12-2017

One-shot Learning In Deep Sequential Generative Models

Hanyu Guo
Clemson University

One-shot Learning In Deep Sequential Generative Models

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Hanyu Guo
Decemeber 2017

Accepted by:
Dr. Melissa C. Smith, Committee Chair
Dr. Walter B. Ligon III
Dr. Adam W. Hoover

Abstract

Regardless of the Deep Learning community's continuous advancements, the challenging domain of one-shot learning still persists. While the human brain is capable of learning a new visual concept with ease, sometimes even at a glance, Deep Learning-based techniques show serious drawbacks in handling problems with small datasets. Much of the existing work on one-shot learning employs a variety of sophisticated network algorithms, prior domain knowledge, and data manipulation to address the generalization challenges presented in such problems. In this work, we demonstrate a one-shot learning method that contains three learning networks — a deep sequential generative model, a candidate network, and a Matching Network — thus offering an alternative approach to solving the one-shot classification problem. The proposed framework does not require domain knowledge, making it potentially portable to other domains. We show that our algorithm improves accuracy from 95.5% to 96.1% on the Omniglot dataset 20-way one-shot learning compared to current state-of-the-art.

Acknowledgments

This thesis was made possible by the help and support of the following faculty members, family members and colleagues.

I begin by conveying my sincere gratitude to my academic advisor Dr. Melissa Smith. Dr. Smith's continuous support, constant encouragement, and insightful suggestions show me the path for completing of this research and thesis.

I would also like to thank Dr. Walter Ligon and Dr. Adam Hoover for serving on my committee and providing review and valuable comments.

I am also thankful for my parents, who provided constant support during my research and writing this manuscript.

Finally, I would like to acknowledge my fellow colleagues from the Future Computing Technologies Lab at Clemson University: Sufeng, Eddie, Jesse, Colin, Ben and many others whom I fail to mention for providing me an excellent working environment during this process.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Despite the recent successes achieved by Deep Learning (e.g. image classification [19], machine translation [56], speech recognition [52], etc.), state-of-the-art Deep Learning models based on the gradient descent algorithm require large amount of training samples to be incrementally trained. The human brain is able to understand new concepts even with very small amount of data. Consider Figure 1.1 classification example, when humans perform the one-shot learning task, the natural way to classify the data is to mutually distinguish the structure of patterns. If several patterns are similar, it is psychologically plausible to cache them as candidates, and the final decision is made based upon these candidates. In this paper, we develop a novel method that follows this intuition: a model represents spatial structure information through attention, narrowing down the problem to a small set, then using meta-learning for classification.

There are two notable clues driving our method: 1) Unlike the traditional method of training by end-to-end fashion, during the one-shot classification process, presumably, the human brain does not recall the memorized concept based on experiences, but rather uses working memory to make a distinction between different

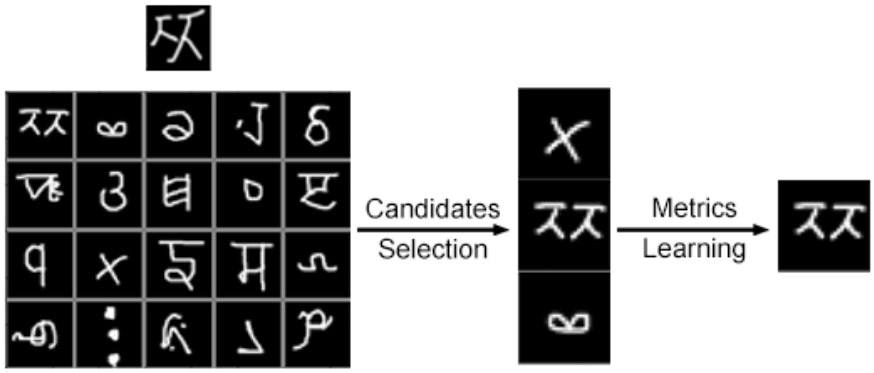


Figure 1.1: Proposed one-shot learner procedure. An example shows how boosted 20-way one-shot learning works. **Left:** a 20-way one-shot classification sample. **Middle:** the candidate selection first narrows down the answer by choosing top-k results based on trained prior knowledge. **Right:** a differentiable metric learning could be adopted for final classification in top-k results (k=3 in the figure).

patterns. The study findings in [33] suggest that the ventrolateral prefrontal cortex may act as a "switch," selectively turning one-shot learning 'on' and 'off' as required. Presumably, when the human brain performs incremental learning, we gradually acquire knowledge, but with one-shot learning, we rapidly learn from a single pairing of a potential stimulus and an outcome, separately. 2) Recent work [17, 43] developed a deep sequential generative model that combines deep neural networks with hierarchical latent variable models to show competitive results of one/few shot generalization, while one-shot learning is still missing from this scenario. Our work starts from these two clues to resolve one-shot learning tasks, and we have made the following observations:

• Hypothetically, the human brain already stores rich low-level features and domain knowledge. During one-shot classification processes, the brain organizes the pattern as spatial structure with casual representation. The domain knowledge and one-shot classification use two different brain areas.

• One possible way for the human brain to differentiate the one-shot pattern

might be first picking up several candidates when ambiguities appear.

• Presumably, one-shot classification in the human brain involves more neural dynamics rather than synaptic dynamics, meaning that forward propagation can be understood as an optimization procedure rather than traditional training-testing configuration.

• Meta-learning-based algorithms show higher prediction accuracy when N is small, where N is the number of classes (e.g. 98.1% for 5-way 1-shot vs. 93.8% for 20-way 1-shot [54]).

These observations inspired us to put forward the following questions: can we first narrow down selections to small set of candidates, while using meta-learning to handle the similar patterns (corresponding to observations 2 & 4)? Instead of directly using traditional meta-learning, is there a way to separate the domain structure knowledge with the meta-learning module (corresponding to observation 1)? How can we execute the optimization procedure upon a generative process or associative memory recall to do inference with such a small amount of data (corresponding to observation 3)?

To address these questions, we proposed a one-shot learning framework using a deep sequential generative model. The framework consists of three separate networks that mimic the human decision making process in one-shot classification with enhancements: the exploration support set are generated by conditional sequential generative process; these support sets are forwarded to the inference module, which is followed by a candidate network to filter out candidate support set; the final result is obtained by fetching the candidate support set into any existing one-shot classifier (e.g. Matching Networks [54]). We compare the model with other competitive computational models on the Omniglot dataset. We find that the presented method surpasses state-of-the-art accuracy developed by [30]. Note that the introduced framework is generic in the

sense that the three modules could be replaced by other well-developed algorithms.

This thesis offers three contributions as initial steps:

• **General framework for one-shot learning.** We introduce a framework that takes advantage of a deep sequential generative model and existing work on one-shot classification. Each component could be replaced by other alternatives.

• **Sequential generative model.** We employ the deep sequential generative model to extract spatial structure information of the given input data, which offers good generalization and feature representation.

• **Candidate network.** We introduce a candidate network into one-shot learning that improves the metric-learning by reduced candidates.

Chapter 2

Research Design and Methods

In this chapter, we first define the task we try to solve. A one-shot learning framework would be introduced and the training procedure would be explained.

2.1 Task Description

We first begin by describing the dataset used. For a general machine learning problem, we are interested in dataset D, which is split into two groups: D_{train} and D_{test}. D_{train} is used for optimizing the parameter θ and the generalization of parameter θ is tested on D_{test}. However, in our cases, the dataset is split into three groups: $D_{gen-train}$, $D_{os-train}$ and $D_{os-test}$. On $D_{gen-train}$, we are interested in training the generative model to generate a reasonable target that is conditioned on the given input.

We consider the set of N-way, k-shot classification tasks, where each dataset $D_{os-train}$ contains k labeled training examples for each of the N classes and $D_{os-test}$ consists of a set of examples for evaluation. In this research, we focus on the case where $N = 20$ and $k = 1$ (one-shot).

2.2 Proposed Methodology

2.2.1 Framework

To recognize patterns amongst k-training samples, it is natural to enable the model to represent features in an abstract way. In order to enrich the feature representation, we use a deep sequential generative model to extract the "structure description" instead of using conventional convolutional neural networks (CNN). The principles behind this solution are modeled after human techniques used during one-shot classification: humans discriminate one-shot patterns through both spatial structure information (e.g. stroke, sub-stroke) and temporal information (e.g. sequence of strokes, relationships between strokes). In our framework, we use DRAW [17] as our sequential generative model, however, it should be noted that alternative models [8, 43] could be applied. DRAW encodes the trajectory into a sequence of vectors, which represents the spatial and sequential information. The sequence of vectors is then fed into the candidate network, which chooses encoder states from the top-n predictions to be used as support sets for the matching network. In order to increase the generalization capabilities of the candidate network, we use the decoder provided by DRAW to generate a large amount of synthetic data. Specifically, we sample the latent variables z_t in the synthetic data to generate a variety of image data (with the same label) that is conditioned on k-shot samples from $\mathbf{x}'$. Since existing work suggests that the prediction accuracy increases as the number of support sets decreases, we choose the top-n candidates (the support set) to be classified by either meta-learning (e.g. MANN [45]) or metric-learning (e.g. Matching Networks [54]). In this research, we use the matching network as the final one-shot classifier because of its reportedly superior performance.

Our framework can be divided into three parts: DRAW, the candidate network,

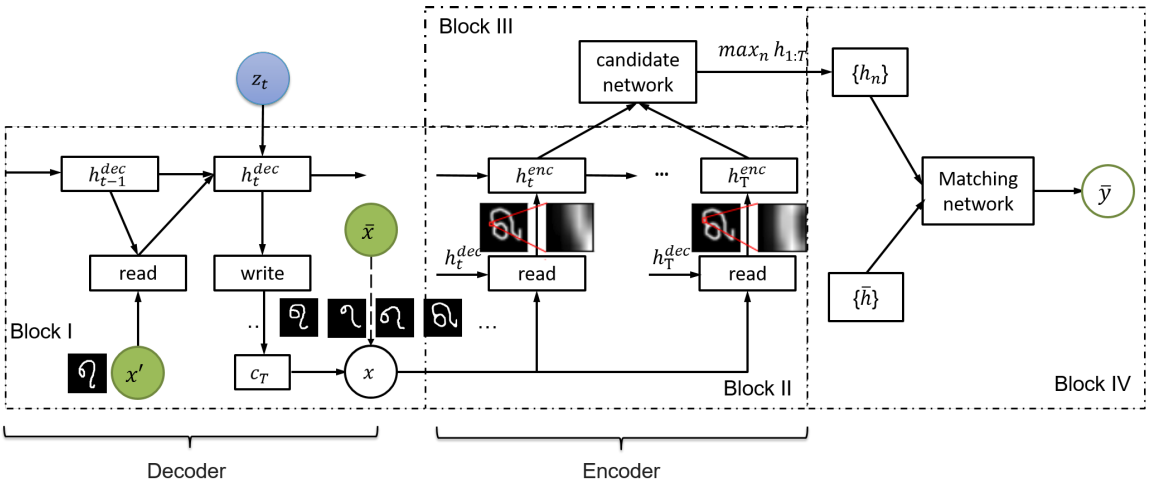


Figure 2.1: Proposed one-shot learning architecture. The green circle represents the input for the model, where x' is the one shot training set, $\bar{x}$ is the test image, and $\bar{y}$ is the prediction. z_t is the time-variant latent variable for the generative process.

and the matching network, as shown in Figure 2.1. All parameters are represented as $\theta = \{\theta_D, \theta_E, \theta_C, \theta_M\}$ for the (DRAW) decoder, (DRAW) encoder, candidate network, and matching network, respectively. Additionally, we define four distinct models, as shown in Figure 2.1, which will be discussed in later sections. Note that the modularity of our framework allows for each component to be optionally replaced by a similar technique. Note that our method does not require any data augmentation because the previously mentioned generative process provides guaranteed diversity amongst the conditioned samples.

