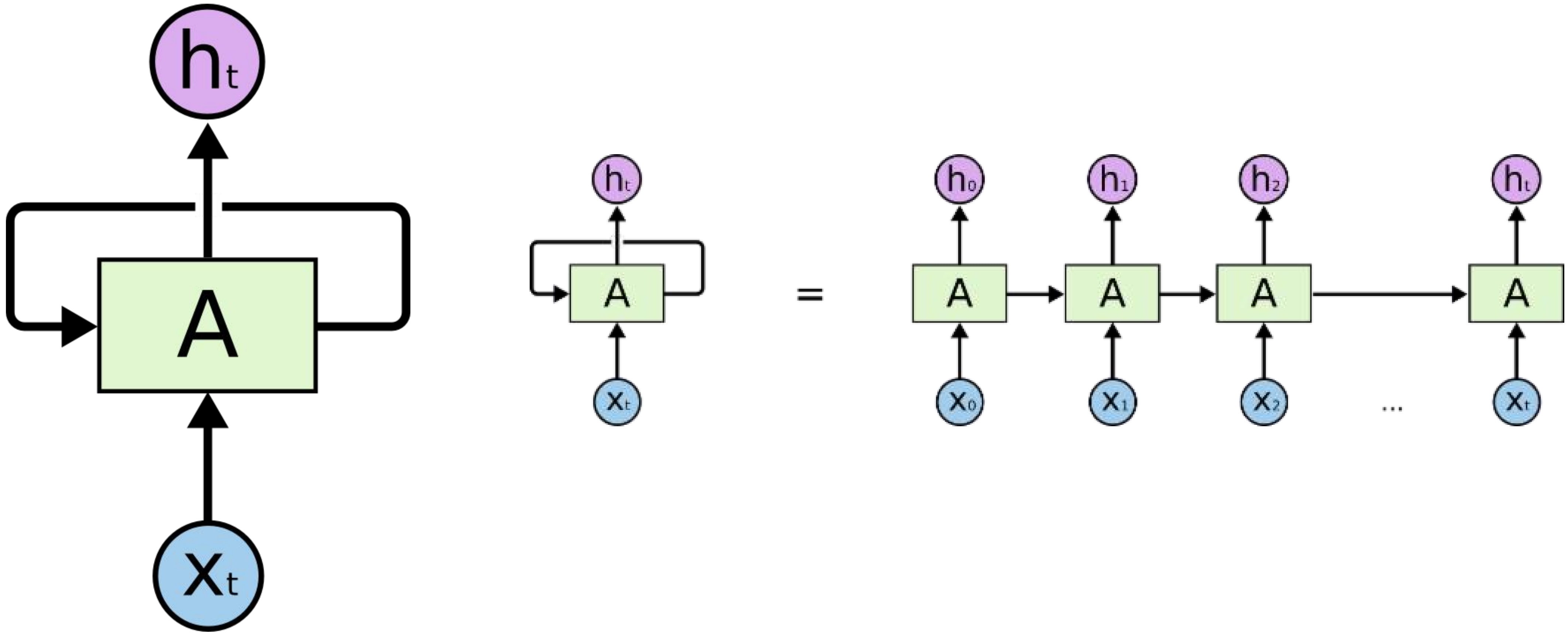
# Attention,Transformer And BERT

Bohao.Zou

# RNN (Sequence Model)



# Encoder-Decoder Model

- **ENCODER MODEL:**
  ① In the Encoder–Decoder framework, an encoder reads the input sentence, a sequence of vectors $\mathbf{x} = (x_1, \cdots, x_{T_x})$ , into a vector **c** .
  ② The most common approach is to use an RNN such that $h_t = f(x_t, h_{t-1})$ and $c = q(\{h_1, \cdots, h_{T_x}\})$
  ③ Where $h_t \in \mathbb{R}^n$ is a hidden state at time **t**, and **c** is a vector generated from the sequence of the hidden states . **f()** and **q()** are some **nonlinear** functions.**f()** and **q()** function are LSTM Model as usual .
- **DECODER MODEL :**
  ① The decoder is often trained to predict the next word $y_{t'}$ given the context vector **c** and all the previously predicted words $\{y_1, \cdots, y_{t'-1}\}$ .
  ② In other words, the decoder defines a probability over the translation **y** by decomposing the joint probability into the ordered conditionals :

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c)$$

  ③ With an RNN, each conditional probability is modeled as $p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c),$
  ④ where **g()** is a nonlinear, potentially multi-layered, function that outputs the probability of $y_{t'}$ and **St** is the hidden state of the RNN.

