CME 213

SPRING 2019

Eric Darve

Final project

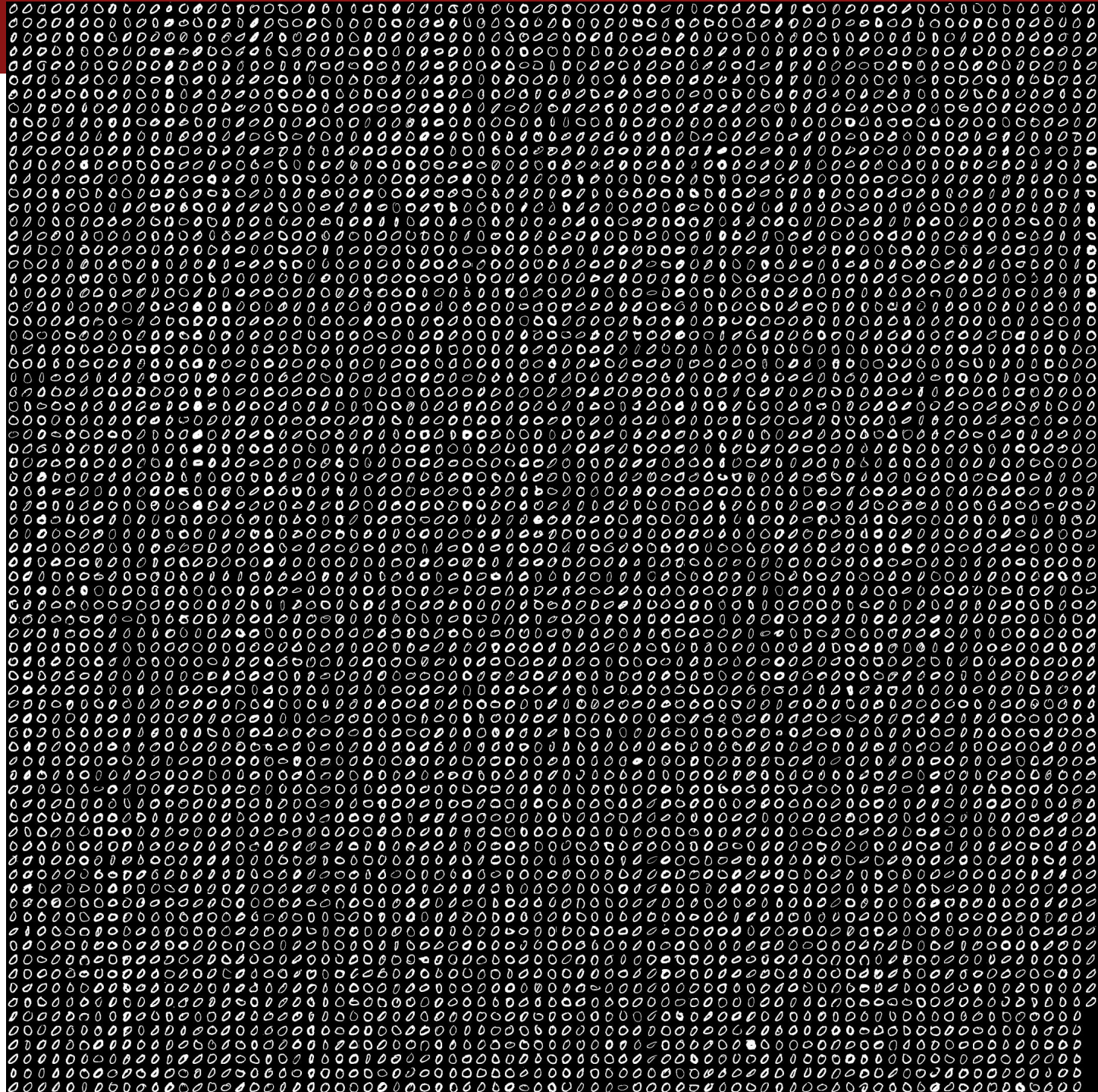
Final project is about implementing a neural network in order to recognize hand-written digits.

Logistics:

