> a final regression model



If we think of a representation as an encoding of the raw input, the part of the NN that produces that encoding can be viewed as an encoder. A DNN “contains” many encoders.



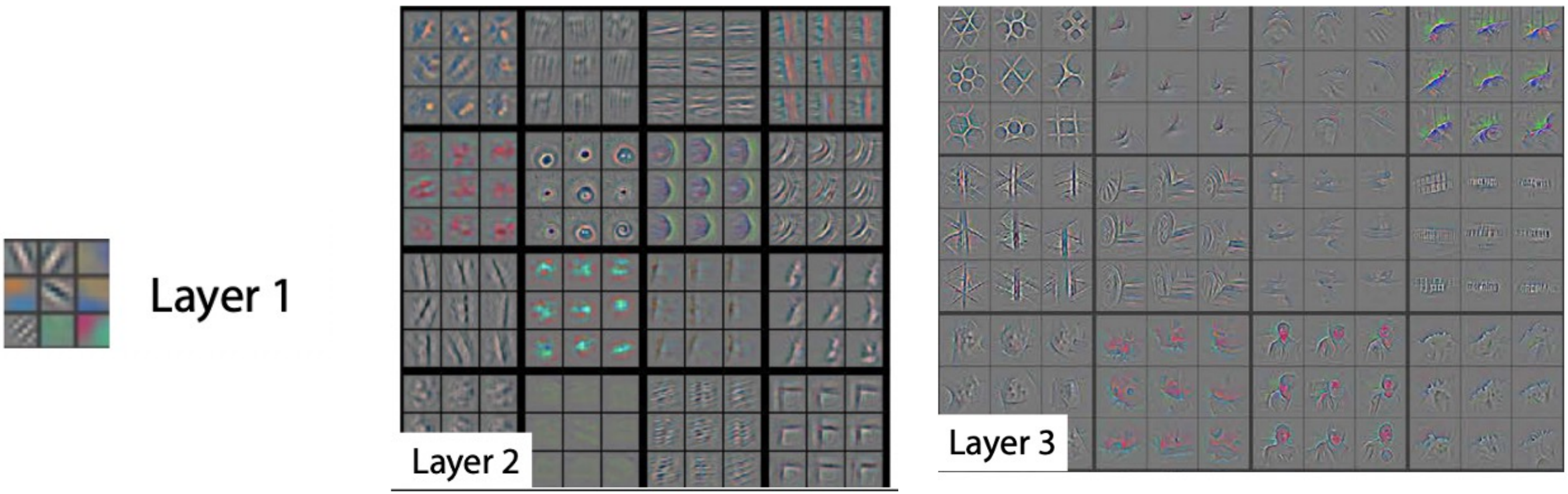
# What do representations/encoders capture?

- Is it specific knowledge needed to connect the input to the *particular* output the NN was trained to predict?

- Or is it general knowledge about the input data that can be useful to predict other outputs?

# Turns out representations *do* capture a lot of *general* knowledge about the input data

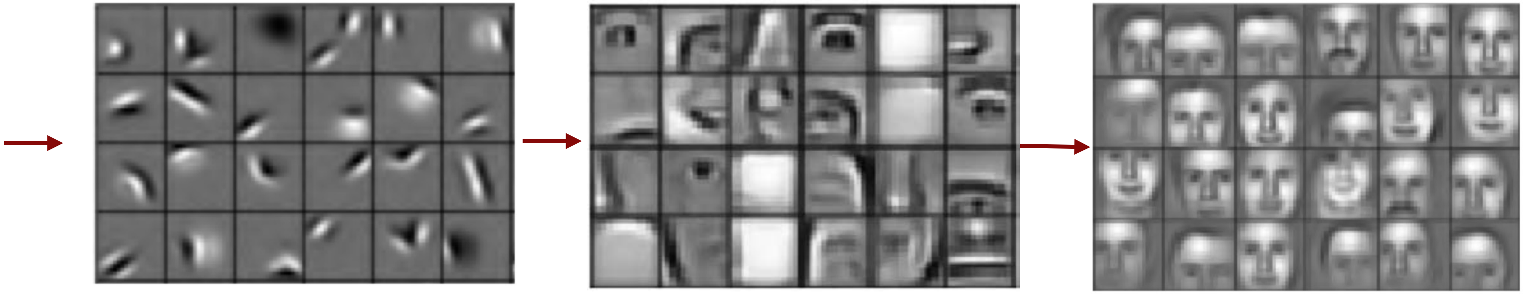
# In a deep network trained to classify “everyday” objects into one of 1000 categories, the representations from the first three layers correspond to lines, then edges, then more complex shapes






https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

# In a deep network trained to detect faces, the representations correspond to lines, edges, circles and finally faces



lines => edges, circles => faces!



Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations by Lee at al (2009)

# Leveraging the general knowledge in these representations

- Since these representations are capturing various intrinsic aspects of the images, they could be used for prediction tasks other than the ones they were initially trained for.

# Leveraging the general knowledge in these representations

- Since these representations are capturing various <u>intrinsic</u> aspects of the images, they could be used for prediction tasks other than the ones they were initially trained for.
  - For example, the representations from the face-detection DNN could be plausibly used to build an emotion-detection DNN.

# Leveraging the general knowledge in these representations

- Since these representations are capturing various <u>intrinsic</u> aspects of the images, they could be used for prediction tasks other than the ones they were initially trained for.
  - For example, the representations from the face-detection DNN could be plausibly used to build an emotion-detection DNN.

- If we can "somehow" get an encoder that generates good representations of our input data, we can simply build a regression model with the representations as input and labels as output

# Leveraging the general knowledge in these representations

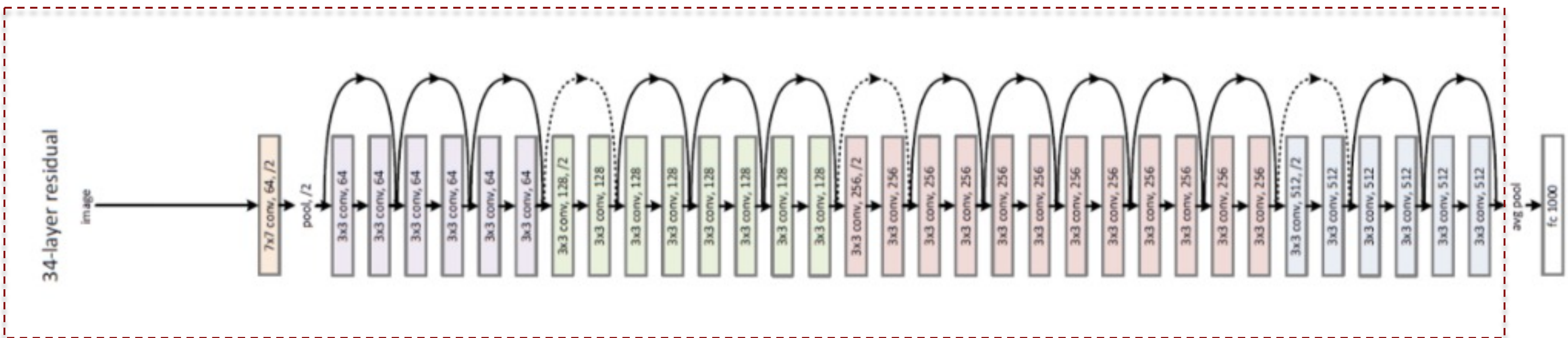
- Since these representations are capturing various intrinsic aspects of the images, they could be used for prediction tasks other than the ones they were initially trained for.
  - For example, the representations from the face-detection DNN could be plausibly used to build an emotion-detection DNN.

- If we can "somehow" get an encoder that generates good representations of our input data, we can simply build a regression model with the representations as input and labels as output

- Since we won't have to "spend" precious data on learning good representations any more, we won't need as much labeled data in the first place.

