EasyChair Preprint

№ 1504

Human-like Thinking in Machines : Vision and Language Implementations

Poondru Prithvinath Reddy

September 12, 2019

Human-like Thinking in Machines : Vision and Language Implementations

POONDRU PRITHVINATH REDDY

ABSTRACT

The human brain is a complex structure responsible for thinking skills. Several parts of the brain work together to integrate information and develop thoughts. Prefrontal Cortex, the frontal lobe where majority of thinking related processes happen in the brain. A PFC(mimicking the human Prefrontal Cortex) implementation has been proposed separately for both vision and language system. A FC-LSTM algorithm is proposed for vision system which can be used for image classification and sequence prediction tasks. However, we have removed components unsupported by neuroscience from AI architecture such as error back propagation and used Fully Connected module instead of CNN. A LSTM algorithm is proposed for language system for text generation and sequence prediction. A sequential model with a linear stack of layers consisting of last layer as a fully connected layer without softmax activation is used. The test results show that the algorithms converge and with low prediction accuracy of image classification and text generation.

INTRODUCTION

For decades, we have been trying to get the machine to think like a human. In order to make this happen, we need to understand how humans think in the first place. How do we understand the nature of human thinking? One way to do this would be to note down how we respond to things. But this quickly becomes intractable, because there are too many things to note down. Another way to do this is to conduct an experiment based on a predefined format. We develop a certain number of questions to encompass a wide variety of human topics, and then see how people respond to it.

Once we gather enough data, we can create a model to simulate the human process. This model can be used to create software that can think like humans. Of course this is easier said than done! All we care about is the output of the program given a particular input. If the program behaves in a way that matches human behavior, then we can say that humans have a similar thinking mechanism.

Within computer science, there is a field of study called **Cognitive Modeling** that deals with simulating the human thinking process. It tries to understand how humans solve problems. It takes the mental processes that go into this problem solving process and turns it into a software model. This model can then be used to simulate

human behavior. Cognitive modeling is used in a variety of AI applications such as deep learning, expert systems, Natural Language Processing, robotics, and so on.

BRAIN PARTS WITH THINKING SKILLS

The brain is a remarkable and complex structure responsible for thinking skills. Several parts of the brain work together in a sophisticated manner to integrate information and develop thoughts. The term thinking skills refers to a wide range of processes, including consciously remembering facts and information, solving concrete or abstract problems using known information, and incorporating reasoning, insight or ingenuity.

Prefrontal Cortex The frontal lobe is the largest region of the brain and it is more advanced in humans than other animals. The frontal lobe, particularly the region located furthest to the front called the prefrontal cortex, is involved in sophisticated interpersonal thinking skills, memory and the competence required for emotional well-being.

The majority of thinking-related processes happen in the frontal lobe. These include decision-making, problem-solving, and planning. The frontal lobe also helps the development of cognition, language processing, and intelligence.

The human- like thinking system often requires specific neural substrates to support the corresponding functionalities. The most important brain area related to thinking is the prefrontalcortex(PFC),where the working memory takes place, including but not confined to, the maintenance and manipulation of particular information. With the PFC, human beings can analyze and execute various tasks via ‘phonological loop’ and ‘visuospatial scratchpad’ etc. Inspired by the human-like brain organization, we build a ‘PFC’ network to language and vision streams to achieve tasks such as object detection, and language process based thinking process. Our results show that the PFC network could incrementally learn different syntaxes and detect multiple objects rapidly. Based on the PFC, we present the language and vision related continual thinking process, which shows considerable promise for the human-like machine intelligence.