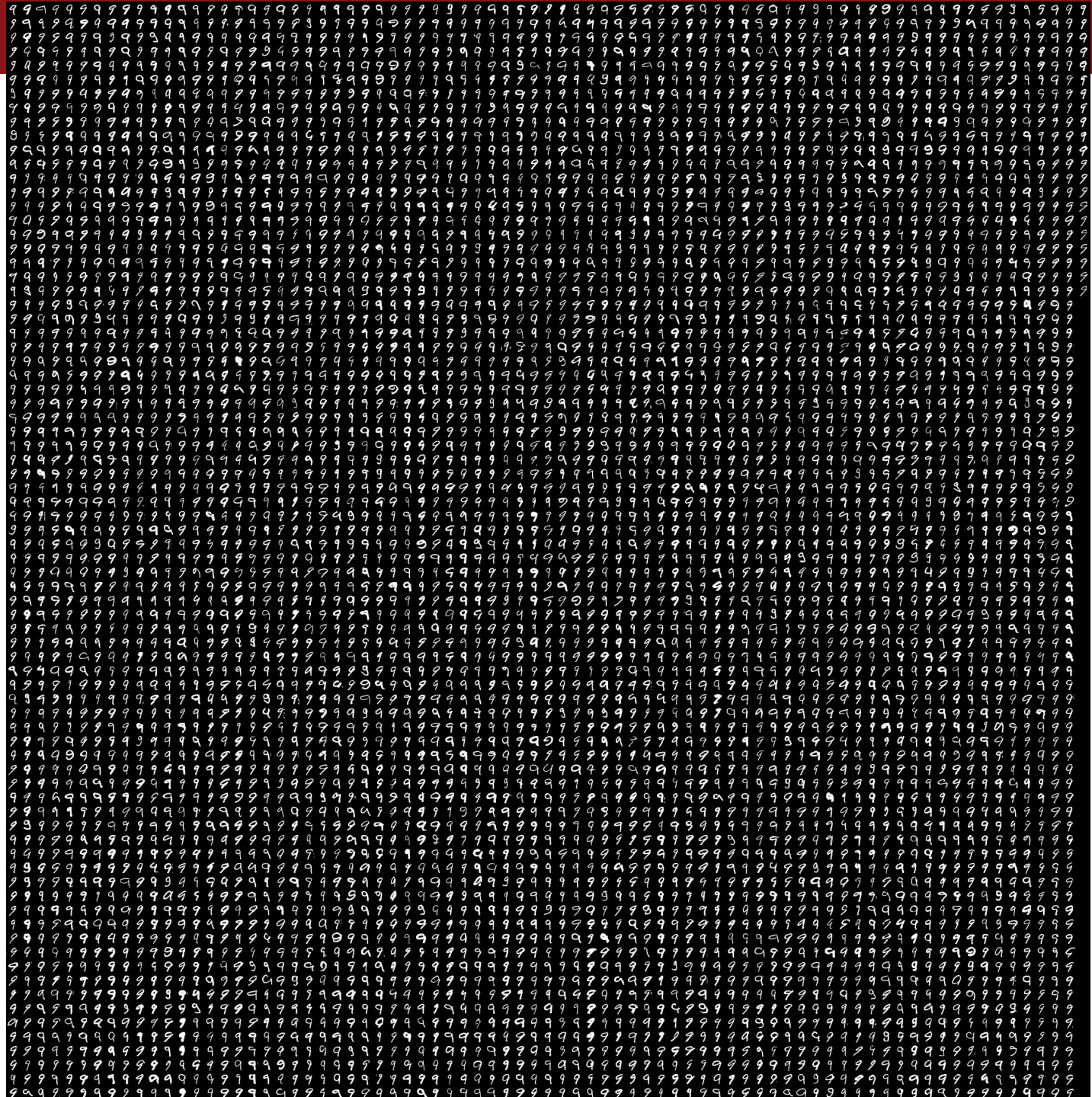
- **Preliminary report + code: Friday May 31**
- **Final report (4 pages) + code: Sunday June 9**

- Preliminary report: focus is on correctness.
- Final report: profiling and analysis, performance, quality of report
- What are the performance bottlenecks in your code? How can they be addressed?
- Correctness: discuss your strategy to test your code; for various functions, test output for valid inputs. Make sure you distinguish roundoff errors from genuine bugs.