# This is exactly what happened with the handbags-shoes example

ResNet 34*

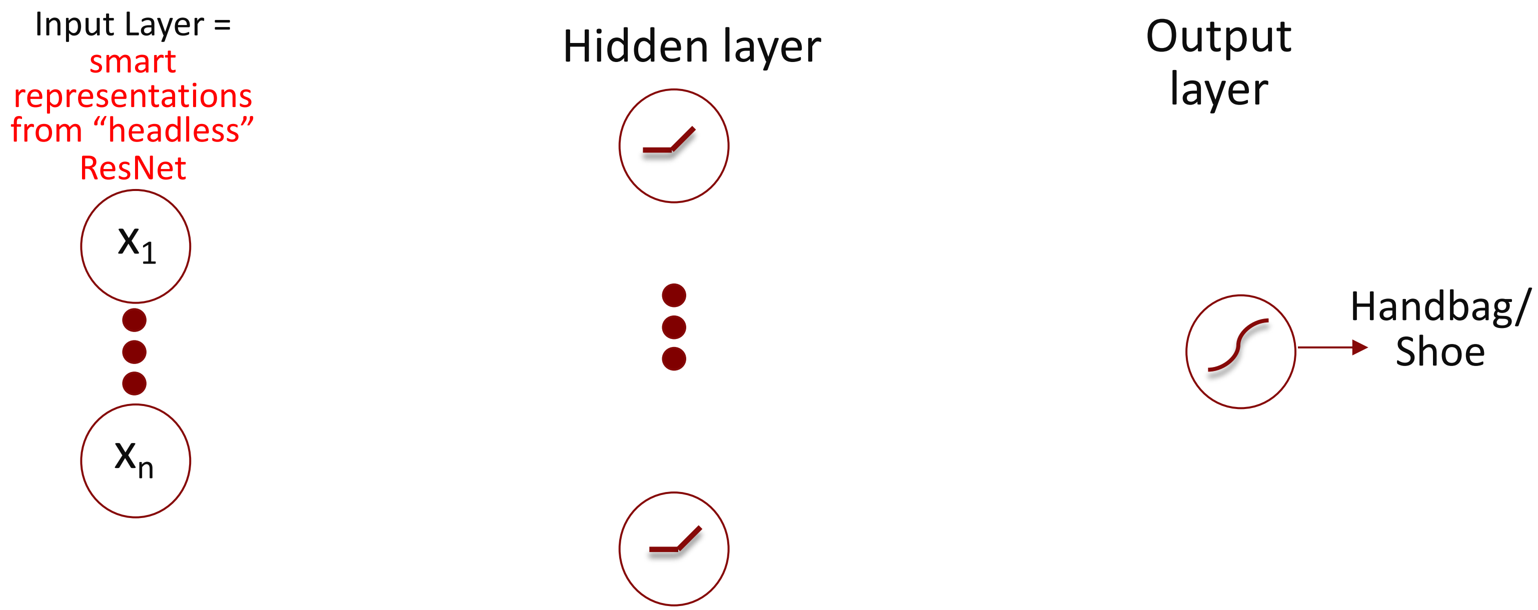



We used "headless Resnet" as an <u>encoder</u> that can take raw input and transform it into useful representations



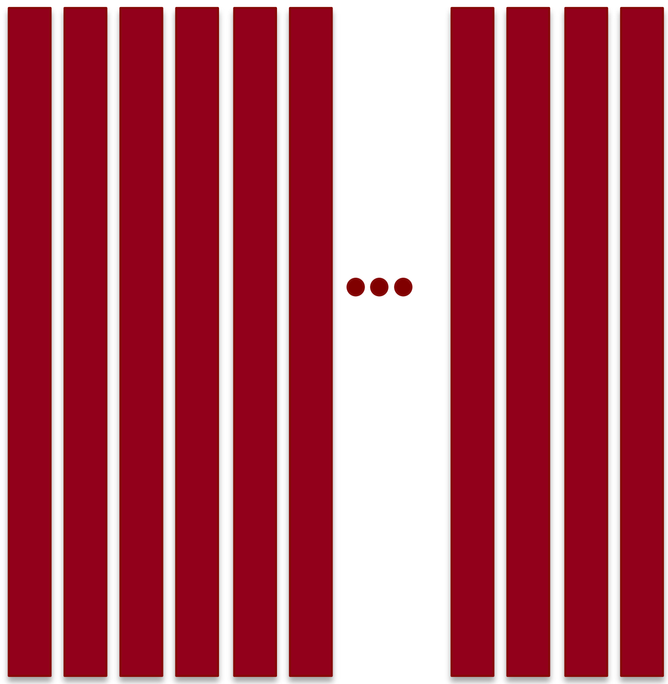
*https://arxiv.org/abs/1512.03385

# By using these smart representations, we could build a very accurate handbags/shoes classifier with only 100 examples



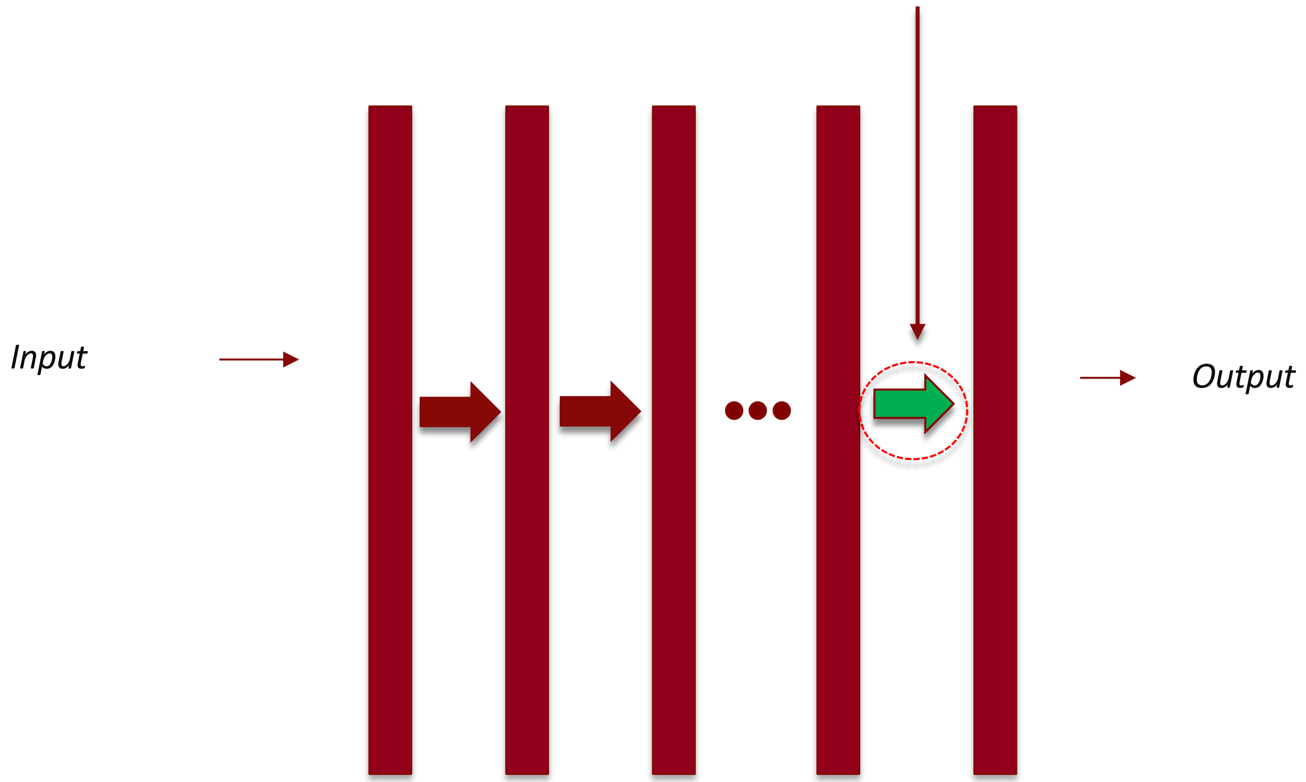
# The general approach is to find a deep NN built on similar inputs but different outputs

