My First Neural Networks:

Classification, Convolutions, and Autoencoders

Follow along at: https://github.com/dwgb93/SIAM-Neural-Nets

Dylan Bates

April 22, 2021

North Carolina State University

NC STATE UNIVERSITY

Introduction

Machine Learning

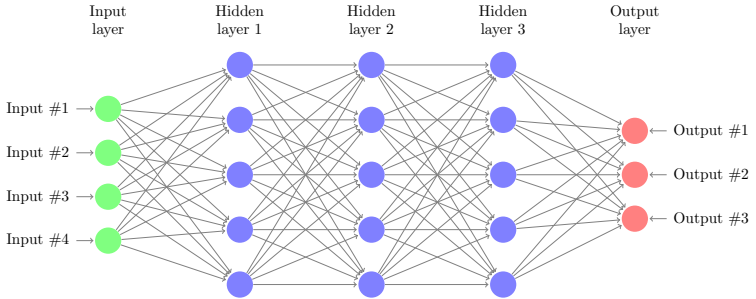
Machine Learning is the study of algorithms that let a computer learn from experience instead of being coded step-by-step

- Supervised Learning: Classification, Regression
  - What we'll be focusing on today!
- Unsupervised Learning: Clustering, Generation
  - We'll touch on this at the end
- Reinforcement Learning: Control synthesis, Games
  - This is my research!

Neural Networks

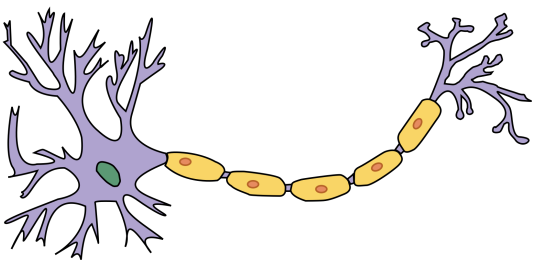
- A popular tool used for machine learning (deep learning)
- General function approximator
- Just needs data! (and an error function $J(\theta)$)

We can update the parameters of our network using gradient descent:
$\theta_{t+1} = \theta_t - \eta J(\theta)$



History and Motivation

Neural networks are loosely inspired by neural connections in the brain.



Neural networks are inspired by this

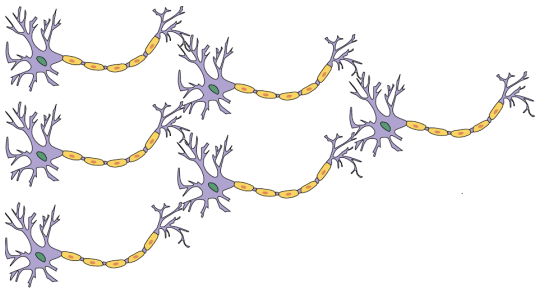
- layers of neurons
- connected with weight matrices
- activation function

$$a_j^\ell = \sigma\left(\sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell\right)$$

It's not *quite* like this.

History and Motivation

Neural networks are loosely inspired by neural connections in the brain.



It's not *quite* like this.

Neural networks are inspired by this

- layers of neurons
- connected with weight matrices
- activation function

$$a_j^\ell = \sigma\left(\sum_k w_{jk}^\ell a_k^{\ell-1} + b_j^\ell\right)$$

$$a^\ell = \sigma\left(w^\ell \cdot a^{\ell-1}\right)$$

Universal Approximation Theorem

Theorem
Let I_n be the n-dimensional unit cube $[0,1]^n$. Then finite sums of the form

$$F(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(w_j^\top x + b_j\right)$$

are dense in (I_n). That is, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a neural network $F(x)$ where

$$|F(x) - f(x)| < \varepsilon \text{ for all } x \in I_n.$$

Learning with Neural Networks

There is another theorem that tells us we can arbitrarily approximate any Lebesgue integrable function using a neural network with fixed width.

- These theorems do not tell us how many neurons we need
- They do not tell us how to tune the parameters to approximate a function
- We turn to general function minimizers: gradient descent

For gradient descent to work, we need to calculate the gradients of our function with respect to each parameter.

That means we also need a *loss function*.

