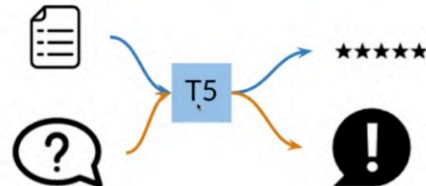


## Week 3

Question Answering



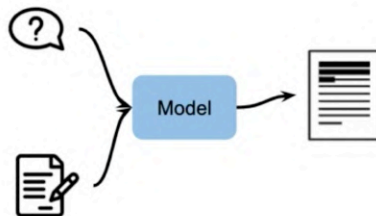
Transfer learning



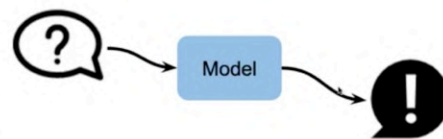
You're going to cover many different applications of NLP. One thing you are going to look at is question answering. Given the question and some contexts, can you tell us what the answer is going to be inside that context. Another thing you're going to cover is transfer learning. For example, knowing some information by training something in a specific task. How can you make use of that information and apply it to a different task? You're going to look at BERT, which is known as the Bidirectional Encoder Representation, which makes use of transformers. You'll see how you can use bidirectionality to improve performance. Then you're going to look at the T5 model. Basically what this model does, you can see here it has several possible inputs. So it could be question, you get an answer. It could be review and you will get the rating over here. It's all being fed into one model.

## Question Answering

Context-based

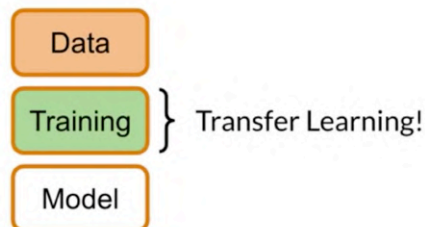
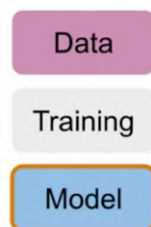


Closed book



Let's look at question answering. Over here you have context-based question-answering, meaning you take in a question and the contexts. It tells you where the answer is inside that's context over here. So this is the highlighted stuff, which is the answer. Then you have closed book question answering, which only takes the question and it returns the answer without having access to a context, so it comes up with its own answer.

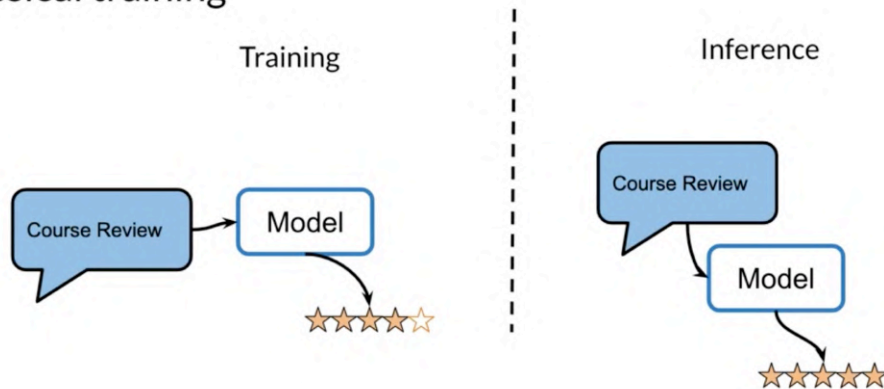
## Not just the model



Previously we've seen how innovations in model architecture, improved performance, and we've also seen how data preparation could help. But over here, you're going to see that innovations in the way that training is being done also improves performance. In which case, you will see how transfer learning will improve performance.

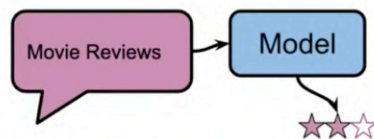
## Classical training

This is the classical training that you're used to seeing. You have a course review, this goes through a model and let's you predict the rating. Then you just predict the rating the same way as you've always been doing. So nothing changed here, this is just an overview of the classical training that you're used to seeing.

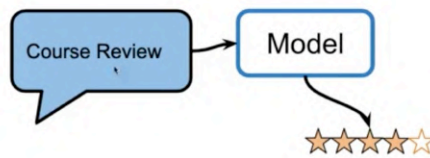


## Transfer learning

Pre-training



Training on "Downstream" Task

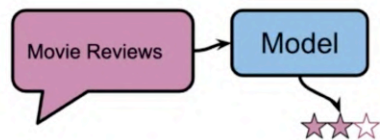


Inference

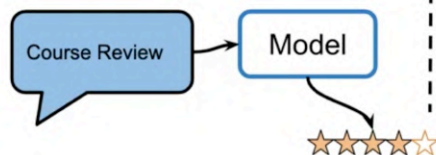
You have a course review, this goes through a model and let's you predict the rating. Then you just predict the rating the same way as you've always been doing. So nothing changed here, this is just an overview of the classical training that you're used to. Now in transfer learning, let's look at this example. Let's say that you have movie reviews and then you feed them into your model and you predict a rating. Over here you have the pretrain task, which is on movie reviews. Now when training, you're going to take the existing model or movie reviews and then you're going to find units or train it again on course reviews. You'll predict the rating for that review. So as you can see over here, instead of initializing the weights from scratch, you start with the weights that you got from the movie reviews and you use them as a starter point when training for the course reviews.

## Transfer learning

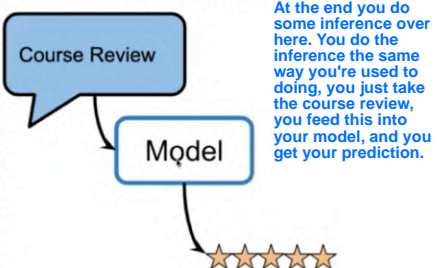
Pre-training



Training on "Downstream" Task

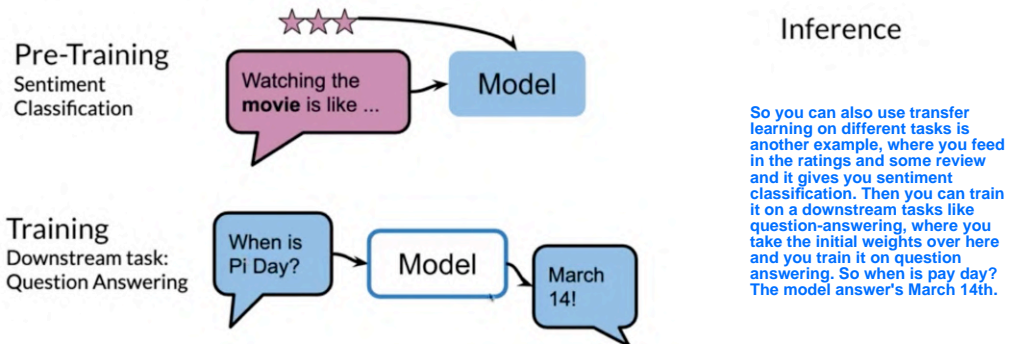


Inference

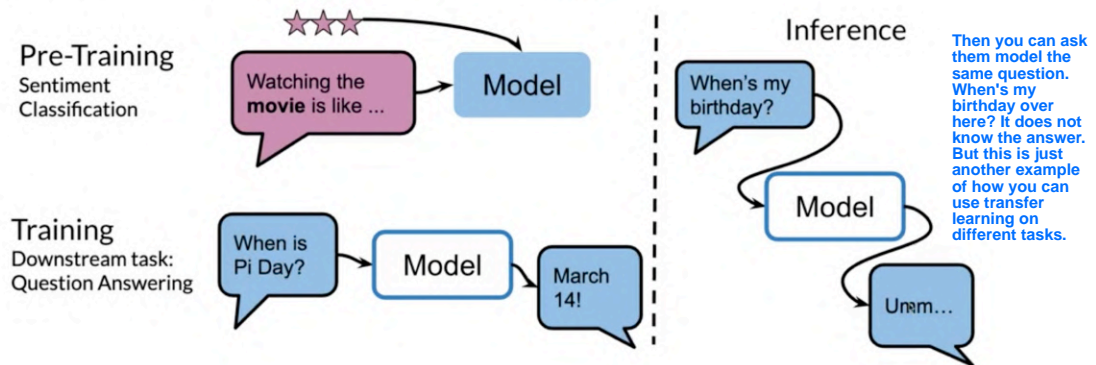


At the end you do some inference over here. You do the inference the same way you're used to doing, you just take the course review, you feed this into your model, and you get your prediction.

## Transfer Learning: Different Tasks

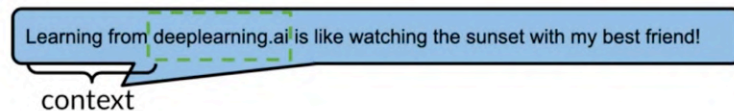


## Transfer Learning: Different Tasks

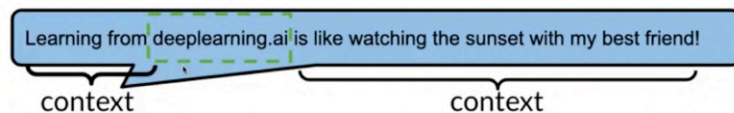


## BERT: Bi-directional Context

### Uni-directional

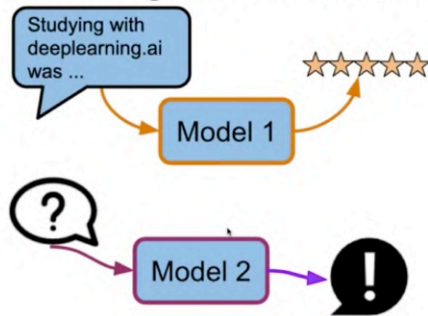


### Bi-directional



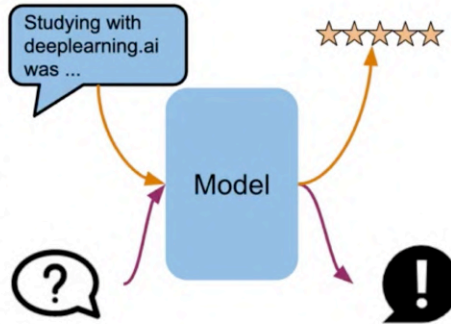
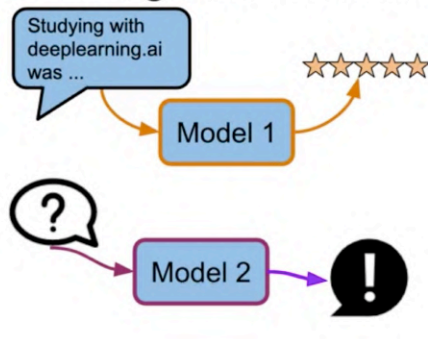
Now we're going to look at BERT, which makes use of bi-directional context. In this case, you have learning from deep learning AI, it's like watching the sunset with my best friend. Over here the context is everything that's come before. Then let's say you're trying to predict the next word, deep-learning AI. Now when doing bi-directional representations, you'll be looking at the context from this side and from this side to predict the middle word. This is one of the main takeaways for bidirectionality.

