# Adaptive Variational Memory Encoding for Recurrent Data Synthesis

**Sam Griesemer** SAMGRIESEMER@GMAIL.COM
Joplin High School

## Abstract

This work proposes a generative architecture that incorporates a natural approach to data synthesis. The Adaptive Variational Memory Encoding (AVME) model learns to represent data in an iterative manner during training, allowing for adaptive generation when sampling new data. One-shot approaches to data synthesis lack recurrent communication between dependencies during generation; this work attempts to lessen the likelihood of incoherent samples by giving the model time to iteratively update its output. Such a mechanism is largely inspired by the adaptive capabilities of the human mind for continuous predictive insight, resulting in a statistical model entwined with natural techniques. The AVME model is applied to both classical and conditional image generation, and shown to be effective at both tasks, achieving state of the art results on the MNIST dataset.

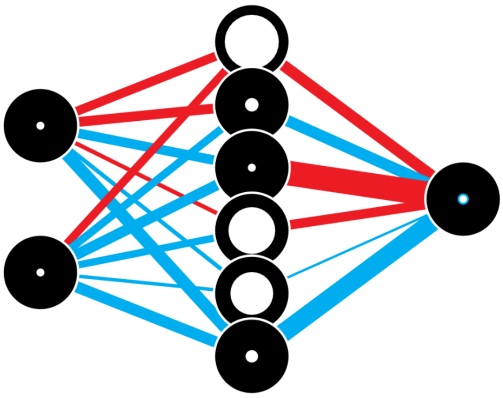

*Figure 1.* A simple diagram of neural network structure represented visually. In this particular example, the network has three layers, each containing vertically grouped nodes. The leftmost layer contains two input nodes, middle layer contains seven hidden nodes, and rightmost layer contains one output node. In this diagram, the red lines connecting neurons represents positive weights, while blues lines represent negative weights. (Figure created by Sam Griesemer).

## 1. Introduction

Machine learning is a field devoted to the development of algorithms that allow machines to learn from data. Methods in the field largely draw from biology, the human mind being the ultimate source of learning techniques. Endowing machines with the ability to learn from data permits use of computational "intelligence" to solve previously unapproachable problems. Such a task begins with allowing computers to model and understand data structure; yet, this is fundamentally difficult to accomplish accurately, as high dimensional data have complex interconnected dependencies that are not easily modeled.

There are two types of models typically assigned to the task of modeling data: discriminative and generative models. Discriminative models excel in classification tasks by finding boundaries to separate data classes. This type of model essentially represents the probability distribution $p(y|x)$, where $x$ are input data points and $y$ are class labels. This effectively assigns a probability distribution over predicted class labels given a certain input data point. One of the most common and effective discriminative models is the artificial neural network, a statistical architecture that utilizes fundamental neural mechanisms of the brain to approximate functional relations between variables.

A basic unit known as the *perceptron* mathematically models the functionality of a biological neuron. By definition, an artificial neural network is simply a network of perceptrons, similar to how to the human mind is a network of neurons. Through use of multiple layers of perceptrons, neural networks implement sequential abstraction, building increasingly complex behavior and representation from simple understandings.

Generative models take an inverse approach to the task of modeling data; they model $p(x|y)$. Rather than classifying or making predictions from input data, they do just the opposite: produce a probability distribution over potential inputs given a certain output value. This approach approximates the distribution from which the data are assumed to originate, providing a generalized interpretation of the

data's structure. It also provides a means for sampling new data points, a process termed data synthesis.

To understand why generative models are important, one must understand the difference between supervised and unsupervised learning tasks. For sake of simplicity, assume the statistical model used in the following examples is an artificial neural network (see Figure 1). Generally speaking, a supervised learning task consists of training a particular model on labeled data. This entails having some data $x$ and their paired output values $y$. The paired outputs $y$ can either be real values or discrete labels, depending on the task at hand. In the case of real valued outputs, the model is being trained to perform regression, as it outputs real valued predictions from real valued inputs. As for discrete output labels, the model is often being trained for classification, outputting a predictive class label based on the input. Neural networks perform well in both of these situations and are applied across a variety of domains.

Unfortunately, the vast majority of available raw information is unlabeled. Unless a large amount of time and effort is spent curating a dataset, we are generally left with some data $x$ and no corresponding outputs $y$ to train a model on. To understand more about unlabeled data, a model can learn to represent its inherent structure, and observations can be made accordingly. This is where generative models step in, as they can be trained to embed and represent input data. Additionally, certain models can then sample from this internally represented data distribution to obtain novel, realistic data samples.