Sequential Generative Model. When compared with the original DRAW model, our framework has the following minor changes. Instead of using a single, shared RNN, we deploy two separate RNNs to account for both the conditional generative phase and the inference phase. From our observations, the prediction performance decreases when using only one RNN. The attention mechanism is embedded inside the read/write operations. For the write attention module, the spatial transformer

network is used as a general affine transformation, allowing for the model to address position, scale, and rotation of the independent image parts. As we show later (see Figure 4.2), adopting a selective read attention module that convolves input images with a learned 2D Gaussian filter performs more consistently than when using a spatial transformer network.

Candidate Network. We introduce a candidate network that narrows down the original N classes to n candidates for the k-shot classifier. We simply choose a multi-layer neural network as the candidate network as shown in Equation 2.6. For each k-shot training and testing pair, the candidate network must be re-optimized, which could be interpreted as emulating the optimization of forward propagation (similar to the behavior in neural dynamics). In addition, using a candidate network allows for the model to refresh its parameter θ_C for each new training-testing pair without affecting parameters θ_D and θ_E (pre-trained domain experience/knowledge).

Compared with conventional one-shot classification methods [54, 27], we use the encoder states provided by DRAW as input instead of CNN encoded feature data. For each class, the probability is calculated via $\mathbf{c} = softmax(f([h_{1:T}; \theta_C]))$, where f is the multi-layer neural network, $h_{1:T}$ represents the concatenation of all the encoder states produced by DRAW, and where T is the RNN step size. We keep the top-n predictions, where n is required to match with the number of support sets provided to the matching network. This advantageously reduces the overall difficulty of the problem from N-possibilities to n-possibilities, where it has been proved that the n-way one-shot classifier demonstrates better performance than N-way classification [54, 45].

Matching Network. We replace the CNN in the matching network by concatenating the encoder state vectors. The matching network applies the cosine similarity measurement between the support set and the encoder states of unseen

images.

2.2.2 Training and Classification

Our algorithm consists of three phases: the pre-training phase, which trains the DRAW model via unsupervised learning and uses the $\mathbf{h}_{1:T}$ set from DRAW to train the matching network; the one-shot training phase, which trains the candidate network via the generative process; and the one-shot classification phase, which creates predictions by using DRAW, the candidate network, and the matching network.

Pre-training. During the pre-training phase, we train the DRAW model and the matching network separately. The loss function of DRAW optimizes the lower bound of marginal likelihood by using Equation 2.1 (Appendix A provides details of DRAW training) and the loss of the matching network is Equation 2.2, where $\bar{\mathbf{h}}$ is the encoder states of unseen sample, $\bar{\mathbf{y}}$ is the label of $\bar{\mathbf{h}}$, and $\mathbf{H}^{(n)}$ is the support sets (n encoder states of samples). We aim to optimize the parameters θ_D, θ_E, and θ_M. The algorithm is detailed in Algorithm 1.

$$\mathcal{L}_{DRAW} = -\log D(x|c_T) + \sum_{i=1}^{T} KL[Q(z|x)||P(z)] \text{ [17]} \tag{2.1}$$

$$\mathcal{L}_{MN} = \mathbb{E}_{S' \sim L}[-\log P(\bar{y}|\bar{\mathbf{h}}, \mathbf{H}^{(n)})] \text{ [54]} \tag{2.2}$$

One-shot training. During the one-shot training phase, for each new training/testing pair, we optimize the candidate network by Equation 2.3, where $\hat{\mathbf{y}}$ is the label of $\mathbf{H}$ and $\mathbf{y}$ is the prediction. We form the one-shot training process as follows:

$$\mathcal{L}_C = CrossEntropy(\hat{\mathbf{y}}, \mathbf{y}) \tag{2.3}$$

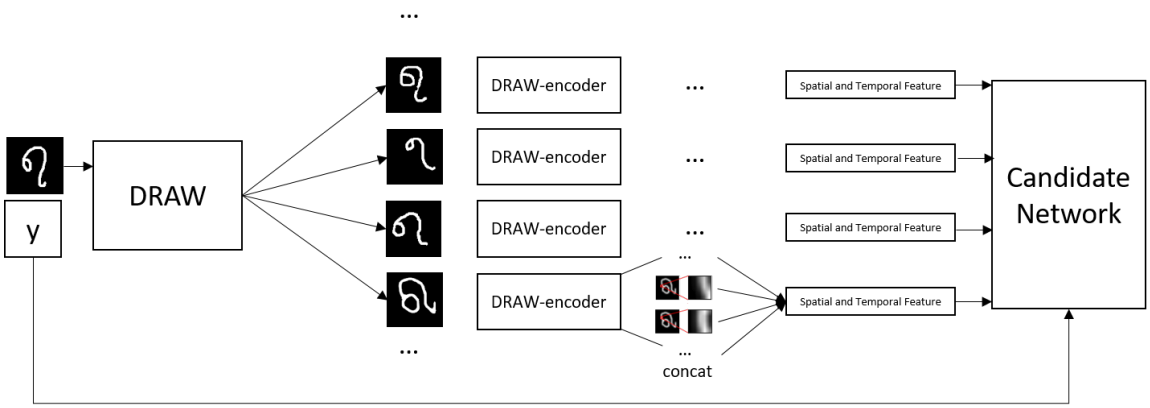


Figure 2.2: One-shot training. given single training example and its label. From left to the right: the training example is fed into DRAW model, which generates homogeneous examples; these examples are fed into the DRAW-encoder and for each homogeneous example, its encoder states are concatenated to form a spatial and temporal feature, which is then fed into candidate network. For the homogeneous examples they share the same label of the given training example.

$$\mathbf{X}_T = f_D(\mathbf{X}', \mathbf{z}_{1:T}; \theta_D), \tag{2.4}$$

$$\mathbf{H}_{1:T} = f_E(\mathbf{X}_T; \theta_E), \tag{2.5}$$

$$\mathbf{y} = f_C(\mathbf{H}_{1:T}; \theta_C), \tag{2.6}$$

where $\mathbf{X}'$ is the one/few-shot training set; f_E and f_D are the encoder and decoder of the DRAW model, respectively; T is the DRAW time step; f_C represents the candidate network; $\mathbf{X}_T$ is the samples produced via generative process; and $\mathbf{z}_{1:T}$ is the time-variant latent variables. We keep θ_D and θ_E fixed, while optimizing on θ_C.

One-shot classification. During the classification phase, we form the prediction process as follows:

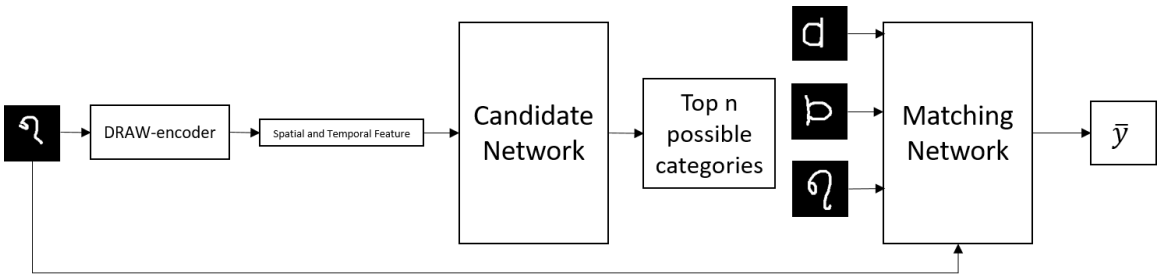


Figure 2.3: One-shot classification. given single testing example. From left to right: the example is directly fed into the DRAW-encoder and its concatenated encoder states are passed into the candidate network; candidate networks produce the top n possible categories of the testing example; the top n category training examples and the testing example are fed into matching network.

$$\mathbf{H}_{1:T} = f_E(\mathbf{X}'; \theta_E), \tag{2.7}$$

$$\bar{\mathbf{h}}_{1:T} = f_E(\bar{\mathbf{x}}; \theta_E), \tag{2.8}$$

$$\mathbf{C}^{(N)} = f_C(\bar{\mathbf{h}}_{1:T}; \theta_C), \tag{2.9}$$

$$\mathbf{P}^{(n)} = \arg max_n \mathbf{C}^{(N)}, \tag{2.10}$$

$$\bar{\mathbf{y}} = f_M(\mathbf{H}[\mathbf{P}^{(n)}], \bar{\mathbf{h}}_{1:T}; \theta_M), \tag{2.11}$$

where $\bar{\mathbf{x}}$ is the one/few-shot testing examples; the candidate network f_C outputs the probability of N classes $\mathbf{C}^{(N)}$; $\arg\max_n$ selects top-n candidate indices $\mathbf{P}^{(n)}$ from $\mathbf{C}^{(N)}$, where $n < N$; and f_M is the matching network that takes support sets $\mathbf{H}_{1:T}[\mathbf{P}^{(n)}]$ and unlabeled sets $\bar{\mathbf{h}}_{1:T}$ as input to make the final prediction. Other notation follows the same as Equations (2.4), (2.5) and (2.6). The algorithm is detailed in Algorithm 2.

Algorithm 1 Pre-Training Procedure

Input: sample with replacement from D_{train}

θ_D, θ_E , $\theta_M \leftarrow$ random initialization

$\triangleright$ Train DRAW

for $i = 1$ **to** i_{max} **do**

$X_i \leftarrow$ random batch from D_{train}

Update θ_D and θ_E using $\bigtriangledown_{\mathcal{L}_{DRAW}}(X_i; \theta_D, \theta_E)$

end for

$\triangleright$ Train Matching Networks

for $i = 1$ **to** i_{max} **do**

$X, \bar{x}, \bar{y} \leftarrow$ random batch from D_{train}

$H \leftarrow f_E(X; \theta_E)$

$\bar{h} \leftarrow f_E(\bar{x}; \theta_E)$

Update θ_M using $\bigtriangledown_{\mathcal{L}_M}(H, \bar{h}, \bar{y}; \theta_M)$

end for

Algorithm 2 One-shot Training Procedure

Input: sample with replacement from D_{test}

θ_D, θ_E , $\theta_M \leftarrow$ pre-trained parameter

$\theta_C \leftarrow$ random initialize

$X, \hat{Y} \leftarrow$ random batch from D_{test}

for $i = 1$ **to** i_{max} **do**

$X_T \leftarrow f_D(X; \theta_D)$

$H \leftarrow f_E(X_T; \theta_E)$

$Y \leftarrow f_C(H; \theta_C)$

Update θ_C using $\bigtriangledown_{\mathcal{L}_C}(\hat{Y}, Y; \theta_C)$

end for

2.3 Summary

In this chapter, we described the dataset used. The problem this research solved is 20-way, one-shot classification problem. The methodology is then presented in the perspective of its intuition. The proposed one-shot learning framework can be divided into four components: a generative model, a feature extraction encoder, a candidate network and a Matching Network. Lastly, training and classification procedure are presented.

Chapter 3

Background and Related Work

In recent years, deep neural networks (DNN) have emerged as the dominant approach for solving problems such as image processing [20, 51, 48, 11, 35], natural language processing [28, 50], speech recognition [18], robot control [60, 34], etc. The simplicity and efficiency of DNNs ease the process of designing a target function and extracting features manually by using the gradient descent-based method to approximate an arbitrary function. However, one obvious restriction of the gradient descent method is that neural networks require large amounts of training data. In many practical cases, there might not be an adequate amount of data because of inaccessibility or cost, which greatly limits the utilization of DNN in some scenarios.

Although the exact data necessary may not be available, in many scenarios there are options to use similar data for which transfer learning [59, 10] can be applied. Transfer learning is a common training technique that focuses on solving one problem and then applying the solution to related problems. In the domain of image recognition, few people train DNNs from scratch, since having a sufficient dataset is a rare case. Commonly, researchers first pre-train the DNN using any available dataset (e.g. ImageNet [44], which contains 1.2 million images with 1000 categories)

and then fine-tune the neural network using the domain-specific data. The pre-trained weights for DNNs are used as initialization techniques for feature extraction. Even though transfer learning can help alleviate the requirement of the size of data, in many scenarios we only have one or few training samples. Using traditional deep learning based algorithms show significant failure.