# Attention Mechanism

- In a new model architecture, we define each conditional probability in :
  $p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i)$
- Where **Si** is an RNN hidden state for time **i**, computed by: $s_i = f(s_{i-1}, y_{i-1}, c_i)$
- The context vector **Ci** depends on a sequence of annotations **(h1, · · · , hTx)** to which an encoder maps the input sentence. Each annotation **hi** contains information about the whole input sequence with a **strong focus on** the parts surrounding the **i-th** word of the input sequence.
- The context vector **Ci** is, then, computed as a **weighted sum** of these annotations **hi**:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

- The weight **αij** of each annotation **hj** is computed by: $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$
- $e_{ij} = a(s_{i-1}, h_j)$ is an **alignment model** which scores how well the inputs around **position j** and the output at **position i** match.
- We parametrize the alignment model **a()** as a **feedforward** neural network which is jointly **trained** with all the other components of the proposed system.

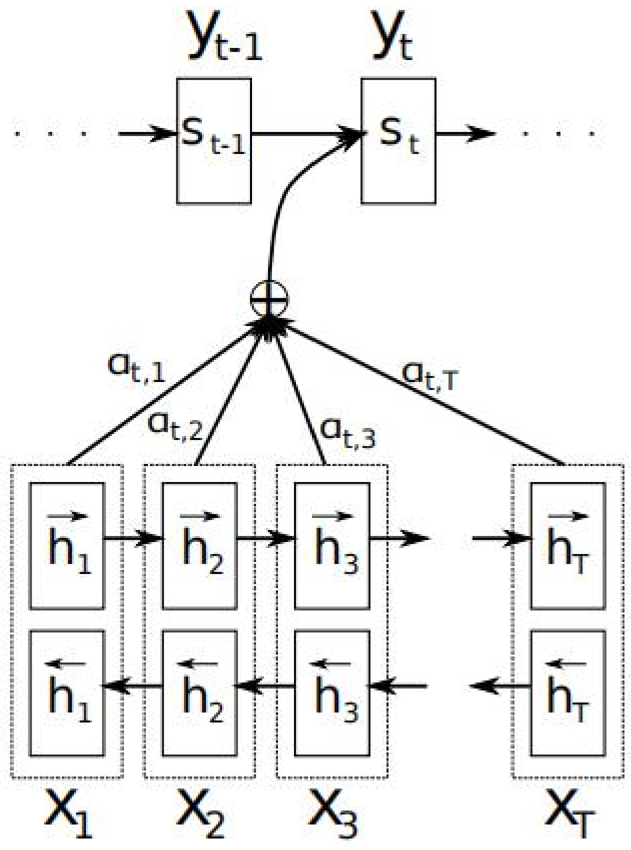



Figure 1: The graphical illustration of the proposed model trying to generate the $t$-th target word $y_t$ given a source sentence $(x_1, x_2, \ldots, x_T)$.

# Transformer Model (Self-Attention model)

1. Most competitive neural sequence transduction models have an encoder-decoder structure .

2. Here, the **encoder** maps an input sequence of symbol representations **(x1, ..., xn)** to a sequence of continuous representations **z = (z1, ..., zn)**.

3. Given **z**, the **decoder** then generates an output sequence **(y1, ..., ym)** of symbols one element at a time.

4. At each step the model is **auto-regressive** consuming the previously generated symbols as additional input when generating the next.

