1. **Explain the architecture of BERT**

**Explanation of BERT Model – NLP**

Last Updated : 10 Jan, 2024

**BERT, an acronym** **for Bidirectional Encoder Representations from Transformers**, stands as an open-source **machine learning framework**designed for the realm of **natural language processing (NLP)**. Originating in 2018, this framework was crafted by researchers from Google AI Language. The article aims to explore the **architecture, working and applications of BERT**.

**What is BERT?**

[BERT (Bidirectional Encoder Representations from Transformers)](https://www.geeksforgeeks.org/understanding-bert-nlp/) leverages a transformer-based neural network to understand and generate human-like language. BERT employs an encoder-only architecture. In the original [Transformer architecture](https://www.geeksforgeeks.org/getting-started-with-transformers/), there are both encoder and decoder modules. The decision to use an encoder-only architecture in BERT suggests a primary emphasis on understanding input sequences rather than generating output sequences.

**Bidirectional Approach of BERT**

Traditional language models process text sequentially, either from left to right or right to left. This method limits the model’s awareness to the immediate context preceding the target word. BERT uses a bi-directional approach considering both the left and right context of words in a sentence, instead of analyzing the text sequentially, BERT looks at all the words in a sentence simultaneously.

***Example: “The bank is situated on the \_\_\_\_\_\_\_ of the river.”***

*In a unidirectional model, the understanding of the blank would heavily depend on the preceding words, and the model might struggle to discern whether “bank” refers to a financial institution or the side of the river.*

*BERT, being bidirectional, simultaneously considers both the left (“The bank is situated on the”) and right context (“of the river”), enabling a more nuanced understanding. It comprehends that the missing word is likely related to the geographical location of the bank, demonstrating the contextual richness that the bidirectional approach brings.*

**Pre-training and Fine-tuning**

The BERT model undergoes a two-step process:

1. Pre-training on Large amounts of unlabeled text to learn contextual embeddings.
2. Fine-tuning on labeled data for specific [NLP](https://www.geeksforgeeks.org/natural-language-processing-overview/) tasks.

**Pre-Training on Large Data**

* BERT is pre-trained on large amount of unlabeled text data. The model learns contextual embeddings, which are the representations of words that take into account their surrounding context in a sentence.
* BERT engages in various unsupervised pre-training tasks. For instance, it might learn to predict missing words in a sentence (Masked Language Model or MLM task), understand the relationship between two sentences, or predict the next sentence in a pair.

**Fine-Tuning on Labeled Data**

* After the pre-training phase, the BERT model, armed with its contextual embeddings, is then fine-tuned for specific natural language processing (NLP) tasks. This step tailors the model to more targeted applications by adapting its general language understanding to the nuances of the particular task.
* BERT is fine-tuned using labeled data specific to the downstream tasks of interest. These tasks could include sentiment analysis, question-answering, [named entity recognition](https://www.geeksforgeeks.org/named-entity-recognition/), or any other NLP application. The model’s parameters are adjusted to optimize its performance for the particular requirements of the task at hand.

BERT’s unified architecture allows it to adapt to various downstream tasks with minimal modifications, making it a versatile and highly effective tool in [natural language understanding](https://www.geeksforgeeks.org/nlp-vs-nlu-vs-nlg/) and processing.

**How BERT work?**

BERT is designed to generate a language model so, only the encoder mechanism is used. Sequence of tokens are fed to the Transformer encoder. These tokens are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors, each corresponding to an input token, providing contextualized representations.

When training language models, defining a prediction goal is a challenge. Many models predict the next word in a sequence, which is a directional approach and may limit context learning. BERT addresses this challenge with two innovative training strategies:

1. Masked Language Model (MLM)
2. Next Sentence Prediction (NSP)

**1. Masked Language Model (MLM)**

In BERT’s pre-training process, a portion of words in each input sequence is masked and the model is trained to predict the original values of these masked words based on the context provided by the surrounding words.

In simple terms,

1. **Masking words:** Before BERT learns from sentences, it hides some words (about 15%) and replaces them with a special symbol, like [MASK].
2. **Guessing Hidden Words:** BERT’s job is to figure out what these hidden words are by looking at the words around them. It’s like a game of guessing where some words are missing, and BERT tries to fill in the blanks.
3. **How BERT learns:**
   * BERT adds a special layer on top of its learning system to make these guesses. It then checks how close its guesses are to the actual hidden words.
   * It does this by converting its guesses into probabilities, saying, “I think this word is X, and I’m this much sure about it.”
4. **Special Attention to Hidden Words**
   * BERT’s main focus during training is on getting these hidden words right. It cares less about predicting the words that are not hidden.
   * This is because the real challenge is figuring out the missing parts, and this strategy helps BERT become really good at understanding the meaning and context of words.

In technical terms,

1. BERT adds a classification layer on top of the output from the encoder. This layer is crucial for predicting the masked words.
2. The output vectors from the classification layer are multiplied by the embedding matrix, transforming them into the vocabulary dimension. This step helps align the predicted representations with the vocabulary space.
3. The probability of each word in the vocabulary is calculated using the [SoftMax activation function](https://www.geeksforgeeks.org/activation-functions-neural-networks/). This step generates a probability distribution over the entire vocabulary for each masked position.
4. The loss function used during training considers only the prediction of the masked values. The model is penalized for the deviation between its predictions and the actual values of the masked words.
5. The model converges slower than directional models. This is because, during training, BERT is only concerned with predicting the masked values, ignoring the prediction of the non-masked words. The increased context awareness achieved through this strategy compensates for the slower convergence.

**2. Next Sentence Prediction (NSP)**

BERT predicts if the second sentence is connected to the first. This is done by transforming the output of the [CLS] token into a 2×1 shaped vector using a classification layer, and then calculating the probability of whether the second sentence follows the first using SoftMax.

1. In the training process, BERT learns to understand the relationship between pairs of sentences, predicting if the second sentence follows the first in the original document.
2. 50% of the input pairs have the second sentence as the subsequent sentence in the original document, and the other 50% have a randomly chosen sentence.
3. To help the model distinguish between connected and disconnected sentence pairs. The input is processed before entering the model:
   * A [CLS] token is inserted at the beginning of the first sentence, and a [SEP] token is added at the end of each sentence.
   * A sentence embedding indicating Sentence A or Sentence B is added to each token.
   * A positional embedding indicates the position of each token in the sequence.
4. BERT predicts if the second sentence is connected to the first. This is done by transforming the output of the [CLS] token into a 2×1 shaped vector using a classification layer, and then calculating the probability of whether the second sentence follows the first using SoftMax.

During the training of BERT model, the Masked LM and Next Sentence Prediction are trained together. The model aims to minimize the combined loss function of the Masked LM and Next Sentence Prediction, leading to a robust language model with enhanced capabilities in understanding context within sentences and relationships between sentences.

