Indian Institute of Technology Bombay

Department of Computer Science & Engineering

Seminar Report

Adapting Deep Learning Models

Author:
Rishabh Agarwal

Supervisor:
Prof. Sunita Sarawagi

May 4, 2017

Contents

Abstract

Learning quickly is a hallmark of human intelligence, whether it involves recognizing objects from a few examples or quickly learning new skills after just minutes of experience. One of the current limitations of deep learning is the need for tremendous amounts of data. Finding techniques to achieve state-of-the-art performance on tasks with orders of magnitude less data is a very active research area. In this work, we survey some of the recent approaches to tackle this problem of learning new concepts rapidly with very little data.

1 Introduction

We need for massive amounts of data to train deep neural networks. In contrast, humans require comparatively little data to learn a new behavior or to rapidly shift away from an old behavior. For example, a child can generalize the concept of "giraffe" from a single picture in a book, yet our best deep learning systems need hundreds or thousands of examples.

It is estimated that a child has learned almost all of the $10 \sim 30$ thousand object categories in the world by the age of six. This is due not only to the human mind's computational power, but also to its ability to synthesize and learn new object classes from existing information about different, previously learned classes.

Our artificial agents should be able to do the same, learning and adapting quickly from only a few examples, and continuing to adapt as more data becomes available. This kind of fast and flexible learning is challenging, since the agent must integrate its prior experience with a small amount of new information, while avoiding overfitting to the new data. Furthermore, the form of prior experience and new data will depend on the task.

2 Problem Statement

The problem which we'd discuss in this work is as follows:
We are given a large global dataset and a small amount of "personal" labeled data. We trained a model (mostly a deep neural network) using the global data. The goal is to adapt this model for the limited labeled data so as

to personalize the trained model. This problem has various applications in speech recognition, hand writing recognition, and text conversation bots.

Suppose we want to build an app to recognize the handwriting of a person. Assume we are given access to a large global dataset of handwritten characters such as MNIST[?]. Using this dataset, we trained a DNN model and deployed this model in our app. But this app would not work well for all the users as each user have a unique handwriting which may be significantly differ from the handwritings the model have seen in the global dataset. Therefore, we need to adapt our model for each user utilizing the few samples ($1 \sim 10$) we obtain from them.

3 How to solve the problem?

Since we want to incorporate our prior experience in solving the adaptation problem, the area of **Transfer Learning and Domain Adaptation** are quite relevant to it.

According to Goodfellow et al.[?] Transfer Learning and Domain Adaptation refer to "the situation where what has been learned in one setting (i.e., distribution P1) is exploited to improve generalization in another setting (say distribution P2)". These fields are described in detail in the sections 3.1 and 3.2.

This problem can also be posed as a **few-shot learning** problem and is discussed further in section 3.3.

3.1 Transfer Learning

In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task.

The usual approach is to train a base network and then copy its first **N** layers to the first n layers of a target network.Currently, no basis is provided for picking a particular layer from the pretrained network when using transfer learning.

Therefore, it's natural to think about this question: **How much effective is Transfer Learning?** What value of **N** should be used? Do we fine-tune the transferred network to the new task, or are they left frozen?

Yosinski et al.[**?**], sheds some light on this issue and tries to experimentally quantify the generality versus specificity of neurons in each layer of a deep CNN when used for transfer learning tasks. I have discussed their experimental setup and the results below.

Experiment Setup & Results Experiments were performed by training two **M**-layered neural nets (here n = 8) (call them A & B) on two disjoint datasets obtained from ImageNet.

Then, transfer learning was performed by picking first K layers of one of the networks (let's say A) and using them with last N-K layers of either A or B both with and without fine-tuning.

From these experiments, some of the non-obvious conclusions are:

- There are 2 separate issues that cause performance degradation when using transferred features without fine-tuning:
  (i) the specificity of the features themselves, and
  (ii) optimization difficulties due to splitting the base network between co-adapted neurons on neighboring layers.

- Initializing a network with transferred features from almost any number of layers can produce a boost to generalization performance after fine-tuning to a new dataset.

Though, the above results, should be taken with a pinch of salt, as they are heavily dependent on the optimization algorithm used (here, SGD+Momentum) and the transfer task. But the take-away from this paper was that **transfer learning always helps if there is some similarity between the two tasks**.

