DeepNAG: Deep Non-Adversarial Gesture Generation

Mehran Maghoumi  
NVIDIA, University of Central Florida
mehran@cs.ucf.edu

Eugene M. Taranta II  
University of Central Florida
etaranta@gmail.com

Joseph J. LaViola Jr.  
University of Central Florida
jjl@cs.ucf.edu

ABSTRACT
Sythetic data generation to improve classification performance (data augmentation) is a well-studied problem. Recently, generative adversarial networks (GAN) have shown superior image data augmentation performance, but their suitability in gesture synthesis has received inadequate attention. Further, GANs prohibitively require simultaneous generator and discriminator network training. We tackle both issues in this work. We first discuss a novel, device-agnostic GAN model for gesture synthesis called DeepGAN. Thereafter, we formulate DeepNAG by introducing a new differentiable loss function based on dynamic time warping and the average Hausdorff distance, which allows us to train DeepGAN’s generator without requiring a discriminator. Through evaluations, we compare the utility of DeepGAN and DeepNAG against two alternative techniques for training five recognizers using data augmentation over six datasets. We further investigate the perceived quality of synthesized samples via an Amazon Mechanical Turk user study based on the HYPE benchmark. We find that DeepNAG outperforms DeepGAN in accuracy, training time (up to 17x faster), and realism, thereby opening the door to a new line of research in generator network design and training for gesture synthesis. Our source code is available at https://www.deepnag.com.

CCS CONCEPTS
• Human-centered computing → Gestural input; • Computing methodologies → Neural networks.

KEYWORDS
deep neural networks, generative modeling, gesture generation, generative adversarial networks, dynamic time warping

ACM Reference Format:

1 INTRODUCTION
Recently we observe that system designers are integrating gestures into almost every product with a user interface, igniting the need for accurate gesture recognizers [28, 38]. As these recognizers get more sophisticated and accurate, so does their need for more data. While the size of publicly available datasets continues to grow, obtaining more task-specific data is not always easy which highlights the importance of synthetic data generation. Among such methods, generative adversarial networks (GAN)[12] have shown great promise in various problem domains [10, 32, 48], including handwriting and gesture generation [40, 50]. Typically, these networks consist of a generator and a discriminator. The discriminator aims to determine if a given example is real or fake, whereas the generator aims to fool the discriminator into confusing fake examples with the real ones. To our knowledge, such models have not received much attention towards modality-agnostic gesture generation wherein gestures are represented by a sequence of 2D or 3D spatio-temporal features typically produced by touch interfaces, Kinect or similar input devices. Additionally, GANs require the concurrent training of two networks, which makes the training procedure long and challenging.
We call this approach when our generator is used for data augmentation. which have recently become common practice in the literature of character generation which generates the temporal pen movements. 

Zhang’s et al. [50] proposed a recurrent GAN for Chinese character generation which generates the temporal pen movements. The problem domain of these works is different from ours. We focus on modality agnostic gesture generation to produce hand, full-body or 2D pen gestures. Our model learns the representation of a given gesture and produces new gestures in that same representation. Also, our generator can be trained without a discriminator, which sets us apart from the work of Zhang et al. [50].

Training generative models without a discriminator has also been explored in the literature. Yu et al. [48] and Guo et al. [15] proposed generating text sequences using reinforcement learning techniques. Lin et al. [27] presented the use of a ranking mechanism instead of a discriminator, and Li et al. [26] introduced an adversarial optimization procedure to train a text generator. All of these work focus on generating sequences of discrete tokens (e.g., text), whereas our goal is to generate real-valued and continuous multidimensional gesture sequences, which is highly challenging as data can take arbitrary values. Lastly, our loss formulation is different from [26] in that our formulation is inherently non-adversarial.

3 GESTURE GENERATION WITH DEEP RECURRENT NETWORKS

In this section we present our proposed deep learning approaches for gesture synthesis. We first discuss our initial GAN approach from which the DeepNAG generator is born. We then describe the intuition behind our loss function, followed by its formal definition.

3.1 Notations and Problem Definition

In this work, we represent gestures as a temporal sequence of input device samples (e.g. time-series of 3D joint positions or 2D touch coordinates), thus our data is spatio-temporal. At time step $t$, the gesture data is the column vector $x_t \in \mathbb{R}^N$, where $N$ is the dimensionality of the feature vector. Thus, the entire temporal sequence of a single gesture sample is the matrix $x \in \mathbb{R}^{N \times L}$, where $L$ is the length of the sequence in time steps. For simplicity, and as typical in most gesture recognition work [41, 43], we spatially resample all gesture samples to the same length $L^1$ as described in [45]. We denote the vector trajectory path of a gesture $x$ with $\tilde{x} = \{(x_t - x_{t-1}) \mid \forall x_{t-1} \in x\}$. Lastly, we use $|A|$ to denote the cardinality of a point set $A$, thus $|x| = L$ and $|\tilde{x}| = L - 1$.