METHOD

Our goal is to build a human-like neural network by removing components unsupported by neuroscience from AI architecture while introducing novel neural mechanisms and algorithms into it. Taking the convolution neural network (CNN) as an example, although

it has reached human-level performance in image recognition tasks, animal neural systems do not support such kernel scanning operation across retinal neurons, and thus the neuronal responses measured on monkeys do not match that of CNN units. Therefore, instead of CNN, we used fully connected (FC) module to build our neural network, which achieved more resemblance to animal neurophysiology in term of the network development, neuronal firing patterns, object recognition mechanism, learning and forgetting mechanisms. In addition, the error back propagation technique is generally used to modify network weights to learn representation and achieve training objectives. However, in neuroscience, it is the activity-dependent molecular events (e.g. the inflow of calcium ion and the switching of glutamate N-methyl-D-aspartate receptor etc.) that modify synaptic connections. Indeed, the real neural feedback connection provides the top-down imagery information, which is usually ignored by AI network constructions due to the concept of error back propagation. What's more, the invariance property of visual recognition under the rotation, scaling, and translation of an object is supported by coordinated population coding rather than the max-pooling mechanism. The softmax classification is usually used to compute the probability of each category (or word) in the repository(or vocabulary)before prediction. However, in reality, we never evaluate all fruit categories in mind before saying 'it is an apple', let alone the complicated computation of the normalization term in the softmax. Therefore object classification is directly output by neurons via a simple rounding operation, rather than the neuroscience unsupported softmax classification. Modern auto encoder techniques could synthesize an unseen view for the desired viewpoint. Using car as an example, during training, the auto encoder learns the 3D characteristics of a car with a pair of images from two views of the same car together with the view point of the output view. During testing, the auto encoder could predict the desired image from a single image of the car given the expected viewpoint. However, this architecture is task-specific, namely that the network can only make predictions on cars' unseen views. To include multiple tasks, we added an additional PFC layer that can receive task commands conveyed via language stream and object representation via the visual encoder pathway, and output the images and the desired text prediction associated with the images. In addition, by transmitting the output image from the decoder to the encoder and this enables the continual operation of a human-like thinking process involving both language and image.

ARCHITECTURE

The architecture contains three subsystems: (1) vision system that contains an encoder to disentangle the input or imagined scenarios into abstract population representations, (2) Language system, which consists of a binarizer to transfer symbol texts into binary

vectors, an IPS (mimicking the human IntraParietal Sulcus, implemented by an LSTM) to extract the quantity information from the input texts, and a textizer to convert binary vectors into text symbols; (3) a PFC (mimicking the human PreFrontal Cortex, implemented by an LSTM) that inputs in the form of vision, and predict sequence of text symbols and images accordingly.

VISION SYSTEM – TIME SERIES PREDICTION

Fully Connected Long Short-Term Memory Networks (FC LSTM) Architecture

The FC LSTM architecture involves using Fully Connected (FC) layers for feature extraction on input data combined with LSTMs to support sequence prediction.

FC LSTMs were developed for visual time series prediction problems and the application of generating textual descriptions from sequences of images (e.g. videos). Specifically, the problems of:

- **Activity Recognition**: Generating a textual description of an activity demonstrated in a sequence of images.
- **Image Description**: Generating a textual description of a single image.
- **Video Description**: Generating a textual description of a sequence of images.

FC LSTMs are a class of models that is both spatially and temporally deep, and has the flexibility to be applied to a variety of vision tasks involving sequential inputs and outputs. This architecture is used for the task of generating textual descriptions of images. Key is the use of a FC that is trained on a challenging image classification task that is re-purposed as a feature extractor for the caption generating problem..It is natural to use a FC as an image “encoder”, by first pre-training it for an image classification task and using the last hidden layer as an input to the RNN decoder that generates. This architecture has also been used on speech recognition and natural language processing problems where FCs are used as feature extractors for the LSTMs on audio and textual input data.

We need to repeat this operation across multiple images and allow the LSTM to build up internal state and update weights using BPTT across a sequence of the internal vector representations of input images.

The critical component of the LSTM is the memory cell and the gates (including the forget gate, but also the input gate). The contents of the memory cell are modulated by the input gates and forget gates. Assuming that both of these gates are closed, the contents of the memory cell will remain unmodified between one time-step and the next. The gating structure allows information to be retained across many time-steps, and consequently also allows gradients to flow across many time-steps. This allows the

LSTM model to overcome the vanishing gradient problem that occurs with most Recurrent Neural Network models.

