Introduction to Large Language Models (LLMs)

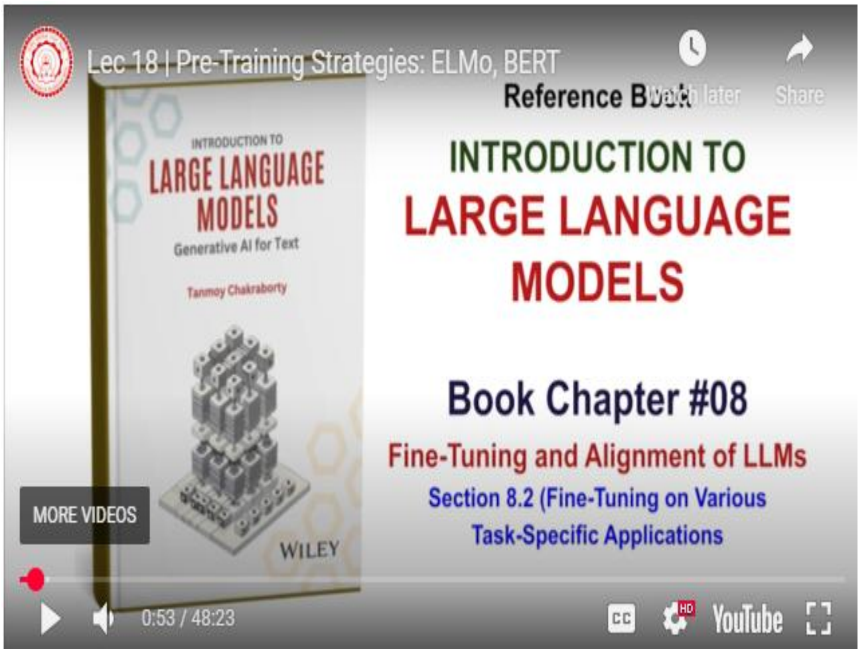
Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 18

Pre-Training Strategies: ELMo, BERT



Lec 18 | Pre-Training Strategies: ELMo, BERT

Reference Book

INTRODUCTION TO

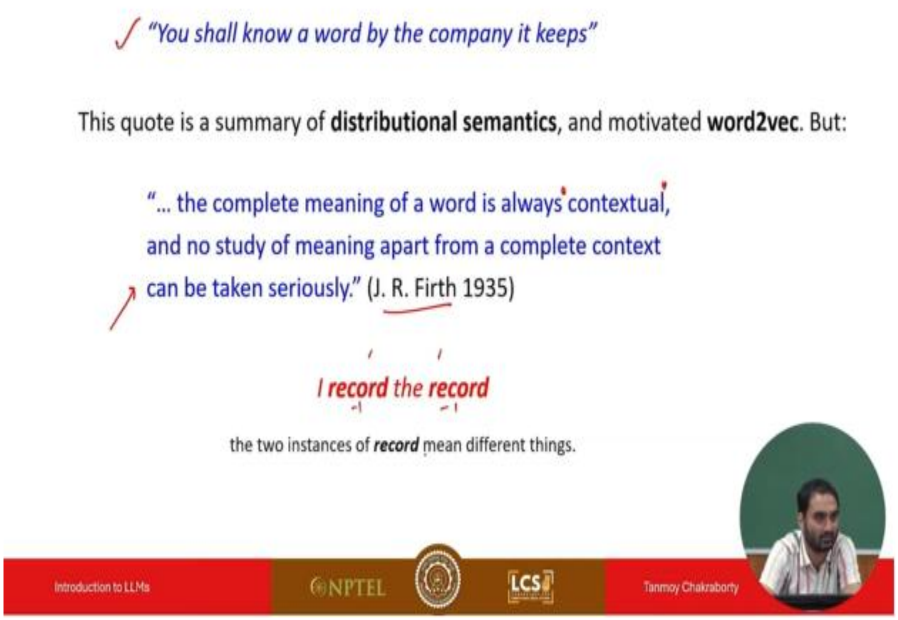
LARGE LANGUAGE MODELS

Book Chapter #08

Fine-Tuning and Alignment of LLMs

Section 8.2 (Fine-Tuning on Various Task-Specific Applications

Hello everyone, welcome to this course again. So in this lecture we are going to talk about pre-training strategies. Last lecture we covered transformer, different blocks of a transformer model, right. We also talked about what to wear glove models which are pre-trained, what embedding methods, right. We will see the overall arena of pre-training with transformer and then we will also see how the paradigm of pre-training actually changed from pre-trained word embedding to pre-trained language model.



"You shall know a word by the company it keeps"

This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

"... the complete meaning of a word is always contextual, and no study of meaning apart from a complete context can be taken seriously." (J. R. Firth 1935)

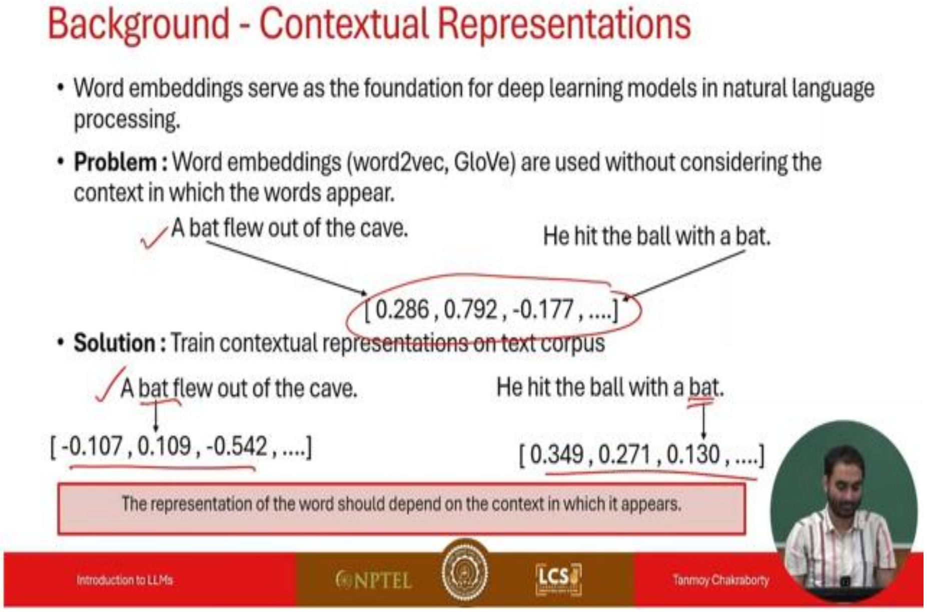
I record the record

the two instances of ***record*** mean different things.

Let us look at the pre-training strategy.

So while discussing this, let us start with this famous quote. which we already mentioned earlier in the distributional semantics chapter, in the word embedding chapter that "you shall know a word by the company it keeps", right. So the same person modified the quote later on and then he said that the complete meaning of a word is always contextual. okay this is always contextual and no study of meaning apart from a complete context can be taken seriously okay and this is essentially the foundation behind building the pre-training strategies okay.

This is an example, I record the record right. If you look at these two, the two positions of the word record right, the meanings are completely different. Now if you use a word to way kind of method or a glove kind of method which basically produce pre-trained embeddings, right? You will get same embedding for these two positions of record, right? But the meanings are different. Okay.



Background - Contextual Representations

- Word embeddings serve as the foundation for deep learning models in natural language processing.
- **Problem :** Word embeddings (word2vec, GloVe) are used without considering the context in which the words appear.

A bat flew out of the cave.

He hit the ball with a bat.

[0.286 , 0.792 , -0.177 , ...]

- **Solution :** Train contextual representations on text corpus

A bat flew out of the cave.

[-0.107 , 0.109 , -0.542 , ...]

He hit the ball with a bat.

[0.349 , 0.271 , 0.130 , ...]

The representation of the word should depend on the context in which it appears.

