A PROJECT REPORT

ON

REASONING SYSTEMS

Submitted for the completion of SIV895 Course Project

Submitted To:

Prof Prem Kalra

Course Coordinator(SIV895)

Submitted By:

Mehak

2018MCS2143

INTRODUCTION

In the AI research world, one of the major goals is to make models that can reason or infer using a series of long term dependencies from sequential data in form of answering a question or completing a task.

To measure progress towards that goal, the usefulness of a set of proxy tasks that evaluate reading comprehension via question answering is considered. These tasks measure understanding in several ways: whether a system is able to answer questions via chaining facts, simple induction, deduction and many more. These tasks are considered to be prerequisites for any system that aims to be capable of conversing with a human.

In this project, I have tried to draw a line of comparison using bABI question answering dataset between two models i.e. end to end memory network and fine-tuned BERT model. This work tries to argue over the capabilities of a machine to reason using these tasks over different sets of networks available in the literature. Also, it tries to draw a comparison between different networks using these tasks.

DATASET

The presented work uses a series of QA tasks. A given QA task consists of a set of statements, followed by a question whose answer is typically a single word (in a few tasks, answers are a set of words). The answer is available to the model at training time, but must be predicted at test time. There are different types of tasks that probe different forms of reasoning and deduction. Here are samples of three of the tasks:

Sam walks into the kitchen.	Brian is a lion.	Mary journeyed to the den.
Sam picks up an apple.	Julius is a lion.	Mary went back to the kitchen.
Sam walks into the bedroom.	Julius is white.	John journeyed to the bedroom.
Sam drops the apple.	Bernhard is green.	Mary discarded the milk.
Q: Where is the apple?	Q: What color is Brian?	Q: Where was the milk before the den?
A. Bedroom	A. White	A. Hallway

For each question, only some subset of the statements contain information needed for the answer, and the others are essentially irrelevant distractors. The model must deduce for itself at training and test time which sentences are relevant and which are not.

Formally, for one of the 20 QA tasks, we are given example problems, each having a set of i sentences $\{x_i\}$; a question sentence q and answer a. Let the jth word of sentence i be x_{ij} , represented by a one-hot vector of length V (where V is the vocabulary size = 177, reflecting the simplistic nature of the QA language). The same representation is used for the question q and answer a. The dataset used has 10000 training problems per task.

MODELS

There are different sets of network models that are available in the literature that describe the enhancement in the reasoning tasks over a period of time. For the purpose of analysis over the considered task, the following network models are considered:-

1. **END to END MEMORY NETWORKS**

   End to end memory network can be considered as a novel recurrent neural network architecture where the recurrence reads from a possibly large external memory multiple times before outputting a symbol. The architecture is a continuous form of Memory Network but it is trained end-to-end, and hence requires significantly less supervision during training, making it more generally applicable in realistic settings. It can also be seen as an extension of RNN search to the case where multiple computational steps (hops) are performed per output symbol. The flexibility of the model allows it to apply it to tasks as diverse as question answering and to language modeling.

The model takes a discrete set of inputs x_1 , ..., x_n that are to be stored in the memory, a query q, and outputs an answer a. Each of the x_i , q, and a contains symbols coming from a dictionary with V words. The model writes all x to the memory up to a fixed buffer size, and then finds a continuous representation for the x and q. The continuous representation is then processed via multiple hops to output a. This allows backpropagation of the error signal through multiple memory accesses back to the input during training.

Let us start by describing the model in the single layer case, which implements a single memory hop operation. Then we will see it can be stacked to give multiple hops in memory.

Input memory representation: Suppose for a given input set x_1 , .., x_i to be stored in memory. The entire set of $\{x_i\}$ are converted into memory vectors $\{m_i\}$ of dimension d computed by embedding each x_i in a continuous space, in the simplest case, using an embedding matrix A (of size d×V). The query q is also embedded (again, in the simplest case via another embedding matrix B with the same dimensions as A) to obtain an internal state u. In the embedding space, compute the match between u and each memory m_i by taking the inner product followed by a softmax:

$$p_i = \text{Softmax}(u^T m_i)$$

where p will be a probability vector over the inputs.