Classification

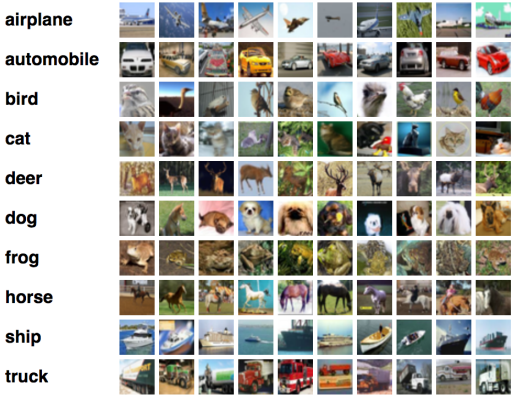
Supervised Learning

Arguably the most well-known type of machine learning, supervised learning involves computers learning to classify labeled data.

This can be something like identifying what's in an image, providing a medical diagnosis, detecting spam, identifying what each pixel in an image corresponds to.

Examples

Classification of handwriting digits.



Colour images in 10 different classes.

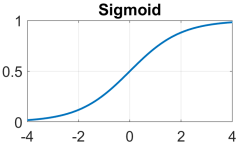
Backpropagation

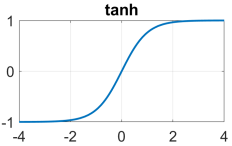
Used calculate the gradients to automatically minimize the difference between neural network and a function. This is just repeated chain rule.

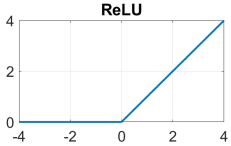
Common error functions:

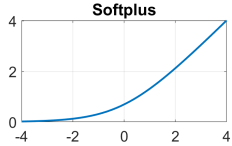
- Regression: Mean squared error
- Categorical: Crossentropy (measures differences between distributions)

Common activation functions:





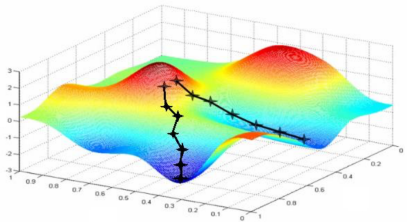




Gradient Descent

Once we know the gradients of each of the parameters, we take a step in the direction of steepest descent by updating each of the parameters with $w^\ell := w^\ell - \eta \nabla_{w^\ell} J$

- Step size controlled by learning rate
- Batching inputs helps
- Often converges to *local* min



There are several techniques we use for regularization, to smooth the loss function, escape plateaus, and converge as quickly as possible. Let's try it!

Convolutions

Train-Test Split

One important thing that all machine learning algorithms must consider is how well the model generalizes.

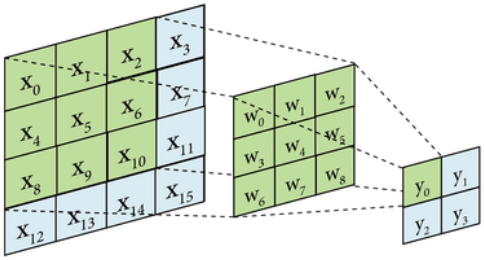
We can split the dataset into a training, validation, and test set:

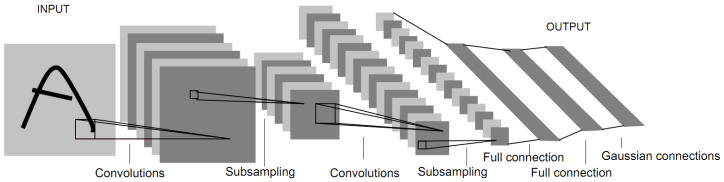
- Training set: data used to train the model
- Validation set: used for hyperparameter optimization
- Test set: brand new data to evaluate final model

Convolutional Neural Networks

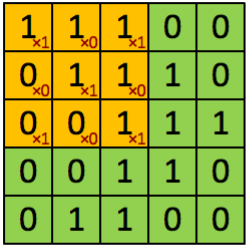
Instead of connected layers, CNNs deal with localized features of image data.

