Developing Deep Learning Models to Simulate Human Declarative Episodic Memory Storage

Abu Kamruzzaman[1], Charles C. Tappert[2]
Seidenberg School of Computer Science and Information Systems
Pace University
Pleasantville, NY

***Abstract*—Human like visual and auditory sensory devices became very popular in recent years through the work of deep learning models that incorporate aspects of brain processing such as edge and line detectors found in the visual cortex. However, very little work has been done on the human memory, and thus our aim is to model human long-term declarative episodic memory storage using deep learning methods. An innovative way of deep neural network was created on supervised feature learning dataset such as MNIST to achieve high accuracy as well as storing the models hidden layers for future extraction. Convolutional Neural Network (CNN) learning models with transfer learning models were trained to imitate the long-term declarative episodic memory storage of human. A Recurrent Neural Network (RNN) in the form of Long Short Term Memory (LSTM) model was assembled in layers and then trained and evaluated. A Variational Autoencoder was also used for training and evaluation to mimic the human memory model. Frameworks were constructed using TensorFlow for training and testing the deep learning models.**

Keywords—Convolutional neural network; long short term memory; Variational Autoencoder; deep learning; memory model; machine learning

I. Introduction

The aim of this research is to construct a deep learning model to simulate the human brain long-term declarative episodic memory storage, focusing primarily on the computer science perspective of the Rosenblatt Model for experiential storage in neural networks [1]. It is not known completely how human memory remembers past events. Previous work showed that Convolutional Neural Networks (CNN) models work well for classification of spatial data while CNN was unable to store the hidden layers for future predictions [2]. Our new hypothesis is that an integrated framework of CNN, Long Short Term Memory (LSTM) and Variational Autoencoder (VAE) adequately stores images for future recall.

Deep learning models can produce highly accurate results while trained and tested on datasets. However, the dataset might not generate accurate results while used inaccurately and larger dataset increase the amount of inconsistency of generating errors [3]. This issue can be resolved through additional training on the larger dataset.

The MNIST (Modified National Institute of Standards and Technology) is a well-known database of handwritten characters for image processing comprised of 60,000 training set examples and 10,000 test set examples [4]. MNIST is a subset of NIST which have been size-normalized and have been aligned in the center [4]. The current test error rate for MNIST is very low reported to be 0.23% using CNN [5].

This research focuses on simulated deep learning memory models using simple CNN and pre trained CNN transfer learning VGG16, ResNet, Inception, MobileNet, LSTM and VAE to mimic the human brain's long-term declarative episodic memory of human mind. The research experiment and result show that the deep learning models built using TensorFlow API (Application Programing Interface) works well store the model for future usage. Our experiments in this journal uses CNN, LSTM, VAE conducted on MNIST handwritten dataset images with TensorFlow frameworks to simulate the human brain's long-term declarative episodic memory of human mind.

II. Literature Review

Deep learning is a subsection of machine learning where models are graph structures with multiple layers and typically non-linear. Both supervised and unsupervised methods are used for fitting models to data. Deep learning is used for prediction and generation and its application domains are image, audio and texts. Our literature review focuses on proving the similarities with human memory and deep learning model while also explaining on the deep learning algorithms such as CNN, LSTM, VAE which is the primary focus for this research to be used for storage mechanism to mimic human memory model.

Both human memory and deep learning models are mostly comprised of neurons. Hebbs states that the basics of human learning are that when a neuron accepts input from another neuron and if both neurons are highly active, the weight for both of the neurons should be strengthened [6].

A. Deep Learning and Memory Model Similarities

Human brain and deep learning core functionality is memory or storage [7]. Deep learning neural network contains input, weight parameters and works with calculated dataset and memory in brain acts similar way. Deep learning stores in dynamic RAM (DRAM), static RAM (SRAM) internally and externally which is the functionality of classical computers to save new data where as human brain dynamically and nomadically the patterns of neurons and synapses accomplish the behavior of neural networks storage mechanism [8].

Brain stores the input dataset of pattern recognizers in the hippocampus and learns from the frequency of the high-level features from cortical neurons and in similar fashion neural network store the complete dataset in the computer memory for frequent access to the dataset for learning the data behavior [9].

B. Long-Term Declarative Episodic Memory

Atkinson-Shiffrin memory model divided primarily into three categories named as sensory, short-term and long-term which are very popular for understanding memory as shown in Fig. 1. This research primarily studies the long-term declarative episodic (experiential) memory. This study is the focus of storing experiences and events that took place in different times in memory in a serial form and human can recreate these events and experiences that happened in a person lifetime that might have been forgotten for the time being. The permanent storage of the long-term declarative episodic (experiential) memory is infinite and limitless. The invention of Miller [11] discusses on the short-term memory that can hold only 5-9 chunks of information (seven plus or minus two) and a chunk is somewhat meaningful unit. The meaning of chunk is digits, words, chess positions, or people's faces. All the following theories of memory after Miller's chunk invention followed the concept of chunking and the limited capacity of short-term memory as a basic. The long-term memory comes from short-term memory once the memory saved permanently.

C. Convolutional Neural Network

Convolutional neural network known as CNN is very popular in deep neural networks aka deep learning for image processing and analysis. CNN apply multilayer perceptron with input, output, single or multiple hidden layers and does not require preloading of the images [12]. The CNN interprets images into pixels and features to classify the objects in the images during the training of the model. The images output classification allocated a probability from the numeric translation and learnt data as the training of the model completes.