## T5: Single task vs. Multi task



Now let's look at single task versus multitask. Over here you have a single model which takes in a review and then predicts a rating. Over here you have another model which takes in a question and predicts an answer. This is a single task, each like one module per task.

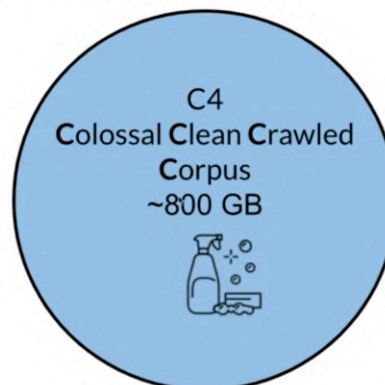
## T5: Single task vs. Multi task



Now, what you can do here with T5 is it is the same model that's being used to take the review, predict the rating, and then take the question and predict the answer. So instead of having two independent models, you end up having one model.

## T5: more data, better performance

English wikipedia  
~13 GB



Let's look at T5. Over here, the main takeaway is that the more data you have, generally the better performance there is. For example, the English Wikipedia dataset is around 13 gigabytes compared to the C4 Colossal Clean Crawled Corpus is about a 800 gigabytes, which is what T5 was trained on. This is just to give you, how much larger the C4 data sets is, when compared to the English Wikipedia.

## Desirable Goals



- Reduce training time



- Improve predictions

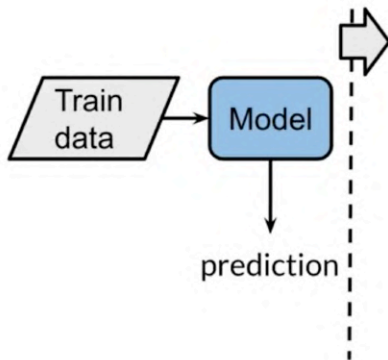


- Small datasets

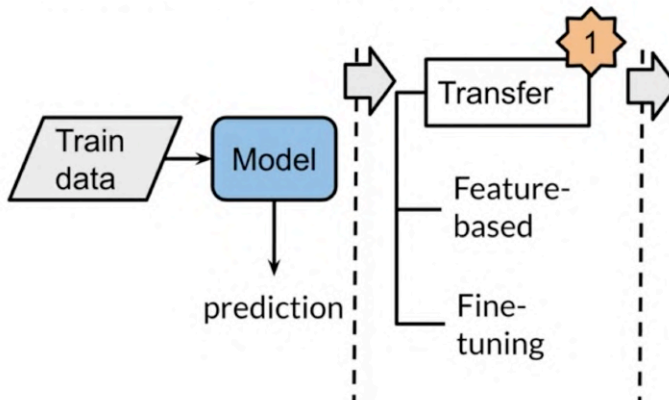
What are the desirable goals for transfer learning? First of all, you want to reduce training time because you already have a pre-trained model. Hopefully, once you use transfer learning, you'll get faster convergence. It will also improve predictions because you'll learn a few things from different tasks that might be helpful and useful for your current predictions on the task you're training on. Finally, you might require or need less data because your model has already learned a lot from other tasks. So if you have a smaller dataset, then transfer learning might help you.

Transfer Learning!

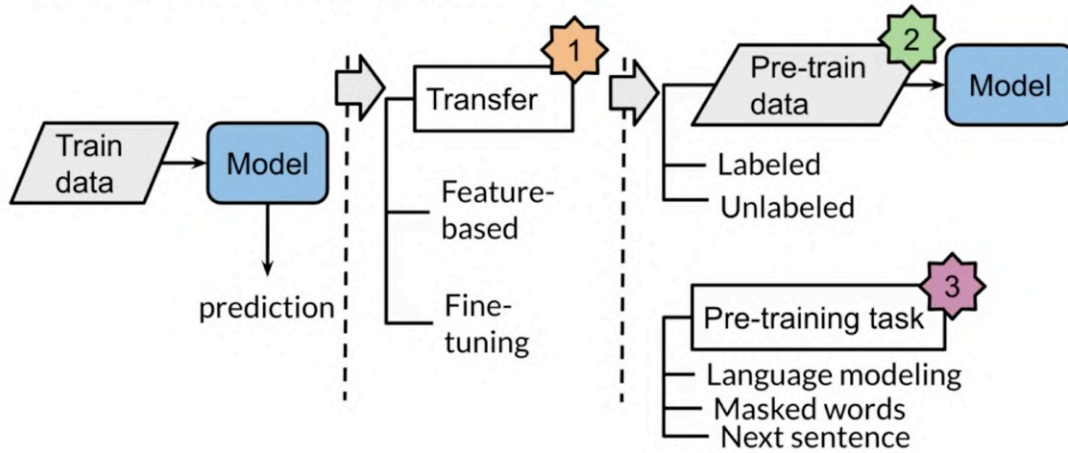
## Transfer learning options



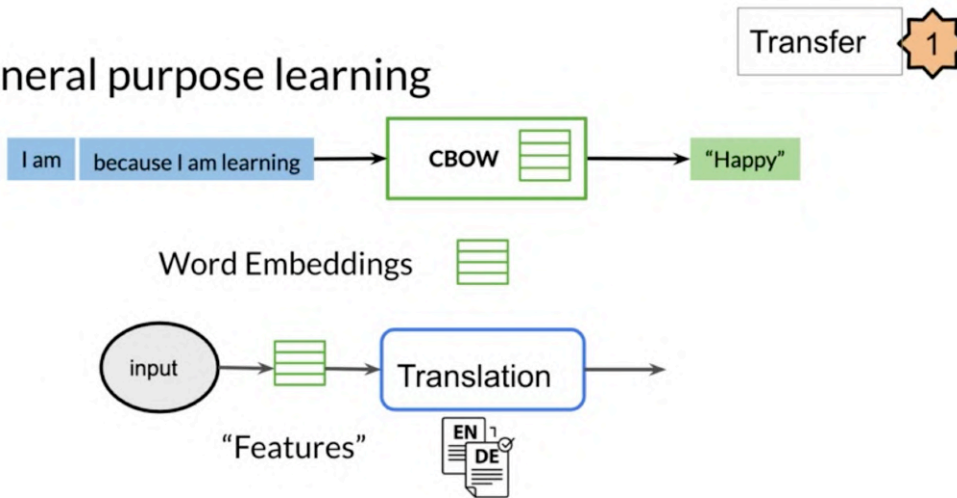
## Transfer learning options



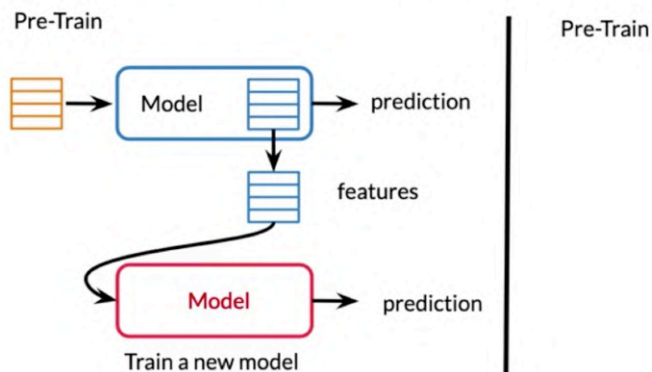
## Transfer learning options



## General purpose learning



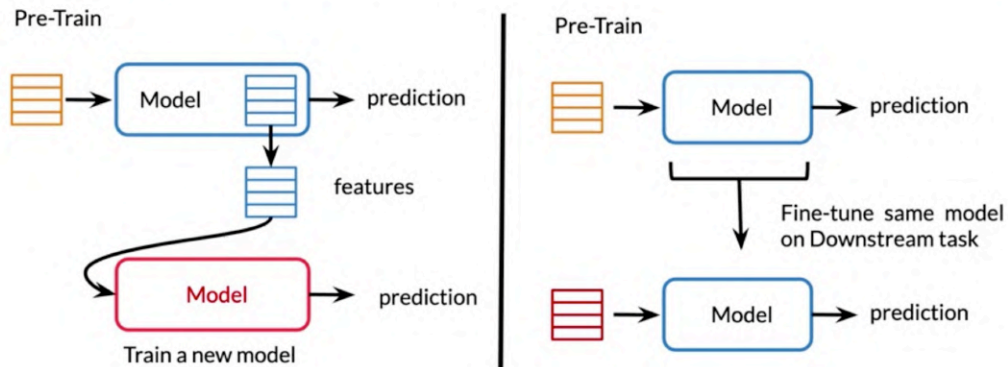
## Feature-based vs. Fine-Tuning





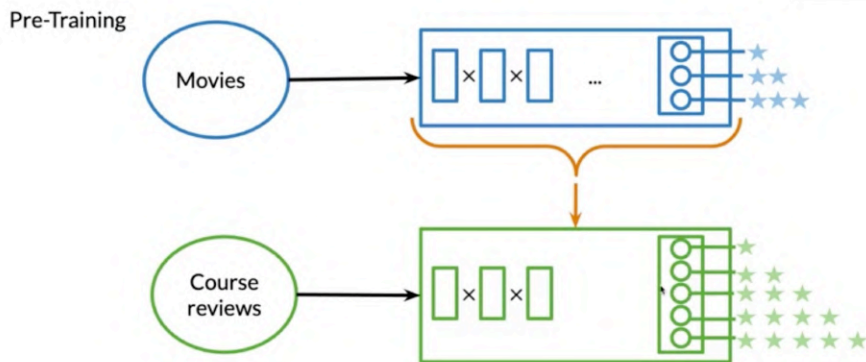
Transfer 

## Feature-based vs. Fine-Tuning




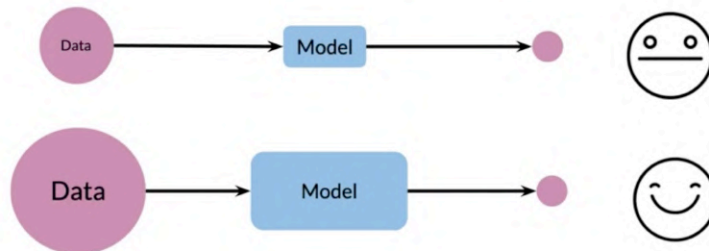
## Fine-tune: adding a layer

Transfer 



## Data and performance

Pre-train data 

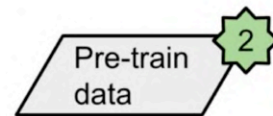
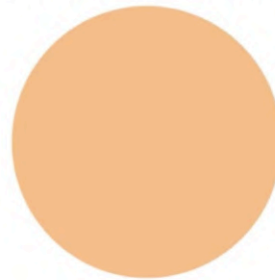


## Labeled vs Unlabeled Data

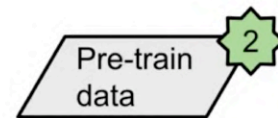
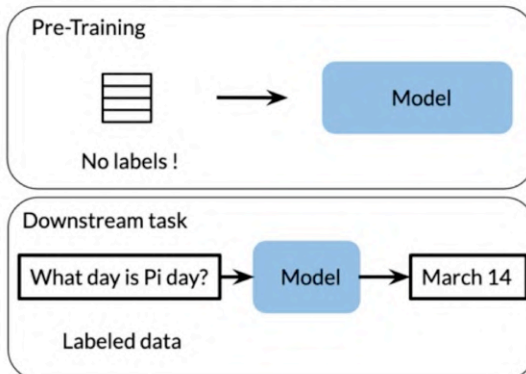
Labeled text data