- Feature maps are multiplied by sections of each image
- Features are learned automatically
- Fewer parameters, but TONS of connections
- Very slow to learn, need GPU

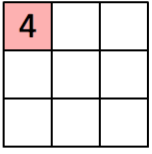




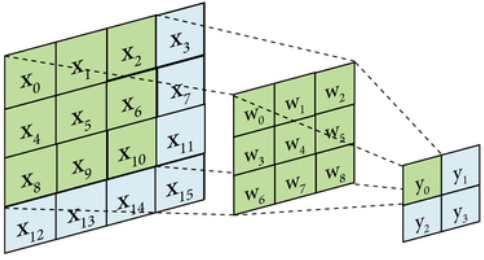
Convolving Feature Maps



Image



Convolved Feature

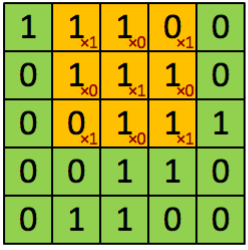


The size of your input and feature maps determine the size of your output.

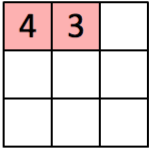
Also strides

Let's try it!

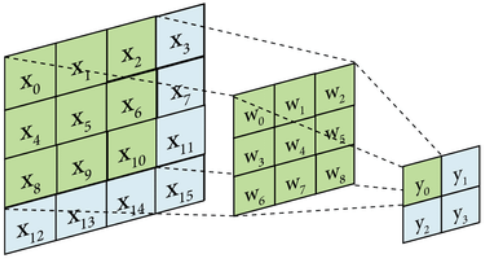
Convolving Feature Maps



Image



Convolved Feature



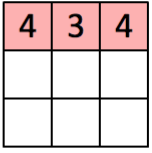
The size of your input and feature maps determine the size of your output.

Also strides

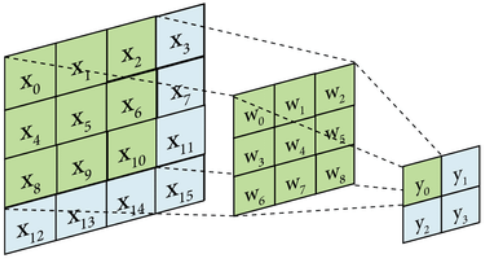
Let's try it!

Convolving Feature Maps

Image



Convolved Feature

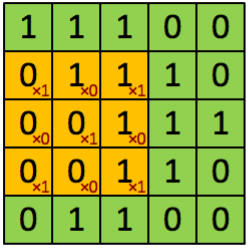


The size of your input and feature maps determine the size of your output.

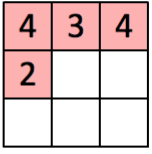
Also strides

Let's try it!

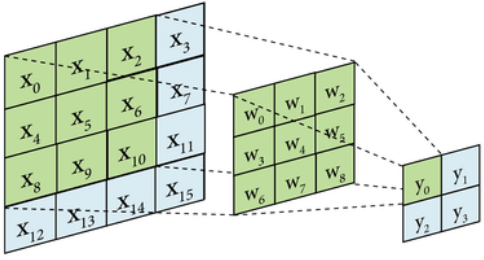
Convolving Feature Maps



Image



Convolved Feature

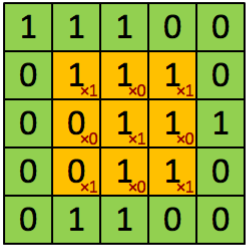


The size of your input and feature maps determine the size of your output.

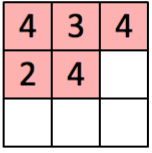
Also strides

Let's try it!

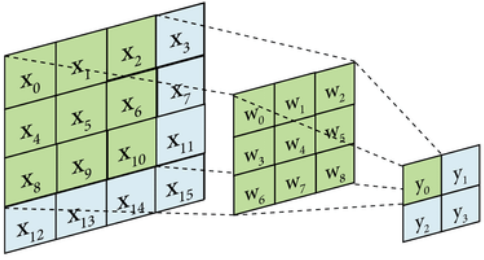
Convolving Feature Maps



Image



Convolved Feature

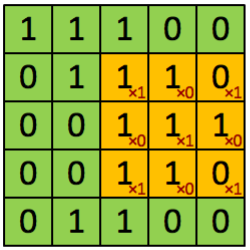


The size of your input and feature maps determine the size of your output.

