Topics in Astronomical Research
Computational Astrophysics

PHYS 913

Dr. Mark Richardson
ZoomID: 990 2595 7333: Pass: PHYS913
Mark.Richardson@queensu.ca

Today:

- Machine Learning
  - Overview, Supervised, Unsupervised, and other forms
  - Supervised learning & Convolution Neural Networks
  - Facilitation
- Course Conclusion
- Presentation Logistics

Machine Learning

- Broadly speaking:
  - An automatic system to learn, improve, optimize, and/or characterize data without being explicitly programmed to do so in a particular way.
- Common examples/uses:
  - Image classification: Hand writing, facial recognition, crater counting, Animal recognition etc.
  - SNe and other transient identification (visual, audio, etc.)
  - Diagnosis
- Given how quick humans are for some classification, it's surprising how hard it is to program such a process.

Machine Learning

- In the 60s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene. He figured they'd have the problem solved by the end of the summer. Half a century later, we're still working on it.

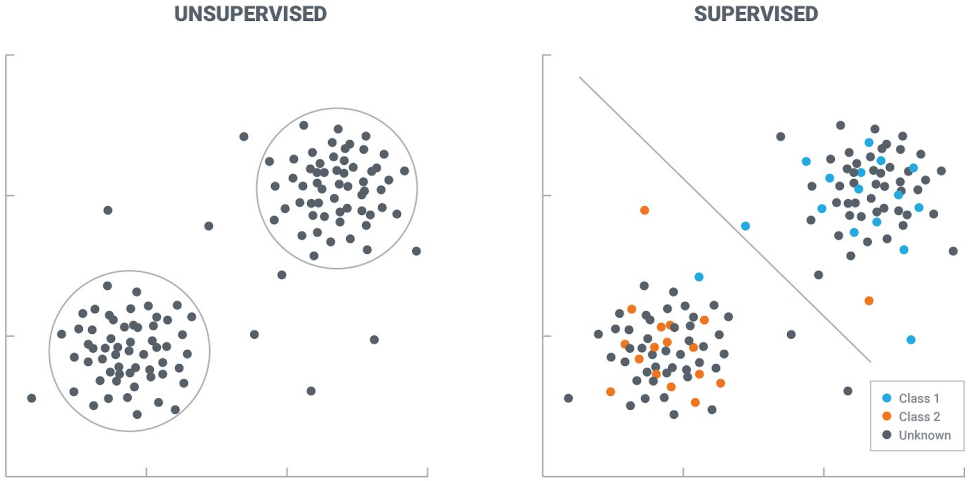
xkcd

Machine Learning: Unsupervised vs Supervised

- Supervised: You have training data that is already characterized, and you use this data train your machine learning program to characterize input data.
- Supervised: Typically done with Neural Networks. We will mostly look at these.
- Examples: Classification (Dog or Cat? Chihuahua or Muffin), Regression (result of a bunch of data; e.g., stock market or stellar mass, etc.)

Machine Learning: Unsupervised vs Supervised

- Unsupervised: Finds features, clusters, or associations in a dataset. Typically exploring high-dimensional dataspace where visually characterizing the data is difficult for humans.
- Here there is no clear answer (at least not one known in advance)
- Unsupervised: Commonly principal component analysis & cluster analysis.



Other machine learning

- Reinforcement learning: Determine a transition process (e.g., in a Markov Chain) to maximize a reward measure.

- Genetic Algorithms: Sometimes paired with e.g., reinforcement learning, using the ideas of genetic offspring to let successful models procreate and create new ones e.g., https://rednuht.org/genetic_cars_2/. The idea is there is a parameterization of their state. These act as chromosomes that are split between two parents when they mate to create a new proposed state. Also include mutations.

- Semi-supervised: Mix of supervised (known answers) and unsupervised.

Supervised Learning: Neural Networks

- Neural networks are an approach to supervised machine learning.
- Neural networks attempt to recreate the action of neurons in the brain.
- Some uses include:
  - Image recognition
  - AI for games
  - Data classification
  - Finance

Neural Networks

- Basic idea is to create a nonlinear fitting routine with free parameters.
- The network is trained on data with known inputs and outputs (e.g., images of hand-written numbers and the corresponding actual number).
- Once trained, a network can then predict an output given a new input.
- An example of linear network is **y = Ax**, where **x** is your inputs (e.g., the pixel values of an image), and **y** is a vector telling you how confident the computer is the the number is y_i. **A** is the map that acts on the input to predict the output.

