

# MNIST and machine learning

---

CLASSIFICATION OF HANDWRITTEN DIGITS BY A SIMPLE LINEAR MODEL

A presentation by Lynn St.James Hanten and Steve Dias Da Cruz



# MNIST data-set

---

## **Goal:**

1. Train a model
2. Look at images
3. Predict what digits they are.

General approach and explanations on how to use machine learning

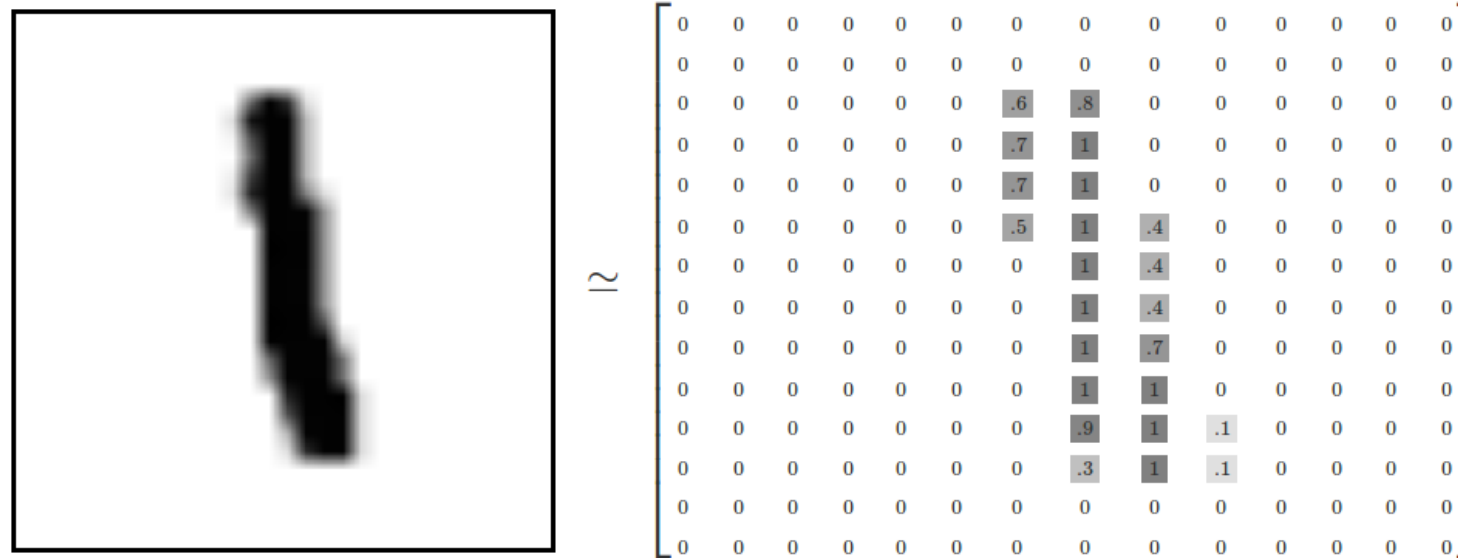
## Relation to Data Science:

- A lot of data needed to train the model
- Classify future input images
- Find characterization of each digit

# MNIST data-set

---

# MNIST data-set



**28x28** pixel images of hand-written digits

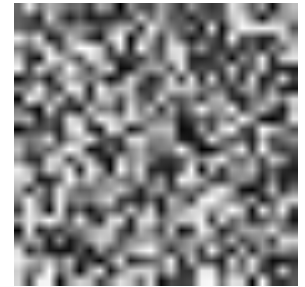
Every image can be thought of as an array of numbers between **0** and **1** describing how dark each pixel is (intensity of pixel)

# MNIST data-set

---

Not all arrays are MNIST digits:

1. Randomly pick a few points
2. Each pixel is randomly black, white or some shade of gray
3. We most probably get a noisy image



The data-set is split into **3 mutually exclusive** sub-sets.

- **Training** data (55000 images used to train the algorithm)
- **Test** data (10000 images used to test the algorithm)
- **Validation** data (5000 images used to optimize algorithm)

In machine learning we need **separated data**:

- To make sure that what we've learned actually generalizes

Test data:

- Used **to test** the algorithm, **not to optimize** or improve the algorithm

# MNIST data-set

---



Every MNIST data point has **two parts**:

1. **Image** of a handwritten digit
2. Corresponding **label** (number between **0** and **9**) representing the digit drawn in the image.

The labels for the above images are 5, 0, 4, and 1.