3.2 Domain Adaptation

Adaptation of neural network models in the context of speech, **speaker adaptation**, is well researched. Acoustic models have evolved starting from GMM[?] or HMM models to hybrid models that are NN/HMM[?], deep neural networks to LSTM[?] based RNN models.

The problem of low performance due to train and test domain differences is acute in speech and is well addressed in the past even before the introduction of neural networks.

We shall further discuss the domain adaptation problem only in the context of neural networks.

3.2.1 Speaker Adaptation of hybrid NN/HMM based on speaker codes

In this work[?], a fast speaker adaptation method was proposed for the hybrid NN-HMM speech recognition model. In a NN/HMM model, the GMM component that is used to estimate the emission probabilities of each state is replaced by neural network which when fed with the features vector outputs the posterior over HMM state labels.

As shown in figure 1, the proposed adaptation method relies on learning adaptation NN and speaker codes. All weights of adaptation NN are standard fully connected layers. The transformed feature vector should have the same dimension as the input feature vector. During the adaptation phase, both the weights of adaptation NN and speaker need to be learned.

3.2.2 Using I-Vector inputs to improve speaker independence

This paper[?] proposed providing additional utterance-level features as inputs to a DNN to facilitate speaker, channel and background normalization.

i-Vectors or identity vectors According to the authors "i-vectors encode precisely those effects to which we want our ASR system to be invariant: speaker, channel and background noise." These vectors are generally used in speaker recognition and verification.

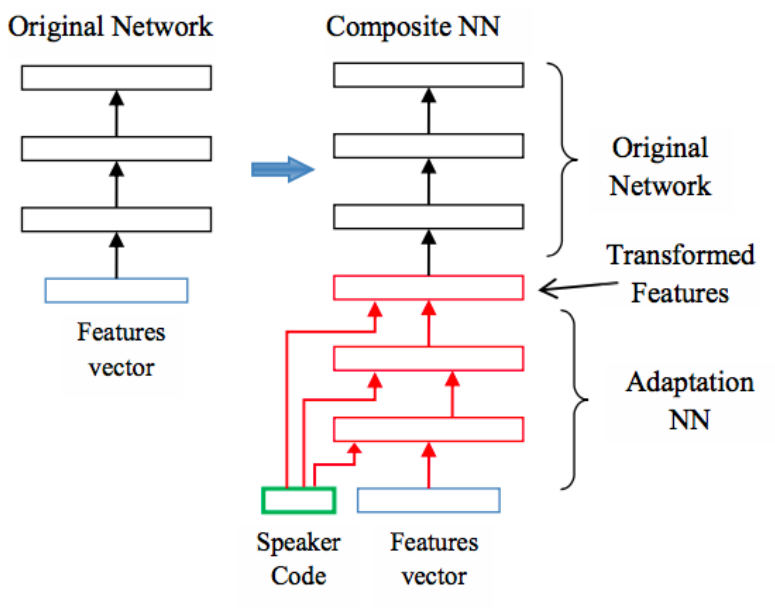


Figure 1: Speaker adaptation of the hybrid NN-HMM model based on speaker code

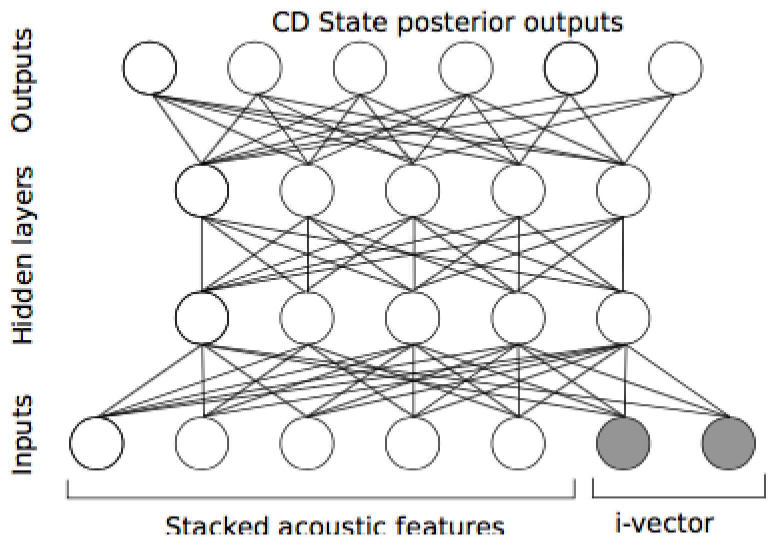


Figure 2: Diagram of a 2-hidden layer neural network with inputs augmented with i-vectors

As shown in the figure 2, the idea is to provide the input with characterization of the speaker should enable it to normalize the signal with respect to them and thus better able to make its outputs invariant to them.