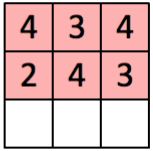
Also strides

Let's try it!

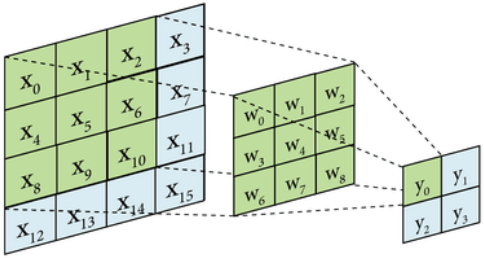
Convolving Feature Maps



Image



Convolved Feature

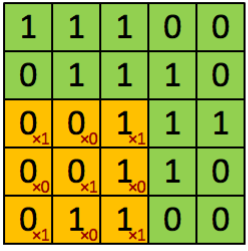


The size of your input and feature maps determine the size of your output.

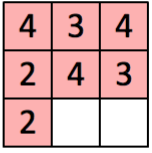
Also strides

Let's try it!

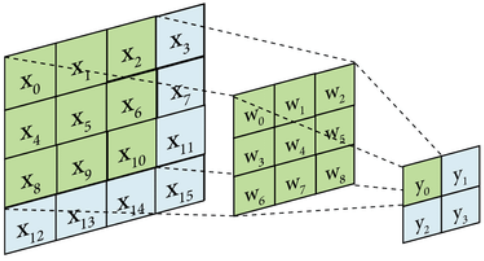
Convolving Feature Maps



Image



Convolved Feature

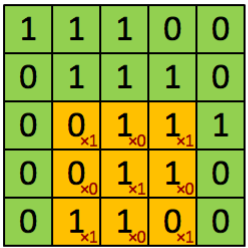


The size of your input and feature maps determine the size of your output.

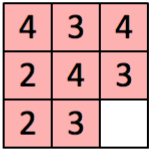
Also strides

Let's try it!

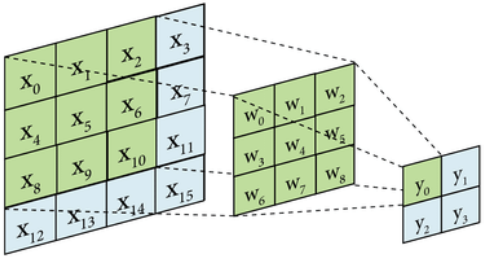
Convolving Feature Maps



Image



Convolved Feature

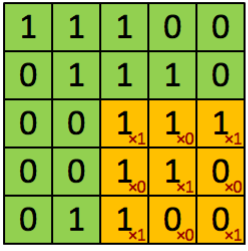


The size of your input and feature maps determine the size of your output.

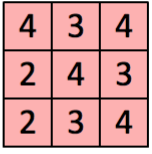
Also strides

Let's try it!

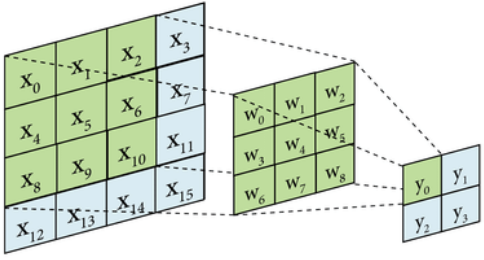
Convolving Feature Maps



Image



Convolved Feature



The size of your input and feature maps determine the size of your output.

Also strides

Let's try it!

Autoencoders

Unsupervised Learning

The other main type of machine learning involves unlabeled data. Common tasks include clustering data and using autoencoders to find lower-dimensional representations of the data.

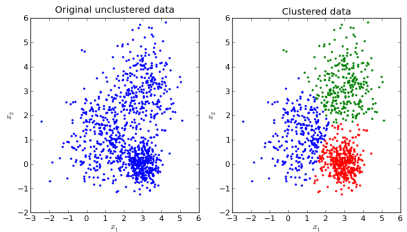


Figure 1: Clustering data points.