This label will be used **to compare** the **predicted** digit (by the model) with the **true** digit (given by the data)

# Weights

---

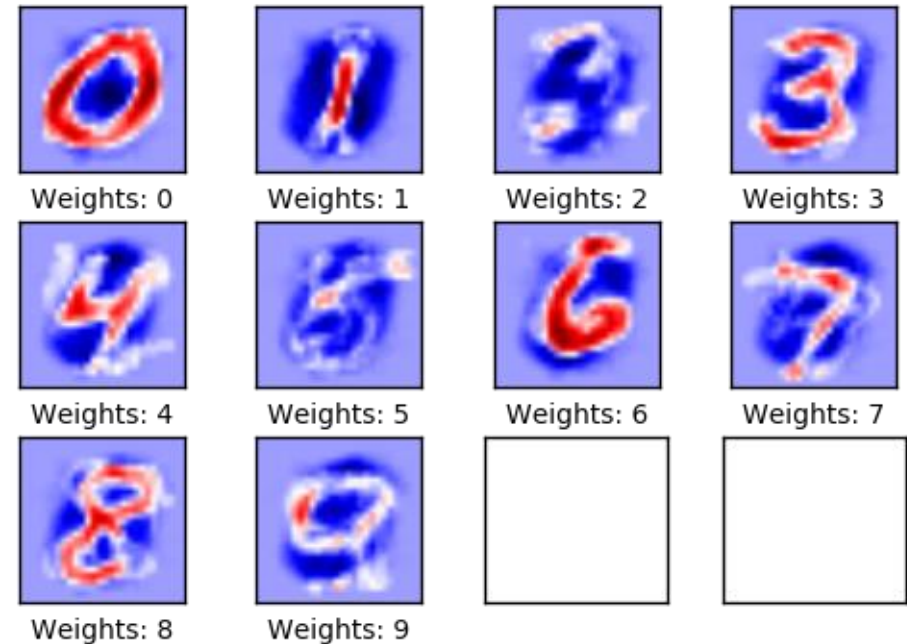
# Weights

## Goal of the model:

In order to distinguish the images, we will need a **filter**.

Better filter  $\Rightarrow$  Better detection of handwritten digits.

The following diagram shows an example of the weights (the filter) one model learned for each of these classes.



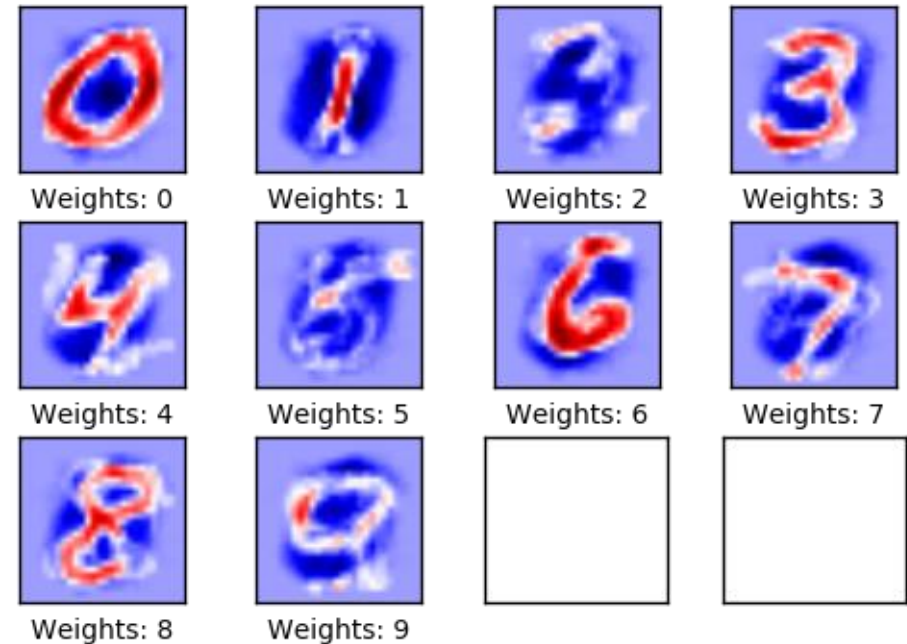


# Weights

The **weight** is a matrix, **for each digit**, which stores a value for each pixel.

The weight of a pixel is

- **negative**, if that pixel has a high evidence **against** the image being in that class
- **positive**, if it is evidence in **favor**.