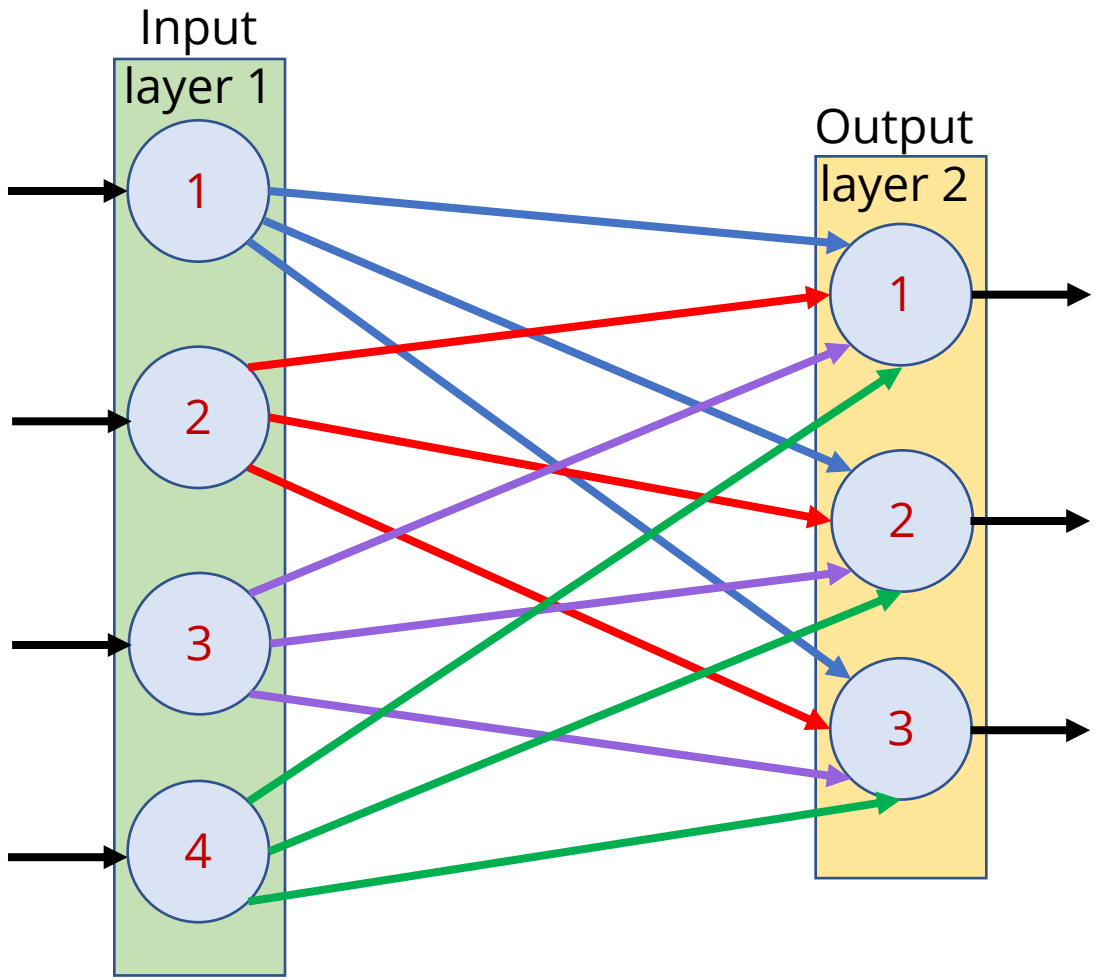
Linear networks don't work

- Consider
- $\boldsymbol{x}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \boldsymbol{x}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \boldsymbol{x}_3 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$
- $\boldsymbol{y}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \boldsymbol{y}_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \boldsymbol{y}_3 = \begin{pmatrix} 0 \\ -8 \end{pmatrix}$
- Where **y=Ax** such that:
- $\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{x}_1;\ \boldsymbol{y}_2 = \boldsymbol{A}\boldsymbol{x}_2;\ \ \boldsymbol{y}_3 = \boldsymbol{A}\boldsymbol{x}_3$

- All three cannot be satisfied with a linear model.

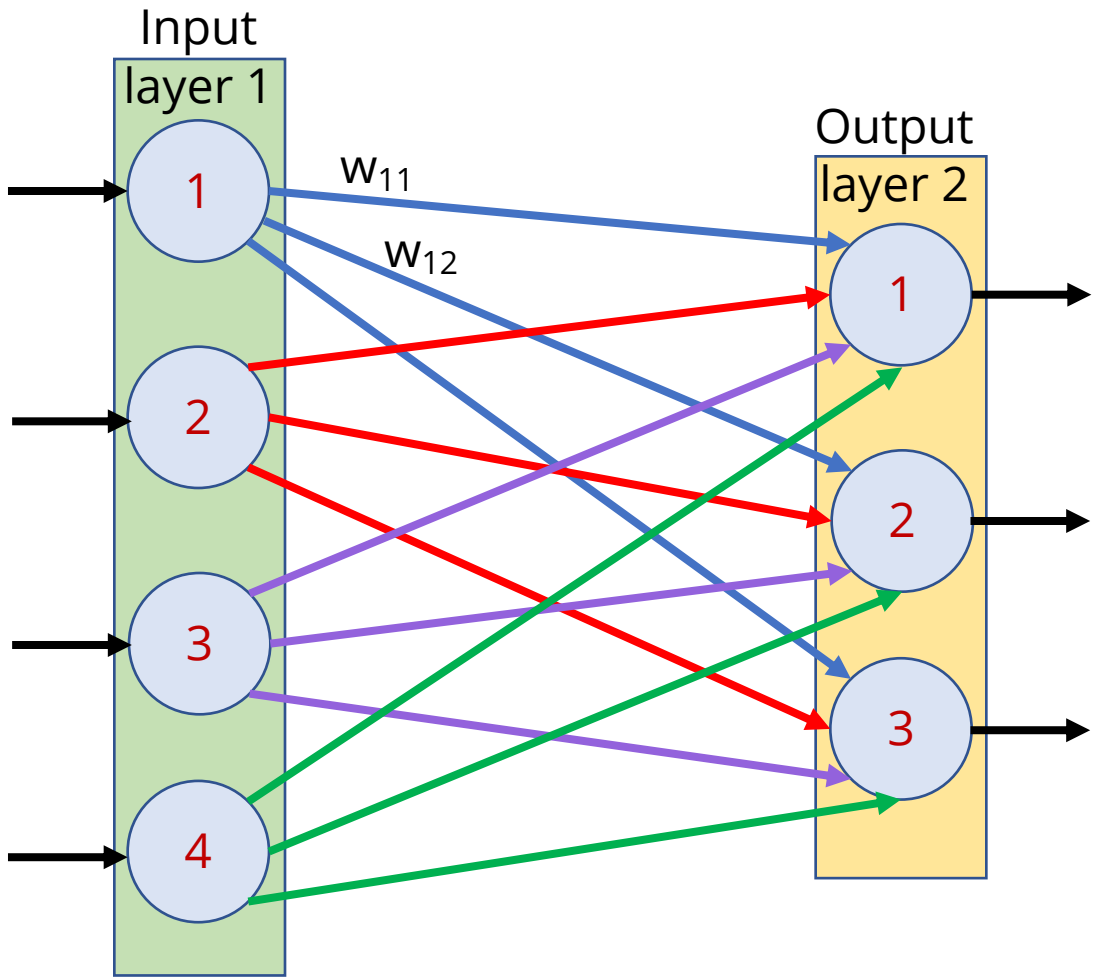
Neural Networks: The Lingo

- Neural networks are divided into *layers*.
  - Always an input layer which is essentially the input.
  - There is always an output layer.
- Within a layer there are neurons, or *nodes*.
  - For input, there will be one node for each input variable.
- Every node in the first layer connects to every node in the next layer.
  - Weights connecting nodes can vary.
- Here we see the processing done in layer 2 (output).



