THE COOPER UNION
ALBERT NERKEN SCHOOL OF ENGINEERING

# Alternative Architectures for Generative Adversarial Networks

by
Christopher Curro

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Engineering

September 2016

Professor Sam Keene, Advisor

THE COOPER UNION FOR THE
ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

Dean, School of Engineering Date

Prof. Sam Keene, Thesis Advisor Date

Ack

## Abstract

# Contents

# List of Figures

# Table of Nomenclature

# 1 Introduction

## 1.1 Machine Learning at Large

Machine learning in general encompasses all tasks where we utilize data to make some kind of descision. Classical examples include handwritten digit recognition, and flower species classification based on sepal width and petal length. As rough mathematical example, consider a sample of N pairs of examples drawn from larger population $\mathcal{P}$:

$$\mathcal{D} = \{(x_i, y_i) \colon i = 1, 2, \ldots, N\} \tag{1}$$

$$\mathcal{D} \subset \mathcal{P} \tag{2}$$

Our machine learning task may be to find some function $f(x)$ that estimates $y$ for any $(x, y)$ sample drawn from the population at large. To learn the function we might setup a loss function $l(y, \hat{y})$ to quantify how well our model performs on our sample $\mathcal{D}$. As a general rule, the loss function satisfies the following limit:

$$\lim_{\hat{y_i} \to y_i} l(y_i, \hat{y_i}) = 0 \tag{3}$$

In other words our goal is to find some $f(x)$ that minimizes the loss function $l(y, \hat{y})$ for all $(x_i, y_i)$ pairs in $\mathcal{D}$. This type of task is referred to as supervised learning, because we have a domain of inputs and a specified codomain of targets; there are other applications with unknown targets, these tasks are referred to as unsupervised learning tasks. With a few extra constraints we are able use a number of techniques to accomplish this supervised learning goal.

### 1.1.1 Differentiable parametric modeling

Let us restrict our study of functions $f$ specifically to functions that are differentiable and parametric in form, so from now on we refer to $f(x \mid \theta)$ where $\theta$ refers to the function's set of parameters. Using these new terms our goal is to find some $\hat{\theta}$ that satisfies the following:

$$\hat{\theta} = \arg\min_{\theta} l(y, f(x \mid \theta)), \ \forall x, y \tag{4}$$

For any problem of this type we would like to calculate gradient of $l$ with respect to each $\theta_i$, set them all to zero and solve for each $\theta_i$:

$$\frac{\partial l(x, f(x \mid \theta))}{\partial \theta_i} = 0, \ \forall i \tag{5}$$

However this is not always possible analytically because without convexity constraints we are not able to guarantee the existence of a unique minimum in the loss function.

In lieu of an analytic solution we can use a numerical optimization approach, and jointly optimize the $\theta_i$'s via a gradient descent algorithm. For example we could use this update rule, which we refer to as steepest descent:

$$\theta_i \leftarrow \theta_i - \eta \frac{\partial l(x, f(x \mid \theta))}{\partial \theta_i} \tag{6}$$

where $\eta$ is a the learning rate parameter, which controls how large our steps are towards the minimum.

### 1.1.2 Basis expansions

Let us consider our first supervised learning example. Consider a noisy sine-wave consisting of $k$ samples:

$$\mathbf{y} = \sin \mathbf{x} + \mathbf{n}, \ \mathbf{y}, \mathbf{x}, \mathbf{n} \in \mathbb{R}^k \tag{7}$$

We would like to find some $f(\mathbf{x} \mid \theta)$ that produces an estimate $\hat{\mathbf{y}}$. Let us consider the following functional form for $f$:

$$f(\mathbf{x}) = m\mathbf{x} + b, \ m, b \in \mathbb{R} \tag{8}$$

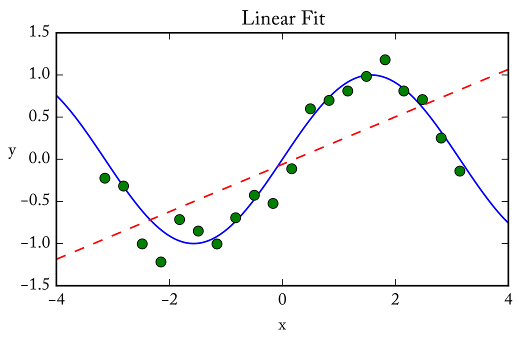

Figure 1: Linear regression of a noisy sine-wave.