Introduction to LLMs

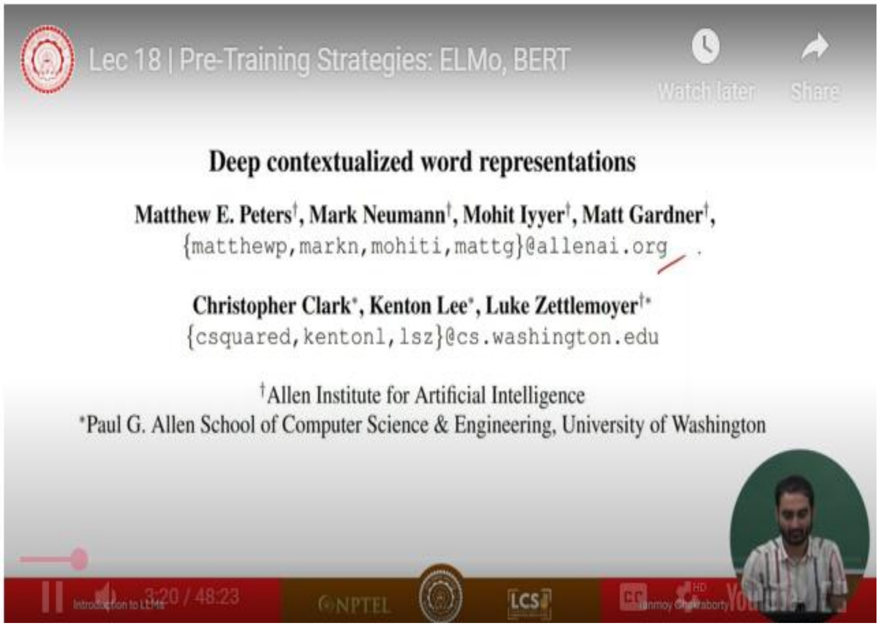
NPTEL

LCS

Tanmoy Chakraborty

Let's take another example.

A bat flew out of the cave. He hit the ball with the bat. Again, here the two words bat, the two positions of the word bat, the two contexts of the word bat are different, right? If you use the same embedding to represent this, you will not be able to do the proper processing of it, right? So what we will do here? We will produce embeddings what embeddings which are going to be contextualized meaning depending on the context the embedding will change. For example here let us say for the word bat you will get one embedding for the word bat here you get another embedding which are different okay. How do you do this?

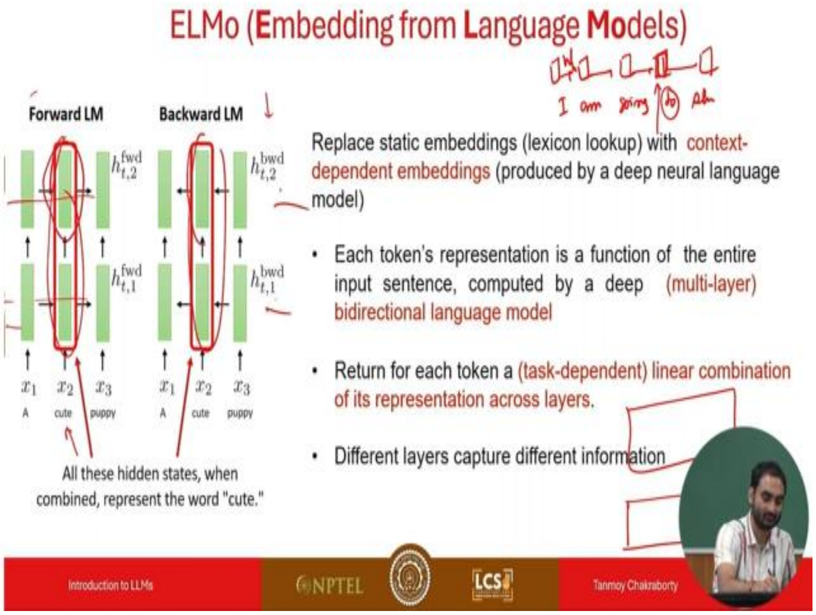


Deep contextualized word representations

Matthew E. Peters[†], Mark Neumann[†], Mohit Iyyer[†], Matt Gardner[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], Kenton Lee[*], Luke Zettlemoyer[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington



ELMo (Embedding from Language Models)

All these hidden states, when combined, represent the word "cute."

Replace static embeddings (lexicon lookup) with context-dependent embeddings (produced by a deep neural language model)

- Each token's representation is a function of the entire input sentence, computed by a deep (multi-layer) bidirectional language model
- Return for each token a (task-dependent) linear combination of its representation across layers.
- Different layers capture different information

The first approach which was proposed which is not a transformer based approach this was proposed in 2018 is called ELMO.

Deep contextualized word representation. This was done mostly by LNAI team and some folks from this university also. One of them is a very good friend of mine. So the idea behind the ELMO method is as follows. So ELMO is a non-transformer based approach.

There is no concept of transformer. Transformers was introduced in 2017, around the same time it was introduced, 2018. So ELMO stands for embedding from language models. What it does? It essentially relies on RNNs can be LSTM can be GRU right RNNs and what it does, it essentially processes a sequence right when you process a sequence using RNN each of the hidden states which correspond to basically the tokens, right? This hidden states will basically be used as embeddings, right? Let's say I am going to school. This is my RNN, okay? In RNN, what do we learn? We learn the weight matrix W.

Right there are how many matrices are there in general? Generally there are three matrices one is a matrix between layers right, wh another is this we which corresponds to the input, and other is this u which basically projects it to the output space right, for the time being you just assume that there is only one matrix which is wh. Okay so you have an RNN, a fixed length RNN, right? And then what you do, you pass one training instance, okay? And you train this RNN using language model objective, next word prediction, right? You pass one sequence, you predict the first position of the sequence, second position, third position, fourth position, and so on and so forth, right? You do backdrop and then you update WH, right? Then again you pass another sequence, you update this and so on and so forth. Now think about it, what is happening here? So the claim is that the hidden state will act as an embedding. The hidden state here, let's say this hidden state will act as an embedding for the word to. Now you may wonder how would it be possible because there are so many sequences, there is only one RNN.

So let's think of a dictionary where every row indicates an embedding of a particular token. Let's assume that this is a WAD level token, not a character level WAD level tokenization. So there is an entry I and this is that vector, entry J this is a vector and so on and so forth and let us assume that this matrix is initially randomly initialized okay. Now when you start this RNN you need to feed the input right, this input is fetched from the dictionary. So when the input is I you basically fetch the corresponding embedding vector from the dictionary and you feed it as an input.

Right to start with, then you train the model, let's say you take one instance you train the model, your hidden states are updated right, the hidden state corresponding to the word i is also updated right. So you when the training process is done you take that hidden state and you replace the previous embedding of i by this hidden state done. Another instance, another training sentence comes in you use the same RNN okay. Let's say this is she is eating a pizza okay. So what you do first how do you initialize this? You take the embedding of she from the dictionary, is right, eating and so on and so forth right you already have this weight matrix which came from the previous iteration you again update You update the hidden states and the corresponding dictionary is updated.

You keep on doing this thing with all these instances. Now you can say that, you know, I will train the model using batches. That is also possible. You take a batch and then you do this update or let's say per instance, you basically update this RNN.

Okay. So this is a language model objective based RNN model. There is no task involved so far. Now, so this is, now what you do, so this is your first RNN, which is a language model of the TBH RNN. When all these things are done, you freeze this RNN. When you freeze this, what does it mean? It means all the embeddings are freezed, right? And now let's say you are interested in doing some task.

Okay. And let's say you have some task specific data. You have some task specific data which you want to use to train the model further. Some sort of fine tuning. So on top of this, you apply another RNN which will be responsible for doing this task. Now let's say the task is sentiment analysis.

This is sentiment analysis. So this is frozen. It is frozen means the hidden states are coming from this, the first layer. So given the task specific input, all the tokens embeddings are coming from this layer, which are frozen, right? And here in the second layer, you have another weight matrix, WH dash, which corresponds to that layer. And you have the task, you produce some output, there's some error, right? You back prop and you update the weights.

The task specific weights are updated. So during the inference time, let's see what given a sentence, right? What do you do first? You're given a sentence, then you first retrieve. Now remember, there are two layers. Don't forget there are two layers. The first layer is a language model layer.

The second layer is a task specific layer, right? During the inference time, what you do? During the inference time, you feed here, right? You basically feed here. The bottom layer, the corresponding weight wh is already stored it has already been learned right. So let's say you are feeding a new sentence. Let's say Akshay is a great cricket player, okay So what you do so this is my first layer okay, so you feed the input the input vector of Akshay from the dictionary right, you multiply this with the weight wh, right, it will and then akshay is then you also have this is embedding for the word is this and the previous hidden state will basically be used to generate the next hidden state, right and so on and so forth you roll out. Okay when you roll out, now let's look at the current hidden states now, now these hidden states, the current hidden states of the bottom layer, are very different from the hidden states which are stored earlier, because these are multiplied by these are basically functions of the weight matrix right and this is contextual.