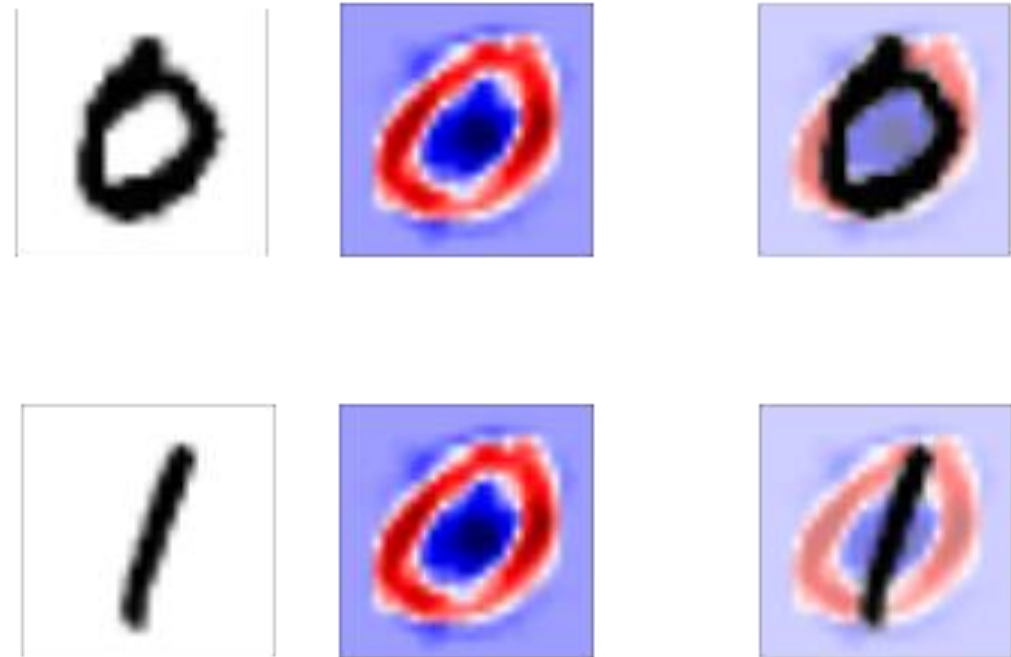
# Evaluation of the model

---

## Example:

The weights used to determine if an image shows a **0-digit** have a

- **positive** reaction to an image of a circle
- **negative** reaction to images with content in the center of the circle.



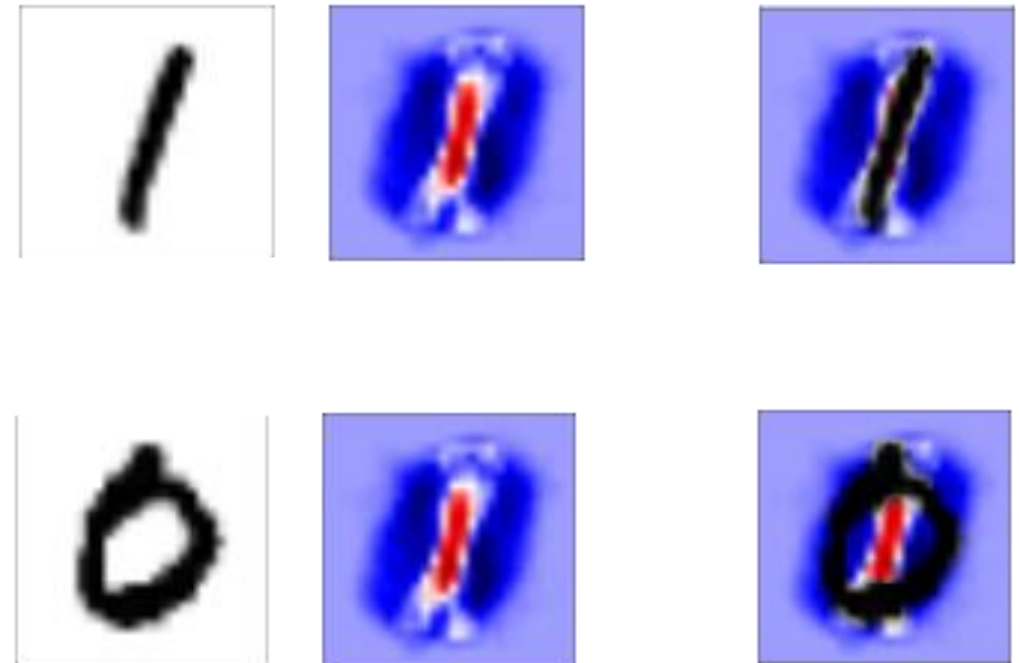
# Evaluation of the model

---

## Example:

The weights used to determine if an image shows a **1-digit** react

- **positively** to a vertical line in the center of the image
- **negatively** to images with content surrounding that line.