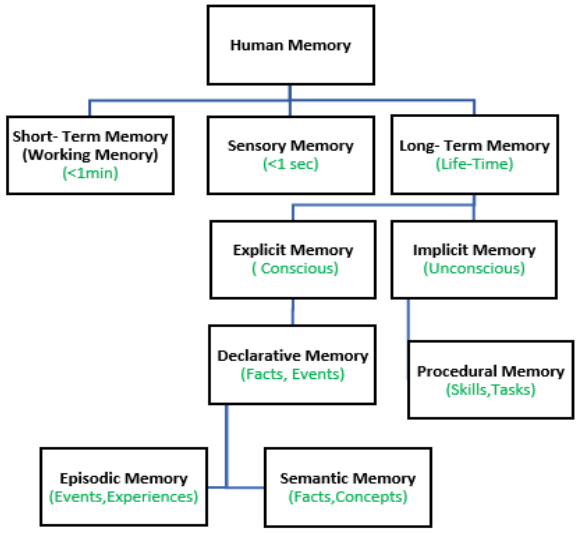

Human Memory
Short- Term Memory (Working Menory) (<1min)
Sensory Memory (<1 sec)
Long- Term Memory (Life-Time)
Explicit Memory (Conscious)
Implicit Memory (Unconscious)
Declarative Memory (Facts, Events)
Procedural Memory (Skills,Tasks)
Episodic Memory (Events,Experiences)
Semantic Memory (Facts,Concepts)

Fig. 1. Types of Human Memory (Adapted) [10].

CNN model training consists of several steps. Primarily the model takes an input from the image for sample analysis. Convolution is an evaluation of the sample area of pixels known as 'features' with the other parts of the image. The model uses a simple mathematical formula to select a match of these features. The network applies multiplication of each pixel to match the feature with is the source area. This method applied throughout the image to match every pixel. The model will apply these matches everywhere possible to attempt the highest accuracy of the image. The other subsamples are recognized and this technique repeated across the complete image. Pooling known to shrinking the large area of an image for calculation also applied. CNN also applies "Rectified Linear Units" which is known as ReLU where model swaps out negative calculations from convolution for a zero. ReLU helps identify the valuable units of the images and keeps the accuracy into stable position.

Fig. 2 below an input image (e.g. dog.jpg) sent to convolution layer for the CNN model to train. The CNN will train the model using the neural network hidden layers, acquire the features of the image, and extract the labels from the pre-trained weights to identify the output label of the image. We used VGG16, ResNet50, MobileNets and InceptionV3 pre-trained CNN transfer learning models with TensorFlow framework used in this research. However, these experiments are not presented in this journal.

D. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) introduced in 1997 by Hochreiter & Schmidhuber is a branch derived from Recurrent Neural Network (RNN) [14]. LSTM remembers previously stored information into memory as needed. It has mechanism to forget and utilize the newly stored information or mix the newly stored input with the old stored memory information.

Fig. 3 below shows the architecture of RNN with three gates (input, forget and output) for LSTM. The input gate collects the new information and transfers to output gate with the current time stamp whereas forget gate deletes the information that is not required anymore.

The RNN gates act on incoming signals as to pass or block the data utilizing its strength and import that is similar to the neural network's nodes. The filtering of RNN works with weights as well. The weights used through iterative process of guesses, backpropagation error. The input and output states monitored through weights using recurrent network learning mechanism. The weights adjusted through gradient descent.

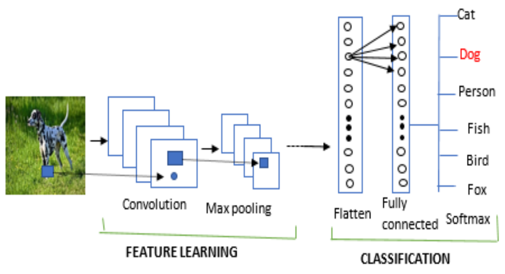


Fig. 2. A General Depiction of the Convolution Process (Adapted) [13].

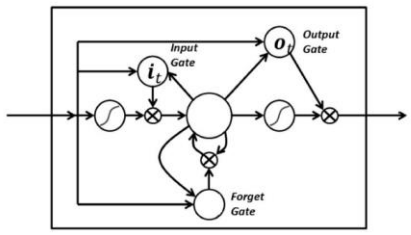


Fig. 3. RNN with Input, Output and Forget Gates [15].

Fig. 4 below shows the LSTM with inputs, outputs normalization and vector operations components in detail. The network takes three inputs. The LSTM network takes three inputs. The input is the X_t keeps track of current time step. h_t is the output and C_t is the memory of the current LSTM network. C_t-1 is known to be the "memory" of the previous unit plays a very important role. h_t-1 is the output of an LSTM.

Our research focus will be mostly on building an LSTM model to preserve episodic memory like the human brain. LSTMs can preserve the errors through time and layers using backpropagation. A supplementary constant error maintained through the recurrent nets learning using time steps that opens a channel to link sources and outcome remotely. This study looks through the LSTM deep learning model to imitate the episodic memory of human brain.

E. Variational Autoencoder (VAE)

Autoencoder is a function used in model to process the input data with restrictive sensitive manner. Variational Autoencoder (VAE) is a form of Autoencoder divided into two parts known as encoder and decoder. Encoder collects the input data and adds the most important data features to a vector form with a lower dimension than the original input. Decoder reconstructs the features vector to represent the output. Below Fig. 5 illustrates the VAE. VAE can take a principled Bayesian approach toward building systems. It's mostly used for semi-supervised machine learning. VAEs have one fundamentally unique property very useful for generative modeling different from vanilla autoencoders. VAEs contain latent spaces provide random sampling and interpolation with continuity by design.

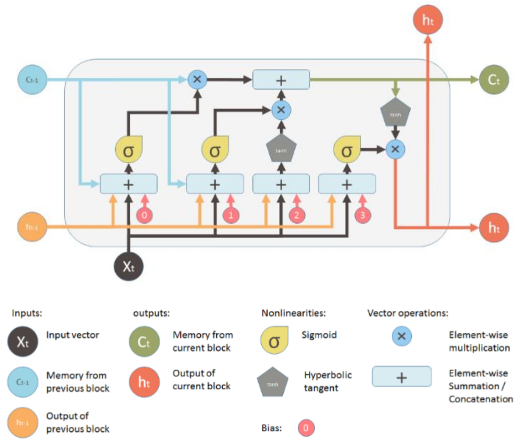


Fig. 4. Illustration of a Single LSTM Building Block [16].