**Same inputs as our problem …**

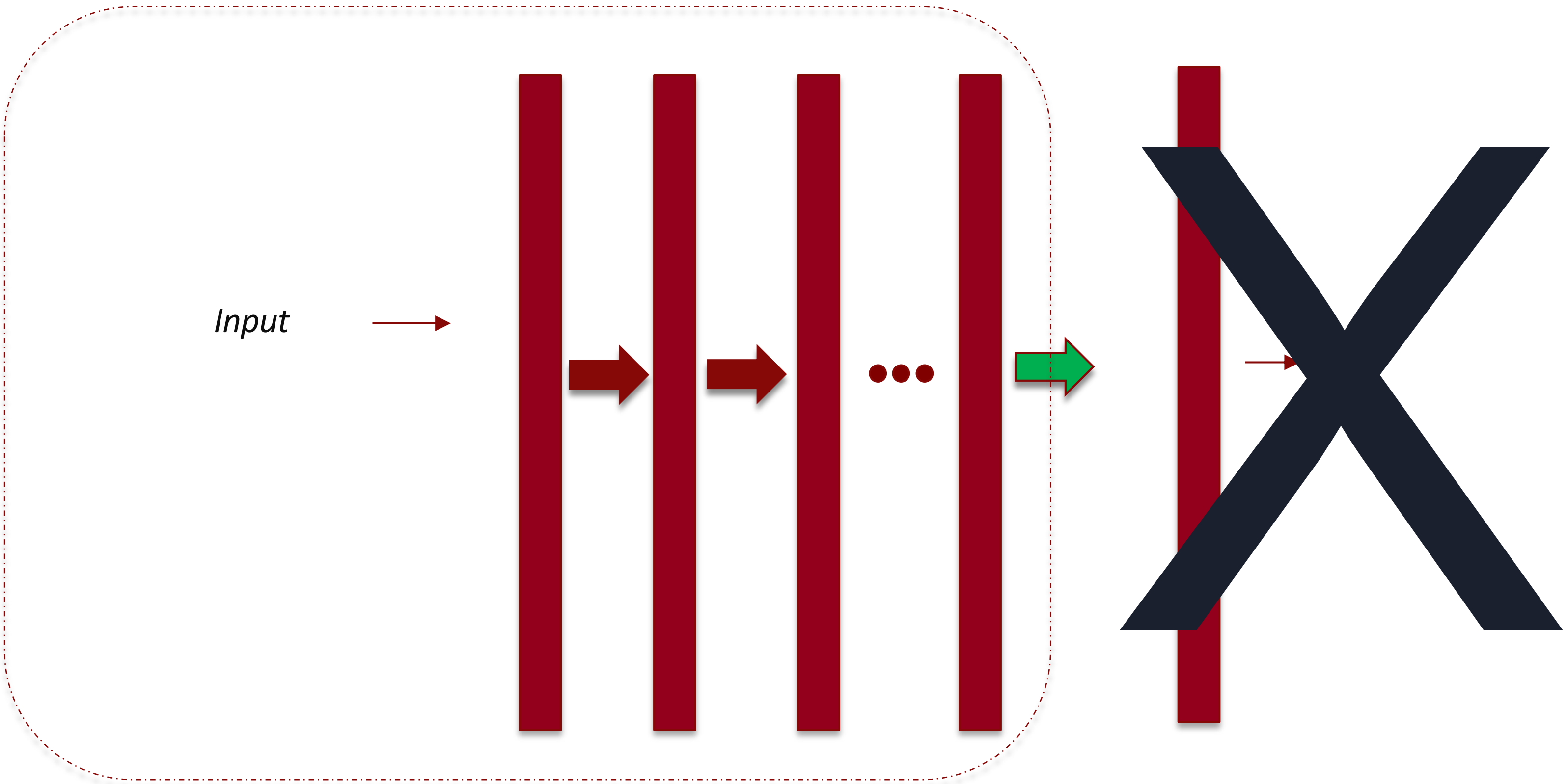


**… but different outputs**

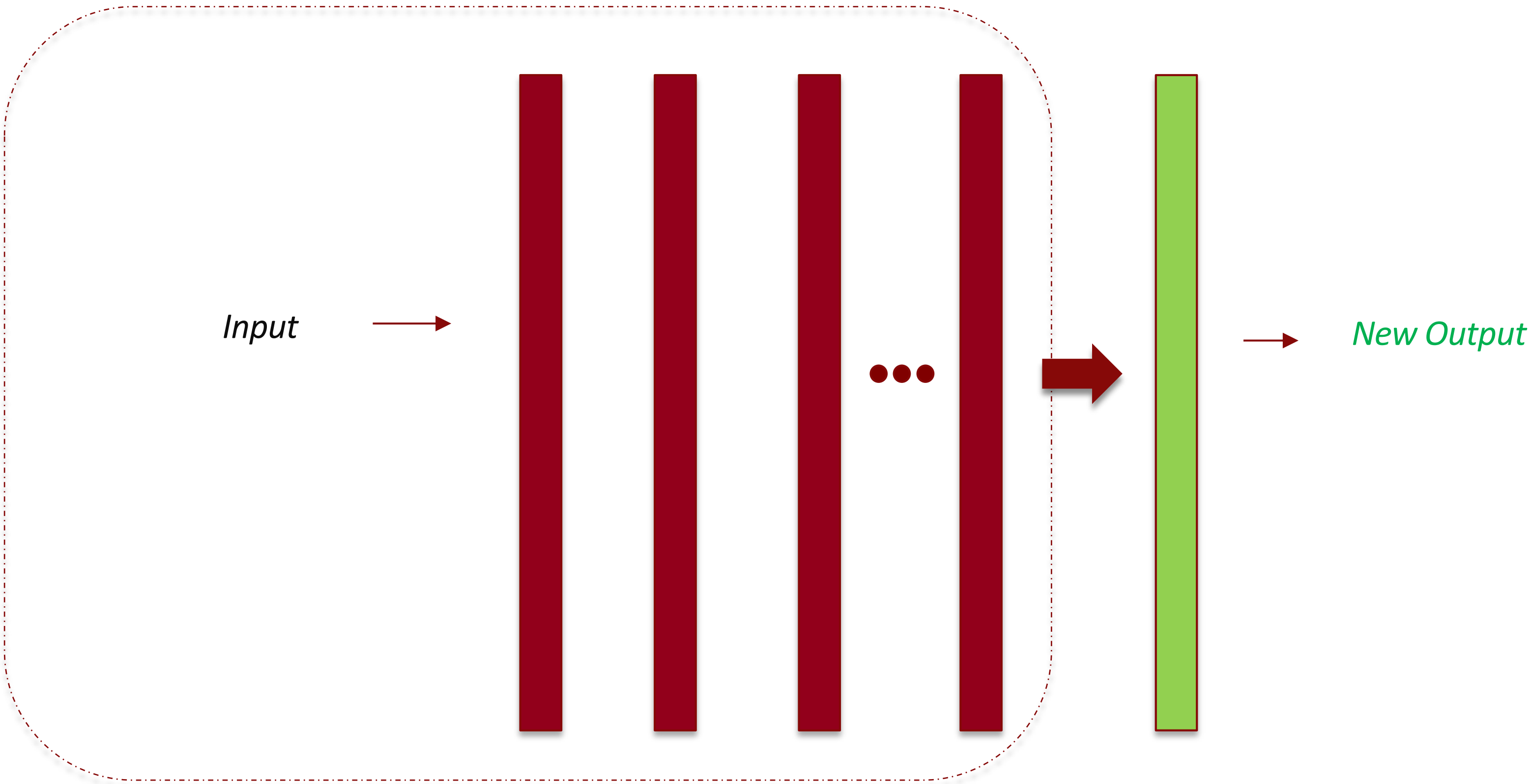
What comes out of the last layer (before the output layer) of this deep NN is likely to be an excellent representation of the input



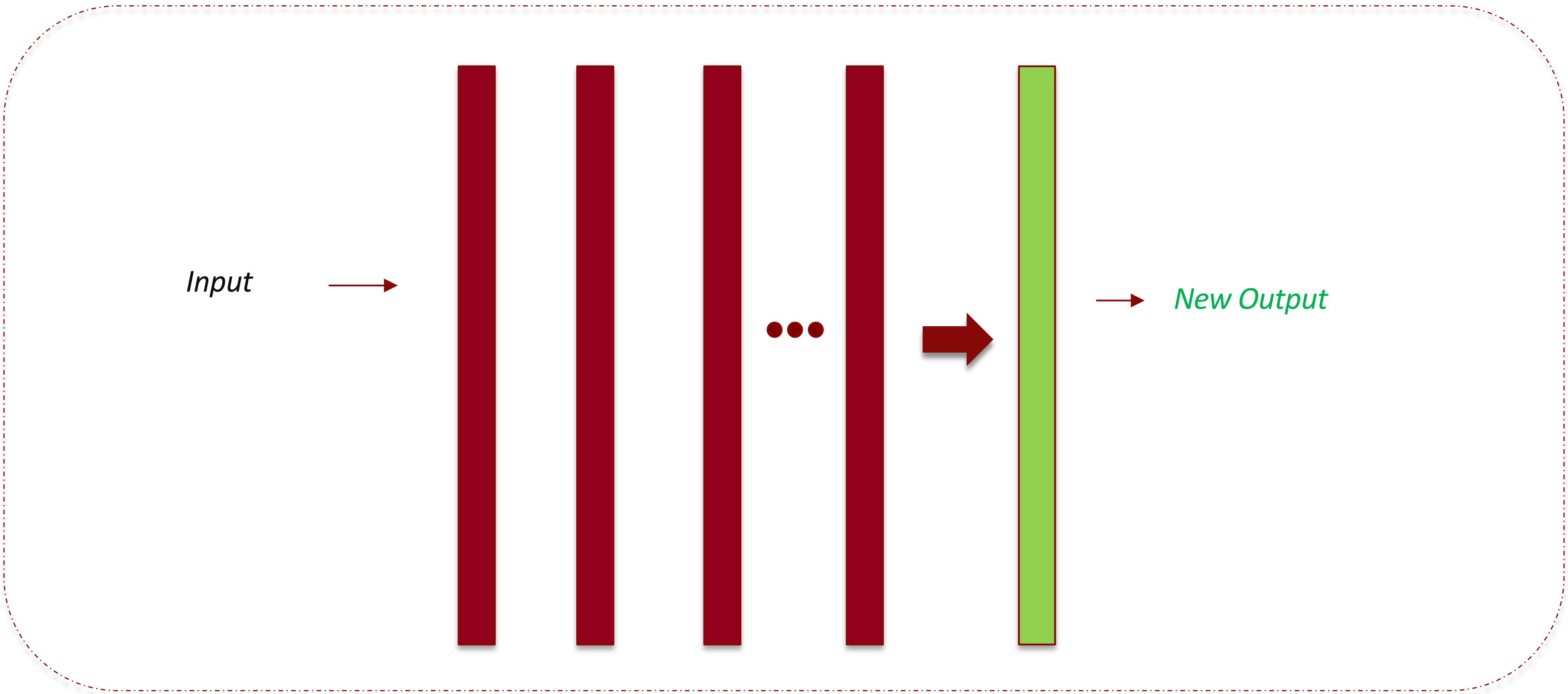
# So we “chop off” the output layer and use the resulting “headless model” as an encoder