3.3 Few-Shot Learning

One obvious approach to this problem is to use transfer learning as described in Section 3.1 above. But since only have few examples, the normal approach to transfer learning doesn't work.

This can be mitigated by viewing the problem as a few-shot learning problem, an extreme case of transfer learning, which involves learning a new class with only few labeled examples. In this formulation, the limited personal data acts as the new examples on which we apply few-shot learning.

There are two major approaches for this formulation of the problem which are quite different from the speaker adaptation approaches discussed above in section 3.2

- Meta Learning or "Learning to learn"
- Memory Augmented Neural Networks

3.4 Meta Learning

According to Santaro et al.[**?**], meta-learning generally refers to a scenario in which an agent learns at two levels, each associated with different time scales. Rapid learning occurs within a task, for example, when learning to accurately classify within a particular dataset. This learning is guided by knowledge accrued more gradually across tasks, which captures the way in which task structure varies across target domains.

3.4.1 Meta-Learning Problem Set-Up

- Consider a model, denoted $\mathbf{M}$, that maps observations $\mathbf{x}$ to outputs $\mathbf{a}$
- $\mathbf{M}$ is trained to be able to adapt to a large or infinite number of tasks
- Each task $\boldsymbol{T} = \{L(x_1, a_1, \ldots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$ where **L, H** are the loss function and episode length resp,

$q(x_1)$ is the initial distribution over examples,
$q(x_{t+1}|x_t, a_t)$ is the transition distribution

- Define $\rho(T)$ to be a distribution over tasks

In the case of i.i.d. supervised learning problems, the length is H = 1.
During meta-training,

- A task T_i is sampled from $\rho(T)$
- In the K-shot learning setting, **M** is trained with K samples and feedback from the corresponding loss L_{T_i} from T_i
- **M** is then tested on new samples from T_i
- The model **M** is then improved by considering how the test error on new data from q_i changes with respect to the parameters.

In effect, the test error on sampled tasks T_i serves as the training error of the meta-learning process.
During meta-testing:

- New tasks (held out during meta-training) are sampled from $\rho(T)$
- Meta-performance is measured by the model's performance after learning from K samples. Generally, tasks used for meta-testing are held out during meta-training

3.4.2 Matching Networks for One Shot Learning

This is a paper[**?**] on one-shot learning, where we try to learn a class based on very few (or indeed, 1) training examples. The Matching Nets architecture described in this paper **blends deep learning with nearest neighbour, using neural networks augmented with memory (LSTMs)!**

- **The Training Protocol**
  As the authors point out in the conclusion "one-shot learning is much easier if you train the network to do one-shot learning". Therefore, we want the test-time protocol to exactly match the training time protocol. To create each"episode" of training from a dataset of examples then:

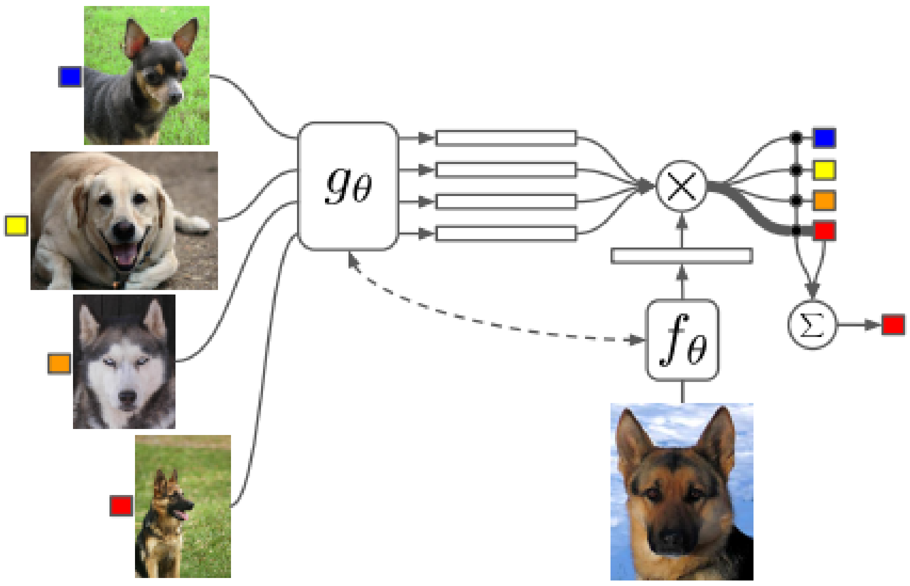


Figure 3: Matching Nets Architecture.