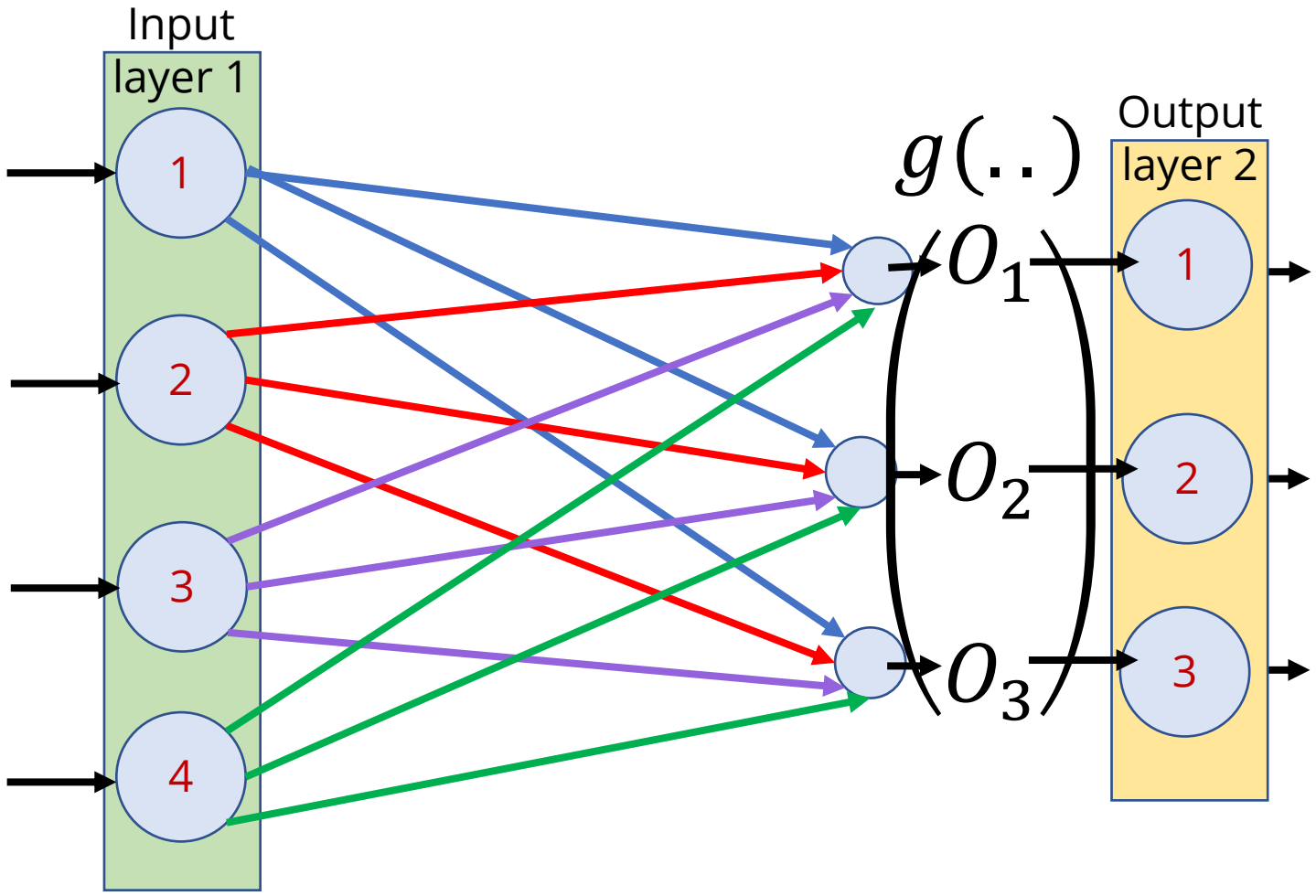
Neural Network Approach

- Training a neural network means adjusting the weights connecting nodes.
- Weights can be zero.
- Equivalent to nature: neurons in the brain can connect to many or a few other neurons.
- The weights are the values in **A**.
- We're basically trying to solve for these weights so that when I give the network an input, it produces an output that matches the test result.



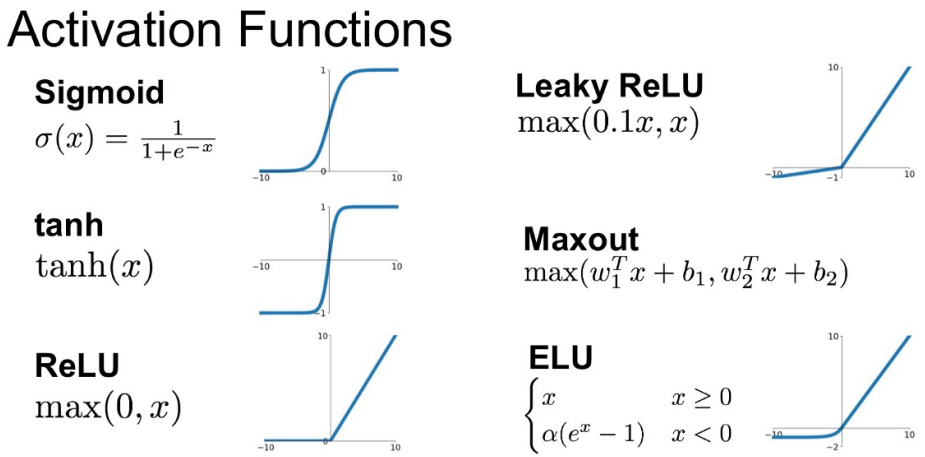
Nonlinear model

- Here we can use a nonlinear function, g(p), that acts on a vector, e.g., g(**x**) = $\begin{pmatrix} g(x_1) \\ g(x_2) \end{pmatrix}$
- Then **y** = g(**Ax**). E.g., g(p) = p^3 would fit all the data in the previous example.
- g(p) is prescribed so we still only train the network to determine A_{ij}, and then g acts on the output layer:



Nonlinear Model

- The use of g(p) still mirrors biology.
  - Neurons don't act linearly.
  - There is an activation threshold required before a neuron will "fire"
- What should we use for g(p)? Typically something approaching a step function is used, albeit one continuous and differentiable.
- Many choices in the literature

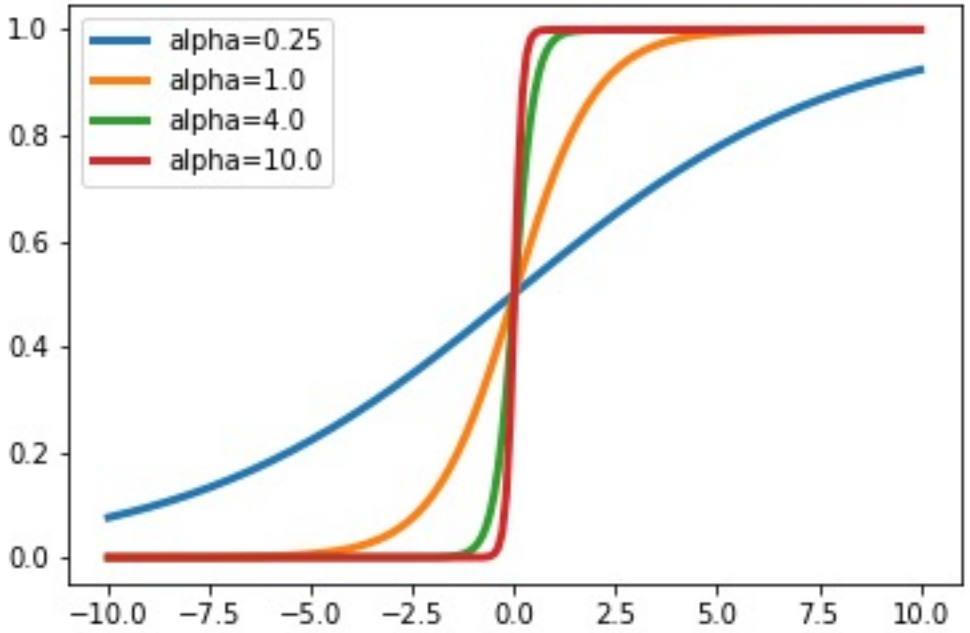


Sigmoid Activation Function

- A common choice is the sigmoid function:
- $g(p) = \frac{1}{1+e^{-\alpha x}}$
- Note: all outputs on (0,1) exclusive.

- Some discussion about α, but a value of 1 is fine. Could rescale to (0,1].
- Don't want a 0 value as then the weights are unconstrained.

- Note some saturation for large input.