We define gesture generation as producing synthetic examples $x' = \{x'_0, x'_1, ..., x'_{L-1}\}$ over a dataset of gestures $D$ such these samples mirror data-specific properties of the samples in $D$, as if these examples were seemingly sampled from $D$. Formally, if $p_D$ is dataset’s distribution such that $x \in D \implies x \sim p_D$, our goal is to synthesize $x'$ where $x' \notin D$ but $x' \sim p_D$. We aim to achieve this using a deep recurrent network $G$ which maps a class-conditioned latent vector $z_t$ to a synthetic example $x' = G(z_t; \theta_G)$, where $\theta_G$ are the trainable parameters of $G$. Henceforth, we use $G_{\theta_G}(z_t)$ in place of $G(z_t; \theta_G)$, and use $\mathcal{L}$ to denote an objective function to minimize (a training loss function).

3.2 Gesture Generation with GANs

Our initial approach for gesture generation uses the well-known GAN training setting comprised of a generator and a discriminator.

\textsuperscript{1}We use $L = 64$
We call this recurrent model DeepGAN, which we designed incrementally, informed by the latest developments in deep learning. Early on, the simplicity and the recognition power of the recently proposed DeepGRU model [28] inspired us to adopt it as our discriminator. This encoder-style model has shown promising results in various recognition tasks [6, 38]. Through experiments guided by an ablation study on DeepGRU [28], and with the goal of managing design complexity, we settled for the simpler uDeepGRU [6] variant as our discriminator. As for our generator, we conducted experiments across different datasets with generators consisting of both LSTM and GRU units, as well as a varying number of recurrent layers. We observed stabler training, less overfitting and more plausible outputs with a decoder-style network resembling the flipped version of our discriminator. A possible explanation for this could be that this choice potentially benefits from the balance between the two D and G networks. Figure 1 depicts the architecture of DeepGAN, which we believe is easy to understand and straightforward to implement in any modern deep learning framework. A common design for RNN-based generators is the use of the \( \tanh(\) activation function in the last layer, which is what we use as well.

To generate a gesture sample \( x^* \) of class \( c \), the class-conditioned latent vector \( z_c \) is fed to \( G \) where \( z_c \) is defined as Equation 1. Note that each time step \( z_t \in z_c \) is sampled independently from the standard normal distribution\(^2\), and class-conditioning is done by appending the one-hot representation of \( c \) to each time step which avoids ignoring the conditioning through forgetting [10].

\[
z_c = \left\{ z_t; \hat{c} \right\}; \forall z_t \sim \mathcal{N}(0, 1), \hat{c} = \text{one-hot}(c) \quad (1)
\]

We experimented with different loss functions to train DeepGAN. Even though training with the classic adversarial loss [12] yielded plausible results, we observed improved sample quality and better convergence with the improved version [14] of the Wasserstein loss (WGAN) [2], which is what we settled on using for our evaluations. Figure 1 depicts DeepGAN’s architecture.

### 3.3 Non-Adversarial Gesture Generation

Although DeepGAN shows promising results (see Section 4), it demonstrated a few shortcomings early on. Most importantly, the need for training two networks simultaneously increases the training burden: changes in one network may adversely affect the other and most hyperparameters need to be tuned twice. Moreover, training times are long and we observed that the model required tens of thousands of generator iterations to converge. Aiming to reconcile these challenges, we present our loss function that completely replaces the discriminator. We start by providing an intuition for training a sequence generator without a discriminator, then proceed with the formal definition of our loss function.

**Intuition.** The key to simplify the network design is the answer to a fundamental question: can we possibly train the single network generator network \( G \)? A generator network aims to learn the distribution of the underlying dataset \( D \), so that new examples can be sampled from the distribution, which is typically done with the help of a discriminator (or a critic, in the case of WGAN-based models). To simplify the gesture generation procedure, we pose the problem in a slightly different way: let us train a generator that aims to produce gestures that are similar to their real counterparts. The fundamental question then becomes, how to train a generator that increases the similarity between the generated and the real gestures? The answer is surprisingly simple: by reducing the dissimilarity between the two! Conveniently, a well-studied dissimilarity measure for time-series (as well as gestures [43]) is the dynamic time warping (DTW) [36]. A differentiable formulation of DTW called soft DTW (sDTW) was recently proposed by Cuturi and Blondel [7].

