Expertise
and insight
for the future

Yasamin Salami

Managing adversarial networks via a web interface

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

21 August 2018

Author Title Number of Pages Date	Yasamin Salami Managing adversarial networks via a web interface 29 pages 30 August 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Mike Solomon, Instructor Antti Piironen, Supervisor

Recently generative adversarial networks are becoming the main focus area of machine learning. It was first introduced by Ian Goodfellow in 2014. The structure of GAN consists of two neural networks, which are constantly competing with each other: generator and discriminator. One learns to produce fake samples similar to the training data and the other tries to differentiate between fake and real data samples. The goal is to train the discriminator using samples from a known dataset until it reaches a good level of accuracy. Training GANs are challenging and can be time taking, but the result is quite impressive. GANs are being used in many industries, for instance, games, films, and medicine and for any distribution of data such as images, music and speech.

Keywords	GAN, Unsupervised learning, Generator, Discriminator

Acknowledgements

This thesis would not have been possible without the involvement of several people.

First of all, I would like to thank my supervisor Antti Piironen for his time and great advice during this time.

Secondly, I want to thank my dear friend and instructor, Mike Solomon who suggested this amazing topic and has been a great help all along this road.

Thirdly, I want to thank my family and friends, especially my parents, who have always been supporting me and believing in me. Without your support this thesis would not have been done.

Thank you!

Table of Contents

List of Abbreviations

AI	Artificial intelligence
GAN	Generative Adversarial Networks
CGAN	Conditional Generative Adversarial Networks
TF	TensorFlow
MSE	Mean squared error

1 Introduction

Artificial intelligence has been making impressive progress in different areas, such as technology, business and science every day. Many of our daily experiences are affected by AI and machine learning. Siri, Google now and Alexa are good examples of this.

The main difference between humans and machines is that humans can learn from the past experiences, while machines need to be told what to do and follow instructions. But there is a way to make computers also learn from the past experiences and that is precisely what machine learning is about. And for computers the past experience is called data.

The basic concept of machine learning is that computers do not only fetch and display data but also make decisions based on this data. This concept includes different types of fields: from spam filters and automated transportation to medical diagnosis. There has been an increasing demand for the computers to learn from data and make predictions and decisions based on those data.

Generative adversarial networks (GANs) were introduced first time in 2014 by Ian Goodfellow and a few other researchers at Montreal University. According to an AI research director Yann Lecun, this was one of the most interesting ideas in machine learning in the past ten years [1]. GANs have a great capacity to be developed and led to success in near future, because they can easily learn to imitate any data distribution such as images, music, emails and many other categories.

Generative adversarial networks are a recently introduced class of generative models, designed to produce realistic samples [2]. The basic idea of generative adversarial networks is their way of solving a generative modeling problem.

In today's world GANs are being used in different areas. The primary reason for this is GANs are filling a huge gap: not only can GANs make decisions and predictions based on data like neural networks, but they also can learn by themselves and generate data that never existed. Therefore, they can thoroughly and independently study the data or practically any other object and consequently create new versions of those objects.

This exposes the reason why GANs are being so much used in different industries nowadays, from health care to retail and game industry. There has been an increasing need for the computers to learn from data and apply that knowledge to make predictions and decisions and this is while the algorithm will continue to improve.

On the basis of a number of problems that are possible to solve through GANs, the following two research questions were raised to be answered in this thesis:

1) How long do GANs need to be trained to output accurate results?
2) To what extent is it possible to improve GANs performance?

This thesis aims to review and investigate recent development of GANs and present behaviors that occur during training a GAN in practice, especially in a web browser, and lists several solutions to avoid these problems.

The primary focus of this thesis is on neural networks, which are inspired by the way human brain works; as well as generative adversarial networks, which are considered to be an approach to unsupervised machine learning. Unsupervised machine learning is based around the idea that it should be possible to give machines access to the data and let them learn from those data by themselves. The term *generative* in computer vision means that a model can create new objects from scratch, which might have never existed before.

2 Functioning of GANs

In machine learning, the algorithms can be divided into two main groups based on the way they learn about the data: supervised and unsupervised learning.
In supervised learning we “teach the model” and then with that knowledge have it predict the future samples but for that we need a large dataset containing features as well as its corresponding label [3].