With the capability to accurately generate novel data, many difficult real world problems can be attempted. One such problem is text to audio synthesis, or generating realistic sounding voices to automatically read books or articles. In this case, a generative model is able to look at a body of text and generate a corresponding audio file with voices reading the text input. The model could then be trained to generate a book on its own after gaining a structural understanding of books.

This paper introduces the Adaptive Variational Memory Encoding (AVME) model, capable of generating data recurrently, in hopes of incorporating a natural mechanism for its generative procedure. One can imagine how a human might draw (or "generate") an image an paper. First, the drawing is envisioned and planned out before anything is drawn. In reality, this plan can only be created to a certain extent. As soon as pen touches paper, the envisioned plan must change continually, at each moment reevaluating where and what should be drawn next. This inherently adaptive capability of the human mind allows for coherency in everyday life. Being able to continuously predict the next instance at every moment in time is part of what allows us to perform well in new situations and make informed decisions about the future. The AVME model attempts to capture this adaptive capability for data generation. The model adaptively updates its internal control structure by evaluating its previous output at each moment to cater towards future output. Mathematically imitating these natural mechanisms of the human brain are hypothesized to create an effective data synthesis model.

## 2. Related Work

In recent years, generative model efficiency and architecture have seen a surplus of attention in machine learning research. Emerging state of the art frameworks such as variational autoencoders (VAE) (Kingma & Welling, 2013), Generative Adversarial Networks (GAN) (Goodfellow et al., 2014), and the Deep Recurrent Attentive Writer (DRAW) (Gregor et al., 2015) embody the variance in generative model architecture. Each model holds a particular benefit in performance; this paper incorporates ideas from the VAE and DRAW models. These generative techniques are implemented in a recurrent manner, a process effectively encapsulated by the recurrent neural network.

### 2.1. Variational Autoencoder

#### 2.1.1. Variational Inference

Bayesian inference is a common method for drawing conclusions about data. By the nature of Bayesian statistics, this type of statistical inference incorporates both the observed data and prior beliefs about its structure.

In a general inferential setting, there are some data $x$ for which conclusions are to be drawn. This can be done by defining a distribution over the data, which makes assumptions about the nature of how the data was sampled. These parametric distributions are commonly referred to as probability density functions, and a number of them exist for a variety of settings.

A common probability density function is the standard Gaussian $\mathcal{N}(x; \mu, \Sigma)$. For the sake of example, one might wish to parameterize a Gaussian with respect to some observed data $x$. To do so, inference must be performed over the density function's parameters, $\mu$ and $\Sigma$, so as to maximize the likelihood of the data under the resulting parameterized density.

To generalize the above example, a given density is often referred to as $p(x; \theta)$, where $x$ are data and $\theta$ is a vector of parameters that define the distribution. A joint probability distribution can be defined over $x$ and $\theta$:

$$p(x, \theta) = p(x|\theta)p(\theta) \tag{1}$$

The term $p(x|\theta)$ captures the probability density with re-

spect to the parameters $\theta$, known as the likelihood function. The prior distribution $p(\theta)$ incorporates information known prior to observation of the data, a defining aspect of Bayesian statistics.

This joint distribution sets the stage for the task of inference over the density function's parameters. Ultimately, the goal is to compute a posterior distribution $p(\theta|x)$ over parameter vector $\theta$. This means $\theta$ is being treated as a continuous random variable conditioned on $x$, and can thus give a good indication to probable parameters under the data.

According to Bayes' rule, the posterior distribution $p(\theta|x)$ can be calculated using the information as defined under the joint density:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{\int_\theta p(x|\theta)p(\theta)} \tag{2}$$

The marginalization integral in the denominator is often intractable, however, disallowing exact computation of the posterior. The goal of variational inference is to find a close approximation of this intractable posterior and thus allow Bayesian inference to take place. In order to make this approximation as simple and efficient as possible, it is assumed a well-defined density function $q_\lambda(\theta|x)$ can be parameterized so that it sufficiently resembles the true posterior $p(\theta|x)$. The subscript $\lambda$ represents the vector of parameters defining the probability density $q(\theta|x)$, and could similarly be denoted $q(\theta|x, \lambda)$. For example, if $q$ were Gaussian (as it often is), $\lambda$ would be the vector of model parameters $[\mu, \Sigma]$.