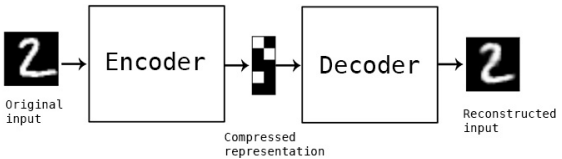
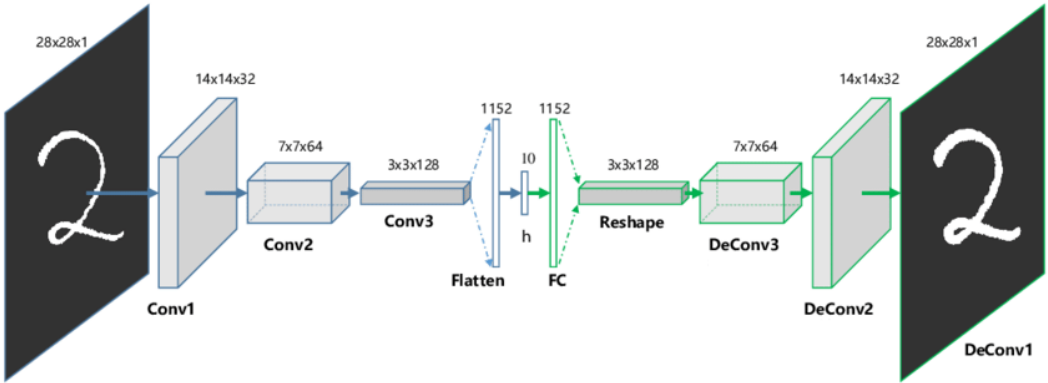


Figure 2: Encoding MNIST digits in a low-dimensional latent-space might make them easier to classify.

Autoencoders

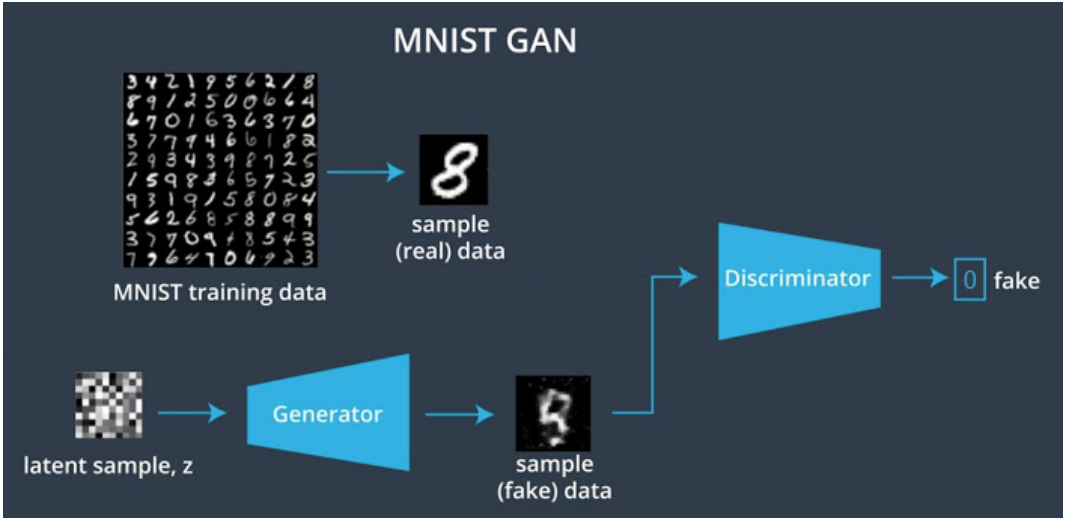
Encoder and decoder can be fully-connected nets, convolutional nets, or both.



Let's try it!