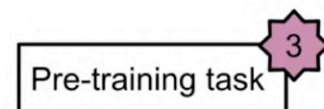
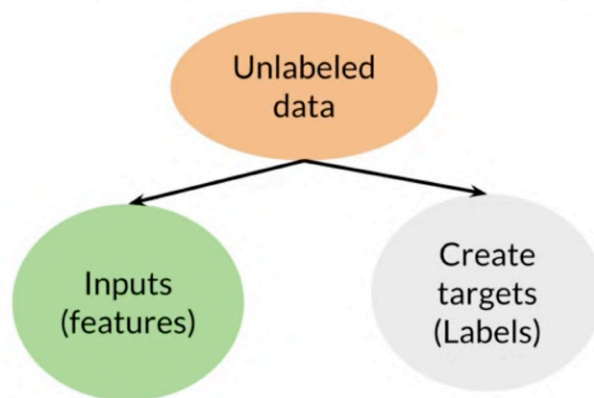
Unlabeled text data



## Transfer learning with unlabeled data

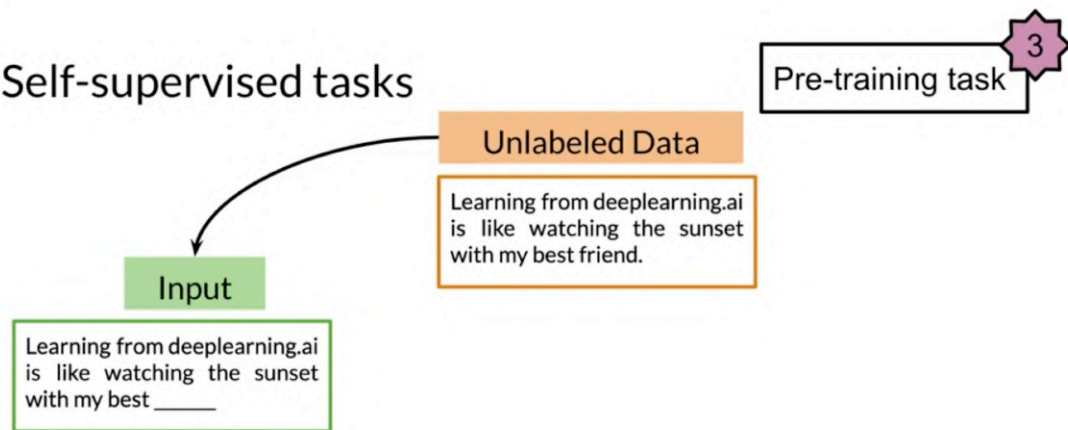


## Self-supervised task

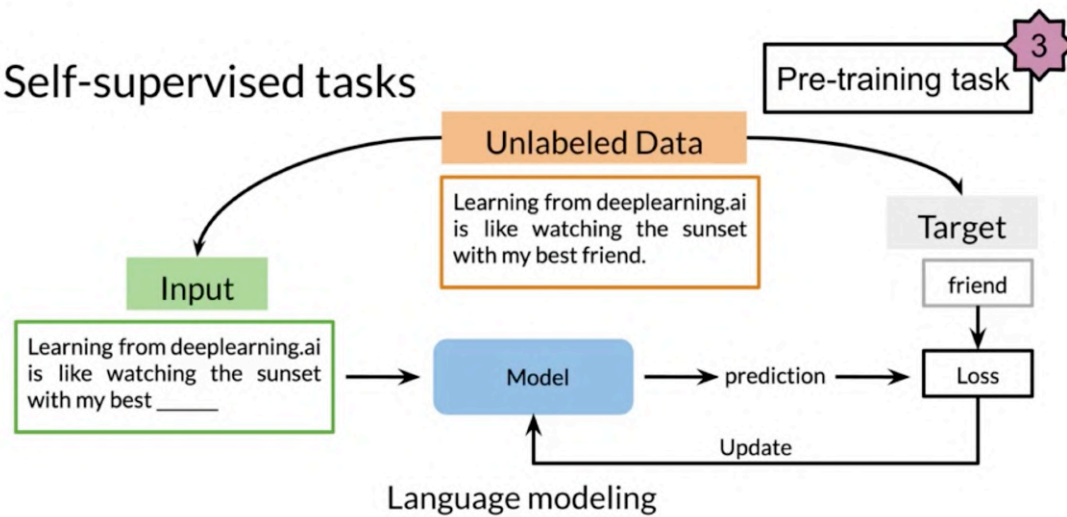




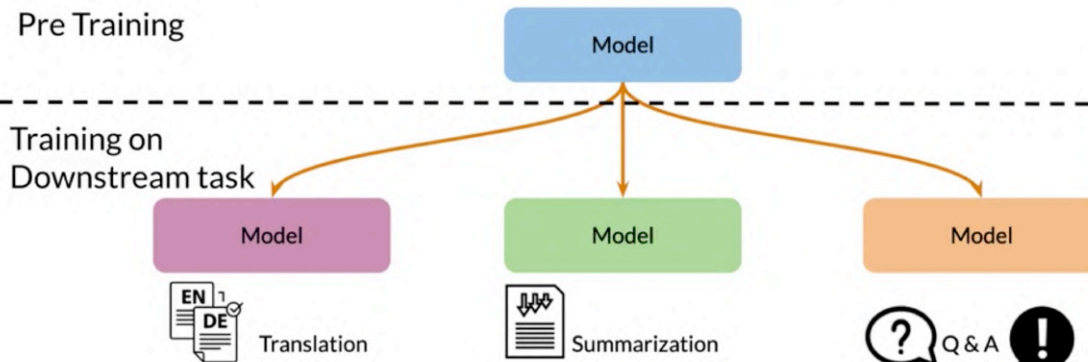
## Self-supervised tasks



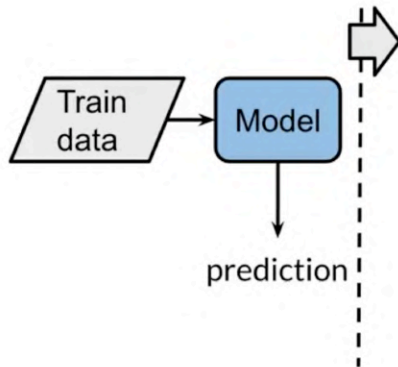
## Self-supervised tasks



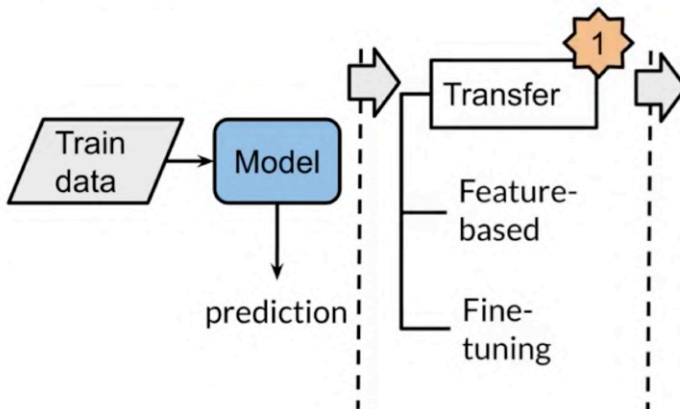
## Fine-tune a model for each downstream task



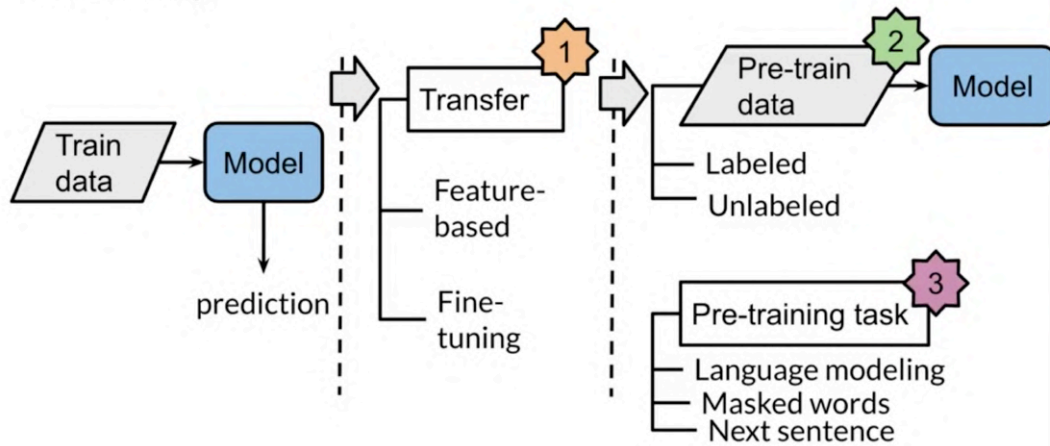
## Summary



## Summary



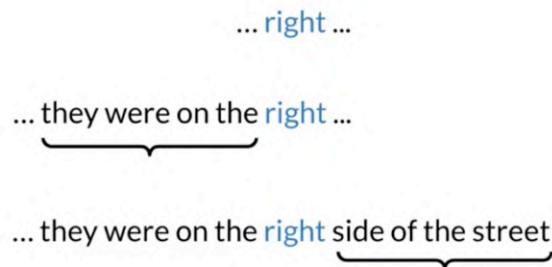
## Summary



## Outline

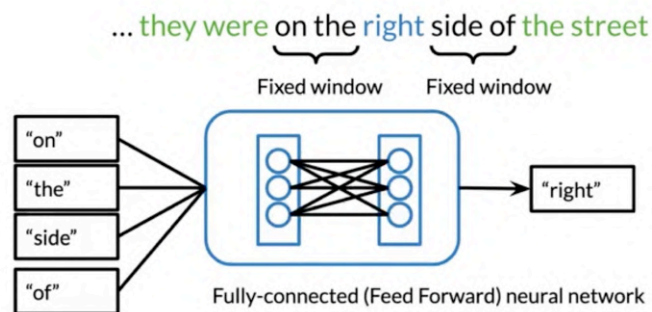


## Context



Over here, if you see a word and you want to learn what the word means. To do this as you've seen previously, you use the context before the word and the context after the word and the CBOW as an example to learn and create features for the word "right".

## Continuous Bag of Words



Now one of the issues with this CBOW model was that we used a fixed window. So we used a context window of size  $c$ , in this case 2. The goal was that we used a fixed window and given the words, we fed them into a neural network and predicted the output which was the center word.

Need more context?

... they were on the right side of the street.  
Fixed window Fixed window

... they were on the right side of history.

Now the issue is what if you need more context? So in this case, you have a fixed window, but let's say you want to add "the street" as part of the context to the right. On the otherhand, maybe it could be "history" instead of "the street". So how do you incorporate this?