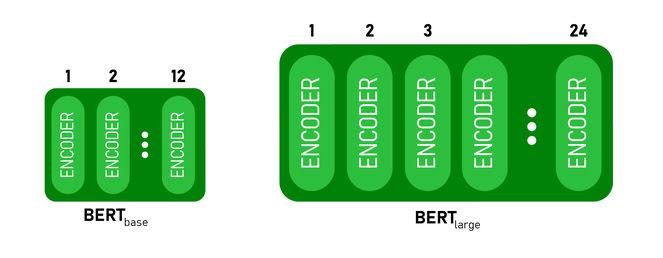
**Why to train Masked LM and Next Sentence Prediction together?**

Masked LM helps BERT to understand the context within a sentence and[Next Sentence Prediction](https://www.geeksforgeeks.org/next-sentence-prediction-using-bert/)helps BERT grasp the connection or relationship between pairs of sentences. Hence, training both the strategies together ensures that BERT learns a broad and comprehensive understanding of language, capturing both details within sentences and the flow between sentences.

**BERT Architectures**

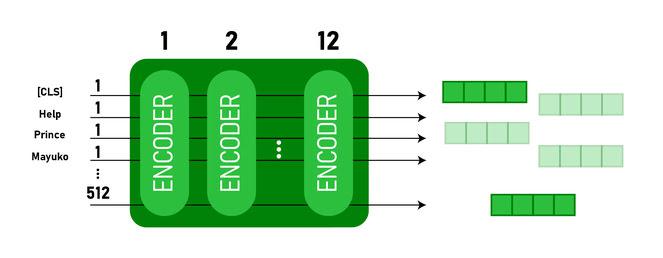
The architecture of BERT is a multilayer bidirectional transformer encoder which is quite similar to the transformer model. A transformer architecture is an encoder-decoder network that uses [self-attention](https://www.geeksforgeeks.org/self-attention-in-nlp/) on the encoder side and attention on the decoder side.

1. BERTBASE has 1*2 layers in the Encoder stack* while BERTLARGE has *24 layers in the Encoder stack*. These are more than the Transformer architecture described in the original paper (*6 encoder layers*).
2. BERT architectures (BASE and LARGE) also have larger feedforward networks (768 and 1024 hidden units respectively), and *more attention heads (12 and 16 respectively)* than the Transformer architecture suggested in the original paper. It contains *512 hidden units and 8 attention heads*.
3. BERTBASE contains 110M parameters while BERTLARGE has 340M parameters.



*BERT BASE and BERT LARGE architecture.*

This model takes the **CLS**token as input first, then it is followed by a sequence of words as input. Here CLS is a classification token. It then passes the input to the above layers. Each layer applies [self-attention](https://www.geeksforgeeks.org/self-attention-in-nlp/) and passes the result through a feedforward network after then it hands off to the next encoder. The model outputs a vector of hidden size (*768*for BERT BASE). If we want to output a classifier from this model we can take the output corresponding to the CLS token.



*BERT output as Embeddings*

Now, this trained vector can be used to perform a number of tasks such as classification, translation, etc. For Example, the paper achieves great results just by using a single layer [Neural Network](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/) on the BERT model in the classification task.

**How to use BERT model in NLP?**

BERT can be used for various natural language processing (NLP) tasks such as:

**1. Classification Task**

* BERT can be used for classification task like [sentiment analysis](https://www.geeksforgeeks.org/what-is-sentiment-analysis/), the goal is to classify the text into different categories (positive/ negative/ neutral), BERT can be employed by adding a classification layer on the top of the Transformer output for the [CLS] token.
* The [CLS] token represents the aggregated information from the entire input sequence. This pooled representation can then be used as input for a classification layer to make predictions for the specific task.

**2. Question Answering**

* In question answering tasks, where the model is required to locate and mark the answer within a given text sequence, BERT can be trained for this purpose.
* BERT is trained for question answering by learning two additional vectors that mark the beginning and end of the answer. During training, the model is provided with questions and corresponding passages, and it learns to predict the start and end positions of the answer within the passage.

**3. Named Entity Recognition (NER)**

* BERT can be utilized for NER, where the goal is to identify and classify entities (e.g., Person, Organization, Date) in a text sequence.
* A BERT-based NER model is trained by taking the output vector of each token form the Transformer and feeding it into a classification layer. The layer predicts the named entity label for each token, indicating the type of entity it represents.

1. **Explain Masked Language Modeling (MLM)**

Masked language models (MLMs) are used in natural language processing ([NLP](https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP)) tasks for training language models. Certain words and tokens in a specific input are randomly masked or hidden in this approach and the model is then trained to predict these masked elements by using the context provided by the surrounding words.

Masked language modeling is a type of self-supervised learning in which the model learns to produce text without explicit labels or annotations. Instead, it draws its supervision from the incoming text. Because of this feature, masked language modeling can be used to carry out various NLP tasks such as text classification, answering questions and text generation.

Masked language modeling particularly helps with training transformer models such as Bidirectional Encoder Representations from Transformers ([BERT](https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model)), GPT and RoBERTa.

**How do Masked Language Models work?**

As a pretraining technique for [deep learning](https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network) models in NLP, MLMs work by masking a portion of the input tokens in a sentence at random and then asking the model to predict the masked tokens. The model is trained on huge volumes of text data so that it can learn to recognize word context and forecast masked tokens depending on their context. For example, in the sentence, "The cat [MASK] the tree," the model would predict the word *climbed* as the masked token.

[**What is generative AI? Everything you need to know**](https://www.techtarget.com/searchenterpriseai/definition/generative-AI)

Throughout the training process, the model is updated based on the difference between its predictions and the words in the sentence. The pretraining phase assists the model in learning valuable contextual representations of words, which can then be fine-tuned for specific NLP tasks.

The goal of masked language modeling is to use the large amounts of text data available to train a general-purpose language model that can be applied to a variety of NLP challenges.

**What is Hugging Face?**

[Hugging Face](https://www.techtarget.com/whatis/definition/Hugging-Face) is an artificial intelligence ([AI](https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence)) research organization that specializes in creating open source tools and libraries for NLP tasks. Serving as a hub for both AI experts and enthusiasts, it functions similarly to a [GitHub](https://www.techtarget.com/searchitoperations/definition/GitHub) for AI. Initially introduced in 2017 as a [chatbot](https://www.techtarget.com/searchcustomerexperience/answer/What-chatbot-evaluation-metrics-do-you-use-to-measure-performance) app for teenagers, Hugging Face has transformed over the years into a platform where a user can host, train and collaborate on AI models with their teams.