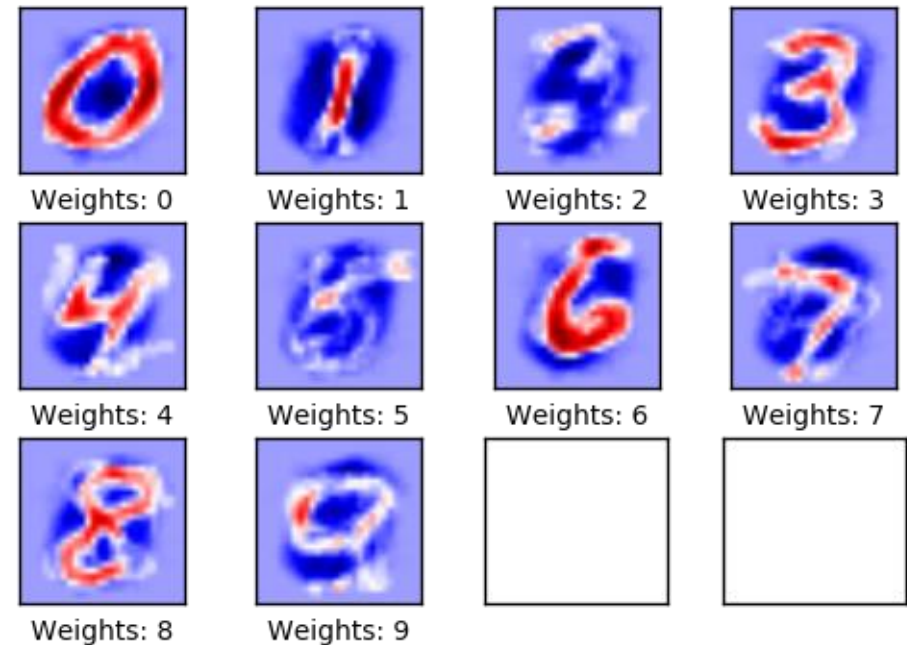
# Weights

**Positive** weight = pixel **should not** be 0 at that position.

The higher the weight, the closer to 1 the pixel intensity should be

**Negative** weight = pixel **should** be 0 at that position.

The lower the weight, the closer to 0 the pixel intensity should be

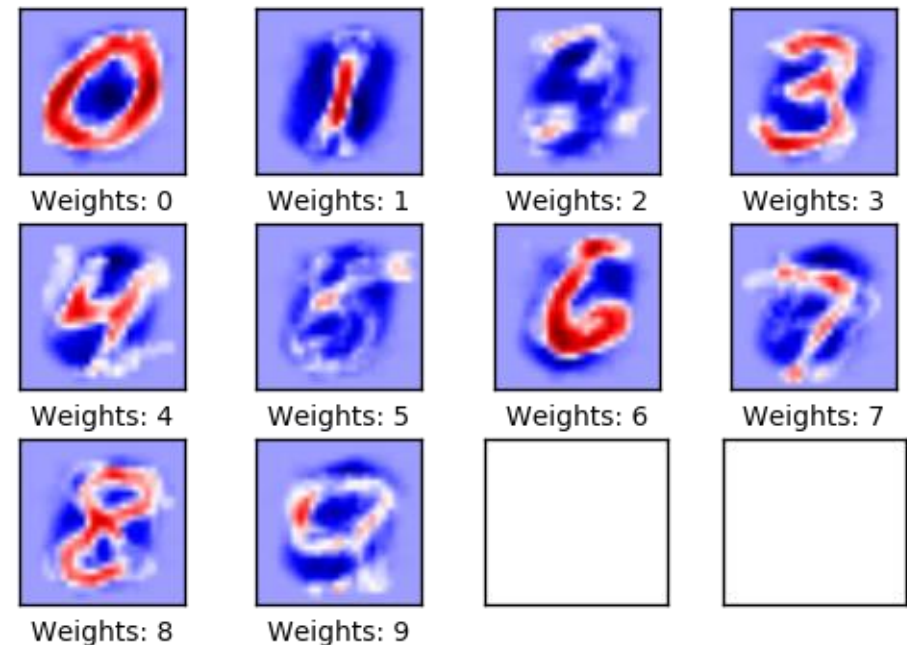


# Weights

If there is a non zero pixel at a position with **negative** weights, then this contributes to not being that number.

If there is non zero pixel at a position, where there are **positive** weights, then it contributes to being that number.

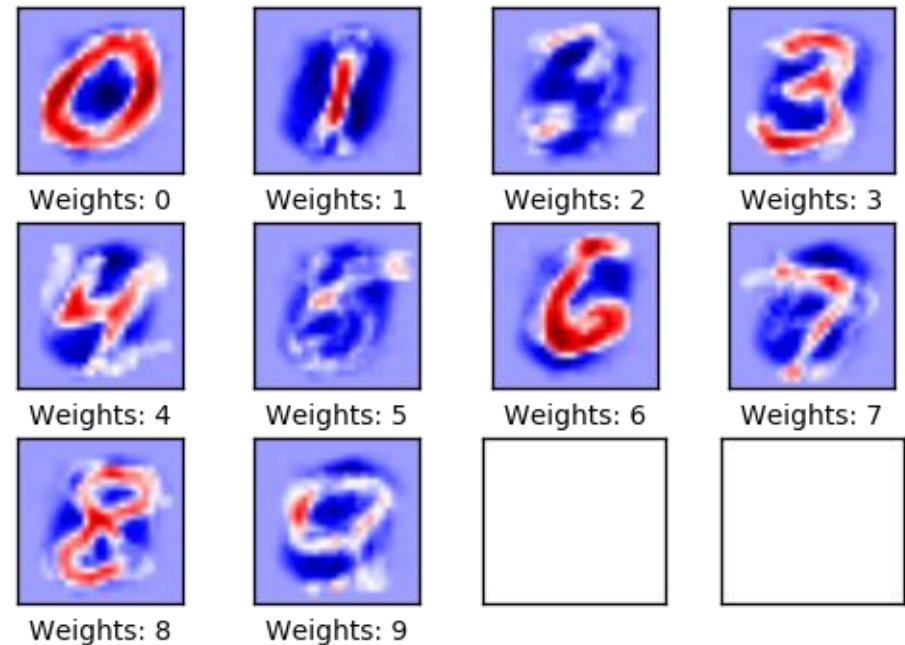
Weight matrix describes **for each** digit where the usual non zero pixels should be



