CME 213

SPRING 2017

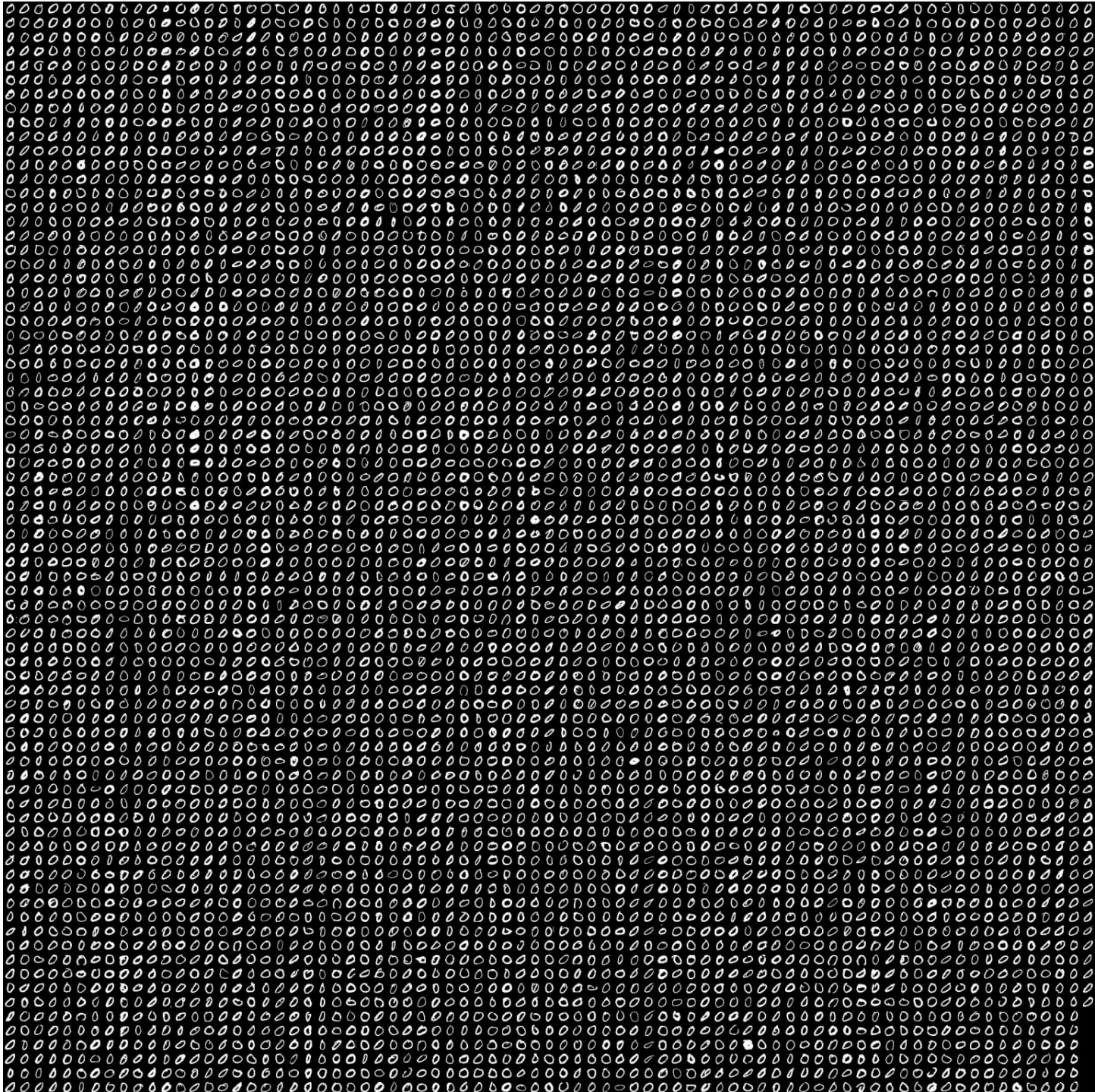
Eric Darve

Final project

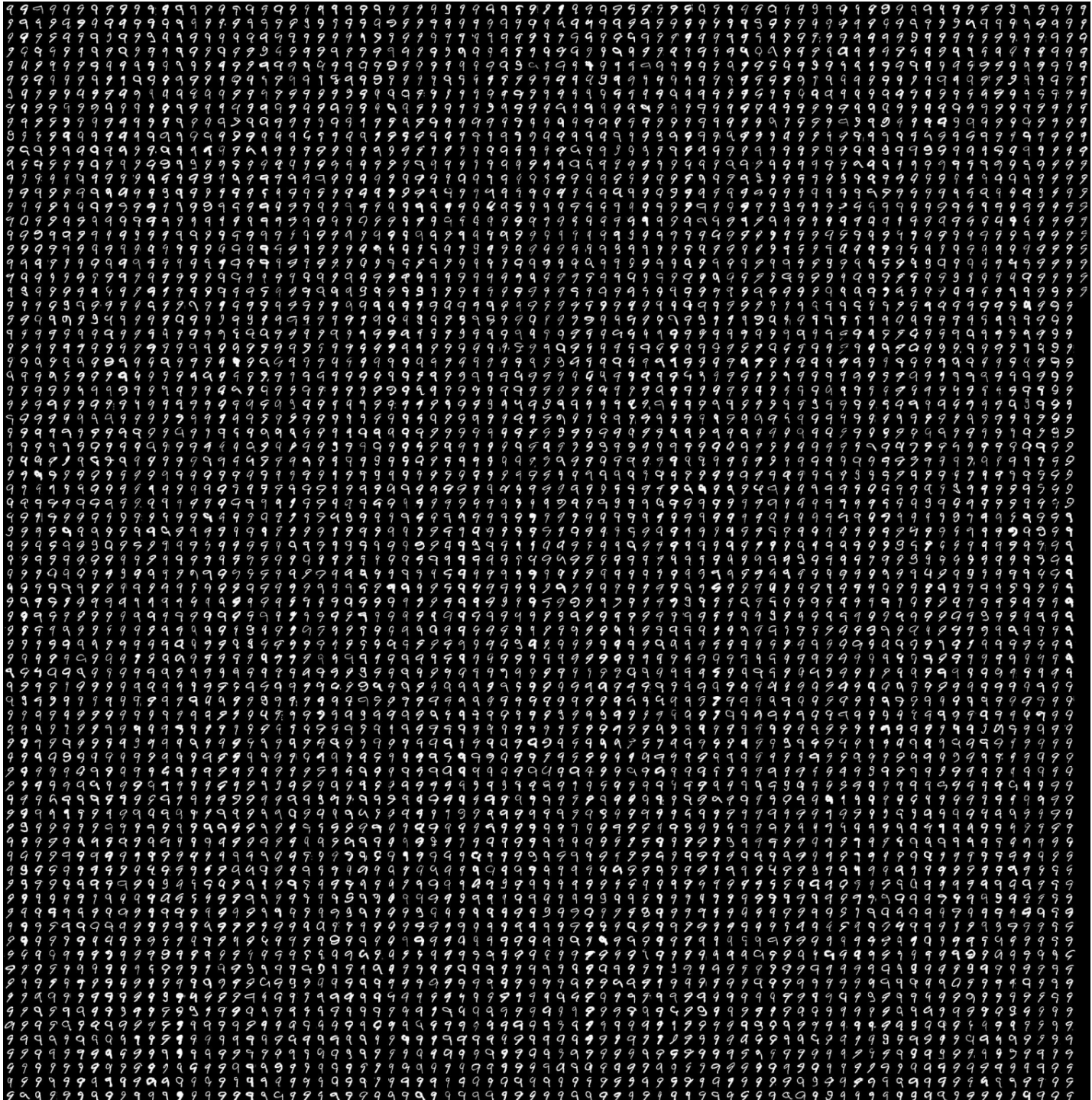
- Final project is about implementing a neural network in order to recognize hand-written digits.
- Logistics:
  - Preliminary report: Friday June 2nd
  - Final report (4 pages): Sunday June 11th