Why this is contextual? Because the word great is now dependent on the previous words which is akshay is. Okay, maybe the word great was used in some other context. But in this context, this is a function of the previous words. Clear. So the bottom layer will produce will produce some hidden states, right? Now these hidden states will be moved to the to the task specific layer.

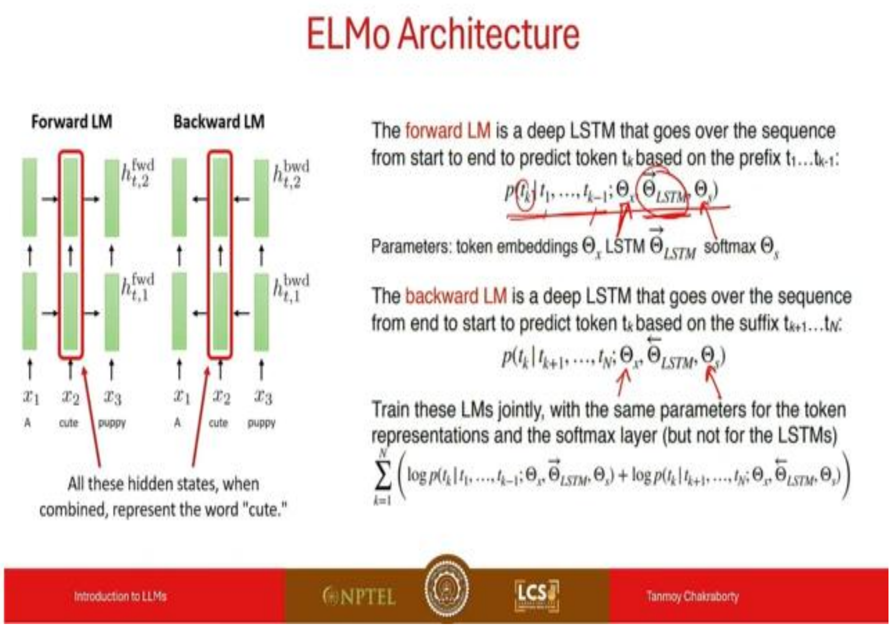
And the task specific layer, we already know the corresponding weight, which was learned earlier, that will be used to predict the task. Okay. Now there are some changes here and there. What I mentioned is a very high level overview of this. Let's look at the changes, but the bottom line stories should be clear, right? There are two layers.

One layer is responsible for language model objective. The top layer is responsible for tasks, right? Here you store the hidden states as embeddings of the tokens right and then the embeddings will be changed when you apply for a specific context it will change okay. RNN is not a very great approach. I mean, unidirectional RNN is not a very great approach because it only scans from left to right. As you understand, for a language model objective, why do we just need to scan? Because I want to process the entire sentence.

So along with the left to right RNN, I will also use a right to left RNN. This will process the entire sentence. Now left to right RNN will process, will basically predict the next word from left to right. Right to left RNN will basically predict the word from right to left.

Okay. Now, I mean, there are two ways to essentially do this thing. Let's say if you if you have access to only a left to right RNN, not an R, right to left RNN, what we can do? you can just rotate the sentence, right as I am going to school, school to going am I and then you run a left to right RNN. Okay. In fact, what they did, they used a stacked RNN.

There are two layers. Now this RNN that I'm talking about is a language model RNN, the bottom RNN that I mentioned earlier. So the bottom RNN was essentially a stacked RNN. There are two layers, right? you see here this is one RNN called the forward language model, there are two layers layer 1 and layer 2 right, and this is a backward LM backward RNN here also there are two layers, layer 1 and layer 2 right a cute puppy and a cute puppy okay, left to right and right to left. What is going to be my final hidden state which I am going to store into the dictionary the final hidden state corresponding to the word cute right can be either the concatenation of this and this right or it can be let's say only the top layers this plus this plus means concatenation.



ELMo Architecture

All these hidden states, when combined, represent the word "cute."

The forward LM is a deep LSTM that goes over the sequence from start to end to predict token t_k based on the prefix $t_1 \ldots t_{k-1}$:

$$p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s)$$

Parameters: token embeddings Θ_x LSTM $\overrightarrow{\Theta}_{LSTM}$ softmax Θ_s

The backward LM is a deep LSTM that goes over the sequence from end to start to predict token t_k based on the suffix $t_{k+1} \ldots t_N$:

$$p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)$$

Train these LMs jointly, with the same parameters for the token representations and the softmax layer (but not for the LSTMs)

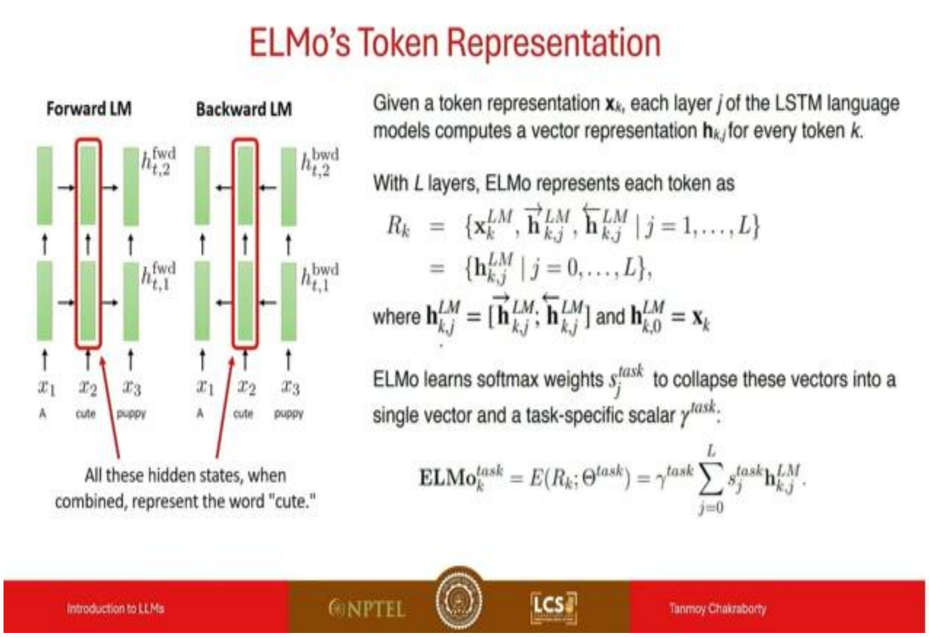
$$\sum_{k=1}^{N} \left(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \right)$$

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

Right there can be multiple ways of combining this, let's look at it here.

So you have a left to right RNN okay. Look at this equation very carefully. So this is a language model objective here. So you are given a sequence of tokens t1 t2 dot dot dot tk minus 1 you are predicting tk. What is this one? This is essentially the LSTM I mean the RNN parameter which is wh right, the all the other parameters meaning the let us say we, right all the other parameters in the RNN will remain the same. What is we w is the parameter corresponding to the input right, in this case this is theta x.

This will remain the same in the forward LSTM and the backward LSTM okay all the other parameters what is this theta S? what is this theta S? Theta S is the U okay the soft max parameter this will also remain the same in forward and backward. So the only parameter which is going to change is the WH okay so then you essentially add this to look at here, this is I think better way to present it.


Forward LM
Backward LM
$h_{t,2}^{\text{fwd}}$
$h_{t,1}^{\text{fwd}}$
$h_{t,2}^{\text{bwd}}$
$h_{t,1}^{\text{bwd}}$
x_1 A
x_2 cute
x_3 puppy
x_1 A
x_2 cute
x_3 puppy

ELMo's Token Representation

All these hidden states, when combined, represent the word "cute."

Given a token representation $\mathbf{x}_k$, each layer j of the LSTM language models computes a vector representation $\mathbf{h}_{k,j}$ for every token k.