In order to parameterize $q$ so that it best approximates the true posterior $p(\theta|x)$, a measure of the difference between the two probability distributions must be calculated, and subsequently minimized. This difference can be calculated through the Kullback-Leibler divergence between the two distributions:

$$\mathcal{D}_{KL}[q_\lambda(z)||p(z|x)] \tag{3}$$

Therefore, the optimal parameters for making $q$ similar to $p(\theta|x)$ are computed by minimizing the KL divergence between the two distributions:

$$q^*_\lambda = argmin_\lambda \, \mathcal{D}_{KL}[q_\lambda(z)||p(z|x)] \tag{4}$$

This term can be expanded to a tractable function, known as the Evidence Lower Bound (ELBO). The resulting function can be computed for each data point $x_i$ in some dataset $D$:

$$ELBO_i(\lambda) = E_{q_\lambda(z)}[\log p(x_i|z)] - \mathcal{D}_{KL}[q_\lambda(z)||p(z)] \tag{5}$$

The terms in the above equation can be computed in a variety of ways, one of which is outlined by the variational autoencoder model.

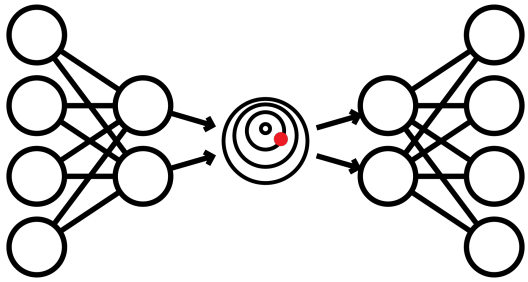

*Figure 2.* The general structure of the variational autoencoder model. The leftmost encoder network encodes the input data into a lower dimension by outputting parameters to define the latent distribution. This latent distribution is represented by the countour plot in the middle of the figure, which is subsequently sampled from, as represented by the red dot. This sample is then fed through rightmost decoder network, bringing the latent sample back into the higher dimensional data space. (Figure created by Sam Griesemer).

#### 2.1.2. Variational Autoencoder Approach

The VAE model is a particularly effective approach to modeling a data distribution. It solves the above optimization problem of variational inference by utilizing deep learning techniques, allowing for efficient and accurate approximations of the data distribution in question.

Intuitively, VAE's attempt to model a data distribution by learning to effectively represent the data $x$ with the model's internal parameters. This is done by training the VAE to reconstruct the data given to it as input, forcing the model to embed its structure. Reconstruction takes place over the course of three steps: encoding, sampling, and decoding. First, the high-dimensional input data is encoded into low-dimensional space, so as to extract the most defining features of the data. Then a sample is drawn from this low-dimensional distribution, as parameterized by the encoding process. This sample is then decoded back into the higher dimensional data space, and the resulting output is the model's reconstruction of the data given to it.

Specifically, the encoding and decoding processes are implemented using neural networks. The input data are passed into an encoder network, mapping each data point $x$ to output values that parameterize a lower dimensional distribution. This low-dimensioal distribution is Gaussian, and represents $q_\lambda(\theta|x)$. The encoder's job is to output an appropriate parameter vector $\lambda$ to define $q$, the posterior approximation. A sample is then drawn from this distribution and subsequently passed into the decoder network. This neural network maps the low-dimensional sample back to the same dimension as the input data.

The output of the VAE is then compared to the original

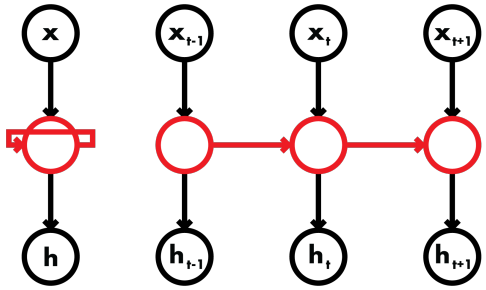


*Figure 3.* **Left:** A standard recurrent neural network with inputs $x$ and outputs $h$. The recurrent node (red) is the hidden node of the network, connecting inputs to outputs and recurring through time. **Right:** An unrolled recurrent neural network. Although visually different, it is the same as the diagram on the left, simply with unrolled layers to show function through time. (Figure created by Sam Griesemer).

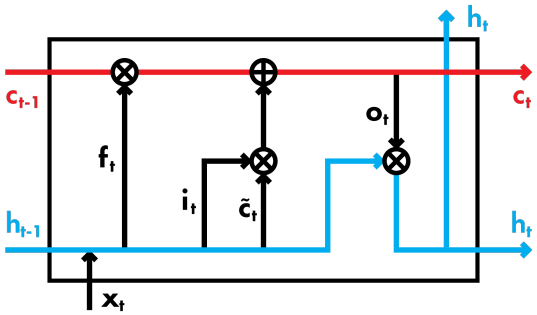


*Figure 4.* Standard LSTM structure. Gates are labeled in accordance to equations 6,7,8,9. Variables $c_t$ and $h_t$ represent the LSTM's cell state and hidden state, respectively, at time $t$. (Figure created by Sam Griesemer).

input, and a reconstruction loss is computed. This loss describes how well the original image was recreated after being passed through the internal parameters of the VAE.