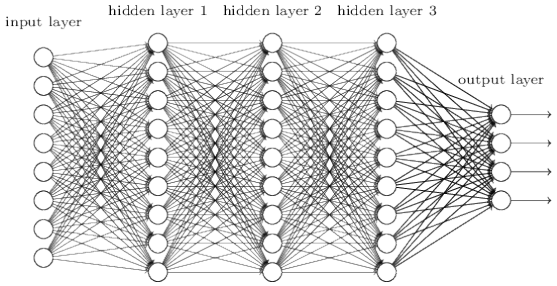
3.1 Multilayer Perceptron

The Multilayer perceptron (MLP), also known as feedforward neural network (FNN), consists of multiple (more than two) fully-connected layers. A fully-connected layer connects every neuron of its previous layer with every neuron in its own layer. Each layer is followed with an nonlinear activation function, which differentiates the MLP from a linear model and enables it to process the data a in highly nonlinear transformation. In the perspective of graph theory, the MLP could be viewed as a layer-wise bipartite graph. Mathematically, each layer of a MLP could be formulated as a matrix multiplication with vector addition. In Equation 3.1, w is the weight matrix $\in R^{m \times n}$, b is the bias vector $\in R^n$ and $\sigma(\cdot)$ is the activation function; x could either be the input or the activation of the previous layer.

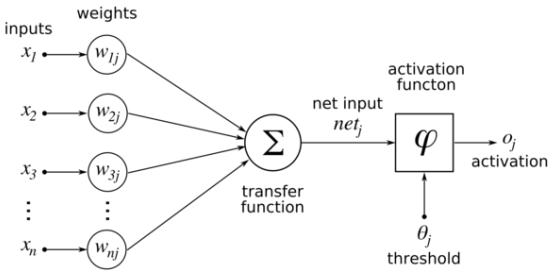
$$f(x) = \sigma(wx + b) \tag{3.1}$$

3.1.1 Activation Functions

In deep learning, activation functions request a set of non-linear functions. By adding activation functions into neural networks, it enables the network to distinguish features that are not linearly-separable. An activation function usually has the



(a) Multilayer perceptron[13]



(b) Transfer (Activation) function[13]

following properties: non-linear, differentiable, and monotonic. Popular activation functions include sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU) [38], Leaky Rectified Linear Unit (Leaky ReLU) [36], Exponential Linear unit (ELu) [6], Randomized leaky Rectified Linear Unit (RReLU) [57], SoftPlus [12], maxout [15] etc.

3.2 Convolutional Neural Networks

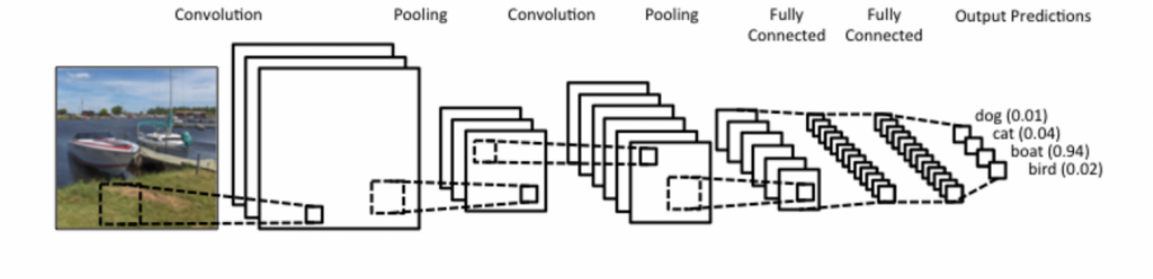


Figure 3.1: LeNet5-architecture[32]

The convolutional neural network (CNN) architecture has proven to be a efficient and effective tool for image detection and natural language processing. The sharing weights mechanism of a CNN vastly reduces the magnitude of the model and hence accelerate computation. The convolutional filter is a two-dimension proceptron that is able to extract the spatial invariance of the input. A CNN usually consists of

three types of layer as shown in Figure 3.1: convolutional layer, pooling layer, and fully-connected layer.

The convolutional layer consists of a rectangular grid of neurons and requires that the input be in the shape of a rectangle. Each neuron takes a rectangular section of input then performs a dot product with the weights of that same section as shown in Figure 3.2. That is, the convolutional layer is an image convolution of the input, where the convolutional filter (weights) are shared layer-wise. A single neuron x_{ij}^{l} is computed as shown in Equation 3.2, where the size of weights w is $m \times m$ and y is the input.

$$x_{ij}^{l} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} w_{ab} y_{(i+a)(j+b)}^{l-1} \tag{3.2}$$

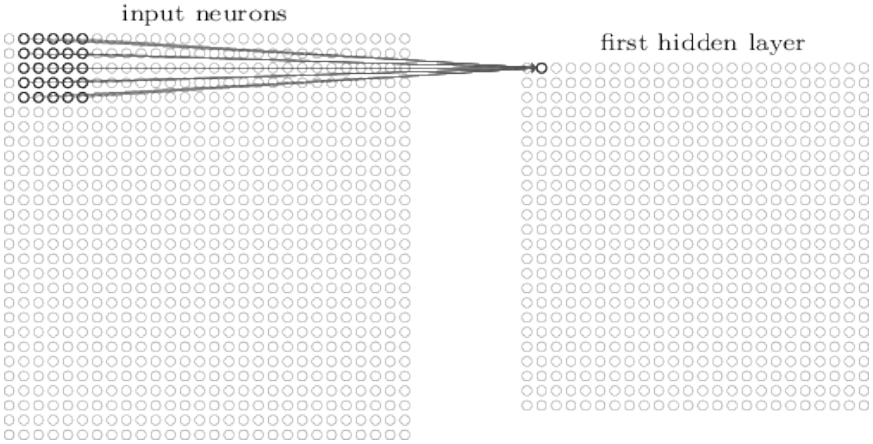


Figure 3.2: The computation convolution for a single neuron [39]

After a convolutional layer, there may be a pooling layer, which divides the input from a convolutional layer into small square blocks and subsamples each small block to one value as shown in Figure 3.3. There are several types of pooling layers, such as average pooling (outputs the average value of the small block); and max pooling (outputs the maximum value of the small block). Usually, a pooling layer

does not contain trainable parameters.

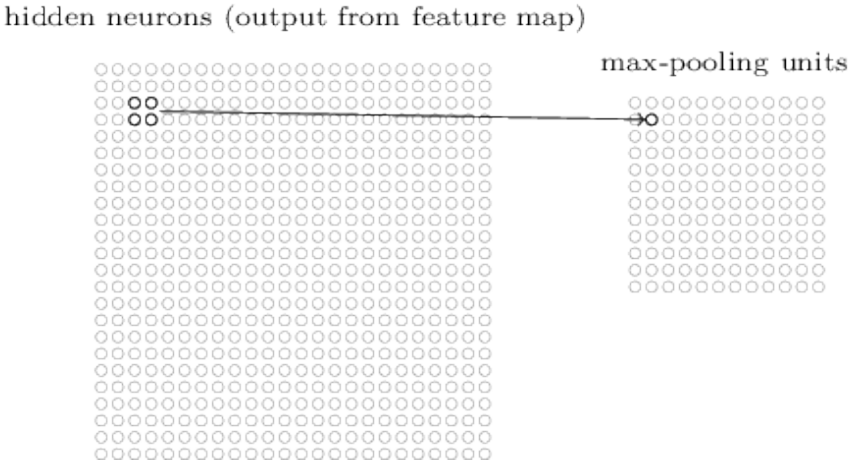


Figure 3.3: The computation of max pooling for a single neuron [39]

Fully-connected layers usually follow after several convolutional layers and pooling layers. There may be several fully-connected layers with the last layer predicting an output.

3.3 Recurrent Neural Networks

Different from feed-forward neural networks, recurrent neural networks (RNNs) are designed for processing sequential information, such as machine translation, natural language processing (NLP), video, etc. RNNs usually receive two inputs, data and the previous neuron states, which impose the RNNs the capability to grasp the relationship between each time step. RNNs can be thought of as several copies of the same network, and each passing message to its successor. A unrolled RNN is shown in the Figure 3.4 Since RNNs are usually used to process time series data, the depth of the RNN is dependent on the data, which prompts the problem of vanishing gradient or exploding gradient. LSTM and GRU are two typical RNN algorithms used to avoid gradient

vanishing or gradient exploding.

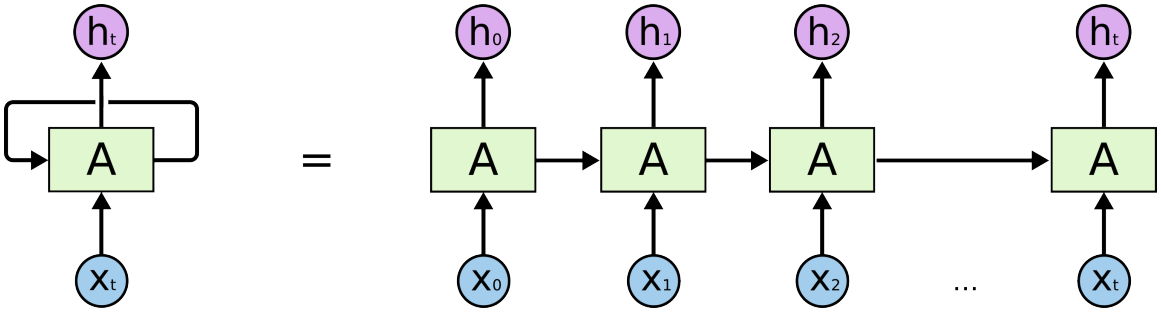


Figure 3.4: An unrolled RNN. **Left:** a RNN cell. **Right:** unfolded RNN cells in time sequence [1]

The Long Short Term Memory network (LSTM) is a famous architecture of RNN, introduced by [22]. LSTM shows effectiveness in handling backpropagation through time (BPTT). LSTM can be formulated as follows: $\sigma(\cdot)$ is the sigmoid activation function, $\mathbf{W}$ and $\mathbf{b}$ are the weights and biases of each gate repectively, $\mathbf{o}$ is the output and $\mathbf{h}$ is the internal state. Equation 3.3 is the forget gate, which determines the states the LSTM should forget when receiving new data. Equation 3.4 is the input gate, which determines the state used for updates. Equation 3.5 is the candidate value that can be added to the state. Equation 3.6 updates the cell state by adding the previous state and current state. Equation 3.7 and 3.8 are the output and cell state, respectively.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.3}$$

$$i_t = \sigma(W_I \cdot [h_{t_t}, x_t] + b_i) \tag{3.4}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_t - 1, x_t] + b_C) \tag{3.5}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.6}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.7}$$

$$h_t = o_t * tanh(C_t) \tag{3.8}$$

The Gated Recurrent Unit (GRU) [5] is another popular architecture for RNNs. GRU merges the forget gate and input gate into an "update" gate after which the output and cell states are combined to a single hidden state. GRU is simpler than standard LSTM and shows comparative performance for many cases. Because of the simplicity, GRU is more computationally efficient than LSTM and has been used broadly. GRU can be formulated as follows in Equations 3.9, 3.10, and 3.11: z_t is the update gate vector; r_t is the reset gate vector; and h_t is the output vector.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{3.9}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{3.10}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{3.11}$$

3.4 Backpropagation

Backpropagation [40] is an algorithm used in neural networks to calculate the error contribution of each neuron after a batch of data it processed. Given a dataset, a neural network, and an error function, the backpropagation algorithm calculates the gradients of the error function with respect to the trainable parameters of the neural network. The word "back" stems from the fact that the computation of the

gradient proceeds backwards from the last layer to the first layer of the neural network. Partial gradients of the current layer are reused for the calculation of previous layer gradients. The backpropagation algorithm allows the for flow of error information spread effectively and is widely used with deep learning in many applications, including image recognition, speech recognition, and natural language processing.