Hugging Face offers various libraries and tools that can be used for masked language projects, including the following:

* **Transformers.** [Transformers are a recent breakthrough](https://www.techtarget.com/searchenterpriseai/feature/Transformer-neural-networks-are-shaking-up-AI) in machine learning ([ML](https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML)) and AI models and have been creating a lot of buzz. Hugging Face includes Python [libraries](https://www.techtarget.com/searchapparchitecture/tip/Frameworks-libraries-and-languages-for-machine-learning) with pretrained transformer models and tools for fine-tuning models.
* **Tokenizers.** Tokenizers are a library for effective preprocessing and tokenization of text. Since models can only handle numerical data, tokenizers are necessary to translate text inputs into numbers. Therefore, the main function of tokenizers is to convert text into data that the MLM can process.
* **Data sets.** Hugging Face offers an extensive collection of NLP data sets that can be accessed, downloaded and processed.
* **Inference application programming interface.** These are hosted APIs for pretrained language models that can be used for various NLP tasks.
* **Model Hub.** Model Hub is a resource for discovering, sharing and deploying transformer models that have already been trained.

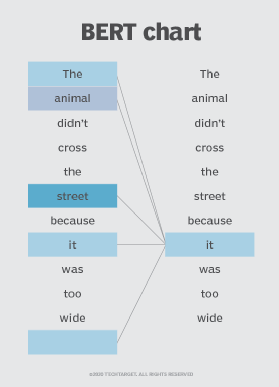
Many pretrained deep learning models, such as BERT, GPT-2 and Google's Text-to-Text Tranfer Transformer (T5), are available in their well-known transformers collection, along with resources for optimizing these models for particular workloads. Hugging Face aims to promote NLP research and democratize access to cutting-edge [AI technologies and trends](https://www.techtarget.com/searchenterpriseai/tip/9-top-AI-and-machine-learning-trends).

**Masked language modeling in BERT**

The BERT model is an example of a pretrained MLM that consists of multiple layers of transformer encoders stacked on top of each other. [Various large language models](https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models), such as BERT, use a fill-in-the-blank approach in which the model uses the context words around a mask token to anticipate what the masked word should be.

BERT is classified into two types -- BERTBASE and BERTLARGE -- based on the number of encoder layers, self-attention heads and hidden vector size. For the masked language modeling task, the BERTBASE architecture used is bidirectional. This means that it considers both the left and right context for each token. Because of this bidirectional context, the model can capture dependencies and interactions between words in a phrase.

This BERT method is ideal for training a language model in a self-supervised situation without human-annotated labels. The model can then be fine-tuned for various supervised NLP tasks.

Bidirectional BERT language modeling involves understanding the context of a word by considering both its left and right surroundings.

**Benefits of masked language models**

MLMs offer several benefits in NLP tasks. Key advantages of MLMs include the following:

* **Enhanced contextual understanding.** By forecasting masked tokens depending on the surrounding context, MLMs help language models learn contextual information. This makes it possible for the model to represent the connections and dependencies among words in a sequence.
* **Bidirectional information.** During training, MLMs such as BERT consider the context of a masked token. This bidirectional strategy results in better language understanding, which helps the model derive meaning and context from the words that surround a given word.
* **Pretraining for downstream tasks.** Masked language modeling works as an effective pretraining technique for different downstream NLP tasks. MLMs can acquire broad language representations that can be optimized for particular tasks such as [sentiment analysis](https://www.techtarget.com/searchbusinessanalytics/definition/opinion-mining-sentiment-mining), text categorization, named entity recognition and question answering by pretraining on vast amounts of unlabeled data.
* **Semantic similarity.** MLMs can quantify semantic similarity between sentences or phrases. Through a comparison of the representations of masked tokens in distinct sentences, an MLM can discern the similarity or correlation within the underlying text.
* **Transfer learning.** MLMs such as BERT showcase strong transfer learning capabilities. The initial pretraining on an extensive corpus lets the model grasp general language understanding, subsequently enabling fine-tuning on smaller labeled [data sets](https://www.techtarget.com/searchdatamanagement/definition/SQL) that are tailored to specific tasks.

**How is MLM different from CLM?**

The two main language modeling approaches are masked language modeling and causal language modeling (CLM). The following points highlight the differences between the two models:

* Masked language modeling is a self-supervised learning process that involves training a linguistic model to predict masked tokens in a sequence. While CLM is also a self-supervised learning task, the language model in CLM is trained to predict the next word in a sequence given the previous words.
* In masked language modeling, a certain percentage of tokens in a sequence are randomly masked and the model is trained to estimate the original value of these masked tokens based on the context provided by the other tokens in the sequence. With CLM, the model is trained to produce the subsequent word in a sequence by using the context the preceding words provide.
* Language models such as BERT typically use masked language modeling as their pretraining objective, while CLM is utilized by GPT models such as GPT-2 for their pretraining goal.
* Masked language modeling helps the model learn context and grasp bidirectional relationships between words in a sentence. Since CLM only considers the words that came before it when predicting a word, it concentrates on capturing the unidirectional dependencies in a sequence.

In short, both masked language modeling and CLM are [self-supervised learning tasks](https://www.techtarget.com/searchenterpriseai/feature/Comparing-supervised-vs-unsupervised-learning) used in language modeling. Masked language modeling predicts masked tokens in a sequence, enabling the model to capture bidirectional dependencies, while CLM predicts the next word in a sequence, focusing on unidirectional dependencies. Both approaches have been successful in pretraining language models and have been used in various NLP applications.

**How is MLM different from Word2Vec?**

Similar to masked language modeling and CLM, Word2Vec is an approach used in NLP where the vectors capture the semantics of the words and the relationships between them by using a neural network to learn the vector representations.

However, Word2Vec differs from self-supervised training models such as masked language modeling in the following ways:

* Word2Vec is an unsupervised learning algorithm that's used to generate word embeddings.
* It captures the syntactic and semantic links between words by representing them as dense vectors in a continuous vector space.
* Word2Vec acquires word embeddings by training on large corpora and predicting the context of words within a designated text window, encompassing either the target word itself or the surrounding words.
* It can be trained using two different algorithms -- [Continuous Bag of Words](https://medium.com/@codethulo/understanding-the-continuous-bag-of-words-cbow-model-architecture-working-mechanism-and-math-78c7284a8d5a) and Skip-Gram.
* Word2Vec embeddings are often used to measure word similarity or as input features for downstream natural language processing tasks.