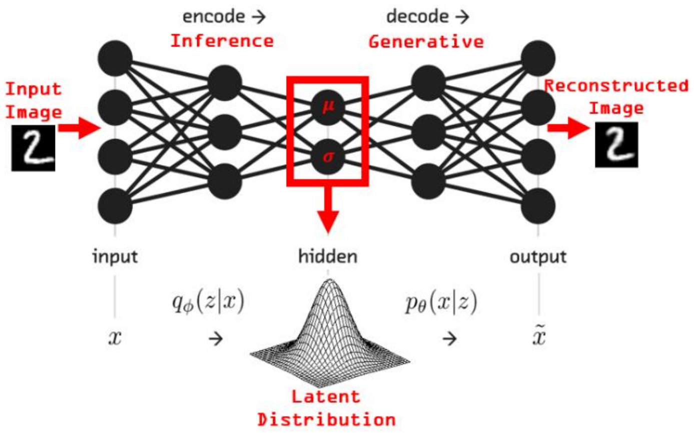


Fig. 5. Variational Autoencoder [17].

III. Methodology

The focus of this research is to learn and study the deep learning models with a c-system added on incorporated from the Rosenblatt Brain model [1] with storage mechanism. The deep neural network models are built from scratch using TensorFlow estimator framework. Experiments conducted on MNIST dataset to see how the representations work and stored for future predictions. In these experiments and results, CNN image classification and recognition tasks generated excellent output. In addition, CNN learning and storing of the model for future predictions were successful in these deep learning models.

Finally, we extended our MNIST experiments on LSTM and on VAE, using TensorFlow save and restore framework. In these experiments and results, MNIST image classification and recognition tasks generated excellent output and we were able to store the model for future predictions as well.

Our experiments and results focuses on the following:

A. Framework used to Store the Models for Future Extraction

The initial experiments were unable to store the dataset for future prediction using plain vanilla CNN and transfer learning CNN experiments. Therefore, our new deep learning experiments behave as long-term declarative episodic memory models using a framework with CNN, LSTM and VAE. The new experiments and results prove that the added framework combined with simple CNN model or pre-trained CNN models using input dataset MNIST handwritten dataset were able to classify the dataset and restore the output at later time.

Fig. 6 below depicts our architecture of the deep learning model in the visualization form that described below.

The newly proposed models collects input images from S-system, hidden layers shown in A-System and output layers are for R-System a normal display of a neural network classification model. The extended C-System works as the memory unit to store the output of the classification model for future retrieval. The C-system will maintain connections with both A-system (hidden layer) and R-system (output layer). We use different mechanisms to build our proposed C-system that explained later section in detail.

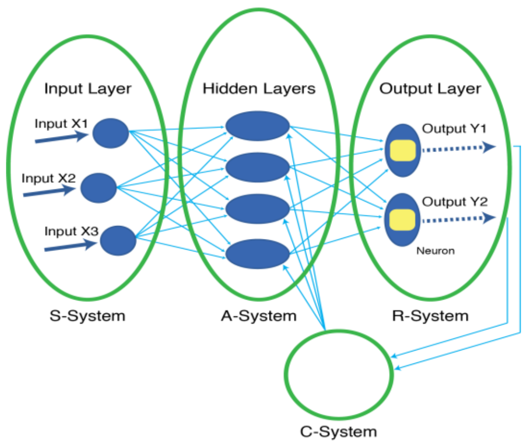


Fig. 6. Proposed Deep Learning Model Storage Architecture.

Our initial experiments were not able to store as complete memory model similar to human memory model while experiments conducted using CNN with MNIST images, LSTM with computer-generated numbers, ImageNet datasets applied on transfer learning CNN models such as ResNet, VGG16, InceptionV3 and MobileNet [2].

B. Proposed C-System

The proposed C-system is built using the TensorFlow Premade and Custom Estimators API and Save and Restore API frameworks. We also used the pre-trained ImageNet models transfer learning mechanism to test the C-system storage mechanism.

Google Brain team developed the TensorFlow framework which is an open source machine-learning framework [18]. TensorFlow bundles together multiple deep learning and machine learning models and algorithms to make the models useful. TensorFlow and open source platform helps to write lazy evaluation, imperative programs, graphs, sessions, variables, debug, etc. [1]. This framework is build using C++ works on Python. Tensorflow can train and run various deep learning models such as word embedding, image recognition, handwritten digit classification, recurrent neural networks, natural language processing, and sequence-to-sequence models for machine translation.

Our goal using the TensorFlow API to enable the C-System storage and retrieval feature as required. The Figure 7 below shows the hierarchy of TensorFlow API which is mostly divided into three levels. The top level of this hierarchy encapsulates the framework into a deep learning model. The mid-level APIs are a set of reusable packages to create computational graphs. The low level API give access to the runtime. In this level, tf.Session provides the flexibility to fine tune the models as needed. We customized the estimator of the High-level API to build our proposed C-system for majority of our experiments. We also use save and restore low-level API with tf.Session in our LSTM and VAE MNIST experiments to enable the C-system features. Below are the descriptions of high-level API Estimator and the low-level API Save and Restore that we used to build the C-system.

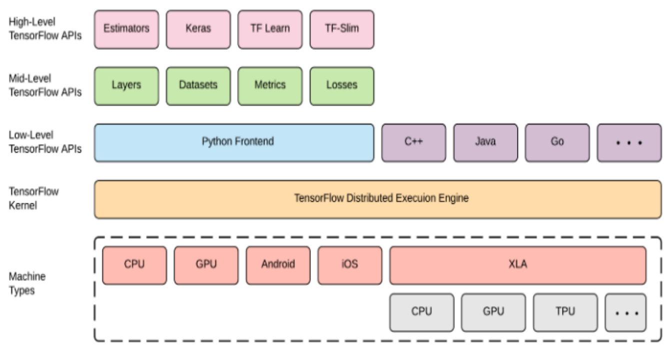


Fig. 7. Tensor Flow API Hierarchy [19].