The derivation of the backpropagation starts from gradient descent. Training a neural network using gradient descent requires the calculation of the gradient of the error function $E(x_i, y_i, \theta)$, where x_i is the feature vector, y_i is the label and trainable parameters θ include the weights w_{ij}^k and biases b_i^k. We update θ using Equation 3.12, where α is the learning rate and θ^t denotes the trainable paramter in t^{th} iteration.

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(x_i, y_i, \theta^t)}{\partial \theta} \tag{3.12}$$

By applying the chain rule to the error function partial derivative, we are able to deliver error information to the previous layers. Equation 3.13 shows how to calculate the derivative with respect to the weights of the second last layer, where a_j^k is the activation of node j in layer k.

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k} \tag{3.13}$$

3.5 Attention Mechanism

The attention mechanism in neural networks is loosely inspired by the visual attention mechanism found in humans. The attention mechanism allows neural networks to focus on certain parts of objects that are important in the current time step while temporally ignoring other parts. It can be applied to the domain of

NLP [28, 4] and image processing [58, 37]. For example, in machine translation, one of the most popular architectures is the sequence to sequence model [50], which can be decomposed into an encoder and decoder. The encoder encodes the source sentence into an embedding vector. The decoder generates the target language sentence based on the embedding vector. One typical problem occurs when generating long sentences, the compressed vector shows limited information to organize target sentences. Attention is introduced to ease this problem. In each recurrent time-step, attention would peek each word of the source sentence attentively to provide more information for the decoder as shown in Figure 3.5.

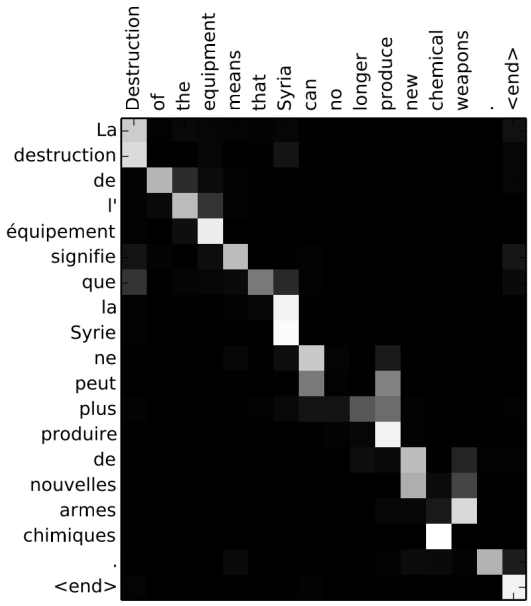


Figure 3.5: An alignment found by RNNsearch-50. As translation proceeding, the network attends on one or two words in another language while translating a single word [4].

The attention mechanism can be formulated as follows. The decoder state s_i is computed by Equation 3.14, where $f(\cdot)$ is a RNN, s_{i-1} is the previous decoder state vector, y_{i-1} is the previous neuron output vector and c_i is the current context vector.

The context vector c_i depends on a sequence of *annotations* h_j, where h_j contains valuable information about the corresponding j^{th} input. c_i is computed by Equation 3.15, where α_{ij} are the weights of *annotations* h_j. α_{ij} is calculated using Equation 3.16 and 3.17, where 3.16 is a softmax function over an *alignment model* e_{ij}, which scores how well the inputs around position j match the output at postion i. $a(\cdot)$ is a parametrized function approximator, such as neuron network [4].

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{3.14}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.15}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{3.16}$$

$$e_{ij} = a(s_{i-1}, h_j) \tag{3.17}$$

3.6 Generative Models

According to Bayes' rule, a conditional distribution could be represented as Equation 3.18, in which the prior $P(x)$ could be inferred from data distribution.

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{3.18}$$

The discriminative models, also known as conditional models, are used to evaluate the dependence of the unobserved y given the observed x. Within the probabilistic domain, discriminative models could be modeled as conditional distribution $P(y|x)$. Generative models, opposite to discriminative models, directly model the joint distribution $P(x, y)$, which could either be used as an intermediate step for calculating the conditional

probability $P(y|x)$ or model the data directly. Generative models are able to model all variables of the model, whereas discriminative models only predict the target variables given the observed data.

3.6.1 Variational Inference

Variational inference is a general term for algorithms that cast posterior inference as optimization problems [21, 25, 55]. Variational inference can be divided into two steps: assuming a family of distributions $q(z;\lambda)$ and approximating $q(z;\lambda)$ to the posterior by optimizing over its parameter λ. This idea converts the problem of calculating the posterior $p(z|x)$ into an optimization problem that minimizes a divergence of $p(z|x)$ and $q(z;\lambda)$.

One branch of variational inference minimizes the Kullback-Leibler (KL) divergence from $q(z;\lambda)$ to $p(z|x)$, which can be formulated as Equation 3.19. KL divergence is a non-symmetric, information theoretical measurement of similarity between two probability distributions [21, 25, 55].

$$\begin{aligned} \lambda^* &= \arg\min_{\lambda} \mathrm{KL}(q(z;\lambda)||p(z|x)) \\ &= \arg\min_{\lambda} \mathbb{E}_{q(z;\lambda)}(\log q(z;\lambda) - \log p(z|x)) \end{aligned} \tag{3.19}$$

In Equation 3.19, the posterior $p(z|x)$ is intractable, but by applying Bayes' rule, we could have the following property in Equation 3.20,

$$\begin{aligned} \log p(x) =& \mathrm{KL}(q(z;\lambda)||p(z|x)) \\ &+ \mathbb{E}_{q(z;\lambda)}(\log q(x,z) - \log p(z;x)) \end{aligned} \tag{3.20}$$

where $p(x) = \int p(x,z)\mathrm{d}z$ is the model evidence. This evidence is a constant with regard to the variational parameters λ, hence we could convert optimizing Equation

3.19 into maximizing the Evidence Lower BOund (ELBO) as shown in the Equation 3.21, where both $p(x, z)$ and $q(z; \lambda)$ are tractable.

$$\text{ELBO}(\lambda) = \mathbb{E}_{q(z;\lambda)}(\log q(x, z) - \log p(z; x)) \tag{3.21}$$

The optimization problem becomes Equation 3.22

$$\lambda^* = \arg\max_{\lambda} \text{ELBO}(\lambda) \tag{3.22}$$

By splitting the ELBO, we have Equation 3.23.

$$\text{ELBO}(\lambda) = \mathbb{E}_{q(z;\lambda)}\log q(x, z) - \mathbb{E}_{q(z;\lambda)}\log p(z; x)) \tag{3.23}$$

The first term in Equation 3.23, $\mathbb{E}_{q(z;\lambda)}\log q(x, z)$, is an energy that prompts q to concentrate on where the model places high probability. The second term, $-\mathbb{E}_{q(z;\lambda)}\log p(z; x)$, stands for the entropy of q, which prompts q to spread probability mass to avoid focus on one position.

3.7 Deep Generative Models

3.7.1 Variational Autoencoders

Variational Autoencoders (VAEs) [26] place variational inference in the architecture of autoencoders, in which the encoder maps the input into a posterior distribution over a latent variable **z** and the decoder is able to map the original data distribution given any arbitrary latent variable.

The loss function of VAE is the negative log-likelihood with a regularizer as

shown in Equation 3.24,

$$L(\theta, \phi) = -E_{z \sim q(z|x_i)}\Big[\log p_\phi(x_i|z)\Big] + \mathrm{KL}(q_\theta(z|x_i)||p(z)) \tag{3.24}$$

where the first term is the reconstruction loss, which encourages the decoder to learn to generate the data properly and the second term is the KL divergence, which measures the divergence when using q to represent p.

3.7.2 Deep Recurrent Attentive Writer

Instead of generating images at once, [17] introduced Deep Recurrent Attentive Writer (DRAW) based on VAE. DRAW uses a sequence of modifications to generate images step by step. The intuition behind this is that when artists create paintings, they do not actually finish their work immediately, but they split the process into a sequence of brush strokes that finally makes a wonderful painting. [17] tries to mimic this behavior by using RNNs for both the encoder and the decoder to read and write part of image from the original image and canvas respectively. See Appendix A for more detail.

Attend, Infer, Repeat (AIR) [9] extends DRAW by combining a group of latent variables that are able to determine the existence of additional objectives. [43] is an extension work based on the architecture of DRAW, which uses a spatial transfer network [24] as attention instead of selective-based attention.

3.7.3 Generative Adversarial Networks

Generative Adversarial Networks [14] consist of two networks: the Generator and the Discriminator. The Generator is a neural network that given some random noise as input, data is generated by a deterministic function. The Discriminator is

another neural network that learns to distinguish the samples from the generator and training data. Both networks are trained simultaneously. During training, the Generator is learning to generate more realistic data while the Discriminator is learning to be able to distinguish generated data from real data. The two networks are essentially playing a game. In the ideal situation, the Generator should be able to generate indistinguishable data while the Discriminator is unable to distinguish whether the generated data is real or not.

The formal definition of the loss function for the Discriminator can be formulated as follows,

$$L_{\theta_d} = \frac{1}{m}\sum_{i=1}^{m}\left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))\right] \tag{3.25}$$

where $D(\cdot)$ and $G(\cdot)$ are the Discriminator and the Generator, respectively. By applying the gradient ascent algorithm on this loss function, we intend to optimize the probability of the real data with high value and the probability of the generated data with poor value. Note that the gradient in Equation 3.25 only applies to the Discriminator.

The formal definition of the loss function for the Generator can be formulated as follows,

$$L_{\theta_g} = \frac{1}{m}\sum_{i=1}^{m}\left[\log(1 - D(G(z^{(i)})))\right] \tag{3.26}$$

where by applying the gradient descent algorithm on the loss function, we intend to optimize the probability of the generated data G(z). Note that the gradient in Equation 3.26 only applies to the Generator.

3.8 Meta Learning

Researchers from the Deep Learning community have investigated meta-learning [46, 47] and metrics learning [3] for a diverse set of neural network models. Meta-learning refers to a scenario that an agent learns at two different time scales, acquiring information on similarities and differences between tasks. Metric-learning is intended to learn a distance function that computes how similar or related two objects are.

3.8.1 Memory-Augmented Neural Networks

Memory-Augmented Neural Networks (MANN) [45], which incorporate the Neural Turing Machine [16] with meta-learning using a content based addressed method, is trained to learn how to store and retrieve memories for classification tasks. However, the results show the performance degrades once the number of patterns exceeds memory capacity.

3.8.2 Siamese Networks

Siamese Networks [27] use two deep convolution neural networks to embed pair-wise dataset and decide if they come from the same set via metric learning. When doing one-shot classification, each image of the test set is compared with the support set. Then the network would predict the probability of being the same category.

The loss function of Siamese Networks can be formulated as a regularized cross-entropy objective seen Equation 3.27. Where $y(x_1, x_2) = 1$ when x_1 and x_2 are from the same category and $y(x_1, x_2) = 0$ otherwise. $\mathbf{p}(\cdot)$ is the Siamese Network, which takes two images as input and generates the probability of the two images being

the same category.

$$\begin{aligned} L(x_1, x_2) =& y(x_1, x_2)\log \mathbf{p}(x_1, x_2) \\ &+ (1 - y(x_1, x_2))\log(1 - \mathbf{p}(x_1, x_2)) + \lambda^T |w|^2 \end{aligned} \tag{3.27}$$

3.8.3 Matching Networks

Matching Networks [54] as shown in the Figure 3.6, extend the meta-learning paradigm by developing a differentiable nearest neighbor loss via cosine similarity. This approach is a form of meta-learning because the neural networks learn from a given support set to minimize a loss over a batch. Matching Networks obtain results compared with human performance of 95.2% on Omniglot. Matching Networks is closely related to Siamese Networks [27], but Matching Networks uses asymmetric distance functions. That is, the test set and the support set are mapped to knowledge vectors via different functions.