Use all context words

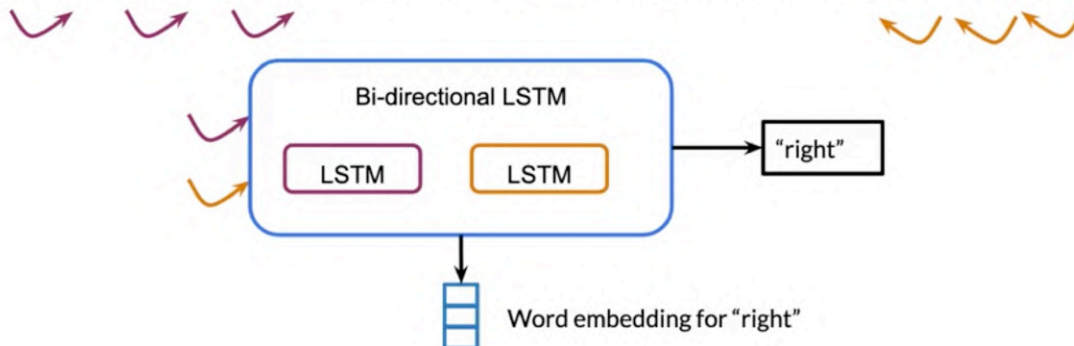
To incorporate this, we want to find a way such that we can use all the context words. So here the text starts with "the legislators believed that they were on the" and then to the right we have "side of history so they changed the law". Now to use all the context words, we can use ELMo.

The legislators believed that they were on the right side of history, so they changed the law.

ELMo: Full context using RNN

ELMo is a model which makes use of an RNN on both sides and uses these two RNNs to predict the center word. So concretely, it uses a bidirectional LSTM which is another version of an RNN and we have the input from both sides that we feed in and it predicts the output word, in this case "right". You also get the word embedding for the word "right" from this model.

The legislators believed that they were on the \_\_\_\_ side of history so they changed the law.



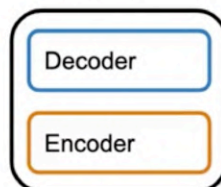
Open AI GPT

Now, OpenAI brought GPT and basically we first started by using a transformer which has an encoder and decoder. Then we started using GPT which just uses the decoder stack. We also had ELMo which just uses RNNs.

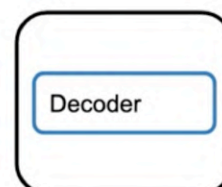
ELMo



Transformer

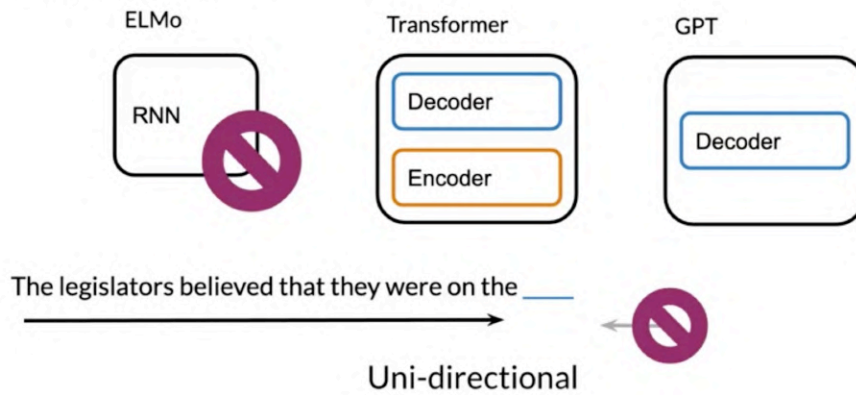


GPT

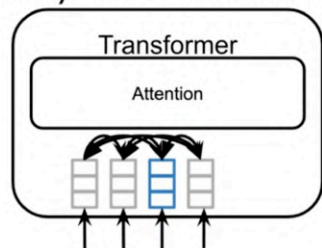


So given the following sentence, ideally we also want to know what's coming from the other side. Unfortunately, GPT is uni-directional. Although ELMo was bi-directional, it also suffered from some issues such as capturing longer term dependencies.

## Open AI GPT



## Why not bi-directional?



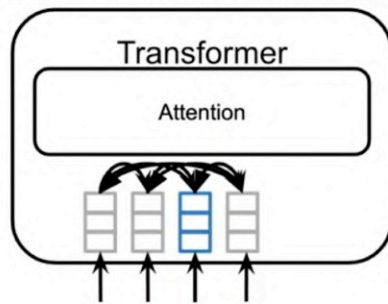
... on the right side...

Each word can peek at itself!

So why not use bi-directional?  
So in transformers, remember  
we had the following which  
was self-attention and each  
word can peek at itself.

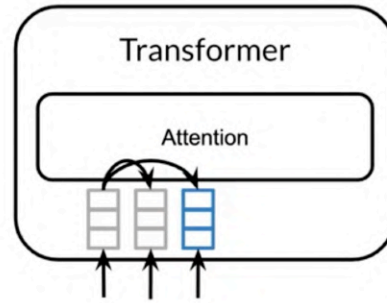
But the issue here is that GPT is still uni-directional and remember from previous lessons you've seen casual attention and how you can only look at previous inputs so there's no peeking.

## GPT: Uni-directional



... on the right side...

Each word can peek at itself!



... on the right

No peeking!

So to recap, we had transformers which had encoder and decoder, we had GPT which was only a decoder, and then we had BERT which is only an encoder. And as it works, we have the following "The legislators believed that they were on the <blank> side of history so they changed the law". And ideally, we want to find a way such that we use bidirectional representations from both sides and that's what bidirectional encoder representation from transformer does basically.

## BERT

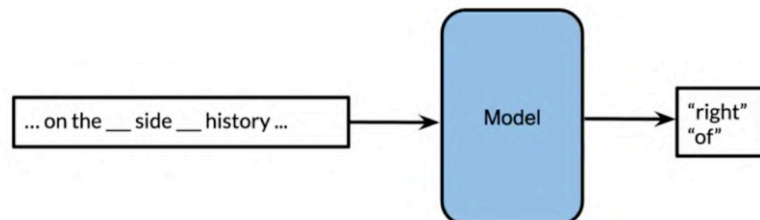


The legislators believed that they were on the \_\_\_\_ side of history, so they changed the law.

Bi-directional

So now lets try to combine the transformer and the bi-directional context and feed it into our model and we get the following output. This is an example of multi-mask language

## Transformer + Bi-directional Context

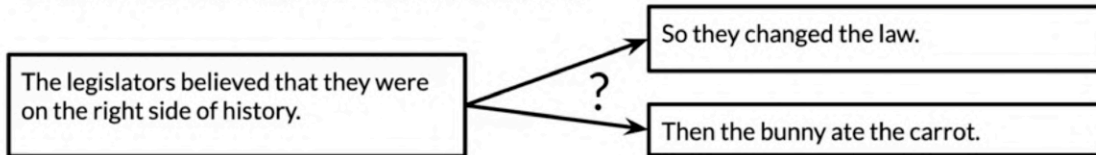


Multi-Mask Language Modeling



Now, for BERT from words to sentences, we can have the sentence on the left and two possible sentences coming up next and we want to predict which one makes more sense. So given sentence A, we try to predict the next sentence B.

## BERT: Words to Sentences



Next Sentence Prediction

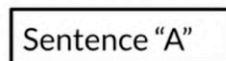
So for BERT pre-training task, we have multi-mask language modeling meaning we mask several words in a sentence and try to predict what the masked words were. Feed it into the model and we get the two mask words in this case. Then we also have the next sentence prediction, so given sentence A, we predict which one is sentence B.

## BERT Pre-training Tasks

Multi-Mask Language Modeling

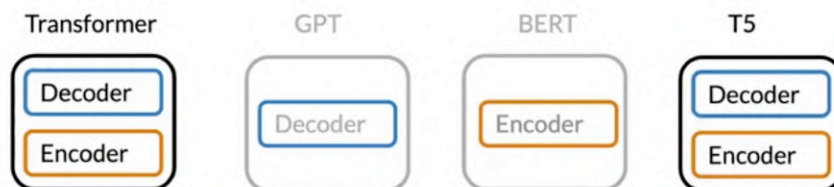


Next Sentence Prediction



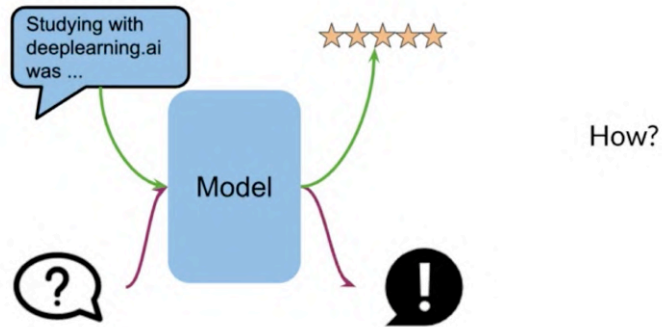
Then T5 came and T5 did not only use an encoder, but it used both an encoder and decoder. So to recap, the following GPT, BERT, and transformer models are shown below.

## T5: Encoder vs. Encoder-Decoder



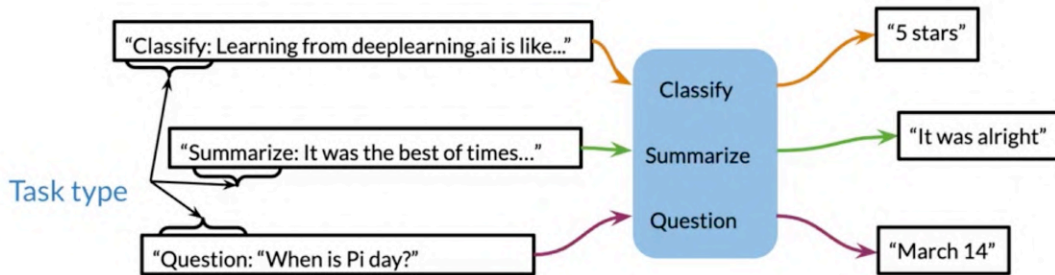
## T5: Multi-task

So T5 uses multi-task learning. so given the following input "Studying with deeplearning.ai was...", feed it into the model and we get 5 stars. We can use the same model and given a question, get an answer. So how can we do this?

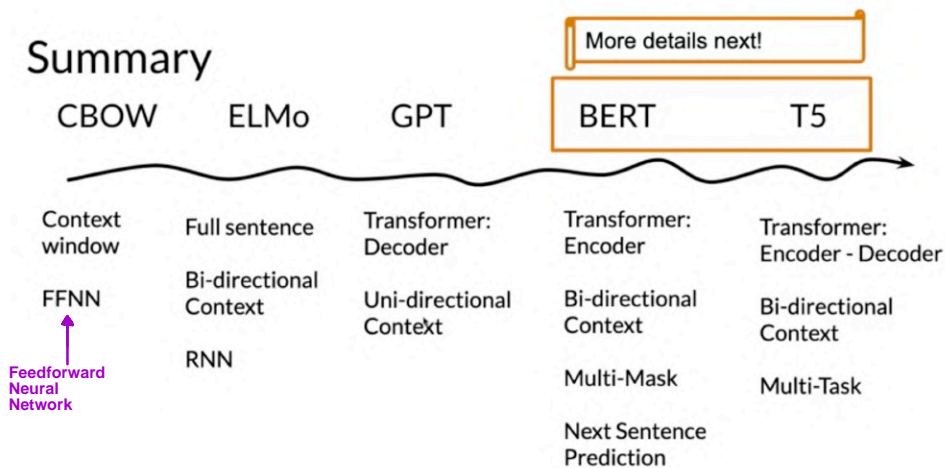


## T5: Text-to-Text

So T5 is all text to text. So what we can do is append a tag on the left, such as classify or summarize, and it'll carry out the given task. So basically what T5 does is that it allows you to append a label, note that these are not the corresponding labels but are just for simplicity, and basically you can append a label and the model will handle the output.