- Sample a task T from the training data, e.g. select 5 labels, and up to 5 examples per label (i.e. 5-25 examples).
- To form one episode sample a label set L (e.g. cats, dogs) and then use L to sample the support set S and a batch B of examples to evaluate loss on.
- The Matching Net is then trained to minimize the error predicting the labels in the batch B conditioned on the support set S.

This is a form of meta-learning since the training procedure explicitly learns to learn from a given support set to minimize a loss over a batch.

- **The model**
  Basically we're building a differentiable nearest neighbor++. The output $\hat{y}$ for a test example $\hat{x}$ is computed very similar to what you might see in Nearest Neighbors:

$$\hat{y} = \sum_{i=1}^{n} a(x, x_i) y_i$$

where **a** acts as a kernel, computing the extent to which $\hat{x}$ is similar to a training example x_i, and then the labels from the training examples y_i are weight-blended together accordingly. The paper doesn't mention this but I assume for classification y_i would presumably be one-hot vectors.

Now, we're going to embed both the training examples x_i and the test example $\hat{x}$, and we'll interpret their inner products (or here a cosine similarity) as the "match", and pass that through a softmax to get normalized mixing weights so they add up to 1.

$$a(x, x_i) = e^{c(f(\hat{x}),g(x_i))} / \sum_{i=1}^{n} e^{c(f(\hat{x}),g(x_i))}$$

Here **c()** is cosine distance, which is implemented by normalizing the two input vectors to have unit L2 norm and taking a dot product. $f(\hat{x})$ corresponds to the embedding of a test example and the function $g(x_i)$ is the training embedding of the i^{th} training example.

Embedding the training examples. The function g is a bidirectional LSTM over the examples:

$$\overrightarrow{h}_i, \overrightarrow{c}_i = LSTM(g'(x_i), \overrightarrow{h}_{i-1}, \overrightarrow{c}_{i-1})$$
$$\overleftarrow{h}_i, \overleftarrow{c}_i = LSTM(g'(x_i), \overleftarrow{h}_{i-1}, \overleftarrow{c}_{i-1})$$

i.e. encoding of i^{th} example x_i is a function of its"raw" embedding $g'(x_i)$ and the embedding of its friends, communicated through the bidirectional network's hidden states i.e. each training example is a function of not just itself but all of its friends in the set. This is part of the ++ above, because in a normal nearest neighbor you wouldn't change the representation of an example as a function of the other data points in the training set.

Embedding the test example The function f is an LSTM that processes for a fixed amount (K time steps) and at each point also attends over the examples in the training set. The encoding is the last hidden state of the LSTM. Again, this way we're allowing the network to change its encoding of the test example as a function of the training examples.

$$\hat{h}_k, c_k = LSTM(f'(\hat{x}, [h_{k-1}, r_{k-1}], c_{k-1})$$
$$h_k = \hat{h}_k + f'(\hat{(}x)$$
$$r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i))g(x_i)$$
$$a(h_{k-1}, g(x_i)) = softmax(h_{k-1}^T g(x_i))$$

It's really just a vanilla LSTM with attention where the input at each time step is constant ($f'(\hat{x})$, an encoding of the test example all by itself) and the hidden state is a function of previous hidden state but also a concatenated readout vector r, which we obtain by attending over the encoded training examples (encoded with g from above).

Experiments were done on the Omniglot dataset, ImageNet and they also introduced an one-shot task on the Penn Treebank.

3.4.3 MAML for Fast Adaptation of Deep Networks

The novel idea of this paper[?] is to **train the model's initial parameters such that it has maximal performance on a new task after the parameters have been updated**. This can be viewed as building an internal representation that is broadly suitable for many tasks.

This method assumes that the model will be fine-tuned using a gradient-based learning rule on a new task and therefore, tries to learn a model such that this gradient-based learning rule can make rapid progress on new tasks drawn from $\rho(T)$, without overfitting.