DTW is a dynamic programming algorithm that was originally proposed for speech recognition [36]. It is a dissimilarity measure of two time-series that can be used to find their optimal alignment for various time-series analysis tasks [33]. Given two time-series \( X = \{x_1, x_2, \ldots, x_n\} \) and \( Y = \{y_1, y_2, \ldots, y_m\} \), a cost matrix \( \Delta \) of size \( n \times m \) is built. Each element \( \Delta_{ij} \) is the matching cost of \( x_i \) to \( y_j \), computed via the following recursion:

\[
\Delta_{ij} = f(x_i, y_j) + \min \{ \Delta_{i-1,j}, \Delta_{i,j-1}, \Delta_{i-1,j-1} \} \quad (2)
\]

where \( f(x_i, y_j) \) is a problem-specific cost (distance) function. Although Euclidean distance (ED) is widely used, Taranta et al. [41, 43] demonstrated the superiority of using the cosine similarity measure (COS) for gesture recognition problems. Once \( \Delta \) is fully computed, the value \( \Delta_{nm} \) is the dissimilarity measure of \( X \) and \( Y \), and the path through the matrix that yields \( \Delta_{nm} \) is the optimal alignment between the two time-series. Cuturi and Blondel’s sDTW formulation [7] replaces the \( \min() \) operator with a differentiable (soft) minimum defined as:

\[
\min_Y \gamma\log \sum_{i=1}^n e^{-\Delta_{i,Y}/\gamma} \quad (3)
\]

where \( \gamma \) controls the smoothness (smaller \( \gamma \) yields a closer approximation of classic DTW). Cuturi and Blondel show that the resulting \( \Delta_{nm} \) would be the expected value of dissimilarity between \( X \) and \( Y \), over every possible alignment between them weighted by their probability under the Gibbs distribution [7]. Further, they formulate the derivative of sDTW using backpropagation through a computation graph. A detailed explanation is available in [7].

**Loss formulation.** To increase the similarity of a real example \( x \) and a fake example \( x' \), one could naively decide to learn \( \theta \) by minimizing \( \mathcal{L}_F = \text{WDTW}(G(x_c); x; f) \) where \( f \) is the cost function of Equation 2. Unfortunately, this formulation merely states that generated samples \( x' \) should be as similar to the real samples \( x \) as possible, ignoring inter-class variations. A trivial solution to this formulation, that was easily achievable in our tests, is the per-class centroid sample.

To overcome this issue and account for inter-class variability, we formulate the loss function as a coverage measure (point set similarity) of a set of fake and real examples. For this, we propose using the Hausdorff distance, a well-studied measure for point set similarity [9] that can be easily implemented and computed. Conveniently, the average Hausdorff distance (denoted as \( d_H \)) between two point sets \( A \) and \( B \) is differentiable [34] and is defined as:
where $d(a, b; f)$ is the distance (dissimilarity) between two points $a$ and $b$ parameterized by $f$. We use $d(a, b; f) = \text{sDTW}(a, b; f)$, and will discuss the choice of $f$ (sDTW’s cost function) shortly. Finally, we propose the following as DeepNAG’s loss function to minimize.

To our knowledge, this is the very first formulation of a single loss measure to train a deep recurrent neural network for generating synthetic gesture sequences:

$$
\mathcal{L}_f(X', X) = d_f^f(x'_1, x_1) + \left| d_f^f(x'_1, x'_2) - d_f^f(x_1, x_2) \right|
$$

(5)

where $X' = \{x'_1, x'_2\}$ (two generated samples), and $X = \{x_1, x_2\}$ (two real samples), and both sample sets belong to the same gesture class. Intuitively, Equation 5 expresses that training $G$ should aim to increase the similarity of fake and real examples (similarity term), while maintaining the similarity balance between two batches of fake and real examples (variation term). The former term ensures real and generated samples are similar, while the latter term ensures generated examples maintain proper overall inter-class variations, effectively avoiding pitfalls such as mode-collapse typically encountered in GANs.

We briefly experimented with non-elastic dissimilarity functions as alternatives to sDTW in our formulation. This was motivated by the fact that our training samples were all resampled to the same length. Our experiments revealed a clear benefit in using sDTW over Euclidean distance in terms of sample generation quality. This observation is in line with prior work [43], and we posit that this is due to the elastic nature of gesture articulations.

**Practical notes.** During training, Equation 5 is computed over a batch of training samples, thus the number of gesture samples involved in the computation of either loss term depends on the training batch size. Also, we only used the derivative $\partial \mathcal{L}_f/\partial x'_1$ to optimize the model since training was faster and more stable. This implies that the computation of $d_f^f(x'_2, x_2)$ as an additional similarity term in Equation 5 is unnecessary, simply because its gradient w.r.t. $x'_1$ is zero.