Or we can rewrite this as a matrix multiplication by padding x in with a ones vector to create a $2 \times k$ matrix we refer to as $\Phi$, and combining $m$ and $b$ into a vector $\mathbf{w}$:

$$\hat{\mathbf{y}} = \mathbf{w}^T \Phi \tag{9}$$

We are able to compute $\hat{m}$ and $\hat{b}$ analytically, with a mean squared error loss function:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \sum_{i=1}^{k} (\hat{y}_i - y_i)^2 \tag{10}$$

Now let's substitute our model in for $\hat{y}$ and set the gradient to zero:

$$\frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} \left( \frac{1}{2} \sum_{i=1}^{k} (\mathbf{w}^T \Phi - y_i)^2 \right) = 0 \tag{11}$$

Carrying out the derivative we see:

$$\frac{1}{2} \sum_{i=1}^{k} (\mathbf{w}^T \Phi - \mathbf{y}) \Phi^T = 0 \tag{12}$$

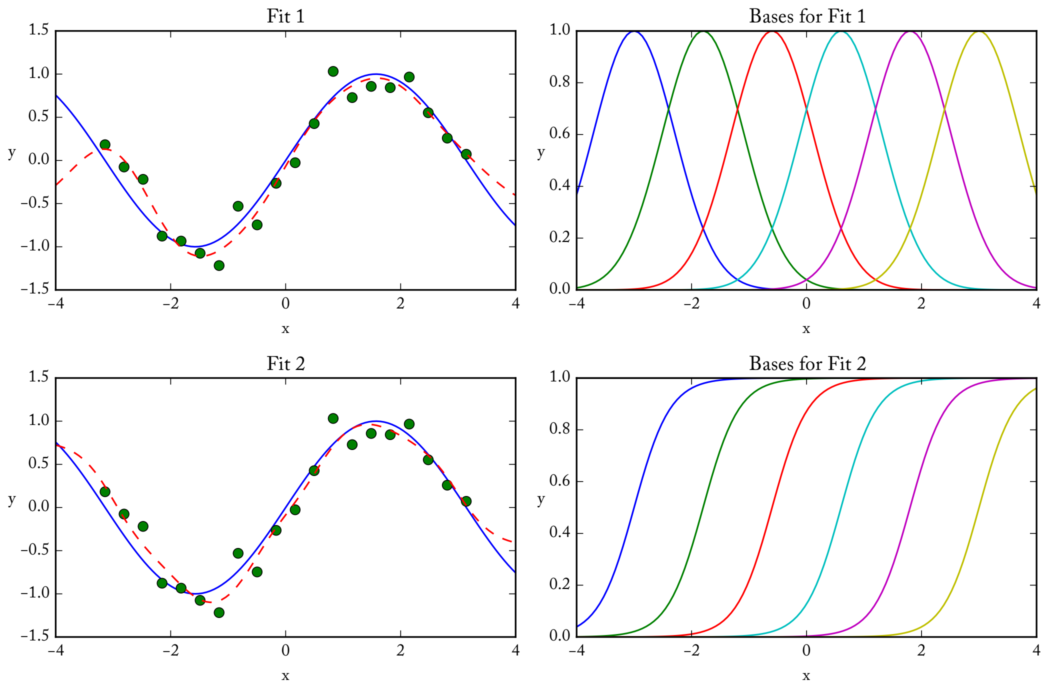

Figure 2: Linear regression of a noisy sine-wave with basis expansion

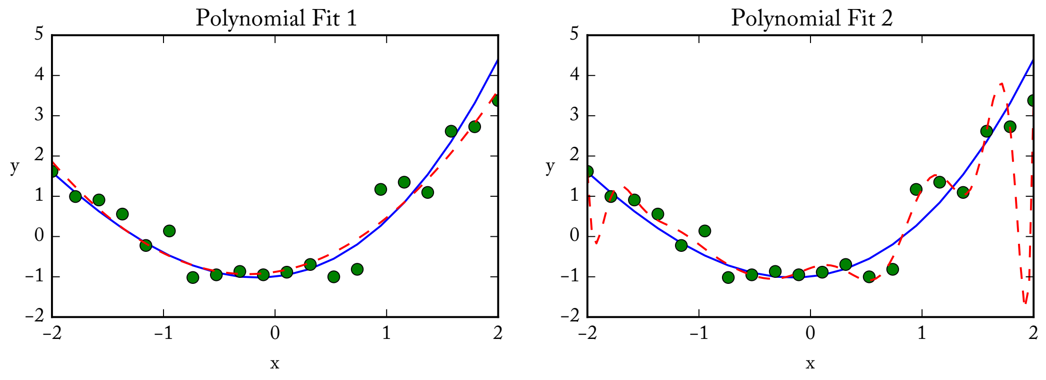

Figure 3: Polynomial Fit

### 1.1.3 Kernel methods

### 1.1.4 Curse of dimensionality

### 1.1.5 Over-fitting

## 1.2 Neural Networks

### 1.2.1 Neural networks as computational graphs

A neural network generally is a computational graph $G = (V, E)$ where the vertices $V$ correspond to nodes of computation, and the edges $E$ correspond to data flow paths connecting the computational nodes. We limit our discussion to feed-forward or directed acyclic graphs, in other words graphs where given a starting vertex $v$ we are unable to follow the directed edges away from it and return to $v$. With this limitation in place we are able to focus on stateless graphs, which treat input data examples independently of one another.

With the additional limitation that all computational nodes in the graph perform differentiable operations, and that some nodes are parametric, we are able to use automatic differentiation and a gradient descent like optimization algorithm to optimize the parameters of the graph against a differentiable loss function [1]. Automatic differentiation is an alternative to numeric and symbolic differentiation, that is efficient, accurate to machine precision, and scalable to arbitrary graphs consisting of elementary operations. The core tenet of automatic differentiation is that since graph vertices are differentiable we can perform the chain rule at each node to build up gradients of the full graph. Several machine learning frameworks, such as Theano and TensorFlow implement automatic differentiation, allowing us to specify the functional form of the graph and getting gradients at minimal cost [2, 3].

