

PHY604 Lecture 24

November 16, 2021

Review: Importance sampling in DQMC

- Note that P exponentially suppresses propagation into high-potential areas, and potential may vary quickly and significantly
- We can make this more efficient with importance sampling
- Construct a “probability-like” function:

$$F(\mathbf{R}, t) = \Phi(\mathbf{R}, t)\Psi(\mathbf{R})$$

- Where Ψ is a trial wave function, e.g., from variational QMC
- This satisfies the diffusion equation:

$$\frac{\partial F}{\partial t} = \frac{1}{2}\nabla^2 F - \nabla \cdot F \mathbf{U} + [E_c - \mathcal{E}(\mathbf{R})]F$$

- Where we have a “drift” velocity: $U = \nabla \ln \Psi(\mathbf{R})$
- And we see again the local energy: $\mathcal{E}(\mathbf{R}) = \frac{1}{\Psi(\mathbf{R})}H\Psi(\mathbf{R})$

Review: Procedure for diffusion QMC

- 1. Perform a variational Monte Carlo simulation to optimize variational parameters in trial wave function.
- 2. Use the wavefunction from step 1 to generate an initial ensemble of configurations
- 3. Update with drift term and random walk χ : $\mathbf{R}' = \mathbf{R} + \mathbf{U}\tau + \chi$
- 4. Reject any step that crosses a node.
- 5. Accept the move with probability:

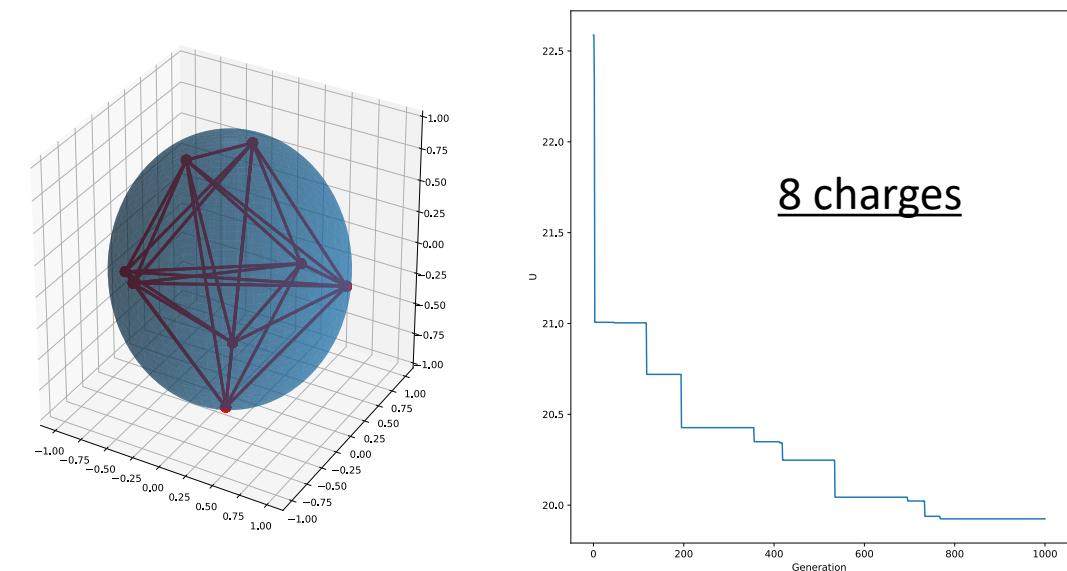
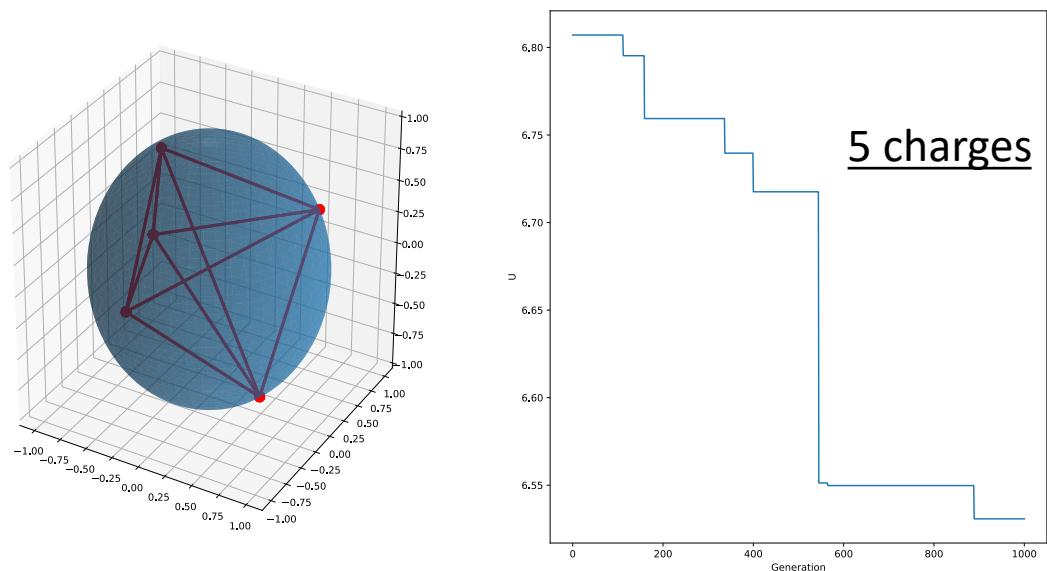
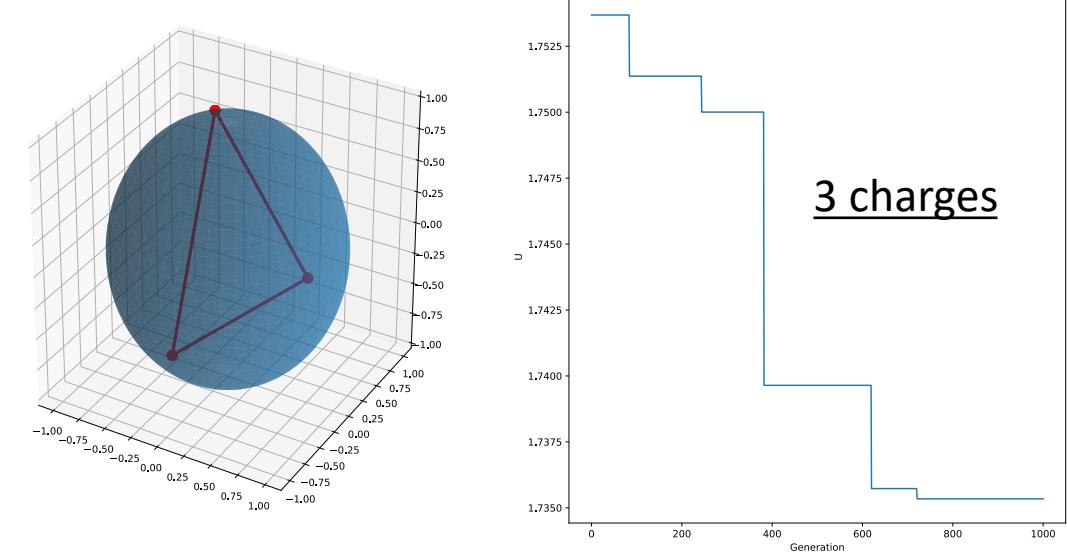
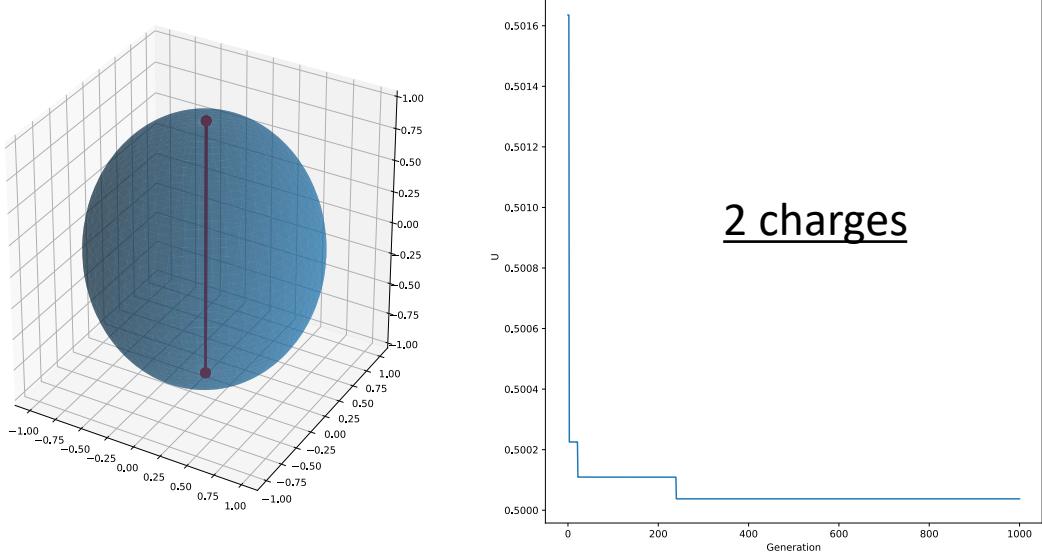
$$w(\mathbf{R}', \mathbf{R}, \tau) = \frac{\Psi(\mathbf{R}')^2 G(\mathbf{R}', \mathbf{R}; \tau)}{\Psi(\mathbf{R})^2 G(\mathbf{R}, \mathbf{R}'; \tau)}$$