## Summary



## Outline

- Learn about the BERT architecture
- Understand how BERT pre-training works

## BERT

- Makes use of transfer learning/pre-training:

BERT is the bidirectional encoder representation from transformers and it makes use of transfer learning and pre-training. How does this work?

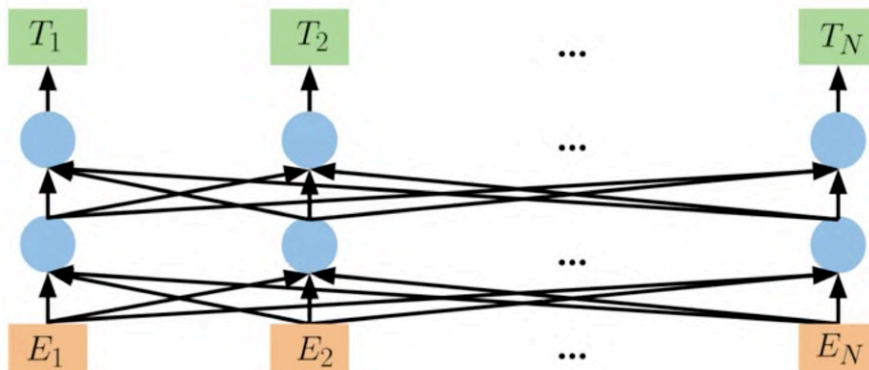
Usually starts with some inputs embedding so  $E_1, E_2$ , all the way to some random number,  $E_N$ , and then, you go through some transformer blocks, as you can see here. Each blue circle is a transformer block, goes up furthermore,



## BERT

Then, you get your  $T_1, T_2, T_N$ . Basically, there are two steps in BERT's framework, pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks as you've already seen before, and for fine-tuning, the BERT model is first initialized with the pre-trained parameters and all of the parameters are fine-tuned using labeled data from the downstream tasks. For example, in the figure over here, you get the corresponding embeddings. You went through a few transformer blocks, and then, you make the prediction.

- Makes use of transfer learning/pre-training:



## BERT

- A multi layer bidirectional transformer
- Positional embeddings
- BERT\_base:

12 layers (12 transformer blocks)

12 attentions heads

110 million parameters

The famous model is BERT base which has 12 layers or 12 transformer blocks, 12 attention heads, and 110 million parameters, and this new models that are coming out now like GPT-3 and so forth. They have way, way more parameters and way, way more blocks and layers.

## BERT pre-training

After school Lukasz does his \_\_\_\_\_ in the library.

Let's talk about pre-training. Before feeding the word sequences to the BERT model, we mask 15% of the words, and then, the training data generator chooses 15% of these positions at random for prediction. Then, if the  $i$ th token is chosen, we replace the  $i$ th token with one, the mask token 80% of the time, and then, two, a random token 10% of the time, and then, three, the unchanging  $i$ th token 10% of the time.

- Masked language modeling (MLM)

In this case, then  $T_i$ , which you've seen in the previous slide, will be used to predict the original token with cross-entropy loss. In this case, this is known as the masked language model. Over here, we have "After school Lukasz does his blank in the library." Maybe work, maybe homework. One of these words that your BERT's model is going to try to predict.

To do so, usually, what you do, you just add the dense layer after the  $T_i$  token and use it to classify after the encoder outputs. You just multiply the output's vectors by the embedding matrix, and then, to transform them into vocabulary dimension and you add a softmax at the end.

## BERT pre-training

After school Lukasz does his homework in the library.

After school \_\_\_\_\_ his homework in the \_\_\_\_\_ .

## Summary

- Choose 15% of the tokens at random: mask them 80% of the time, replace them with a random token 10% of the time, or keep as is 10% of the time.
- There could be multiple masked spans in a sentence
- Next sentence prediction is also used when pre-training.

## Outline

- Understand how BERT inputs are fed into the model
- Visualize the output
- Learn about the BERT objective

You'll see what you are trying to minimize. Specifically, I'll show you how you can combine word embeddings, sentence embeddings, and positional embeddings as inputs.

## Formalizing the input

Formalizing the inputs, this is the Bert's input representation. You start with positional embedding, so they allow you to indicate the position in the sentence of the word. Where each word is in the corresponding sentence.

Position  
Embeddings

|                |                |                |                |                |                |                |                |                |                |                 |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-----------------|
| E <sub>0</sub> | E <sub>1</sub> | E <sub>2</sub> | E <sub>3</sub> | E <sub>4</sub> | E <sub>5</sub> | E <sub>6</sub> | E <sub>7</sub> | E <sub>8</sub> | E <sub>9</sub> | E <sub>10</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|-----------------|

## Formalizing the input

You have this, then you have the segment embeddings. They allow you to indicate whether it's a sentence A or sentence B. Remember, in BERT we also use next census prediction.

|                     |       |       |       |       |       |       |       |       |       |       |          |
|---------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| Segment Embeddings  | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_A$ | $E_B$ | $E_B$ | $E_B$ | $E_B$ | $E_B$    |
| Position Embeddings | $E_0$ | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | $E_7$ | $E_8$ | $E_9$ | $E_{10}$ |

## Formalizing the input

Then you have the token embeddings or the inputs embeddings. You also have a CLS token, which is used to indicate the beginning of the sentence and accept token, which is used to indicate the end of the sentence.

|                     |             |          |           |          |            |             |          |             |            |             |             |
|---------------------|-------------|----------|-----------|----------|------------|-------------|----------|-------------|------------|-------------|-------------|
| Token Embeddings    | $E_{[CLS]}$ | $E_{my}$ | $E_{dog}$ | $E_{is}$ | $E_{cute}$ | $E_{[SEP]}$ | $E_{he}$ | $E_{likes}$ | $E_{play}$ | $E_{\#ing}$ | $E_{[SEP]}$ |
| Segment Embeddings  | $E_A$       | $E_A$    | $E_A$     | $E_A$    | $E_A$      | $E_A$       | $E_B$    | $E_B$       | $E_B$      | $E_B$       | $E_B$       |
| Position Embeddings | $E_0$       | $E_1$    | $E_2$     | $E_3$    | $E_4$      | $E_5$       | $E_6$    | $E_7$       | $E_8$      | $E_9$       | $E_{10}$    |

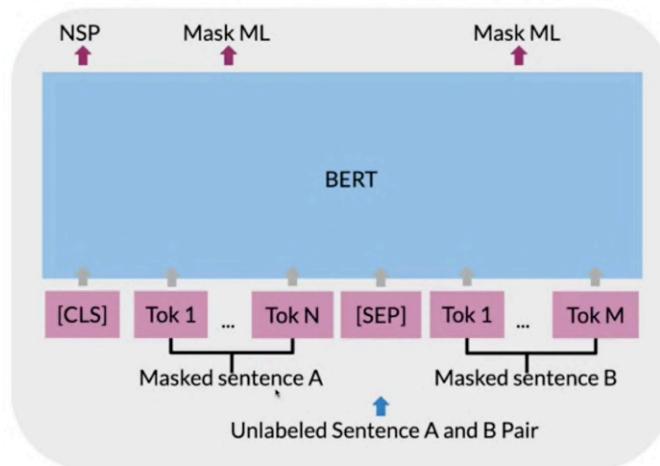
## Formalizing the input

What you do, is just take the sum of the token embeddings, the segmentation embeddings, and the position embeddings, and then you get your new inputs.

|                     |             |          |           |          |            |             |          |             |            |             |             |
|---------------------|-------------|----------|-----------|----------|------------|-------------|----------|-------------|------------|-------------|-------------|
| Input               | $[CLS]$     | $my$     | $dog$     | $is$     | $cute$     | $[SEP]$     | $he$     | $likes$     | $play$     | $\#ing$     | $[SEP]$     |
| Token Embeddings    | $E_{[CLS]}$ | $E_{my}$ | $E_{dog}$ | $E_{is}$ | $E_{cute}$ | $E_{[SEP]}$ | $E_{he}$ | $E_{likes}$ | $E_{play}$ | $E_{\#ing}$ | $E_{[SEP]}$ |
| Segment Embeddings  | $E_A$       | $E_A$    | $E_A$     | $E_A$    | $E_A$      | $E_A$       | $E_B$    | $E_B$       | $E_B$      | $E_B$       | $E_B$       |
| Position Embeddings | $E_0$       | $E_1$    | $E_2$     | $E_3$    | $E_4$      | $E_5$       | $E_6$    | $E_7$       | $E_8$      | $E_9$       | $E_{10}$    |

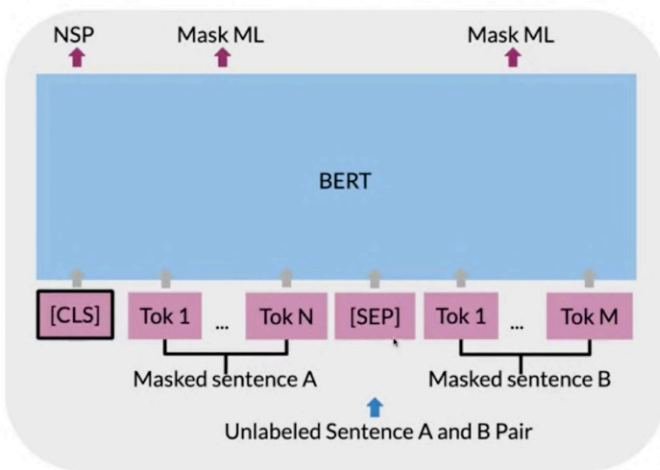


## Visualizing the output



Over here you can see you have masked sentence A. You have must sentence B, they go into tokens.

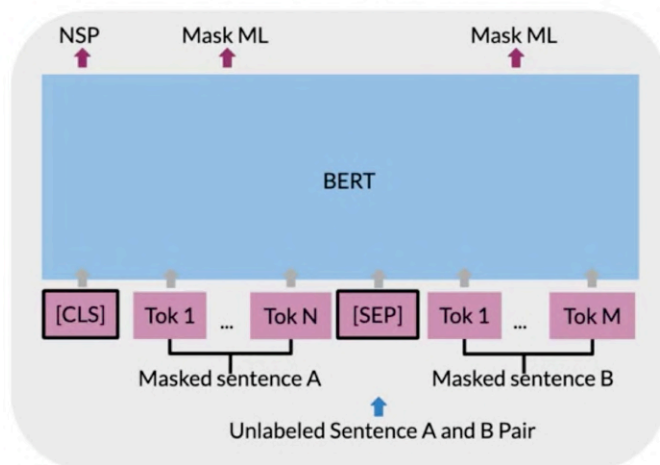
## Visualizing the output



- **[CLS]:** a special classification symbol added in front of every input

Then you have the CLS token, which is a special classification symbol added in front of every input.