GANs are used in unsupervised machine learning. Unsupervised learning is where you let the model to work on its own and discover information that may not be visible to the human eye. Unsupervised learning uses machine-learning algorithms that draw conclusions on unlabeled data. Therefore, it creates a less controllable environment as the machine is creating outcomes for us.

2.1 Generator vs. Discriminator

In order to understand GANs, it is good to know how generative algorithm and discriminative algorithm work and how they differ from each other. These two are the main components of a GAN.
The generator tries to produce new unreal data, which is similar to the real ones while discriminator has to compare the generated data with the real ones, check their legitimacy and labels the difference.

Figure 1 shows both neural network’s task:

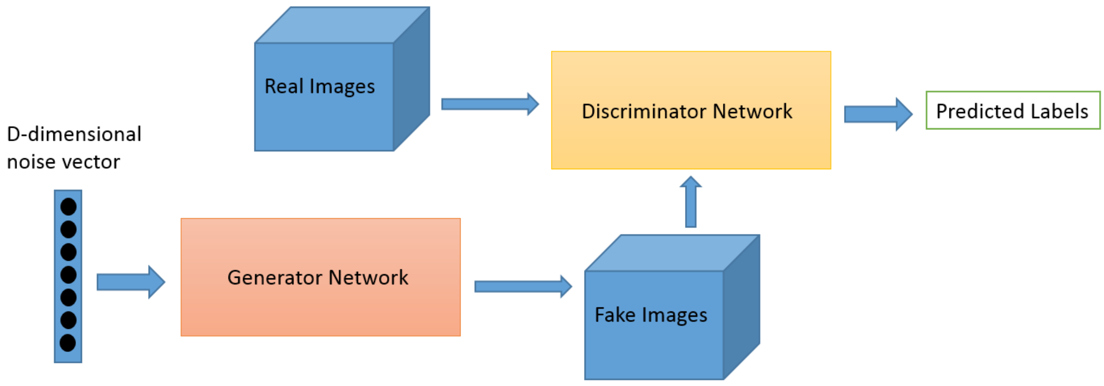

Figure 1. Comparison between the generator's task and the discriminator's task [4].

The idea state is when the generator knows how to generate realistic fake samples and the discriminator can distinguish perfectly if it is fake or real.

Here are the steps in a GAN:

- The generator generates new data instances such as images.
- This generated image will be given to the discriminator for an evaluation beside a series of images from the real dataset.
- The discriminator takes both real and fake images and returns a probability between 0 and 1, with 1 representing a true case and 0 representing a fake one [5].

Discriminative model unlike a generative model does not pay attention to how data were generated, it only categorizes the data and give signals. The goal of the whole process is to train the discriminator using samples from a known dataset in order to reach some level of accuracy. In other words, a model continuously tries to fool another model, until it can do so with ease. At that point, it can generate authentic looking data. GANs are designed to mimic any distribution of data such as music, images and speech.

3 TensorFlow

TensorFlow is the most popular machine learning library in the world, which was originally developed by researchers from Google's AI organization. Almost every single Google products uses machine learning in some way, whether its image search, image captioning, translation or recommendations. Google needs machine learning to take advantage of their large data sets to give users the best experience [6].

TensorFlow can be installed via python package manager using below command:

```
Install Tensorflow via pip
```

It can easily be include it in our code by importing it in the beginning:

```
import * as tf from '@tensorflow/tfjs'
```

3.1 Creating and training the model

The next step is to define a model for linear regression. Models are a set of layers and in a sequential model, the outputs of one layer are the inputs of the next layer.

```
const model = tf.sequential();
```

After that, training the model starts:

```
model.add(tf.layers.dense({units: 100, activation: 'relu', inputShape: [10]}));
model.add(tf.layers.dense({units: 1, activation: 'linear'}));
  model.compile({optimizer: 'sgd', loss: 'meanSquaredError'});

  const xs = tf.randomNormal([100, 10]);
  const ys = tf.randomNormal([100, 1]);
  xs.print();
  ys.print();
```

At the beginning of the App.vue file a templateis defined using vue.js. This template contains a div where the data should show.

```
<template>

  <div id="app">

    <h1>TensorFlow training </h1>

    <div v-if="tfProgress.epochs !== 0">

      <h2>Epoch: {{ tfProgress.epochs }}, loss
{{ tfProgress.loss.toFixed(3) }}</h2>

    </div>

    <div v-if="tfProgress.epochs === 0">

      <h2>Waiting for data...</h2>

    </div>

  </div>

</template>
```

An interesting aspect is that there is conditional rendering in Vue.js, which makes updating the HTML much easier. It is possible to use <div v-if> to insert/remove <h2> based on if the value of its expression is true or not. Also the single-file components in vue js is beneficial as the styles and JavaScript are all in one file with a .vue extension instead of separating them into separate files. This is very useful because in component-based system each component is a single concern, not the three technologies it is built with.