When computing sDTW($a, b$) we ensure class-awareness: the value is only computed if samples $a$ and $b$ belong to the same gesture class. As for the choice of sDTW’s internal cost function $f$, we started with ED, but the benefits of using COS quickly became apparent to us: minimizing $L_{\text{ED}}$ yielded high-quality results but convergence was slow. Conversely, minimizing $L_{\text{COS}}$ led to much faster convergence with sometimes noisier results. In the end, we settled for minimizing both and leave a thorough study on the effects of each cost function to future work. Lastly, recall our use of fixed-length sequences ($L = 64$) where the points in the sequence are equidistant. To enforce the production of such sequences by $G$ we add an additional term $L_{\text{sample}}$ to our objective. Putting everything together, the following is the loss function that we minimize for our experiments:

$$
\mathcal{L}_{\text{DeepNAG}}(X', X) = \mathcal{L}_{\text{ED}}(X', X) + \mathcal{L}_{\text{COS}}(X', X) + \alpha \cdot L_{\text{sample}}(x'_1)
$$

(6)

where $\mathcal{L}_{\text{sample}}$ is the length of each $\tilde{x}_i \in \tilde{x}$ after $x$ is resampled to $L$ equidistant points. Thus, $L_{\text{sample}}$ simply enforces that points in $x'$ be equidistant with $\alpha$ as its regularizer. Note that minimizing $L_{\text{ED}}$ alone (Equation 5) yields good-quality results in most cases. Yet, we achieved faster convergence and better data augmentation performance using Equation 3.3.

As we discuss shortly, a generator trained with our loss function demonstrates promising results compared to the same generator trained in a GAN setting. In addition to these two training configurations, we considered training our generator in a variational autoencoder (VAE) [19] setting. Although we successfully generated gesture sequences using our proposed VAE, the quality and the variety of our produced samples were sub-par compared to either DeepGAN or DeepNAG. Refer to Appendix A for more details.

## 4 EVALUATION

We evaluate DeepGAN and DeepNAG from two aspects. First, we conduct experiments to determine the efficacy of either model in data augmentation tasks to address data shortage. We then evaluate the perceived realism of our synthetic gestures through a user study on Amazon Mechanical Turk based on a recently introduced benchmark [51].

### 4.1 Data Augmentation Performance

Our experiment design for this study is as follows. Given a dataset of gestures collected from multiple participants, we simulate small training sets by splitting the data into training (50%), validation (20%) and test (30%) sets. Our experiments are all subject-independent; i.e. the data of each participant only appears in one of these sets. This is a more challenging and realistic evaluation protocol, as it ensures that during training, the recognizer never sees any data from the participant that it will be evaluated on during the validation and testing phases.

We begin by training a gesture recognizer on the training set, and use the validation set for model selection. We evaluate the best performing model on the test set and record its recognition error (baseline). Next, we augment the training set with a selected data generation method and repeat the experiment: train the recognizing with this new training set, use the validation data for model selection, and evaluate the best model on the test set. We record the recognizer’s recognition error again, which will be the error after augmenting the training set. Comparing this number with the baseline benchmarks the synthetic data generation method. In total we perform 150 experiments: we train five gesture recognizers on six different datasets to evaluate four synthetic data generation methods against the baseline.

---

1In other words decrease their dissimilarity
Datasets. We selected six datasets among the ones frequently studied in the literature. They vary in size and span across gesture modalities and input devices: JK2017 (Kinect) [43] (14 full-body fighting gestures of 20 participants with Kinect v2), JK2017 (Leap Motion) [43] (eight hand-gestures of 20 participants with Leap Motion), UT-Kinect [46] (ten full-body daily activities of ten participants with Kinect v1), MSR Action 3D [25] (20 full-body actions of ten participants with Kinect v1), SBU Kinect Interactions [49] (8 two-person interaction of seven participants with Kinect v1) and S1-GDS [45] (16 2D pen gestures of ten participants).

Recognizers. We selected five gesture recognizers: support vector machine (SVM), random forest, naïve Bayes, DeepGRU [28] and Jackknife [43]. These represent classic machine learning algorithms, deep learning as well as rapid prototyping [43] approaches, which are common choices for gesture recognizers. The first three methods require explicit feature extraction for which we use the Rubine [35] feature set extended to 3D gestures [37]. Jackknife [43] is a 1-nearest neighbor DTW-based template matching recognizer.

Data generation methods. We compare four data generation methods against the baseline: random Gaussian noises, GPSR [42], DeepGAN and DeepNAG. The effectiveness of GPSR for 3D gestures has been previously demonstrated [6, 28].