A bunch of Zeros

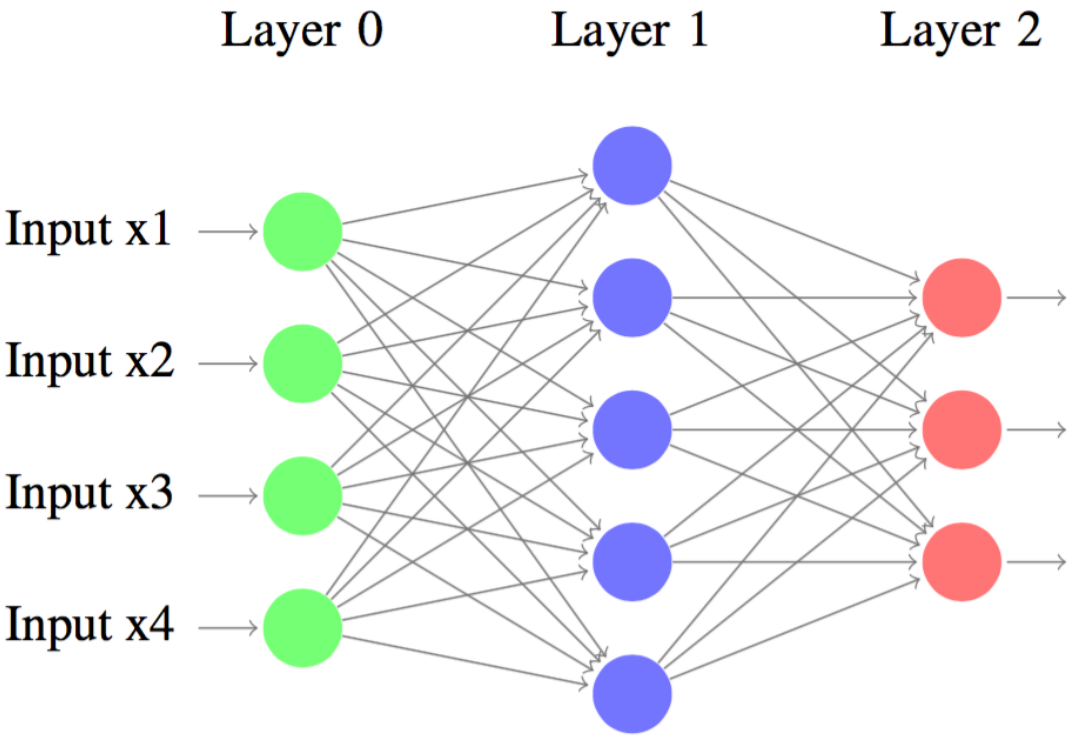


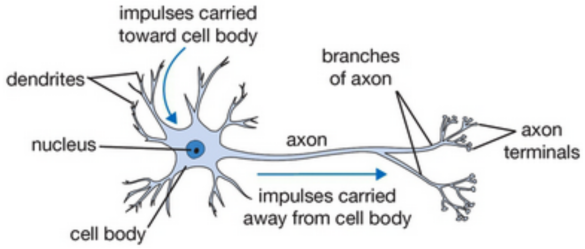
A bunch of Nines



What is a neural network?

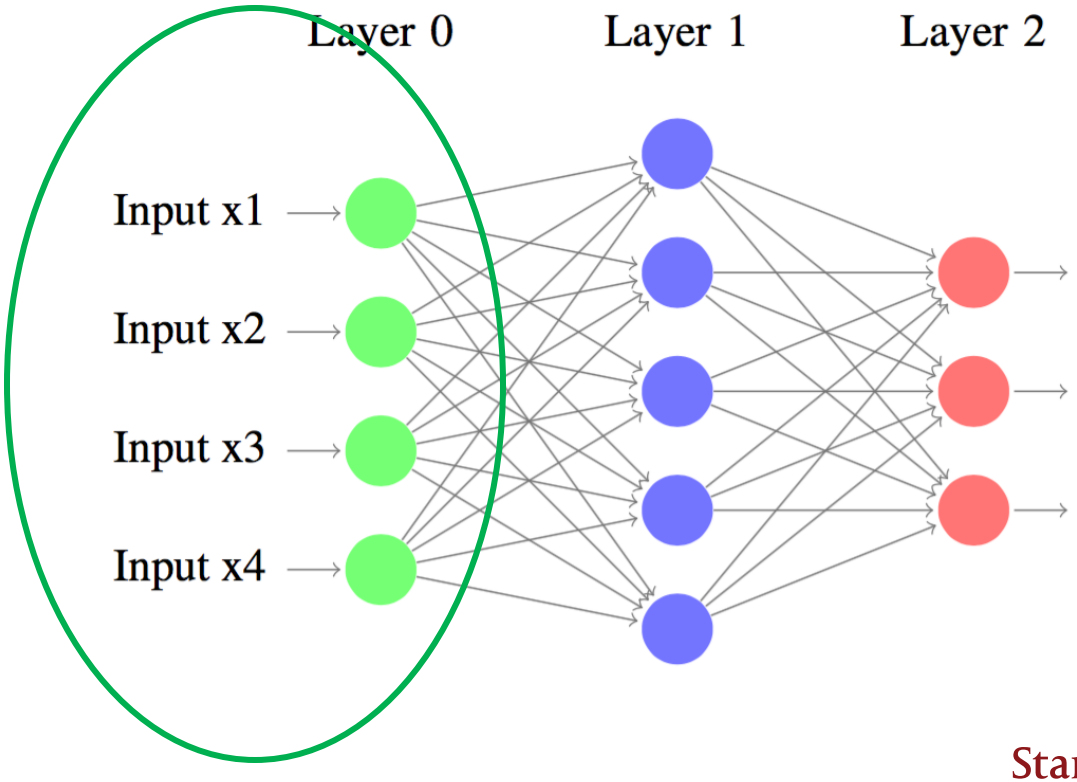
- It has little to do with actual neurons in the brain, although the idea derived from neuron connections in the brain.
- A neural network is a sequence of layers.