LSTM networks consist of many connected LSTM cells and perform well in how efficient they are during learning.

To better understand the complex arrangement of connections between units and layers in LSTM networks, let's present some earlier concepts.

Earlier development of the concept of the feed-forward fully connected multilayer neural network, as depicted in Figure 1.

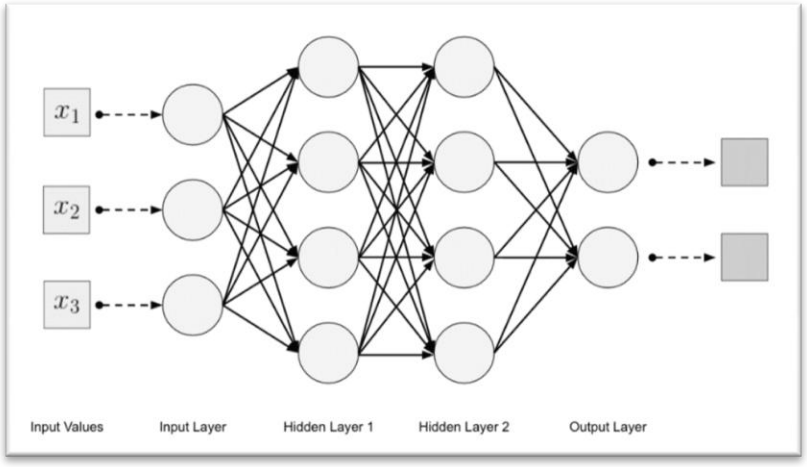


Figure 1. Feed-forward fully connected multilayer neural network architecture

In Recurrent Neural Networks, the idea of a type of connection that connects the output of a hidden-layer neuron as an input to the same hidden-layer neuron. With this recurrent connection, we can take input from the previous time-step into the neuron as part of the incoming information.

Figure 2 demonstrates that by flattening the Recurrent Neural Network , we can more easily visualize the recurrent connections in an LSTM.

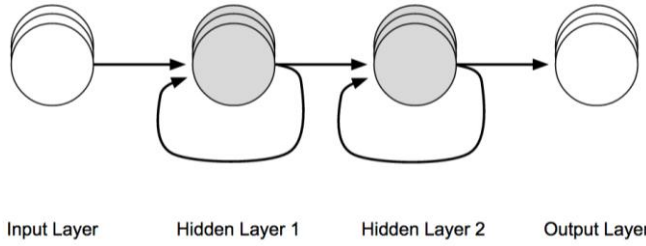


Figure 2. Showing recurrent connections on hidden-layer nodes

LSTM networks pass more information across the recurrent connection than the traditional Recurrent Neural Networks.

LSTM units

Each LSTM unit has two types of connections:

- Connections from the previous time-step (outputs of those units)
- Connections from the previous layer

The memory cell in an LSTM network is the central concept that allows the network to maintain state over time. The main body of the LSTM unit is referred to as the *LSTM block*.

Here are the components in an LSTM unit:

- Three gates
- input gate (input modulation gate)
- forget gate
- output gate
- Block input
- Memory cell
- Output activation function
- Peephole connections

The output of the LSTM block is recurrent connected back to the block input and all of the gates for the LSTM block. The input, forget, and output gates in an LSTM unit have sigmoid activation functions for [0, 1] restriction. The LSTM block input and output activation function (usually) is a tanh activation function.

The self-recurrent connection has a fixed weight of 1.0 (except when modulated) to overcome issues with vanishing gradients. This core unit enables LSTM units to discover longer-range events in sequences. These events can span up to 1,000 discrete time-steps compared to older recurrent architectures, which could model events across around only 10 time-steps.

LSTM layers

A basic layer accepts an input vector *x* (nonfixed) and gives output *y*. The output *y* is influenced by the input *x* and the history of all inputs. The layer is influenced by the history of inputs through the recurrent connections. The RNN has some internal state that is updated every time we input a vector to the layer. The state consists of a single hidden vector.