The Matching Networks loss function is shown in Equation 3.28, where T is the whole dataset, label set L is sampled from T, and support set S and batch B are sampled using L. The Matching Networks are trained to minimise the error predicting the labels in batch B conditioned on support set S.

$$\theta = \arg\max_{\theta} E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_\theta(y|x, S)\right]\right] \tag{3.28}$$

3.9 One-shot Learning

A wide variety of works have suggested the one-shot classification approach by transferring the abstraction of old/prior knowledge.

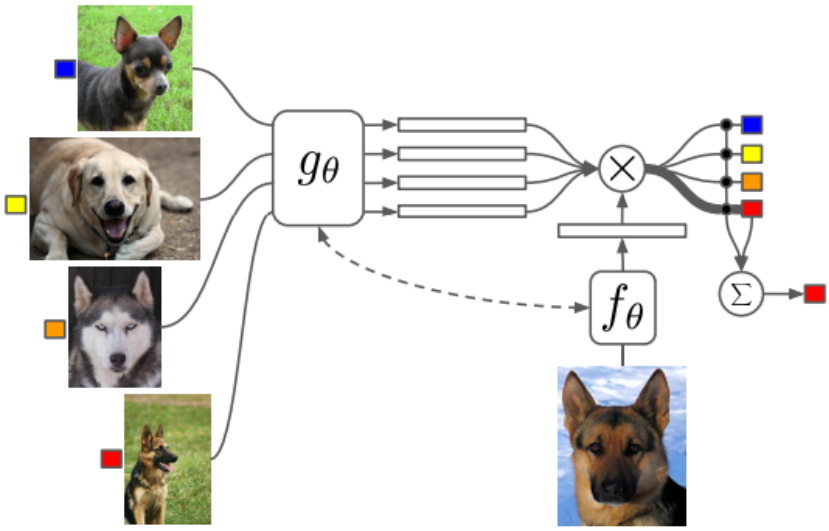


Figure 3.6: Architecture of Matching Networks. Support set (left four images) is fed into $g\theta$, which generates a feature vector for each case; the testing set (right image) is fed into $f\theta$, which generates a feature vector. Cosine similarity is used to determine which category the test belongs [54].

3.9.1 Hierarchical Bayesian Program Learning

[29, 31, 30] introduced Hierarchical Bayesian Program Learn-ing (HBPL) that utilizes the principles of compositionality and causality to build generative models of handwritten characters, achieving state-of-the-art performance. In the paper, the author decomposes the drawing characters to handcrafted stroke, sub-stroke and relations, then generates the joint distribution by hierarchy. However, the model shows two primary limitations: 1) the model heavily depends on handcrafted domain knowledge, which is not easily portable to other applications, 2) the posterior inference under HBPL involves intractable searching space, leading to integral problems.

3.9.2 Few-shot Learning with Meta-learner

[42] proposed a LSTM based few-shot learning framework to learn the optimization of parameters of another training dataset. In this paper, LSTM acts as a meta-learner, and the update parameters of the base learner are directly modeled using the memory cells in the second layer. In doing so, the authors noticed the similarity between the update rules of the LSTM memory cell and gradients (base-learner) of gradients (meta-learner). LSTM meta-learner is updated based on base learner's predicted loss of the last test batch. One drawback of this method is that if the number of parameters of the base-learner is too large, the model would become cumbersome.

3.10 Relation to Hierarchical Bayesian Program Learning(HBPL)

As many world scenes are inherently decomposed as hierarchical structures, the natural way to analyze the compositional relation is to search the structure description of patterns (e.g. Hierarchical Bayesian Program Learning (HBPL) [30]). In HBPL, the generation step could be decomposed into two levels (types and tokens) by joint distribution: type-level distinguishes between different classes, whereas token-level specifies different styles in one class. Obviously, the type-level provide structural information for generating different classes. Here we show the relation between the DRAW and HBPL model, then provide a recipe for inference procedure.

The internal mechanics of the DRAW [17] is defined in Appendix A. Recall the HBPL at a character type [30] $\psi = \kappa, S, R$, where $S = S_1, ..., S_n$ represent strokes, R is the spatial relations between strokes, and κ is the number of strokes in the set.

The joint distribution of type-level is:

$$P(\psi) = P(\kappa)\prod_{i=1}^{\kappa} P(S_i)P(R_i|S_1, ..., S_{i-1}) \tag{3.29}$$

and the token-level distribution is:

$$P(\theta^{(m)}|\psi) = P(L^{(m)}|\theta^{(m)}, \psi)\prod_{i} P(\hat{\theta}_i^{(m)}|\hat{\theta}_i)P(\theta'^{(m)}) \tag{3.30}$$

In the type-level generative process (in Equation 3.29), $P(\kappa)$ represents the generation step size of $RNN^{dec}(\cdot)$ in the DRAW model, which is fixed and pre-defined. Intuitively, for i^{th} stroke iteration, the term of $P(S_i)P(R_i|S_1, ..., S_{i-1})$ is one step propagation of Long-Short Term Memory (LSTM) [22] in DRAW. It is then straightforward to unfold $\prod_{i=1}^{\kappa} P(S_i)P(R_i|S_1, ..., S_{i-1})$ as RNN^{dec} in κ steps. It should be noted that RNN does not explicitly define the stroke, but learns to represent it in a smoother way.

The same principle is applied to pen trajectories in Equation 3.30. $\hat{\theta}^{(m)}$ and $\theta'^{(m)}$ represent the parameters of trajectories and affine transformations, respectively, both of which are modulated in the DRAW attention model. We should point out that the original selective attention in DRAW lacks ability for affine transformation, and thereby shows less diversity compared with spatial transformer network [24], we provide details of the effects of different attention strategy in the Results section.

Thus, the generative part of the DRAW model could be interpreted relative to HBPL: While HBPL explicitly modulates the generative process in a two-level hierarchical manner, DRAW encodes the hierarchy process implicitly in fixed steps of RNN with attention. Note that this is not a rigorous mathematics proof demonstrating that sequential generative model is a differentiable version of HBPL. Rather we show that different components of the DRAW-like model are comparable with components

of HBPL that compute the same tasks.

In HBPL of one-shot classification, in order to compute posterior predictive distribution for k test images $I^{(T)}$ given a $N \times k$ training image $I^{(c)}$, the solution can be approximated by:

$$\arg\max_{c} \log \sum_{i=1}^{K} \omega_i \max_{\theta^{(T)}} P(I^{(T)}|\theta^{(T)}) \frac{1}{N} \sum_{j=1}^{N} P(\theta^{(T)}|\psi^{[ij]}) \tag{3.31}$$

Equation 3.31 involves $P(I^{(T)}|\theta^{(T)})$ (generates test image conditional on given token) and $P(\theta^{(T)}|\psi^{[ij]})$ (indicates the token conditional on the structure distribution). The Equation 10 could be explained as searching top-K parses ($K = 5$) of image $I^{(c)}$, optimizing variable $\theta^{(T)}$ to fit the image $I^{(T)}$. This intuition perfectly matches the conditional DRAW generation as Equation from 4 to 7. Intuitively, conditional DRAW model can be directly mapped to an estimation step that evaluates type-level variability for each parse, but the current DRAW model lacks the optimization for variable $\theta^{(T)}$, and this is missing in one-shot classification.

3.11 Summary

In this chapter, we first introduced several concepts in machine learning, such as MLP, RNN, backpropagation, variational inference, etc., which are the "bricks" of this research. We futher introduced deep generative model and meta-learning, which are the main concepts to support each module of the proposed framework respectively. Lastly, several recent one-shot learning frameworks are introduced.

Chapter 4

Results

In this chapter, we evaluate our model on the Omniglot dataset using different hyper-parameters and demonstrate that our method produces state-of-the-art results for one-shot classification tasks. Additionally, we illustrate that each module is essential to maximize final performance.

4.1 The Omniglot Dataset

We evaluate our model using the Omniglot dataset [30], which consists of 1623 classes of characters in the form of 105x105 gray scale images with each class containing 20 examples. The dataset is randomly permuted and split into 30 background sets and 20 evaluation sets, each paired with labels. We set up our experiments exactly as [30], which uses the background set (964 classes) to pre-train the DRAW model, with the evaluation set being used for testing. To reduce the computation time, we downscaled the images to 28×28.

Due to the variety of related works using different experimental configurations, it would be difficult to compare our model's performance using the same baseline.

Table 4.1: One-shot learning environment compared with previous works

Configuration	MANN[1] [45]	MN [2] [54]	Siamese Network [27]	HBPL [30]	This Work
BG vs Eval	1200 vs 423	1200 vs 423	1200 vs 423 [3]	964 vs 659	964 vs 659
Data manipulation?	rotation	rotation	rotation+affine	no	no
Input size	20×20	28×28	105×105	105×105	28×28
No domain knowledge	√	√	√	×	√

We chose the most challenging experimental environment, which is used by [30]. In our experimental environment, 60% of the dataset are chosen for training and 40% of the data are used for evalution; no data argumentation and domain knowledge are used; the image size is 28×28. We have compared different environment setups in Table 4.1. We are interested in 1-shot 20-way classification, which is arguably the most challenging environment. All accuracy results reported in the subsequent sections are for 1-shot 20-way classification.

4.2 Model Hyperparameters

For the DRAW model in our experiments, we use 512 LSTM hidden units for the encoder and decoder, respectively. We set the number of time steps as $T = 64$, and we use a 10×10 kernel as the spatial transformer for both read and write attention. Alternatively, we tried selective attention for read with sizes varying from 2-10. We set the latent variables z_t to be 32-dimensional Gaussian distributions and trained it for 100K iterations, where the batch size is 100 randomly sampled images from the background set (sampled with replacements). We use Stochastic Gradient Descent (SGD) as an optimizer with the learning rate set to $5e - 4$. To combat overfitting, we

[1]MANN represent as Memory-Augmented Neural Networks

[2]MN represent as Matching Networks

[3]Original paper shows as 40 vs 10 alphabet categories

apply both dropout [49] and batch normalization [23], which both help stabilize and accelerate the learning process.

The candidate network consists of a 2-layer, fully connected neural network with dropout enabled for the 1st and 2nd layers (dropout rates are 0.3 and 0.5, respectively). For each test example, the candidate network receives 20 training samples, each from different classes (20-way one-shot learning), and the rest of the 380 samples are used for the validation set. The candidate network training procedure can be found in Chapter 2 Algorithm 2.

For the matching network, we build a 3-layer, fully connected neural network with $[500, 500, 500]$ neurons for each layer (appended with batch normalization) and using $Relu(\cdot)$ non-linearity. The matching network is trained by sampling background images from the test set and support set with tensor size equal to $100 \times n \times 512 \times 64$, where the batch size is 100, n is candidates number $n = 1, 3, 5, 20$, hidden vector size (512), and the DRAW time step is ($T = 64$). The matching network training procedure can be viewed in Chapter 2 Algorithm 1.

4.3 Performance Analysis

In the subsequent section, we demonstrate that DRAW offers impressive feature representations, and we show the performance effectiveness of each module.