- 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

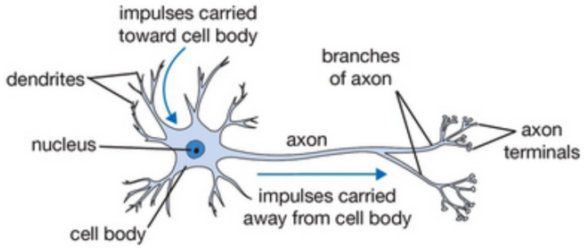


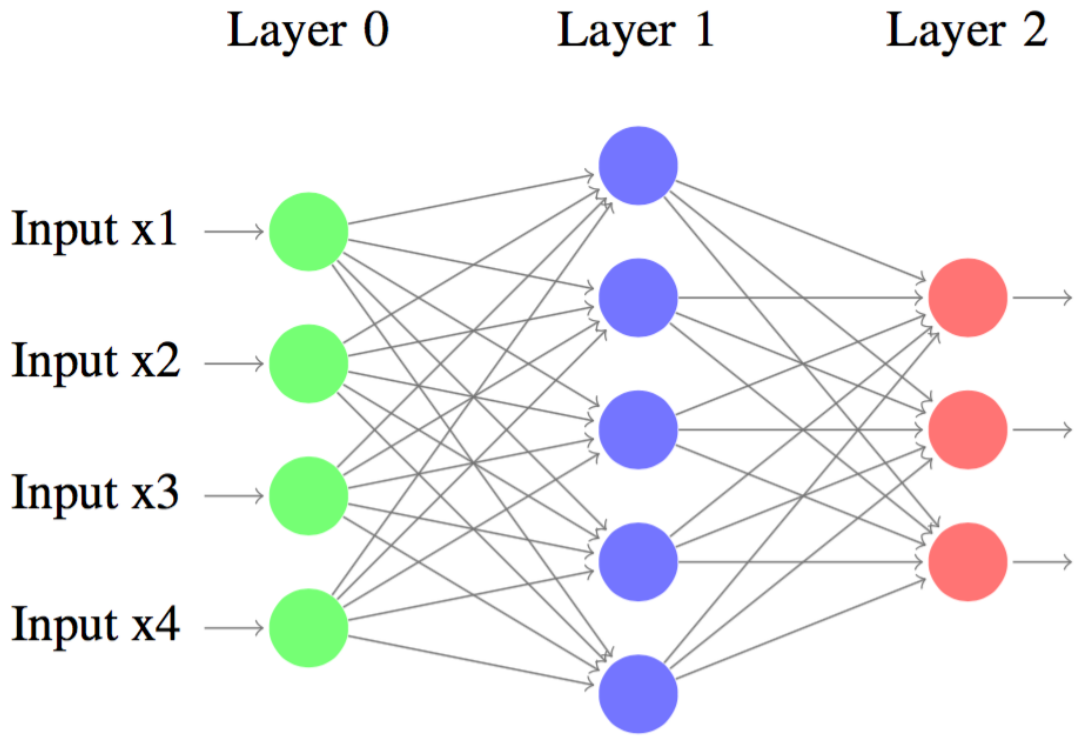
A bunch of Nines



What is a neural network?

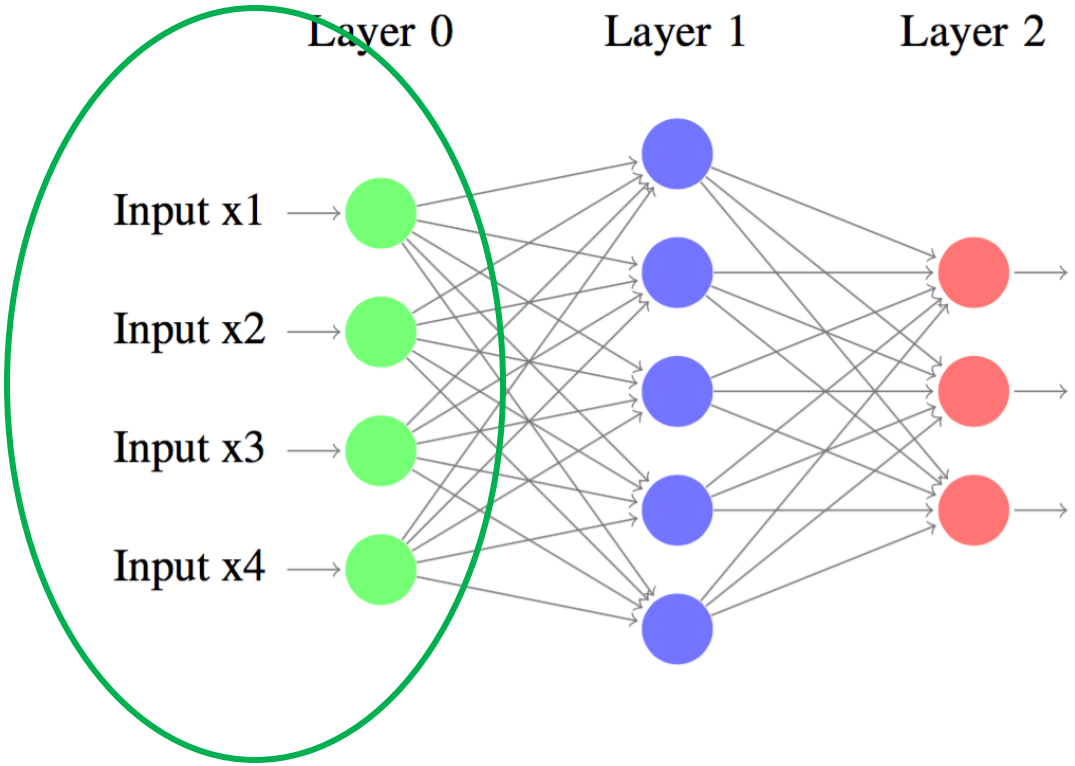
- It has little to do with actual neurons in the brain, although the idea derived from neuron connections in the brain.
- A neural network is a sequence of layers.





Input layer

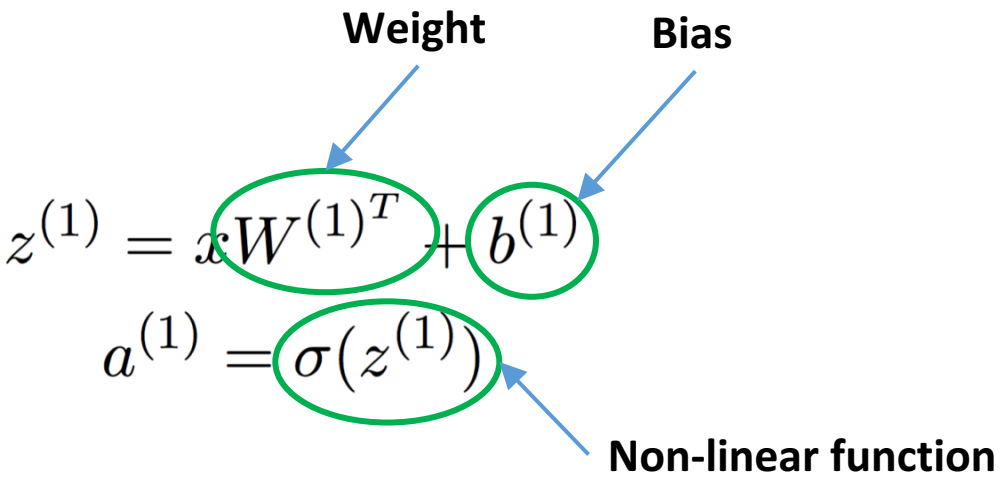
This is the data input, for example all the digits in the image.