### 1.2.2 Fully connected neural networks

While we have introduced the idea that neural networks can be made up of arbitrary computational nodes, there are several common types of nodes that are used to build up a toolkit of functional forms at our disposal. First, let us a consider a simple inner product with signature:

$$\langle \cdot, \cdot \rangle \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \tag{13}$$

And functional form:

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=1}^{n} w_i x_i \tag{14}$$

We can interpret this inner product as a single node in our graph, with parameters $\mathbf{w}$ and input vector $\mathbf{x}$. While this functional form is nice, in that it's linear, and easy to define, it's limited by the fact that it brings the rich space $\mathbb{R}^n \times \mathbb{R}^n$ down to $\mathbb{R}$ which has limited representational power for problems of interest in machine learning. Instead we'll follow a common design pattern, and create an array of inner product nodes at a constant number of hops from the source node in the graph, or as we'll refer to from here on, at a constant depth. In this case the signature would be:

$$g \colon \mathbb{R}^{n \times m} \times \mathbb{R}^n \to \mathbb{R}^m \tag{15}$$

And functional form:

$$g\left(W, \mathbf{x}\right)_j = \sum_{i=1}^{n} w_{ij} x_i \tag{16}$$

which we recognize as ordinary matrix multiplication $W\mathbf{x}$, where $W \in \mathbb{R}^{n \times m}$. We call this collection of nodes at a common depth a layer. This type of matrix multiplication layer is generally referred to as a linear, dense, or fully connected layer.

Layers can be connected together with the goal of creating a more powerful model. If we feed one dense layer into another directly we have have a functional form:

$$\mathbf{y} = W_n \cdots W_3 W_2 W_1 \mathbf{x} \tag{17}$$

where the $W_i$'s are all compatible. However this approach is actually equivalent to a single matrix multiply $W\mathbf{x}$ where $W$ is the matrix product of the $W_i$'s, so we did not actually achieve our goal of higher representational power. Our model is still merely linear.

To address this issue the we introduce the addition of a non-linear transformation $f(\cdot)$ (known as an activation function) at the output of the matrix multiply. So instead our functional form would be:

$$\mathbf{y} = f(W_n \cdots f(W_3 f(W_2 f(W_1\mathbf{x}))) \cdots) \tag{18}$$

As long as $f(\cdot)$ is differentiable we are able to train our now highly non-linear model with our methods of automatic differentiation and gradient based optimization.