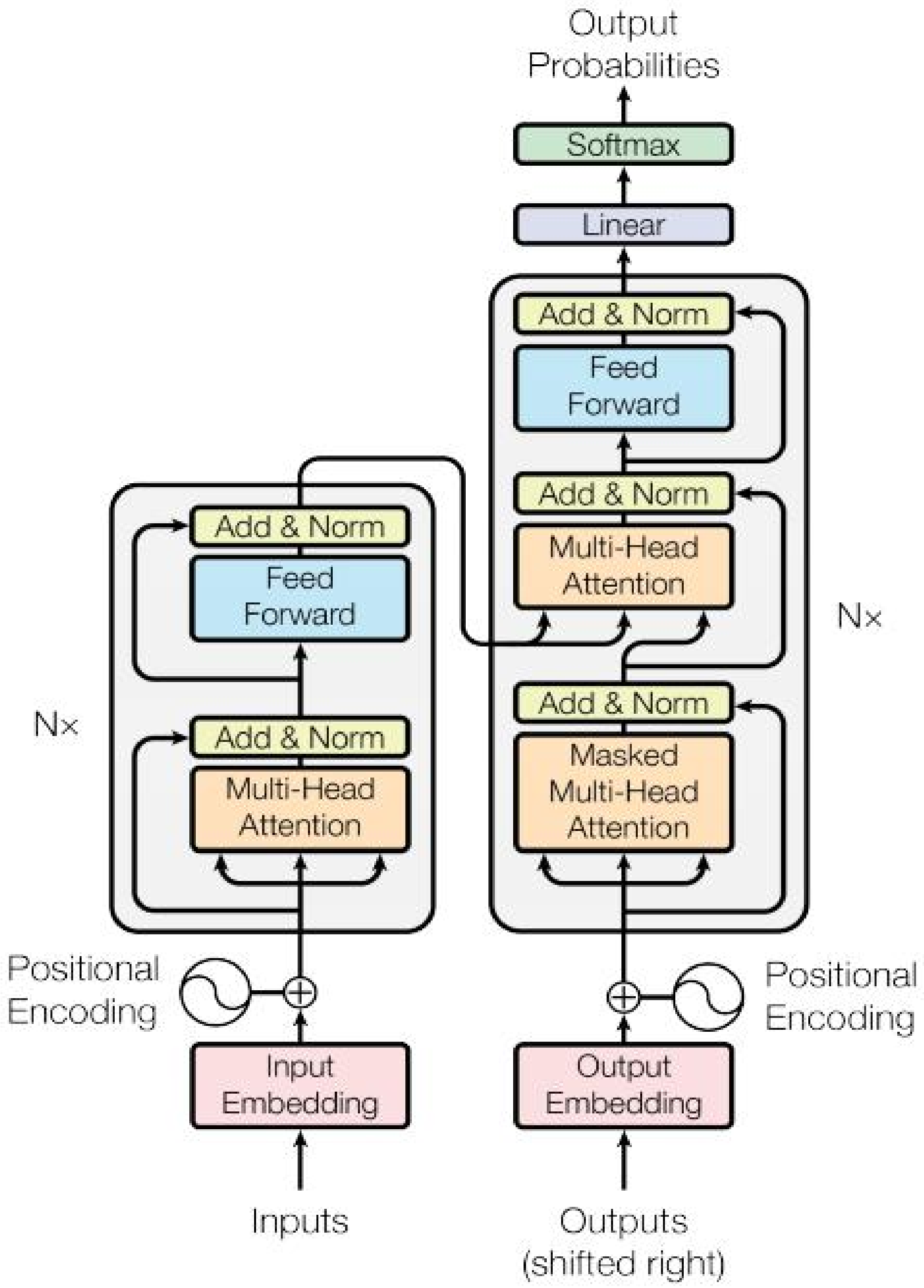


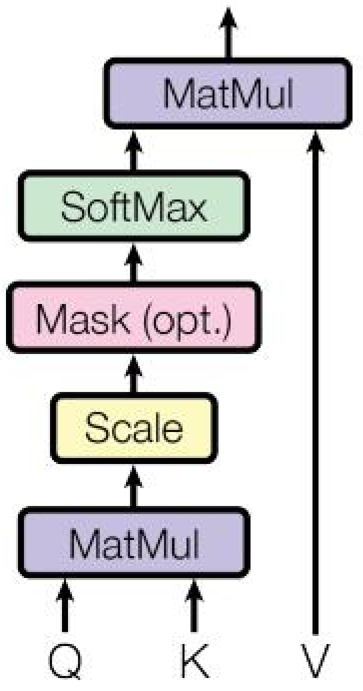

Figure 1: The Transformer - model architecture.

# Transformer Model

- Attention:
  1. An attention function can be described as **mapping a query** and a **set of key-value pairs to an output** , where the query, keys, values, and output are all vectors. The output is computed as a **weighted sum** of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
  2. We call our particular attention "**Scaled Dot-Product Attention**". The input consists of queries and keys of dimension **dk**, and values of dimension **dv**. We compute the dot products of the query with all keys, divide each by √dk, and apply a softmax function to obtain the weights on the values.
  3. In practice, we compute the attention function on a set of queries **simultaneously**, packed together into a **matrix Q**. The keys and values are also packed together into **matrices K** and **V** . We compute the matrix of outputs as:

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

Scaled Dot-Product Attention

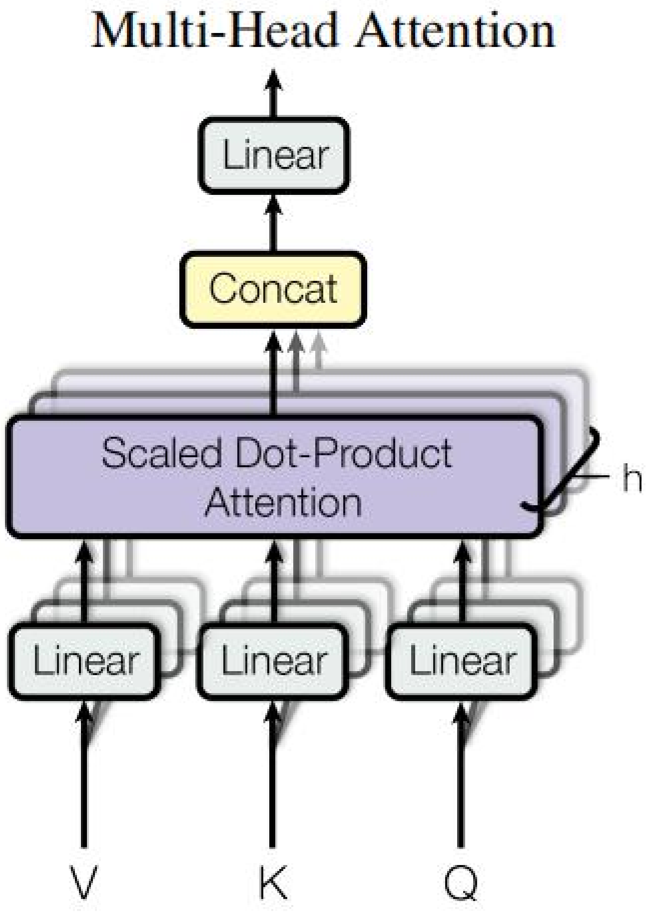


# Transformer Model

- Multi-Head Attention
  1. Instead of performing a single attention function , we found it **beneficial to linearly** project the queries, keys and values **h** times with different, learned linear projections to **dk**, **dk** and **dv** dimensions, respectively.
  2. On each of these queries, keys and values we can perform the attention function in parallel, yielding **dv-dimensional** output values.
  3. These are **concatenated** and once again do linearly trans, resulting in the final values.

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

  4. In this work we employ **h = 8** parallel attention layers, or heads. For each of these we use **dk = dv = 64.**



# Transformer Model

- The Transformer uses multi-head attention in three different ways:
  1. In "encoder-decoder attention" layers, the **queries** come from the previous **decoder** layer, and the memory **keys and values** come from the output of the **encoder**. This allows every position in the decoder to attend over all positions in the input sequence.
  2. The **encoder** contains self-attention layers. In a self-attention layer all of the keys, values and queries come **from the same place**, in this case,the queries , keys and values are the output of previous layer . **Each position in the encoder can attend to all positions in the previous layer of the encoder.**
  3. Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to **prevent rightward information flow in the decoder to preserve the auto-regressive property**. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

## BERT Model (pre-training model)

- BERT's model architecture is a multi-layer bidirectional **Transformer encoder.**
- **Input Representation :**