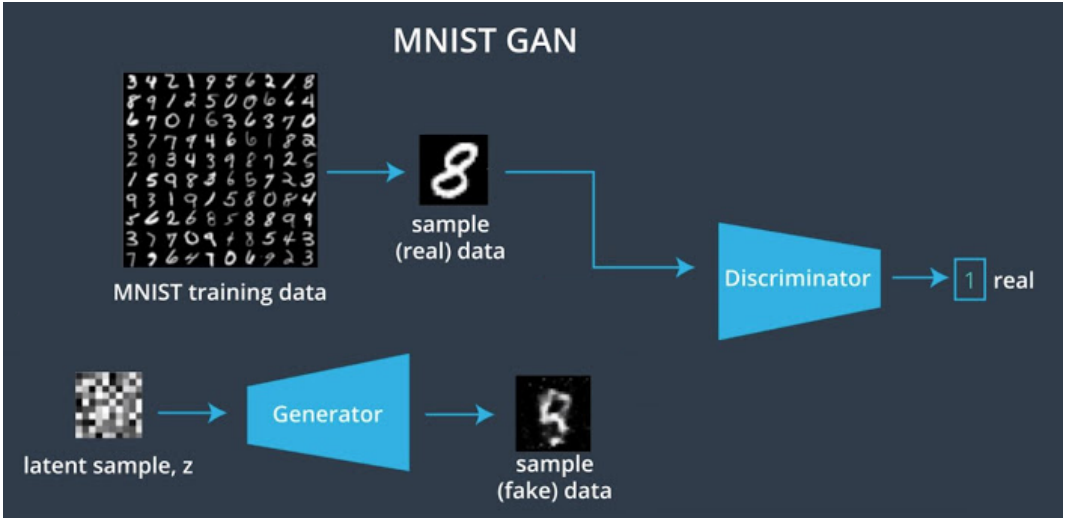
Generative Adversarial Networks

By exploiting the latent space, we can teach a network to generate new data.



Generative Adversarial Networks

By exploiting the latent space, we can teach a network to generate new data.



Examples

Figure 3: These people do not exist. https://thispersondoesnotexist.com/

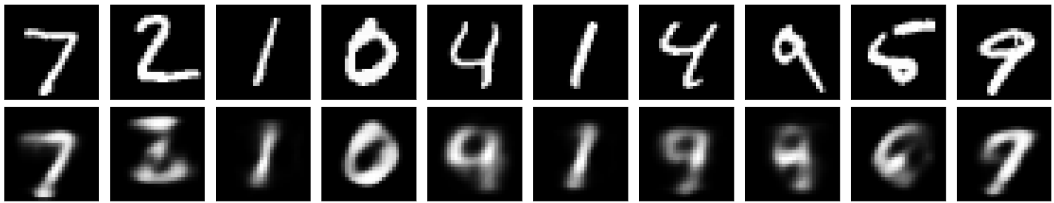
Results - Fully Connected

Layers	Parameters	Epochs	Accuracy	Test Acc.	Training Time
784-128-10	102k	5	89.05%	90.06%	15s
784-128-10	102k	20	**92.84%**	93.18%	60s
784-16-10	12.7k	10	89.89%	90.57%	20s
784-16-16-10	13.0k	10	92.77%	**94.50%**	30s

Results - Convolutional

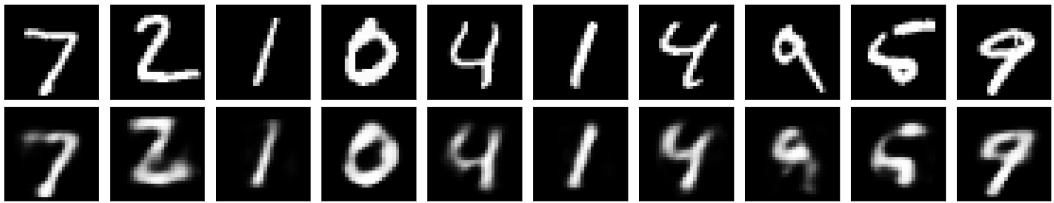
Layers	Parameters	Epochs	Accuracy	Test Acc.	Training Time
2	34.8k	10	99.09%	99.06%	30s
LeNet-5	61.7k	20	98.59%	98.54%	50s
LeNet-5+	61.7k	20	**100%**	99.03%	51s
SimpleNet-13	5.5M	10	99.90%	99.49%	380s
SimpleNet-13 Smol	1.3M	10	99.94%	**99.60%**	234s