# Weights

To measure the **evidence** of a digit being in one class, one computes a **weighted sum** of the pixel intensities:

- Each image is an array of values between 0 and 1
- Each filter has a weight for each pixel
- Multiply each pixel's intensity with its corresponding weight and compute the sum



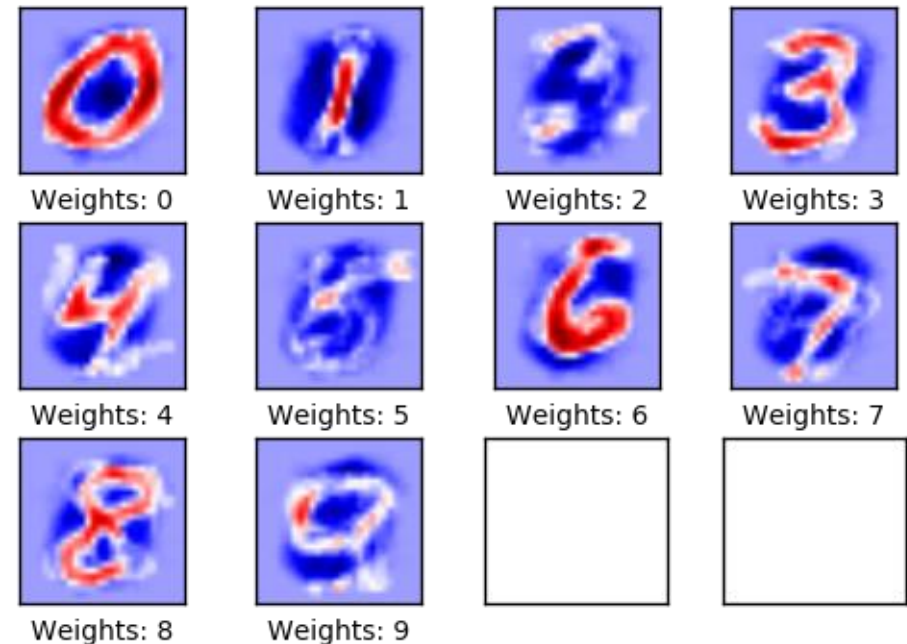
# Weights

For **ONE** handwritten digit, we compute a weighted sum with respect to **EACH** filter

- We get **10** different values representing the evidences for being each digit

For independence, we add some extra evidence, called a **bias**, to the weighted sum.

- Since linear model, we do not necessarily want the line to go through the origin



# Weights

---

The **evidence** for a class **i** (i.e. being the digit **i**) given an input **x** (the image) is:

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

We **multiply** the images variable **x** with the weights and then **add** the biases.

- **$W_i$**  is the **weight** for class **i**
- **$b_i$**  is the **bias** for class **i**
- **j** is an index for summing over the pixels in our input image **x**.



# Weights

---

These **evidences** are **difficult to interpret**, because the numbers may be very small or large, positive or negative.

Every image in MNIST is of a handwritten digit between **0** and **9**.

So there are only **10** possible things that a given image can be.

For each image, we want to give the **probabilities** for it being **each** digit.

Hence, we want to **normalize** the evidences, so that all the evidences for one image **sum up** to **1** and each element is limited **between 0** and **1** (data transformation).

- Converting evidences into a **probability** distribution over 10 cases representing **probabilities** of our input being in each class.

# Weights

---

## For example:

Our model might look at a picture of a 9 and output

- an **80%** chance of being a 9
- a **5%** chance of being an 8 (because of the top loop)
- a bit of probability to all the others because it isn't 100% sure.

# Weights

---

Convert the evidences into (predicted) probabilities using the **softmax** function:

$$y = \text{softmax}(\text{evidence})$$

From this, we can get the **predicted class** of our handwritten digit by taking the label of the **largest** element.

Hence, we take the **highest probability** for being in a specific class, which will represent **the prediction** of the handwritten digit.

For the previous example, this will be the digit 9, since it has a 80% chance.

# Weights

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

The element of the **i'th** row is an **estimate** of how likely the input image is to be of the **i'th** class.

# Weights

---

Remember: The weights and biases represent the filter .

In order to make the model **better** at classifying the input images, the **machine learning** part will try to

- **change** the weights and biases
- **optimize** them

But first, we need to know **how well** the model currently performs:

- Compare the **predicted** output of the model to the **desired** output.

# Training

---

# Training

---

In our case, **training** means to get a better filter, so that we can distinguish the handwritten digits better.

We need a definition

- to describe what it means for the model to be **good**.
- or to describe what it means for a model to be **bad** (typically used in machine learning).

We call this the **cost**

- It represents how far off our model is from our desired outcome.

We try to **minimize** that error/cost

- the **smaller** the error margin, the **better** our model is.