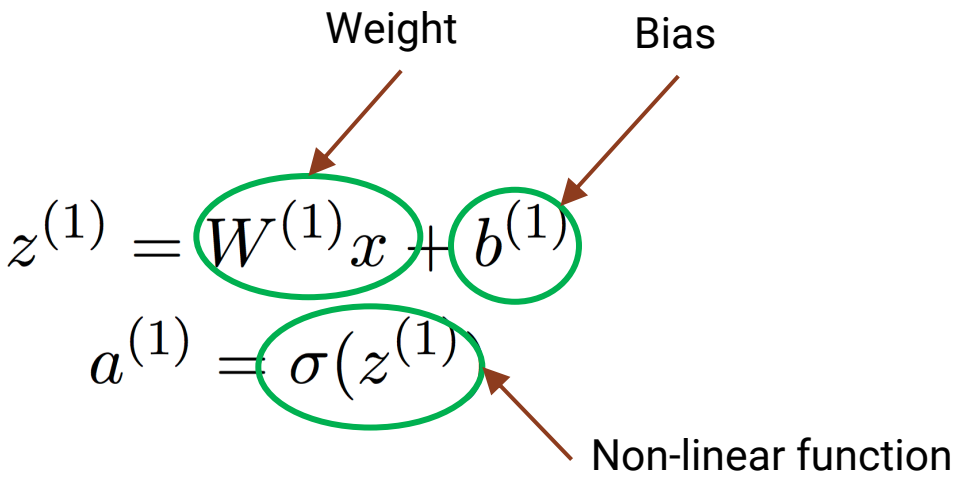
Input layer

This is the data input, for example all the digits in the image.

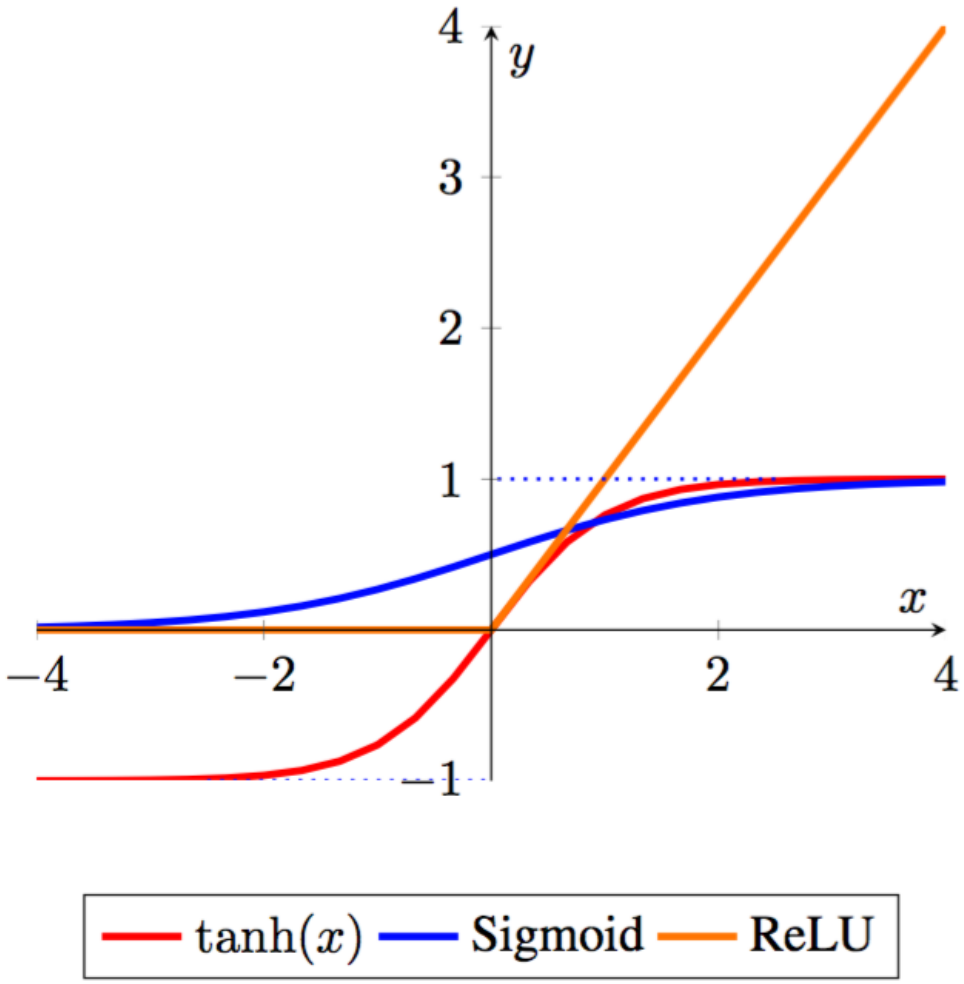


Connection between layers

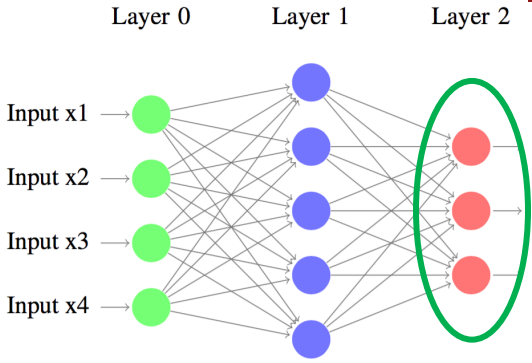
- To calculate data for the next layer, we perform two operations.
- A matrix-vector multiplication with matrix W.
- Then we take the value at each node and apply a non-linear function:



Examples of non-linear functions typically used



The last layer



- The last layer is a bit different.
- In our case, we want to produce a probability vector, that gives the probability for each possible digit, from 0 to 9.
- This can be done using a **`softmax`** function:

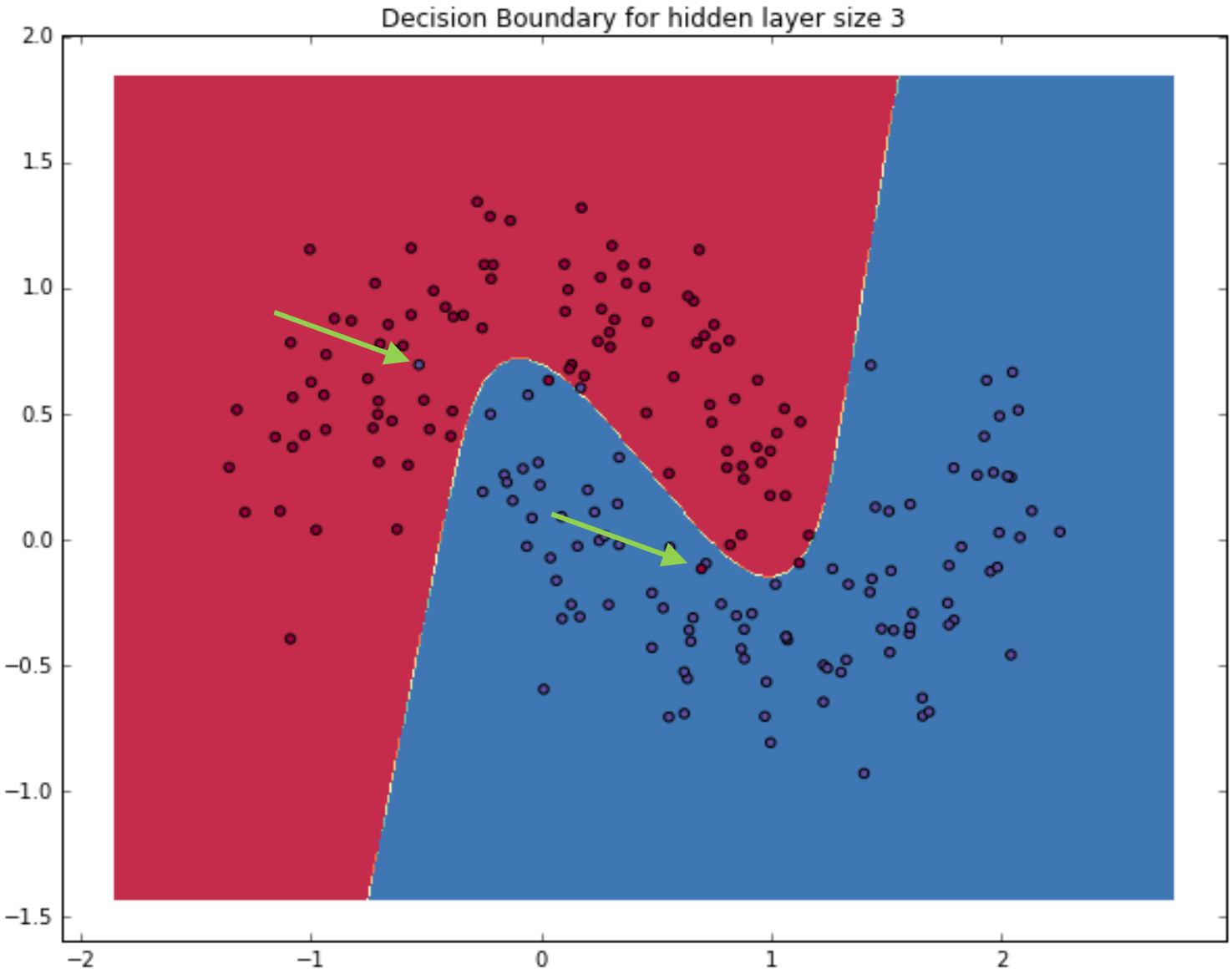
$$\text{softmax}(z^{(2)})_j \stackrel{\text{def}}{=} P(\text{label} = j|x) \stackrel{\text{def}}{=} \frac{\exp(z_j^{(2)})}{\sum_{i=1}^{C} \exp(z_i^{(2)})}$$

What is this whole thing about?

- So we have a sequence of layers, matrix-vector multiplications, non-linear functions.
- What does this look like in the end?