Results - Autoencoders



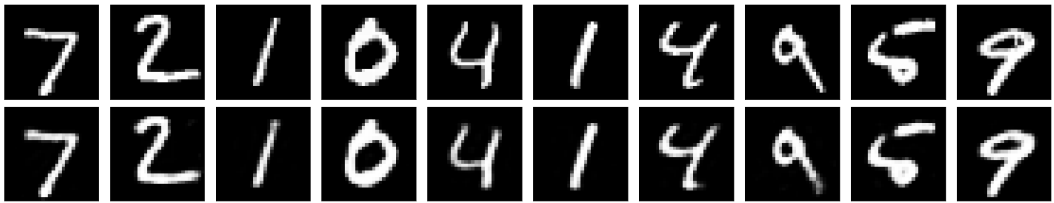
8 dimensional latent space

Results - Autoencoders



16 dimensional latent space

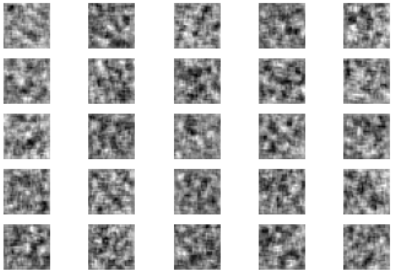
Results - Autoencoders



64 dimensional latent space

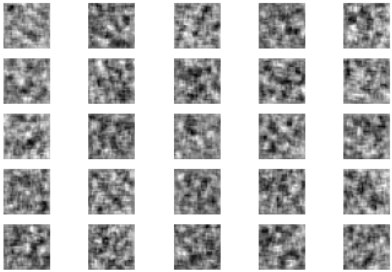
Results - GANs

Working well



Epoch: 0 This starts as just random noise

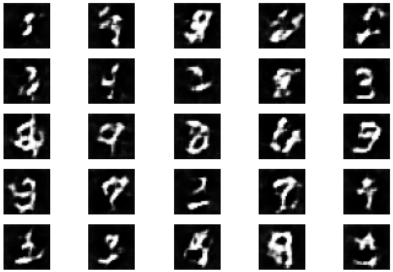
Working poorly



Epoch: 0

Results - GANs

Working well



Epoch: 1000

Working poorly

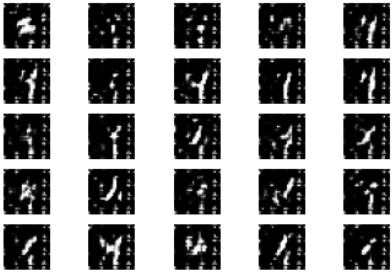
Epoch: 500

Results - GANs

Working well

Epoch: 2000

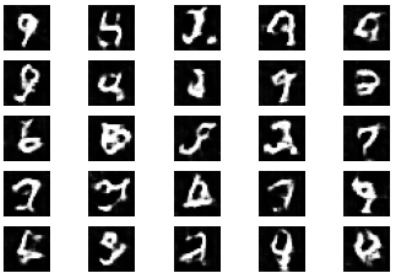
Working poorly



Epoch: 1000

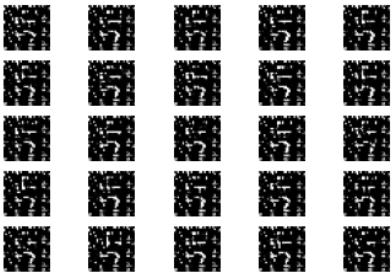
Results - GANs

Working well



Epoch: 3000

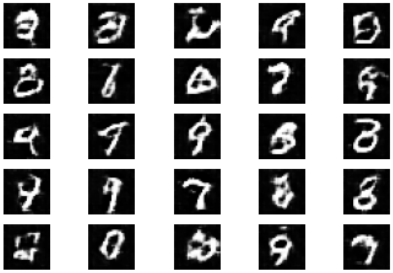
Working poorly



Epoch: 1500

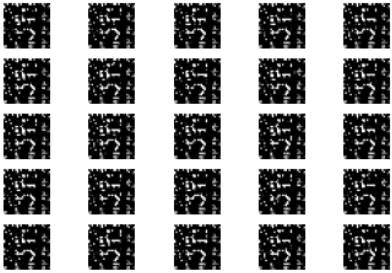
Results - GANs

Working well



Epoch: 4000

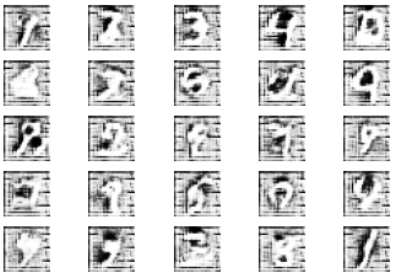
Working poorly



Epoch: 2000

Results - GANs

Working well



Epoch: 5000...oops