In order to get this running in the browser the latest version of the node needs to be installed and then the following command is used:

```
npm run dev
```

After that, it will be seen that the project is running at port 8080. Figure 2 shows training of TensorFlow in the browser:

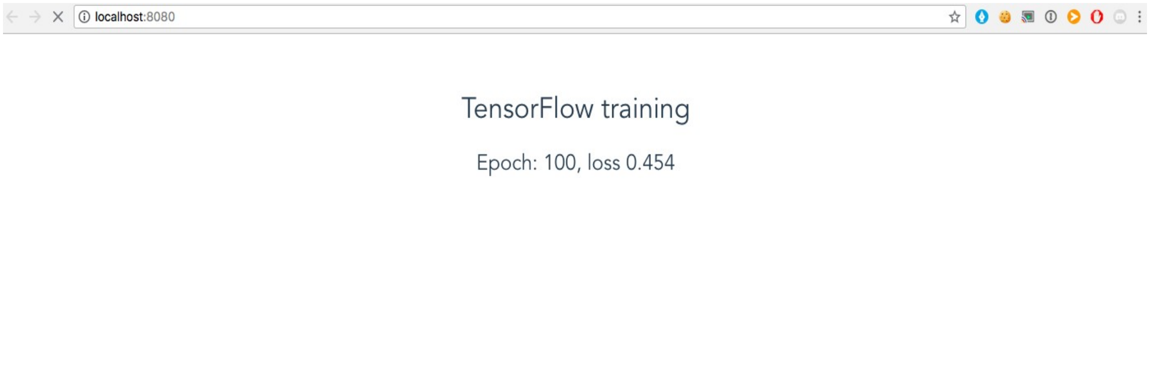

Figure 2. Training TensorFlow in the browser

As stated earlier in this chapter, TensorFlow is a primary tool that many large companies are using for their machine learning working in all of their products in some ways. This library is built to scale so that it can run on multiple CPUs or GPUs and even mobile operating systems with several languages such as python, Java and C. It is much easier to implement the generator and discriminator networks using TensorFlow layers.

2 Improve GAN performance

As mentioned in chapter 2, generative adversarial networks are used in unsupervised learning which means data comes in with no labels. Although, there are a considerable number of new training strategies applying to the GAN's framework. Unlike most people might think in generative models, the model is not require to generalize any kind of prediction to new data or to be able to learn everything well enough without using any labels. In fact, adding labels to the data means to break it up into categories, which also results in improving the performance of GANs [7].

2.1 Conditional generative adversarial network (CGAN)

In a conditional generative adversarial network, labels give a head start to GAN for what to search. It could also be that the visual system is more sensitive to these labels; hence, the generated images are continuously improving. In a nutshell, In CGAN a fake example with a specific characteristic or condition will be generated. To add such a condition to both the generator and the discriminator a vector y is simply fed to both networks:

G (z,y), D (x,y)

Figure 3 shows the architecture of CGAN:

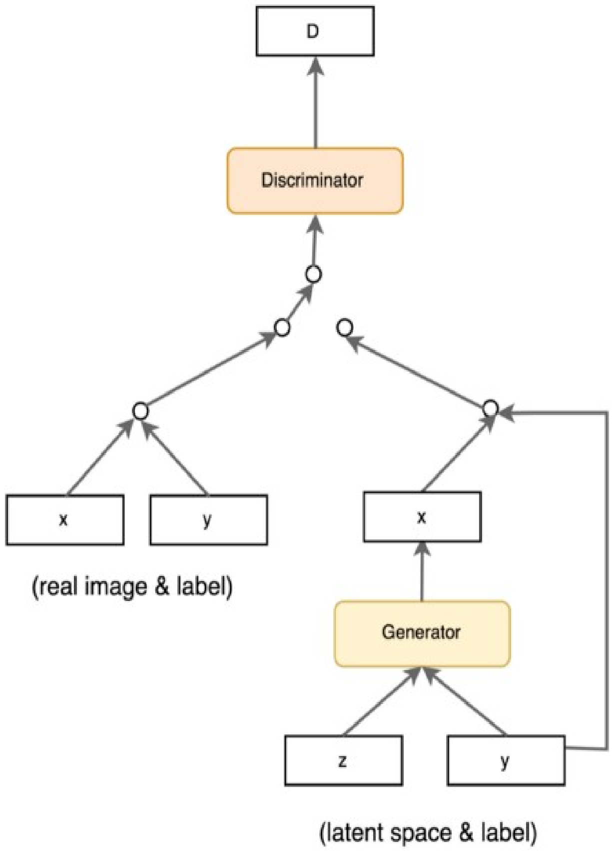

Figure 3. Dataflow used in CGAN to take advantage of the labels in the samples [8].

As can be seen in the above graph, there is an additional input layer in the form of conditional vector y that is fed into both discriminator network and generator network. Compare to GAN, the conditional GAN has a control over modes of the generated data by adding the label y as a parameter to both generator so that corresponding images will be generated and the discriminator in order to distinguish the real images better.

5 GANs problem

Many GAN models are suffering from a considerable problem, which is called mode collapse. The word *collapse* here is describing the generator when it produces a limited variety of samples [9].

In order to understand this better, here is one example that can happen when mode collapse appears. MNIST database is a huge database of 10 digits from digit 0 to digit 9. This also means there are 10 modes in MNIST database from 0 to 9. Figure 4 below shows the generated samples by two different GANs where in the top one all 10 modes were produced while the other one creates a single mode which is digit 6 in this case [10].

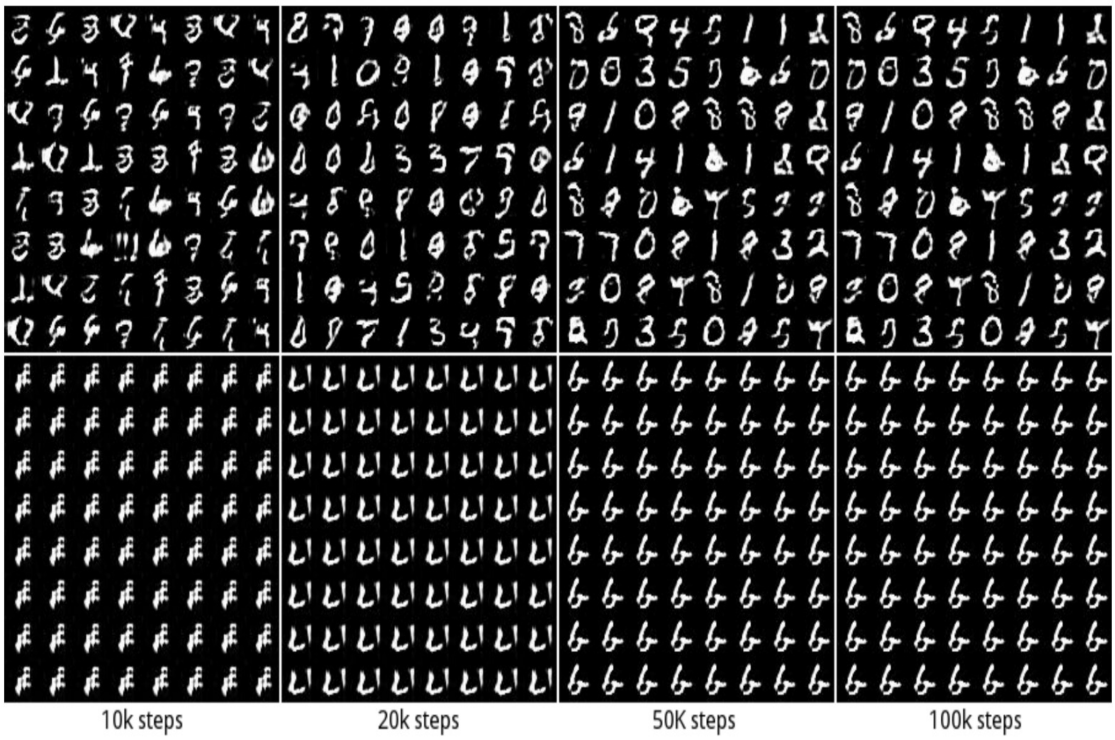

Figure 4. Generated samples of MNIST dataset by two different GANs.