Output memory representation: Each x_i has a corresponding output vector c_i (given in the simplest case by another embedding matrix C). The response vector from the memory o is then a sum over the transformed inputs c_i, weighted by the probability vector from the input:

$$o = \sum_i p_i c_i .$$

Because the function from input to output is smooth, it can easily compute gradients and back-propagate through it.

Generating the final prediction: In the single layer case, the sum of the output vector o and the input embedding u is then passed through a final weight matrix W (of size V × d) and a softmax to produce the predicted label:

$$\hat{a} = \text{Softmax}(W\,(o + u))$$

During training, all three embedding matrices A, B and C, as well as W are jointly learned by minimizing a standard cross-entropy loss between $\hat{a}$ and the true label a.

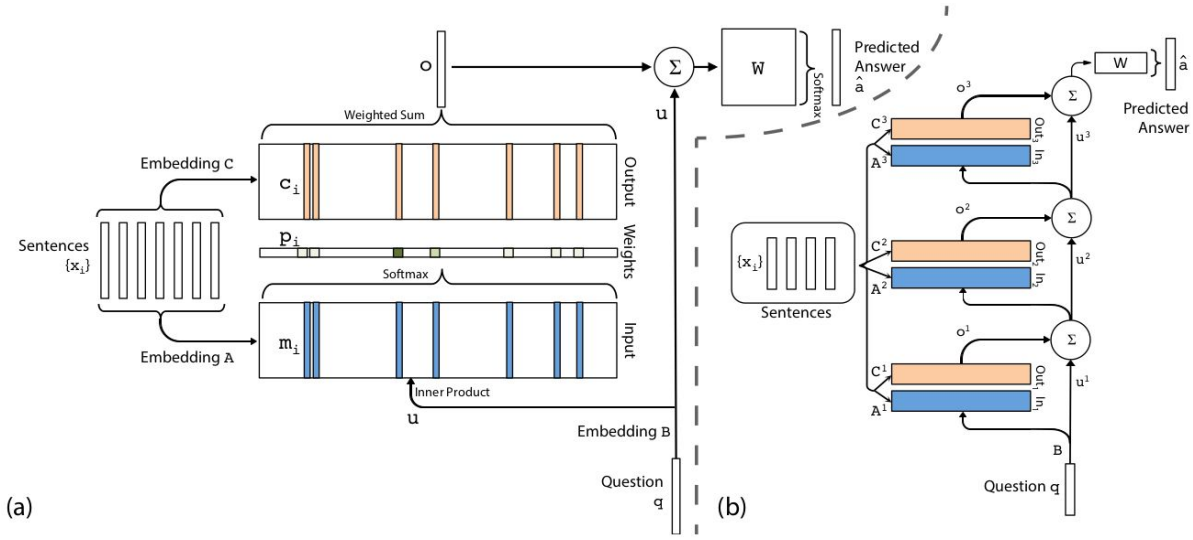


Fig: (a) Single layer version of end to end memory network model (b) Multi layer version

Multiple Layers

Now extend the model to handle K hop operations. The memory layers are stacked in the following way:

• The input to layers above the first is the sum of the output o_k and the input u_k from layer k:

$$u_{k+1} = u_k + o_k .$$

• Each layer has its own embedding matrices A_k , C_k , used to embed the inputs $\{x_i\}$. However, they are constrained to be same to ease training and reduce the number of parameters.

• At the top of the network, the input to W also combines the input and the output of the top memory layer: $\hat{a} = \text{Softmax}(W u_{k+1}) = \text{Softmax}(W (o_k + u_k))$.