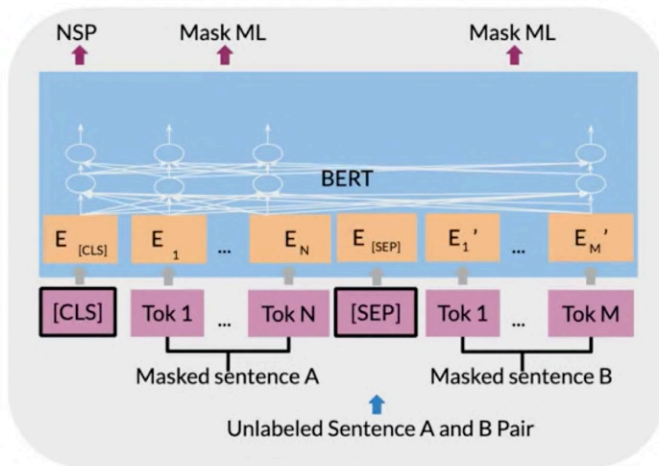
## Visualizing the output



- **[CLS]:** a special classification symbol added in front of every input
- **[SEP]:** a special separator token

Then you have the sub-token, which is the special separator token.

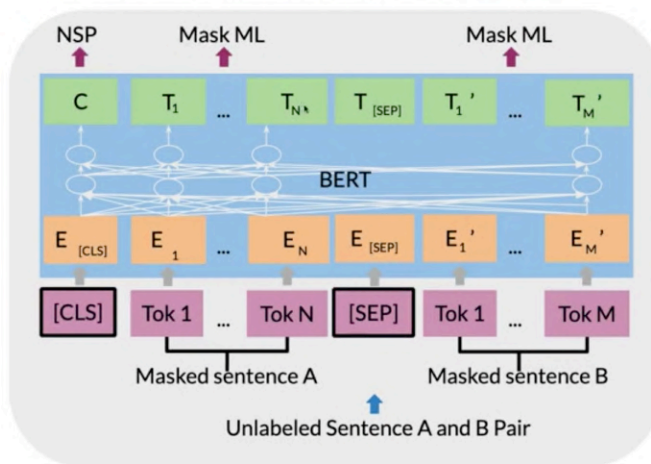
## Visualizing the output



- **[CLS]:** a special classification symbol added in front of every input
- **[SEP]:** a special separator token

You convert them into the embeddings

## Visualizing the output



- **[CLS]:** a special classification symbol added in front of every input
- **[SEP]:** a special separator token

Then you get your transformer blocks. Then you can see at the end you get the your  $T_1$  to  $T_N$ . Your  $T_1'$ ,  $T_M'$  prime

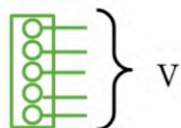
These are each  $T_i$  embedding will be used to predict the mask toward VI simple soft max.

You have this  $c$  also embedding which can be used for next sentence prediction.

## BERT Objective

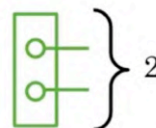
Objective 1:  
Multi-Mask LM

Loss: Cross Entropy Loss



Objective 2:  
Next Sentence Prediction

Loss: Binary Loss



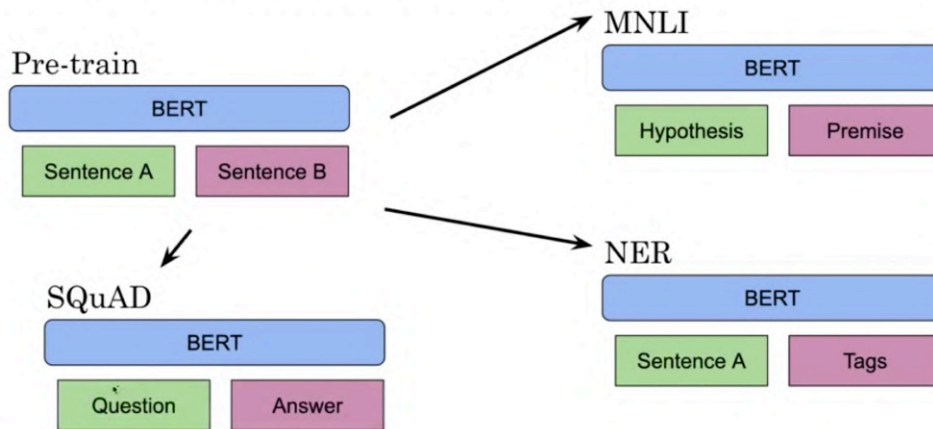
Let's look at the BERT objective. For the multi-mask language model, use a cross entropy loss to predict the word that's being masked or the words that are being masked. Then you add this to a binary loss for the next sentence predictions. Given the two sentences to the follow one another or not.

## Summary

- BERT objective
- Model inputs/outputs

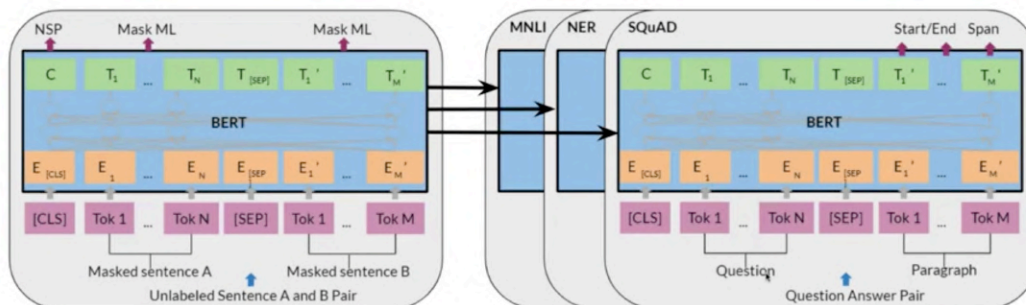
Right now you're going to see how you fine-tune BERT. During pre-training, remember, you had sentence A and sentence B, and then you use next sentence prediction and use mask tokens to predict the mask tokens that you mask from each sentence, that's in pre-training. Now, if you want to go on to MNLI or like hypothesis premise scenario, then instead of having sentence A and sentence B, you're going to feed in the hypothesis over here, and the premise over here. For NER you're going to feed the sentence A over here, and then the corresponding tags over here. For question answering, you'll have SQUAD. For example, you'll have your question over here and then your answer over here.

## Fine-tuning BERT: Outline

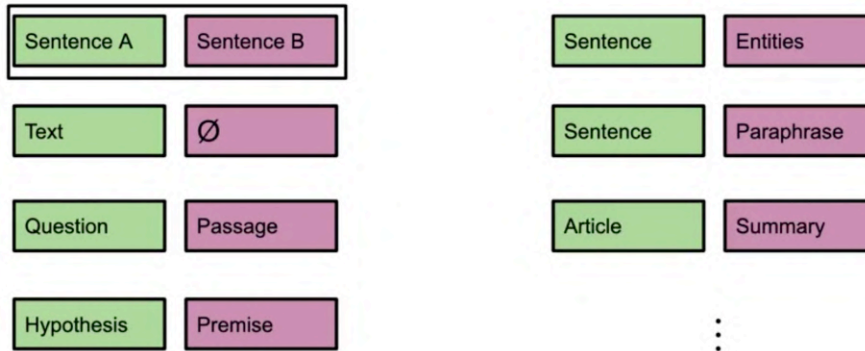


## Inputs

Visually, what does this look like? Remember this image from the BERT's paper, this is what ends up happening over here you have the question. Over here, you'll have the paragraph and this will give you like your answer. This starts in the end of the answer. Then for NER, again, you'll have the sentence and the corresponding named entities. For MNLI, you will have the hypothesis and then the premise and so forth.



## Summary



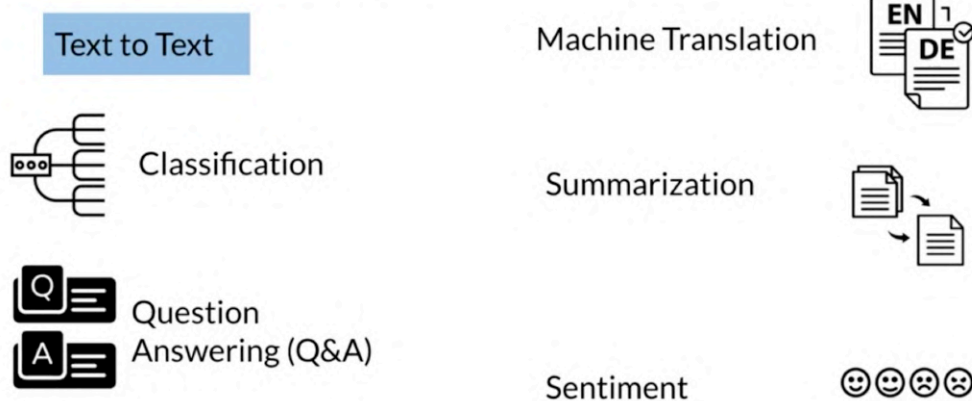
In summary given the place of sentence A and sentence B, you can fill it with a text and parts for sentence A and then say like an all symbol to say that you're trying to classify the text, whether it's for sentiment analysis, like happy or sad, so that's one way to do it. Question passage for question answering, you could have hypothesis premise for MNLI. You could have sentence with the named entities for NER. You can have a sentence and a paraphrase of the sentence. You could have an article, the summary and so forth. This is just the inputs into your BERT's model.

## Outline

- Understand how T5 works
- Recognize the different types of attention used
- Overview of model architecture

I'll now be talking about the T5 model. The T5 model could be used on several NLP tasks. And it's also uses a similar training strategy to birds. Concretely it makes use of transfer learning and mask language modeling. The T5 model also uses transformers when training.

## Transformer - T5 Model



## Transformer - T5 Model

Original text

Thank you for inviting me to your party last week.

So the model architecture and the way the pre-training is done first of all is you have an original text, something like this where you have, thank you for inviting me to your party last week.

## Transformer - T5 Model

Original text

Thank you for inviting me to your party last week.

Inputs

Thank you <X> me to your party <Y> week.

So you mask this certain words so for inviting me last, and then you replace it with these tokens like brackets X brackets Y. So brackets X corresponding to for inviting and brackets Y corresponding to last.

## Transformer - T5 Model

Original text

Thank you for inviting me to your party last week.

Inputs

Thank you <X> me to your party <Y> week.

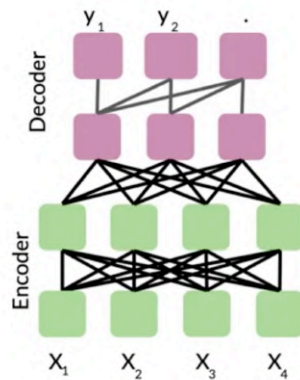
Targets

<X> for inviting <Y> last <Z>

And your targets are going to be brackets X for inviting, brackets Y last and these tokens or these brackets like they keep going in increments order. So then it's bracket Z, then maybe brackets A brackets B and so forth. So each bracket corresponds to a certain target.