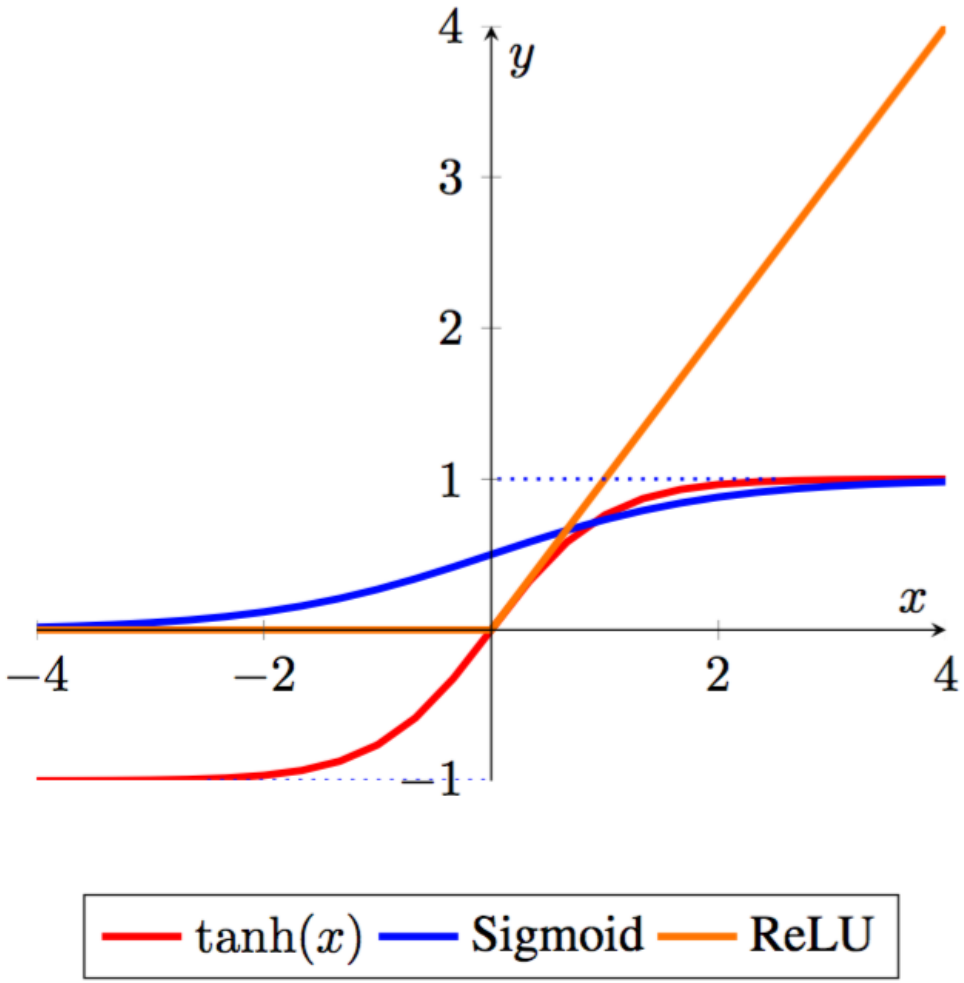
Connection between layers

To calculate data for the next layer, we perform two operations.

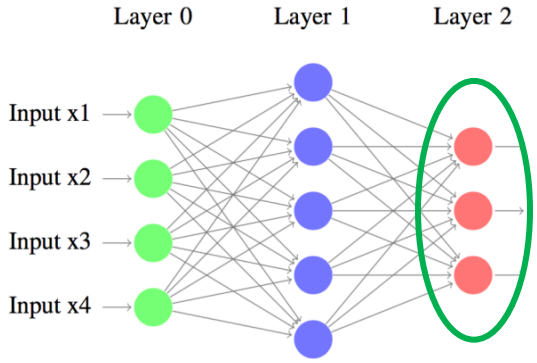
1. **A matrix-vector multiplication with matrix W.**
2. **Then we take the value at each node and apply a non-linear function:**