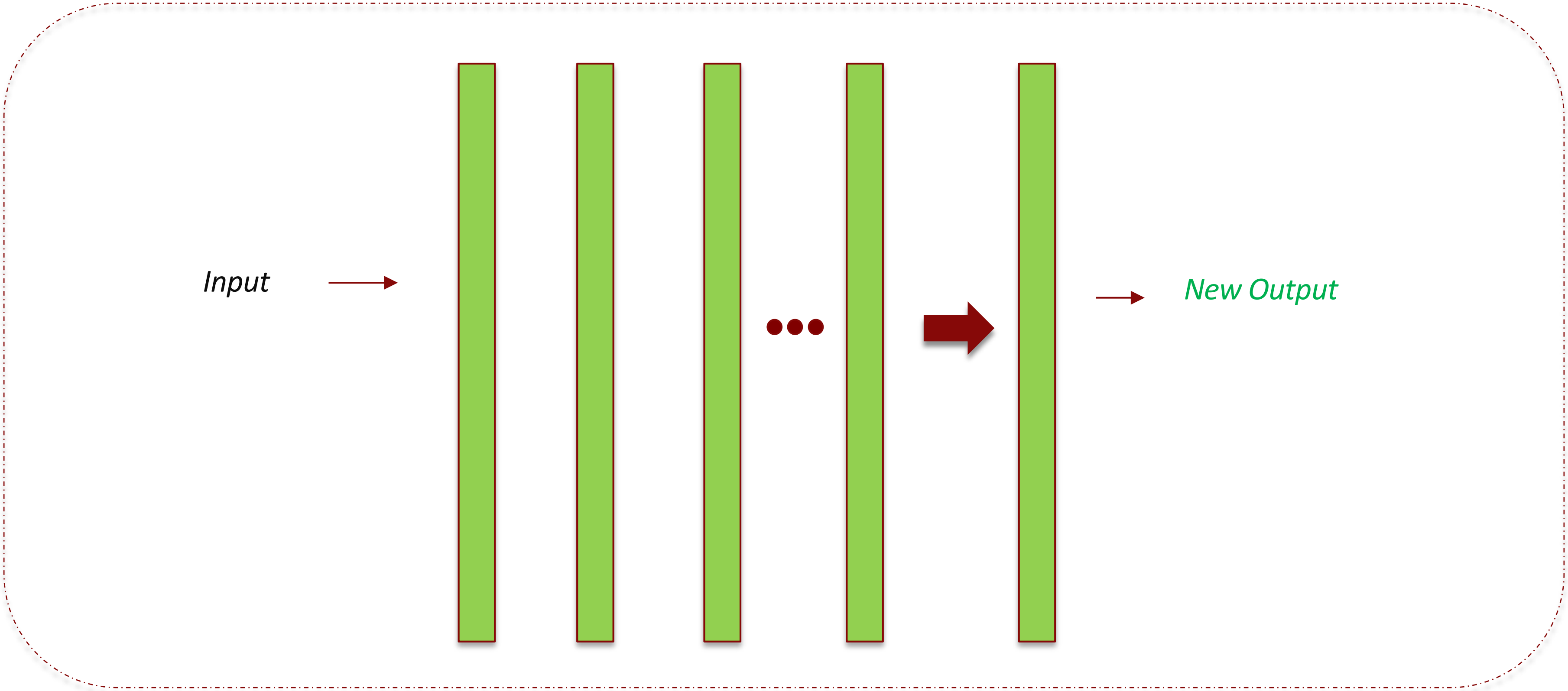
# We can “attach” a new output layer to this encoder and train the network with the *actual output labels* we care about!



# We can keep the encoder fixed and learn only the weights of the new final layer.



# … or fine-tune all the layers



To build such a generally useful pretrained model, we need labeled data.

For example, ResNet was trained on everyday images which were labeled with one of 1000 categories

To build a generally useful model (like ResNet) for *text* data, we need to (1) collect a lot of text data. This is no problem – there's plenty of text data on the Internet e.g., Wikipedia.

To build a generally useful model (like ResNet) for text data, we need to

(1) collect a lot of text data. This is no problem – there's plenty of text data on the Internet e.g., Wikipedia.

(2) we need to define output labels for every piece of text we feed into the model.

For an input sentence, what should the output label be?

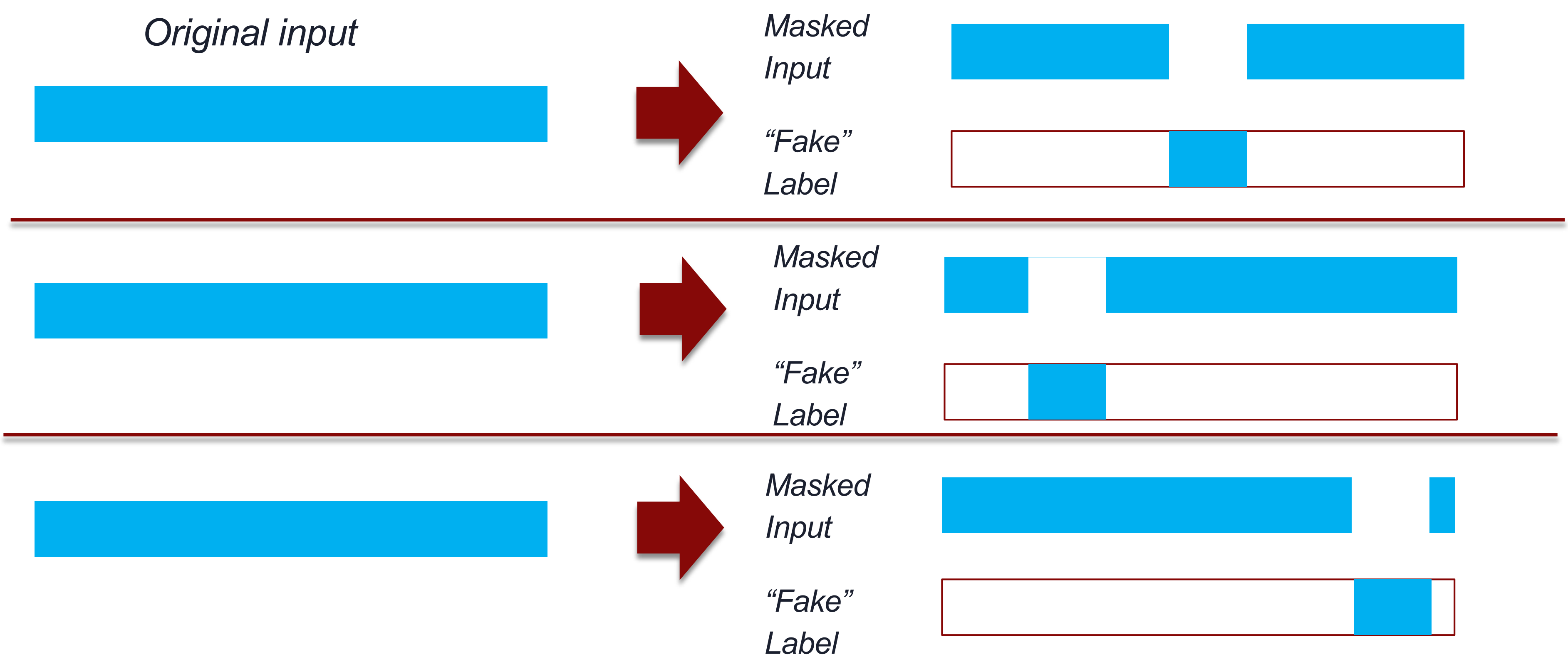
# A powerful approach to building pretrained models without labeled data: *Self-supervised Learning*

*The key idea behind self-supervised learning:*

***Predict a subset of the input data using the rest of the input***

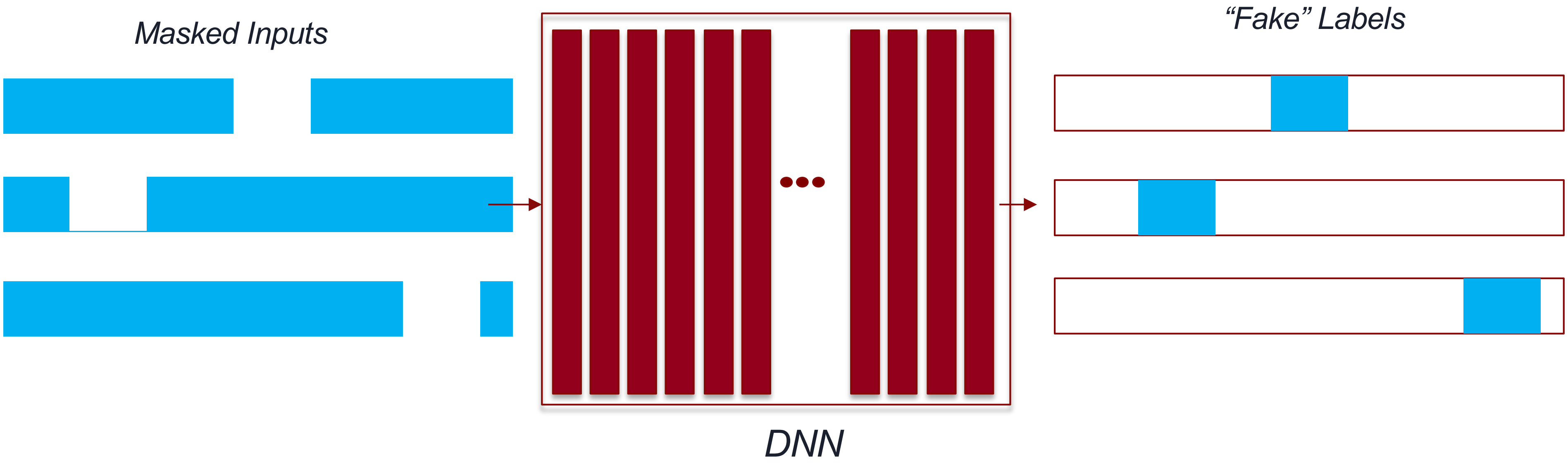
# Masking: A Self-supervised Learning Technique

*1. We modify the original input data to create “fake” (input, label) pairs by masking a part of the input and making it the label*