5.1 Multiple GANs and Loss function

In order to avoid mode collapse GANs need to be trained with multiple generators instead of a single one. As mentioned before, the discriminator and the generator are in constant competition to fool each other and the mode collapse is known as lack of balance between the discriminator and the generator.

GAN can only cover a subset of modes in the dataset but in order to cover all modes we need to train multiple GANs instead. The process of using multiple GANs might be more time consuming and complicated, but it definitely improves GANs performance.

According to multiple GANs concept a ratio of 5 discriminator iterations per generator update can be tested instead of keeping it a one to one ratio. The first step to improve GAN is to balance the loss between the generator and the discriminator and this happens by minimizing their own loss. A good example would be when the generator generates perfect fakes, which means its loss would be 0 and discriminator can distinguish between fake and real data perfectly which also results in 0 loss.

Figure 5 explains loss function in a GAN, which has a key role in producing better result:

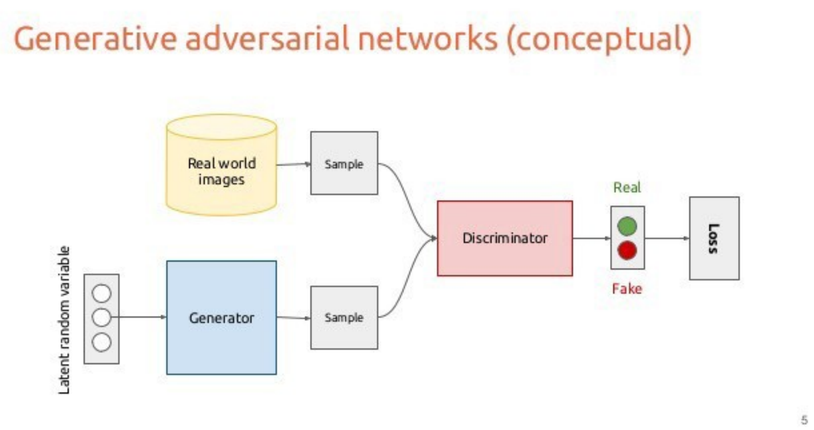

Figure 5. Loss function in a GAN

In the loss function the error on each input will be calculated by looking at what output it predicted for that specific input and take the difference of that output value. For example, let's say the model was classifying images of cats with a label of 0 and dogs with a label of 1. If an image of a cat is passed to the model and the model outputs 0.15 for this image, then the error between the model output versus the true label for the image would be calculated this way:

e = output – true

= 0.15 – 0 = 0.15

the individual error for each input is accumulated and passed through to a loss function and then the loss function is calculated at the end of each epoch. For example, a common loss function would be mean squared error (MSE) [11]. With MSE, the error from an input is received in exactly the same way as was calculated above and once the list of errors are received, the average of the squared errors is calculated.

$$MSE = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

The important question is if there is a way to minimize both generator's loss and discriminator's loss. The main goal is to minimize the loss function as more epochs are run. Two cases are presented below:

Generator Loss 0

Consider the case when the generator generates perfect fakes and the discriminator is fooled every time. This is when the generator's loss would be zero.

G(loss) => 0

Discriminator Loss 0

Second case is when the discriminator is able to tell which samples are fake and which are real which means the generator can never fool the discriminator and is not doing a good job.

D(loss) ==> 0

One important note here is that when we talk about training a GAN we actually mean training the generator as the generator is the trigger factor in this whole training process. Samples are continuously fed to the GAN and the outcome is expected to be 1 (real case) as the generator is expected to generate realistic images and the discriminator to say it is real or not.

Even though the Generator initially produces poor images and therefore Its loss is high, but through the training process it gets better and better. This is how the generator is trained via training the GAN.

6 Generating human faces and Colorization

In the recent years GAN has made a massive progress in image generation from numbers like MNIST dataset to celebrity image generation. In a traditional GAN, the input is randomly generated noise, which is not suitable for colorization. Instead, the generator must be modified to accept grayscale images as inputs and this would be possible by using conditional generative adversarial networks (CGAN).