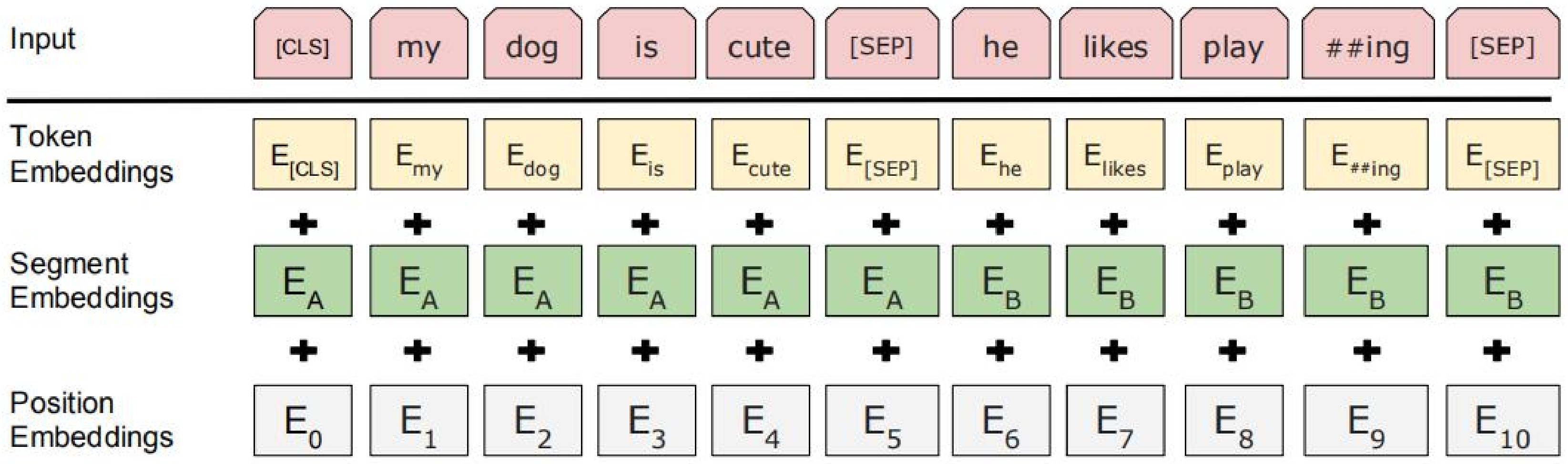



Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

# BERT Model

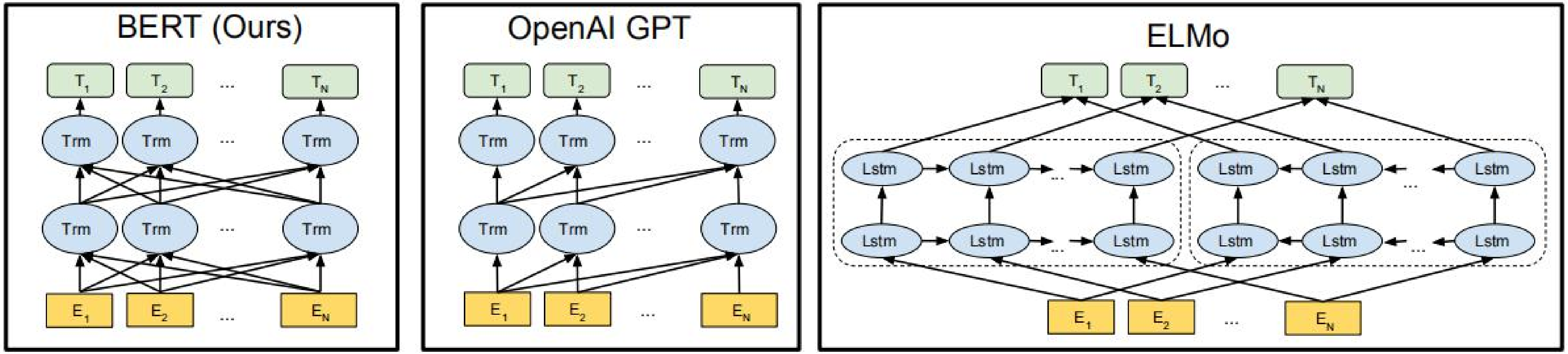



Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

# BERT Model

- **Task #1: Masked LM :**
  1. Intuitively, it is reasonable to believe that a deep bidirectional model is strictly **more powerful** than either a left-to-right model or the shallow concatenation of a left-to-right and right-to left model. **Unfortunately**, standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to **indirectly "see itself" in a multi-layered context.**
  2. In order to train a deep bidirectional representation, we take a straightforward approach of **masking some percentage of the input tokens at random,and then predicting only those masked tokens.**
  3. In this case, the final hidden vectors corresponding to the mask tokens are fed into an **output softmax over the vocabulary**, as in a standard LM.
  4. In all of our experiments, we mask 15% of all **WordPiece** tokens in each sequence at random.

Although this does allow us to obtain a bidirectional pre-trained model, there are two downsides to such an approach. The first is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token is never seen during fine-tuning. To mitigate this, we do not always replace "masked" words with the actual [MASK] token. Instead, the training data generator chooses 15% of tokens at random, e.g., in the sentence `my dog is hairy` it chooses `hairy`. It then performs the following procedure:

- Rather than *always* replacing the chosen words with [MASK], the data generator will do the following:
- 80% of the time: Replace the word with the [MASK] token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

# BERT Model

- Task #2: Next Sentence Prediction
    1. In order to train a model that **understands sentence relationships**, we pre-train a binarized next sentence **prediction task** that can be generated from any monolingual corpus.
    2. Specifically, when choosing the sentences A and B for each pre-training example, **50% of the time B is the actual next sentence** that follows A, and **50% of the time it is a random sentence from the corpus**. For example:

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

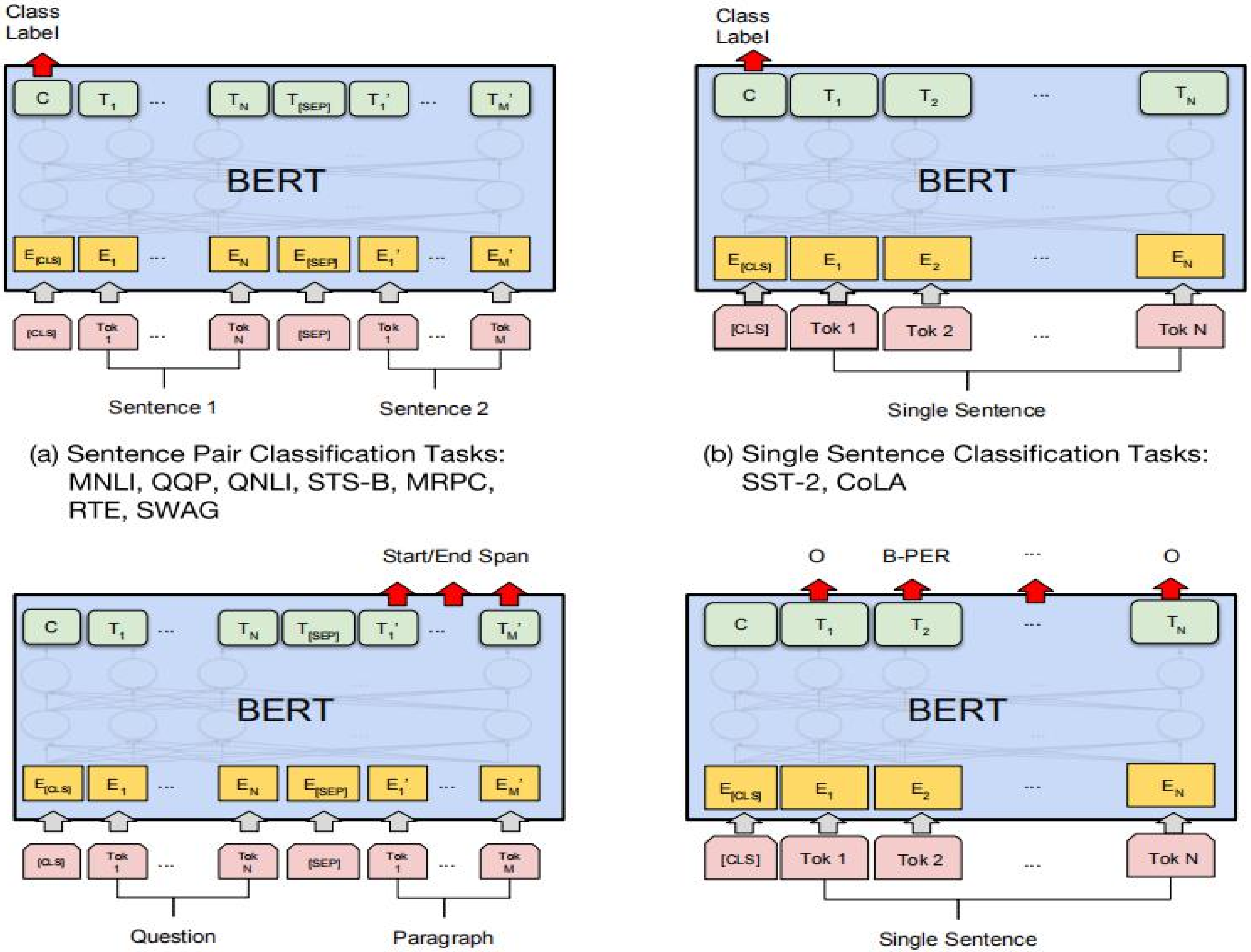
Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

3. We choose the NotNext sentences completely at random , and let BERT to predict the label of the input .

# BERT Workflow for different task




(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

# Actual Example (Recently)

Predict the PN_flag of every samples and extract the essential features of each sample .