1. **Explain Next Sentence Prediction (NSP)**

**Next Sentence Prediction using BERT**

**Pre-requisite**: [BERT-GFG](https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/)

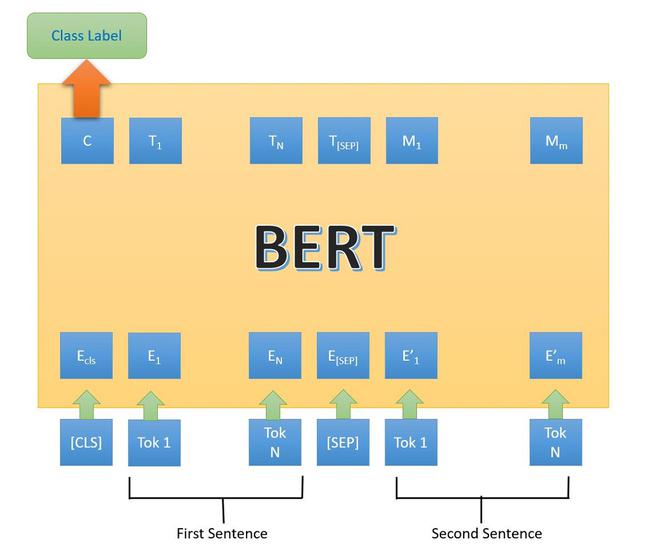
BERT stands for **Bidirectional Representation for Transformers**. It was proposed by researchers at Google Research in 2018. Although, the main aim of that was to improve the understanding of the meaning of queries related to Google Search. A study shows that Google encountered 15% of new queries every day. Therefore, it requires the Google search engine to have a much better understanding of the language in order to comprehend the search query.

However, BERT is trained on a variety of different tasks to improve the language understanding of the model. In this article, we will discuss the tasks under the next sentence prediction for BERT.

**Next Sentence Prediction Using BERT**

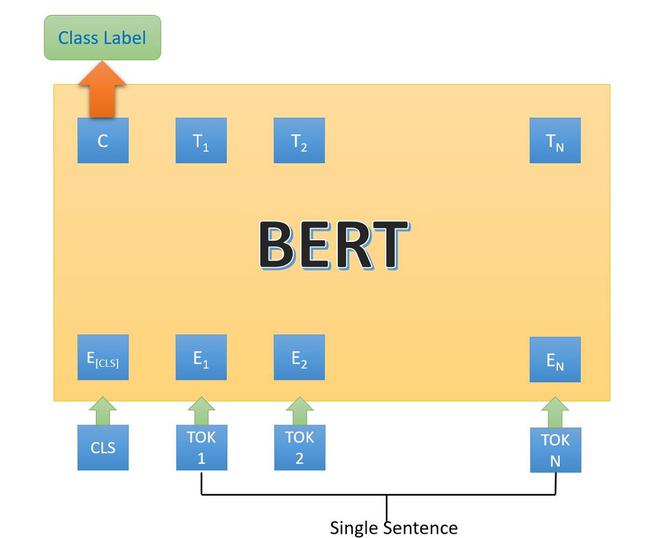
BERT is fine-tuned on 3 methods for the next sentence prediction task:

* In the first type, we have sentences as input and there is only one class label output, such as for the following task:
  + **MNLI**(Multi-Genre Natural Language Inference)**:** It is a large-scale classification task. In this task, we have given a pair of sentences. The goal is to identify whether the second sentence is entailment, contradiction, or neutral with respect to the first sentence.
  + **QQP**(Quora Question Pairs): In this dataset, the goal is to determine whether two questions are semantically equal.
  + **QNLI** (Question Natural Language Inference): In this task, the model needs to determine whether the second sentence is the answer to the question asked in the first sentence.
  + **SWAG** (Situations With Adversarial Generations): This dataset contains 113k sentence classifications. The task is to determine whether the second sentence is the continuation of the first or not.



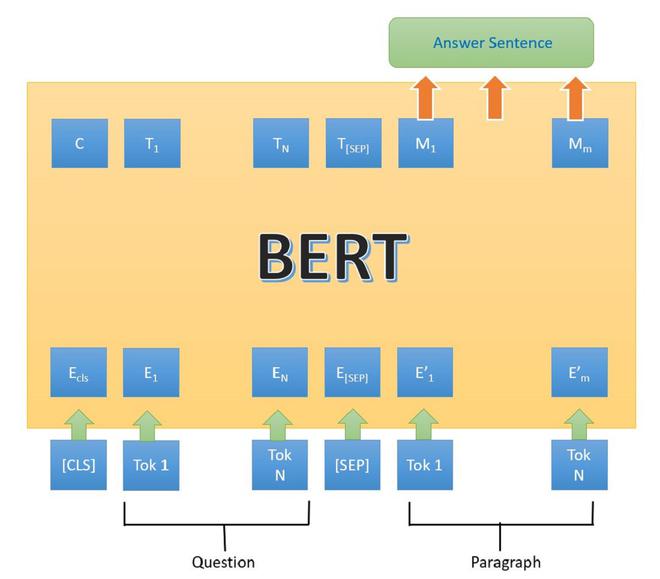
*BERT architecture first type*

* In the second type, we have only one sentence as input, but the output is similar to the next class label. Following are the task/datasets used for it:
  + **SST-2**(The Stanford Sentiment Treebank): It is a binary sentence classification task consisting of sentences extracted from movie reviews with annotations of their sentiment representing in the sentence. BERT generated state-of-the-art results on SST-2.
  + **CoLA:**(Corpus of Linguistic Acceptability): is the binary classification task. The goal of this task to predict whether an English sentence that is provided is linguistically acceptable or not.



*BERT architecture second type*

* In the third type of next sentence, prediction, we have been provided with a question and paragraph and outputs a sentence from the paragraph that is the answer to that question. It is performed on SQuAD (Stanford Question Answer D) v1.1 and 2.0 datasets.



*BERT architecture 3rd type.*

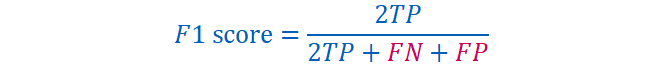
In the above architecture, the [CLS] token is the first token in the input. This means an input sentence is coming, the [SEP] represents the separation between the different inputs. Here, the inputs sentence are tokenized according to BERT vocab, and output is also tokenized.

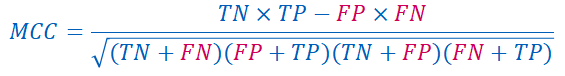
1. **What is Matthews evaluation?**

**I got to know the Matthews Correlation Coefficient (MCC) when I run into**[**this question on StackExchange**](https://datascience.stackexchange.com/questions/51808/multiclass-classification-on-imbalanced-dataset-accuracy-or-micro-f1-or-macro?rq=1)**regarding metrics for imbalanced classification problems. Boaz did an excellent job in explaining the advantages of using MCC in his Medium Story titled**[***‘Matthews Correlation Coefficient Is The Best Classification Metric You’ve Never Heard Of’***](https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3bf50a2f3e9a)**, and I believe many people are inspired and excited to use it when they encounter a tough imbalanced classification problem next time.**

**But wait… the *best*classificationmetric?**

**Let’s consider MCC more carefully in the context of binary classification problems. Similar to F1 score, MCC is a single-value metric that summarizes the confusion matrix. A confusion matrix, also known as an error matrix, has four entries: True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The main benefit gained by using MCC instead of F1 score can be guessed easily by peeking into their formulae:**

****

****

**FP and FN are in red when I typed them, as they are not desirable**

**F1 score ignores the count of True Negatives. In contrast, MCC kindly extends its care to all four entries of the confusion matrix. Davide Chicco, the author of*Ten quick tips for machine learning in computational biology*, commented that MCC “is high only if your classifier is doing well on both the negative and the positive elements.”**