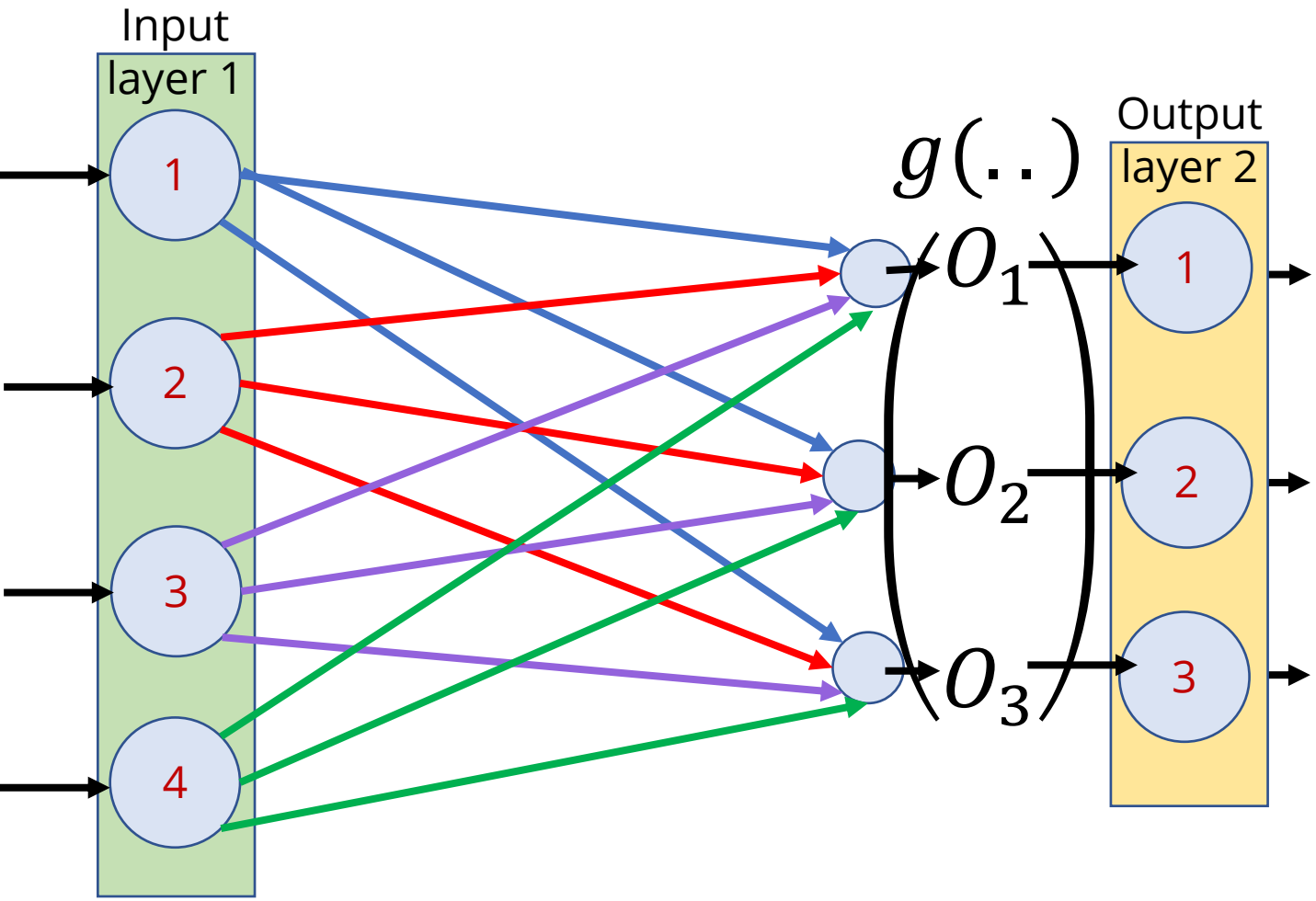


Scaling

- So we've constructed a function that takes us from input to output: **y =** g(**Ax**), but for a sigmoid, we know the output is on (0,1).
- Need to make sure the true values we're trying to recover are also on (0,1). If not:

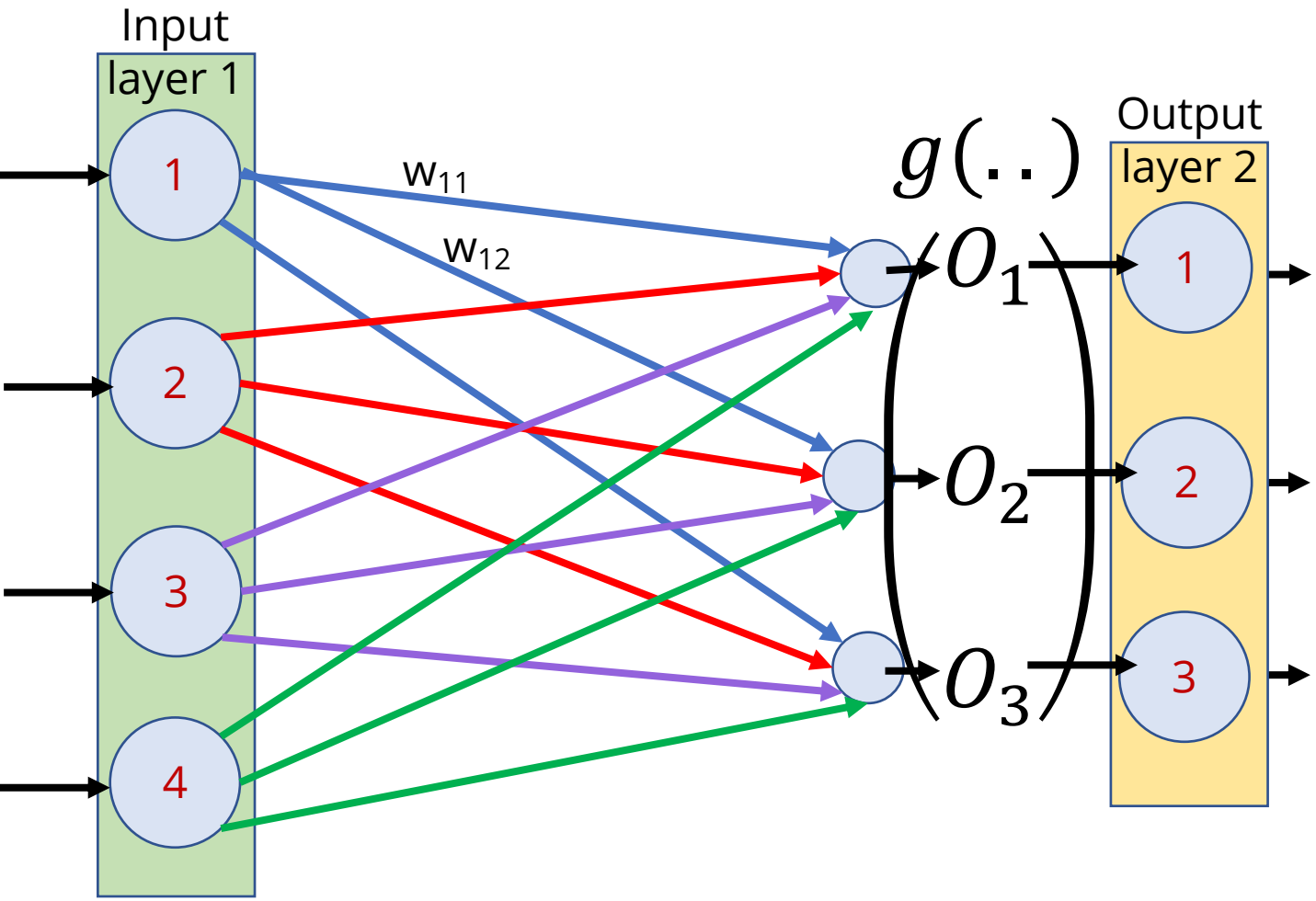
- $y' = 0.99(y - \min\_y)/\Delta y + 0.01$.