With L layers, ELMo represents each token as

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

where $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$ and $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k$

ELMo learns softmax weights s_j^{task} to collapse these vectors into a single vector and a task-specific scalar γ^{task}:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

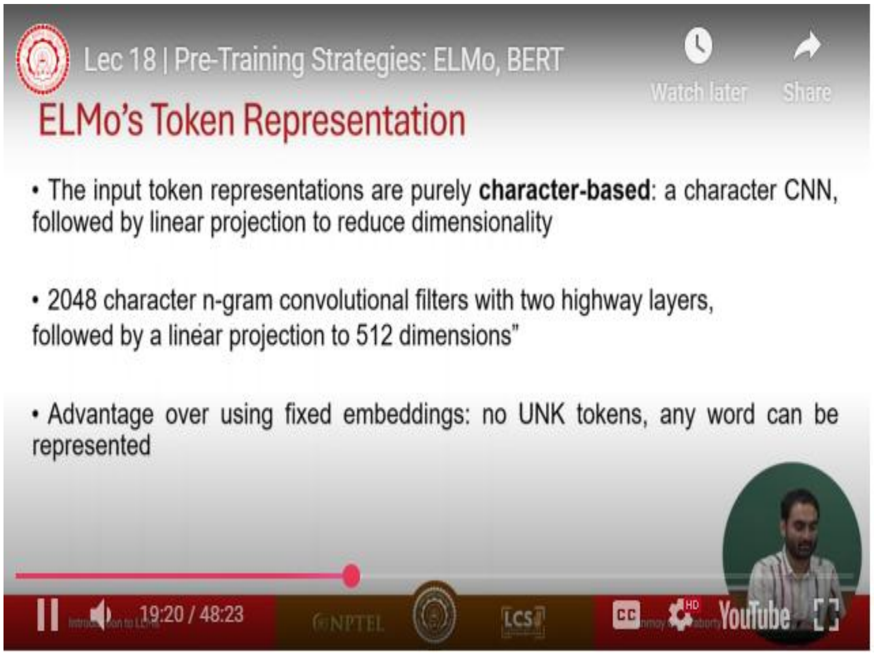
So for every token K, you got one representation from the forward layer and one representation from the backward layer. Either you just concatenate them. So what they argued is that instead of taking this embedding, this embedding, this embedding and this embedding, instead of taking four embeddings for a particular token, we will only use the top layers, not the bottom layers.

Why bottom layers are not that interesting? So after the top layers, we have this U matrix and the soft max. So the when you do backprop right this layer will be affected more compared to the bottom layer. So task specific weight is just above the top layer I will only use the top two top layers and the corresponding vectors will be fetched and concatenated okay. In fact what they said is that instead of doing this you can also add some parameters. So along with this, which is the combination of right combination of backward and forward, you add a parameter that this parameter is a layer specific parameter.

Look at your layer j ranges from zero to n. And there are two parameters. One is this gamma was this is right. Now this is corresponds to a specific layer. And this gamma corresponds to a specific task. What does it mean? It means that when you on top of this language model layer, when you apply a task specific layer, right? I mentioned earlier that the language model is going to be frozen.

Right? What they said is that instead of freezing it completely, you add some parameters which will be dependent on the task. Okay, this hidden state will not be updated. But the parameters will be updated parameters corresponding to the hidden states will be updated. It means during the inference time, when you feed one specific instance for that specific task, you basically fetch is s and gamma using because these parameters are already learned. So now, your hidden states are little bit dependent on the parameters of the task.

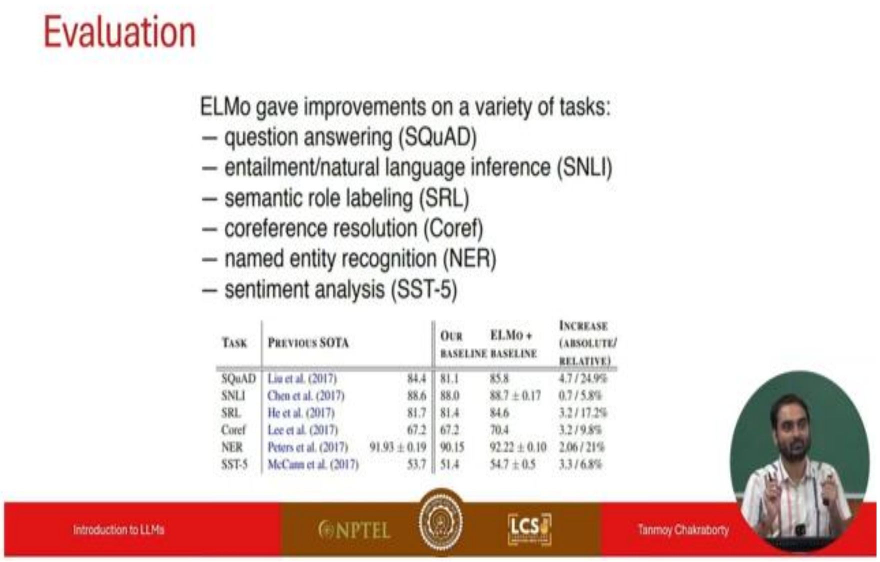
Now if you want you can discard this dependency you can just remove this class specific parameters only use the hidden state right but if you feel that okay you know some dependencies should be there you can use this right.



ELMo's Token Representation

- The input token representations are purely **character-based**: a character CNN, followed by linear projection to reduce dimensionality
- 2048 character n-gram convolutional filters with two highway layers, followed by a linear projection to 512 dimensions"
- Advantage over using fixed embeddings: no UNK tokens, any word can be represented

Some other properties of this ELMO uses token label representation this token is essentially character label token, not a word level token. Okay. So for that, they use the CNN, right? Uh, two zero four eight characters, CNN, Conv layer. And then, you know, some linear projection and so on and so forth because, kettlebell kettlebell encoding always helps you overcome the UNK kind of problem, right? On token problem.



Evaluation

ELMo gave improvements on a variety of tasks:

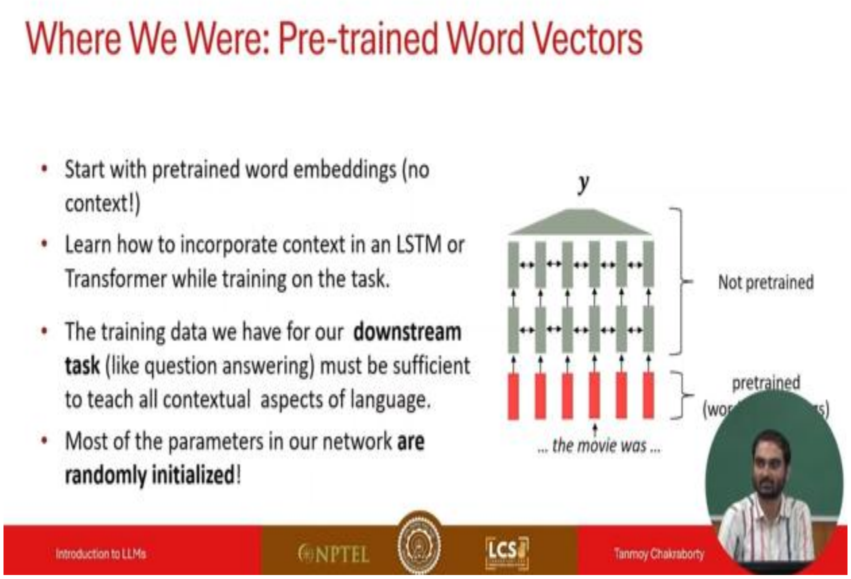
- question answering (SQuAD)
- entailment/natural language inference (SNLI)
- semantic role labeling (SRL)
- coreference resolution (Coref)
- named entity recognition (NER)
- sentiment analysis (SST-5)

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Okay. Let's look at the result very quickly. So ELMO was one of the first large scale models, okay, million size models, I think 90 million or something like that, forget the exact size. It was tested on different tasks, question answering task, squad question answering task, the squad data set was proposed by Stanford, again in 2017, 2018, during that time, we'll talk about it later. Another task was natural language inference task, semantic role-leveling task, co-reference resolution task, named entity recognition task, sentiment analysis task. SST5 is a very famous dataset. And if you look at the results, there is always an increase.

In fact, in some cases, increase is quite significant compared to the other models. Though this is the previous SOTA output right state of the art model and this is ELMO. In fact you can use ELMO to initiate transformer right. In a transformer also you need to feed embeddings right. Instead of using Watt2vec you can essentially use ELMOS representation okay.