Finally we introduce a bias $\mathbf{b}$ so that each node in a layer is not constrained to intercept zero. Therefore our full functional form for a single fully connected layer with a non-linearity is:

$$\mathbf{y} = f\left(W\mathbf{x} + \mathbf{b}\right) \tag{19}$$

with $W \in \mathbb{R}^{n\times m}$ and $\mathbf{b} \in \mathbb{R}^m$.

A traditional activation function is the sigmoid function:

$$S(t) = \frac{1}{1+e^{-t}} \tag{20}$$

which maps $t \in \mathbb{R}$ (i.e the interval $[-\infty, \infty]$) to $S(t)$ on the interval $[0, 1]$. This can allow the interpretation of $S(t)$ as a probability. Using this activation function Cybenko proved the universal approximation theorem [4] which shows that neural networks of the form in Figure 4 can approximate any function with appropriate domain and codomains given a large of enough number nodes in the middle layer, generally referred to as a hidden layer.

Figure 4 shows a feed-forward network with one hidden-layer. The outputs of the hidden layer are referred to as latent variables and are a new type of representation

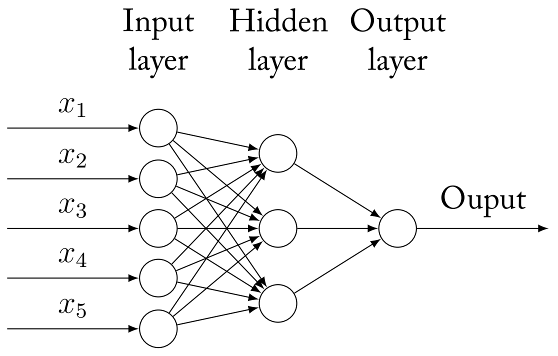

Figure 4: A single hidden layer feed-forward neural network. Each directed connection has a scaling factor $w_{ij}$ associated with it; these entries make up the $W$ matrix. Each circle, now referred to as a neuron, performs the sum and activation functions corresponding to the matrix multiply, and mapping by $f(\cdot)$ in Equation 19. Layers in the middle of the network are referred to as hidden layers because they are not directly observable — instead these are latent variables.

for the input data $\mathbf{x}$. With each hidden-layer added to the network increasingly complex data representations may be available, but the networks will also tend to over-fit their training data as the number of parameters increase. In order to combat over-fitting, and increase the generalization performance of the network new types of architectures have been proposed.

#### 1.2.3 Convolutional neural networks

#### 1.2.4 Convolutions with holes

#### 1.2.5 Tranposed convolutions

#### 1.2.6 Additional techniques

##### 1.2.6.1 Dropout

Various teams have demonstrated that in deep networks neurons can learn complex co-adaptation schemes; that is deep neurons respond to “mistakes” in shallower neurons. In order to prevent co-adaptation, so that each neuron learns a meaningful representation, the dropout scheme has been proposed [5–7]. Dropout is a masking process. In order to apply dropout to a layer during training, a binary mask is applied across neurons. The mask is determined by sampling a Bernoulli distribution with a parameter $p$ corresponding to the probability that a neuron will be masked. When a neuron is masked its output is considered fixed at zero. When the network is used for evaluation no masks are applied and all neurons are connected.

#### 1.2.6.2 Batch normalization

#### 1.2.6.3 Rectified linear units

Various activations have been proposed on the basis of similarity to the potential activations in actual biological neurons in human eyes. Recently the rectified linear unit (ReLU) has demonstrated significant performance increases for network generalization and increased training speed [7, 8]. The ReLU activation is defined as $\max(0, x)$. In other words a ReLU activation forwards along any positive inputs and sets and negative inputs to zero.

