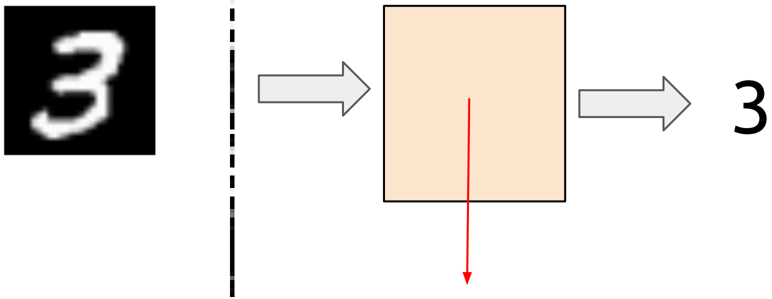
Dive into the world of machine learning!

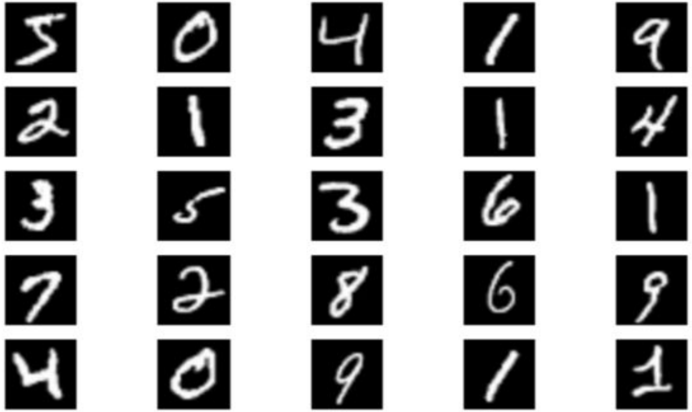
Annica Ivert
aniv@netlight.com

Case: Handwritten digit recognition

Goal: Train a ML model that can classify a 28x28 pixel image of a digit.



Find some function that takes 784 input pixels and transforms it to the correct output response



Artificial neural networks

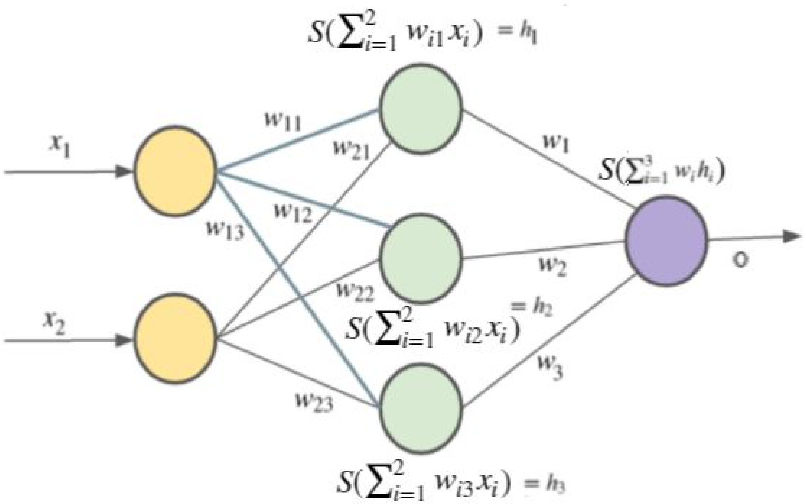
- The ML model we will use is called an **ANN** and is inspired by our brains

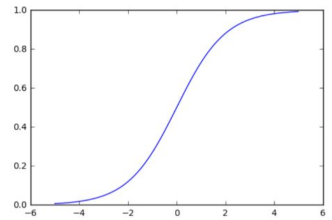
- It basically represents a function that takes some inputs (an image) and produces some output (the label)

- Between the input and the output there exist a number of “**hidden layers**”