Implementation. We implemented DeepGAN and DeepNAG with the PyTorch [29] framework which we have publicly released. Additionally, our implementation requirements yielded multiple other standalone projects, which we have made public in the hope of benefiting the deep learning community. Inspired by [52], we implemented a CUDA version of sDTW with a PyTorch interface using Numba [20]. Our novel implementation runs more than 100× faster than any other publicly available implementation that we know of. Additionally, we implemented fast GRU units using PyTorch’s just-in-time (JIT) compilation features to allow computing their higher-order derivatives, a feature that is missing in PyTorch. Such derivatives are required to implement the improved WGAN loss [14] for GRUs.

Hyperparameters. All hyperparameters were tuned across different datasets, but the same set of parameters were used for every experiment. Both DeepGAN and DeepNAG were trained on the 50% split training set, and shared most hyperparameter settings. We use the Adam [18] solver (β1 = 0.5, β2 = 0.9), with a learning rate of 10−4 and a mini-batch size of 64. DeepGAN-specific hyperparameters were chosen from [14], as they performed the best in our validation runs. Other parameters were chosen via cross-validation as follows. For DeepNAG we used y = 0.1 and α = 105. GPSR parameters were set to γ = 2, σ = 0.25 and the magnitude of random noise was set to 2% of the bounding box of each feature.

Results and discussion. Figure 2 depicts the results of our experiments. In many cases the use of some form of data augmentation decreases the recognition error, indicating that our 50% split to simulate small training sets is working as expected. To better contrast the generation methods we employ a scoring scheme that quantifies whether the use of a given augmentation method is both warranted and effective. Data augmentation is only warranted if a recognizer trained with the additional data outperforms the baseline. Additionally, a method is effective only if it outperforms random noise. We start with a score of zero for a given generator. In each experiment set, we increment this score by one if the method outperforms all other methods in addition to the baseline and random noise. Ties are only counted if the method outperforms both random noise and the baseline, and we use the cumulative score for comparison.

Table 1 presents the computed score aggregates over each dataset and recognizer. We observe that across both aggregate groups, DeepNAG outperforms other methods by a large margin, suggesting its suitability for data augmentation regardless of the choice of dataset or recognizer. In a few cases, DeepNAG reduced the recognition error to zero. Compared to GPSR, these results are notable as DeepNAG generates new examples purely from random noise. Conversely, GPSR perturbs existing examples to generate new ones. This process leaves some characteristics of the original gesture (e.g. bounding box size) largely unchanged, which benefits recognizers that rely on such features.

Figure 2 also shows cases wherein data augmentation seems harmful. In particular, we observe increased errors in almost all cases where data generation is used with multi-actor gestures (Figure 2e). This suggests that our generators may not be suitable for generating multi-actor gestures, which we confirmed by visual inspection. In some cases, both DeepGAN and DeepNAG confuse the main and the secondary actors, yielding malformed gestures. We intend to study the generation of such gestures in future work.

We additionally inspected some of the generated examples of Figure 2f wherein our generators increased recognition errors. Most synthetic examples were visually fine which suggests that the use of domain adaptation techniques may be helpful [1, 30, 53]. We plan to explore this in the future.

During visual inspection, we did not observe any mode-collapse issues with DeepNAG. We observed healthy variations across all gesture classes and datasets with minimal amounts of degenerate samples (except for the few cases noted above, typically less than 1% of the generation samples were degenerate). Lastly, factors such as ease of training and training times compel the use of DeepNAG over DeepGAN as the former offers a significant reduction in training times. Training DeepGAN on a Tesla V100 GPU takes between 3-5 days depending on the dataset size, whereas DeepNAG takes around 3-7 hours under the same conditions, a speedup of 12–17×.

Overall, our results indicate that DeepNAG outperforms DeepGAN on data augmentation tasks, regardless of the choice of dataset or recognizer. These results are notable, as the generator model in both DeepGAN and DeepNAG is exactly the same. In other words, the generator which is trained using our novel loss function outperforms the same generator trained in a GAN setting with the improved Wasserstein loss. Additionally, our loss function trains the generator in a much shorter period of time.

---

1https://github.com/Maghoumi/DeepNAG
2https://github.com/Maghoumi/pytorch-softdtw-cuda
3https://github.com/Maghoumi/JitGRU
4To our knowledge, the cuDNN framework is missing this feature. Thus, at the time of this writing one cannot compute higher-order derivatives for GPU-based GRUs in any deep learning framework that relies on cuDNN.
5A video demo of generated gestures is available at https://www.deepnag.com
Figure 2: Results of evaluation across six datasets (best viewed in color).