One great and fun use of CGANs is to generate human faces. The architecture for generating human faces is pretty similar to MINIST dataset except that for face generation three-color channels are used for an RGB image instead of one channel for a black and white image [12]. Generation of human faces starts from a very low resolution, which is improved by adding new layers until producing a better quality / less blurry image.

The objective is to add color to black and white images. The generator takes in the black and white version of an image and outputs a full RGB version of it. This very first output is a low-quality image, which will be enhanced with every iteration. The discriminator compares colored images from both original dataset and the generator with grayscale input as condition and then tries to distinguish the fake image from the real one. Figure 6 shows generated faces using CGAN:

Figure 6. Randomly selected samples generated with the DCGAN architecture which specialized GAN towards image generation.

As seen above, the image on the left is the grayscale image; the middle one is the original image and the right one is the color added by the generator.

The same method can be applied to generate not only images of people who do not even exist in real World but also other fake objects and these would look more realistic after a few iterations because both networks work harder against each other.

In this thesis, huge dataset that is a collection of over 200,000 celebrity faces called CelebA was used. There are a few steps to take into consideration:

1- Preprocessing the images:

we define get_image function with four parameters, which are Image_path, width, height and mode:

```
def get_img(image_path, width, height, mode):
    """
    parameters: image path, image width, image height and image mode
    return: Image data
    """
```

2- Network Architecture

In order to get accurate result we need to have a very good GPU (above 4GB). We take Image width, Image Height and Image channel as parameters, which will be passed to the generator for generating fake images. On the other side the discriminator's job is to identify which image is from the training set and which is from the generator (real or fake). Once the discriminator finds the difference in the image, it sends the gradient signal to the generator

3- Generator and Discriminator loss

There are two cases here:

1. The discriminator should be able to output a high value when it receives a real image, meaning that it is confident about the image's reality

2. The discriminator should be able to output a low value when it receives a fake image, meaning that it is confident about the image being fake.

4- Training

After training the neural network for over a thousand or even a million times we get an unbelievable result. Generator generates perfect fake images that made it almost impossible for the discriminator to distinguish if it was fake or real. This would bring considerable benefits to creative industries such as video games and advertising.

3 Porting GAN to the browser

The next step was to train the GAN in the browser and observe both networks learn in real time. This was much more complicated than it seemed and in some cases it was easy to validate the hypothesis that a GAN is too difficult for the browser to handle. The most important issue when porting GAN to the browser was that GANs were computationally expensive, in the sense that, they require a very powerful GPU's to produce good results. For example, the previous topic about fake celebrity faces was generated by GANs after training many epochs using a high-power GPU (8 Tesla V 100) for 4 days [13].

How GAN can be trained in the browser is illustrated with a less intensive example where a MNIST dataset as a training dataset, train it only using Tensorflow.js and then port it to the browser. We will look at a model for recognizing handwritten digits (MNIST dataset) by looking at each pixel in the image and then using TensorFlow to train the model to predict the image by making it look at thousands of examples, which are already labeled [14]. In other words, first we will train the classifier by showing it the handwritten digit images and their labels; then we will evaluate the classifier's accuracy using test data.

To run the code locally we need NPM CLI or yarn installed and then we use the

```
npm run watch
yarn watch
```

A new tab will be opened immediately on localhost:1234 as it is shown in Figure 7:

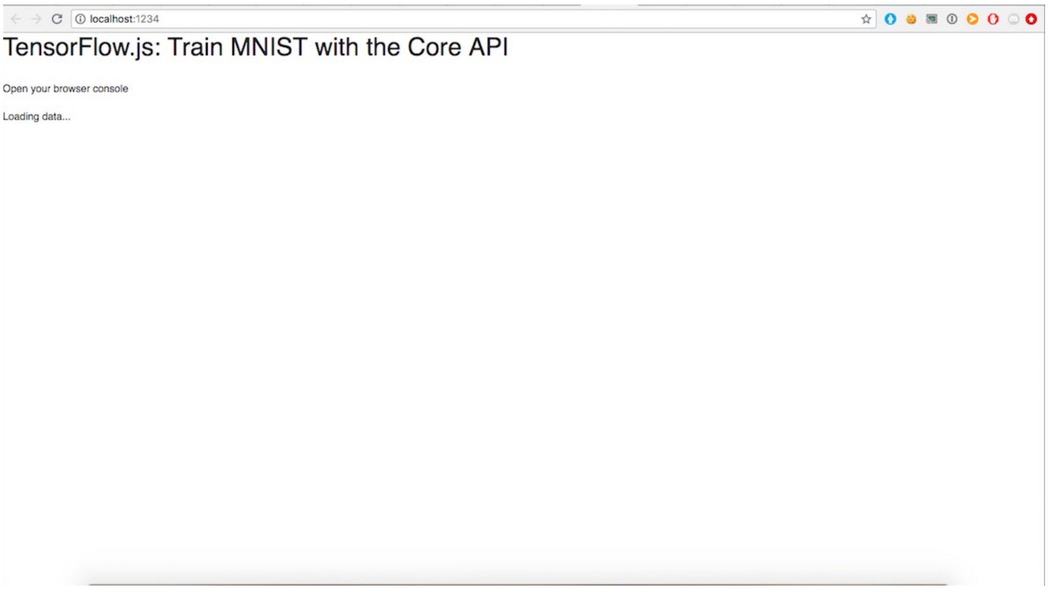

Figure 7. Train MNIST and port it to the browser

7.1 Training MNIST using TensorFlow

First, TensorFlow was imported as the first line of code and then a few constants were included as shown below:

- IMAGE_SIZE = IMAGE_H * IMAGE_W (height and width = 28x28)
- NUM_CLASSES – number of label categories (10 digits between 0-9 or 10 classes)
- NUM_DATASET_ELEMENTS –total number of images (65000 images in this case)
- NUM_TRAIN_ELEMENTS – number of training images (55000 in this case)
- NUM_TEST_ELEMENTS – number of test images
- MNIST_IMAGES_SPRITE_PATH & MNIST_LABELS_PATH – Paths to the images and the labels

Next is MnistData class for fetching MNIST dataset, which includes an important function called load () that is responsible for loading the images and label data.

```
async
load()
{
```

```
const image = new Image();
const canvas = document.createElement('canvas');
const ctx = canvas.getContext('2d');
const imageRequest = new Promise((resolve, reject) => {
      image.crossOrigin = '';
      image.onload = () => {
        image.width = image.naturalWidth;
        image.height = image.naturalHeight;
```

Async is a JavaScript feature to handle asynchronous actions. One way to handle this is to use JavaScript promises. This means we define a promise, which in this case is an image request. The promise takes a callback function with two arguments that are:

- Resolve: when the promise is actually fulfilled and finished
- Reject: When the promise is not fulfilled in given time

In our example we make a request for an MNIST image with specific image attributes such as width and height and once the image is loaded the promise is resolved.
Next, a new buffer will be initialized to contain every pixel of every image. This code will be looping through each image and initialize a new TypeArray for that iteration.

```
/* modified code obtained from https://thekevinscott.com/dealing-with-mnist-image-data-in-tensorflowjs/?cv=1*/

const datasetBytesBuffer =

            new ArrayBuffer(NUM_DATASET_ELEMENTS * IMAGE_SIZE * 4);

        const chunkSize = 5000;

        canvas.width = image.width;

        canvas.height = chunkSize;

        for (let i = 0; i < NUM_DATASET_ELEMENTS / chunkSize; i++) {

          const datasetBytesView = new Float32Array(

              datasetBytesBuffer, i * IMAGE_SIZE * chunkSize * 4,
```

```
                IMAGE_SIZE * chunkSize);
```

Then an image will be drawn and using getImageData() function that drawn image will turn into image data which returns an object representing pixels.

```
            ctx.drawImage(
                image, 0, i * chunkSize, image.width, chunkSize, 0, 0,
img.width,
                chunkSize);

            const imageData = ctx.getImageData(0, 0, canvas.width, can-
vas.height);
```

Last step is to remodel the buffer into a new TypedArray that holds the pixel data. At the end the promise is resolved and the image source (src) is set.

```
          this.datasetImages = new Float32Array(datasetBytesBuffer);

          resolve();

        };

        image.src = MNIST_IMAGES_SPRITE_PATH;

      });
```

At the end we set up the training process. Figure 8 is the result when porting the code to the browser:

Figure 8. Training MNIST dataset in the browser

This is an impressive outcome considering the percentage of the accuracy, which is 88%. We produce a prediction based on the output of the data through our neural network and if that prediction is correct we get the green label otherwise it's a red label as shown in the above picture. We repeat this process as many times as possible until we get a higher accuracy level.