Examples of non-linear functions typically used



The last layer

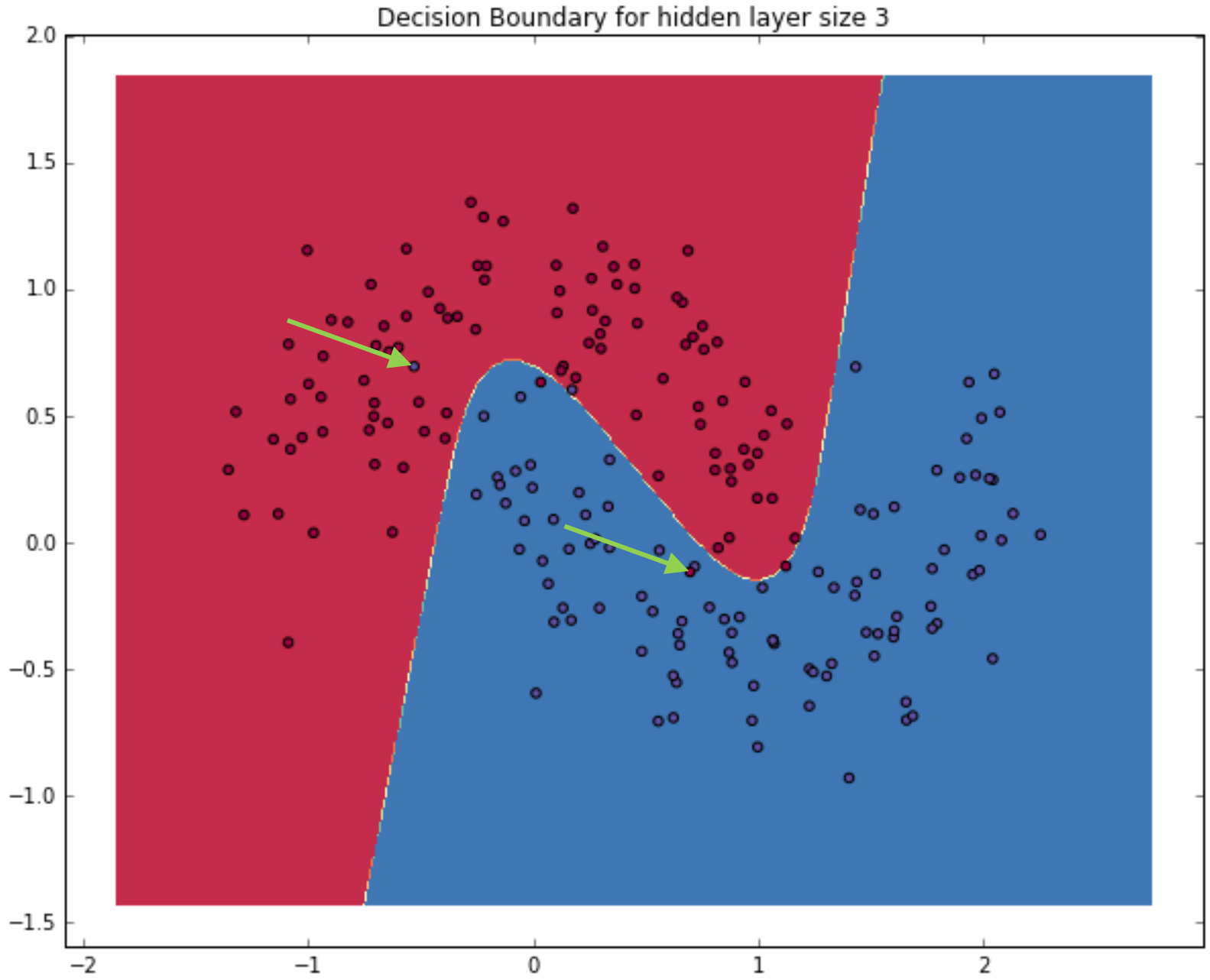


- The last layer is a bit different.
- In our case, we want to produce a probability vector, that gives the probability for each possible digit, from 0 to 9.
- This can be done using a softmax function:

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

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?
- 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