# Masking: A Self-supervised Learning Technique

2. *We then use train a Deep Neural Network to predict the "fake" labels from the modified inputs i.e., to fill in the blanks*



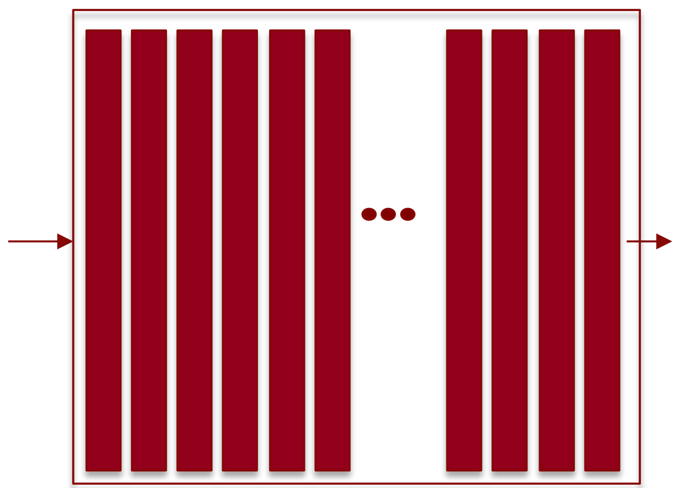
# Masking Example

*Original Input*

*“The mission of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance management practice.”*

*Modified Input*

*“The <MASK> of the MIT Sloan School of <MASK> is to develop principled, innovative <MASK> who improve the world and to <MASK> ideas that advance <MASK> practice.”*



*DNN*

*“Fake” Labels*

*The mission of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance management practice.*

Now for the amazing part.

In the process of learning to “fill in the blanks”” successfully, the Deep Neural Network learns a good representation of the input data.

Now for the amazing part.

In the process of learning to “fill in the blanks”, the Deep Neural Network learns a good representation of the input data.

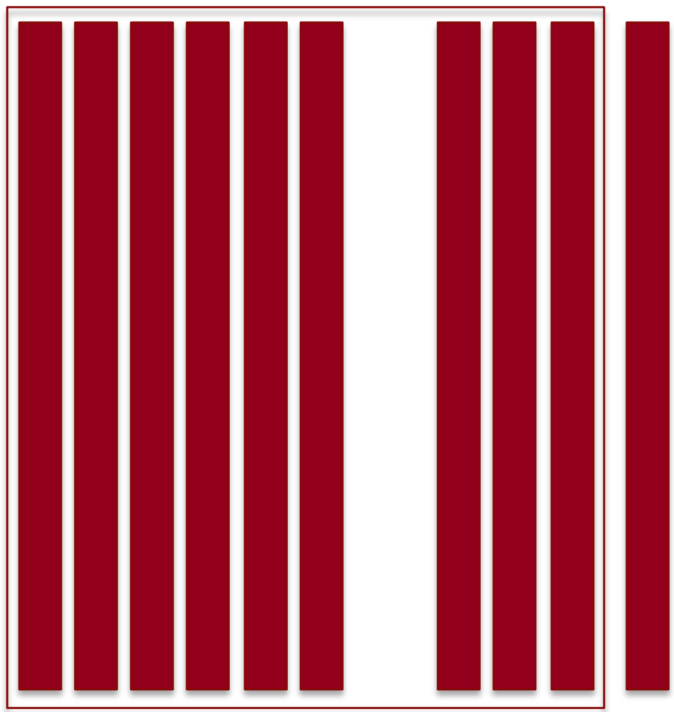
This intuitively makes sense. To fill in the blanks successfully, the model has to learn how the variables are related to each other.

# Once a self-supervised model is built, we can extract an encoder from it …

*Input*

*The ______ of the MIT Sloan School of ________ is to develop principled, innovative ______ who improve the world and to ______ ideas that advance ______ practice.*

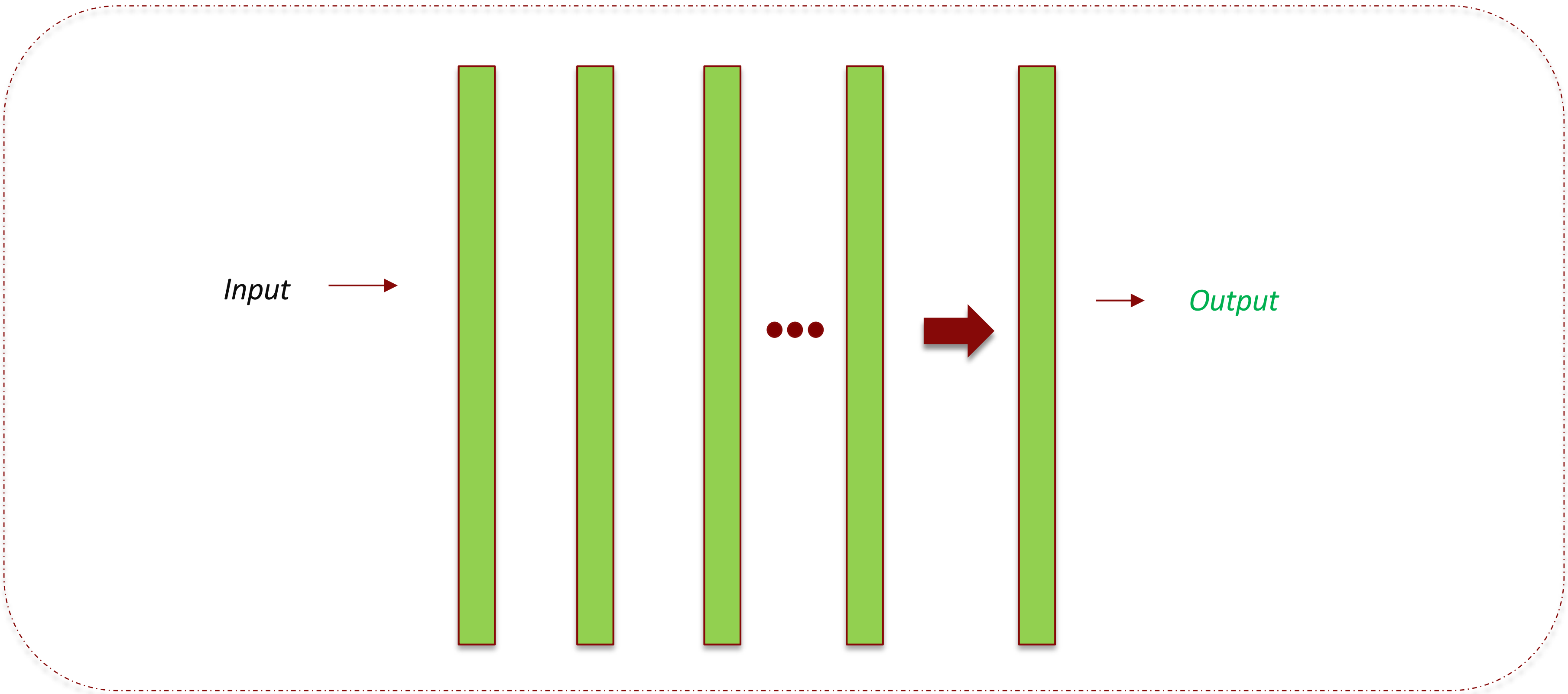
**Encoder**



*Output*

*The mission of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance management practice.*

# … and fine-tune it like we did in Transfer Learning



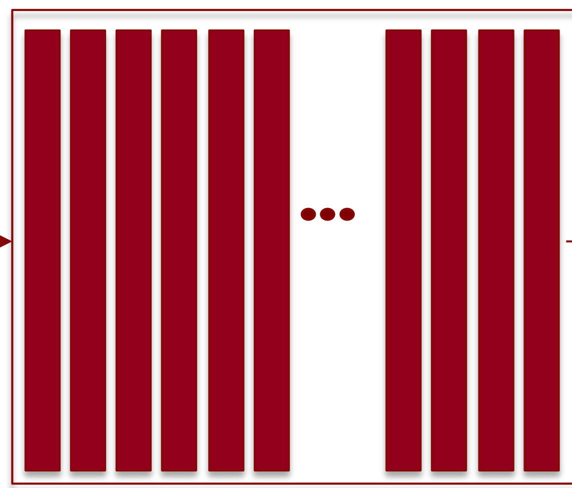
# We can use a Transformer Encoder to build this Self-supervised Learning model for text

*Original Input*

*"The mission of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance management practice."*

*Modified Input*

*"The <MASK> of the MIT Sloan School of <MASK> is to develop principled, innovative <MASK> who improve the world and to <MASK> ideas that advance <MASK> practice."*