# Training

---

To determine the cost of our model, we are using a function called **cross-entropy**:

$$H_{y'}(y) = - \sum_i y'_i \log(y_i)$$

Where

- $y$  is our **predicted** probability distribution and
- $y'$  is the **true** distribution for one handwritten digit.

The cross-entropy is measuring how **inefficient** our predictions are for describing the truth.



# Training

---

## The **cross-entropy**

- is a performance measure used in classification.
- is a function that is always positive.
- **equals zero**, if the predicted output of the model **exactly matches** the desired output.

The **goal** of our training is to minimize the cross-entropy so it gets as close to zero as possible.

How to use the cross-entropy:

1. Calculate the cross-entropy for **each** of the image classifications
  - To get a measure of how well the model performs on **each** image individually.
2. Take the **average** of the cross-entropy for all the image classifications
  - We **need a single scalar** value to define the performance of our model

# Training

---

The preceding procedure gives us a **cost function**, which we want to minimize in order to get a better model.

To minimize the cross-entropy one can use the **gradient descent algorithm** (see numerical analysis).

**Gradient descent** is an algorithm, where it

- shifts each variable a little bit in the direction that **reduces** the cost.
- adjusts the current **weights** and **biases** .
- uses a learning rate of 0.5.

Consequently, it **minimizes** the **mean** of the entropies for the images which we used to train!

# Training

---

In machine learning, and especially in TensorFlow, the model:

- is represented by a **graph**, which stores all the computations
- can use the **backpropagation** algorithm to efficiently determine how your variables (weights and biases) affect the cost you ask it to minimize.

**Backpropagation** is a common method of training artificial neural networks used in combination with an optimization method (here: gradient descent).

## Meaning:

1. Calculate the gradient of a cost function (cross-entropy)
2. Go back in the model
3. Adjust parameters (weights and biases) in order to optimize the cost function at the current state.

# Training

---

There are 55.000 images in the training-set. It takes a **long time** and it is **expensive** to calculate the gradient of the model using **all** these images.

It is better to do an iteration:

- Use a **small batch** of images (for example 100) in each iteration of the optimizer
  - Use a new subset every time. Doing this is cheap and has much of the same benefit as taking all images.
- Execute the optimizer using those 100 training samples.
  - gradually improve the weights and biases of the model.
- **Repeat** this procedure a lot of times.
  - adapt and generalize the weights for all different kind of handwritten images.

Using small batches of random data is called **stochastic training**

# Training

---

## Summary of the training phase:

1. Calculate the entropy of the predicted and true digit.
2. Calculate the mean over all the entropies (measure of performance).
3. Minimize the mean (0 means that all our predictions are correct).
4. Use the gradient method to minimize the cost.

# Evaluation of the model

---

# Evaluation of the model

---

We want to check where we predicted the correct label.

For **each** image (of the **test data**):

1. Check if the predicted digit is the same as the true digit
2. This gives us a vector of **Booleans**
3. Transform the Booleans into **numbers**
  - False becomes **0**
  - True becomes **1**
4. Calculate the average of these numbers.
5. Calculate the accuracy of the model

For example:

- [True, False, True, True] would become [1,0,1,1]
- The average is 0.75
- Hence our accuracy is 75%.

# Evaluation of the model

---

After **no** optimization iteration, the accuracy on the **test-set** is **9.8%**.

This is because the model has only been **initialized** and **not optimized** at all:

- At the initial step, the weights and biases are 0.
- All the evidences will become zero and we get 10 zero values.
- Softmax will output  $[0.1, \dots, 0.1]$ .
- The maximum of this is 0.1 (the maximum will take the first digit since all are equal).
- All the images are predicted as 0.

It always predicts that the image shows a **0** digit and it turns out that 9.8% of the images in the test-set happens to be **0** digits.



# Evaluation of the model

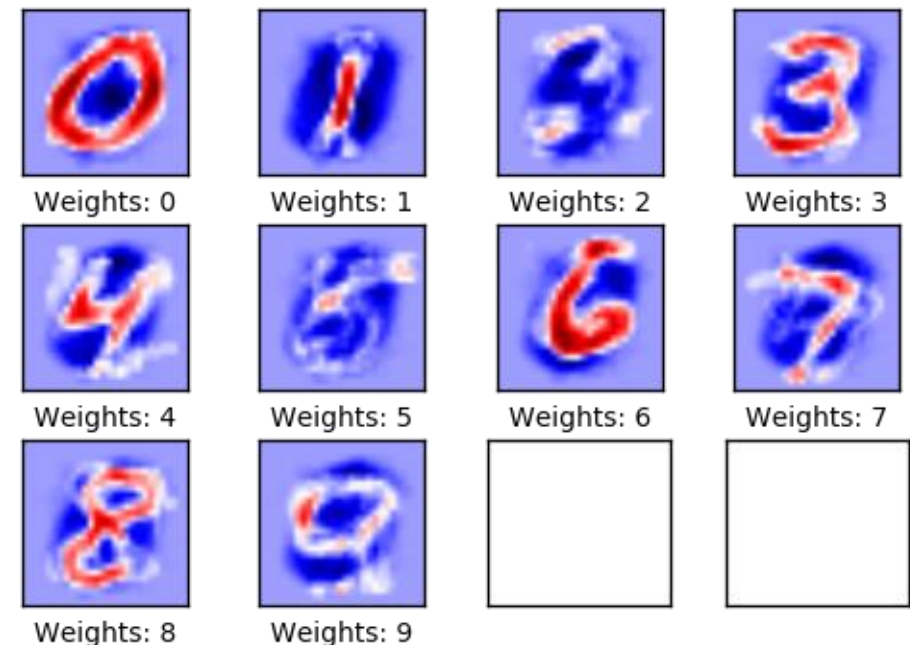
## The weights for our model after 1 iteration:

The model has increased its accuracy on the **test-set** to **40%** up from 9.8%.

The weights mostly look like the digits they're supposed to recognize. This is because only **1** optimization iteration has been performed.

The weights are only trained on 100 images.

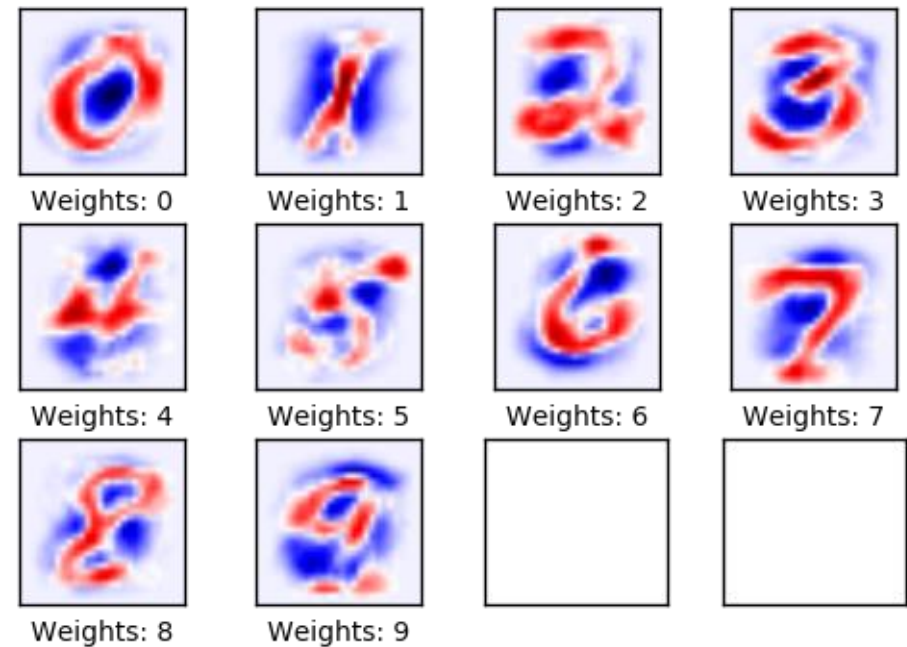
After training on several thousand images, the weights become more difficult to interpret, because they have to recognize many variations of how digits can be written.



# Evaluation of the model

---

The weights for our model after 10 iterations:



# Evaluation of the model

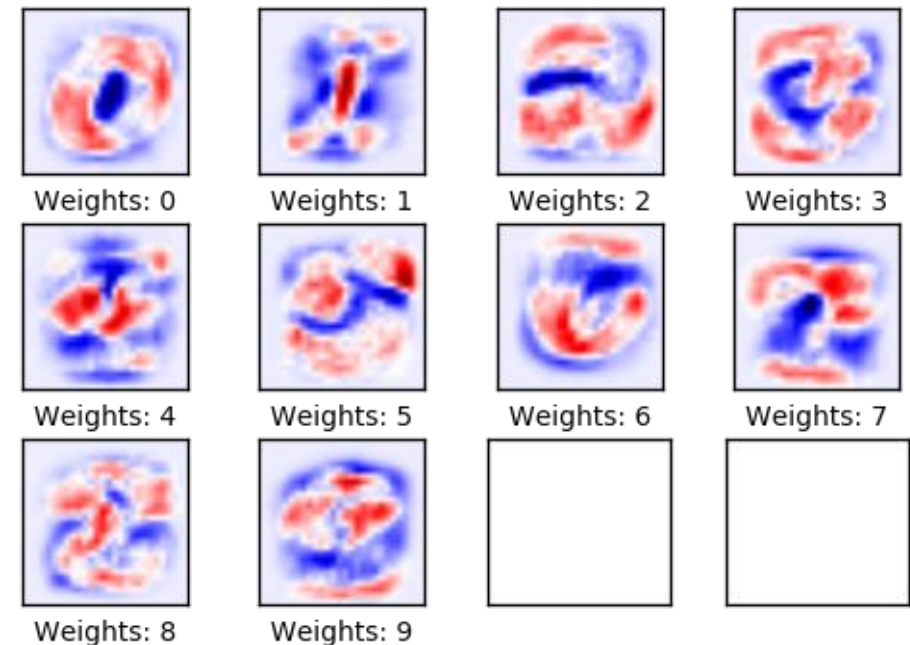
## The weights for our model after 1000 iterations:

The model has increased its accuracy on the **test-set** to **91%**.