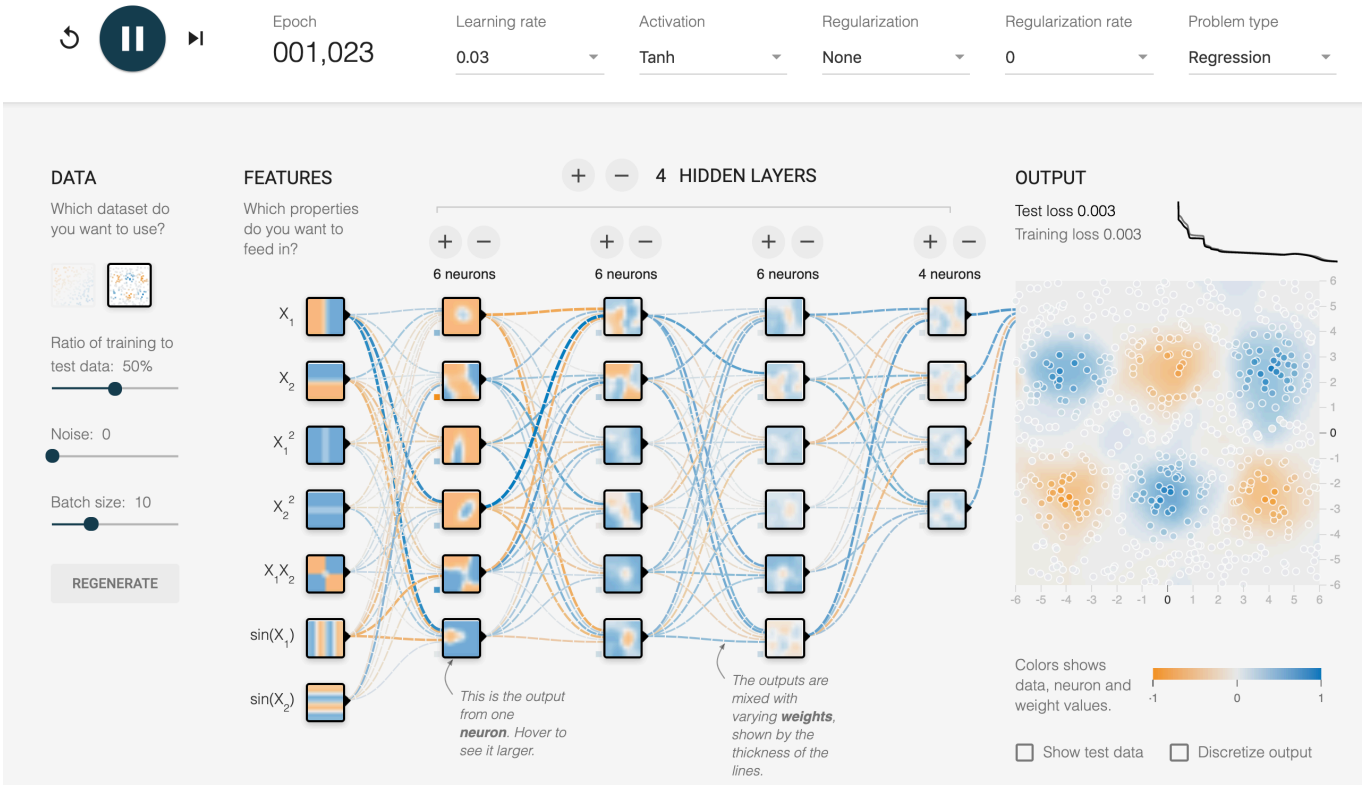
- Universal approximation theorem: any continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely using a single hidden layer. True for a wide range of activation functions.

- Take a simple example where you only have two labels (0 or 1).
- Then the situation may look like the following:


Decision Boundary for hidden layer size 3
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2