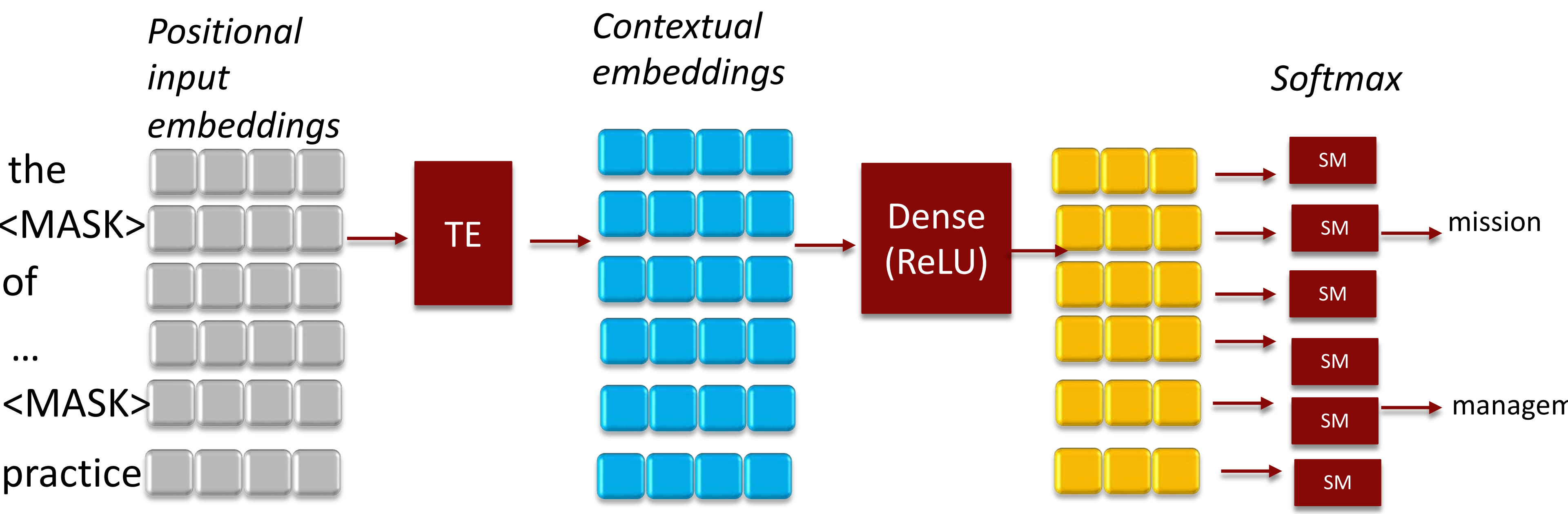


*DNN*

*"Fake" Labels*

*The mission of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance management practice.*

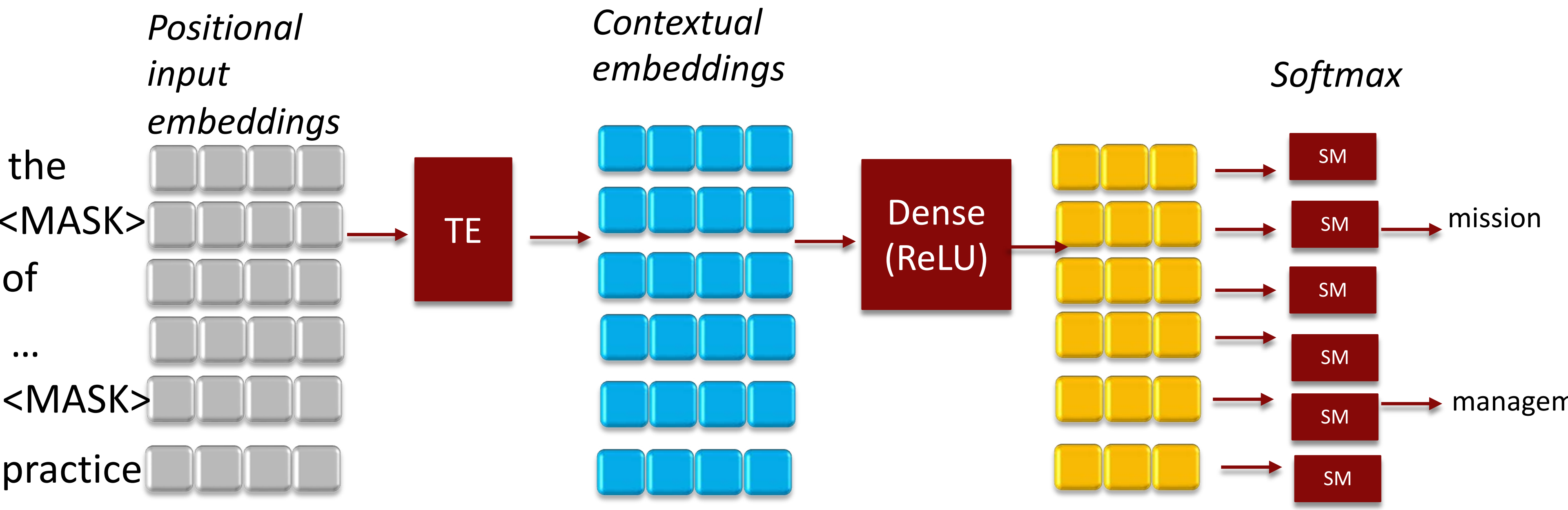
# Masked Self-Supervised Learning is just a sequence labeling problem




*"The _____ of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance _____ practice."*

*The DNN learns to predict the masked words from the rest of the sentence*

# If we pretrain a Transformer model like this on massive amounts of English text, we get …




*"The ______ of the MIT Sloan School of Management is to develop principled, innovative leaders who improve the world and to generate ideas that advance ______ practice."*

# … BERT!


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



**BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**

**Jacob Devlin  Ming-Wei Chang  Kenton Lee  Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

https://arxiv.org/pdf/1810.04805.pdf

# BERT uses the Transformer architecture



https://arxiv.org/pdf/1810.04805.pdf

**Model Architecture** BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017) and released in the `tensor2tensor` library.[1] Because the use of Transformers has become common and our implementation is almost identical to the original, we will omit an exhaustive background description of the model architecture and refer readers to Vaswani et al. (2017) as well as excellent guides such as "The Annotated Transformer."[2]

In this work, we denote the number of layers (i.e., Transformer blocks) as $L$, the hidden size as $H$, and the number of self-attention heads as $A$.[3] We primarily report results on two model sizes: $\mathbf{BERT}_{\mathbf{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M) and $\mathbf{BERT}_{\mathbf{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

$\mathrm{BERT}_{\mathrm{BASE}}$ was chosen to have the same model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses constrained self-attention where every token can only attend to context to its left.[4]

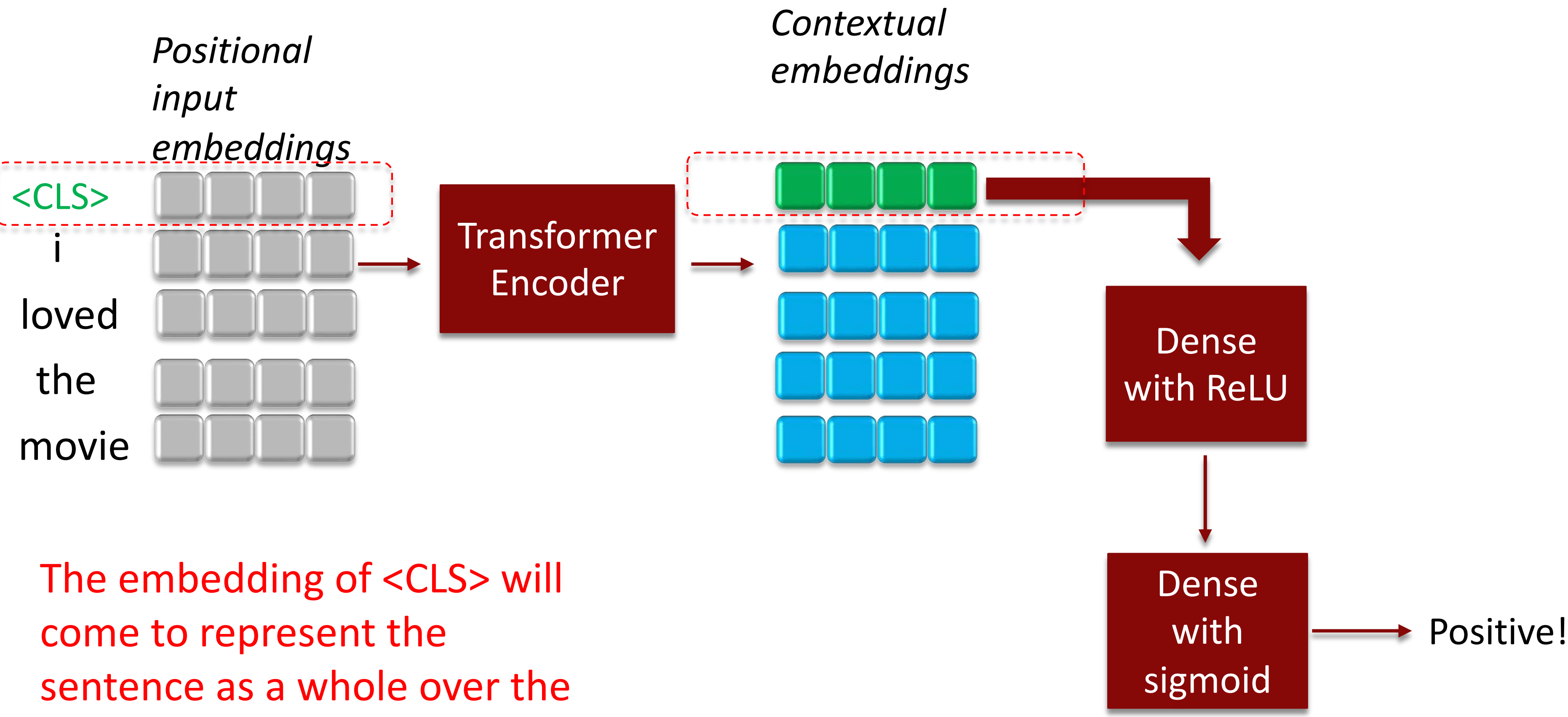
[1] https://github.com/tensorflow/tensor2tensor

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

[3] In all cases we set the feed-forward/filter size to be $4H$, i.e., 3072 for the $H = 768$ and 4096 for the $H = 1024$.