MAML Algorithm Consider a model represented by a parametrized function $\mathbf{M}_\theta$ with parameters θ. When adapting to a new task T_i, the model's parameters θ becomes $\theta_i{}'$. Train the model parameters by optimizing for the performance of $\mathbf{M}_{\theta_{\mathrm{i}}{}'}$ w.r.t θ across tasks sampled from $\rho(T)$.

```
Algorithm 1 Model-Agnostic Meta-Learning
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
 1: randomly initialize $\theta$
 2: while not done do
 3:   Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:   for all $\mathcal{T}_i$ do
 5:     Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
 6:     Compute adapted parameters with gradient de-
        scent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
 7:   end for
 8:   Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
 9: end while
```

Figure 4: The generic MAML algorithm

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
 1: randomly initialize $\theta$
 2: while not done do
 3:   Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:   for all $\mathcal{T}_i$ do
 5:     Sample $K$ datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
 6:     Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (1)
        or (2)
 7:     Compute adapted parameters with gradient descent:
        $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
 8:     Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the
        meta-update
 9:   end for
10:   Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$
      and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 1 or 2
11: end while
```

Figure 5: MAML for few-shot supervised learning

Please refer to Figure 4 and 5 for the generic and supervised learning specific algorithm respectively.

This model has the advantage of being agnostic to the form of the model **M** and to the particular learning task. Also, it does not introduce any additional parameters into the learning process. In addition to above, it produces a good weight initialization, and thus, further adaptation can be performed with any amount of data and any number of gradient steps.

The only limitation of this paper is that meta-learning training phase can underfit because of small number of adaptation steps. Also, an additional hyperparameter, which is the model's step size α is introduced, which was obtained though fine tuning.

3.5 Memory Augmented Neural Networks

3.5.1 Learning to Remember Rare Events

Memory-augmented DNNs are still limited when it comes to life-long and one-shot learning, especially in remembering rare events! It is necessary to extend the training data and re-train them to handle rare or new events. Humans, on the other hand, learn in a life-long fashion, often from single examples.

Key Idea Kaiser et al[**?**] introduced a memory module for life-long learning which essentially uses efficient k-nearest neighbors classification and adds one-shot learning capability to any supervised neural network.

Memory Module Memory module, consists of key-value pairs. Keys are activations of a chosen layer of a neural network. Values are the ground-truth targets for the given example. Formally,
$M = (K_{memory-size \times key-size}, V_{memory-size}, A_{memory-size})$ where
K is a matrix of memory keys,
V is a vector of memory values,
A tracks the age of items stored in memory

As the network is trained, its memory increases and becomes more useful. Given a new example, the network writes it to memory and is able to use it afterwards, even if the example was presented just once.

We can query this memory module using a suitable normalized query vector **q**. The nearest neighbor of q in M is defined as any of the keys that maximize the dot product with q i.e. NN(q, M) = $argmax_i \, q.K[i]$.

When given a query q, the memory M= (K, V, A) will compute k-nearest neighbors(sorted by decreasing cosine similarity)

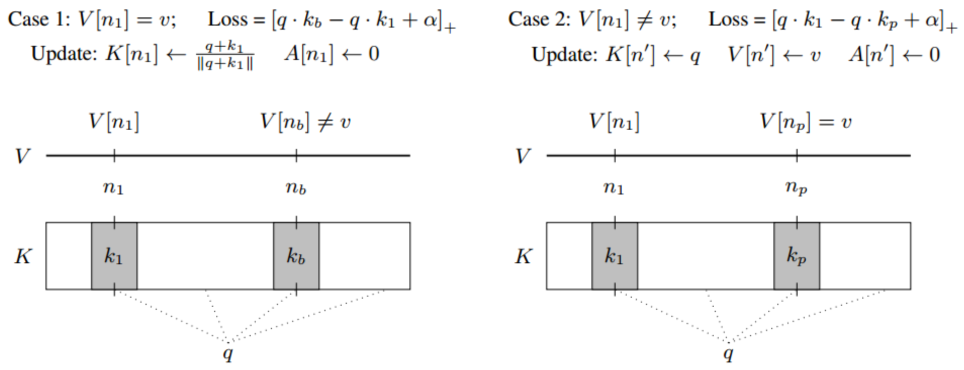
$$(n_1, \ldots, n_k) = NN_k(q, M)$$