### 2.2. Deep Recurrent Attentive Writer

The Deep Recurrent Attentive Writer (DRAW) is an effective approach to generating data recurrently. The model implements recurrent neural networks with the mechanisms of the variational autoencoder. The AVME model draws inspiration from this architecture, namely the cumulative output sample mechanism and certain recurrent weight connections between iterations. These specific mechanisms are described further in following sections.

## 3. Model

Neural networks, as mentioned earlier, are universal function approximators capable of mapping inputs to outputs through sequences of nonlinear mappings. In a typical feedforward setting, these models are capable of mapping single inputs to single outputs. However, there are many settings in which data are sequential, and thus intertwined with the dimension of time. Take, for example, the task of language translation. In order to translate an English sentence to Spanish, one has to deal with a sequence of words through time. Each word is a high-dimensional data point on its own; the resulting sentence as a whole inherently connects each data point through the dimension of time. By their very nature, standard feedforward neural networks are not suited for modeling sequential data. As a result, the recurrent neural network was introduced (see Figure 3).

Unfortunately, standard recurrent neural networks suffer greatly from the "vanishing/exploding gradient" problem. As error is backpropagated through the network, computed gradients are sequentially multiplied by values greater or less than one. As a result, the gradient either vanishes or increases rapidly, rendering the network ineffective. In order to resolve this issue, a variety of recurrent neural networks variants have been introduced to better control the flow of data through the network, thereby diminishing the gradient's sporadic tendencies. One such variant is the Long-Short Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997), outlined in greater detail below.

### 3.1. Network Architecture

LSTM networks are a recurrent neural network variant capable of effectively utilizing both current data and data seen many iterations earlier. A model of such capacity helps to remove issues surrounding standard recurrent networks, such as the previously mentioned vanishing/exploding gradient. LSTM's have proved to be incredibly successful at understanding sequences, and are thus used as an integral component in our model. In particular, this paper extends the Long Short-Term Memory architecture (Hochreiter & Schmidhuber, 1997) with the forget gate extension (Gers et al., 1997).

The model builds from a standard LSTM network with forget matrix $f_t$, input matrix $i_t$, and new cell state candidates $\hat{c}_t$:

$$f_t = \sigma(w_f \cdot [x_t, h_{t-1}]) \tag{6}$$

$$i_t = \sigma(w_i \cdot [x_t, h_{t-1}]) \tag{7}$$

$$\hat{c}_t = tanh(w_c \cdot [x_t, h_{t-1}]) \tag{8}$$

$$c_t = (c_{t-1} \cdot f_t) + (i_t \cdot \hat{c}_t) \tag{9}$$

where $\sigma$ denotes the logistic sigmoid function and $x_t$ is the current training example. The above terms are computed and the cell state $c_t$ is updated during each iteration.

These LSTM nodes are used in conjunction with core ideas from the variational autoencoder. As opposed to using feed-

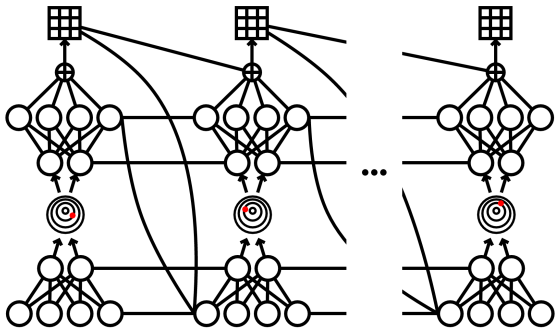

*Figure 5.* Generalized structure behind the AVME model. The body of the network is quite similar to that of Figure 2, but one should note the black lines denoting recurrent weights connecting information from one iteration to the next. The leftmost instance of the network represents the first of $K$ iterations for generating samples, while the rightmost instance represents the last of $K$ iterations in the current generative series. (Figure created by Sam Griesemer).

forward neural networks for a VAE's encoder and decoder network, LSTM recurrent networks are used instead. This allows the model's information control flow to persist through time. Intuitively, an LSTM network controls the flow of information into the VAE's encoder network, while another LSTM network controls how much of the VAE's decoder network can contribute to the output.

With the MNIST handwritten digit dataset (LeCun et al., 1998) in mind, we found an effective encoder network structure to contain 784 input nodes (number of pixels per MNIST image), 400 hidden nodes, and 100 output nodes. The 100 output nodes correspond to the number of dimensions in the latent space; the encoder network maps from the data space into this lower dimensional latent space. The decoder network structure is simply the opposite of the encoder; 100 input nodes, 400 hidden nodes, and 784 output nodes. The decoder maps from the parameterized latent space back to the data space.