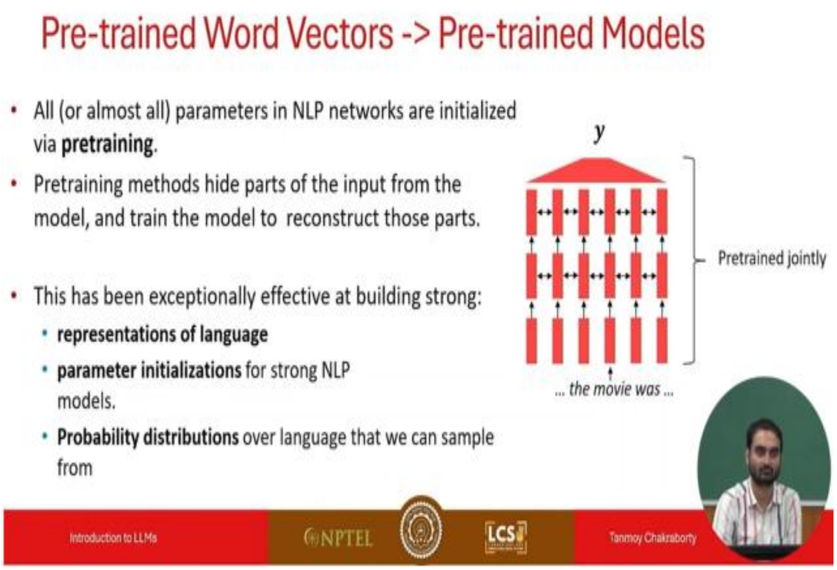
So this is all about ELMO.



Where We Were: Pre-trained Word Vectors

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.
- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network **are randomly initialized**!

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty



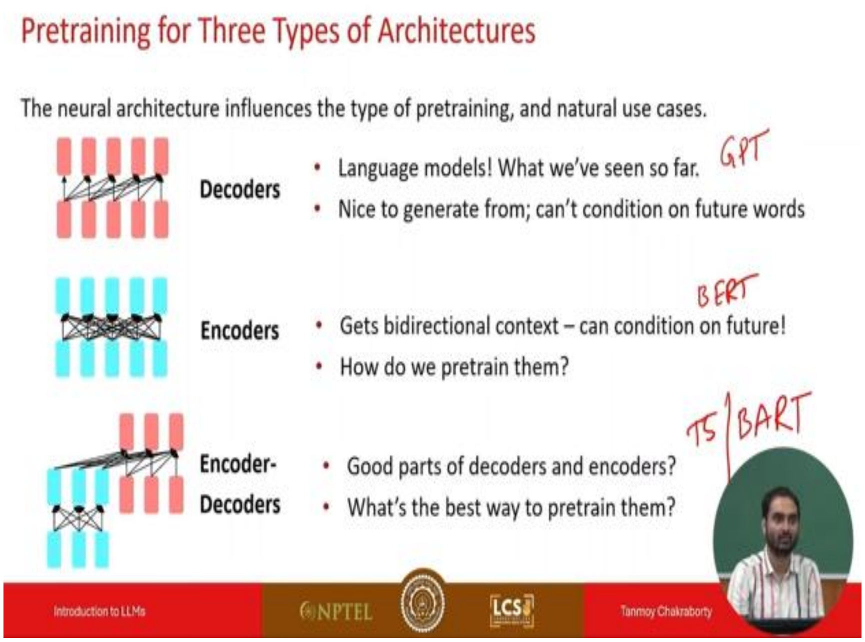
Pre-trained Word Vectors -> Pre-trained Models

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

So ELMO is yet another word embedding technique but contextual, dynamic, unlike what to be a glove, which was static, right? Context independent. So we have been talking about pre-trained word vectors. In pre-trained word vectors, your embedding layer is frozen, right? And your task specific layer is essentially something that you are interested in, okay? And this part is pre-trained. model is not pre-trained, the entire model is not pre-trained, only the inputs are pre-trained, right? Now, from pre-trained word vectors we are moving to pre-trained models. In the pre-trained model, all the parameters are pre-trained, right? All the parameters in a transformer model, the parameters are the parameters corresponding to the attentions, right? Parameters corresponding to feed forward network, all of them are pre-trained.

Right then for a specific task you can fine tune it okay.



Pretraining for Three Types of Architectures

The neural architecture influences the type of pretraining, and natural use cases.

Decoders
- Language models! What we've seen so far. GPT
- Nice to generate from; can't condition on future words

Encoders
- Gets bidirectional context – can condition on future! BERT
- How do we pretrain them?

Encoder-Decoders
- Good parts of decoders and encoders? T5 / BART
- What's the best way to pretrain them?

So let's look at so there are three kinds of pretrainings that we are interested in right depending on which part of the transformer we are interested right, transformer, there is an encoder block, there is a decoder block. So one type of pretraining only considers the decoder part of the transformer, okay. So all these GPT series models, they are only decoder only models, decoder based models, right. They do not consider the encoder part, therefore this cross attention is not there, okay.

The other type of models are encoder only models where we only look at the encoder part of it, we forget the decoder part and then we pretend an encoder model. BERT, ALBERT, ROBATA, all these models are encoder-only model. And then the normal structure, which is the encoder-decoder kind of model, the actual transformer model, this can also be used for pre-training.

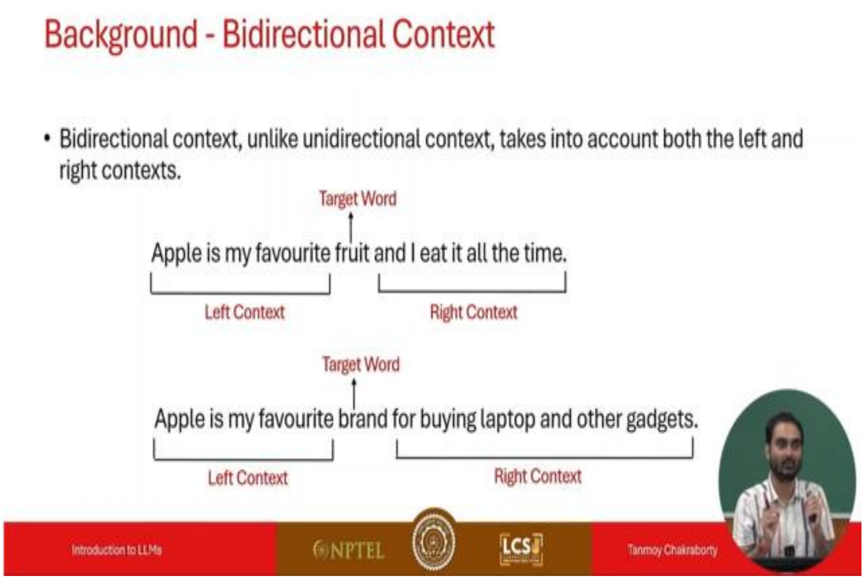
So GPT comes here. BERT series model comes here. And T5, another model is called BART, B-A-R-T. These are all basically encoder-decoder models.

BERT: Bidirectional Encoder Representations from Transformers

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

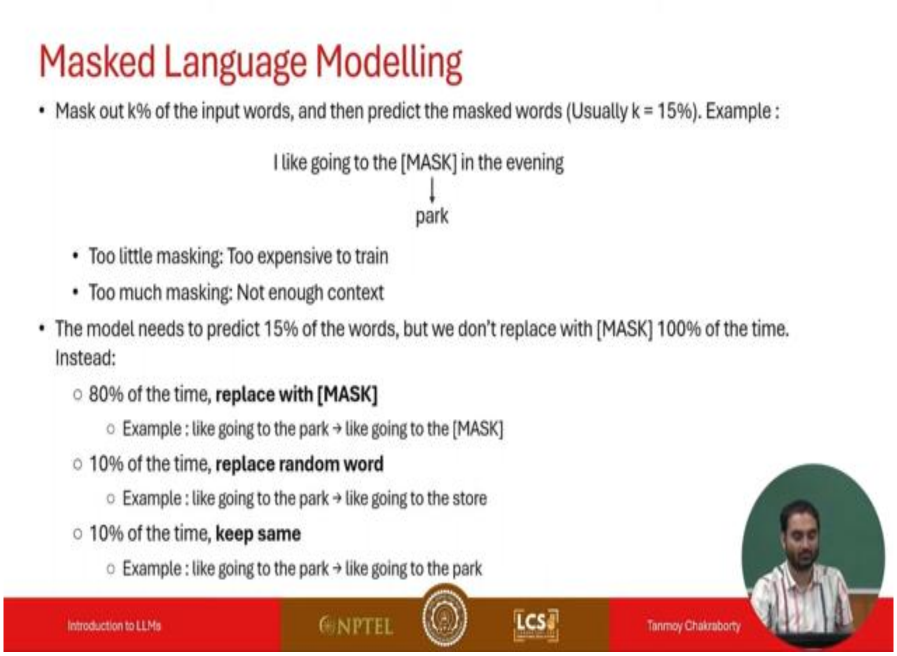
Slides are adopted from Jacob Devlin



Background - Bidirectional Context

- Bidirectional context, unlike unidirectional context, takes into account both the left and right contexts.