Temporal Encoding:

Many of the QA tasks require some notion of temporal context, as in the first example of dataset, the model needs to understand that Sam is in the bedroom after he is in the kitchen. To enable our model to address them, we modify the memory vector so that

$$m_i = \sum A x_{ij} + T_A(i),$$

where $T_A(i)$ is the ith row of a special matrix T_A that encodes temporal information. The output embedding is augmented in the same way with a matrix T_C (e.g. $c_i = \sum C x_{ij} + T_C(i)$). Both T_A and T_C are learned during training. They are also subject to the same sharing constraints as A and C.

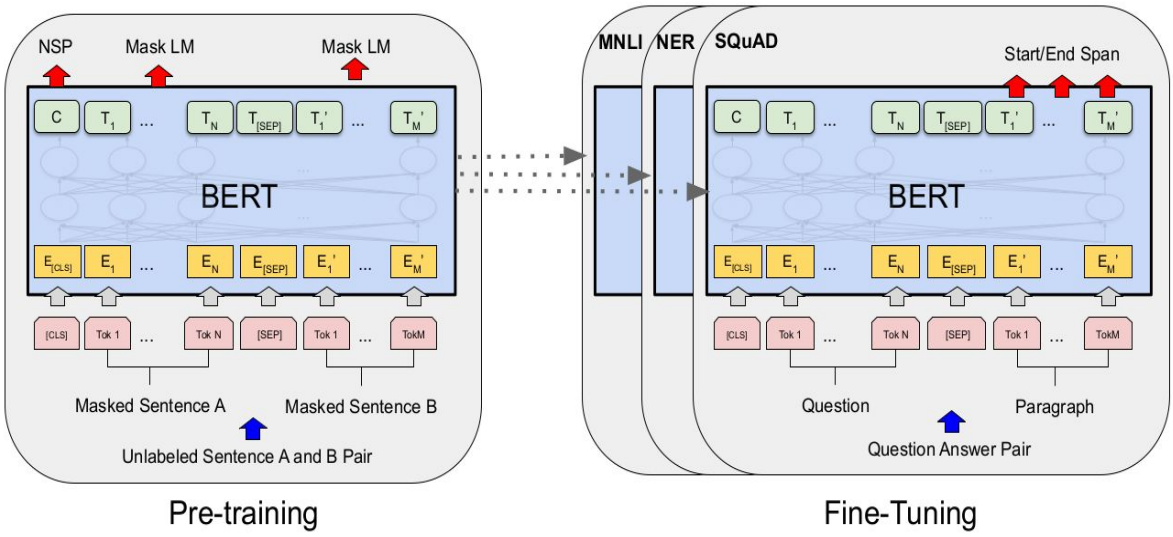
Training Details

10% of the bAbI training set was held-out to form a validation set, which was used to select the optimal model architecture and hyperparameters. The model was trained using a learning rate of $\eta = 0.01$, with anneals every 25 epochs by $\eta/2$ until 100 epochs were reached. No momentum or weight decay was used. The weights were initialized randomly from a Gaussian distribution with zero mean and $\sigma = 0.1$. Since the number of sentences and the number of words per sentence varied between problems, a null symbol was used to pad them all to a fixed size. The embedding of the null symbol was constrained to be zero.

2. BERT

BERT is a new language representation model which stands for Bidirectional Encoder Representations from Transformers. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT alleviates unidirectionality constraint by using a "masked language model" (MLM) pre-training objective. The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, which allows us to pre-train a deep bidirectional Transformer. In addition to the masked language model, it also use a "next sentence prediction" task that jointly pre-trains text-pair representations.



There are two steps in this framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters.

A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture.

Model Architecture

BERT's model architecture is a multi-layer bidirectional Transformer encoder. The Transformer follows the overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder. Here, the encoder maps an input sequence of symbol representations $(x_1,...,x_n)$ to a sequence of continuous representations $z= (z_1,...,z_n)$. Given z, the decoder then generates an output sequence $(y_1,...,y_m)$ of symbols one element at a time. At each step the model is auto-regressive (value from a time series is regressed on previous values from that same time series), consuming the previously generated symbols as additional input when generating the next.