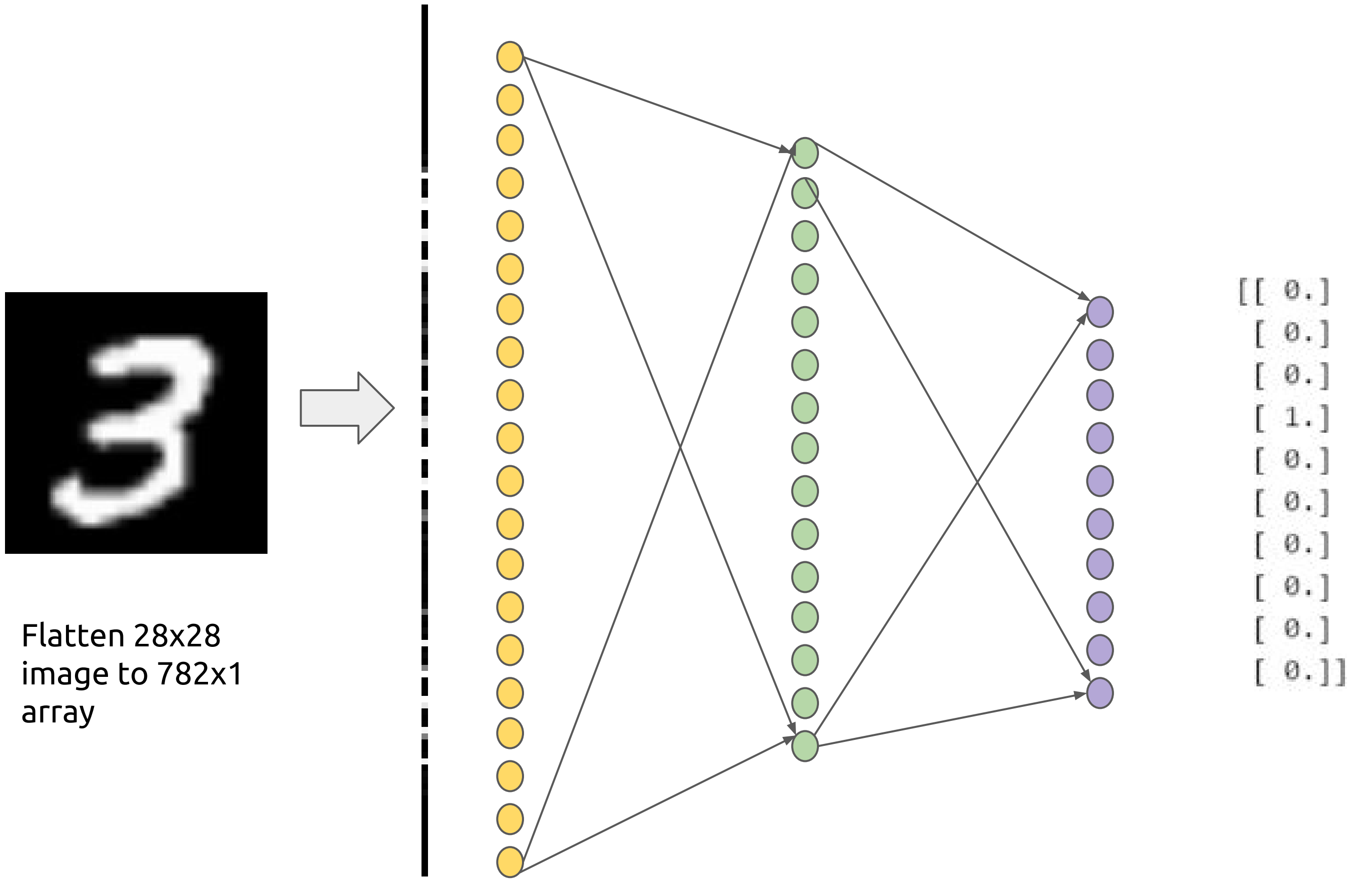
- When calculating the output, the input is multiplied with the weights and passed through an **activation function** (in our brains neurons either “fire” or do not)

- Our goal is to find the optimal set of weights





ANN - digit recognition



Flatten 28x28 image to 782x1 array

One-hot encoding of the number 3

How do we train the model?

1. Initialize model with some random weights
2. Calculate output response for some **training data**.

   Our data will be tuples : 

   This is a **supervised learning** problem, i.e. we know what the response should be for each data sample.