**Let’s now put our first feeling aside and look at a concrete example (the same example used in**[**MCC’s wiki page**](https://en.wikipedia.org/wiki/Matthews_correlation_coefficient)**):**

**Confusion matrix with entries: TP = 90, FP = 4; TN = 1, FN = 5.**

**F1 score = 0.9524, which misleads us into believing that the classifier is extremely good. In contrast, by plugging in those numbers in the formula of MCC, we get a miserable 0.14. MCC ranges from -1 to 1 (hey, it is a correlation coefficient anyway) and 0.14 means the classifier is very close to a random guess classifier. From this example, we can tell that MCC helps one identify the ineffectiveness of the classifier in classifying especially the negative class samples.**

**However, F1 score is highly influenced by which class is labeled as positive. The F1 score differs so much from MCC in magnitude as the minority class is labeled as negative. Let’s look at another example by reversing the positive and negative labels:**

**Confusion matrix with entries: TP = 1, FP = 5; TN = 90, FN = 4.**

**F1 score is 0.18, and MCC is 0.103. Both metrics send a signal to the practitioner that the classifier is not performing well.**

**F1 score is usually good enough**

**It is important to recognize that the majority class is normally labeled as negative (as in the second example). This is because of the convention that rarer or more ‘interesting’ samples are usually labeled as positive, such as patients who have a rare disease (they are tested positive). For these problems, F1 score fulfills its purpose of being a good metric by placing more emphasis on the positive class.**

**F1 score may be preferred over MCC**

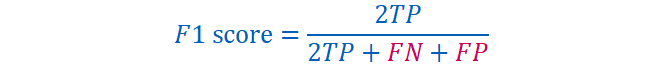
**The hard work that we have put in building a good model and selecting informative metrics serves a primary purpose: solve the real life problem. Therefore, try not to jump to conclusion that a certain method or methodology is the ‘best’, as one size can hardly fit all. If the data analytics practitioner is blessed with business domain knowledge, he or she might know how precision and recall are close to the heart of their clients. F1 score, in this case, merges precision and recall in a more interpretable way than MCC does. Only if the cost of low precision and low recall is really unknown or unquantifiable, MCC is preferred over F1 score as it is a more ‘balanced’ assessment of classifiers, no matter which class is positive.**

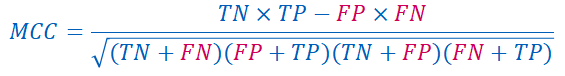
1. **What is Matthews Correlation Coefficient (MCC)?**

**I got to know the Matthews Correlation Coefficient (MCC) when I run into**[**this question on StackExchange**](https://datascience.stackexchange.com/questions/51808/multiclass-classification-on-imbalanced-dataset-accuracy-or-micro-f1-or-macro?rq=1)**regarding metrics for imbalanced classification problems. Boaz did an excellent job in explaining the advantages of using MCC in his Medium Story titled**[***‘Matthews Correlation Coefficient Is The Best Classification Metric You’ve Never Heard Of’***](https://towardsdatascience.com/the-best-classification-metric-youve-never-heard-of-the-matthews-correlation-coefficient-3bf50a2f3e9a)**, and I believe many people are inspired and excited to use it when they encounter a tough imbalanced classification problem next time.**

**But wait… the *best*classificationmetric?**

**Let’s consider MCC more carefully in the context of binary classification problems. Similar to F1 score, MCC is a single-value metric that summarizes the confusion matrix. A confusion matrix, also known as an error matrix, has four entries: True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The main benefit gained by using MCC instead of F1 score can be guessed easily by peeking into their formulae:**





**FP and FN are in red when I typed them, as they are not desirable**

**F1 score ignores the count of True Negatives. In contrast, MCC kindly extends its care to all four entries of the confusion matrix. Davide Chicco, the author of*Ten quick tips for machine learning in computational biology*, commented that MCC “is high only if your classifier is doing well on both the negative and the positive elements.”**

**Let’s now put our first feeling aside and look at a concrete example (the same example used in**[**MCC’s wiki page**](https://en.wikipedia.org/wiki/Matthews_correlation_coefficient)**):**

**Confusion matrix with entries: TP = 90, FP = 4; TN = 1, FN = 5.**

**F1 score = 0.9524, which misleads us into believing that the classifier is extremely good. In contrast, by plugging in those numbers in the formula of MCC, we get a miserable 0.14. MCC ranges from -1 to 1 (hey, it is a correlation coefficient anyway) and 0.14 means the classifier is very close to a random guess classifier. From this example, we can tell that MCC helps one identify the ineffectiveness of the classifier in classifying especially the negative class samples.**

**However, F1 score is highly influenced by which class is labeled as positive. The F1 score differs so much from MCC in magnitude as the minority class is labeled as negative. Let’s look at another example by reversing the positive and negative labels:**

**Confusion matrix with entries: TP = 1, FP = 5; TN = 90, FN = 4.**

**F1 score is 0.18, and MCC is 0.103. Both metrics send a signal to the practitioner that the classifier is not performing well.**

**F1 score is usually good enough**

**It is important to recognize that the majority class is normally labeled as negative (as in the second example). This is because of the convention that rarer or more ‘interesting’ samples are usually labeled as positive, such as patients who have a rare disease (they are tested positive). For these problems, F1 score fulfills its purpose of being a good metric by placing more emphasis on the positive class.**

**F1 score may be preferred over MCC**

**The hard work that we have put in building a good model and selecting informative metrics serves a primary purpose: solve the real life problem. Therefore, try not to jump to conclusion that a certain method or methodology is the ‘best’, as one size can hardly fit all. If the data analytics practitioner is blessed with business domain knowledge, he or she might know how precision and recall are close to the heart of their clients. F1 score, in this case, merges precision and recall in a more interpretable way than MCC does. Only if the cost of low precision and low recall is really unknown or unquantifiable, MCC is preferred over F1 score as it is a more ‘balanced’ assessment of classifiers, no matter which class is positive.**

1. **Explain Semantic Role Labeling**

**What is semantic role labeling (SRL)?**

**Get Started With Data Science**

Learn the fundamentals of Data Science with this free course. Future-proof your career by adding Data Science skills to your toolkit — or build foundational Data Science skills to land a job in AI, Machine Learning, or Data Analysis.

[Get Free Course](https://www.educative.io/courses/data-science-handbook)

In our daily lives, we effortlessly use semantic roles to understand the relationships, intentions, and dynamics within language. We can see this in the following example. “This Answer is written by the Educative Team,” where humans can easily understand that “the Educative Team” is the responsible agent for the predicate “is written” to the object “Answer”. However, capturing and comprehending these roles can be challenging for computer software, as it requires contextual understanding and the ability to infer the implied meaning beyond literal interpretation. This example shows one of the most minor yet quite significant tasks done in the field of natural language processing (NLP).