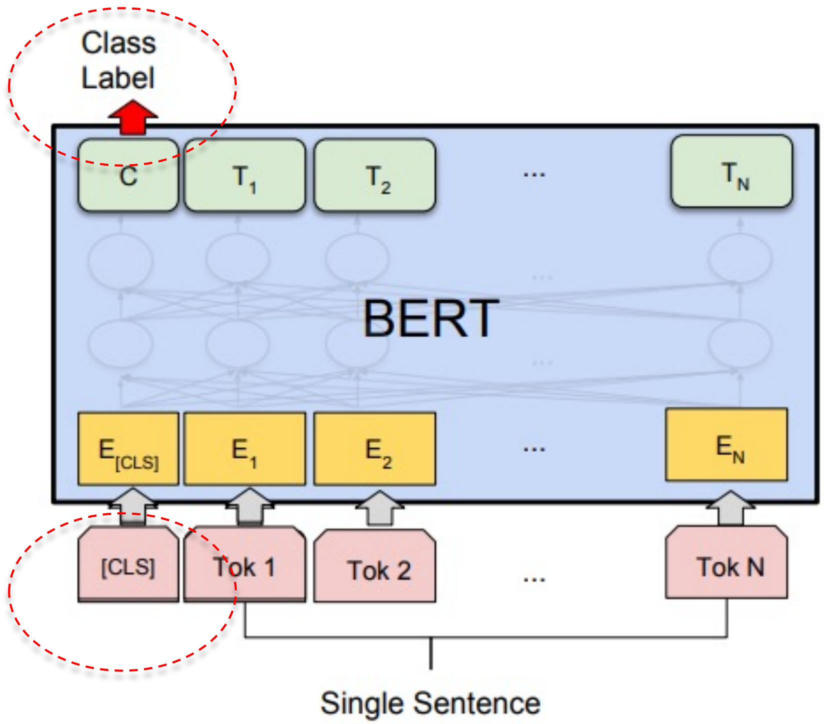
[4] We note that in the literature the bidirectional Trans-

# Earlier, we recommended adding a special token at the beginning of each sentence and just using its output embedding as the sentence-embedding




The embedding of <CLS> will come to represent the sentence as a whole over the course of training

# Conveniently, BERT was trained with the <CLS> token so it can be used for sequence classification "out of the box"*



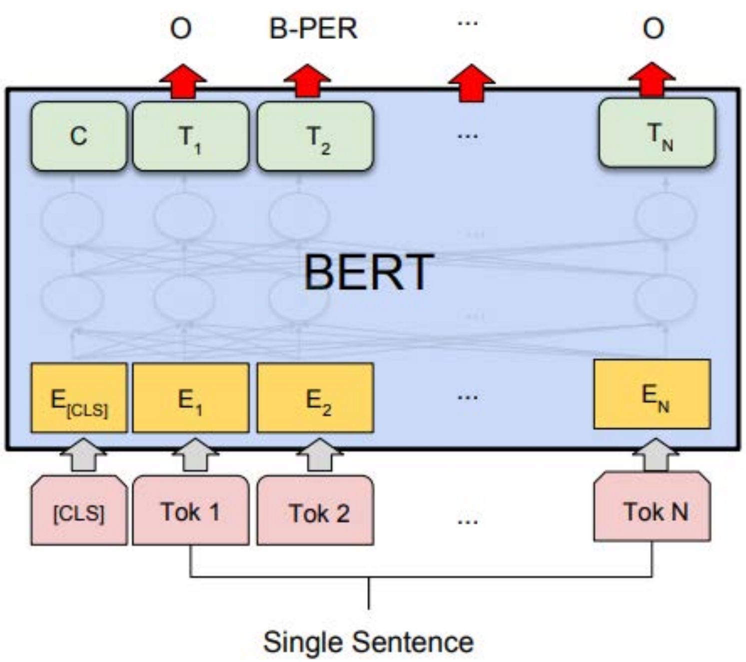

*Sequence classification*



Image credit: https://arxiv.org/pdf/1810.04805.pdf

*HW2 colab

# BERT is an excellent pretrained model for sequence labeling problems as well




*Sequence labeling*



Image credit: https://arxiv.org/pdf/1810.04805.pdf

A number of variations/improvements of BERT have appeared over the years and these can be used for many tasks.

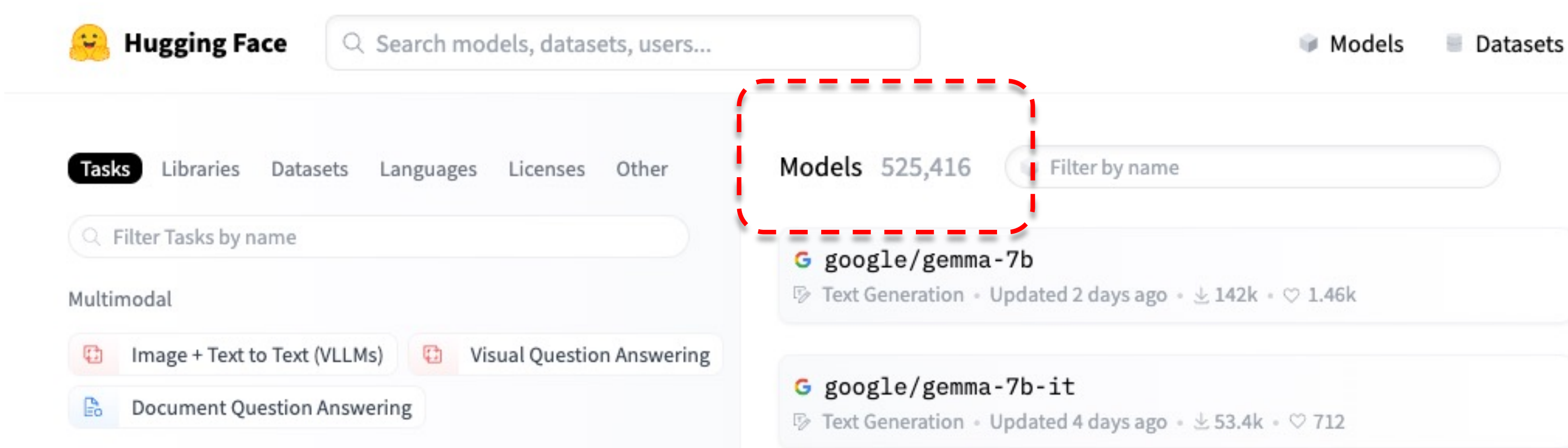
The Sentence Transformers library is a good resource

https://www.sbert.net/index.html

To solve <u>any</u> sequence classification or sequence labeling problem where the input is natural language text, we can use a model like BERT as a pre-trained encoder. Label “a few hundred” examples, attach the right final layers to BERT and fine-tune.

But if your particular problem is a “standard” NLP problem, this may not be necessary. Numerous pretrained models are available on various Hubs for all the “standard” NLP problems and you can start using them <u>without any fine-tuning at all.</u>

# The Hugging Face Hub is very popular




Over 500,000 pretrained models available!! (as of Feb 27, 2024)

https://huggingface.co/models

# [Huggingface Colab]

# Transformers have proven to be an effective DNN architecture across a vast array of domains

Information Retrieval/Search

Machine Translation

Speech Recognition

Text-to-Speech

Computer Vision

Reinforcement Learning

Generative AI (LLMs, Text-to-image models,
Image Captioning, …)

Numerous special-purpose systems (e.g., AlphaFold)

…

# Transformers have proven to be an effective DNN architecture across a vast array of domains

- The architecture of the Transformer block can be used as-is for a wide range of applications

- What tends to vary from application to application is how the inputs are encoded/tokenized in a form that can be fed to the Transformer