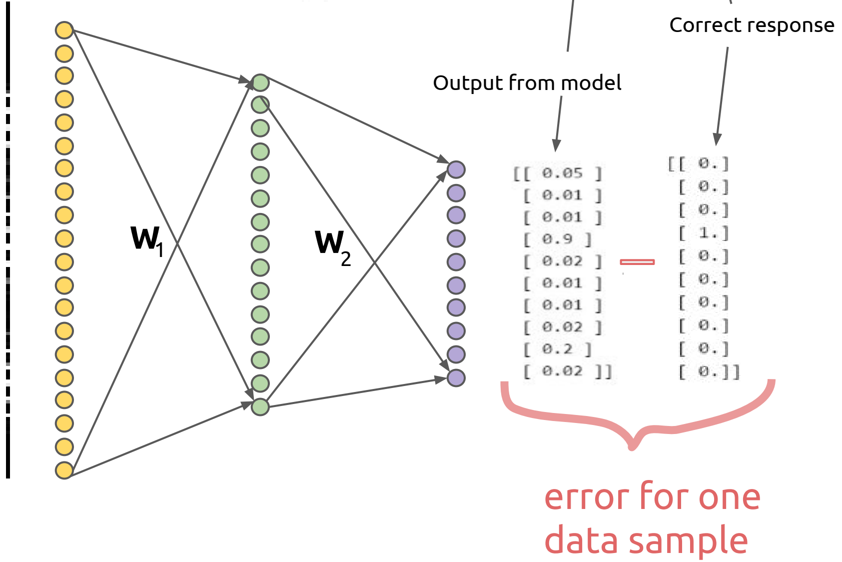
3. We want to update the weights of the model in such a way that the error decreases. 
   The error is a function of the input, weights and output.
   The only thing we can change are the weights.
4. How do we find min/max of a function? Differentiate the error function with respect to weights!
5. Update weights in the direction where the error decreases (**gradient descent**).

Minimize cost function

$$e = \sum_{i=1}^{N} \frac{1}{2}(o_i - y_i)^2$$

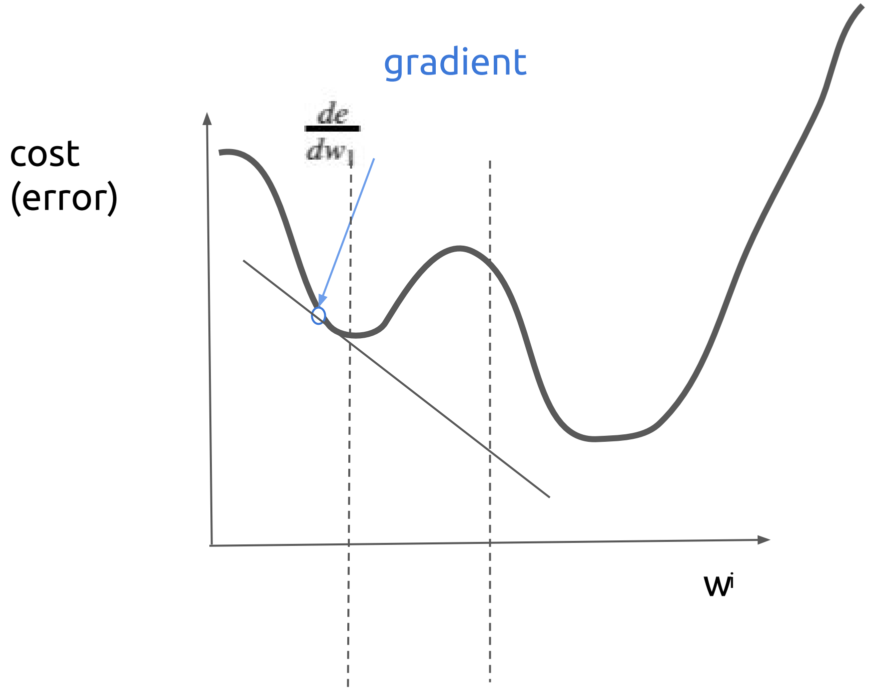
Differentiate!

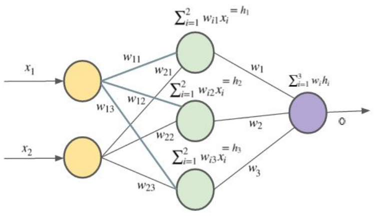
$$\frac{de}{dw_1} \qquad \frac{de}{dw_2}$$



Gradient descent

- We want to **minimize the cost function,** i.e. the error of our predictions
- If we calculate the **derivative of the cost function with respect to the weights**, we know in which direction to update them in order to decrease the error
- If we do this repeatedly the hope is that we find a **global minimum**
- If we update the weights to little in each time step we might get stuck in a local minima
- If we update them to much, we might not converge
- **We stop updating the weights after a number of epochs** or when the gradient is smaller than some threshold