Training

LSTM networks use supervised learning to update the weights in the network. They train on one input vector at a time in a sequence of vectors. Vectors are real-valued and become sequences of activations of the input nodes. Every noninput unit computes its current activation at any given time-step. This activation value is computed as the nonlinear function of the weighted sum of the activations of all units from which it receives connections.

For each input vector in the sequence of input, the error is equal to the sum of the deviations of all target signals from corresponding activations computed by the network. The error - the BPTT, variant of back propagation used with Recurrent Neural Networks, including LSTMs.

Recurrent Neural Network training can be computationally expensive. The traditional option is to use BPTT. BPTT is fundamentally the same as standard back propagation: we apply the chain rule to work out the derivatives (gradients) based on the connection structure of the network. It's *through time* in the sense that some of those gradients / error signals will also flow backward from future time-steps to current time-steps, not just from the layer above (as occurs in standard back propagation).

However, we have not used back propagation as neurosciences do not support it and also by removing other components unsupported by neuroscience from AI architecture.

Now let's work on applying an RNN to something simple.and we are going to have us start by using an RNN to predict MNIST, since that's a simple dataset, already in sequences, and we can understand what the model wants from us.

The type of RNN cell that we're going to use is the LSTM cell. Layers will have dropout, and we'll have a dense layer at the end, before the output layer.

This should all be straightforward, where rather than Dense or Conv, we're just using LSTM as the layer type. The only new thing is return_sequences. This flag is used for when we're continuing on to another recurrent layer. If we are, then we want to return sequences. If we're not going to another recurrent-type of the layer, then we don't set this to true.

The FC LSTM model has been implemented in Python with Keras module and with Tensorflow as backend.

Since the objective is to predict sequential data, RNN is a natural choice of a model. LSTM has been proven to be a powerful model for sequential data prediction. However the traditional fully connected LSTM (FC-LSTM) cannot efficiently catch the correlation within the data.

Methods

In traditional FC-LSTM, each cell state transition takes all the input values and previous hidden states as input, which introduces a very large number of parameters.

The model is trained using ADAM Optimizer,.

Hyper Parameters

In our experiment, we choose different combinations of hidden state channel size, and the number of layers. We experiment and also train models with hidden states of 128 units and 256 units. This is to show how performance is related to hidden state size. In addition, we setup models with 2 and 3 layers of FC-LSTM networks. This is to show If a higher level abstraction can help improve the model performance.

The fully connected layer adjusts the feature matrix to 64 × 64, and each time step takes one row as the input to the LSTM network

Baseline Model

We use FC-LSTM to build our baseline model. In the baseline model, we first prepare image data in a format such that each time step takes one row as input and feed it to a 2-layer FC-LSTM. Finally, the model applies FC linear layers to the LSTM output to predict the sequence of next timestamp.

Results

FC-LSTM -256-2 means using FC-LSTM with hidden state of 256 hidden units, and 2 layers. We see that all FC-LSTM models significantly underperforms. But comparing to FC-LSTM, just by having a lot of parameters allows it to be trained much slower and achieve significantly lower performance. However, having larger hidden units and deeper than 2-layer network do not improve the performance. Further having larger hidden units and deeper network significantly increases the number of parameters, which makes the model harder to train. Therefore, the FC-LSTM -128-2 model performs the better, because it is easier to train compared to larger models, but also less expensive than the model with 256 units.

Conclusion

With full matrix multiplication used in FC-LSTM, therefore It cannot be trained faster and failed to achieve better results. The experiment shows that having an excessively large model can harm performance because it makes model harder to train and take longer time to converge.