C. Estimator API Framework

An Estimator API framework works well to specify, train, evaluate and deploy machine learning models and can be used with distributed platform utilizing the TensorFlow distributed training support [4]. This framework saves the complete deep learning neural network model if configured correctly. Google internally benefited introducing the TensorFlow Estimators where multiclass classification models perform 37% better accuracy and reduced required lines of code from 800 to 200 [4]. Estimators can be on the details of initialization, model save and restore, model logging, and other various features. The Estimator API used for training a model, estimating model accuracy, and generating predictions.

Cheng, Heng-Tze, et al. [4] mentions that an internal survey has shown that the Google codebase checked in with 1,000 Estimators and it is recorded that more than 120,000 experiments conducted within one year since Estimators framework is introduced and the prediction is that the true number of experiments are much higher. Fig. 8 below shows in percent usage of multiple Estimators at Google. Our MNIST CNN memory model experiment built showing both pre-made and custom Estimators. We used pre-made DNNClassifier in this experiment. The other pre-trained MNIST CNN memory models experiments using custom Estimators.

TensorFlow has a collection of tf.estimator to implement deep learning algorithms and the Estimator API comes from tf.estimator.Estimator. Estimator API has functions train(), evaluate(), or predict(). Fig. 9 shows how the Estimator is build. It automatically writes the checkpoints and the event files to the disk.

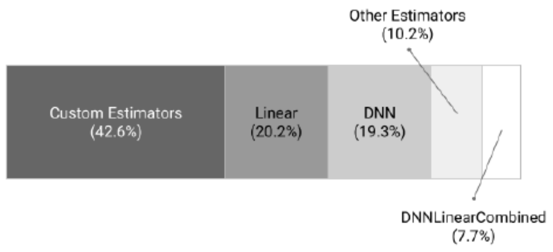


Fig. 8. Estimators Framework usage at Google [20].

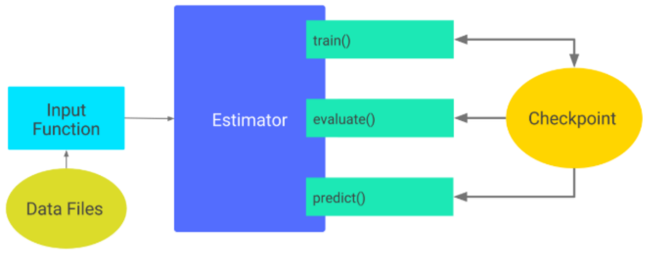


Fig. 9. Estimator Restore the Model [18].

Checkpoints created at training time are the versions of the model. Events files used for visualization on TensorBoard. The Estimator saves the model every 10 minutes (600 seconds) by default until model completely trained (if no steps are defined) in a custom directory defined by the developer through model_dir function call. We used /tmp/mnist_model as storing location for one of our experiment. The ls command in UNIX shows the objects in that directory. The Table 1 below $ ls -1 /tmp/mnist_model are the objects and descriptions are shown as comment as displayed in model_dir which is our proposed C-system. The directory retains five most recent checkpoints.

TABLE I. PROPOSED C-SYSTEM FILES DESCRIPTION

Object Name	**Comments**
checkpoint	model parameters will be reloaded from the checkpoint
events.out.tfevents. timestamp.hostname	TensorFlow events files with summary data; uses to create visualizations
graph.pbtxt	File saves the complete graph (meta + data). To load and use.
model.ckpt-1.data-00000-of-00001	stores the values of each variable
model.ckpt-1.index	identifies the checkpoint; store index of variables
model.ckpt-1.meta	Meta graph stores the graph structure

Our experiments used the default values and did not use the tf.estimator.RunConfig function. Estimator restore the model and saving to a specified directory. There exists two kinds of Estimators as shown in Fig. 10: Pre-made Estimators and custom Estimators. Pre-made Estimators and custom estimators displayed at a later discussion. The pre-made Estimators are plain vanilla models with default setups to build regular machine learning/ deep learning models such as Random Forests Classification/Regression and Linear Classification /Regression, and Deep learning models for classification and regression. Google YouTube Watch Next video recommender system (a user can choose a list of videos from a ranked list after watching the current video) uses a deep model with TensorFlow Estimators (DNNClassifier) framework where it takes multiple days to train a model and model training data are continuously updated [4]. The pre-made Estimators perform the tasks below:

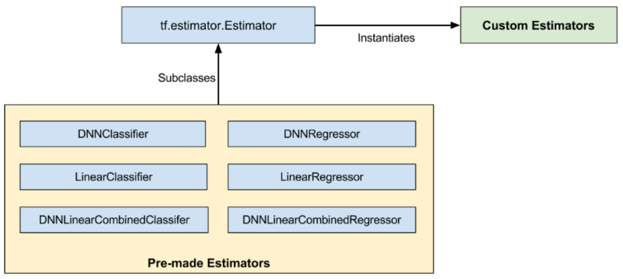


Fig. 10. Estimators API [18].

- Single or multiple input functions created.
- Feature columns for the model defined.
- Estimator defined with the feature columns and various hyperparameters.
- Estimator objects call single or multiple methods and with required input function for the source dataset.

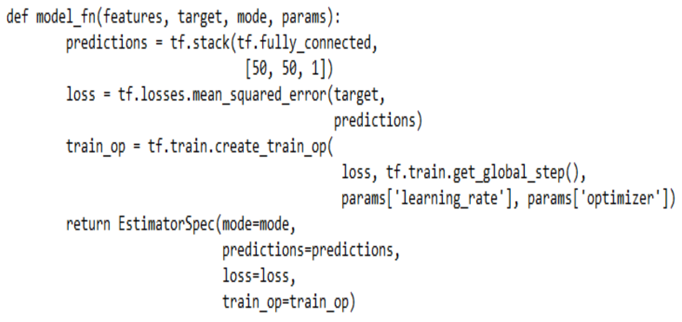
Estimator requires customization built using custom estimators.