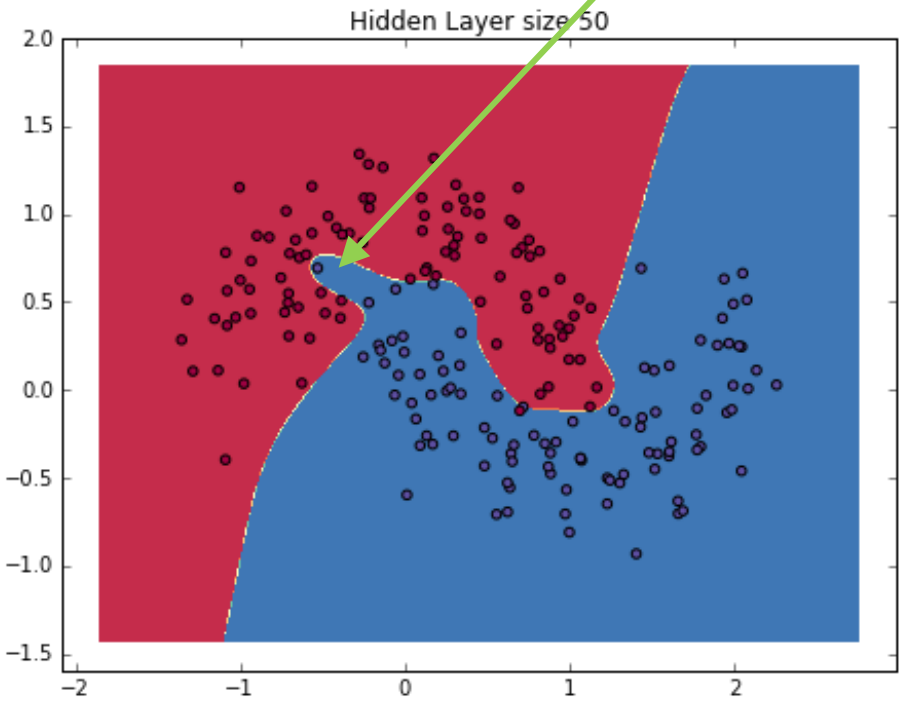
Tuning the network

- 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 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."

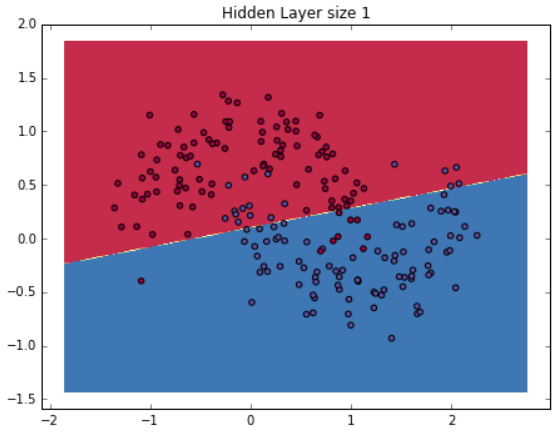
$$z^{(1)} = xW^{(1)^T} + b^{(1)}$$

Penalty makes W smaller.

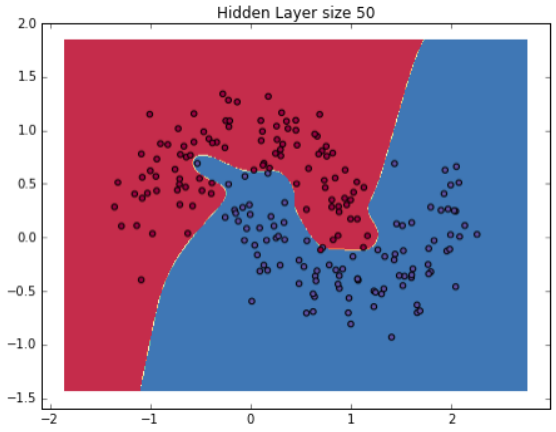
$$a^{(1)} = \sigma(z^{(1)})$$

This reduces the non-linearity of the NN.

Example



Not so good fit but prediction is very robust to noise and data errors.



“Great” fit but very sensitive to data errors.

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:

1. **Training**  **Validation**  **Overfitting!**

   Solution: not enough penalization. Increase λ.

2. **Training**  **Validation**  **Perfect!**

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

3. **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

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

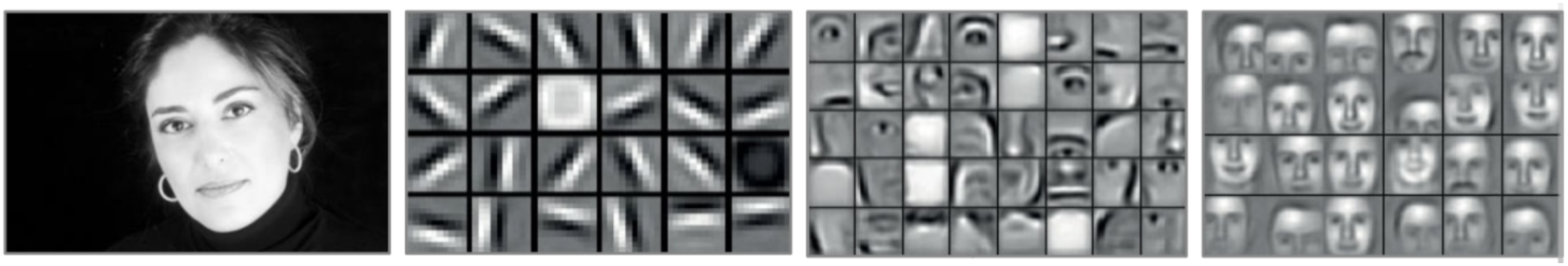
- In practice, we only load some of the images, compute a partial sum, and use its gradient to update the coefficients.
- Convergence guarantees only exist when α varies, but we will use a constant value in this project.