A specified number of iterations $K$ are decided upon before the model is trained. The model subsequently has these $K$ iterations to learn a sequential reconstruction of the current training data point, and the cumulative output matrix $s$ represents the model's final reconstruction. Only after the specified $K$ iterations is the reconstruction of the model compared to the original input, and internal parameters are adjusted accordingly. This training and sampling processes are outlined in further detail in their respective sections below.

A generalized visual interpretation of model can be seen in Figure 5. Mathematically, from any iteration $t-1$ to the next iteration $t$, the AVME model is defined as follows:

$$h_t^{\mathcal{E}} = \mathcal{E}(x_t, s_{t-1}, h_{t-1}^{\mathcal{E}}, h_{t-1}^{\mathcal{D}}) \tag{10}$$

$$z_t \sim q(h_t^{\mathcal{E}}) \tag{11}$$

$$h_t^{\mathcal{D}} = \mathcal{D}(z_t, h_{t-1}^{\mathcal{D}}) \tag{12}$$

$$s_t = h_t^{\mathcal{D}} + s_{t-1} \tag{13}$$

where $\mathcal{E}$ represents the encoder network, $\mathcal{D}$ represents the decoder network, $h_t$ represents corresponding denoted LSTM network's hidden state, $s_t$ represents the cumulative output matrix, and $z_t$ represents the sample drawn from latent distribution $q$. Figure 5 displays the recurring weights between iterations with black lines across network instances.

### 3.2. Loss

The loss of a model is a measure of how well it is performing a given task. In the context of an input classification task, the loss is defined as the difference between the model's predicted classification and the true class the input data point belongs to. This metric describes how "far off" the model's prediction is from its ideal value; minimizing this difference allows the model to learn a better representation of the data.

In the case of generative models, the loss is generally defined to be the negative log-likelihood of generated reconstruction samples $\tilde{x}$ under the data $x$:

$$\mathcal{L} = -log\ p(x|\tilde{x}) \tag{14}$$

Minimizing this loss is mathematically similar to performing maximum likelihood estimation, a procedure for finding probable parameters to define a distribution with respect to some observed data. To see this equivalency, one must understand the maximum likelihood setting. Similar to Bayesian inference, there are some data $x$ and learnable parameters $\theta$ that define a density $p(x|\theta)$. The problem of inference is similar:

$$\theta = argmax_\theta\ p(x|\theta) \tag{15}$$

As can be seen above, the maximum likelihood determines the most likely parameters to define the available data, and thus is a form of statistical inference. Yet, it differs from Bayesian inference in that no prior beliefs regarding the data are incorporated into the resulting parameter estimation. To solve for the parameters $\theta$, the likelihood $p(x|\theta)$ is put into log-space, differentiated with respect to the parameters, and set equal to zero to solve for the function's maximum. Another important note is the fact that the point $x$ that maximizes some function $f(x)$ is also the point that minimizes that function's reflection (over the x-axis), $-f(x)$:

$$argmax_x\ f(x) = argmin_x\ -f(x) \tag{16}$$

Thus

$$argmax_\theta \, log \, p(x|\theta) = argmin_\theta \, - log \, p(x|\theta) \quad (17)$$

The right-hand side of this equation is similar to the loss function in equation 14. The objective is to modify the parameters $\theta$ of a model to minimize the negative log-likelihood with respect to the available data. In the case of the loss $-log \, p(x|\tilde{x})$, the model is trying to generate samples $\tilde{x}$ that are likely under the data, and can be done by minimizing the negative log-likelihood.

The loss of the AVME model is a measure of how well it is able to reconstruct input data. Minimizing this loss subsequently minimizes the negative log-likelihood of its generative samples under the data, which are drawn after the model has been trained.

To train the AVME model (as described in the next section), the binarized MNIST dataset (Salakhutdinov & Murray, 2008) was used. This puts the model is a binary setting, and binary cross entropy loss was used as the loss function to minimize. Generalized formulae from this section derived from (Goodfellow et al., 2016).

### 3.3. Training

Training the AVME model is the process of minimizing its loss with respect to the training data. A common benchmark dataset in machine learning is the MNIST handwritten digit dataset. MNIST was used to train the AVME model and test its performance on the task of generating new data samples.