Different attention models and window sizes. We also inquired if different attention mechanisms have any impact on the prediction. We first perform experiment of different attention models. As shown in Figure 4.1, selective attention outperforms the spatial transformer network, where we use the read window size set to be 2 (worst case of selective attention reader). Then we perform experiments under different window sizes using selective attention model. As shown in Figure 4.2, window

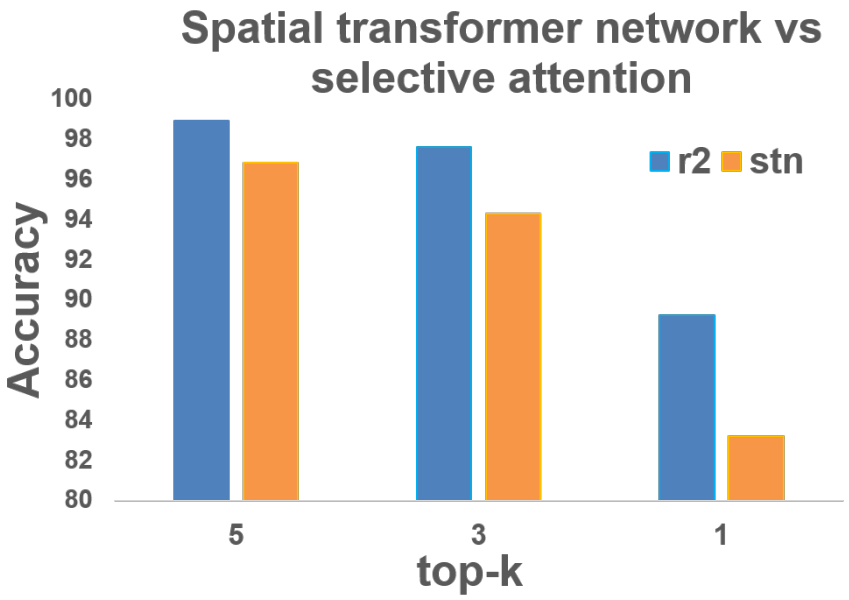


Figure 4.1: Prediction comparison between different attention models. Spatial transformer (orange) and selective attention (blue) with read window size set to 2, y-axis is the final prediction accuracy

size affects prediction accuracy, and the best performance is reached when the window size is 7.

Matching Network. We evaluate our model that directly pipes the DRAW encoder states to a 20-way matching network for classification without candidate networks. We found that the prediction accuracy of the 20-way matching network is 92.6%. This result is reasonable, since the original Matching Networks model (93.8%) is trained with data augmentation and more background samples (1200-class for [54], 964 class for this work), while our results only use structural feature extraction and metric-learning.

Necessity of each module. We show the necessity of each component of the proposed framework by performing experiments with different module configurations as shown in Figure 4.3.

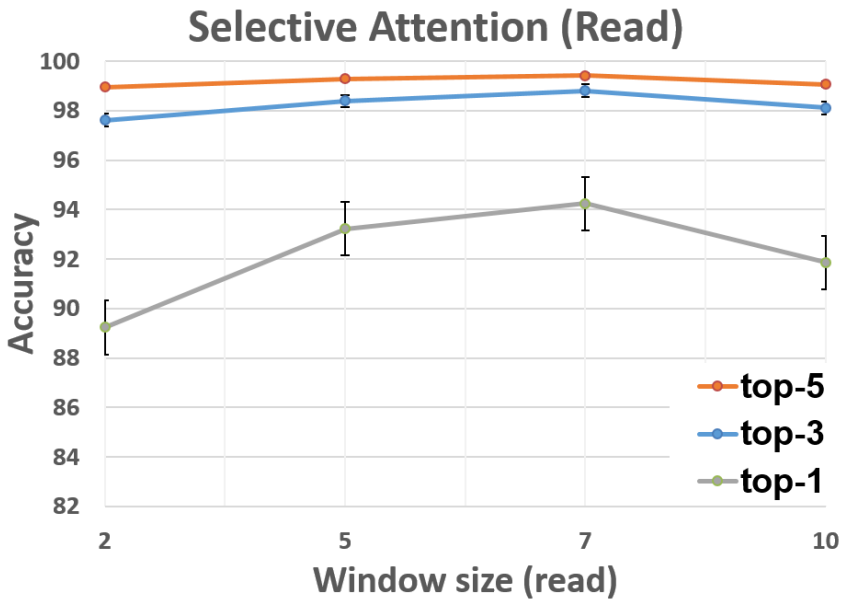


Figure 4.2: Performance under different windows size. x-axis is the read window size of the selective attention reader, y-axis is the final prediction accuracy.

Table 4.2: 1-nearest neighbor performance comparison by extracting structure knowledge from the DRAW

Input	Method	Accuracy
raw-pixel	KNN	21.7%
DRAW inference	KNN	73.95%

We show that DRAW offers excellent feature representation by comparing 1-nearest neighbor results on raw pixels and the encoder states of DRAW. Table 4.2 shows that a naive 1-nearest neighbor (euclidean distance) classifier provides significant improvement in prediction accuracy (from 21.7% to 73.95%), indicating that DRAW offers a better feature representation than raw pixels because of its use of both spatial and temporal information.

As shown in the middle of Figure 4.3, using a Matching Network or a candidate network is able to enhance the prediction accuracy. Moreover, using the candidate network slightly outperforms using the Matching Network. Note that when using a

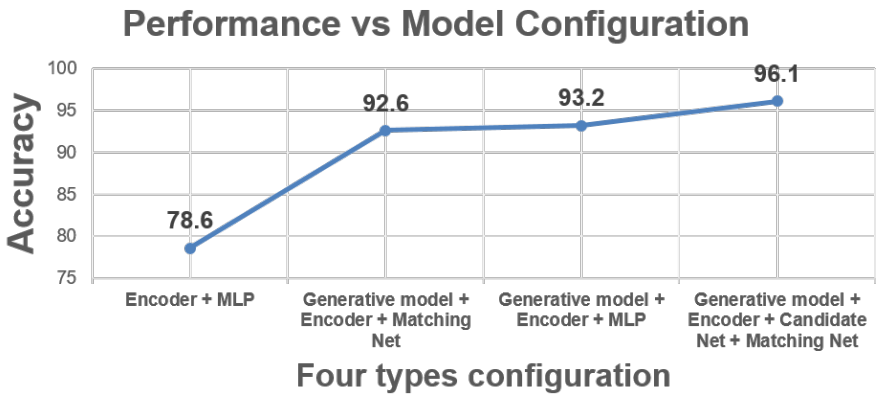


Figure 4.3: The performance comparison among four different types of model configuration; x-axis is different model types, y-axis is the prediction accuracy

Table 4.3: Classification accuracy comparison for different works on Omniglot dataset.

Model	20-way 1-shot accuracy
Our work	**96.1%**
Human	95.2%
HBPL	95.5%
Matching Networks	93.8%
MANN	82.8% [4]
Siamese Network	92.0%
DBM	72%
Affined Model	81.8%
1-nearest neighbor	21.7%

candidate network as the final classifier, it downgrades to a regular MLP.

We combine all modules together to the final framework, which outperforms all other configurations, indicating that a candidate network increases performance by narrowing down the overall number of possible selections, generating the best performance 96.1%.

The complete performance results are listed in Table 4.3, which indicates that our model achieves state-of-the-art performance, better than any previously proposed

[4] 5-way 1-shot classification results

Deep Learning work. Additionally, our testing configuration, as mentioned in Table 4.1, has the same configuration as HBPL, which is the most challenging environment for one-shot classification. It should be noted that the results from MANN do not use the same metrics as other works; their results shown 82.8% accuracy for 5-way one-shot classification, which is a comparatively easier problem than 20-way one-shot.

4.4 Visualization

To better understand the advantages of our model, we visualized the hidden layer of RNN (512×64 dimensional) in DRAW, by means of t-SNE [53], as shown in Figure 4.4 to 4.4. The original raw pixel data shown in Figure 4.4 is directly visualized using t-SNE (from 784 to 2-dim), where stars represent a one-shot training image and squares stand for one-shot testing image. Figure 4.5 and Figure 4.6 show the embeddings obtained from the DRAW hidden layer (encoder), where the Figure 4.5 is the proposed framework without using the generative process of DRAW and the Figure 4.5 is the proposed framework with the generative model to allow the candidate network to explore a variety of image samples. Finally, circles represent the generated sample set for the candidate network.

4.5 Summary

In this chapter, we introduced the dataset used in the experiments and provided the hyperparameters of the proposed model. A serials of perofance analysis are conducted to illustrate the necessities of each sub-module. We further proved the effectiveness of the proposed model by using t-SNE to visualize raw image pixel and hidden vector of DRAW encoder.

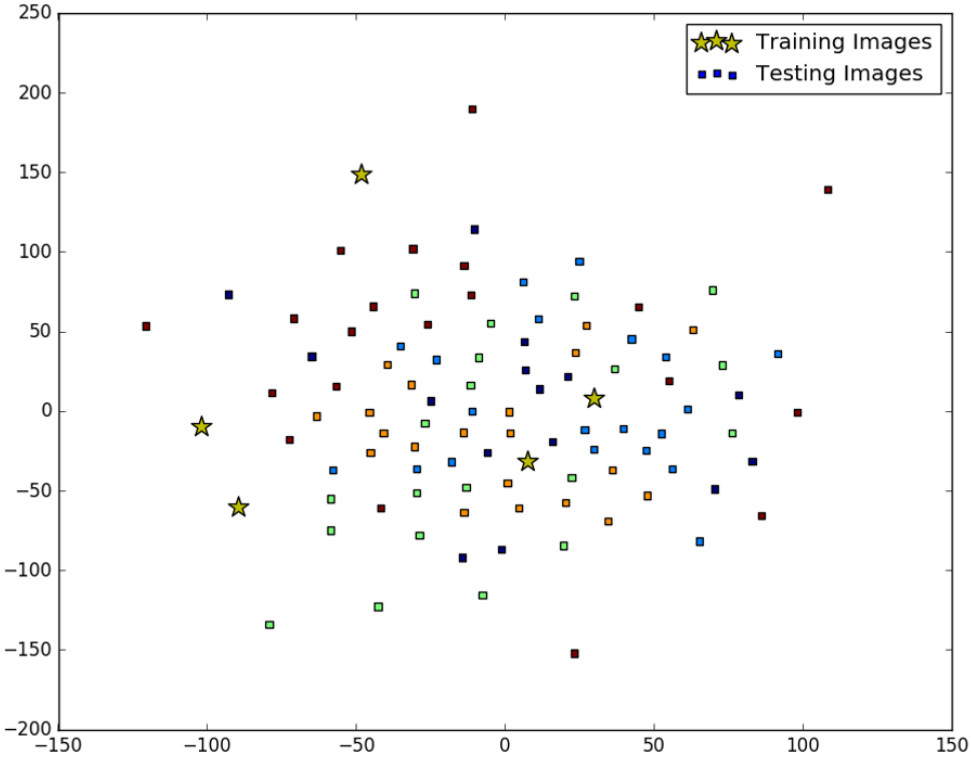


Figure 4.4: 2d-plot using t-SNE on raw image pixels. The original representation is not linearly separable. Star represents one-shot training images, squares are one-shot testing set (total 400 samples from evaluation set) and circles mean generated support set, the same below.

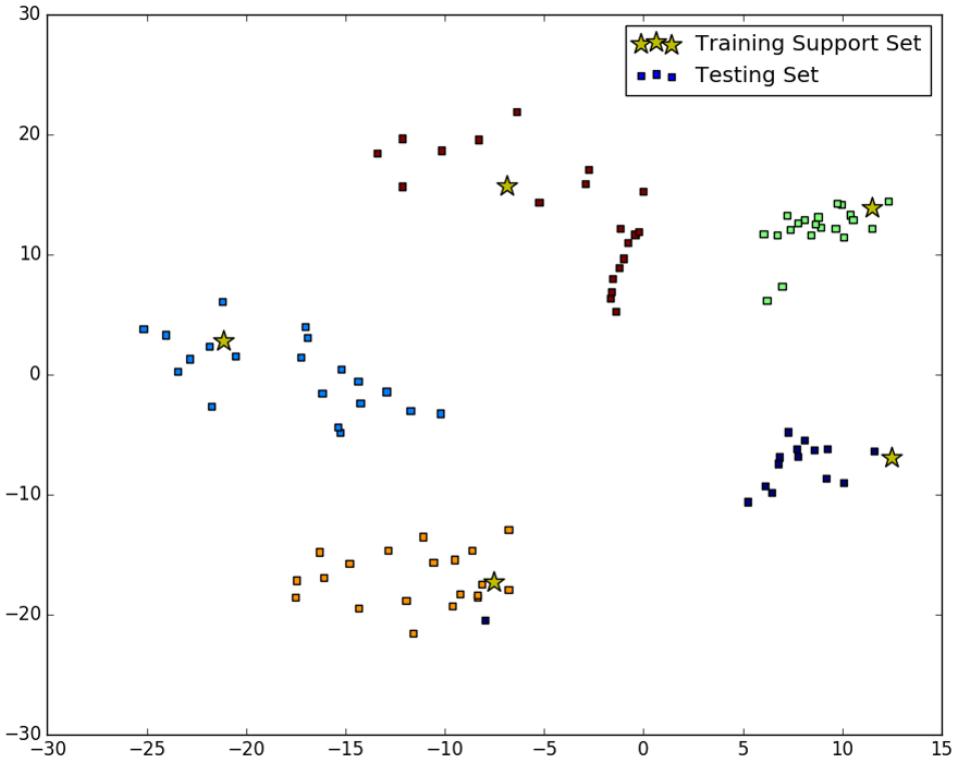


Figure 4.5: 2d-plot using t-SNE on embedded vectors from encoder without generative model.