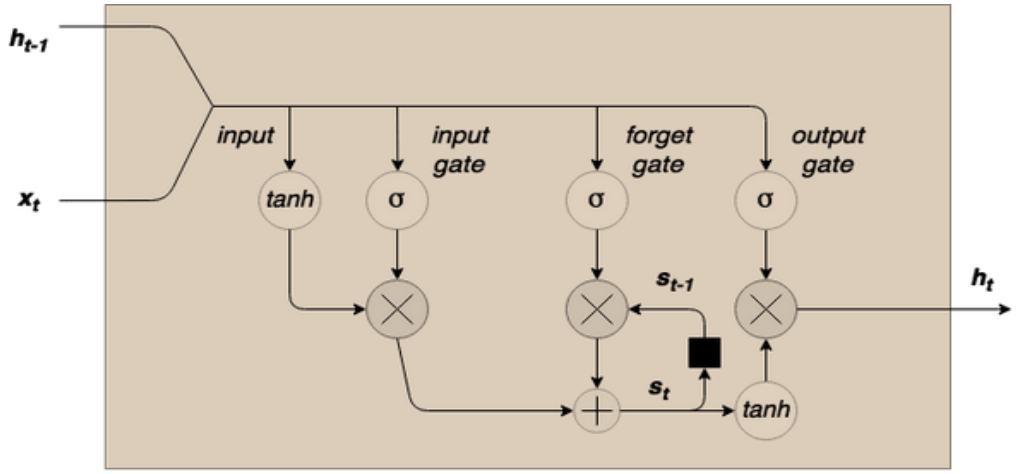
LANGUAGE SYSTEM – TEXT GENERATION

A LSTM network is a kind of recurrent neural network. A recurrent neural network is a neural network that attempts to model time or sequence dependent behaviour – such as language, stock prices, electricity demand and so on. This is performed by feeding back the output of a neural network layer at time *t* to the input of the same network layer at time *t + 1*.

Recurrent neural networks are “unrolled” programmatically during training and prediction, so we get to see that at each time step, a new word is being supplied – the output of the previous *F* (i.e. *ht−1*) is supplied to the network at each time step also.

The problem with recurrent neural networks, constructed from regular neural network nodes, is that as we try to model dependencies between words or sequence values that are separated by a significant number of other words, we experience the vanishing gradient problem (and also sometimes the exploding gradient problem). This is because small gradients or weights (values less than 1) are multiplied many times over through the multiple time steps, and the gradients shrink asymptotically to zero. This means the weights of those earlier layers won’t be changed significantly and therefore the network won’t learn long-term dependencies. Therefore LSTM networks are a way of solving this problem.

An LSTM network is a recurrent neural network that has LSTM cell blocks in place of our standard neural network layers. These cells have various components called the input gate, the forget gate and the output gate . Below is a graphical representation of the LSTM cell:



LSTM cell

On the left hand side, we have new word / sequence value *xt* being concatenated to the previous output from the cell *ht−1*

The first step for this combined input is for it to be squashed via a *tanh* layer. The second step is that this input is passed through an *input gate*. An input gate is a layer of sigmoid activated nodes whose output is multiplied by the squashed input. A sigmoid

function outputs values between 0 and 1, so the weights connecting the input to these nodes can be trained to output values close to zero to “switch off” certain input values (or, conversely, outputs close to 1 to “pass through” other values).

The next step in the flow of data through this cell is the internal state / forget gate loop. LSTM cells have an internal state variable *st* . This variable, lagged one time step i.e. *st−1* is *added* to the input data to create an effective layer of recurrence. This *addition* operation, instead of a multiplication operation, helps to reduce the risk of vanishing gradients. However, this recurrence loop is controlled by a forget gate – this works the same as the input gate, but instead helps the network learn which state variables should be “remembered” or “forgotten”.

Finally, we have an output layer *tanh* squashing function, the output of which is controlled by an *output gate.* This gate determines which values are actually allowed as an output from the cell *ht.*

We need to setup what is called an *embedding* layer, to convert each word into a meaningful word vector. We have to specify the size of the embedding layer – this is the length of the vector each word is represented by – this is usually in the region of between 50-500. In other words, if the embedding layer size is 200, each word will be represented by a 200-length vector i.e. [x1,x2,x3,…,x200].