# Vision Transformer: A Transformer for Image Classification

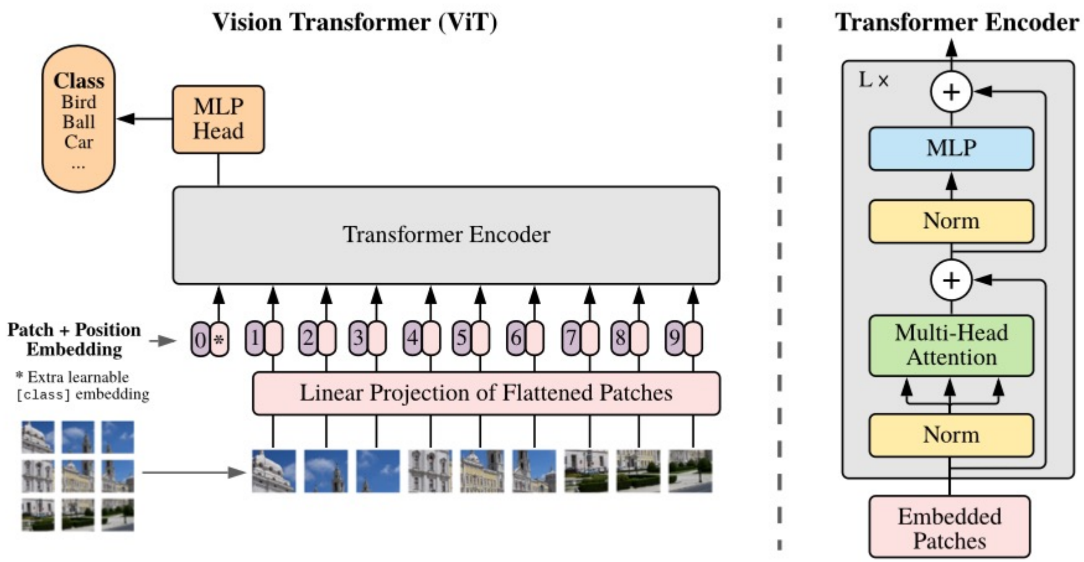



Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

https://arxiv.org/pdf/2010.11929.pdf

# The Tab Transformer: : A Transformer for Tabular Data

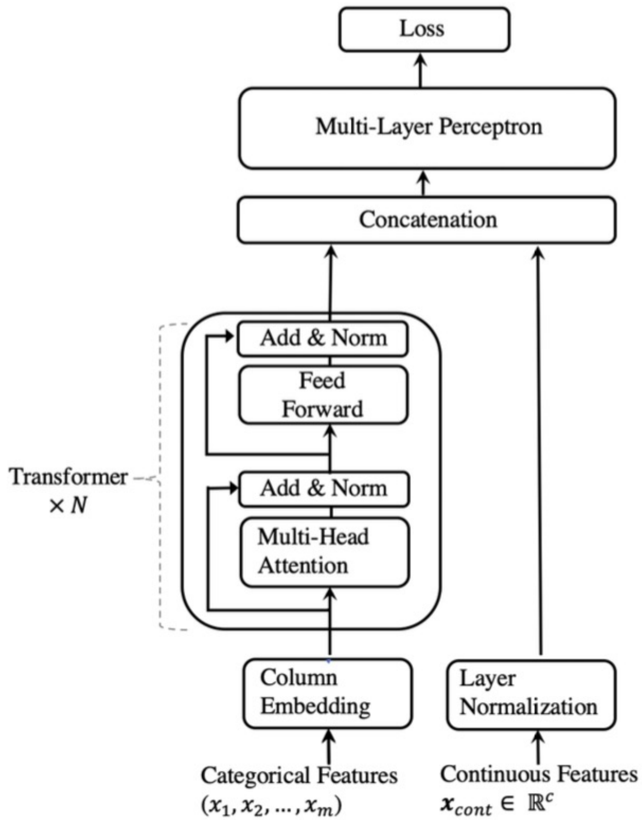



Figure 1: The architecture of TabTransformer.



http://arxiv.org/abs/2012.06678

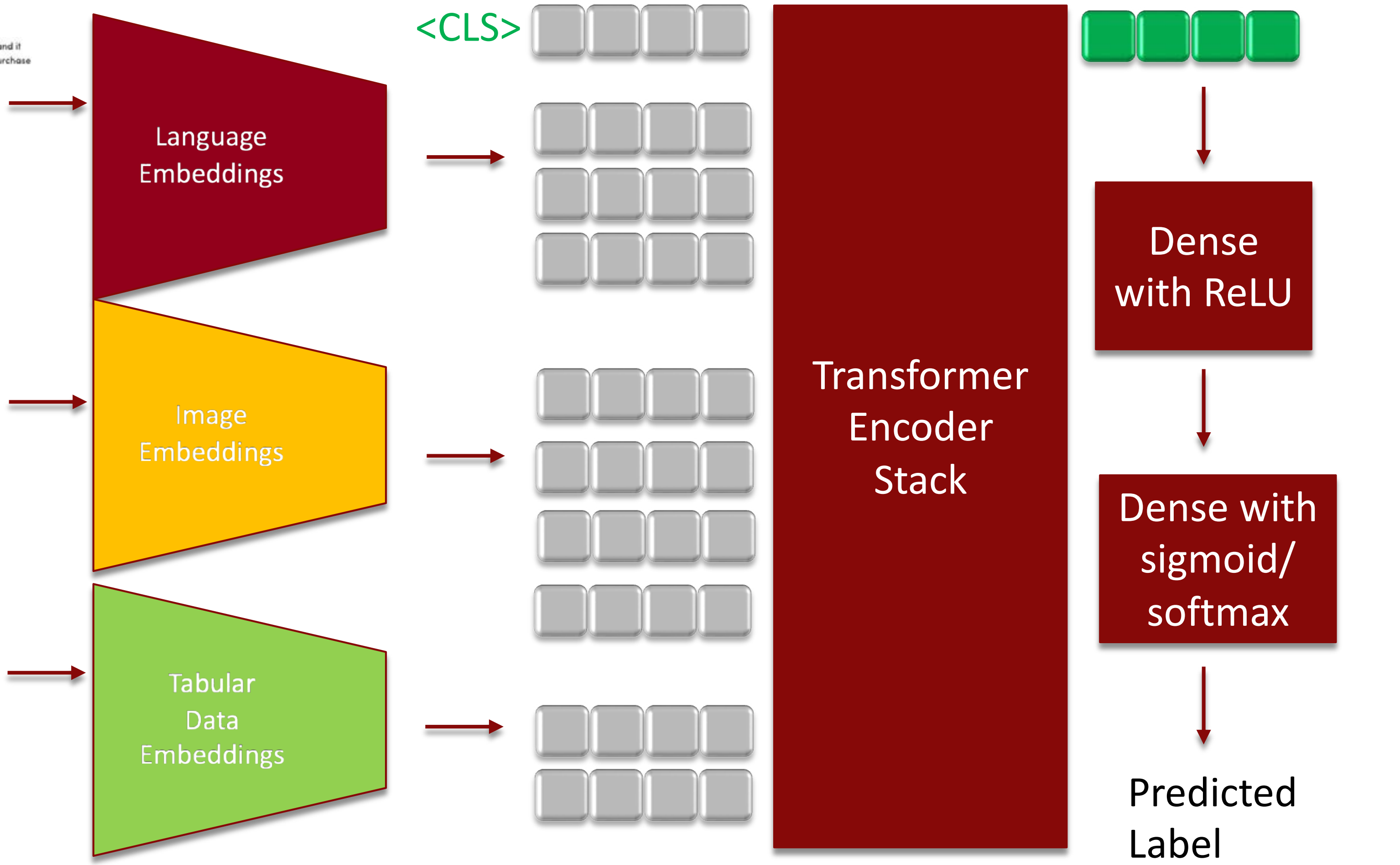
Once the input has been transformed into the “common language” of embeddings, we can process them without changing the architecture of the Transformer Encoder block.

This turns out to very useful for multi-modal data

# Example: A Transformer-based classifier for multi-modal data

Not pleased with this chair - put in my office as I work from home and it wobbles from side to side - back support is non existent . Had to purchase another chair. This one is sitting in the basement now (

| Variable | Value |
|---|---|
| Age | 37 months |
| Supplier | Acme |
| # parts | 13 |
| … | … |



# Contrastive Learning (time permitting)

We can pretrain models on unlabeled text data by using self-supervised learning to create artificial labels (e.g., by masking words and recovering them).

How can we pretrain models on unlabeled image data?

# Contrastive Learning

For self-supervised learning with image inputs, a technique called *contrastive learning* has been found to be very effective*

The basic approach:

- For every original image, artificially construct a pair of "augmented" images
- Train the network to "maximize agreement" i.e., make the learned representations of each augmented pair "close" to each other but "far" from the representations of the other pairs

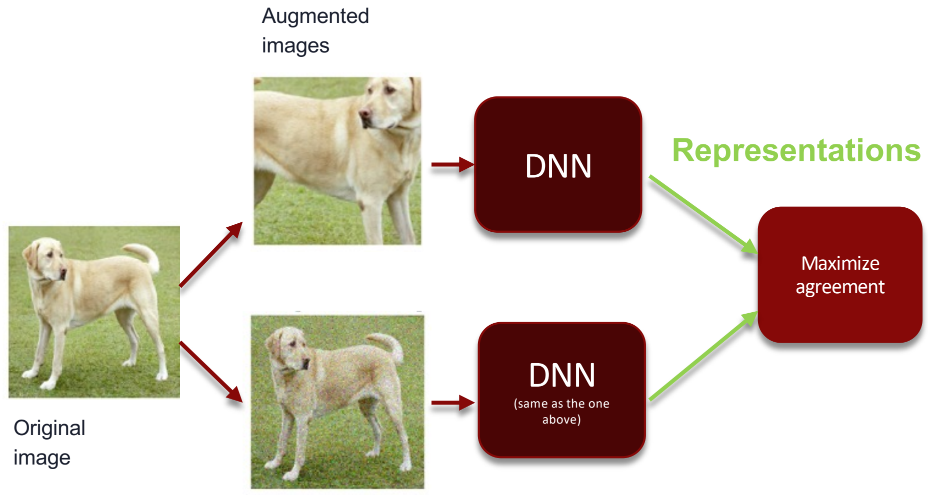



Image credit: http://arxiv.org/abs/2002.05709

*A Simple Framework for Contrastive Learning of Visual Representations by Chen et al (2020)

# Data augmentation examples

(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

Image credit: http://arxiv.org/abs/2002.05709

# Once the contrastive learning model is built, we can extract an encoder from it easily and fine-tune it