Typically, neural networks minimize a loss function using an optimization procedure known as gradient descent. The intuition behind this method is fairly straightforward; the gradient of the loss function is descended, so as to reach a minimum. The use of gradient descent incurs a differentiable model, requiring the partial derivative of the loss be computed with respect to each learnable parameter. In the case of neural networks, the process of modifying underlying weights is known as backpropagation (backward propagating error through the model). In the general case of the feedforward neural network (Figure 1), backpropagation first entails computing the error between the networks prediction and the true training output. The error determined by this computation is then spread backwards throughout the network, assigning each of the model's parameters a degree of responsibility for contributing to the output error. This process is repeated until the model shows a sign of convergence, or when the iterative change in parameters tends to stagnate.

The dimensionality of a model's loss function is determined by the number of learnable parameters within the model. As the number of parameters in the model increase, loss complexity increases. This is the primary reason behind long training intervals before convergence of parameters.

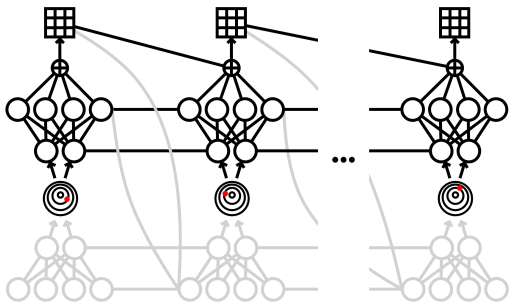

*Figure 6.* Generative procedure of the AVME model. This figure is similar to the original structure (Figure 5), except for greyed out elements of the network not used during generative sampling. Specifically, the encoder network has been greyed out, as data generation only requires the latent distribtuion and the decoder network. (Figure created by Sam Griesemer).

Formally, some defined loss function $\mathcal{L}(x)$ evaluates the error of a model with respect to each of its parameters. To modify each parameter so as to minimize loss, the partial derivative of the loss function with respect to each parameter is computed. This is known as the gradient, and provides information about how to change the parameters to descend the loss function, explaining the term gradient descent. Each parameter $\theta$ of a model is updated as follows:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{L}}{\partial \theta_t} \quad (18)$$

where $\alpha$ represents a learning rate to scale down the gradient's effect. Generally speaking, this is how the model updates its parameters and is ultimately able to learn from the data being used for training.

### 3.4. Sampling

To generate data from the model, iterative samples were drawn from a standard Gaussian and passed through the decoder network. This entails stripping off the encoder network and focusing solely on how the decoder network has learned to map the latent space to the data space. This process is repeated for each of the $K$ specified iterations, with each output contributing to the cumulative sample matrix $s$. The following is repeated $K$ times:

$$z_t \sim \mathcal{N}(0, 1) \quad (19)$$

$$h_t = \mathcal{D}(z_t, h_{t-1}) \quad (20)$$

$$s_t = h_t + s_{t-1} \quad (21)$$

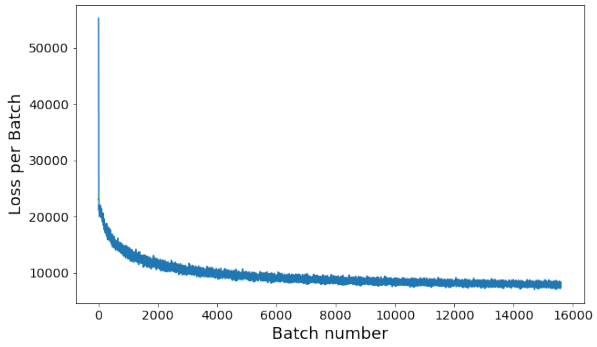


*Figure 7.* The convergence of the model on the MNIST training dataset. The model was trained on data batches of size 100 and the loss was recorded after each batch. The loss per batch is represented by the y-axis, while the number of batches seen by the model is represented by the x-axis. One can clearly note of the graph's tail, indicating convergence. (Figure created by Sam Griesemer).

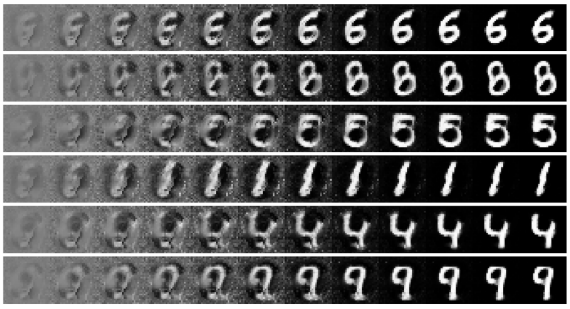

*Figure 8.* Example results of AVME on generating MNIST images. The model was trained to generate images in 12 iterations, and the sequential generation has been unrolled horizontally. (Figure created by Sam Griesemer).