We need to match up the size of the embedding layer output with the number of hidden layers in the LSTM cell. Each *sigmoid*, *tanh* or *hidden state* layer in the cell is actually a set of nodes, whose number is equal to the *hidden layer* size. Therefore each of the “nodes” in the LSTM cell is actually a cluster of normal neural network nodes, as in each layer of a densely connected neural network.

The input shape of the text data is ordered as : (batch size, number of time steps, hidden size). In other words, for each batch sample and each word in the number of time steps, there is a 300 length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. The input data is then fed into two “stacked” layers of LSTM cells (of 300 length hidden size), the LSTM network is unrolled over all the time steps. The output from these unrolled cells is still (batch size, number of time steps, hidden size).

This output data is then passed to a Keras layer called TimeDistributed. Finally, the output layer has a *softmax* activation applied to it. This output is compared to the training *y* data for each batch, and the error and gradient back propagation is performed from there in Keras. The training *y* data in this case is the input *x* words advanced one time step – in other words, at each time step the model is trying to predict the very next word in the sequence. However, it does this at *every* time step – hence the output layer has the same number of time steps as the input layer.

.However, we have not applied gradient back propagation as neurosciences do not support it and by removing other components unsupported by neuroscience from AI architecture.

Text generation using LSTMs

So far we have had concepts about functioning of LSTMs. Now we would build a model that can predict some *n* number of characters after the original text of Macbeth. We first have to download the Macbeth dataset which will be used as the training and test data and need to incorporate the *data_path* variable in the code to match the location of this downloaded data.

This has been implemented using Python, TensorFlow as a backend and Keras library.

We will use the library **Keras,** which is a high-level API for neural networks and works on top of TensorFlow. We have to make sure that **Keras** is installed and functional.

We import all the required dependencies before loading text file..

Next is loading text file and creating character to integer mappings. In order to get the text data into the right shape for input into the Keras LSTM model, then each unique word is identified and assigned a unique integer. Finally, the original text file is converted into a list of these unique integers, where each word is substituted with its new integer identifier. This allows the text data to be consumed in the neural network.

Data is prepared in a format such that if we want the LSTM to predict the expected output *'given text* feed as the input. Similarly, we fixed the length of the sequence that we want (set to 50) and then save the encodings.

A LSTM network expects the input to be in the form [samples, time steps, features] where samples is the number of data points we have, time steps is the number of time-dependent steps that are there in a single data point, features refers to the number of variables we have for the corresponding true value in Y.

- Defining the LSTM model

A sequential model which is a linear stack of layers is used. The first layer is an LSTM layer with 300 memory units and it returns sequences. This is done to ensure that the next LSTM layer receives sequences and not just randomly scattered data. A dropout layer is applied after each LSTM layer to avoid overfitting of the model. Finally, we have the last layer as a fully connected layer without a '*softmax'* activation and neurons equal to the number of unique characters, because we need to output one hot encoded result.

- Fitting the model and generating characters

The model is fit over 50 epochs, with a batch size of 30. We then fix a random seed (for easy reproducibility) and start generating characters. The prediction from the model gives out the character encoding of the predicted character, it is then decoded back to the character value and appended to the pattern and this is how the output of the network is generated.

- Results

After 50 epochs, training data set accuracy was not good, while test data set accuracy reached was approximately 10-15%.

Eventually, after enough training epochs and optinization, it may give better and better results over the time and this is how we would use LSTM to achieve a sequence prediction task.

CONCLUSION

LSTMs are a very promising solution to sequence and time series related problems. Prefrontal Cortex(PFC) , the frontal lobe is the largest region of the brain and is involved in thinking skills and memory, has been implemented separately for both vision and language system using LSTMs. However, the one disadvantage about them, is the difficulty in training them. A lot of time and system resources go into training even a simple model. The test results are not so encouraging with low prediction accuracy of image prediction and text generation as we have not applied gradient back propagation and by removing other components unsupported by neuroscience from AI architecture.

REFERENCES

Feng Qi, Wenchuan Wu. Human-like machine thinking: language guided imagination. arXiv:1905.07562 [cs.CL]. arxiv.org/abs/1905.07562