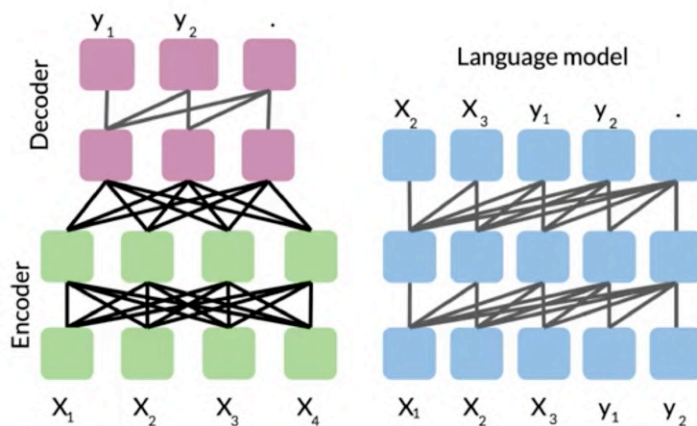


## Model Architecture



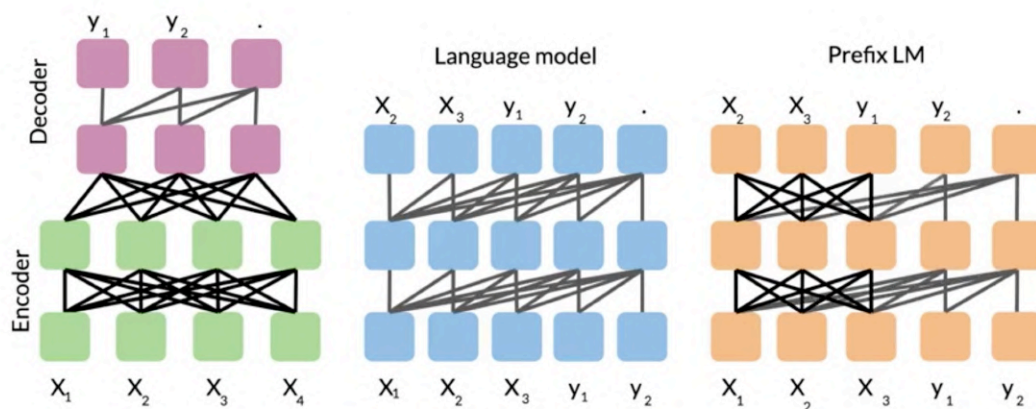
The model architecture and different transformer architecture variance that we're going to consider for the attention part here. So we start with the basic encoder-decoder representation. So you can see over here you have fully visible attention in the encoder and then causal attention in the decoder. And then you have the general encoder-decoder representation just as notation. So light gray lines correspond to causal masking. And dark gray lines correspond to the fully visible masking.

## Model Architecture



So on the left as I said again, it's the standard encoder-decoder architecture. In the middle over here what we have, we have the language model which consists of a single transformer layer stack. And it's being fed the concatenation of the inputs and the target. So it uses causal masking throughout as you can see because they're all gray lines. And you have X1 going inside over here, get at X2, X2 goes into the model X3 and so forth.

## Model Architecture



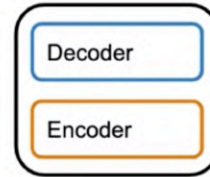
Now over here to the right, we have prefix language model which corresponds to allowing fully visible masking over the inputs as you can see here in the dark arrows. And then causal masking in the rest.



So the model architecture, it uses encoder/decoder stack. It has 12 transformer blocks each. So you can think of it as a dozen eggs and then 220 million parameters.

## Model Architecture

- Encoder/decoder
- 12 transformer blocks each



## Summary

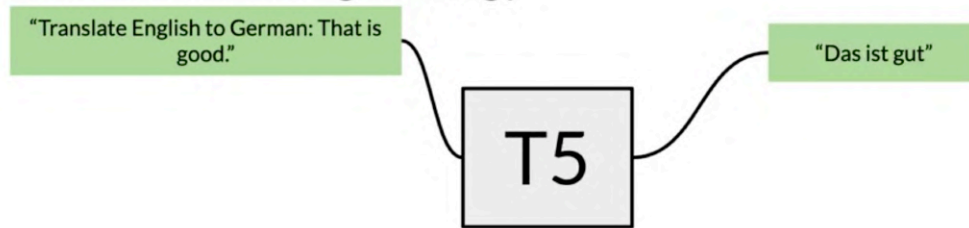
- Prefix LM attention
- Model architecture
- Pre-training T5 (MLM)

You've seen the model architecture for T5. And you've seen how the pre-training is done similar to BERT, but we just use mask language modeling here. You now have an overview of the transformer 5 model. You know how to train it, and you've seen that you can use it on multiple tasks.

You now have an overview of the transformer 5 model. You know how to train it, and you've seen that you can use it on multiple tasks.

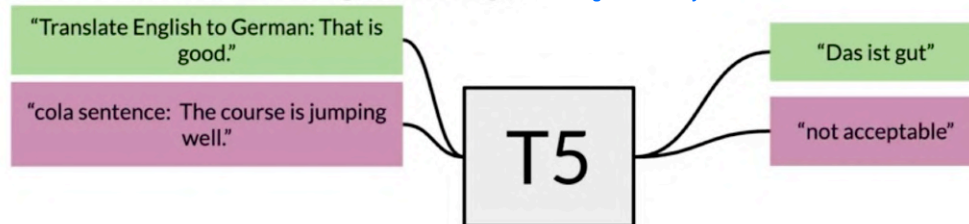
## Multi-task training strategy

You now have an overview of the transformer 5 model. You know how to train it, and you've seen that you can use it on multiple tasks.



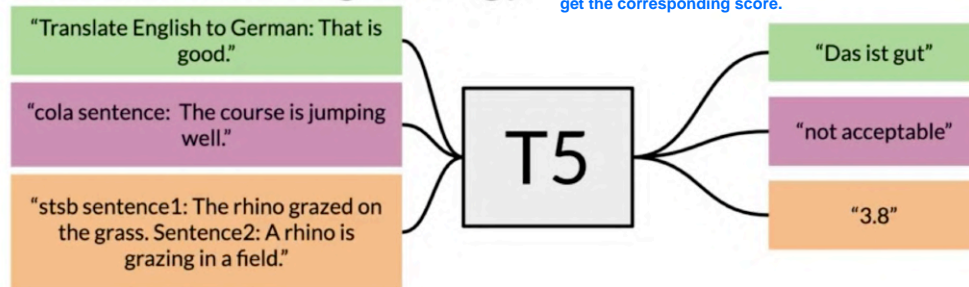
## Multi-task training strategy

For a cola sentence like "The course is jumping well", and it says it's not acceptable because it's grammatically incorrect.



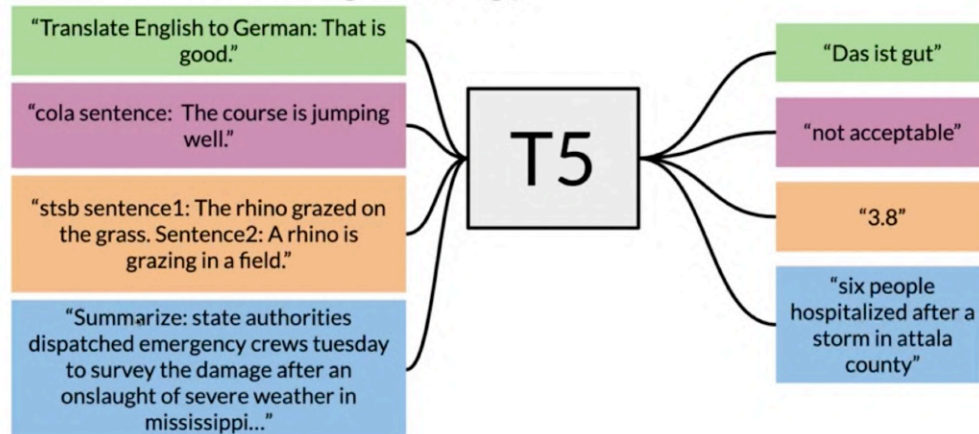
## Multi-task training strategy

If you have two sentences and you want to identify their similarity, you put in the "stsb sentence1" and then sentence2 inside over here, sentence 1, sentence 2. Then you get the corresponding score.



## Multi-task training strategy

If you wanted to summarize, you add the summarize prefix to the article or the text you want to summarize, and it gives you the summary.



This is how it works. Input and output format, so for machine translation you just do translate blank to blank, and you add the sentence.

To predict entailment, contradiction, or whether it's neutral, you would feed in something as follows, so mnli premise: I hates pigeons. Then the hypothesis: My feelings towards pigeons are filled with animosity, and the target is entailment. Basically over here, this is going to try to learn the overall structure of entailment. By feeding in the entire thing, the model would have full visibility over the entire input and then it would be tasked with marking a classification by us putting the word "entailment". It is easy for the model to learn to predict one of the correct class labels given the task prefix mnli in this case. If you know the main difference between prefix lm and the birth architecture is that the classifier is integrated to the output layer of the transformer decoder and the prefix lm.

Over here you have the Winograd schema, which is another way to predict whether a pronoun, for example over here, "The City councilmen refused the demonstrators a permit because they feared violence." You're going to feed this into your model and then it will be tasked to predict "they" as the city councilmen.

## Input and Output Format

Machine translation:

- translate English to German: That is good.
- Predict entailment, contradiction, or neutral
- mnli premise: I hate pigeons hypothesis: My feelings towards pigeons are filled with animosity. target: entailment
- Winograd schema
- The city councilmen refused the demonstrators a permit because "they" feared violence

Source: <https://arxiv.org/pdf/1808.08769v1.pdf>

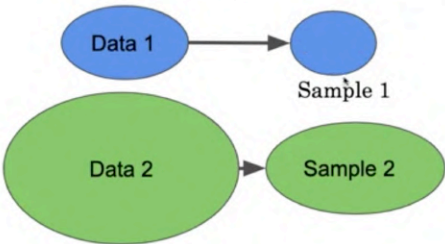
For multi-task training strategy, this is a table found in the original paper, and we'll talk about what the GLUE benchmark is and these other benchmarks, you can check them out. But for the purpose of this week, we'll be focusing on the GLUE benchmark, which will be the next video. We'll talk about adapter layers and gradual unfreezing. But these are the scores reported. You can see that the T5 paper actually reaches states of the art in many tasks. How much data from each task to train on?