D. Custom Estimator API Framework

Custom Estimator API is a lower level method utilizes as a custom black-box model to reuse easily. We customize this API to build our C- system memory unit to store our deep learning models. The deep learning model is stored in a method called the model_fn(). A model_fn on a deep learning model illustrated in Fig. 11 below.

The function of the model has the code to outline the process of training the model. It may include identify the labels, loss function, model prediction, evaluation and training the model as well. Both pre-made and custom Estimator class contains three major methods, which are, the train(), evaluate(), predict().

- To train a dataset in the deep learning model train() method is called and this method is used for iteration through a set of training operations.
- To evaluate a dataset performance by iterating through a set of evaluation operations evaluate() method is called.
- To make predictions predict() method called on a trained model.



```
def model_fn(features, target, mode, params):
        predictions = tf.stack(tf.fully_connected,
                               [50, 50, 1])
        loss = tf.losses.mean_squared_error(target,
                                            predictions)
        train_op = tf.train.create_train_op(
                                             loss, tf.train.get_global_step(),
                                             params['learning_rate'], params['optimizer'])
        return EstimatorSpec(mode=mode,
                             predictions=predictions,
                             loss=loss,
                             train_op=train_op)
```

Fig. 11. Tensor Flow Framework Model_fn Pseudo Code.

Both pre-made and the custom Estimators require writing a method to input the dataset into the pipeline. Both training and evaluation of dataset require this method named model_dir. As model_dir method called during the estimator training, a checkpoint is stored using TensorFlow. This store in a folder in the hard disk storage initialized in model_dir. Every following call to model_dir during training, evaluation, or prediction the following happens:

- A model graph is being builds through estimator by running the model_fn().
- The most recent checkpoint stores the weights of the new model initialized by the estimator.

It can also be said that using checkpoints TensorFlow rebuilds the model as the following function call evaluate(), train(),or predict(). Each model training should be built in separate directory to avoid the bad restoration of the model [21].

The model and checkpoint require being compatible for a model to be restore using checkpoint. For example, if a model trained on DNNClassifier as shown in Fig. 12 below Estimator using two hidden layers where each hidden layer have 10 hidden nodes:

Once the training is completed and checkpoints are created in model_dir and the hidden layer parameters are modified from 10 to 20 to retrain the model will fail because of the state of the checkpoint is incompatible with the new model. It will fail with the following error as shown in Fig. 13.

Different versions of a model should run from separate model_dir. This isolation helps the recovery of the checkpoints.

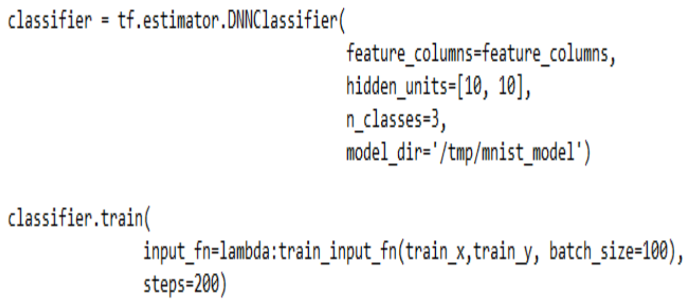
Estimator’s checkpoints can easily save and restore models. Here developer can define the function parameter steps to train the model partially.

E. Save and Restore API Framework

Save and Restore API is a low-level TensorFlow method for saving and restoring deep learning models. Export and import of models using SavedModel is not language dependent, easily recovered, and works on serialization format.

- The graph variables saved and restored using the saver variable through the tf.train.Saver() object.
- To save the variables in a session, session instance run and stored in a directory passed through save_path method. model.ckpt is a prefix added to the checkpoint filename by system while storing the checkpoint files in model directory.
- saver.restore is called to restore the Graphs variables, build the graph and run the session instance.

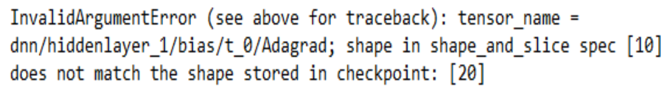
Save and Restore low-level API with tf.Session is used in our LSTM and VAE MNIST experiments to enable the C-system storage and retrieval features.



```
classifier = tf.estimator.DNNClassifier(
                                        feature_columns=feature_columns,
                                        hidden_units=[10, 10],
                                        n_classes=3,
                                        model_dir='/tmp/mnist_model')

classifier.train(
              input_fn=lambda:train_input_fn(train_x,train_y, batch_size=100),
              steps=200)
```

Fig. 12. Model Code with DNN Classifier Estimator.



```
InvalidArgumentError (see above for traceback): tensor_name =
dnn/hiddenlayer_1/bias/t_0/Adagrad; shape in shape_and_slice spec [10]
does not match the shape stored in checkpoint: [20]
```

Fig. 13. Tensor Flow Error Code.

IV. Project Requirements

To experiment all the proposed deep learning models, some programs and libraries installation required. It requires Python 3.5, Keras and TensorFlow 1.10 and numpy and matplotlib need to be installed. TensorBoard 1.10 required for graph visualization. Furthermore, a background knowledge of CNN, LSTM, pre-trained transfer learning, VAE, TensorFlow API and knowledge of Rosenblatt experiential storage model is required for the comparison of the architectures. We mostly used a local laptop environment to conduct all the experiments. Google Colab a free tool could be used for small experiments and Google cloud ML engine with VM instance and CUDA GPU can be utilized to achieve better performance for these experiments as well.

V. Experiments and Results

Deep learning models such as CNN, LSTM or pre-trained CNN models without any framework unable to retrieved the complete c-system with storage mechanism [2]. Pre-trained CNN models transfer the weights of previously trained models but unable to replicate the proposed c-system. We needed a mechanism where we could store the complete model into c-system for future use. The new proposed TensorFlow framework added with CNN, pre-trained CNN (ResNet, VGG16, MobileNet, InceptionV3), LSTM or VAE provided the solution of storing the complete model. It also helps us visualize the model using TensorBoard.