**What is NLP?**

**Natural language processing (NLP)** is a field that combines artificial intelligence and computational linguistics to enable computers to interact with human language. Primarily, its objective is to overcome the challenges of understanding and processing human language by developing techniques and methods. These techniques involve processing, analyzing, and generating natural language text or speech. NLP plays a vital role in various applications such as machine translation, sentiment analysis, chatbots, and information retrieval, revolutionizing the way we interact with technology.

A basic illustration of how NLP works.

One of the most common examples of seeing NLP in action is through the famous ChatGPT. It enables the model to understand and generate human-like text in conversations, allowing ChatGPT to do tasks such as:

* Process user inputs
* Extract relevant information
* Generate contextually appropriate responses
* Provide a more interactive conversational experience

Having discussed NLP, we can now shift our focus to semantic role labeling (SRL) and its applications in this highly tech-influenced era.

**What is SRL?**

In linguistics, a semantic predicate, also known as the main verb, is a word or phrase that expresses the main action, state, or occurrence in a sentence. For this to be analyzed by computer software, we use semantic role labeling (SRL), also known as thematic role labeling. SRL is an NLP task that involves assigning semantic roles to words or phrases in a sentence and capturing their relationships to the main predicate.

The goal of SRL is to understand the underlying meaning and roles played by different entities, such as agents, patients, and locations, in expressing an action or event through a sentence. SRL plays a crucial role in revealing the underlying structure of a sentence, enabling more profound analysis and comprehension of the text.

Its applications contribute to various NLP tasks by uncovering the semantic relationships within sentences. SRL can be divided into four subtasks, each serving a specific purpose in the overall process. Let’s see these steps in a little more detail.

**Overall steps in SRL**

Before we get into the critical steps in the process, we have to focus on the preprocessing. This step involves tokenizing the text into individual words or subword units and performing preprocessing tasks like sentence segmentation, part-of-speech tagging, and syntactic parsing.

In syntactic parsing, there are different types of parse trees, such as constituent parse trees and dependency parse trees. These trees capture the syntactic structure of the sentence and provide the foundation for subsequent steps in the SRL process, such as predicate identification and argument identification. Let’s see an example of a syntactic tree.

Illustration of syntactic tree

**Predicate detection**

This subtask involves identifying the predicates or main verbs in a sentence. The goal is to locate the words that express the main actions or events in the sentence. For example, in the sentence "Educative is a hands-on learning platform for software developers of all levels, the phrase "is a hands-on learning platform" would be identified as the predicate.

**Predicate disambiguation**

Sometimes, a sentence may contain multiple potential predicates, making it necessary to disambiguate and determine the correct one. Predicate disambiguation resolves any ambiguity in identifying the primary verb. For example, in the sentence "Educative offers various courses for learners of all ages" the word "offers" can be either a verb or a noun. Predicate disambiguation helps in correctly identifying "offers" as the verb.

For the above steps, we can use any pruning algorithm for discarding words and selecting the right ones in the syntactic tree constructed. Candidates that are evidently not arguments for a given predicate are eliminated to optimize training time and, more significantly, enhance performance.

**Argument identification**

After identifying the predicate, the next step is to identify the words or phrases that serve as arguments for the predicate. Arguments are the entities or elements that participate in the action or state expressed by the predicate. For example, in the sentence "Educative offers a wide range of courses for learners of all backgrounds, the noun phrase "a wide range of courses" would be identified as the argument of the predicate "offers."

**Argument classification**

Once the arguments are identified, this subtask involves classifying or labeling the roles or semantic functions of each argument. For example, in the sentence "Educative offers a wide range of courses for learners of all backgrounds, the argument "Educative" would be assigned the role of "subject" or "agent" since it is the entity performing the action expressed by the predicate.

**Applications of SRL**

Semantic Role Labeling (SRL) has various applications in natural language processing (NLP) and language understanding. Here are some common applications of SRL:

* **Question answering:** SRL helps identify the semantic roles of words or phrases in a question, enabling a better understanding of the question's structure and extracting relevant information to provide accurate answers.
* **Information extraction:** SRL extracts structured information from unstructured text by identifying and labeling the semantic roles of entities and relationships within the text.
* **Text summarization:**SRL summarizes text by identifying key arguments and their associated roles, helping to generate concise summaries that capture the main points of a document.
* **Sentiment analysis:** SRL can be used in sentiment analysis tasks to identify the semantic roles of words or phrases that express sentiment or opinion, providing a more nuanced understanding.
* **Information retrieval**: SRL enhances information retrieval systems by identifying the semantic roles of search queries and matching them with relevant documents or web pages.
* **Dialogue systems:**SRL can aid in dialogue systems and chatbots such as GPT-3 by understanding the semantic roles of user queries and generating appropriate responses.
* **Text-to-speech synthesis:**SRL helps generate more natural and contextually appropriate speech output by capturing the semantic roles and relationships in the input text.

1. **Why Fine-tuning a BERT model takes less time than pretraining**

Training and fine-tuning are crucial stages in the machine learning model development lifecycle, serving distinct purposes. This article explains the intricacies of both methodologies, highlighting their differences and importance in ensuring optimal model performance.

Training in the context of deep learning and neural networks refers to the phase where a new model learns from a dataset. During this phase, the model adjusts its model weights based on the input data and the corresponding output, often using embeddings and activation functions. While embeddings and activation functions play significant roles in certain model architectures and tasks, they are not universally employed during the training phase of all deep learning models. It's crucial to understand the specific context and model architecture to determine their relevance.

The objective is to diminish the discrepancy between the anticipated and factual output, frequently termed error or loss. This is predominantly achieved using algorithms like backpropagation and optimization techniques like gradient descent.

Fine-tuning, conversely, follows the initial training, where a pre-trained model (previously trained on a vast dataset like **[ImageNet](https://www.image-net.org/" \t "_blank)**) is trained on a smaller, task-specific dataset. The rationale is to leverage the knowledge the model has acquired from the initial training process and tailor it to a more specific task. This becomes invaluable, especially when the new dataset for the new task is limited, as training from scratch might lead to overfitting.

As training stars, the neural network's weights are randomly initialized or set using methods like He or Xavier initialization. These weights are fundamental in determining the model's predictions. As the training progresses, these weights adjust to minimize the error, guided by a specific learning rate.

Conversely, during fine-tuning, the model starts with pre-trained weights from the initial training, which are then fine-tuned to suit the new task better, often involving techniques like unfreezing certain layers or adjusting the batch size.

The training aims to discern patterns and features from the data, creating a base model that excels on unseen data and is often validated using validation sets. Fine-tuning, however, zeroes in on adapting a generalized model for a specific task, often leveraging transfer learning to achieve this.