and return, as the main result, $V[n_1]$. In case $V[n_1]$ is further embedded into a dense vector, compute $d_i = q \cdot [n_i]$ and $softmax(d_1 \cdot t, \ldots, d_k \cdot t)$ where t = 40 and return the embedded vector times the corresponding softmax component t so as to provide a signal about confidence of the memory.

Memory Loss Suppose, in addition to a query **q** we are also given the correct supervised value **v**. Let p and b be the smallest index such that $V[n_p] = v$ and $V[n_b] \neq v$ respectively. When no positive neighbor is among the top-k, we pick any vector from memory with value v instead of $K[n_p]$. The memory loss is given by

$$loss(q, v, M) = max(0, q \cdot K[n_b] - q \cdot K[n_p] + \alpha)$$

Memory Update Using the memory loss, the memory is updated as shown in Figure 6.



Case 1: $V[n_1] = v$; Loss = $[q \cdot k_b - q \cdot k_1 + \alpha]_+$
Update: $K[n_1] \leftarrow \frac{q+k_1}{\|q+k_1\|}$ $A[n_1] \leftarrow 0$
$V[n_1]$
$V[n_b] \neq v$
V
n_1
n_b
K
k_1
k_b
q
Case 2: $V[n_1] \neq v$; Loss = $[q \cdot k_1 - q \cdot k_p + \alpha]_+$
Update: $K[n'] \leftarrow q$ $V[n'] \leftarrow v$ $A[n'] \leftarrow 0$
$V[n_1]$
$V[n_p] = v$
V
n_1
n_p
K
k_1
k_p
q

Figure 6: The operation of the memory module on a query q with correct value v. $n\prime$ is a randomly chosen memory item with maximum age.

Using the Memory Module Which layer to use to generate queries, and how to use the output of the module? n the simplest case, use the final layer of a network as query and the output of the module is directly used for classification.

Alternatively , we can add the memory module to seq2seq RNNs. A query to memory is made in every step of the decoder network.

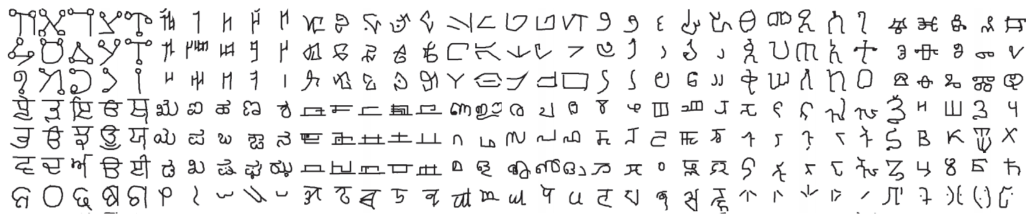
Memory output is embedded again into a dense representation (using the same embedding as the decoder). This embedding is combined with inputs from other layers of the network.

Experiments These experiments were performed in this paper:

- One-shot learning task Omniglot, which is the only dataset with explicit one-shot learning evaluation.
- A synthetic task that requires life-long one-shot learning.
- Training an English-German translation model that has our life-long one-shot learning module.

Results The results in the model are convincing. Omniglot is a good sanity test and the performance is surprisingly good.
The synthetic task shows us that the authors claims hold and highlight the need for better benchmarks in the domain of one-shot learning.
The translation task eventually makes a very strong point on practical usefulness of the proposed model.

3.6 Omniglot - A Common Experiment



Omniglot dataset[?], which consists 20 instances of 1623 characters from 50 different alphabets. Each instance was drawn by a different person.Omniglot is a standard benchmark for few-shot learning.

The experimental protocol, as proposed by Vinyals et al[?], involves fast learning of N-way classification with 1 or 5 shots. The problem of N-way classification is set up as follows: select N unseen character classes, provide the model with K different in-stances of each of the N classes, and evaluate the model's ability to classify new instances within the N classes.

1200 characters are randomly selected for training, irrespective of alphabet, and remaining are used for testing. The dataset is augmented with rotations by multiples of 90 degrees.

Model	**5-way Accuracy**		**20-way Accuracy**	
	1-shot	**5-shot**	**1-shot**	**5-shot**
Matching Networks[?]	98.1%	98.9%	93.8%	98.5%
Memory Module[?]	98.4%	99.6%	95.0%	98.6%
MAML[?]	98.7% ± 0.8%	99 : 9% ± 0.3%	93.1% ± 11.5%	98.8% ± 0.6%

Table 1: Few-Shot Omniglot classification on held-out characters

Using the same architecture, the models discussed above gives the results as presented in Table 1.

3.7 Conclusion

In this work, we explored a lot of possible ways to approach our problem which look quite promising too. The next logical step is to applying these approaches to our problem and testing them out. Possibly, we can build up on these approaches and come up with a entirely new method.