So we're basically set!



- We have N training input/(rescaled) output pairs ($\mathbf{x}^k$,$\mathbf{y}^k$).
- Each input is of dimension *n*, each output of dimension *m*.
- E.g., A galaxy image 128x128 pixels could be the input data with *n*=128x128=16384, and the output could be of dimension *m*=2, with (1,0) being a disk galaxy and (0,1) an elliptical galaxy..

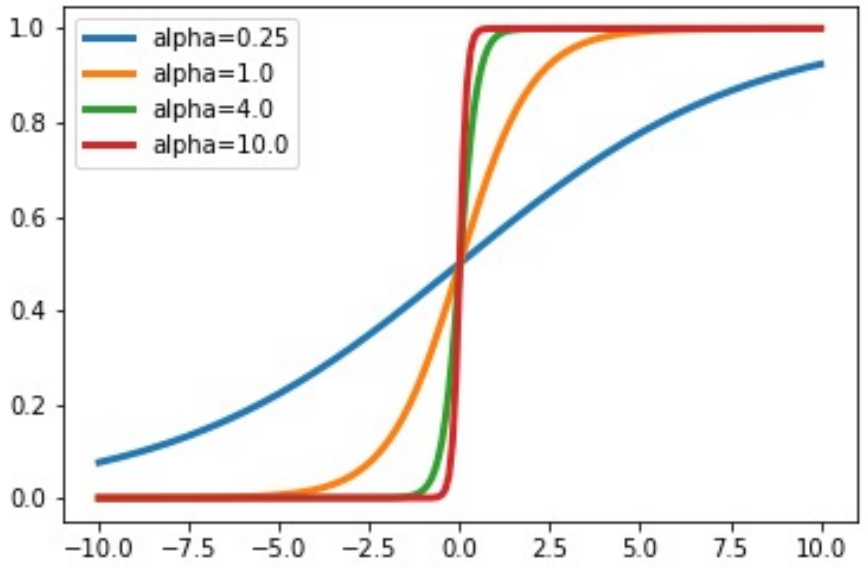
So we're basically set!



- We have a linear matrix of weights, $A_{ij} = w_{ij}$ (that are unknown).
- For some guess of **A**, we can have $\mathbf{z}^k = g(\mathbf{A}\mathbf{x}^k)$. Then we need to adjust **A** to minimize:
- $f(A_{ij}) = \|\boldsymbol{z}^k - \boldsymbol{y}^k\|^2$
- This is the cost function we want to minimize, and it's actually summed over all input/output pairs.

A variation

- Recall the sigmoid we have chosen for *g(p)*:



- It 'activates' at p=0. This can be offset with a bias: $\mathbf{z}^k = g(\mathbf{A}\mathbf{x}^k + \mathbf{b})$. This allows more variation in **A**, but not strictly necessary.

Implementation

- As we've seen, this is a minimization problem:
- $f(A_{ij}) = \left\| g(\boldsymbol{A}\boldsymbol{x}^k) - \boldsymbol{y}^k \right\|^2$ → f is the *cost function*.
- In the literature you may see different choices for the cost function than the error squared.

Implementation

- Minimization of the cost function.
  - The common technique is just to use *gradient descent* that we saw last class.
  - Neural networks are complicated enough, we don't need to use other more complicated optimization routines.
  - Gradient descent looks at the local derivative of the cost function with respect to the *m* x *n* weights in A_{ij}, and moves a small distance downhill.
- Note, you could imagine doing this for the total cost function after running through the entire dataset. However, it's much easier to take one small downhill step according to the gradient from a single input/output training pair.
- The practice is to randomly go through your whole training data pairs, making small edits to A_{ij} along the way. Each sweep through the whole dataset is an *epoch*.

Minimization is Learning

- So for a given training pair we have: $f(A_{ij}) = \left\| g(\boldsymbol{A}\boldsymbol{x}^k) - \boldsymbol{y}^k \right\|^2$
- So: $A_{ij} = A_{ij} - \eta \frac{\partial f}{\partial A_{ij}}$
- Where:
- $f(A_{ij}) = \sum_{i=1}^{m} \left[g\left(\sum_{j=1}^{n} A_{ij} x_j \right) - y_i \right]^2$
- Thus:
- $\frac{\partial f}{\partial A_{ij}} = 2(z_i - y_i)\alpha z_i (1 - z_i) x_j$
- Again, at each training pair we evaluate the gradient and take on small step of length η, which is called the *learning rate*.

Minimization tidied up

- Recall that:
  - **x** is an *n* x 1 vector.
  - **y** is an *m* x 1 vector.
  - **A** is an *m* x n vector.

- So we can write the derivative as:
- $\frac{\partial f}{\partial \boldsymbol{A}} = [2(\boldsymbol{z} - \boldsymbol{y}) \circ \alpha \boldsymbol{z} \circ (1 - \mathbf{z})] \cdot \boldsymbol{x}^T$. (∘ is the Hadamard product: element by element)
- (mxn) = (m x 1) (1xn)
- If $\Delta\mathbf{A} = -\eta \frac{\partial f}{\partial \boldsymbol{A}}$, then the gradient descent stage is $\mathbf{A} \leftarrow \mathbf{A} + \Delta\mathbf{A}$

Initialization

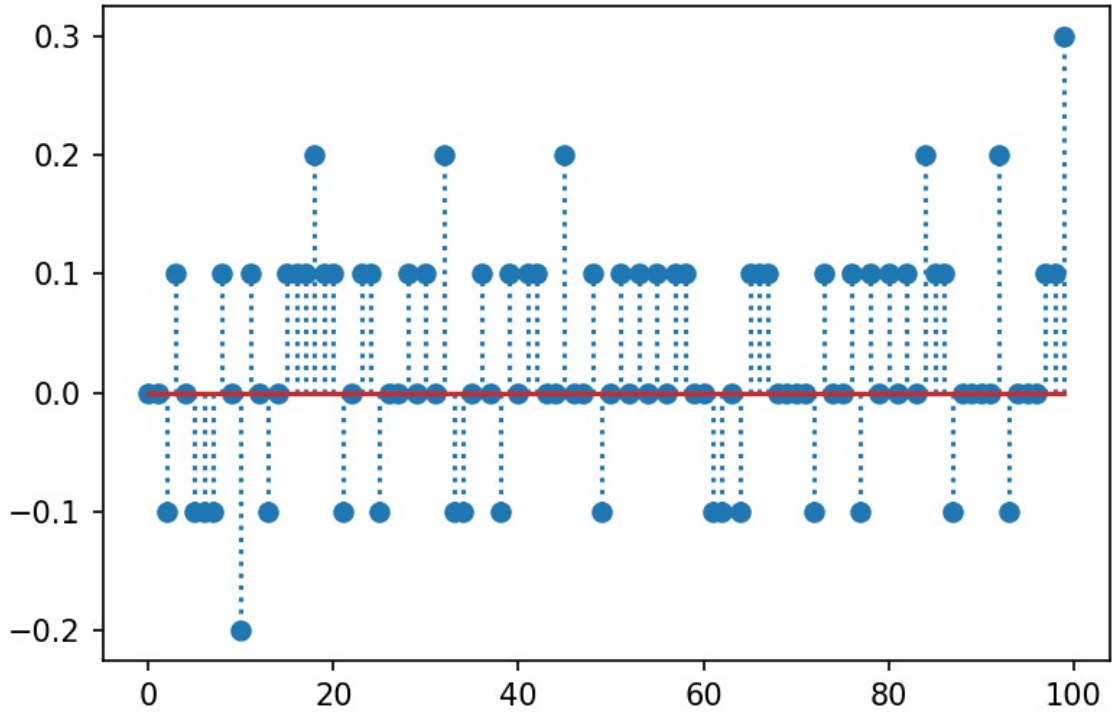
- What should our first guess at **A** be?
- Some options in the literature:
  - Random values on [-1:1]
  - Gaussian normal random numbers about 0 with standard deviation = $1/n^{0.5}$, with $\alpha=1$.
- Recall, gradient descent will find the local minima relative to the initialization. We may not find the global minimum.