Playground

https://playground.tensorflow.org



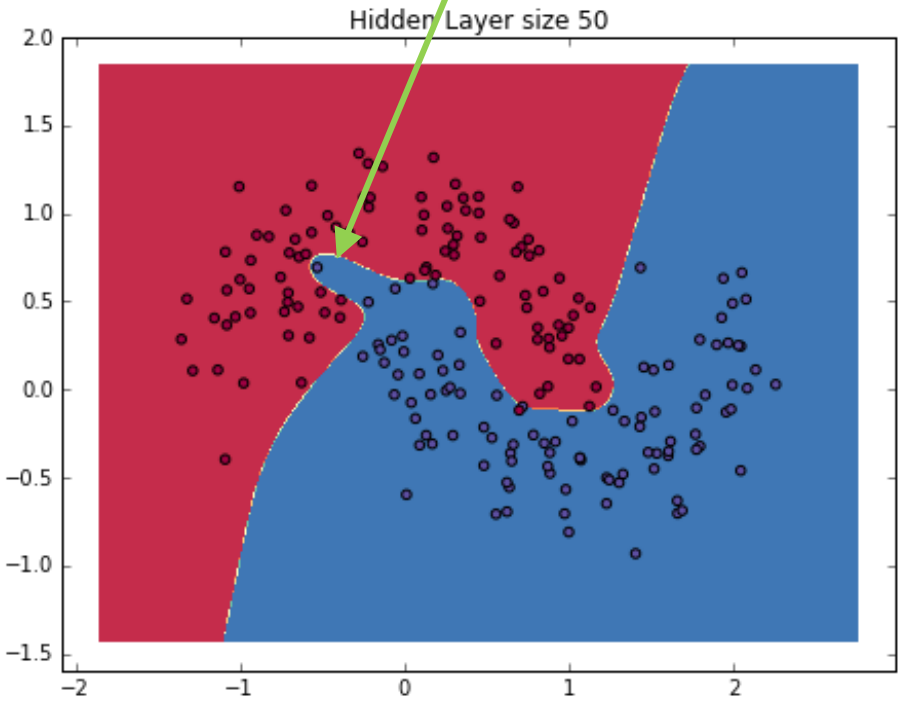
Tuning the network

- Details are given in the handout.
- We start with a training set: given data for which the labels (digits in our case) are known.
- We define an error function that depends on the difference between the labels predicted by the network (say the digit) and the ground truth (the digit a human has determined is shown on the image).
- Then a stochastic gradient descent algorithm is used to minimize this error by adjusting the coefficients W and b of the network.

Overfitting

How realistic is this fit?

Is this noise or a real feature?



Noise

- The problem is that the input data has typically a lot of noise or we may have even some erroneous inputs.
- We cannot trust the data 100%.
- Trying to fit exactly to the given data often makes little sense.
- For example, say we ask you what your best movie is. Your answer may change or may depend on your mood that day.
- So we need to put some constraints in the fit to avoid the problem of overfitting.

Penalization

Measures difference between predicted and true

$$J(W, b; x, y) = \frac{1}{N} \sum_{i=1}^{N} CE^{(i)}(y, \hat{y}) + 0.5 \, \lambda \, \|p\|^2$$

- We add a penalty term at the end.
- p is a vector that contains all the weights W used in the network.
- As a result, W cannot get "too large."

Penalty makes W smaller.

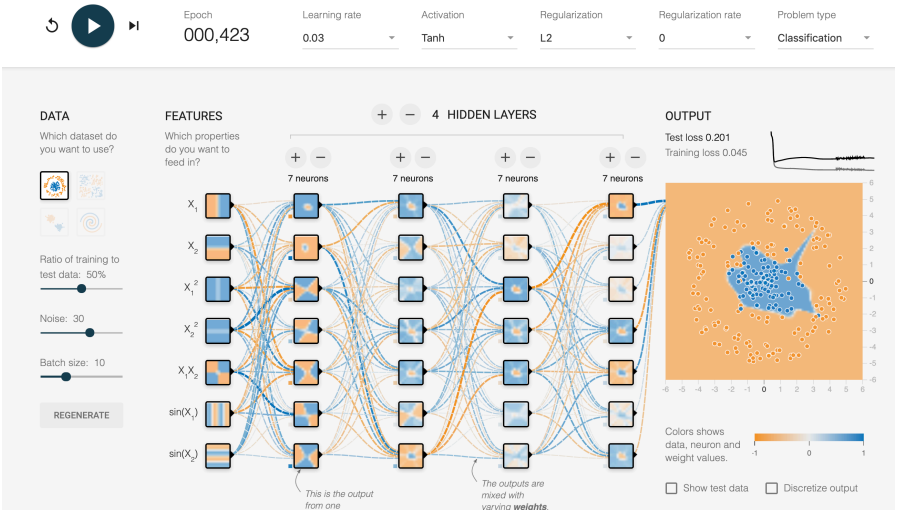
$$z^{(1)} = W^{(1)} x + b^{(1)}$$

$$a^{(1)} = \sigma(z^{(1)})$$

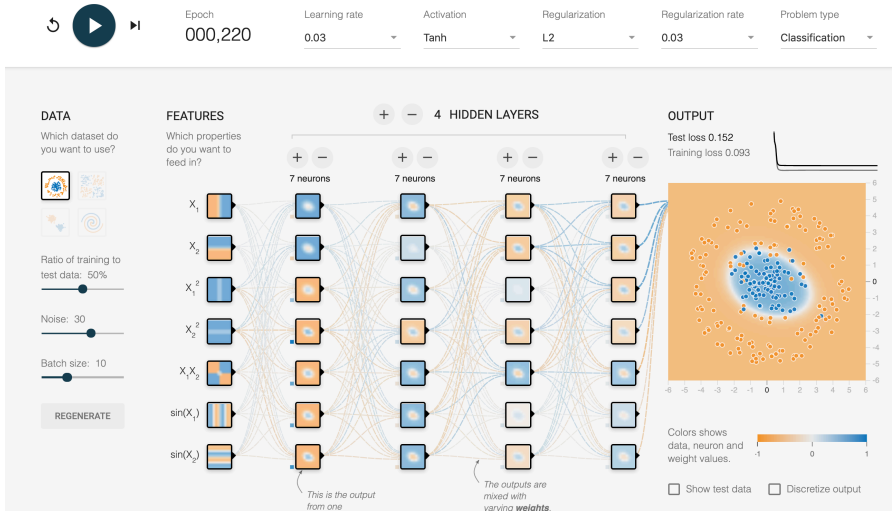
This reduces the non-linearity of the NN.