Target Word
Apple is my favourite fruit and I eat it all the time.
Left Context
Right Context
Target Word
Apple is my favourite brand for buying laptop and other gadgets.
Left Context
Right Context

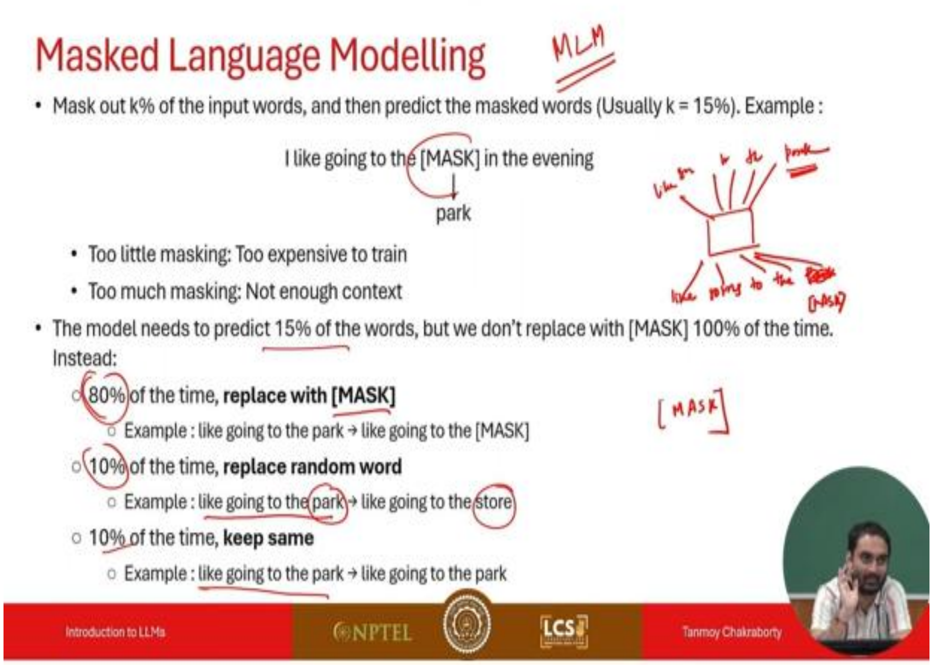


Masked Language Modelling

- Mask out k% of the input words, and then predict the masked words (Usually k = 15%). Example :

  I like going to the [MASK] in the evening
  ↓
  park

  - Too little masking: Too expensive to train
  - Too much masking: Not enough context
- The model needs to predict 15% of the words, but we don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, **replace with [MASK]**
    - Example : like going to the park → like going to the [MASK]
  - 10% of the time, **replace random word**
    - Example : like going to the park → like going to the store
  - 10% of the time, **keep same**
    - Example : like going to the park → like going to the park



Masked Language Modelling

- Mask out k% of the input words, and then predict the masked words (Usually k = 15%). Example :

  I like going to the [MASK] in the evening
  ↓
  park

  - Too little masking: Too expensive to train
  - Too much masking: Not enough context
- The model needs to predict 15% of the words, but we don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, **replace with [MASK]**
    - Example : like going to the park → like going to the [MASK]
  - 10% of the time, **replace random word**
    - Example : like going to the park → like going to the store
  - 10% of the time, **keep same**
    - Example : like going to the park → like going to the park

So let's look at this BERT model proposed by Google. So BERT stands for Bidirectional Encoder Representations from Transformer.

And this is the paper. So now we have only the encoder part of the Transformer model, only the encoder part. And now we are creating this encoder part. Encoder part, the beauty of the encoder part is that there is no masked self-attention. This is unmasked self-attention, meaning all the tokens, each of the tokens can attend to all the other tokens in the sequence. So it is some sort of bi-detection in nature because you can scan from left to right, right to left and whatever.

So in bert, when you pre-train, what we do, we essentially now for pre-training, what is my objective? Because I don't have any task specific data. I only have access to raw corpus, right? Raw corpus, raw sentences. What is going to be my objective? So in this paper, they came up with an objective called mask language model, MLM. Now this is a self supervised approach.

Not an unsupervised approach because there's some gradient flow. Self-supervised approach, you don't need any label data. Right? So what you do in bert in the encoder part is that you feed a sentence, right? And you mask some of the tokens of the sentence. And let's say for the time being the tokens are words. You mask some of the tokens and you let the model predict those tokens.

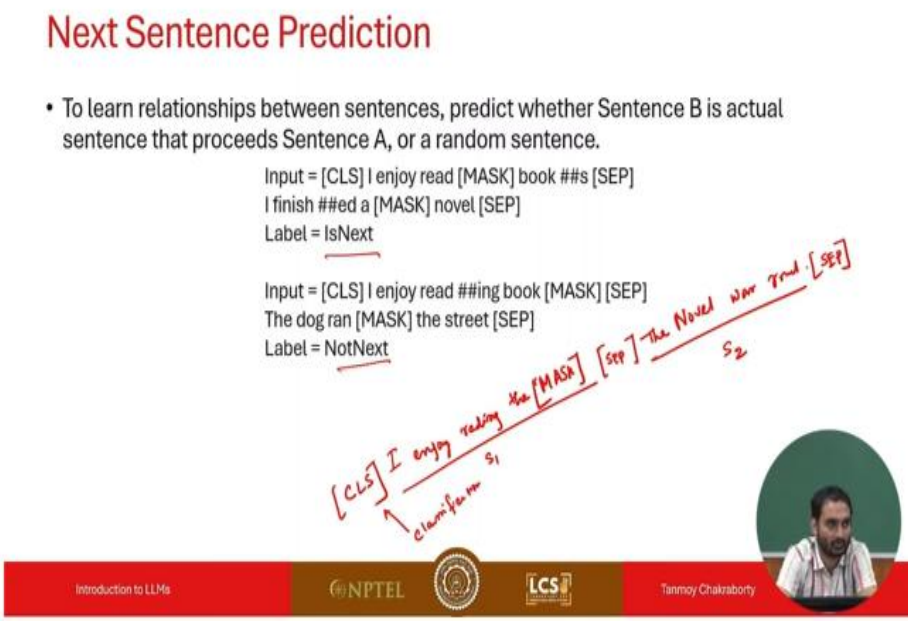
Okay. Now there are so many questions. For example, which tokens I should mask, how many tokens I should mask, right? And so on and so forth. So these are all empirical questions. And what they showed is that 50% masking is enough. So you mask 15% of tokens present in the training set, right? But when you mask it, you mask it in three different ways. Okay, out of this 15% 80% cases, right? You basically just mask that token, right? So what does it mean by masking a token? So they introduced a new token called mask, right? A new token called mask when you feed, you replace a token by mask. And at that position corresponding to the final output, you predict what is going to be the actual word at the place of mask. So 80% time you mask some of the tokens, 10% times you replace that token by another token. Let's say the sentence is like going to the park, you replace the word park by store and you let the model predict the word park. This is some sort of corruption. You do some sort of corruption and you let the model predict the actual word.

And 10% of time you don't do any change but still you let the model predict it. your input is like going to the park, right, you let the model predict toward park. Why this is important? Why this is important? This is important because if you only teach the model to predict the mask token, the model will basically start forgetting predicting the actual tokens. So sometimes you also need to, you know, tell the model that look, I mean, you are learning the this mask output, but you also need to understand what is there in the other tokens.

Okay. So why 15%? Why not 50%? If you mask 50% of the tokens, right? Most of the context will be destructed. Your context, you also need some context, right? The context will be gone, right? So they empirically tested that 15% is a good number. 15% of an 1 billion or a 10 billion size token is good enough. This is my sentence. Let's say like, going to the park, you fit this, you replace this by mask and you want the model to predict like going to the park.

So your error is only computed here. not at the other tokens, other token positions. This is very important, right? Now this is not all about what I'm coming to the other parts of it. So mask language modeling is clear, clear. So one task is this mask language prediction, okay? Mask language model.

So they actually, they use two tasks.



Next Sentence Prediction

- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence.

Input = [CLS] I enjoy read [MASK] book ##s [SEP]
I finish ##ed a [MASK] novel [SEP]
Label = IsNext

Input = [CLS] I enjoy read ##ing book [MASK] [SEP]
The dog ran [MASK] the street [SEP]
Label = NotNext

[CLS] I enjoy reading the [MASK] [SEP] The Novel was read. [SEP]
S1
S2
classifier
Introduction to LLMs
NPTEL
LCS
Tanmoy Chakraborty

One is this mask language prediction and the second is next sentence prediction. What is next sentence prediction? you give two sentences right you have this corpus you know which sentences are actually are there side by side you know the two consecutive sentences you take two sentences w1 and w2 and you predict you let the model predict whether w2 will follow w1 or not. So for the training data what you do? you take those pairs of sentences which actually appear side by side right as your positive samples and as your negative sample you do random shuffling right and you basically take those pairs which do not appear side by side.