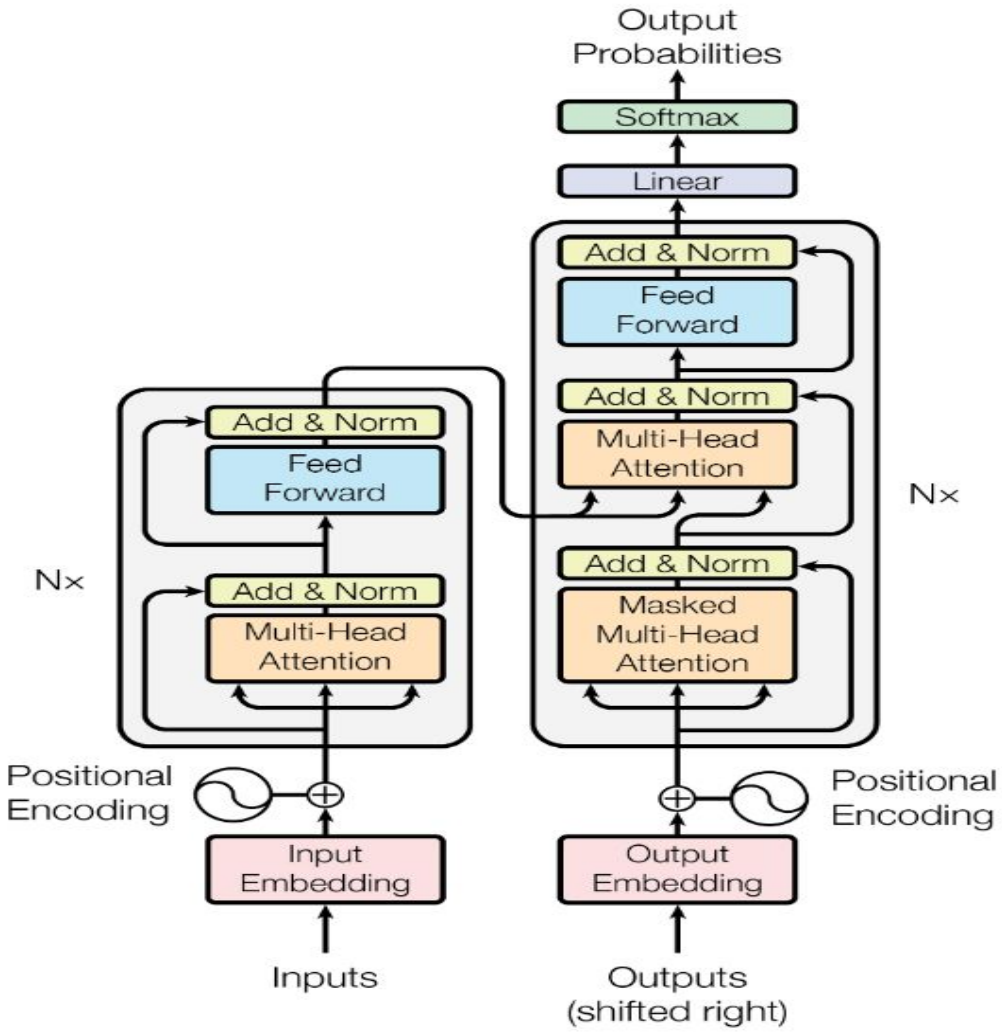


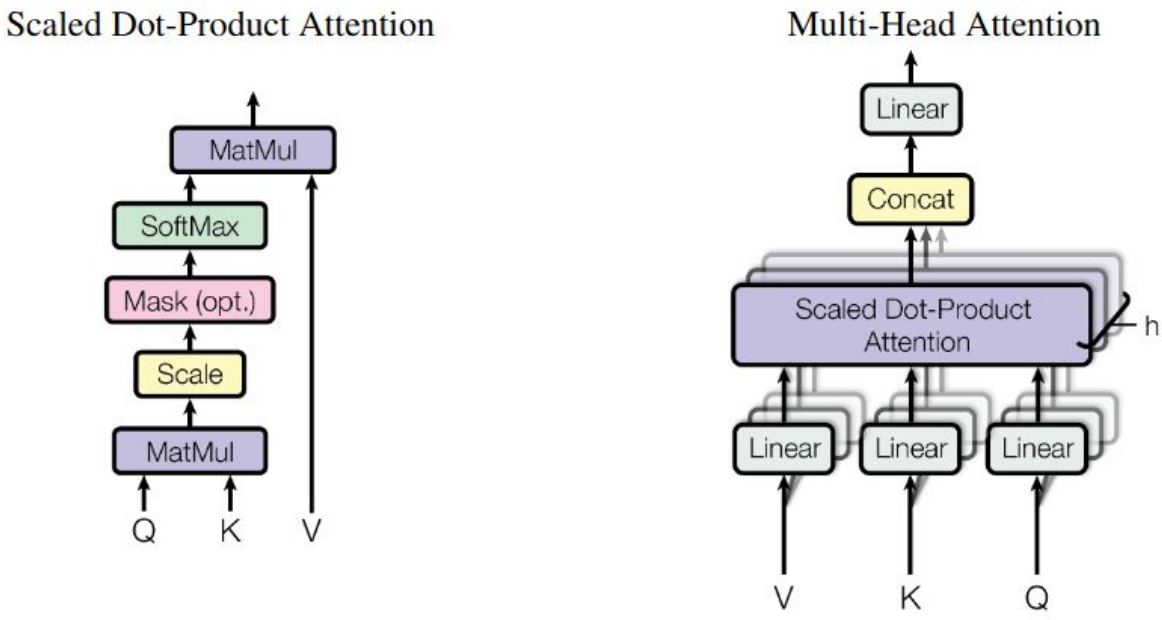
Fig: Transformer architecture

Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of N= 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. A residual connection is employed around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is LayerNorm(x+Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimensional d_{model}= 512.

Decoder: The decoder is also composed of a stack of N= 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. Also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Attention: 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.



Scaled Dot-Product Attention

The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values. In practice, 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. Then compute the matrix of outputs as:

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

The two most commonly used attention functions are additive attention, and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of 1/√dk. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k. It was suspected that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To counteract this effect, we scale the dot products by $1/\sqrt{d_k}$.

Multi-Head Attention: Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where the projections are parameter matrices.

Position wise Feed Forward Networks: In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

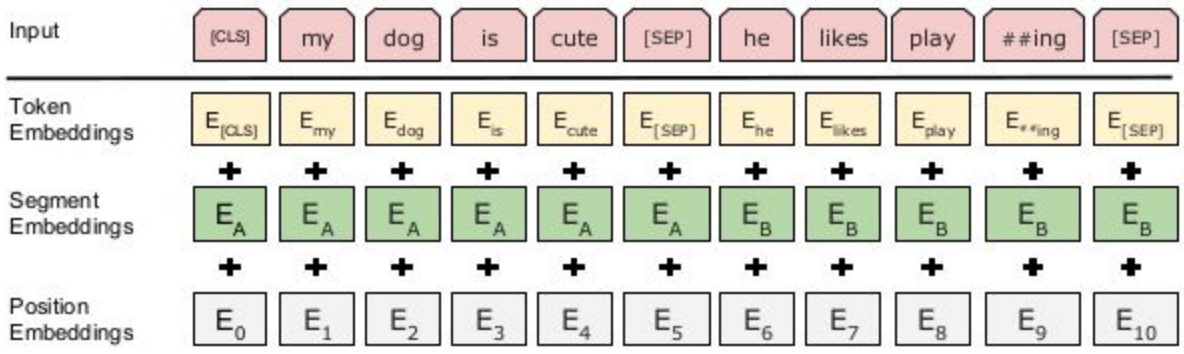
Positional Encoding: Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, “positional encodings" were added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d model as the embeddings, so that the two can be summed.