This process is represented by Figure 6, displaying the absence of the encoder network in the process of data synthesis.

## 4. Results

The primary measure of the AVME model was its performance on the binarized MNIST dataset. This dataset was used for comparison to other models in the field, as it is so commonly used for benchmark tests across machine learning models.

Using procedures outlined in the training and sampling sections, the AVME model was trained and sampled from to produce novel MNIST images. The model was trained on batches, or chunks, of data, so as to speed up the learning procedure. The convergence of the model over time is shown in Figure 7, where the loss per batch was recorded for each batch seen by the model. As can be see by the tail of the graph, the parameters of the model converge to a stable state. Upon convergence, the model was then sampled from, resulting in the generative samples shown in Figure 8.

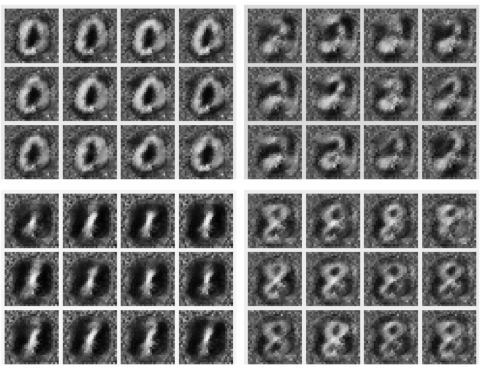

*Figure 9.* Example results of conditional image generation. The model was provided with a class label upon sampling, resulting in selective synthesis. The top left block includes outputs of digit zero queries, top right digit two queries, bottom left digit one queries, and bottom right digit eight queries. (Figure created by Sam Griesemer).

### 4.1. Conditional Generation

The AVME model was additionally tested on its ability to conditionally generate images; that is, take a class label as input during training so that specific classes of images can be generated on command. Results of following procedures can be seen in Figure 9.

Training the model to conditionally generate images is relatively similar to the general training procedure as described in Section 3.3. The model is simply fed a class label $y_t$ as an additional input:

$$h_t^{\mathcal{E}} = \mathcal{E}(x_t, s_{t-1}, h_{t-1}^{\mathcal{E}}, h_{t-1}^{\mathcal{D}}, y_t) \tag{22}$$

$$z_t \sim q(h_t^{\mathcal{E}}) \tag{23}$$

$$h_t^{\mathcal{D}} = \mathcal{D}(z_t, h_{t-1}^{\mathcal{D}}, y_t) \tag{24}$$

$$s_t = h_t^{\mathcal{D}} + s_{t-1} \tag{25}$$

This allows both the encoder and decoder networks of the model to incorporate the correlation between input images and class labels in their respective recurrent hidden states. Additionally, to generate a conditional sample, the general sampling procedure as outlined in Section 3.4 is followed with the addition of a class label as input to the decoder network:

$$z_t \sim \mathcal{N}(0, 1) \tag{26}$$

$$h_t = \mathcal{D}(z_t, h_{t-1}, y_t) \tag{27}$$

$$s_t = h_t + s_{t-1} \tag{28}$$

The model is then allowed $K$ iterations to cumulate a final output image.

## 5. Discussion

The model was trained on a specified $K = 12$ sampling iterations to recurrently produce on output. Figure 8 shows the unrolled iterative generation process, and how the model responds to its previous output to generate an increasingly coherent sample at each iteration. At the left of the samples, the initial image is cloudy and unclear. Over time, however, a more coherent foreground emerges and the cluttered background fades to black behind it. Another interesting thing to note is the model's tendency to change the class of image in focus during generation. Upon observing the third row from the top, one can see how the model began to generate a digit resembling zero. Nearly halfway through the sample, however, the model decides to commit to generating a digit resembling a five, and leaves a gap near the tail of the five further into the sample. Additionally, the sample on fourth row from the top of Figure 8 begins with a cluttered image of what appears to be a six. The six is so narrow that the model then decides early on to commit the sample to the digit one instead.

The results of conditional generation method were slightly worse than classic image synthesis, as can be seen in Figure 9. It appears as though the model had more difficulty separating out primary focus from the background; the digits are much more cloudy than in Figure 8. Initially the output images may each look identical, but upon closer inspection, the images vary slightly. This is a result of the inherit stochasticity of sampling from the latent distribution, ultimately resulting in slight differences between outputs. Each output image encapsulates the model's learned representation between the training label and paired input image; this representation can change according subtle differences early on in the generative sequence.

### 5.1. Statistical Analysis

To evaluate the model's performance of reconstructing data, the negative log-likelihood was computed over the testing dataset. This separate dataset includes 10,000 binarized MNIST example pairs never seen by the model during training. By testing the AVME model's performance on never before seen data, an unbiased metric can be computed and generalization effectiveness can be determined.