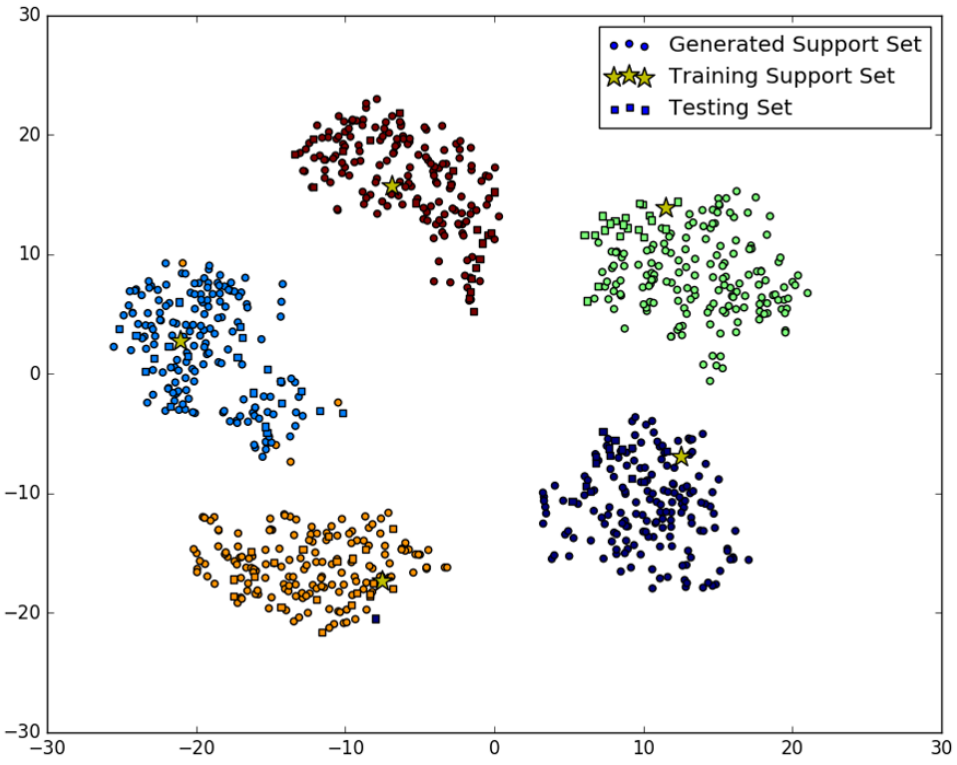


Figure 4.6: 2d-plot using t-SNE on embedded vectors from encoder with generative model.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this research, we introduce a novel framework to solve one-shot learning problems. We deploy a generative model (DRAW [17]) to learn and capture the "structure description" of images, so as to generate more data based on this learned knowledge when conditioning the one-shot training example. Specifically, the encoder of the DRAW is used to provide rich feature representations. A candidate network is proposed to narrow down the possible candidates for the metric learning. Lastly, the Matching Network is used to perform the metric learning. The framework implementation is based on TensorFlow [2]. We validate our work on the Omniglot dataset and achieve state-of-the-art accuracy without any prior domain knowledge. The main contributions can be summarized as following:

General framework for one-shot learning. We introduce a one-shot learning framework that takes advantage of a deep sequential generative model and existing works on one-shot classification. Each component could be replaced by other alternatives.

Sequential generative model in one-shot learning: We employ deep sequential generative model (DRAW in our case) into the one-shot classification problem. This work gives the initial interpretation of the relation between HBPL with sequential generative model, and we mimic the inference phase of HBPL and from this we derive one-shot inference in sequential generative model. We modify the original encoder-decoder setting as decoder-encoder shape, and we show that the encoder hidden state outputs good structure representations using the spatial invariant features (see Table 4.2, Figure 4.4, Figure 4.5, and Figure 4.6). We also show the importance of different attention effects (see Figure 4.2)

Candidate network: We introduce a candidate network that reduces the number of candidates, thus benefiting the meta-learning process. This result has been widely observed from related works [54, 45]. We use the candidate network as filter; then utilize the meta-learning to predict the final result.

5.2 Future Works

We have shown the effectiveness of using a generative model in one-shot learning. However, there are several aspects that could be improved.

One perspective that is currently missing is that the model is not trained end-to-end, rather each individual module is trained independently. The problem araised when one module settings is changed, the produced embedding distribution will changed, and the entire model needs to be re-optimized.

Another extension is to explore alternative generative models. The DRAW model has proved to effectively generate characters and plates images in [43]. However, for more complicated image generation, such as, cifar-100, ImageNet [44], the DRAW model does not show much advantage compared with other recently populated models,

such as the Generative Adversial Network(GAN)[14, 41, 7], which shows strong capability on image generation. Changing the generative model to GAN might extend our framework to a more general domain.

The candidate network used in this experiment is a simple multi-layer perceptron. Concatenation is used for embedded feature vectors, which is simple but brutal. The embedded vectors contain the temporal features of generating a image and RNNs are specialized to capture temporal information. RNNs might be an alternative structure of the candidate network. Due that we have show the effectiveness of MLP based candidate network, we would like to leave exploring the structure of candidate network as future work.

Appendices

Appendix A Deep Recurrent Attentive Writer

equation In this section, we briefly introduce the Deep Recurrent Attentive Writer (DRAW) [17]. DRAW is a sequential generative model using RNN as encoder and decoder, with the Gaussian grids as attention. Comparing with VAE [26] instead of generating the target in one-step, DRAW [17] generates the object in a iterative fashion, as shown in the Figure A.1.

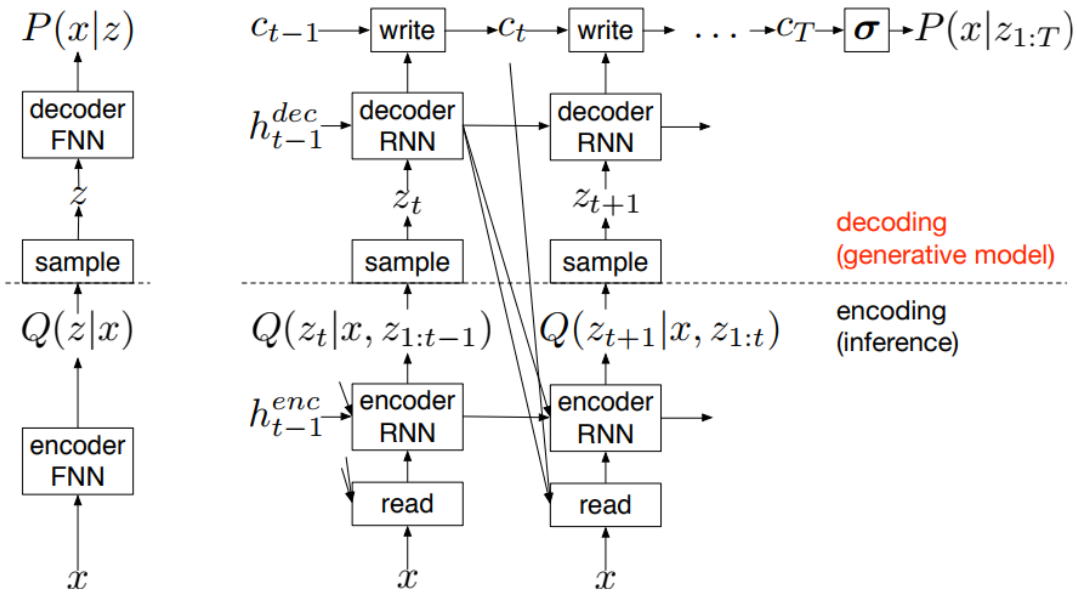


Figure A.1: Comparisons between VAE and DRAW: **Left:** Conventional Variational Autoencoder **Right:** DRAW networks

A.1 Network Architecture

At each time-step t, the encoder receives input from image x_t and previous decoder state vector h_{t-1}^{dec} as shown in Equation 1. Context vector v_t is computed by using the same *read* operator, where x' is the conditioned image as shown in

Equation 3. The encoder state vector is computed by Equation 2, where h_{t-1}^{dec}, r_t, and h_{t-1}^{enc} are concatenated and then passed into $RNN^{enc}(\cdot)$. h_t^{enc} is used to parametrize a distribution $Q(Zt|h_t^{enc})$ as shown in Equation 4.

In each time-step t, a sample z_t is drawn from the latent distribution $Q(Zt|h_t^{enc})$ and is passed as decoder input as shown in Equation 5. Instructed by the *write* operator, the output h_t^{dec} of decoder is added to a *canvas* matrix, which will be drawn to generate the final output image as shown in Equation 6. The final target is sample from observation function $f_o(.)$ that maps the final *canvas* hidden vector c_T into the parameters of the observation model, as shown in Equation 7.

$$r_t = read(x_t, h_{t-1}^{dec}; \theta_r) \tag{1}$$

$$v_t = read(x', h_{t-1}^{dec}; \theta_v) \tag{2}$$

$$h_t^{enc} = RNN^{enc}(h_{t-1}^{enc}, r_t, h_{t-1}^{dec}; \theta_e) \tag{3}$$

$$z_t \sim Q(Z_t|h_t^{enc}; \delta_z) \tag{4}$$

$$h_t^{dec} = RNN^{dec}(h_{t-1}^{dec}, z_t, v_t; \delta_d) \tag{5}$$

$$c_t = f_c(c_{t-1}, write(h_t^{dec}; \delta_w)) \tag{6}$$

$$x \sim p(x|f_o(c_T; \theta_o)) \tag{7}$$

A.2 Read and Write Operators

read and *write* operators are important concepts in DRAW and define the precise form of the read/write attention mechanism. Instead of passing the whole image to the encoder input or writing the decoder output to the whole *canvas*, DRAW uses an attention, called the selective attention model, by focusing only part of the object in each time-step and making DRAW applicable to large scale image generation as shown in Figure A.2.

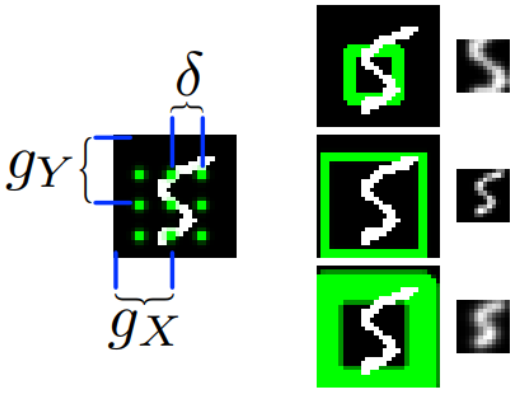


Figure A.2: Selective attention model. **Left:** a 3x3 grid posed on image defined by stride δ and location gX and gY **Right:** the green rectangle indicates the boundary and precision(σ) of the patch. The top patch has small δ and high σ; middle one has large δ and high σ; bottom has small δ and low σ.

The basic idea is to let the selective attention window focus on part of the image(patch). The patch could zoom in or out and be clear or blur as seen in Figure A.2. Based on the experiment in [17], at the very beginning, the patch could be large and blur, which gives the model a basic idea about the input image. Then the patch could be small and clear, so that the model could receive precise information about the input.