Conclusion

According to the examples mentioned before, GANs can be used in many different areas to do tasks deemed impossible before, despite their new state in machine learning. In other words, it has changed the way neural networks are used. Not only can they make decisions and predictions based on various data such as in image classification, but they also can generate data that has never existed before. Therefore, they can learn from the object and generate similar versions of it.

In the introduction of this thesis, two research questions were posed questions were raised to be answered. These questions along with the answers are addressed below:

1. How long do GANs need to be trained to output accurate results?

Training a GAN is relatively hard, especially when it comes to heavier datasets, such as images, because in this case the colorizing or adding color layers needs to be taken care of manually. The initial result after the first training sessions with a GAN is most of the time not accurate at all, but the more it is trained, the better the results become. What makes training difficult is that both neural networks in a GAN try to compete with each other, which continues until one of them overcomes the other. That is when the generator produces perfect fake samples and the discriminator is not able to identify fake from real.

1- To what extent is it possible to improve GAN's performance?

There are many factors affecting GAN's performance. The main factor is taking care of the loss function. GAN can be improved by making a balance between the Generator's loss and the discriminator's loss. This is achieved by minimizing their individual loss, which results in minimizing the whole loss function. At the very beginning, discriminator's accuracy is very high and can easily separate fake objects from real, but after each iteration the generator improves and generates realistic samples and that is when the discriminator's loss starts to increase. In ideal situation the total loss will be zero.

References

1. Wang K, Gou C, Duan Y, Lin Y, Zheng X, Wang F. Generative adversarial networks: introduction and outlook. IEEE/CAA Journal of Automatica Sinica. 2017;4(4):588-598.

2. Goodfellow I. Generative Adversarial Networks [Internet]. Nips.cc. 2016. Available from: https://nips.cc/Conferences/2016/Schedule?showEvent=6202

3. Fumo D. Types of Machine Learning Algorithms You Should Know [Internet]. Medium. 2017. Available from: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

4. Carey O. Generative Adversarial Networks (GANs) — A Beginner's Guide [Internet]. Towards Data Science. 2018 [cited 23 March 2019]. Available from: https://towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-5b38eceece24

5. Gomez Mosquera D. GANs from Scratch 1: A deep introduction. With code in PyTorch and TensorFlow [Internet]. Medium. 2018. Available from: https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

6. Walker M. TensorFlow Machine Learning System [Internet]. Datasciencecentral.com. 2015. Available from: https://www.datasciencecentral.com/profiles/blogs/tensorflow-machine-learning-system

7. NASH C. Create Data from Random Noise with Generative Adversarial Networks [Internet]. Toptal Engineering Blog. Available from: https://www.toptal.com/machine-learning/generative-adversarial-networks

8. Hui J. GAN — Ways to improve GAN performance [Internet]. Towards Data Science. 2018 [cited 24 March 2019]. Available from: https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

9. Nibali A. Mode collapse in GANs [Internet]. 2017 [cited 20 January 2019]. Available from: http://aiden.nibali.org/blog/2017-01-18-mode-collapse-gans/

10. Hui J. GAN — Why it is so hard to train Generative Adversarial Networks! [Internet]. Medium. 2018. Available from: https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

11. Grover P. 5 Regression Loss Functions All Machine Learners Should Know [Internet]. Medium. 2018. Available from: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

12. Sharma S. Celebrity Face Generation using GANs (Tensorflow Implementation) [Internet]. Medium. 2018. Available from: https://medium.com/coinmonks/celebrity-face-generation-using-gans-tensorflow-implementation-eaa2001eef86

13. Dasgupta, T. (2018). *Generative Adversarial Networks using Tensorflow*. [online] Towards Data Science. Available at: https://towardsdatascience.com/generative-adversarial-networks-using-tensorflow-c8f4518406df [Accessed 15 Apr. 2019]

14. Team D. TensorFlow MNIST Dataset and Softmax Regression - DataFlair [Internet]. DataFlair. 2018 [cited 5 March 2019]. Available from: https://data-flair.training/blogs/tensorflow-mnist-dataset