| SampleID | LOD_grou | DilutionG | Var | PN_flag | D_VAF | VAF | alt_count | Score_BG_Polishing | Score_BGMut | Score_Cor | Score_Sec | Score_Str | Score_Low | Score_Low | Score_Low | Score_Poc | Score_Clo | Score_CloseEnd_Global |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FG204 | LODB | D1 | chr10:123 | 1 | 0.006541 | 0.005394 | 21 | 189.4953064 | 97.66627285 | 61.16071 | 118.7035 | 5.281518 | 0 | 0 | 0 | 0 | 11.05604 | 3.417356481 |
| FG204 | LODB | D1 | chr10:123 | 0 | 0 | 0.000606 | 3 | 18.16418435 | 14.78998978 | 7.123886 | 6.940914 | 1.741657 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG204 | LODB | D1 | chr10:123 | 1 | 0.01 | 0.009617 | 52 | 614.7483994 | 333.6835071 | 205.9421 | 288.1249 | 0.571941 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG204 | LODB | D1 | chr10:123 | 0 | 0 | 0.000333 | 2 | 5000 | 7.138189545 | 2.446487 | 24.71159 | 1.551233 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG204 | LODB | D1 | chr4:1803 | 1 | 0.008002 | 0.028531 | 40 | 514.5145467 | 430.3592644 | 276.6684 | 318.9257 | 80.5232 | 16.02537 | 0 | 0 | 0 | 2.310857 | 46.03690524 |
| FG204 | LODB | D1 | chr4:1806 | 1 | 0.007549 | 0.00446 | 13 | 115.8128376 | 177.8018275 | 169.2407 | 105.3149 | 1.906933 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG127 | LODA | D4 | chr10:123 | 1 | 0.001 | 0.001719 | 7 | 5000 | 49.16096352 | 58.11683 | 28.58304 | 2.082251 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG127 | LODA | D4 | chr10:123 | 0 | 0 | 0.000246 | 1 | 0.578200011 | 1.012990623 | 1.759155 | 4.11164 | 0.888634 | 5000 | 0 | 0 | 0 | 0 | 0 |
| FG127 | LODA | D4 | chr10:123 | 1 | 0.000808 | 0.001128 | 5 | 5000 | 24.81930492 | 37.11382 | 48.20922 | 2.055725 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG127 | LODA | D4 | chr4:1803 | 0 | 0 | 0.000814 | 1 | 7.066142558 | 4.11056414 | 3.979545 | 8.789825 | 1.607657 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG127 | LODA | D4 | chr4:1806 | 1 | 0.000542 | 0.002165 | 5 | 45.54388641 | 49.56296123 | 58.33123 | 32.20703 | 3.39762 | 0 | 0 | 0 | 0 | 12.49673 | 9.654966356 |
| FG073 | LODA | D4 | chr10:123 | 1 | 0.001 | 0.001502 | 6 | 5000 | 69.39463287 | 62.22684 | 44.31381 | 0.34931 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG073 | LODA | D4 | chr10:123 | 0 | 0 | 0.000251 | 1 | 0.611373025 | 1.910764638 | 1.314085 | 5.45727 | 0.870897 | 5000 | 0 | 0 | 0 | 0 | 0 |
| FG073 | LODA | D4 | chr10:123 | 0 | 0 | 0.000416 | 2 | 11.93512996 | 12.92037184 | 27.12391 | 4.025676 | 0.645413 | 0 | 0 | 0 | 0 | 17.4705 | 18.59809329 |
| FG073 | LODA | D4 | chr10:123 | 1 | 0.000808 | 0.001049 | 5 | 5000 | 25.06515891 | 25.63131 | 49.93389 | 9.278652 | 0 | 0 | 0 | 0 | 15.39264 | 9.64920527 |
| FG073 | LODA | D4 | chr4:1803 | 0 | 0 | 0.001275 | 2 | 10.5580047 | 6.35025303 | 11.35138 | 11.78537 | 0.652131 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG073 | LODA | D4 | chr4:1806 | 1 | 0.000542 | 0.001201 | 3 | 24.44808888 | 29.99839663 | 35.50502 | 24.57757 | 0.364921 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG305 | LODC | D1 | chr10:123 | 0 | 0 | 0.000219 | 1 | 3.381295004 | 0.511196269 | 0.571149 | 16.83985 | 1.01397 | 5000 | 0 | 0 | 0 | 0 | 0 |
| FG305 | LODC | D1 | chr10:123 | 1 | 0.01 | 0.008107 | 37 | 233.9479594 | 271.0012703 | 223.6123 | 80.81603 | 3.154286 | 0 | 0 | 0 | 0 | 16.33475 | 3.540795669 |
| FG305 | LODC | D1 | chr10:123 | 1 | 0.01115 | 0.009527 | 52 | 694.8238053 | 910.2854643 | 754.3592 | 194.2922 | 27.53365 | 0 | 0 | 0 | 0 | 0.770864 | 1.272073145 |
| FG305 | LODC | D1 | chr4:1803 | 1 | 0.01146 | 0.008977 | 23 | 5000 | 369.8688442 | 222.1836 | 297.0687 | 3.997876 | 21.81202 | 0 | 0 | 0 | 53.65954 | 65.32952772 |