Input/Output Representations

To make BERT handle a variety of down-stream tasks, its input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., h Question, Answer i) in one token sequence. Here, a “sentence” can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A “sequence” refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together.

The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence. It differentiates the sentences in two ways. First, separate them with a special

token ([SEP]). Second, add a learned embedding to every token indicating whether it belongs to sentence A or sentence B. For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.



Pre-training BERT

BERT is pre-trained using two unsupervised tasks, described in this section.

Task #1: Masked LM

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 a 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", and the model could trivially predict the target word in a multi-layered context.

In order to train a deep bidirectional representation, simply mask some percentage of the input tokens at random, and then predict those masked tokens and this procedure is referred as a "masked LM" (MLM). 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. In all of its experiments, 15% of all WordPiece tokens in each sequence were masked at random. In contrast to denoising auto-encoders, it only predicts the masked words rather than reconstructing the entire input.

Task #2: Next Sentence Prediction (NSP)

Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, it was pre-trained for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. 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 (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext).

Fine-tuning BERT

Fine-tuning is straightforward since the self-attention mechanism in the Transformer allows BERT to model many downstream tasks whether they involve single text or text pairs by swapping out the appropriate inputs and outputs.

For applications involving text pairs, a common pattern is to independently encode text pairs before applying bidirectional cross attention. BERT instead uses the self-attention mechanism to unify these two stages, as encoding a concatenated text pair with self-attention effectively includes bidirectional cross attention between two sentences.

For each task, simply plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end. At the input, sentence A and sentence B from pre-training are analogous to (1) sentence pairs in paraphrasing, (2) hypothesis-premise pairs in entailment, (3) question-passage pairs in question answering, and (4) a degenerate text-pair in text classification or sequence tagging. At the output, the token representations are fed into an output layer for token level tasks, such as sequence tagging or question answering, and the [CLS] representation is fed into an output layer for classification, such as entailment or sentiment analysis.

RESULTS

1. Using end-to-end memory network

   I have tried training the network jointly on all the tasks. During training the network, I have tried varying number of hops and the memory dimension. The results show improve on accuracy metric on increasing number of hops *n* and memory dimension *d*. Also, on decreasing training rate, the accuracy starts to improve but training becomes too slow.

Accuracy on joint training	n=3	n=6
d = 20	0.67	0.66
d = 50	0.69	0.67
d = 100	0.65	0.71

2. Using fine-tuned BERT

   Since BERT model is trained can capture multiple contexts as it is a bidirectional model, it performed much better on bABI tasks than end-to-end memory network with an accuracy of ~0.80. Also, when tested using real-time inputs, BERT came out to be performing really well because of its ability to work on multiple contexts.

```
Enter story: Hey! I am Rhea. I love dancing. I am a resident of Singapore. I have one brother and one sister. My father is a busine
ssman. My mother is a teacher. I love my family.
Enter question: What is Rhea's hobby?
Answer is  dancing
Ask more: y/n y
Enter question: What is Rhea's mother occupation?
Answer is  teacher
Ask more: y/n y
Enter question: What do Rhea's father do?
Answer is  businessman
Ask more: y/n y
Enter question: How many siblings she has?
Answer is  one brother and one sister
Ask more: y/n n
```

REFERENCES

1. J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. arXiv preprint: 1502.05698, 2015.
2. J. Weston, S. Chopra, and A. Bordes. Memory networks. In International Conference on Learning Representations (ICLR), 2015.
3. Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus. End-To-End Memory Networks. arXiv preprint: 1503.08895, 2015.
4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers forLanguage Understanding. 1810.04805, 2019
5. Ashish Vaswani, Noam Shazeer, Niki Parmar, JakobUszkoreit, Llion Jones, Aidan N Gomez, LukaszKaiser, and Illia Polosukhin. 2017. Attention is all you need. InAdvances in Neural Information Processing Systems, pages 6000–6010.