## Multi-task Training Strategy

| Fine-tuning method         | GLUE         | CNNDM        | SQuAD        | SGLUE        | EnDe         | EnFr         | EnRo         |
|----------------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| * All parameters           | <b>83.28</b> | <b>19.24</b> | <b>80.88</b> | <b>71.36</b> | <b>26.98</b> | <b>39.82</b> | <b>27.65</b> |
| Adapter layers, $d = 32$   | 80.52        | 15.08        | 79.32        | 60.40        | 13.84        | 17.88        | 15.54        |
| Adapter layers, $d = 128$  | 81.51        | 16.62        | 79.47        | 63.03        | 19.83        | 27.50        | 22.63        |
| Adapter layers, $d = 512$  | 81.54        | 17.78        | 79.18        | 64.30        | 23.45        | 33.98        | 25.81        |
| Adapter layers, $d = 2048$ | 81.51        | 16.62        | 79.47        | 63.03        | 19.83        | 27.50        | 22.63        |
| Gradual unfreezing         | 82.50        | 18.95        | 79.17        | <b>70.79</b> | 26.71        | 39.02        | 26.93        |

## Data Training Strategies

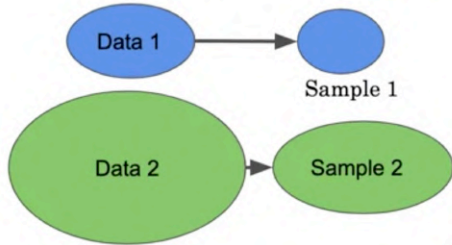
Examples-proportional mixing



For the data training strategies, there's examples, proportional mixing, and in this case, what you end up doing, you take an equal proportion, say like 10 percent from each data that you have. If the first data set for example blue, you take 10 percent of this, then you will get 10 percent over here, 10 percent of this is larger. Ten percent is just a random number I picked but you get the points.

## Data Training Strategies

Examples-proportional mixing

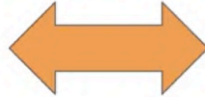


For the other type of data training strategy is equal mixing. Regardless of the size of each data, you take an equal sample. Then there is something in the middle called temperature-scaled mixing, where you try to play with the parameters to get something in-between.

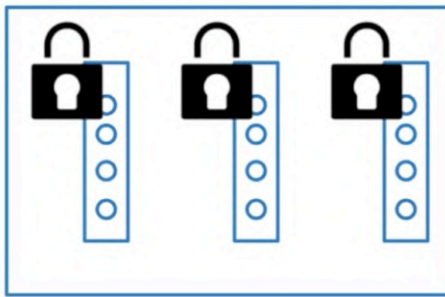
Equal mixing



Temperature-scaled mixing



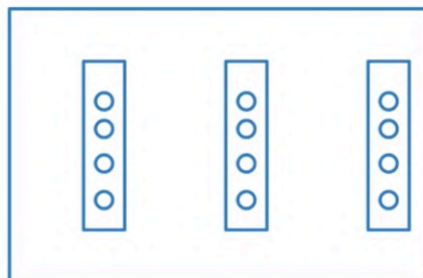
## Gradual unfreezing vs. Adapter layers



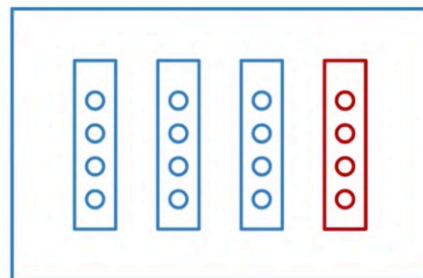
Now we'll talk about gradual unfreezing versus adapter layers. In gradual unfreezing, what ends up happening, you freeze one layer at a time. You say this is your neural network, you freeze the last one, you fine-tune using that you keep the others fixed. Then you freeze this one and then you freeze this one, so you keep gradually unfreezing each layer.

Gradual unfreezing

## Gradual unfreezing vs. Adapter layers



Gradual unfreezing

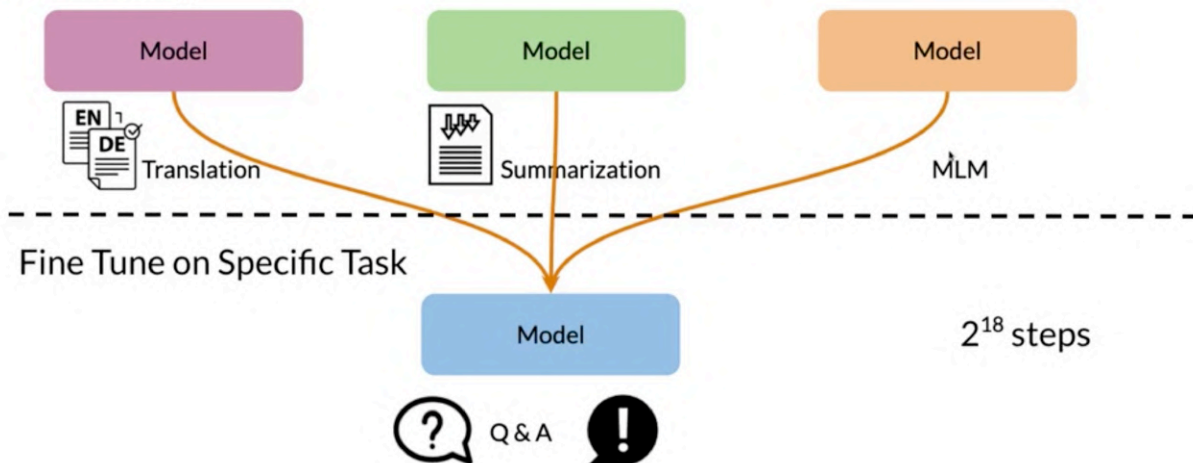


Adapter layers

For the adapter layers, you basically add an existing neural network or you add a neural network to each feed forward and each block of the transformer. Then these new feed forward networks, they're designed so that the output dimension matches the input. This allows them to be inserted without having any structural change. When fine-tuning, only these new adapter layers and the layer normalization parameters are being updated.

# Fine-tuning

## Pre Training



## GLUE

### General Language Understanding Evaluation

- A collection used to train, evaluate, analyze natural language understanding systems
- Datasets with different genres, and of different sizes and difficulties
- Leaderboard

So, the glue benchmark stands for general language understanding evaluation. And it is basically collection that is used to train, evaluate, analyze natural language understanding systems. It has a lot of data sets, and each data set has several genres, and they have different sizes and different difficulties. Some of them are for example, used for coreference resolution. Others are just used for simple sentiment analysis. Others are used for question answering and so forth. And it is used with a leaderboard. So people can use the data sets, and see how well their models perform compared to others.

So the task is evaluated on could be, for example, sentence, grammatical or not. For example, if a sentence makes sense or it does not, it could be used on sentiments. It could be used to paraphrase some texts. It could be used on similarity on question duplicates. Weather question is answerable or not, whether it's a contradiction, whether it's entailment. And also for the Winograd Schema, which is basically trying to identify whether, a pronoun refers to a certain noun or to another noun. So it's used to drive research.

## Tasks Evaluated on

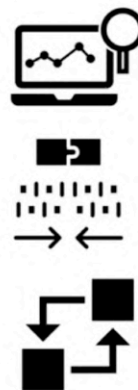
- Sentence grammatical or not?
- Sentiment
- Paraphrase
- Similarity
- Questions duplicates
- Answerable
- Contradiction
- Entailment
- Winograd (co-ref)



As I said, that researchers usually use the glue as a benchmark. It is also model agnostic, so it doesn't matter which model you use. It just evaluates on glue and see how well your model performs. And finally, it allows you to make use of transfer learning, because you have access to several datasets. And you can learn certain things from different datasets, that will help you evaluate on a completely new datasets within Google.

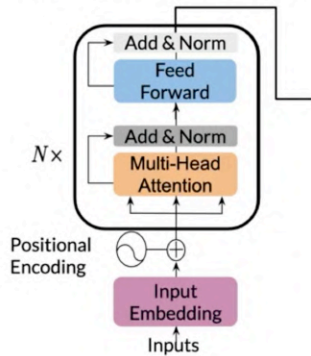
## General Language Understanding Evaluation

- Drive research
- Model agnostic
- Makes use of transfer learning





## Transformer encoder

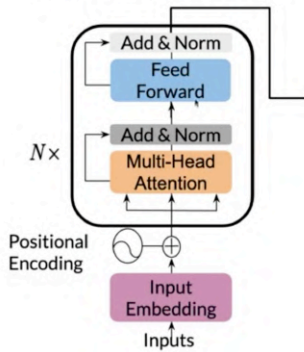


### Feedforward:

```
[
    LayerNorm,
    dense,
    activation,
    dropout_middle,
    dense,
    dropout_final
]
```

So previously you've seen the transformer decoder and now you're going to look at the transformer encoder so it's very similar. You have the inputs embeddings that are going in, then you have positional encoding, you have multi-attention head, add a normalization, then a feed forward, then other add and normalization and then this goes into the decoder architecture. So, right now, we will be looking at the feed forward, so you have a layerNorm, a dense, followed by an activation. Then we have a dropouts in the middle, followed by a dense layer, and then a dropouts final. So this is what you'll be implementing in this weeks programming assignments. Then you have the encoder block.

## Transformer encoder

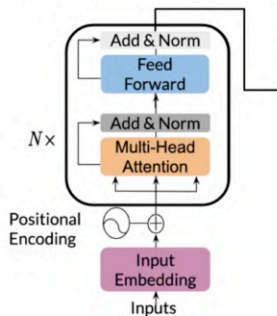


### Encoder block:

```
[
    Residual(
        LayerNorm,
        attention,
        dropout_,
    ),
    Residual(
        feed_forward,
    ),
]
```

Then you have the encoder block. So the encoder block has a layer norm, has the attention, then the dropouts, and then it has another residual connection within a feed forward. So this is the code for this image over here.

## Transformer encoder



### Feedforward:

```
[
    LayerNorm,
    dense,
    activation,
    dropout_middle,
    dense,
    dropout_final
]
```

### Encoder block:

```
[
    Residual(
        LayerNorm,
        attention,
        dropout_,
    ),
    Residual(
        feed_forward,
    )
]
```

So putting them together, you have the feed forward here and then you have the encoder block which has the feed forward and two residual connections. So the first one and the second one here.

## Data examples

Now, the other thing you will be looking at in this weeks programming assignment is the datasets. So, you'll have a question like this, like what percentage of the French population today is non-European. You'll have a context and then you'll have a target. So the answer is approximately 5%, which could be found here.

**Question:** What percentage of the French population today is non - European ?

**Context:** Since the end of the Second World War , France has become an ethnically diverse country . Today , **approximately five percent** of the French population is non - European and non - white . This does not approach the number of non - white citizens in the United States ( roughly 28 – 37 % , depending on how Latinos are classified ; see Demographics of the United States ) . Nevertheless , it amounts to at least three million people , and has forced the issues of ethnic diversity onto the French policy agenda . France has developed an approach to dealing with ethnic problems that stands in contrast to that of many advanced , industrialized countries . Unlike the United States , Britain , or even the Netherlands , France maintains a " color - blind " model of public policy . This means that it targets virtually no policies directly at racial or ethnic groups . Instead , it uses geographic or class criteria to address issues of social inequalities . It has , however , developed an extensive anti - racist policy repertoire since the early 1970s . Until recently , French policies focused primarily on issues of hate speech — going much further than their American counterparts — and relatively less on issues of discrimination in jobs , housing , and in provision of goods and services .

**Target:** **Approximately five percent**

## Implementing Q&A with T5

So here is exactly what you'll be doing. You'll be loading in a pre-trained model and then you're going to process your data set. Such that you'll have question: followed by the question. Contexts: followed by the context as inputs, and A as targets for answer. Then you will find soon your model on the new task and the inputs, and then you predict using your own model. And remember this is the T5 that you'll be working with.

- Load a pre-trained model
- Process data to get the required inputs and outputs: "question: Q context: C" as input and "A" as target
- Fine tune your model on the new task and input
- Predict using your own model