| FG248 | LODC | D4 | chr10:123 | 1 | 0.001 | 0.000459 | 2 | 1.292726106 | 3.316991853 | 1.477309 | 14.15882 | 0.179537 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG248 | LODC | D4 | chr10:123 | 1 | 0.00112 | 0.001753 | 9 | 70.78284885 | 110.2357699 | 97.00092 | 30.94224 | 0.937595 | 0 | 0 | 0 | 0 | 15.77817 | 18.3275627 |
| FG248 | LODC | D4 | chr10:123 | 0 | 0 | 0.000181 | 1 | 3.537136695 | 0.207956466 | 0.283336 | 6.69313 | 2.092521 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG248 | LODC | D4 | chr4:1803 | 1 | 0.00119 | 0.00138 | 3 | 5000 | 38.30570661 | 35.62517 | 49.01659 | 10.93782 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG248 | LODC | D4 | chr4:1803 | 0 | 0 | 0.000513 | 1 | 4.30318859 | 3.237816678 | 2.3948 | 14.9909 | 2.013214 | 0 | 0 | 0 | 0 | 5000 | 5000 |
| FG289 | LODC | D3 | chr10:123 | 1 | 0.002 | 0.001563 | 7 | 13.04160752 | 18.1177738 | 8.146522 | 16.99152 | 0.688349 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG289 | LODC | D3 | chr10:123 | 1 | 0.00223 | 0.003527 | 18 | 178.0008384 | 261.9144785 | 261.8565 | 30.46277 | 34.27927 | 0 | 0 | 0 | 0 | 1.42313 | 1.928179014 |
| FG289 | LODC | D3 | chr4:1803 | 1 | 0.00237 | 0.002235 | 5 | 5000 | 65.42919286 | 63.0355 | 69.14879 | 3.726743 | 0 | 0 | 0 | 0 | 22.39971 | 30.65229432 |
| FG203 | LODB | D3 | chr10:123 | 1 | 0.001306 | 0.000543 | 2 | 8.79994938 | 1.969474676 | 7.062091 | 14.03044 | 0.142708 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG203 | LODB | D3 | chr10:123 | 0 | 0 | 0.000211 | 1 | 6.479508053 | 5.561458104 | 7.324739 | 3.890619 | 2.025236 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG203 | LODB | D3 | chr10:123 | 1 | 0.002 | 0.002926 | 15 | 115.8553361 | 46.34032464 | 45.37208 | 63.87342 | 4.303536 | 0 | 0 | 0 | 0 | 11.53598 | 10.45409402 |
| FG203 | LODB | D3 | chr4:1803 | 1 | 0.001499 | 0.002828 | 6 | 31.94436272 | 21.12404297 | 21.99186 | 24.64087 | 7.850906 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG203 | LODB | D3 | chr4:1806 | 0 | 0 | 0.000296 | 1 | 6.966805437 | 7.69312287 | 8.722864 | 9.596765 | 0.685578 | 5000 | 0 | 0 | 0 | 0 | 0 |
| FG203 | LODB | D3 | chr4:1806 | 1 | 0.001413 | 0.001613 | 5 | 31.78991466 | 56.22734971 | 57.50823 | 44.0701 | 16.66451 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG083 | LODA | D1 | chr10:123 | 1 | 0.01 | 0.009615 | 10 | 5000 | 89.72856068 | 89.72856 | 42.57054 | 3.118826 | 0 | 0 | 0 | 0 | 8.067255 | 4.769340009 |
| FG083 | LODA | D1 | chr10:123 | 0 | 0 | 0.00138 | 2 | 21.18504193 | 4.316079615 | 4.31608 | 8.725612 | 1.011403 | 0 | 0 | 0 | 0 | 18.65764 | 18.57134081 |
| FG083 | LODA | D1 | chr10:123 | 1 | 0.0079 | 0.012908 | 19 | 5000 | 199.5959394 | 199.5959 | 142.4248 | 19.13633 | 15.54735 | 0 | 0 | 0 | 11.09312 | 7.48636067 |
| FG083 | LODA | D1 | chr4:1803 | 0 | 0 | 0.003656 | 2 | 22.17554342 | 13.18319544 | 13.1832 | 13.65295 | 0.911465 | 0 | 0 | 0 | 16.8072 | 5000 | 5000 |
| FG083 | LODA | D1 | chr4:1806 | 1 | 0.005569 | 0.010101 | 9 | 132.7792141 | 94.01965597 | 94.01966 | 56.08611 | 1.538201 | 0 | 0 | 0 | 0 | 11.29546 | 10.98430347 |
| FG282 | LODC | D1 | chr10:123 | 1 | 0.01 | 0.007178 | 22 | 135.720247 | 159.402814 | 103.0129 | 56.54309 | 1.47278 | 0 | 0 | 0 | 0 | 7.302282 | 1.25670819 |
| FG282 | LODC | D1 | chr10:123 | 1 | 0.01115 | 0.008529 | 33 | 434.7463376 | 582.006732 | 513.4198 | 136.4945 | 12.51557 | 0 | 0 | 0 | 0 | 0 | 0 |
| FG282 | LODC | D1 | chr4:1803 | 1 | 0.01146 | 0.010604 | 20 | 5000 | 334.772338 | 257.1624 | 283.3136 | 26.68242 | 0 | 0 | 0 | 0 | 30.28471 | 45.21628351 |

# Actual Example (Recently)

- Traning data : 940 samples of total 1140 samples .
  1. Positive samples : 600
  2. Negtive samples : 340
- Test data : 200 samples of total 1140 samples .
  1. Positive samples : 138
  2. Negtive samples : 62
- Result :
  1. Acuurate ratio : 0.9798
  2. Sensitive ratio : 0.9708
  3. Specificity ratio : 1.0