While training focuses on generalizing models, fine-tuning refines this knowledge to cater to specific tasks, making it a crucial topic in [**NLP**](https://encord.com/glossary/nlp-definition/) with models like BERT, [**computer vision**](https://encord.com/glossary/computer-vision-definition/) tasks like image classification, and, more recently, the proliferation of [**foundation models**](https://encord.com/blog/foundation-models/).

**The Training Process**

**Initialization of Weights**

**Random Initialization**

In deep learning, initializing the weights of [**neural networks**](https://encord.com/glossary/neural-networks-definition/) is crucial for the training process. Random initialization is a common method where weights are assigned random values. This method ensures a break in symmetry among neurons, preventing them from updating similarly during backpropagation. However, random initialization can sometimes lead to slow convergence or the [**vanishing gradient problem**](https://en.wikipedia.org/wiki/Vanishing_gradient_problem).

**He or Xavier Initialization**

Specific strategies, like He or Xavier initialization, have been proposed to address the challenges of random initialization. He initialization, designed for ReLU activation functions, initializes weights based on the size of the previous layer, ensuring that the variance remains consistent across layers. On the other hand, Xavier initialization, suitable for tanh activation functions, considers the sizes of the current and previous layers. These methods help with faster and more stable convergence.

**Backpropagation and Weight Updates**

**Gradient Descent Variants**

Backpropagation computes the gradient of the loss function concerning each weight by applying the [**chain rule**](https://en.wikipedia.org/wiki/Chain_rule). Various gradient descent algorithms update the weights and minimize the loss. The most basic form is the [**Batch Gradient Descent**](https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch). However, other variants like [**Stochastic Gradient Descent**](https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch) (SGD) and [**Mini-Batch Gradient Descent**](https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch) have been introduced to improve efficiency and convergence.

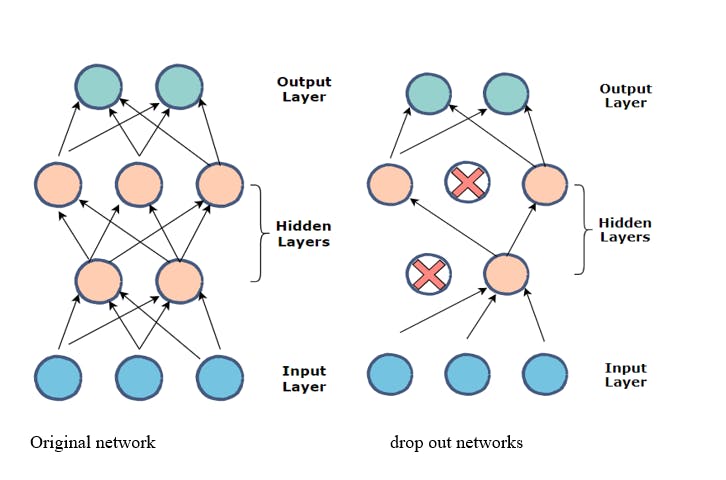
**Role of Learning Rate**

The learning rate is a hyperparameter that dictates the step size during weight updates. A high learning rate might overshoot the optimal point, while a low learning rate might result in slow convergence. Adaptive learning rate methods like [**Adam**](https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/), **[RMSprop](https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/" \t "_blank)**, and **[Adagrad](https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/" \t "_blank)** adjust the learning rate during training, facilitating faster convergence without manual tuning.

**Regularization Techniques**

**Dropout**

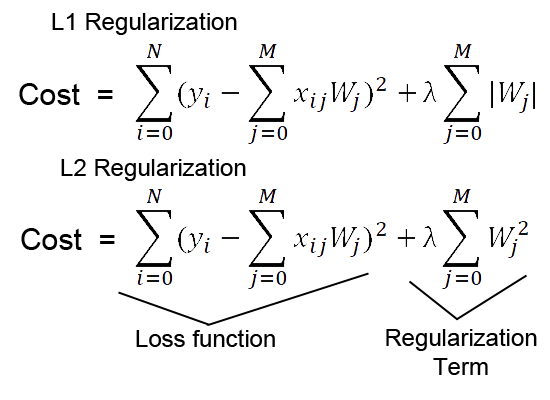
Overfitting is a common pitfall in deep learning, where the model performs exceptionally well on the training data but needs to improve on unseen data. Dropout is a regularization technique that mitigates overfitting. During training, random neurons are "dropped out" or deactivated at each iteration, ensuring the model does not rely heavily on any specific neuron.



[*Dropout Neural Networks*](https://medium.com/unpackai/introduction-of-dropout-and-ensemble-model-in-the-history-of-deep-learning-a4c2a512dcca)

**L1 and L2 Regularization**

L1 and L2 are other regularization techniques that add a penalty to the loss function. L1 regularization adds a penalty equivalent to the absolute value of the weights' magnitude, which aids feature selection. L2 regularization adds a penalty based on the squared magnitude of weights, preventing weights from reaching extremely high values. Both methods help in preventing overfitting, penalizing complex models, and producing a more generalized model.



[*L1 and L2 Regualization*](https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c)

**The Fine-tuning Process**

**Transfer Learning: The Backbone of Fine-tuning**

[**Transfer learning**](https://encord.com/blog/transfer-learning/) is a technique where a model developed for a task is adapted for a second related task. It is a popular approach in deep learning where pre-trained models are used as the starting point for computer vision and natural language processing tasks due to the extensive computational resources and time required to train models from scratch.

Pre-trained models save the time and resources needed to train a model from scratch. They have already learned features from large datasets, which can be leveraged for a new task with a smaller dataset. This is especially useful when acquiring labeled data is challenging or costly.

When fine-tuning, it's common to adjust the deeper layers of the model while keeping the initial layers fixed. The rationale is that the initial layers capture generic features (like edges or textures), while the deeper layers capture more task-specific patterns. However, the extent to which layers are fine-tuned can vary based on the similarity between the new task and the original task.