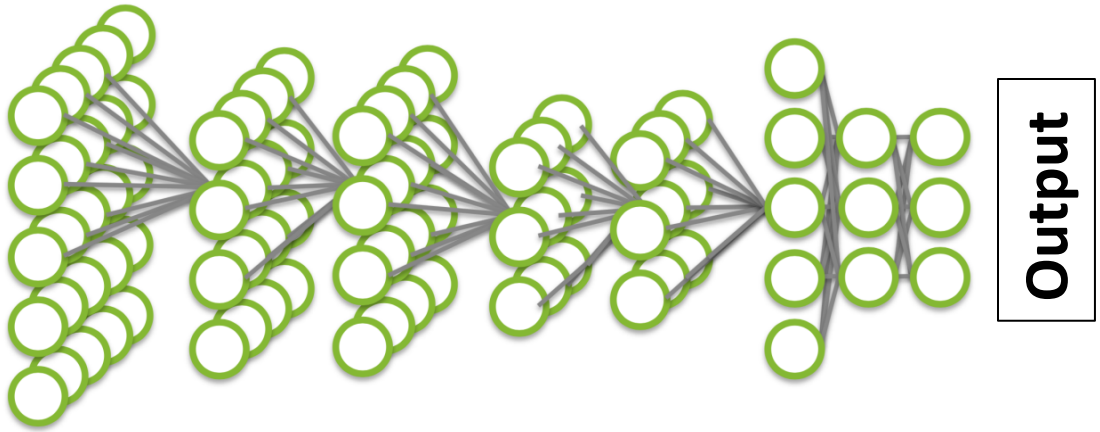
What you need to do

- **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.**

Neural networks can be very complex and powerful

Raw data | Low-level features | Mid-level features | High-level features





Application components:

- **Task objective**
  e.g., identify face
- **Training data**
  10-100M images
- **Network architecture**
  ~10 layers
  1B parameters
- **Learning algorithm**
  ~30 Exaflops
  ~30 GPU days

Visit http://cs231n.stanford.edu/

Neural network for AlphaGo

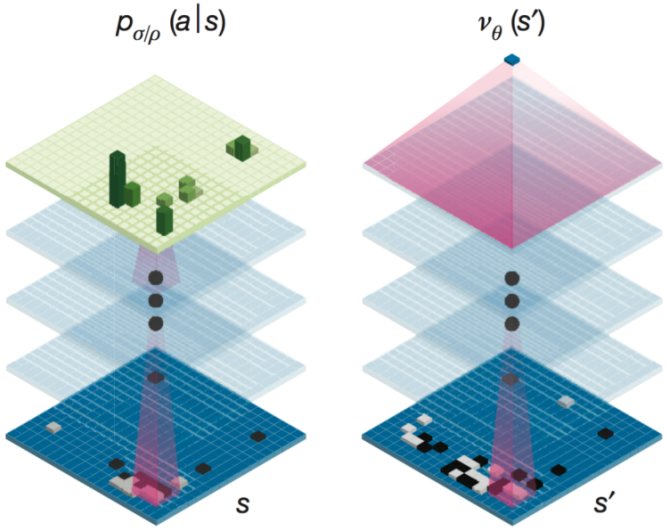
Go is difficult! b^d possible moves where

- **Chess: b ~ 35, d ~ 80**
- **Go: b ~ 250, d ~ 150**

This is massive!

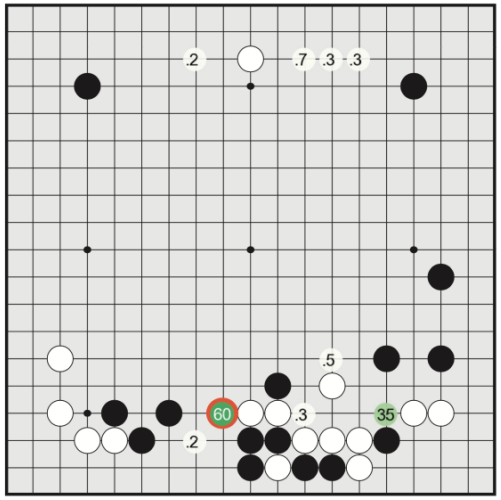
Two networks were developed to play Go:

1. **Policy network: predict what the next best move is.**
2. **Value network: predict how strong the current position is.**