Table 1: Generator scores aggregated over dataset and recognizer.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Generator Score</th>
<th>Recognizer</th>
<th>Generator Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>JK2017 (Kinect) [43]</td>
<td>0 2 2</td>
<td>SVM</td>
<td>2 0 4</td>
</tr>
<tr>
<td>JK2017 (Leap Motion) [43]</td>
<td>0 1 4</td>
<td>Naïve Bayes</td>
<td>1 3 2</td>
</tr>
<tr>
<td>UT-Kinect [46]</td>
<td>0 1 2</td>
<td>Random Forest</td>
<td>2 1 3</td>
</tr>
<tr>
<td>MSR Action3D [25]</td>
<td>1 0 3</td>
<td>DeepGRU [28]</td>
<td>0 1 3</td>
</tr>
<tr>
<td>SBU Kinect</td>
<td>2 0 0</td>
<td>Jackknife [43]</td>
<td>0 0 2</td>
</tr>
<tr>
<td>$1$-GDS [45]</td>
<td>2 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Score</strong></td>
<td><strong>5 5 12</strong></td>
<td></td>
<td><strong>5 5 14</strong></td>
</tr>
</tbody>
</table>
4.2 User Study

Qualitative evaluation of generative models through user studies has become a common practice in the literature [22, 51]. We therefore turn our focus to the comparison of DeepGAN and DeepNAG based on the perceived quality of the generated samples using human evaluators based on the HYPE\(_{\infty}\) [51] benchmark. This benchmark defines the gold standard for evaluating generative realism on crowd-sourcing platforms.

To compare different generative models, HYPE\(_{\infty}\) defines an experiment with 30 participants: each participant only sees the results from one of the models. For a given generative model, every participant is shown a total of 100 samples comprised of 50 fake and 50 real samples. Given each sample, participants are asked to indicate whether they think that sample is real or computer-generated. Participants have an infinite amount of time to make this binary choice. Afterwards, the percentage of the samples that were judged incorrectly is computed for every participant. Obtained values are averages over 30 participants and the final result is reported as the HYPE\(_{\infty}\) score for the understudied generative model. Zhou et al. [51] showed that this protocol ensures repeatability and maintains the separability between different generative models and can be used as a reliable measure of the generative model’s quality.

Using this protocol, we conduct our user study on Amazon Mechanical Turk for a given dataset \(\mathcal{D}\) and a generator \(G\). We first train the \(G\) on \(\mathcal{D}\) to convergence. Using the trained model, we then sample as many fake samples as the real samples in \(\mathcal{D}\). The 100 samples needed to show a given participant are selected by randomly choosing 50 fake and 50 real samples from the pool of all available samples. We recruit 30 participants for every combination of \(\mathcal{D}\) and \(G\). Participants begin by studying the purpose of the study and answering demographic questions as detailed in Table 2. We then show them each of the 100 samples and ask them to indicate whether they think a given sample is produced by human or computer-generated (see Figure 3). Every gesture sequence is drawn in the form of a looping gif animation with a framerate of 32. Between each animation loop, we display a countdown with a duration of 0.25 seconds. This was inspired by [51] and was done to avoid confusing participants who may be unaware that they are watching an infinitely-looping animation. Note that the order of the samples shown to each participant is random and varies among different participants. This was done to avoid any learning effects in the experiment.

Participants are given an infinite amount of time to respond to each question. Similar to the HYPE\(_{\infty}\) benchmark, we reveal the correct answer to the participant upon submitting a response to every question. Participants can participate in our study only once, which ensures unique responses across all experiment conditions. At the end of the study, we reveal the overall accuracy of the participant in our task and pay them $2 for their time [51].

When posting our study on the Mechanical Turk platform, we created a list of criteria to ensure the selection of a pool of high-quality workers. We refined and validated these criteria through trials on Mechanical Turk prior to starting our actual study. First, participants must have an approval rating of at least 97% to participate in our study to filter out low-quality workers. Our next participation requirement is that workers must have completed at least 5000 studies. This criterion filters out participants who may have high approval ratings because they recently joined the platform. Lastly, participants must be Mechanical Turk Masters to be eligible to participate in our study. Amazon uses proprietary criteria to grant top-performing workers this qualification. Although the exact criteria is not publicly disclosed, Amazon claims they continuously monitor the performance of master workers across different user studies on the platform to ensure consistent performance\(^9\).

Our study consisted in evaluating each of DeepGAN and DeepNAG on three datasets covering different gesture modalities: Kinect (JK-2017 [43]), Leap Motion (JK-2017 [43]) and Pen gestures ($1\text{-GDS}$ [45]). Thus our generator factor has two levels and our dataset factor has three levels, yielding a total of six experiments.