Working poorly

Epoch: 2500

Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 1000

Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 2000

Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 3000

Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 4000

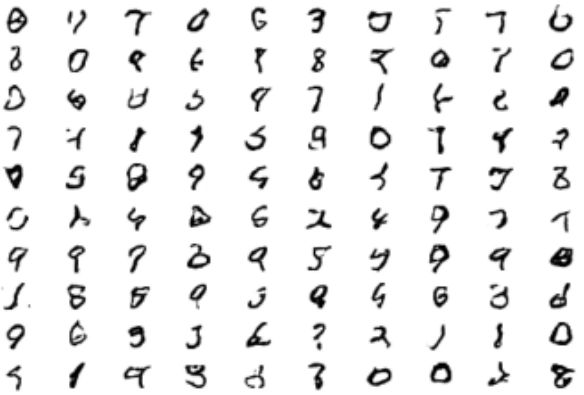
Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 5000

Results - GAN

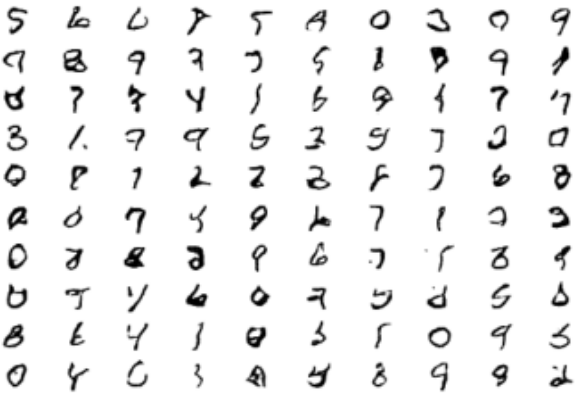
I trained one more GAN, which worked a little better.



Epoch: 6000

Results - GAN

I trained one more GAN, which worked a little better.



Epoch: 7000

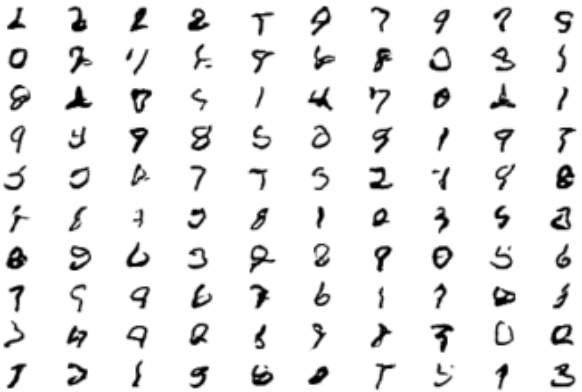
Results - GAN

I trained one more GAN, which worked a little better.

Epoch: 8000

Results - GAN

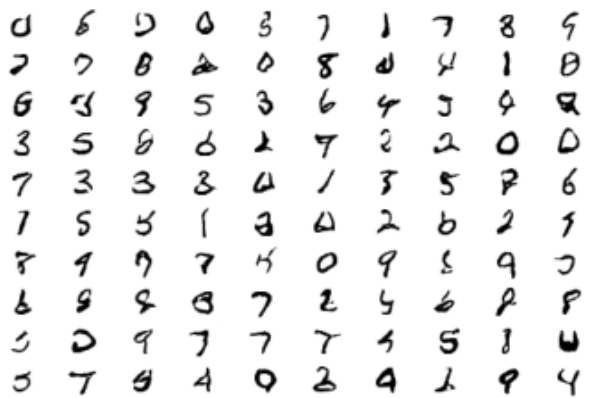
I trained one more GAN, which worked a little better.



Epoch: 9000

Results - GAN

I trained one more GAN, which worked a little better.



Epoch: 10000

Thank you!

Questions?

Notebooks and slides available at:
https://github.com/dwgb93/SIAM-Neural-Nets