**Strategies for Fine-tuning**

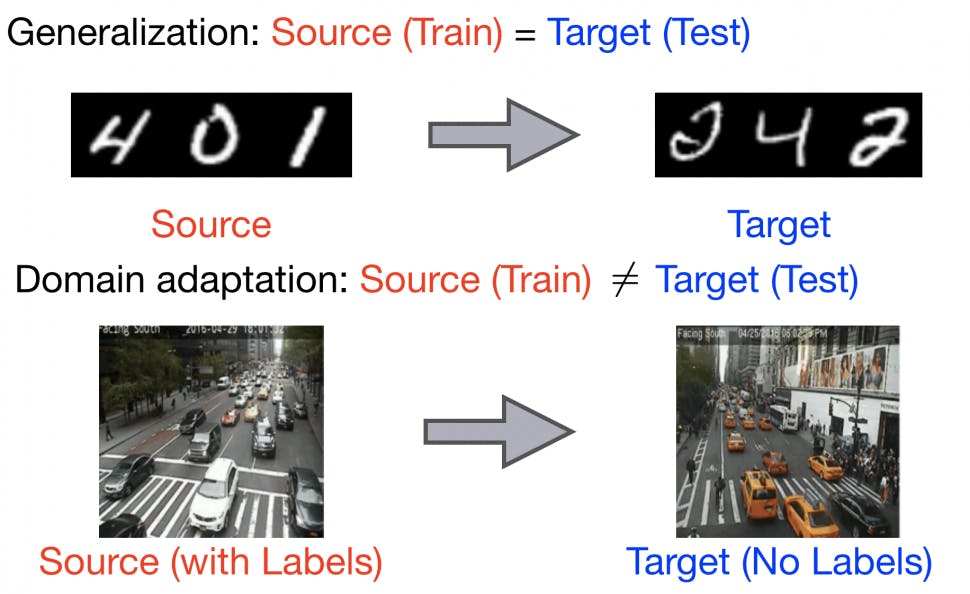
One of the key strategies in fine-tuning is adjusting the learning rates. A lower learning rate is often preferred because it makes the fine-tuning process more stable. This ensures the model retains the previously learned features without drastic alterations.

Another common strategy is freezing the initial layers of the model during the fine-tuning process. This means that these layers won't be updated during training. As mentioned, the initial layers capture more generic features, so fixing them is often beneficial.

**Applications and Use Cases**

**Domain Adaptation**

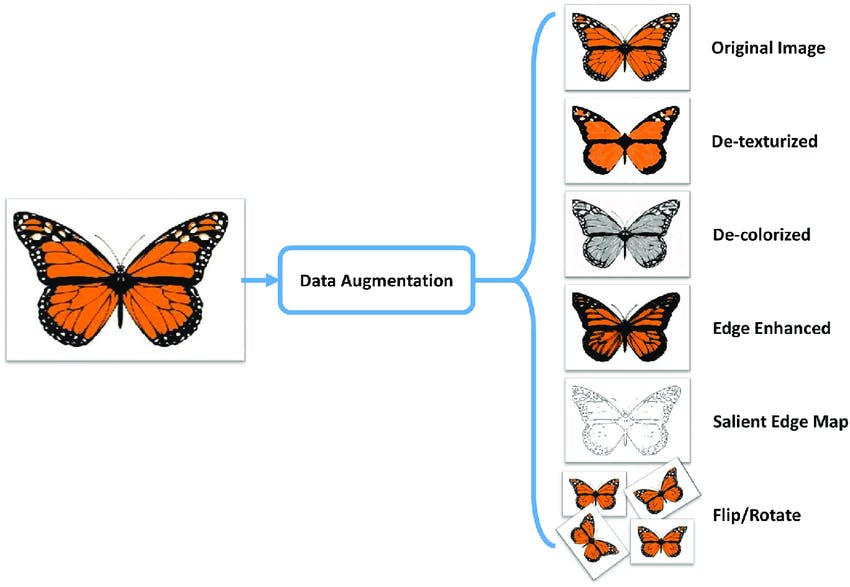
Domain adaptation refers to the scenario where the source and target tasks are the same, but the [**data distributions**](https://www.learninghub.ac.nz/distribution-of-data/) differ. Fine-tuning can be used to adapt a model trained on source data to perform well on target data.



[*Domain Adaptation*](https://blog.ml.cmu.edu/2019/09/13/on-learning-invariant-representations-for-domain-adaptation/)

**Data Augmentation**

Data augmentation involves creating new training samples by applying transformations (like rotations, scaling, and cropping) to the existing data. Combined with fine-tuning, it can improve the model's performance, especially when the available labeled data is limited.



[*Data Augmentation*](https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-your-model-but-how-aa1913468722)

**Comparative Analysis**

**Benefits of Training from Scratch**

* **Customization**: Training a model from scratch allows complete control over its architecture, making it tailored specifically for the task.
* **No Prior Biases**: Starting from scratch ensures the model doesn't inherit any biases or unwanted features from pre-existing datasets.
* **Deep Understanding**: Training a model from the ground up can provide deeper insights into the data's features and patterns, leading to a more robust model for specific datasets.
* **Optimal for Unique Datasets**: For datasets significantly different from existing ones, training from scratch might yield better results as the model learns features unique to that dataset.

**Limitations of Training from Scratch**

This approach requires more time as the model learns features from the ground up and requires a large, diverse dataset for optimal performance. With the right data and regularization, models can easily fit.

* **Extended Training Time**: Starting from the basics means the model has to learn every feature, leading to prolonged training durations.
* **Data Dependency**: Achieving optimal performance mandates access to a vast and varied dataset, which might only sometimes be feasible.
* **Risk of Overfitting**: Without adequate data and proper regularization techniques, models can overfit, limiting their generalization capabilities on unseen data.

**Advantages of Fine-Tuning**

* **Efficiency in Training**: Utilizing pre-trained models can expedite the training process, as they have already grasped foundational features from extensive datasets.
* **Data Economy**: Since the model has undergone training on vast datasets, fine-tuning typically demands a smaller amount of data, making it ideal for tasks with limited datasets.

**Limitations of Fine-Tuning**

* **Compatibility Issues**: Ensuring that the input and output formats, as well as the architectures and frameworks of the pre-trained model, align with the new task can be challenging.
* **Overfitting**: Fine-tuning on a small dataset can lead to overfitting, which reduces the model's ability to generalize to new, unseen data.
* **Knowledge Degradation**: There's a risk that the model might forget some of the features and knowledge acquired during its initial training, a phenomenon often referred to as "catastrophic forgetting."
* **Bias Propagation**: Pre-trained models might carry inherent biases. When fine-tuned, these biases can be exacerbated, especially in applications that require high sensitivity, such as facial recognition.

1. **Recognizing Textual Entailment (RTE)**

**Recognizing Textual Entailment (RTE) was proposed as a unified evaluation framework to compare semantic understanding of different NLP systems. In this survey paper, we provide an overview of different approaches for evaluating and understanding the reasoning capabilities of NLP systems. We then focus our discussion on RTE by highlighting prominent RTE datasets as well as advances in RTE dataset that focus on specific linguistic phenomena that can be used to evaluate NLP systems on a fine-grained level. We conclude by arguing that when evaluating NLP systems, the community should utilize newly introduced RTE datasets that focus on specific linguistic phenomena.**

1. **Explain the decoder stack of GPT models**.

**GPT models, however, do not use an encoder. Instead, they are with a decoder-only architecture. This means that the input data is fed directly into the decoder without being transformed into a higher, more abstract representation by an encoder.**

**The decoder in a GPT model uses a specific type of attention mechanism known as masked self-attention. In a traditional transformer, the attention mechanism allows the model to focus on all parts of the input when generating each part of the output. However, in a decoder-only transformer like GPT, the attention mechanism is “masked” to prevent it from looking at future parts of the input when generating each part of the output. This is necessary because GPT models are trained to predict the next word in a sentence, so they should not have access to future words.**