Results and discussion. In total, we recruited 180 participants (30 participants for each of our six experiments) with an average age of 41 years ($\sigma=10.8$). Figure 4 depicts the demographics of our participants. Across all tasks, participants spent an average of 12.3 minutes ($\sigma=3.8$), and each question was answered in 7.4 seconds on average ($\sigma=2.3$). Considering a payment of $2 per study, our participants were compensated well above the minimum wage specified by the United States federal guidelines ($7.25 per hour at the time of this writing).
Table 3: Amazon Mechanical Turk user study results. Reported values are percentages (averaged over 30 participants). The top performing model on each dataset (with statistical significance) is boldfaced.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Generator</th>
<th>HYPE$_\infty$</th>
<th>Std.</th>
<th>Fake Errors</th>
<th>Real Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>JK2017 (Kinect) [43]</td>
<td>DeepNAG</td>
<td>48.1</td>
<td>8.8</td>
<td>53.9</td>
<td>42.3</td>
</tr>
<tr>
<td></td>
<td>DeepGAN</td>
<td>38.4</td>
<td>12.9</td>
<td>44.3</td>
<td>32.5</td>
</tr>
<tr>
<td>JK2017 (LeapMotion) [43]</td>
<td>DeepNAG</td>
<td>51.0</td>
<td>4.3</td>
<td>56.1</td>
<td>45.8</td>
</tr>
<tr>
<td></td>
<td>DeepGAN</td>
<td>22.7</td>
<td>11.4</td>
<td>23.9</td>
<td>21.4</td>
</tr>
<tr>
<td>$1$-GDS [45]</td>
<td>DeepNAG</td>
<td>50.0</td>
<td>6.7</td>
<td>56.3</td>
<td>43.7</td>
</tr>
<tr>
<td></td>
<td>DeepGAN</td>
<td>44.4</td>
<td>8.3</td>
<td>49.5</td>
<td>39.3</td>
</tr>
</tbody>
</table>

Table 3 summarizes our user study results. In all experiments, we observe higher HYPE$_\infty$ scores for DeepNAG compared to DeepGAN. Unpaired t-tests confirm the difference is significant in every case: $t(58)=3.3, p=0.001$ (JK2017-Kinect [43]), $t(58)=12.4, p<0.001$ (JK2017-Leap Motion [43]) and $t(58)=2.8, p=0.006$ ($1$-GDS [45]).

Focusing on the results with JK2017 (Leap Motion) [43] dataset, we observe a large HYPE$_\infty$ score gap between the two generators. Notably, DeepNAG achieves hyper-realism on this dataset: its fake samples look more realistic to humans than the real ones. These results correlate well with those in Section 4.1: on the Leap Motion dataset, DeepNAG significantly outperformed DeepGAN in reducing the recognition error (Table 1) and in some cases, DeepNAG reduced the recognition error to zero (Figure 2).

Similar to [51], we report a breakdown of the error on the real and fake samples. We observe higher fake errors with DeepNAG in all cases. Inline with Zhou et al.’s observation [51], real and fake errors track each other. This indicates participants become more confused when fake samples are particularly hard to distinguish from the real ones.

In summary, our study shows that it is harder for evaluators to distinguish DeepNAG’s synthetic samples from the real samples compared to those produced by DeepGAN. This trend holds regardless of the dataset and gesture modality. Additionally, DeepNAG not only outperformed DeepGAN in every experiment, but it also achieved hyper-realism on the Leap Motion dataset. These results correlate well with our data augmentation performance evaluations in Section 4.1 and are notable considering that DeepNAG and DeepGAN both use the same underlying generator.

5 CONCLUSION

We discussed modality-agnostic gesture generation with recurrent neural networks. We first presented DeepGAN, our GAN model for synthetic gesture generation across various datasets and gesture modalities. To reduce the training complexity, we formulated a novel loss function based on the dynamic time warping (DTW) algorithm and the average Hausdorff distance. Our loss function obviated the need for a separate discriminator network, and led to 12–17× faster training. We called this approach DeepNAG and evaluated it from two aspects. Our first evaluations focused on the use of either model towards data augmentation for improved gesture recognition. In these evaluations, DeepNAG outperformed DeepGAN, along with other synthetic gesture generators across various datasets and recognizers. Next, we evaluated the perceived quality of the synthetic samples produced by DeepGAN and DeepNAG using human evaluators. Our user study, which was based on the HYPE$_\infty$ benchmark and was conducted using Amazon Mechanical Turk, demonstrated that DeepNAG consistently outperformed DeepGAN in terms of the realism of the synthetic samples. Users confused DeepNAG’s samples with the real ones more frequently,
We thank Amir Gholaminejad, Rahul Sukthankar and Corey Pittman for the helpful discussions. We also thank the anonymous reviewers for their time and valuable feedback.