Example (from Michael Zingale)

- Consider an input vector of 10 numbers, drawn randomly from the sample:
  - [0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95]
- Ex: In: [0.15, 0.55, 0.25, 0.15, 0.75, 0.95, 0.75, 0.35, 0.45, 0.45]
  - Out: [0.45]
- We want to train a neural network on a bunch of in/out pairs and see how it does at predicting the correct output for new input.
- We'll force the output to select the closest input from the sample.

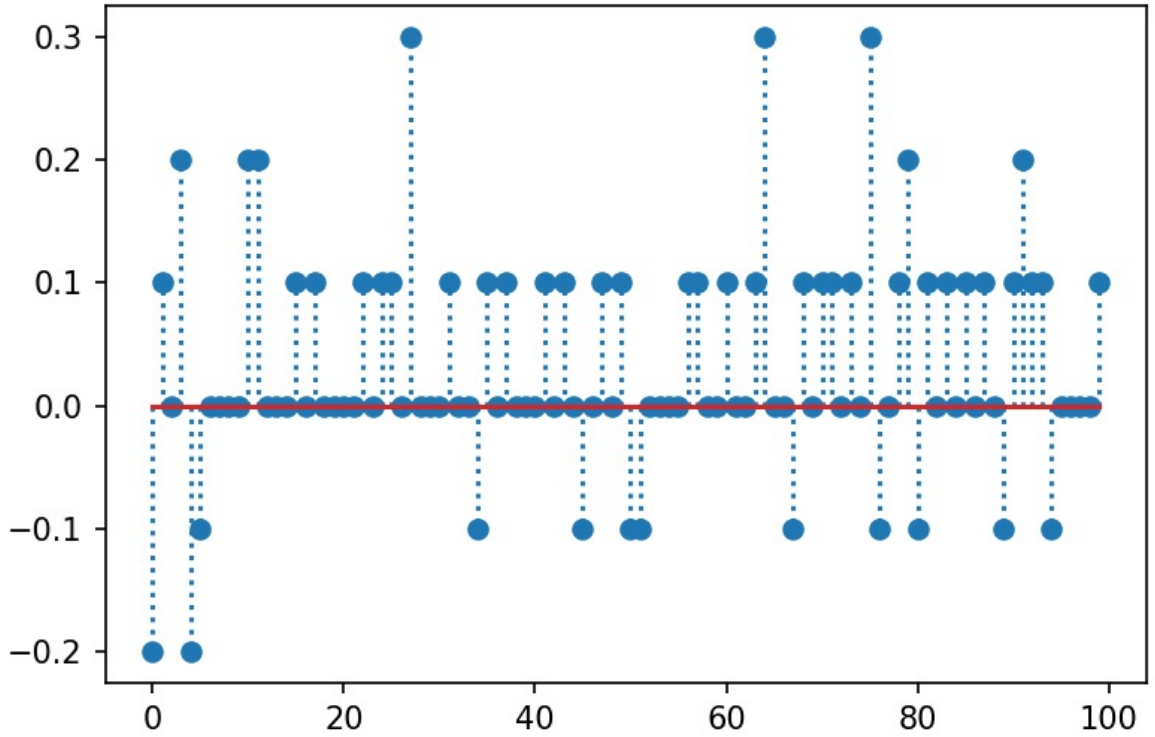
Example

- After training, the model does okay. Here we plot how far the output was from the trained data:



Example

- Now with new testing data it gets 50% correct:

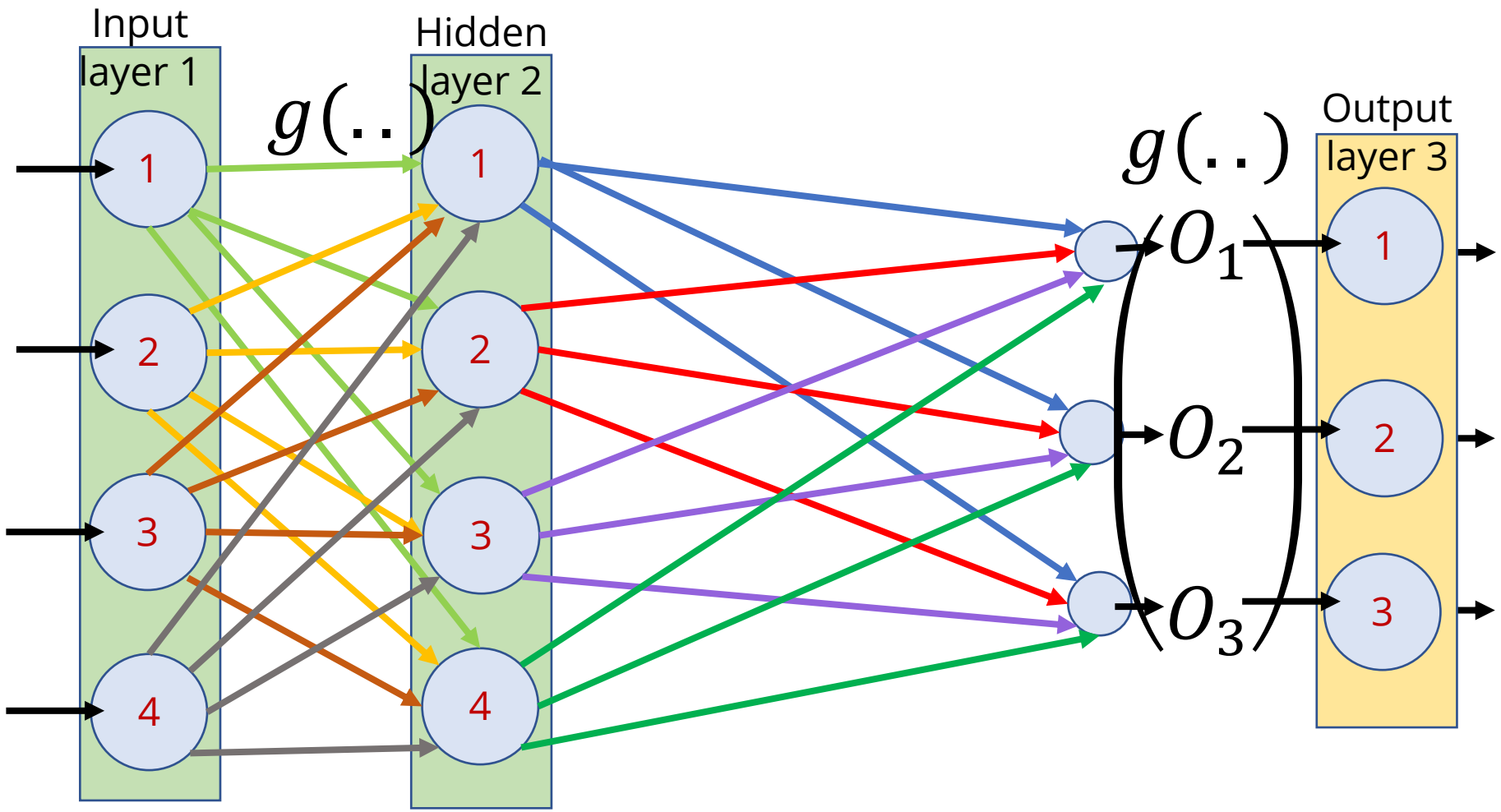


Example

- The final weight matrix (10 x 1) has elements:
- [[-0.53676946 -0.36854924 -0.43439534 -0.80163292 -0.38777711 -0.42561904 -0.43087944 -0.57355182 -0.47560416 4.84437775]]
- Note that the last index is much larger than the rest, as expected.

Can we do better? Hidden Layers

- We can add more parameters with addition layer(s) of nodes:



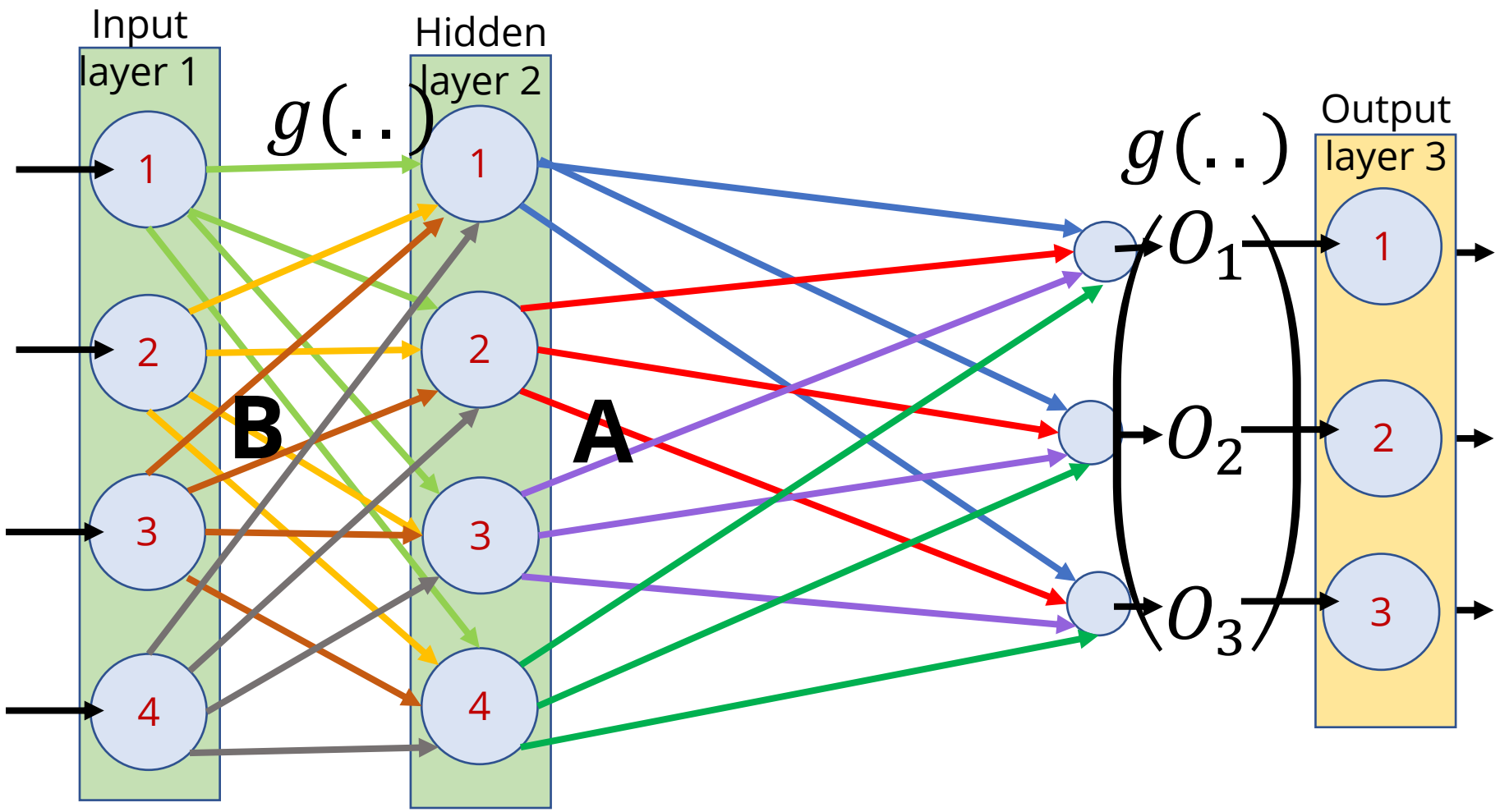
Hidden Layers

- *Hidden layers* live between the input and output layers.
- Let the hidden layer have dimension *k*. Typically *n* <= *k* <= *m*.
- So now:
  - **x** is an *n* x 1 vector.
  - **y** is an *m* x 1 vector.
  - **A** is an *m* x k vector.
  - **B** is an k x n vector.
- Now the product **AB** is *m x n*.
- From here we explicitly assume α=1 in *g(p)*.

Hidden Layers

- Lets break it down into two steps:
- $\mathbf{z'} = g(\mathbf{B}\mathbf{x})$; $\mathbf{z} = g(\mathbf{A}\mathbf{z'})$

This looks like: (recall k need not be n)