Our new experiment models using TensorFlow Framework API on MNIST datasets elaborated in this journal include

- CNN memory model with Premade DNNClassifier Estimator
- CNN memory model with Custom Estimators
- LSTM MNIST Model with Save and Restore Framework API
- VAE Memory Model with Save and Restore Framework API

Often we had to train with a small dataset instead of the complete dataset. As deep learning models consumes enormous powerful resources and it takes significant amount of time running the complete dataset that makes the process very slow. For example, it would take us about 70 days to train ResNet model for 60,000 MNIST dataset in an ordinary machine. We ran for 2000 dataset for our experiment that took us more than 3 days.

MNIST CNN memory model contains two experiments. One experiment conducted using pre-made estimator while the other experiment was conducted using custom estimator. Both experiments were trained successfully for 2000 MNIST datasets as shown in Fig. 14 below. The evaluations conducted by restoring the trained model in both cases. In addition, the predictions were done from restoring the model in both cases to see if the correct images are predicted.

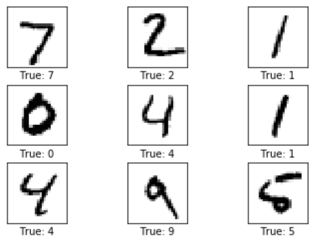


Fig. 14. MNIST Input Images and Labels used.

A. *Experiment#1 CNN Memory Model with TensorFlow DNN Classifier Estimator*

The purpose of building this MNIST CNN memory model with learning and training on MNIST hand written dataset to identify for image classification and recognition. The motivation of this experiment is to identify the sample images and store the model for future use. This experiment is to train a CNN deep learning model from scratch with learning and training on MNIST dataset and extend the model to build the proposed C-system for storing images or model for future prediction. We enhanced the model with tf.estimator.DNN Classifier using a 3-layer hidden units with 512, 256 and 128 units respectively for pre-made estimator. The model_fn()is built for custom estimator. Both experiments utilized the 70,000 MNIST dataset using training-set: 55000, validation-set: 5000 and test-set: 10000. Below is the snapshot of the input dataset with label before training. We trained the model up to 2000 datasets as our research interest is to build and test the storage mechanism for C-system. Therefore, it's not required to train the model for entire 55000 datasets. The C-system storing location was defined as model_dir = "./checkpoints_CNN_DNN" on pre-made DNNClassifier Estimator and model_dir= "./checkpoints_CNN_Custom/" for custom Estimator C-system storing location.

Initially the code calls the required imports and loads the MNIST data. Here is the high-level description for both pre-made and custom Estimator model experiments.

- Define functions for inputting data to the Estimator.
- Train the Estimator using the training-set defined in step 1.
- Evaluate the performance of the Estimator on the test-set defined in step 1.
- Use the trained Estimator to make predictions on other data.

We added the model implementation code of the model including comments in the Appendix section.

B. *Experiment#1 Result*

Fig. 15 below is the snapshot of DNNClassifier Estimator experiment image output with true and predicted label.

Fig. 16 below is the snapshot of DNNClassifier Estimator Model Evaluation result.

Fig. 17 below is the snapshot of C-system disk (./checkpoints_CNN_DNN) directory after training and evaluation for DNNClassifier Estimator.

Fig. 18 below is the snapshot of model graph visualization through TensorBoard for DNNClassifier Estimator from the C-system disk (./checkpoints_CNN_DNN /graph.pbtxt) directory.

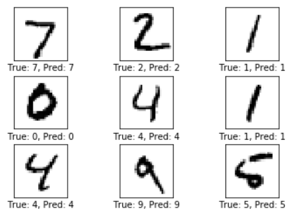


Fig. 15. MNIST Predicted Images and Labels after Training with Pre-Made DNNClassifier Estimator.

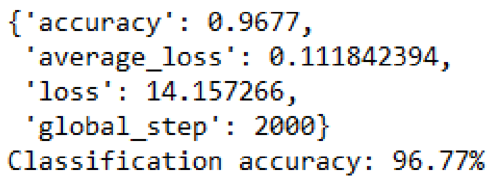
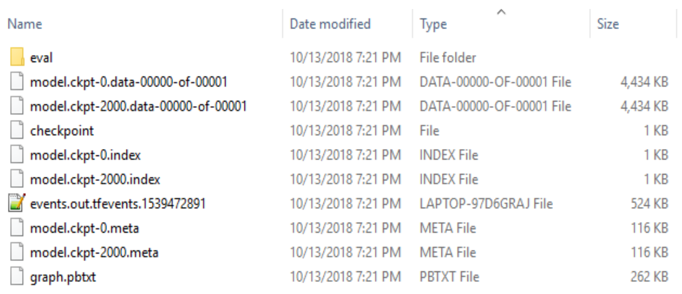

```
{'accuracy': 0.9677,
 'average_loss': 0.111842394,
 'loss': 14.157266,
 'global_step': 2000}
Classification accuracy: 96.77%
```

Fig. 16. Pre-Made Estimator Model Evaluation Result.



Name	Date modified	Type	Size
eval	10/13/2018 7:21 PM	File folder	
model.ckpt-0.data-00000-of-00001	10/13/2018 7:21 PM	DATA-00000-OF-00001 File	4,434 KB
model.ckpt-2000.data-00000-of-00001	10/13/2018 7:21 PM	DATA-00000-OF-00001 File	4,434 KB
checkpoint	10/13/2018 7:21 PM	File	1 KB
model.ckpt-0.index	10/13/2018 7:21 PM	INDEX File	1 KB
model.ckpt-2000.index	10/13/2018 7:21 PM	INDEX File	1 KB
events.out.tfevents.1539472891	10/13/2018 7:21 PM	LAPTOP-97D6GRAJ File	524 KB
model.ckpt-0.meta	10/13/2018 7:21 PM	META File	116 KB
model.ckpt-2000.meta	10/13/2018 7:21 PM	META File	116 KB
graph.pbtxt	10/13/2018 7:21 PM	PBTXT File	262 KB

Fig. 17. Pre-Made Estimator C-System Disk Directory.