REFERENCES


VAEs consist of encoder and decoder networks. However, the goal of VAEs has previously been used to generate 2D sketches, to closely resemble the input data. The regularization term ensures that the learned latent space closely resembles the input data. The reconstruction term ensures that the reconstructed data closely resembles the input data. The regularization term ensures that the learned distribution of the latent space is as close to some known distribution as possible (typically the standard normal distribution). Assuming that $\phi$ and $\theta$ denote the trainable parameters of the encoder and decoder respectively, the following is the loss function that is minimized [5]:

$$\mathcal{L}(\theta, \phi; x, x) = -\mathbb{E}_{q_\phi(z|x)} \left[ \log p_\theta(x|z) \right] + D_{KL}(q_\phi(z|x) \| p(z))$$

reconstruction regularization

(7)

where $D_{KL}(\cdot \| \cdot)$ is the Kullback-Leibler (KL) divergence between two probability distributions. Equation 7 simply aims to minimize the reconstruction error as well as the KL divergence between the learned latent space and the standard normal distribution ($p(z) = N(0, 1)$). Note that in this formulation, class labels are not considered, which means one cannot control what sample class is produced for a given $z$. Sohn et al. [39] proposed conditional VAEs in which a conditioning criteria is applied to the input data $x$ as well as the latent vector $z$ similar to conditional GANs as described in Section 3.2.

A.2 Model Architecture and Objective Function

We iteratively designed our VAE’s overall architecture. Given our goal of training the generator of DeepGAN/DeepNAG in a VAE framework, we reused the aforesaid generator as the decoder in our VAE network. As for the encoder, we started with using the uDeepGRU model as the encoder. This way, our overall VAE network closely resembled that of DeepGAN’s. After running some preliminary experiments, we observed that the choice of the encoder architecture did not result in perceivable difference in the model’s performance. In fact, adding or removing layers in either the encoder or the decoder made little difference in the produced results, inline with what Bowman et al. [4] observed. We ultimately decided to carry on with an architecture similar to Figure 1.

We now discuss our proposed training objective function which can be used to train our RNN-based generator in a VAE framework. As previously mentioned, the VAE objective function typically consist of reconstruction and regularization terms. We can conveniently reuse the regularization term of Equation 7, as it simply ensures that the learned latent space follows the standard normal distribution. The reconstruction term, however, is domain-specific. Recall that the this term ensures that the output of the decoder (generator) closely resembles the input data. Conveniently, a differentiable measure that can be used for this purpose is sDTW. Thus, we propose the following loss function as the objective to minimize during the training our gesture generating VAE:

$$\mathcal{L}(\theta, \phi; x, x_c) = s\text{DTW}(x, G_{\phi}(x_c); f) + D_{KL}(q_\phi(z_c|x) \| p(z_c))$$

(8)

where $f$ is sDTW’s internal cost function and $z_c$ is the latent vector conditioned on the class label $c$. In simple terms, we define the reconstruction error as the sDTW dissimilarity between the input data and the output of the generator. As mentioned in Section 3.3, Cuturi and Blondel show that the computed sDTW value would be the expected value of dissimilarity between two time-series, over every possible alignment between them weighted by their probability under the Gibbs distribution [7]. This closely resembles the original reconstruction term of VAEs (Equation 7). Though VAEs have previously been used to generate 2D sketches, to
Figure 5: Synthetic gestures produced by our VAE-based generator trained on $1$-GDS [45] dataset. Note that most samples are noisy and lack visual fidelity. We further show overlaid rendering of synthetic samples from our VAE model (b – top), and real samples (b – bottom). Each overlay consists of 16 samples per class. Note the lack of variety in the synthetic results compared to the real samples.

As previously mentioned, changes in the architecture of our model made little difference in the produced results. Although training plots showed a steady decrease of the loss value and training converged, the generated results lacked visual quality and diversity. Most gesture trajectories were rather noisy. Again, we observed this trend regardless of the architecture of encoder or decoder networks, various hyperparameter settings, the choice of $f$ (we tried ED and COS) or even the gesture dataset. We additionally experimented with alternative reconstruction terms. Specifically, we experimented with the mean squared error (MSE) of both the Euclidean distance as well as the cosine similarity of gesture paths ($\hat{x}$) between the input and reconstructed samples. These alternate formulations performed worse than our sDTW-based reconstruction term.

Some samples produced by our VAE model when trained on the $1$-GDS dataset [45], along with overlaid samples for each of the real and synthetic data are depicted in Figure 5. These results show that the produced samples lack sufficient diversity when compared to the real samples. Given the visual quality of the results, we hypothesize that our proposed VAE framework is not suitable for training our generator to produce good synthetic samples.