Example

No regularization



Regularized



Finding the right balance using validation

- The way out of this is to use validation.
- Take your training data. Take out a few data points (say 20%) that will be used for validation.
- Use the training set to optimize the NN coefficients.
- Then test your NN using the validation data.

Outcomes

3 possible outcomes:

Training  Validation 

Overfitting!

Solution: not enough penalization. Increase **λ**.

Training  Validation 

Perfect!

Solution: if it's not broke don't fix it.

Training  Validation 

Terrible!

Solution: too much penalization. Reduce **λ**.

Stochastic gradient descent

- The network coefficients are optimized using a simple gradient descent.

$$p \leftarrow p - \alpha \, \nabla_p J$$

- J is given as a sum over images

$$J = \sum_{i=1}^{N} CE^{(i)}$$

- In practice, we only load some of the images, compute a partial sum, and use its gradient to update the coefficients.
- Convergence guarantees only exist when α varies, but we will use a constant value in this project.

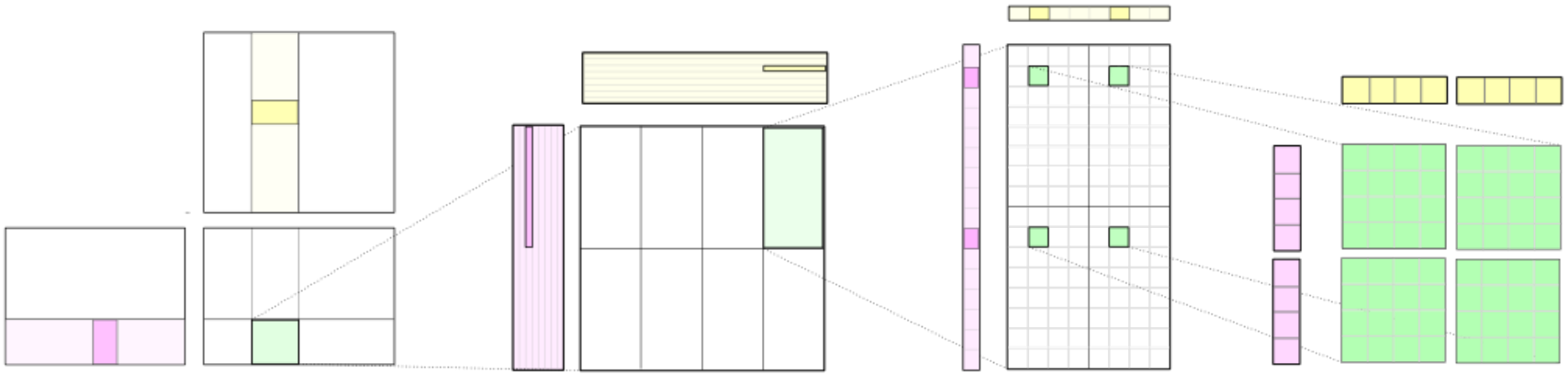
Training

Sequence of steps:

- Load a set of images
- Forward pass: calculate the output of the NN for each image
  This involves going through each layer in the NN from input to output
- Calculate the error or loss function
- Use the gradient to update the NN weights
  - This involves going from output layers back to the input layers; backpropagation
  - Backprop matrix = transpose of forward pass matrix

GEMM

- A big part of the project is writing a routine to efficiently calculate matrix-matrix products or GEMM.
- In the project we link to some starter code to do this (e.g., CUDA programming guide, CUTLASS documentation).
- You can look at this documentation but the code you write must be entirely your own.
- Algorithm 1 (May 31): no use of shared memory
- Algorithm 2 (June 9): use shared memory + tiling



What you need to do

- Implement matrix-matrix products using CUDA
- Implement CUDA kernels to evaluate the non-linear activation functions and softmax.
- Write functions to calculate the output of the network given some inputs (feed-forward).
- Write functions to apply the gradient (back-propagation) and update the network coefficients.
- Write MPI functions to exchange data between nodes: send images, and collect and sum partial gradient contributions from all nodes.
- May 31: report + working parallel code; intermediate milestone
- June 9: 4-page report + optimized CUDA+MPI code

Bonus question

- 15 point extra credit
- Optimize the MPI communications.
- Involve blocking the weight matrices W in different ways in order to reduce communication between nodes.
- Attempt only if you are done with the other parts of the project.