Learning

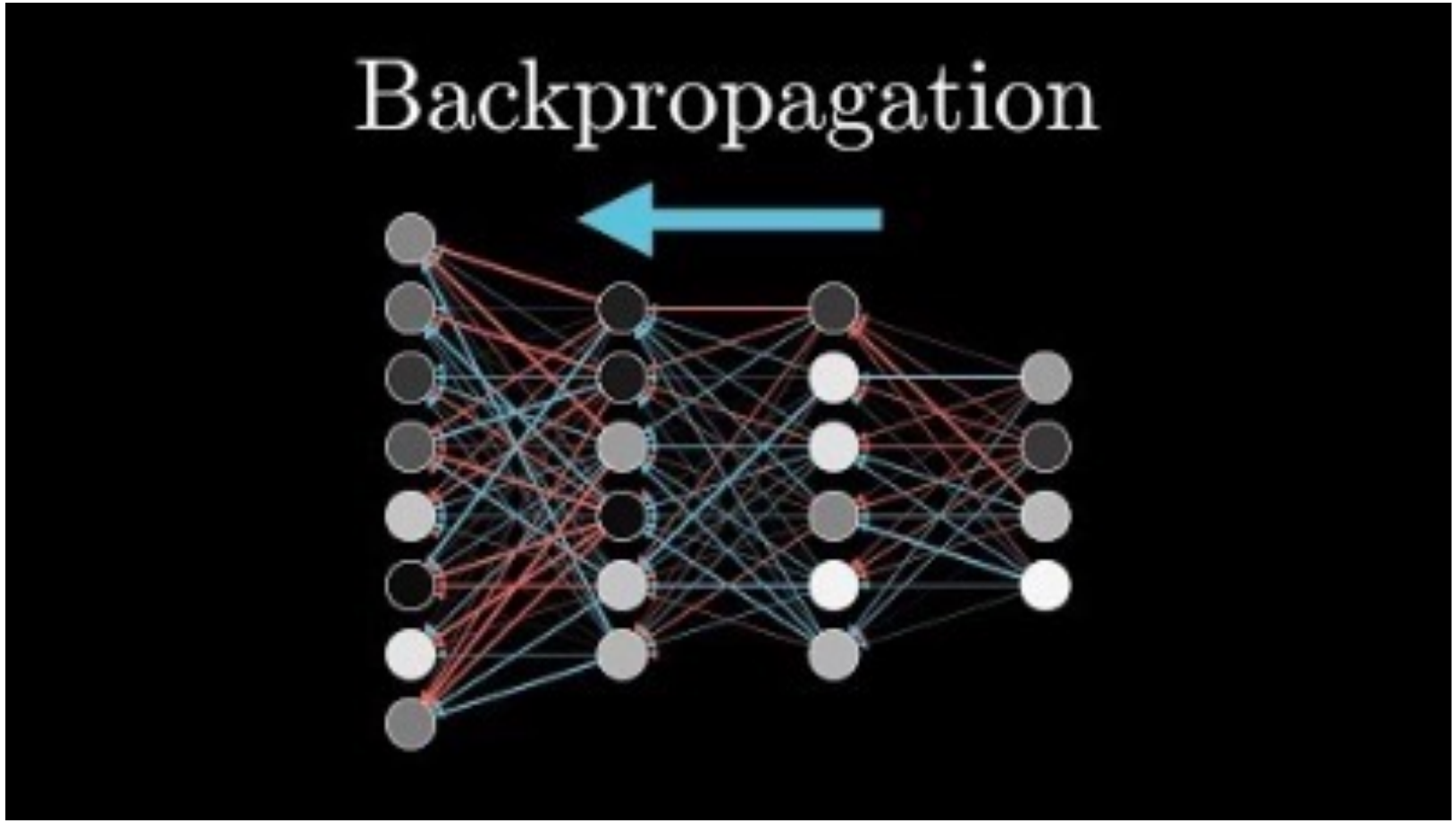
- Now we minimize: $f(A_{lq}, B_{ij}) = \left\| \boldsymbol{z}^k - \boldsymbol{y}^k \right\|^2$
- Where:
- $z'_i = g\left(\sum_{j=1}^{n} B_{ij} x_j\right)$; $z_l = g\left(\sum_{q=1}^{k} A_{lq} z'_q\right)$

- Minimization needs to be done for both sets of weights.
- In practice, we can do them one at a time, moving back from the predicted outputs.
- This process is called *backpropagation* in neural networks, adjusting the weights from right to left in the network.

Backpropagation

- We evaluate the network in a forward direction, moving through the network from input to output.
- Backpropagation takes how the final predicted output compares to the known output and nudges the weights one layer at a time moving backwards to improve the outputs.
- We repeat this for every input/output pair.

Animating in action



Gradient Descent with one Hidden Layer

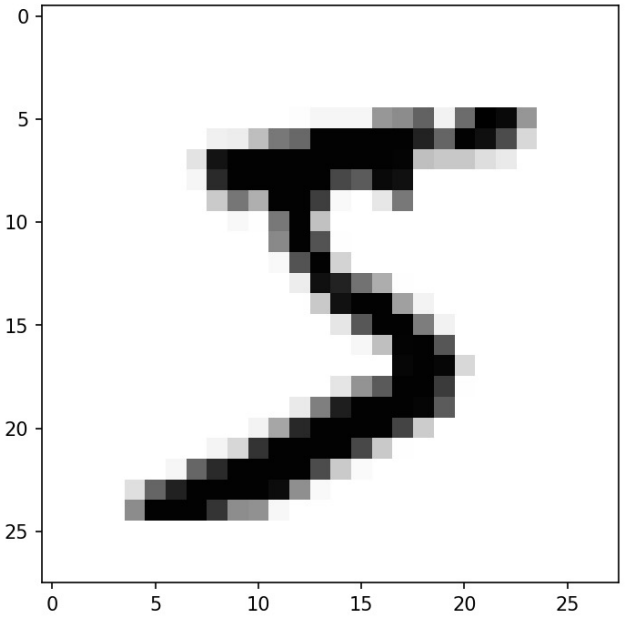
- We do the gradient descent one layer at a time.
- Doing the differentiation, we get something fairly similar to before:
- $\boldsymbol{e} = (\boldsymbol{z} - \boldsymbol{y})$
- $\Delta \boldsymbol{A} = -\eta[2\boldsymbol{e} \circ \boldsymbol{z} \circ (1 - \mathbf{z})] \cdot \boldsymbol{z'}^{T}$
- $\Delta \boldsymbol{B} = -\eta[2\boldsymbol{e'} \circ \boldsymbol{z'} \circ (1 - \mathbf{z'})] \cdot \boldsymbol{x}^{T}$
- $\boldsymbol{e'} = \boldsymbol{A}^{T}\boldsymbol{e} \circ \boldsymbol{z} \circ (1 - \mathbf{z}) \sim \boldsymbol{A}^{T}\boldsymbol{e}$
- Then: **A** ← **A** + Δ**A** & **B** ← **B** + Δ**B**

Hidden Layers

- Often only a single hidden layer is needed.
- Typically larger layers, or more layers will lead to a better network. Though with diminishing returns.
- Check out!
- http://playground.tensorflow.org

Assignment 3 Comprehensive

- The ideas shown here are what you need to do for the comprehensive problem in Assignment 3.
- Use the MNIST dataset of hand-written numbers to train your network to recognize the numbers:
- MNIST seemed down for a bit; has anyone grabbed the data already?
- Data is 28 x 28 pixels = 784 input nodes.
- Output is 10 nodes corresponding to how strongly the network thinks the symbol is 0:9



Facilitation time

Computational Astrophysics

- This course has taken us through three modules:
- General numerical approaches to data analysis and interpretation: computing, coding, parallel programming, interpolation, root solving, integration, ODEs
- Simulation as a tool to evolve PDEs: Boundary Problems, N-Body problems, hydrodynamics, gravity, grid, particle, and hybrid methods.
- Observational data, mock observation, model inference, and machine learning.
- It's been *a lot*. As I said in January, each of these points could be its own course.
- What remains is how you use these topics, and more that you've explored independently, to advance your own numerical work.

Presentations

- Will include peer evaluations.
- Assessed on:
  - Clarity of information (spoken & presented): 4 pts
  - Relevance/consistency to the spirit of the course: 4 pts
    - *** This doesn't mean material outside the course isn't encouraged!
  - Respecting time & professional standards: 2 pts.

- You have 12 minutes for presentations max + 3 minutes Qs.