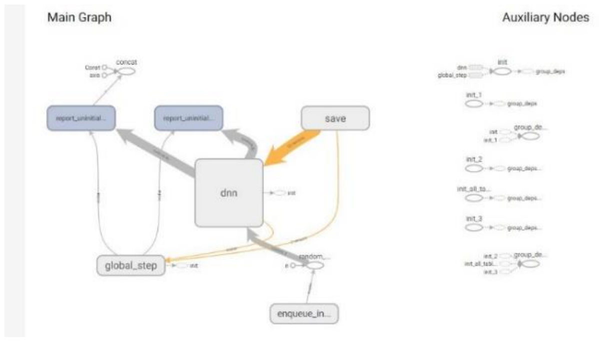


Fig. 18. Pre-Made Estimator Model Graph Visualization.

C. *Experiment#2 CNN Memory Model with TensorFlow custom Estimator*

Fig. 19 below is the snapshot of custom Estimator experiment image output with true and predicted label.

Fig. 20 below is the snapshot of custom Estimator Model Evaluation result:

C-system disk (./checkpoints_CNN_Custom) directory was created after training and evaluation for custom Estimator and graph visualization through TensorBoard for custom Estimator from the C-system disk (./checkpoints_CNN_Custom/graph.pbtxt) directory:

D. *CNN Memory Model Summary*

The MNIST CNN memory model classify the images well and stores the model where there is a mechanism for future prediction while combined with TensorFlow Estimator framework if this model needed to be retrieved at a later time.. In conclusion, based on the above experiments and results, even though CNN deep learning models alone cannot be used to replicate long-term declarative episodic memory. However, we can achieve the research objective while a CNN model is combined with a deep learning framework like Estimator API.

We also conducted custom estimator experiments on pre trained CNN deep learning models such as ResNet, VGG16, InceptionV3 and MobileNet and enable storage capability in similar fashions.

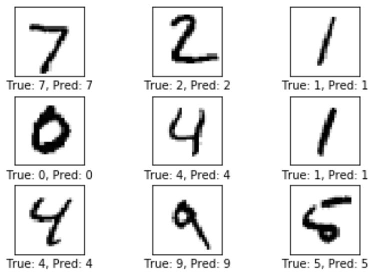
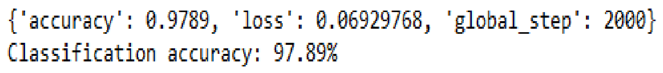


Fig. 19. MNIST Predicted Images and Labels after Training with Custom Estimator.



```
{'accuracy': 0.9789, 'loss': 0.06929768, 'global_step': 2000}
Classification accuracy: 97.89%
```

Fig. 20. Custom Estimator Model Evaluation Result.

E. *Experiment#3 LSTM Memory Model with TensorFlow Save and Restore Framework*

This experiment is to train a LSTM deep learning model from scratch on MNIST dataset and extend the model to build the proposed C-system for storing the model for future prediction. This experiment uses TensorFlow low level API framework tf.train.Saver to save and restore a model by utilizing the tf.Session. We build the LSTM model and pass it to the framework using tf.Session. The model is saved in "/tmp/lstm/" to be our C-system and restored from the same location. The following steps are required to build this model:

1) Make the environment ready through importing the needed libraries.

2) Define the configuration variables

3) Define the Functions

4) Load and preprocess the MNIST dataset and other input parameters to build the Model

5) A functioning Model implementation by TensorFlow

6) Model training on the prepared data

7) Results analysis

F. *Save and Restore API Require of the following Steps*

1) creating an instance of tf.train.Saver() class

2) save the model inside a session

3) Define the Saving Location

4) Call the tf.train.Saver.restore() to restore the model

We added the model implementation code of the model including comments in the Appendix section.

G. *Experiment#3 Result*

Model was trained successfully for 1000 datasets. The evaluation was conducted by restoring the trained model. In addition, the restored model was tested to see if the correct data are preserved. We also captured saver session and restore session log snapshot to observe the memory consistency. Similar to CNN model we observed the C-system disk (/tmp/lstm/) directory after training on MNIST using LSTM. Also, we were able to visualize graph through TensorBoard for LSTM from the C-system disk (/tmp/lstm/) directory.

In conclusion, the LSTM MNIST memory model satisfies the requirement of classifying the images and store the model for future retrieval.

H. *LSTM Memory Model Summary*

The LSTM MNIST memory model classify the images well and stores the model where there is a mechanism for future prediction while combined with TensorFlow Save and Restore framework if this model needed to be retrieved at a later time. Therefore, it is an ideal model to use independently for the objective of this research. In conclusion, based on the above experiment and result, even though LSTM deep learning model alone cannot be used to replicate long-term declarative episodic memory. However, we can achieve the research objective while a LSTM model is combined with a deep learning framework like TensorFlow Save and Restore API.

I. Experiment#4 VAE Memory Model with TensorFlow Save and Restore Framework

This experiment is to train a VAE deep learning model from scratch on MNIST dataset and extend the model to build the proposed C-system for storing the model for future prediction. VAE MNIST uses TensorFlow low level API framework tf.train.Saver to save and restore a model by utilizing the tf.Session. We build the VAE model and pass it to the framework using tf.Session. The model is saved in "/tmp/lstm/" to be our C-system and restored from the same location.

J. Experiment#4 Result

Model was trained successfully for 10000 datasets. The evaluation was conducted by restoring the trained model. In addition, the restored model was tested to see if the correct data are preserved. We observed the Saver session and Restore session C-system disk (/tmp/vae/) directory was created after training on MNIST using VAE. We observed the graph visualization through TensorBoard for VAE from the C-system disk (/tmp/vae/) directory.

In conclusion, the VAE MNIST memory model satisfies the requirement of classifying the images and store the model for future retrieval.

K. VAE Memory Model Summary

The VAE MNIST memory model classify the images well and stores the model where there is a mechanism for future prediction while combined with TensorFlow Save and Restore framework if this model needed to be retrieved at a later time. Therefore, it is an ideal model to use independently for the objective of this research. In conclusion, based on the above experiment and result, even though VAE deep learning model alone cannot be used to replicate long-term declarative episodic memory. However, we can achieve the research objective while a VAE model is combined with a deep learning framework like TensorFlow Save and Restore API.