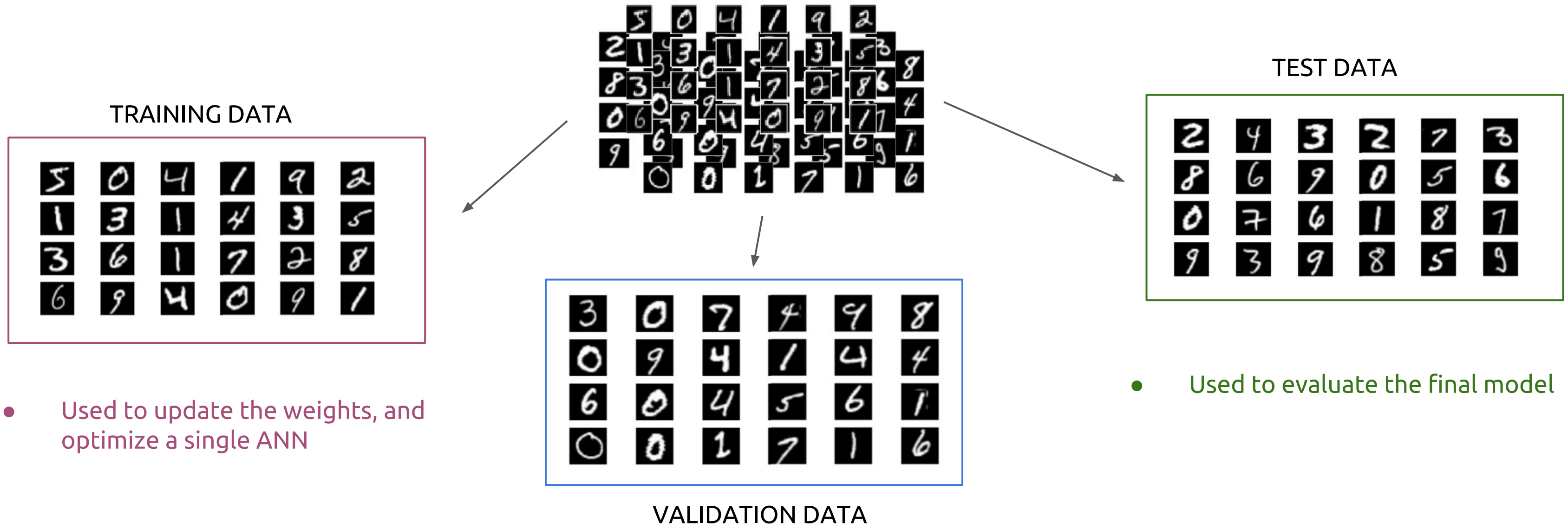




$$e = \sum_{i=1}^{N} \frac{1}{2}(o_i - y_i)^2$$

function of input, weights

Training and validating a ML model



- Used to update the weights, and optimize a single ANN

- Used to evaluate the final model

- Used to optimize the parameters of an ANN such as:
  - Number of hidden nodes
  - Number of training epochs
  - Weight update rate
  - Batch size (if using stochastic gradient descent)

ML Case layout

https://github.com/annicai/codepub

- We will implement parts of an ANN in order to understand better how it works
    - Load the data
    - Plot it, explore
    - Implement some missing parts of the ANN
    - Train the ANN.
        - Vary parameters, observe result
        - Which is the best model?

- There are of course also good libraries for this.

  With only a few lines of code one can train a ML model!

    - Example: Keras

```
num_hidden = 100
num_classes = 10
input_size = 784

def create_keras_model():

    model = Sequential()

    # Adding the first layer, input node size 784 and output 100 (num_hidden)
    model.add(Dense(num_hidden, input_dim=input_size, init='normal', activation='sigmoid'))

    # The next layer will automatically have input size as the output from the previous
    # We only need to specify next output, which is our final output of size 10
    model.add(Dense(10, init='normal', activation = 'sigmoid'))

    # Compile model. Define evaluation metrics and 'Stochastic gradient descent' as the weight-update method.
    model.compile(loss='mse', optimizer='sgd', metrics = ['accuracy'])
    return model
```

```
#Create model
model = create_keras_model()

#Train the model
model.fit(training_data_data, training_data_labels, validation_data=(validation_data_data, validation_data_labels),
          nb_epoch=20, batch_size=100, verbose=2)
```

All code needed to train an ANN with Keras