The model has been trained for **1000** optimization iterations, with each iteration using **100** images from the **training-set**.

Because of the great variety of the images, the weights have now become difficult to interpret.

We may doubt whether the model truly understands how digits are composed, or whether the model has just memorized many different variations of pixels.



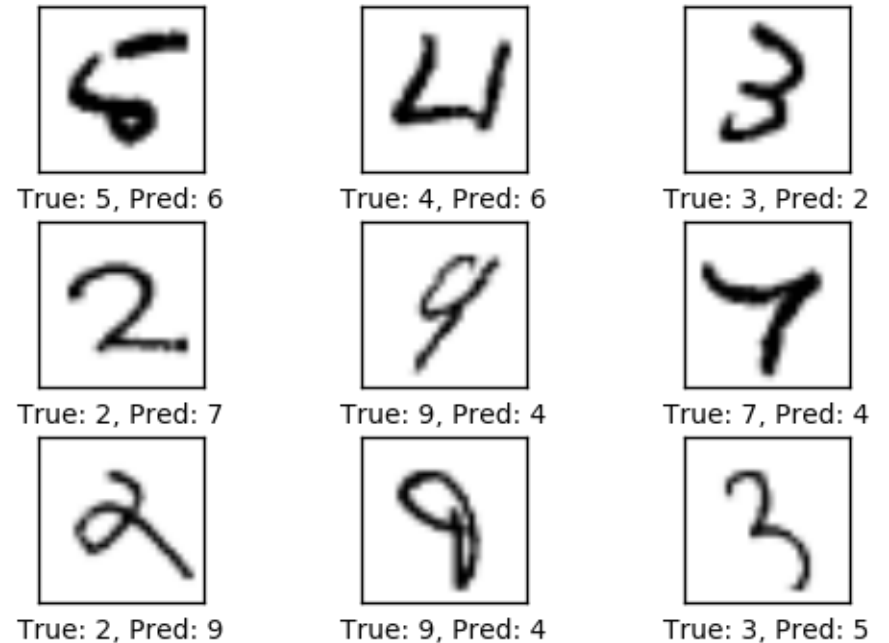
# Evaluation of the model

---

## Some of the misclassified digits after having trained the model by 1000 iterations:

Some of the misclassifications are justified, because the images are very hard to determine with certainty even for humans, while others are quite obvious and should have been classified correctly by a good model.

But this simple model cannot reach much better performance and more complex models are therefore needed.



# Evaluation of the model

---

This simple linear model had about **92%** classification accuracy for recognizing hand-written digits in the MNIST data-set.

Is that good? No, it's pretty bad! This is because we're using a very simple model.

A simple **Convolutional Neural Network** has a classification accuracy of about **99%** or more.

Convolutional Networks work by moving small filters across the input image. This means the filters are re-used for recognizing patterns throughout the entire input image. This makes the Convolutional Networks much more powerful.

The convolutional neural network can then be extended to distinguish, for example, between different animals.



© University of Toronto

# References

---

[https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/01 Simple Linear Model.ipynb](https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/01_Simple_Linear_Model.ipynb)

<https://www.tensorflow.org/versions/r0.11/tutorials/mnist/beginners/index.html>

[https://github.com/aymericdamien/TensorFlow-Examples/blob/master/input\\_data.py](https://github.com/aymericdamien/TensorFlow-Examples/blob/master/input_data.py)

[https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3 NeuralNetworks/convolutional\\_network.ipynb](https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/convolutional_network.ipynb)

[http://learningtensorflow.com/getting\\_started/](http://learningtensorflow.com/getting_started/)

<http://yann.lecun.com/exdb/mnist/>

<http://colah.github.io/posts/2014-10-Visualizing-MNIST/>

<http://colah.github.io/posts/2015-08-Backprop/>

<https://www.tensorflow.org/versions/r0.11/tutorials/mnist/pros/index.html>

<https://en.wikipedia.org/wiki/Backpropagation>

# References

---

[https://en.wikipedia.org/wiki/Softmax\\_function](https://en.wikipedia.org/wiki/Softmax_function)

[https://en.wikipedia.org/wiki/Cross\\_entropy](https://en.wikipedia.org/wiki/Cross_entropy)

[https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)

[https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)

[https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02\\_Convolutional\\_Neural\\_Network.ipynb](https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb)

<http://colah.github.io/posts/2015-09-Visual-Information/>

<http://neuralnetworksanddeeplearning.com/chap3.html#softmax>

<http://neuralnetworksanddeeplearning.com/chap1.html>

[https://en.wikipedia.org/wiki/MNIST\\_database](https://en.wikipedia.org/wiki/MNIST_database)