I enjoy reading the book. I finished the novel. These two sentences appear side by side. So the label is yes, otherwise the label is no. So these are two tasks and we will do these two tasks simultaneously. Meaning, let us say the input is I enjoyed reading the mask. Okay and how to separate these two sequences? This is sequence one, sequence two the novel was great right, we separate them using another token called SEP separator, this separator is a new token like a mask token.

And again you may have a separator here. So you introduce two new tokens. Along with this we introduce another token called the CLS token. And this is prepended with the sequence. This stands for classification. Okay, so you feed this entire thing to the encoder block right. What is the need of the CLS token? Now, let me first tell you what will happen after this.

So the model will predict the vectors right at the end right vector corresponding to CLS vector corresponding to I and so on and so forth and it will also try to predict this mask word on top of this CLS I will use a task specific head meaning basically a linear layer view right which will project it into two classes yes or no. Okay, if two sequences are sequential then it will predict yes otherwise no. There are two losses, one loss is here, another loss is here, mask and those two losses will be back-promoted. Now my question is why do we need this CLS token? Can't we just take the embeddings of all these tokens and then concatenate it and then pass it through a linear layer followed by non-linearity. Okay, what is saying that we can't take the concatenation because we don't essentially know this one, the actual masked token, right.

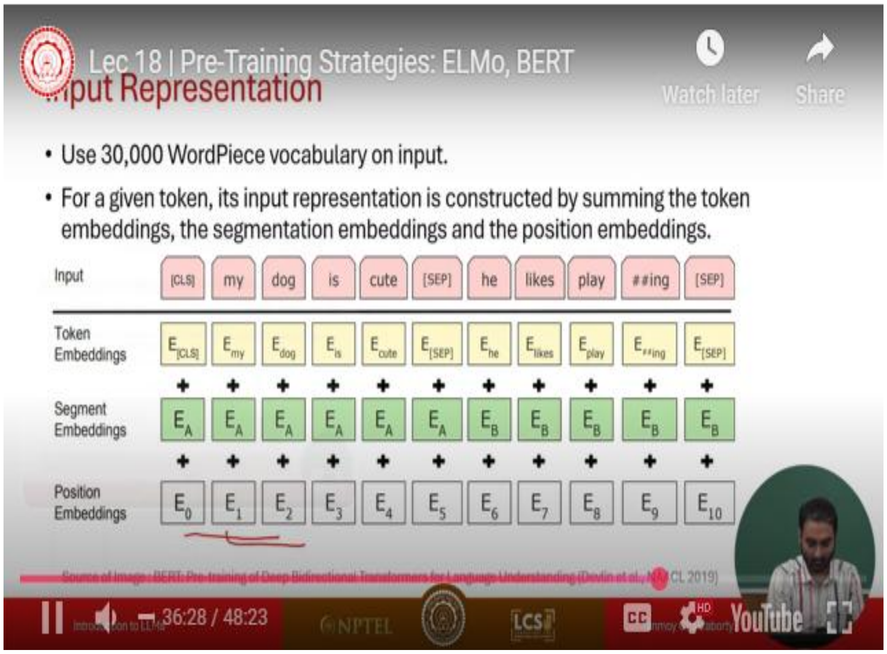
Okay, that's fine. Then we just ignore the masked token and somehow combine the other tokens. So the other tokens are good enough. Only 15% are masked. Why this CLS token? So remember CLS token also acts as a normal token.

Meaning CLS also attends to all the other tokens. In the self-attention part, it has attended to all the other tokens. So CLS has information about all the tokens, right? But CLS token is a token which doesn't indicate any meaning present in the sentence because this is not a part of the sentence at all. You may argue that why do I need a classification layer on top of CLS? I can have a classification layer on top of I, on top of read, they have also attended to all the other tokens, right? But they themselves have some meanings.

A CLS is a token which is not present in the sequence at all. So it is not biased. That token is not biased. The CLS token is not biased to any of the other tokens present in the sequence. Whereas the token I is biased towards itself. could be high, right? In fact, there is another plus point, you know, behind using CLS for the classification. How many weights you are incorporating here? Apart from the, you know, the normal encoder weights, self-attention feed forward weights, these are already there in the encoder.

For this classification, how many weights you are incorporating? Parameters, you are incorporating, only this part, right, only you, And what is the size of you? If the vector size is 512 this is 512 into 2 right, instead of this, if we had a concatenation of tokens, this is going to be bigger, right, the parameter size can also be decreased okay. Now this error which is the task-specific error will get back-propagated end-to-end. The mask, the error related to this mask token and the error which is there on top of this CLS token, those two errors will be back-propagated into it and all the encoder blocks will be updated. Now this is encoder, you can use this encoder block for word embeddings also.

Like Elmo, you can use Bert for contextual word embeddings, okay.



Input Representation

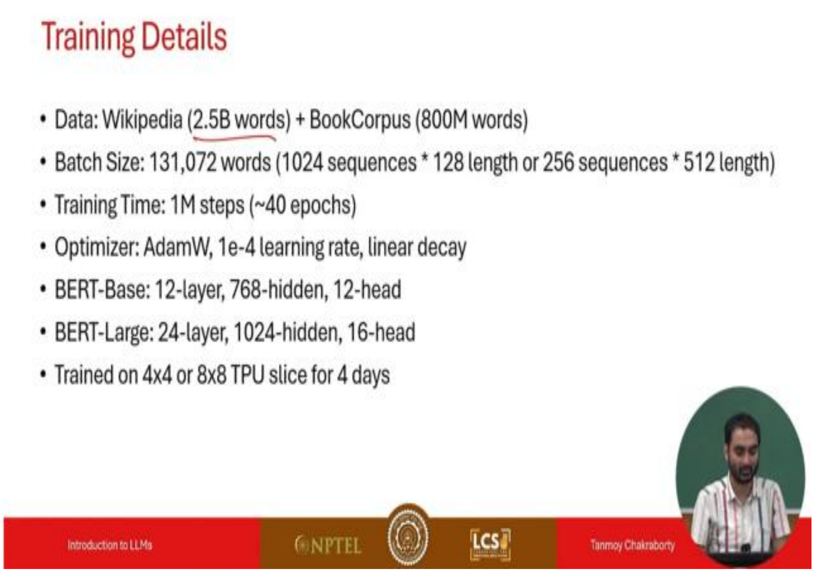
- Use 30,000 WordPiece vocabulary on input.
- For a given token, its input representation is constructed by summing the token embeddings, the segmentation embeddings and the position embeddings.

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}

Source of image: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., NAACL 2019)

All right so this is the graphical depiction look at here. E0, E1, E2, E3, E4 basically E1 to E4 from this to this, this is one sequence, from this to this is another sequence, this to this is another sequence. My dog is cute, he likes playing. Right? You have a Position encoding, right? So in here, they introduce another encoding called the segment encoding. Segment encoding is just indicating whether this token comes from the first sentence or the second sentence, right? So segment is A and this is B.

So position encoding, segment encoding, then from that you also have this normal token encoding. This can come from ELMO, can come from word2vec and so on and so forth. And then you have the CLS prepended, SEP separator and so on and so forth. Now this is my input sequence, you mask some of them and you feed.