To compute the negative log-likelihood of the model's reconstructive output, each input image was passed through the model and the cumulative output was compared to the true data example. Because both the training and testing set are binarized, the Bernoulli likelihood function was used for computing the log-likelihood between the model's output and testing examples.

Formally, the final cumulative output matrix $s_K$ was passed through the sigmoid function $\sigma$ to "squash" all values between zero and one. The resulting matrix includes Bernoulli means, providing the necessary parameters for computing the likelihood.

The Bernoulli likelihood function is as follows:

$$\mathcal{L}(\theta|x) = \prod_{x_i} \theta^{x_i}(1-\theta)^{1-x_i} \tag{29}$$

where $x \in \{0, 1\}$ and $\theta$ is the mean. To make this function computationally efficient, it is placed in log space:

$$\log \mathcal{L}(\theta|x) = \sum_{x_i} x_i \log \theta + (1 - x_i) \log(1 - \theta) \tag{30}$$

In the case of the AVME model, Bernoulli means are defined by $\sigma(s_K)$, which in turn were used as parameters in the above likelihood function along with input data $x$. As a result, the computed log-likelihood function was computed for each $x_i$ in the test set:

$$\log \mathcal{L}(\sigma(s_K)|x) = \sum_{x_i} x_i \log \sigma(s_K) + (1-x_i) \log(1-\sigma(s_K)) \tag{31}$$

This sum was computed over 10,000 test examples. The average log-likelihood per test example was then computed. Table 1 includes previous results from other generative models as well as the AVME model on the binarized MNIST dataset. AVME was able to outperform the state of the art on this dataset, achieving significantly lower negative log-likelihood values.

### 5.2. Further Work

The AVME model should be extended for use on color images. This requires using new assumptions about the data being used for training, removing the simplistic nature of MNIST's binary setting. Additionally, it would be interesting to experiment more with the model's parameters, such as the predetermined number of sample iterations $K$ or sizes of the encoder and decoder network. More extensive research could potentially determine optimal values for parameters of the model similar to those listed above, thus maximizing the model's effectiveness during training and sampling.

| Model | NLL ($-logp$) |
|---|---|
| DRAW | 80.97 |
| NADE | 88.33 |
| DLGM | 86.60 |
| DBM | 84.13 |
| AVME | 80.50 |

*Table 1.* Negative log-likelihood results of a variety of models on the binarized MNIST dataset. The right-hand column displays the average negative log-likelihood per test set example (in nats) on the binarized MNIST test set for the respective model shown in the left-hand column. The model results shown here are DRAW (Gregor et al., 2015), NADE (Uria et al., 2014), DLGM (Rezende et al., 2014), and DBM (Salakhutdinov & Hinton, 2009)

## 6. Conclusion

This paper introduced the Adaptive Variational Memory Encoding (AVME) model, designed to incorporate adaptive mechanisms of the human brain into the process of synthesizing data. It was shown that the model is capable of performing well at both classical and conditional image generation, with slightly worse performance conditionally generating images. The AVME model is also competitive with respect to its reconstructive capabilities, outperforming the state of the art on the binarized MNIST dataset. The natural assumptions inspiring the model's structure and functionality proved to be an effective statistical mechanism for recurrently generating novel data.

*Submitted to Journal of Experimental Secondary Science (JESS) on May 8, 2017

## References

Gers, Felix A., Schmidhuber, Jurgen, and Cummins, Fred, 1997. Learning to forget: Continual prediction with lstm.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. 2016, *Deep Learning*.

Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua, 2014. Generative adversarial networks.

Gregor, Karol, Danihelka, Ivo, Graves, Alex, Rezende, Danilo Jimenez, and Wierstra, Daan, 2015. Draw: A recurrent neural network for image generation.

Hochreiter, Sepp and Schmidhuber, Jurgen, 1997. Long short-term memory.

Kingma, Diederik P and Welling, Max, 2013. Auto-encoding variational bayes.

LeCun, Yann, Bottou, Leon, Bengio, Yoshua, and Haffner, Patrick, 1998. Gradient-based learning applied to document recognition.

Rezende, Danilo J, Mohamed, Shakir, and Wierstra, Daan, 2014. Stochastic backpropagation and approximate inference in deep generative models.

Salakhutdinov, Ruslan and Hinton, Geoffrey E, 2009. Deep boltzmann machines.

Salakhutdinov, Ruslan and Murray, Iain, 2008. On the quantitative analysis of deep belief networks.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo, 2014. A deep and tractable density estimator.