A.3 Training

The loss function of DRAW can be divided into two parts: *reconstructionloss* and *latentloss*. In the last time-step, the canvas c_T is evaluated by a model $D(X|c_T)$ of the input data. The *reconstructionloss* is defined as the negative log probability of x under D:

$$\mathcal{L}^x = -logD(x|c_T) \tag{8}$$

The *latent loss* is used to measure the latent distribution $Q(Z_t|h_t^{enc})$ and is defined as the summation of KL divergence of latent prior $P(Z_t)$ from $Q(Z_t|h_t^{enc})$:

$$\mathcal{L}^z = \sum_{t=1}^{T} KL(Q(Z_t|h_t^{enc})||P(Z_t)) \tag{9}$$

The total loss is the expectation of the summation of the *reconstruction loss* and the *latent loss*:

$$\mathcal{L} =< \mathcal{L}^x + \mathcal{L}^z >_{z\ Q} \tag{10}$$

Appendix B DRAW Image Generation

In this section, we show the process of how DRAW generates an image in eight time-steps.

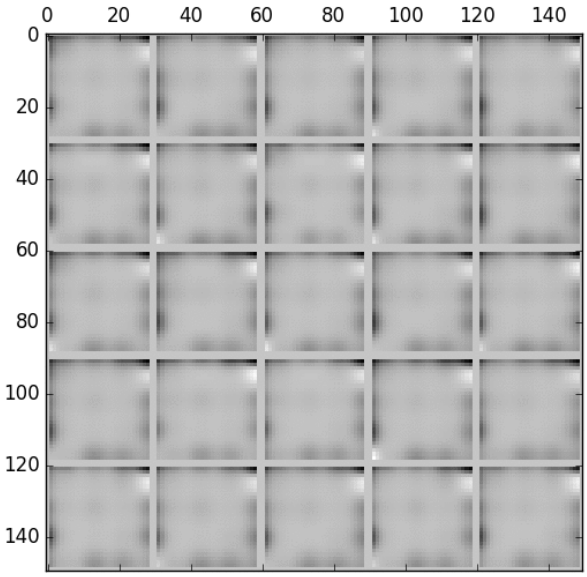


Figure B.1: DRAW model image generation time-step 1

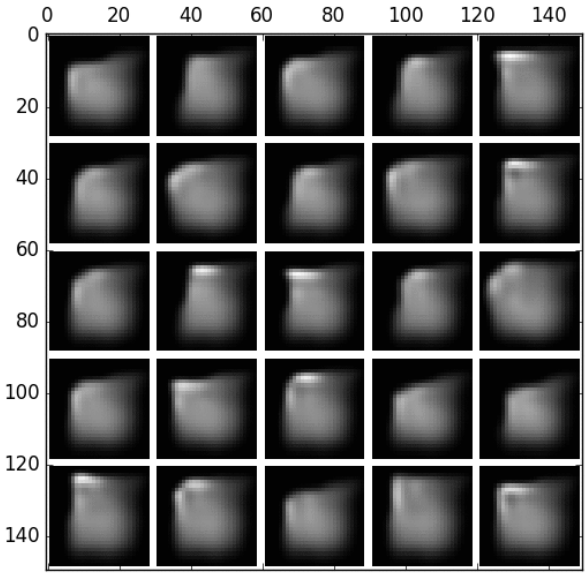


Figure B.2: DRAW model image generation time-step 2

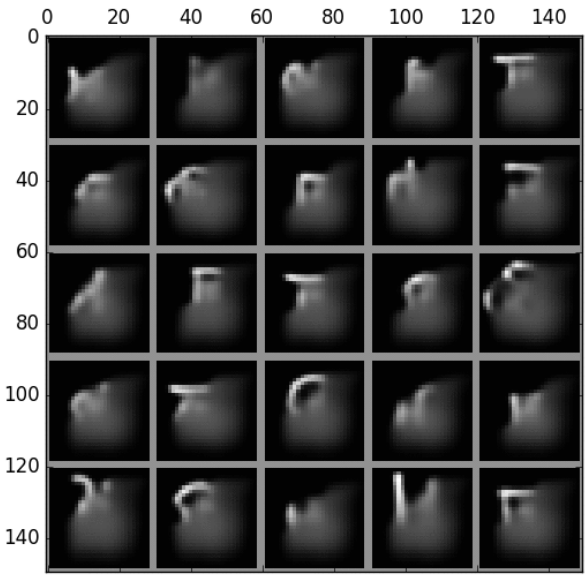


Figure B.3: DRAW model image generation time-step 3

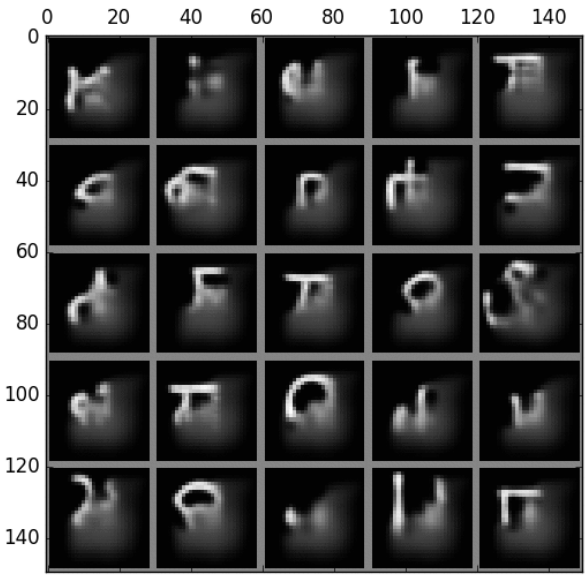


Figure B.4: DRAW model image generation time-step 4

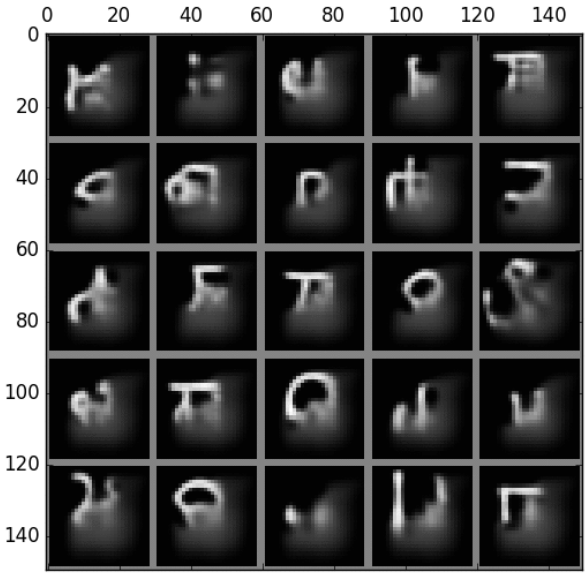


Figure B.5: DRAW model image generation time-step 5

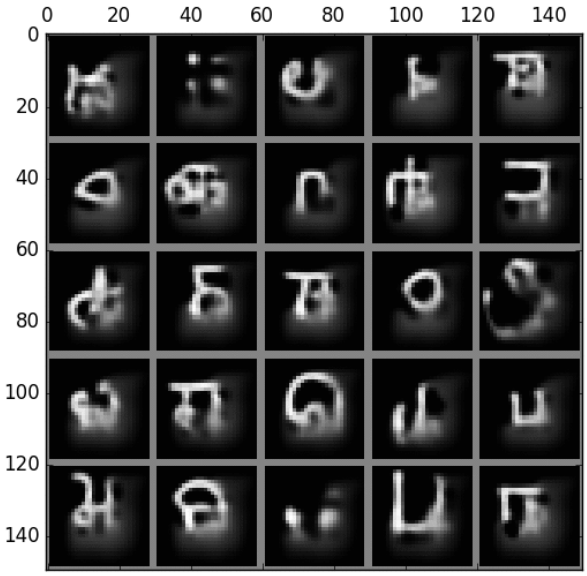


Figure B.6: DRAW model image generation time-step 6

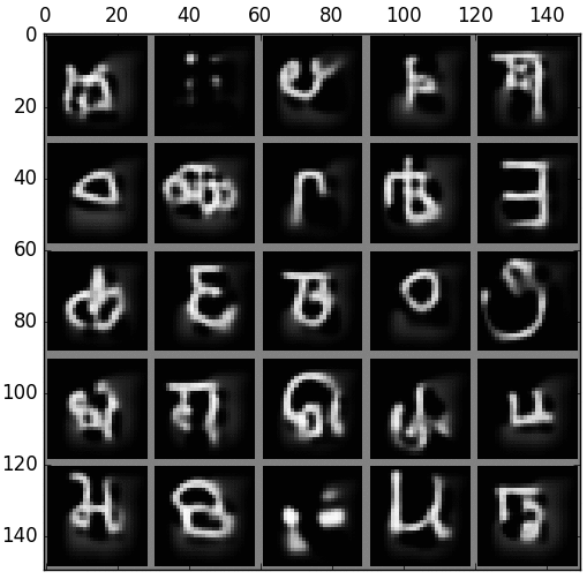


Figure B.7: DRAW model image generation time-step 7

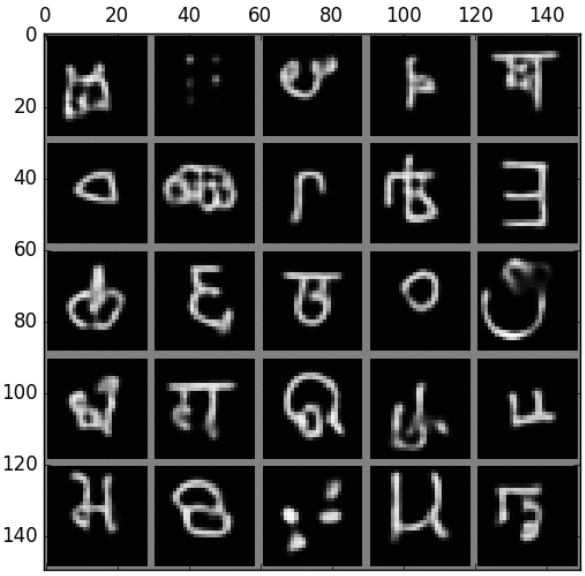


Figure B.8: DRAW model image generation time-step 8

Bibliography

[1] Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2017-07-23.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[7] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[8] SM Eslami, N Heess, and T Weber. Attend, infer, repeat: Fast scene understanding with generative models. arxiv preprint arxiv:..., 2016. *URL http://arxiv.org/abs/1603.08575*.

[9] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances In Neural Information Processing Systems*, pages 3225–3233, 2016.

[10] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[11] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[15] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[16] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[17] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[18] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[24] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[25] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[26] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[27] Gregory Koch. *Siamese neural networks for one-shot image recognition*. PhD thesis, University of Toronto, 2015.

[28] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

[29] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. In *CogSci*, volume 172, page 2, 2011.

[30] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[31] Brenden M Lake, Ruslan R Salakhutdinov, and Josh Tenenbaum. One-shot learning by inverting a compositional causal process. In *Advances in neural information processing systems*, pages 2526–2534, 2013.

[32] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 2015.

[33] Sang Wan Lee, John P O'Doherty, and Shinsuke Shimojo. Neural computations mediating one-shot learning in the human brain. *PLoS Biol*, 13(4):e1002137, 2015.

[34] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[36] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[37] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[38] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[39] Michael A Nielsen. Neural networks and deep learning, 2015.

[40] David C Plaut et al. Experiments on learning by back propagation. 1986.

[41] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[42] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[43] Danilo Rezende, Ivo Danihelka, Karol Gregor, Daan Wierstra, et al. One-shot generalization in deep generative models. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1521–1529, 2016.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[45] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

[46] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[47] Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Neural Networks, 1993., IEEE International Conference on*, pages 407–412. IEEE, 1993.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[50] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[52] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[53] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[54] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[55] Steve R Waterhouse, David MacKay, and Anthony J Robinson. Bayesian methods for mixtures of experts. In *Advances in neural information processing systems*, pages 351–357, 1996.

[56] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[57] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[58] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

[59] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[60] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.