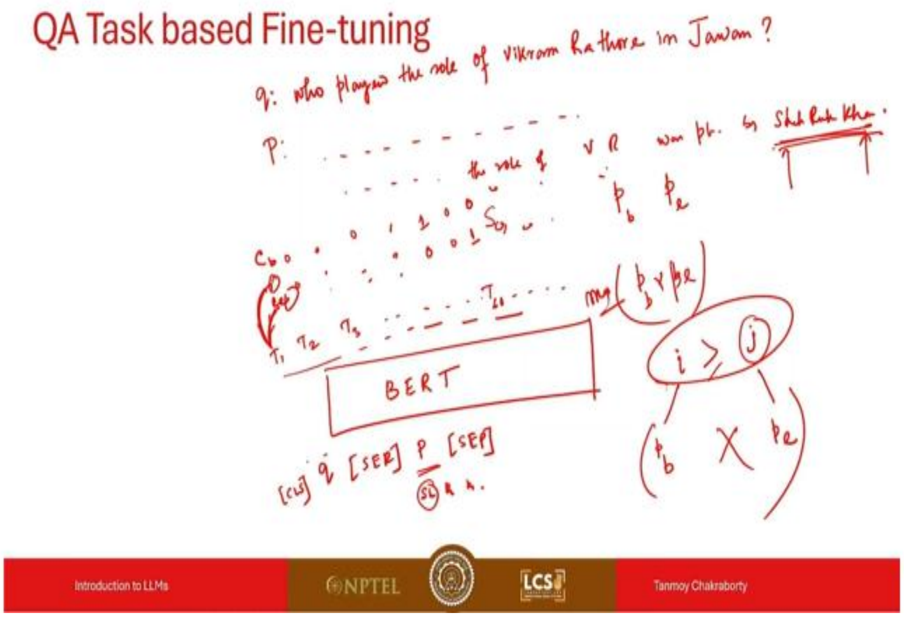


Training Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

So this is the training details. They used 2.5 billion Wikipedia corpus, another book corpus consisting of 800 million words. They tested two different variations of bert. Bert base consisting of only 12 layers of encoder and bert large consists of 24 layers of encoder with different hidden state size and different multi-headed attention.

12 head, 16 head, right and so on and so forth. There are other epochs and so on. Other details are also mentioned here. Okay.



QA Task based Fine-tuning

Now, if I ask you, can we use BERT for a downstream task? And let's say the downstream task is question answering.

Okay. What is the question answering task? You are given a question queue. Okay. and let us say who played the role of Vikram Rathore in Jawan. Okay, this is the question and this is the passage. You have a passage where all these things are written and somewhere in this passage it is written that the role of Vikram Rathore was played by Shah Rukh Khan.

This is the passage. And your task would be to essentially retrieve this information from the passage. Now tell me, how do we use BERT for this kind of task? Okay, this is a typical closed book kind of question answering task where you are given, there are so many data sets, SQuAD, SQuAD is one data set, right? There are other data sets like Quark, right? Hotpot, QA, there are so many data sets available. Some data sets look like this, some data sets basically are conversational data sets, where you have conversation, you ask the question and you retrieve answer from the conversation. How do we use this, the bert model to solve this kind of task? What do you think? The BERT model is already pre-trained using mass language model. Now, when you fine-tune the model for this task, you already have some training data where the question and the corresponding passage is the input and the output is the specific token. So, what you do here in the BERT model? This is BERT, you add up to this model for specific task, this is one such task okay, So during training you feed question queue right separator and the passage P separator okay and you have CLS okay.

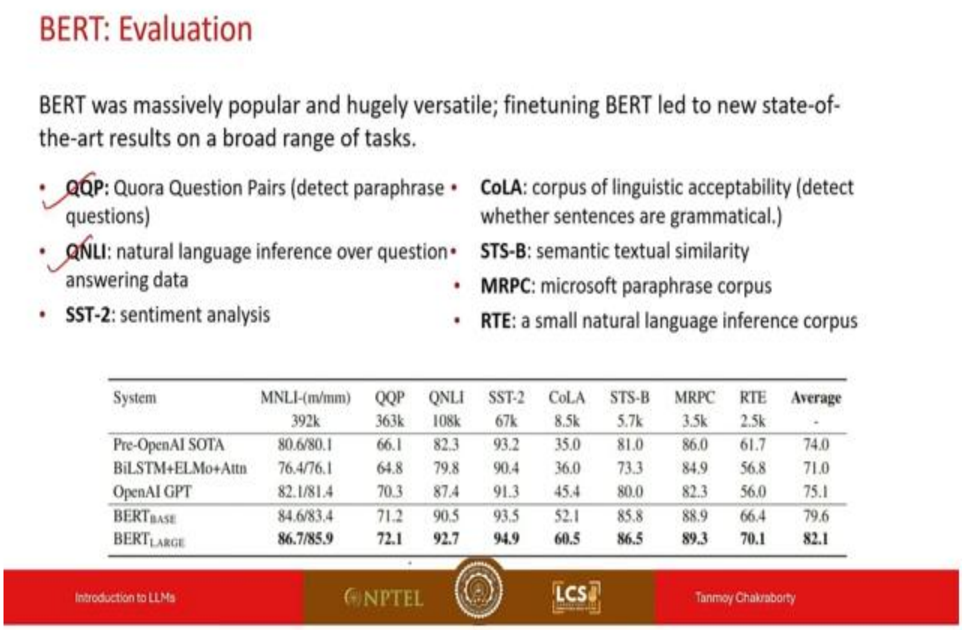
So you have different tokens here you have token, let's say token T1, token T2, T3 dot dot dot dot dot, right. So T60 and so on and so forth. Then what you do, you add a classification layer on top of every token. In fact, you add two classification layers on top of every token, right.

One classification layer will be responsible for, I mean, you can think of this specific phrase as a span. right? So this is start of a span, this is the end of a span. So here in this passage, there is this word called Shah Rukh Khan, right? And the corresponding embedding of Shah, corresponding embedding of Rukh, corresponding embedding of Khan will also be present here. Right you maintain two different classification heads one classification head will basically keep track of the beginning of the span another will keep track of the end of the span let's say you have c1 it's a cb ce okay so cb for all the all these tokens will be 0 0 0 0 0 0 during training right and for the word Sa right Cb is going to be 1 and Cu will be 0 then the other the Rukh will be 0, 0, Khan will be 0 and 1 and the remaining will be 0, 0, 0.

This is my ground truth data. I want the model to predict this labels right. Now you feed the input to the model, the model will produce the embeddings, right? And those embeddings will be fed to the classification layers. Classification layers will essentially produce some probabilities. There are two probabilities. One is the probability that the current token acts as a beginning and the current token acts as an end. So my hope will be that the token correspond to SA is going to be the PB for the token SA is going to be 10 to 1 that tends to 1 and the PE correspond to Khan tends to 1.

If it doesn't happen, then there's a loss and that loss gets back-promoted. Right, task specific layers meaning these two layers, right, these two layers will be mostly updated and the other pretrained layers can also be updated, okay. So now this, the model is now added to specific task. When a test sentence comes in, a test sentence, the test question and the passage, what you you use this pretrained model, pretrained followed by fine-tuned, whatever the model that is there, right? You pass the sequence, right? It will produce P, B and PE for all these tokens. And our task would be to maximize what? The task would be to basically figure out two such positions such that pb times pe will be maximum. Now think about it for every token, you have two probabilities pb pe right all there are so many possible combinations are there right pb times pe so pb1 times pe2 right pb3 times pb pe1 and so on and so forth all these combinations are possible.

Which combination I will take? I will take that combination where number one the int token I mean let's say you choose you have chosen i and j as two different positions right and pb comes from i and pe comes from e pe comes from j j should always be greater than i. Of course, right the end token should come after the beginning token right this is the first condition or whatever get them equals to because only one token can also be possible okay and then you know considering this condition then I want to maximize this probability Pb times P. Now you may ask that in a passage there are multiple appearances of the word Shah Rukh Khan so which one I will consider? So generally you consider the beginning, the first appearance of the words have come. So in this way you can essentially use a pre-trained BERT model for a downstream task like question answering task, which apparently looks very different from the pre-training objective.



BERT: Evaluation

BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks.

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

Now this is all about BERT. It was tested across multiple tasks, right? This question answering task, natural language inference task. Remember, for each of these tasks, you need to adapt your model accordingly, right? This is the cola benchmark, which is a corpus of linguistic acceptability. It basically indicates whether a sentence is grammatically correct or not, right? STS benchmark, SPET benchmark and so on and so forth. And it showed that what large significantly improves the tasks, right? Compared to the other baselines, which are shorter at that time. It also beat the GPT-1 model, okay? BERT had been one of the state-of-the-art models till 2020 and following BERT there were approaches like D-BERT, RobERT, TinyBERT where people started squeezing the model, making the model tiny.

Then this RobERT paper says that the BERT model is under-trained, you need to train them more and more. and so on and so forth. There are papers where people essentially feed a lot of training data across multiple tasks. People also showed when you keep on adding new tasks, whether the model will be generalized or not, and so on and so forth. So this is one direction of research.

I will not go into the details of the follow-up of BERT, but I will strongly suggest you guys to read those papers. At least use those models for your projects. With this I stop here and In the next lecture we will try to address the other types of pre-trainings like GPD and T5. Thank you.