Policy Value

Structure of network

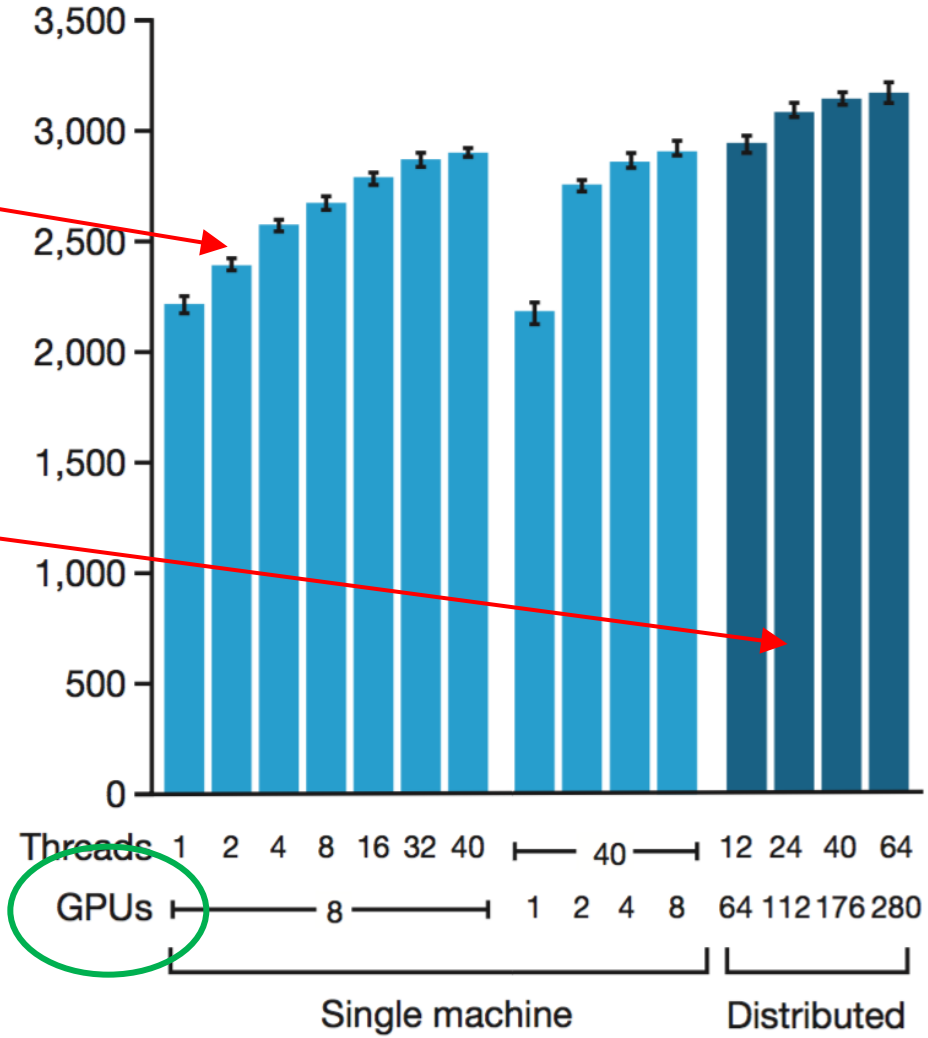


- Example with policy network.
- Input is 19 x 19 x 48 image stack.
- 19x19 is the size of the Go board.
- 48 features: stone color, ones (just a constant plane with 1), turns since, liberties (empty adjacent points, a key Go feature), capture size (how many stones can be captured), self-atari size (own stones), liberties after move, ladder capture (a special sequence of Go moves), ladder escape, sensibleness (is that even a legal move?), zeros (constant plane with 0 this time).
- A big NN is used: 1) k filters of kernel size 5 × 5 + rectifier nonlinearity. 2) k filters of size 3 × 3 + RNL . 3) The final layer: 1 filter of size 1 × 1, with a different bias for each position, and the softmax function. AlphaGo uses k = 192.

Massive parallelism is used

Shared memory

Distributed memory



Training: KGS

- Two different types of training were used.
- Reproduce known master games: study games from KGS (the Kiseido Go Server).
- 29.4 million positions from 160,000 games played by KGS 6 to 9 dan human players.
- The data set was split into a validation set (the first million positions) and a training set (the remaining 28.4 million positions).
- NN is trained is order to successfully reproduce all the winning moves in these games.

Reinforcement learning

- This is unique to board games and computers.
- Algorithms can play against themselves and get better!
- The program plays against earlier versions of itself and determines what moves lead to victory/defeat.
- Then NN is adjusted so that winning moves are reinforced, while losing moves are de-emphasized.
- It's called reinforcement learning.

“It’s not a human move”

- Interestingly, the step of reinforcement learning means that the computer can learn how to play Go using techniques that no human has ever taught it.
- In the second game of the Go match between Lee Sedol and AlphaGo, the computer made a move that flummoxed everyone including Lee Sedol himself. “That’s a very strange move,” said one commentator. “I thought it was a mistake,” said the other. And Lee Sedol, after leaving the match room for a spell, needed nearly fifteen minutes to settle on a response.
- Fan Hui, three-time European Go champion: “It’s not a human move. I’ve never seen a human play this move.” But he also called the move: “So beautiful. So beautiful.”
- Indeed, it changed the path of play, and AlphaGo went on to win the second game.

Upcoming AlphaGo matches

- **May 23–27, Wuzhen, China, *Future of Go Summit*.**
- **The games will include:**
- **“Pair Go” — A game where one Chinese pro will play against another... except they will both have their own AlphaGo teammate, alternating moves**
- **“Team Go” — A game between AlphaGo and a five-player team consisting of China’s top pro players, working together to test AlphaGo’s creativity and adaptability to their combined style**
- **“Ke Jie vs AlphaGo” — classic 1:1 match of three games between AlphaGo and the world’s number one player, Ke Jie**