#### 1.2.6.4 Exponential linear units

#### 1.2.6.5 Residual networks

#### 1.2.6.6 Minibatch training

Traditionally there were two different approaches to optimizing neural network parameters, the online approach and the batch approach. In the online approach, the parameters of the network are updated after each exposure to a training example and gradient calculation. In the batch approach, the network is exposed to all of the training examples, the gradients are accumulated and then the network's parameters are updated. This was long believed to be the better approach, as the accumulated and averaged gradient was more likely to be an estimate of the "true" gradient of the network towards an optimized solution. It was later shown that, while the batch mode may give a better estimate of the gradient, due to the noise and its stochastic nature, the online approach actually leads to a faster convergence time, in terms of number of examples [9]. This is because in the stochastic approach the optimizer is less likely to get stuck in local minima. An alternative approach that recently has become popular is the mini- batch approach. In this approach some number of examples, say $N$, are exposed to the network, the gradients are accumulated, and the parameters are updated. In this case $N$ is much less than the total number of training examples available. This approach, with serial computational resources really only represents a decrease in training speed, because it is fairly similar to the batch approach. The reason this approach has become popular lately is that with the parallel resources afforded by modern high performance graphics processing units (GPUs) the increase in example-wise training time becomes a decrease in wall-time to train.

#### 1.2.6.7 Intelligent parameter initialization

Kaiming et al. have demonstrated an improved parameter initialization technique specifically designed for neurons with ReLU activations [10]. This approach derives a method that enables extremely deep models comprised of ReLU neurons to converge rather than stall, as they would with other initialization schemes. The initial parameters for the convolutional layers are drawn from a zero mean normal distribution with a standard deviation of $\sqrt{2/n_l}$. Where $n_l = k^2c$. This corresponds to the number of connections in the response for $k \times k$ kernels processing $c$ input channels.

##### 1.2.6.8 ADAM optimizer

##### 1.2.6.9 Style loss

### 1.3 Generative Models

#### 1.3.1 Generative models at large

#### 1.3.2 Generative adversarial networks

#### 1.3.3 Deep convolutional generative adversarial newtorks

#### 1.3.4 Additional techniques for traning generative adversarial networks

## 2 System Description

### 2.1 Energy distance matching

### 2.2 Kernel based moment matching

### 2.3 Convolutions with holes

### 2.4 Style Loss term

## References

[1] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, “Automatic differentiation in machine learning: a survey,” *CoRR*, vol. abs/1502.05767, 2015. [Online]. Available: http://arxiv.org/abs/1502.05767

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, “Theano: A cpu and gpu math compiler in python,” in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 3 – 10.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[4] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[6] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[7] G. E. Dahl, T. N. Sainath, and G. E. Hinton, “Improving deep neural networks for lvcsr using rectified linear units and dropout,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8609–8613.

[8] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier networks,” in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, vol. 15, 2011, pp. 315–323.

[9] D. R. Wilson and T. R. Martinez, “The general inefficiency of batch training for gradient descent learning,” *Neural Netw.*, vol. 16, no. 10, pp. 1429–1451, Dec. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(03)00138-2

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[13] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1139–1147.

[14] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[15] S. Geman, E. Bienenstock, and R. Doursat, “Neural networks and the bias/variance dilemma,” *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[16] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[17] J. Luo, “Frequently asked questions,” Hatfield Marine Science Center, 2014. [Online]. Available: https://www.kaggle.com/c/datasciencebowl/data

[18] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. [Online]. Available: http://dx.doi.org/10.1007/BF00344251

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[20] Y. Nesterov *et al.*, "Gradient methods for minimizing composite objective function," 2007.

[21] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4*. Morgan Kaufmann, 1992, pp. 950–957.

[22] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[23] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016. [Online]. Available: http://arxiv.org/abs/1606.03498

[24] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

[25] E. L. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," *CoRR*, vol. abs/1506.05751, 2015. [Online]. Available: http://arxiv.org/abs/1506.05751

[26] Y. Li, K. Swersky, and R. S. Zemel, "Generative moment matching networks," *CoRR*, vol. abs/1502.02761, 2015. [Online]. Available: http://arxiv.org/abs/1502.02761

[27] J. Johnson, A. Alahi, and F. Li, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016. [Online]. Available: http://arxiv.org/abs/1603.08155

[28] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[29] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow, "Adversarial autoencoders," *CoRR*, vol. abs/1511.05644, 2015. [Online]. Available: http://arxiv.org/abs/1511.05644

[30] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully

connected crfs," *CoRR*, vol. abs/1412.7062, 2014. [Online]. Available: http://arxiv.org/abs/1412.7062

[31] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, 2015. [Online]. Available: http://arxiv.org/abs/1511.07122

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[33] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," *CoRR*, vol. abs/1406.2661, 2014. [Online]. Available: http://arxiv.org/abs/1406.2661

[34] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–48.