Finally, we can draw a conclusion based on all these experiments and results that we can produce a desired C-system which can remember and replicate the previous events that occurred while building the deep learning model. Therefore, we can draw a conclusion that deep learning models can be replicated to incorporate human long-term declarative episodic memory storage.

We also conducted experiments on TensorFlow Custom Estimators Framework API on MNIST datasets for CNN ResNet, VGG16, InceptionV3 and MobileNet and enhance the models to be used as storage mechanism.

VI. Discussion

Was a correlation found with human brain?

We assume there is a keen relationship and similarities with human brain long-term episodic memory storage and c-system memory storage unit we built using Tensorflow API. Human long-term episodic memory illustrates events and experiences of previous occurrences. Our deep learning models with c-system storage also originate on events and experiences. Here an event is triggered when a deep learning model is being trained. In addition, the model gathers the experience from the behavior of the data is being trained on. After model gains the experience meaning trained on the data we store the model permanently into the c-system which is a permanent storage location of a disk specified by the framework. In our case, we used our C drive folder to be the replica of the c-system storage unit. In the event, we want the system to bring back the memory we connect the model using framework and the model is able to retrieve the information correctly. We can claim that our model storage capacity is much bigger than a human brain. Herbert Simon's chunk or George A. Miller's magic chunk [11] as illustrated before are very small in comparison to our deep learning memory storage. Here we can store the entire model that may have learn for days and worked on a large set of chunks or numbers. Therefore, we can draw the conclusion that we are able to find a correlation with our deep learning models and the human long-term declarative episodic memory.

VII. Future Work

We have explored into building a framework to build our proposed c-system storage mechanism. However, there are still other different techniques can be applied to build c-system storage mechanism. Below are some recommendations.

- Combine the CNN and LSTM together to build the c-system. CNN will work as the classification model and LSTM will work as the storage unit.
- Other dataset besides MNIST dataset can be tested and evaluated to check how the storage mechanism behaves.
- Our research was to work with images. This research can be enhanced to work with video frames.
- More work can be done to build comparison between human memory storage and the proposed c-system storage.
- Our research was limited to CNN, RNN-LSTM and VAE or pre trained CNN models. This can be expanded to work with other deep learning/ machine learning algorithms.
- We focused into TensorFlow framework for storage. Other frameworks can be evaluated to build the storage mechanism.
- TensorFlow framework low level API allows coding into other languages besides python. This research can be enhanced into multiple directions such as visualization to the web integrating other languages as well.
- TensorFlow.js also can be used to implement web mechanism to connect to the c-system.

VIII. Concluding Remarks

Finally, we reached into conclusion that deep learning models require building a framework to build the c-system to achieve the storage mechanism for an image classification model. In addition, we can replicate the human memory model by enhancing the deep learning classification to model with storage unit.

REFERENCES

[1] Rosenblatt, F. "A model for experiential storage in neural networks". Washington, DC: Spartan Books, 1964.

[2] Abu Kamruzzaman, Yousef Alhwaiti, and Charles C. Tappert " Developing a Deep Learning Model to Implement Rosenblatt's Experiential Memory Brain Model" Future of Information and Communication Conference (FICC) IEEE 2019.

[3] Karayev, S., Jia, Y., Shelhamer, E., Donahue, J., Long, J., Girshick,R., Guadarrama, S., and Darrell, T. "Caffe: Convolutional Architecture for Fast Feature Embedding", UC Berkeley EECS, Berkeley.

[4] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.

[5] Cires, D. , Meier, U., and Schmidhuber, J., "Multi-column Deep Neural Networks for Image Classification", IDSIA / USI-SUPSI, Manno, Switzerland, 2012.

[6] D. O. Hebb, The Organization of Behavior. New York: John Wiley & Sons, 1949.

[7] Teresa Nicole Brooks, Abu Kamruzzaman, Avery Leider and Charles C. Tappert "A Computer Science Perspective on Models of the Mind" Intelligent Systems Conference (IntelliSys), IEEE, 2018.

[8] S. Nahal, The Relationship Between Deep Learning and Brain Function: Proceedings of Student-Faculty Research Day. Pleasantville, NY, USA: CSIS, Pace University, 2017, vol. May 5.

[9] A. Fontana, "A deep learning-inspired model of the hippocampus as storage device of the brain extended dataset," arXiv preprint arXiv:1706.05932, 2017.

[10] Mastin, L. (2010). Types of memory. *The human memory*.

[11] Miller, George A. "The magical number seven, plus or minus two: Some limits on our capacity for processing information." Psychological review 63.2 (1956): 81.

[12] Zhu, W., Lan, C., Xing, J., Zeng, W., Li, Y., Shen, L., & Xie, X. (2016, February). Co-Occurrence Feature Learning for Skeleton Based Action Recognition Using Regularized Deep LSTM Networks. In AAAI (Vol. 2, p. 8).

[13] CNN workflow image - adeshpande3.github.io

[14] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[15] Wikipedia "Long short-term memory" https://en.wikipedia.org /wiki/ Long_short-term_memory, 2018

[16] Cheng, Heng-Tze, et al. "Tensorflow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.

[17] "Visualizing MNIST with a Deep Variational Autoencoder" https://www.kaggle.com/rvislaywade/visualizing-mnist-using-a-variational-autoencoder, 2018

[18] "TensorFlow " https://www.tensorflow.org/ , 2018

[19] "Tensorflow Estimator API" https://blog.10yung.com/tensorflow-estimator-api-note/, 2017

[20] Cheng, Heng-Tze, et al. "Tensorflow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks." Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2017.

[21] "TensorFlow: A proposal of good practices for files, folders and models architecture" https://blog.metaflow.fr/tensorflow-a-proposal-of-good-practices-for-files-folders-and-models-architecture-f23171501ae3, Apr 28, 2017