- 6. Create a new ensemble of walkers using branching probability P
- 7. Measure local energy
- 8. Update E_c by averaging local energy over configurations \mathbf{R} and \mathbf{R}'

Review: Steps for the genetic algorithm

- The problem: find the global minimum of multi-variable function $g(r_1, r_2, \dots, r_n)$
- 1. Create a gene pool, i.e., an initial population of configurations
 - Configurations are values of variables
 - Can be binary or continuous
- 2. **Selection**: Choose members to be parents
- 3. **Crossover**: Produce offspring by mixing their genes
 - Parent chromosomes are cut into segments, exchanged, and joined together
- 4. **Mutation**: Create random changes to the chromosomes
- 5. In all of the steps above, make sure the configurations with lowest cost (evaluation of g) survive

Review: Solutions to the Thomson problem



Today's lecture:

Neural networks

- Neural network examples
 - Selecting numbers
 - Interpreting a noisy signal
 - Identifying hand-written characters

Machine learning

- Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data (Wikipedia)
- ML is a huge subject and is being applied in a wide range of scientific fields
 - Supervised learning: We know the “output” for some set of input data, want to know the output for the rest
 - Unsupervised: Take a set of inputs and find some structure
 - ...
- We will focus our discussion: Supervised learning with neural networks

Patter recognition with computers

- Classic problem: Identify pictures of dogs versus cats
 - Easy for human, difficult for computer



Neural networks

- Neural networks attempt to mimic the action of neurons in a brain
- Good for problems where we have an incomplete or unsophisticated physical model, but a lot of data
 - Create a nonlinear fitting routine with free parameters
 - Train the network on data with known input and output to set the parameters
 - Trained network can be used on new inputs to predict outcome
- Help with pattern recognition, which is difficult for computers (often easy for humans)
 - Classic problem, identifying pictures of cats versus dogs
- Some uses:
 - Character / image recognition
 - AI for games
 - Classification of data
 - Finance

A simple linear model

- Represent input data as a vector x
- Represent output data as a vector z
- Simplest “model” that relates x and z is an unknown matrix \mathbf{A} :

$$z = \mathbf{A}x$$

- This is just the same linear problem we have solved many times, but usually for x with a known \mathbf{A}
- How can we get the values for \mathbf{A} ? If we have enough input/output data, we can figure it out

Solving for our linear model

- Say we have following data of input-output pairs:

$$x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad z_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad z_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$x_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad z_3 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

- We want to find \mathbf{A} such that:

$$z_1 = \mathbf{A}x_1, \quad z_2 = \mathbf{A}x_2, \quad z_3 = \mathbf{A}x_3,$$

Solving our linear model

- We write: $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
- Take the first two pairs: $\mathbf{A}x_1 = \begin{pmatrix} a \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
 $\mathbf{A}x_2 = \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- Now \mathbf{A} is fully specified: $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
- But we can't fulfill the last condition:
 $\mathbf{A}x_3 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \neq \begin{pmatrix} 4 \\ 1 \end{pmatrix} = z_3$

Nonlinear models

- We saw with the previous example:
 - We can “train” a model using known inputs and outputs
 - A linear model is too “definite,” which is too restrictive
- Let’s run our linear model through a nonlinear function $g(x)$:

$$g(x) = \begin{pmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{pmatrix}$$

- To get: $z = g(\mathbf{A}x)$

Nonlinear models

- Consider the simple nonlinear function: $g(p) = p^2$

- Solving the nonlinear equation with our inputs:

$$z_1 = g(\mathbf{A}x_1), \quad z_2 = g(\mathbf{A}x_2), \quad z_3 = g(\mathbf{A}x_3),$$

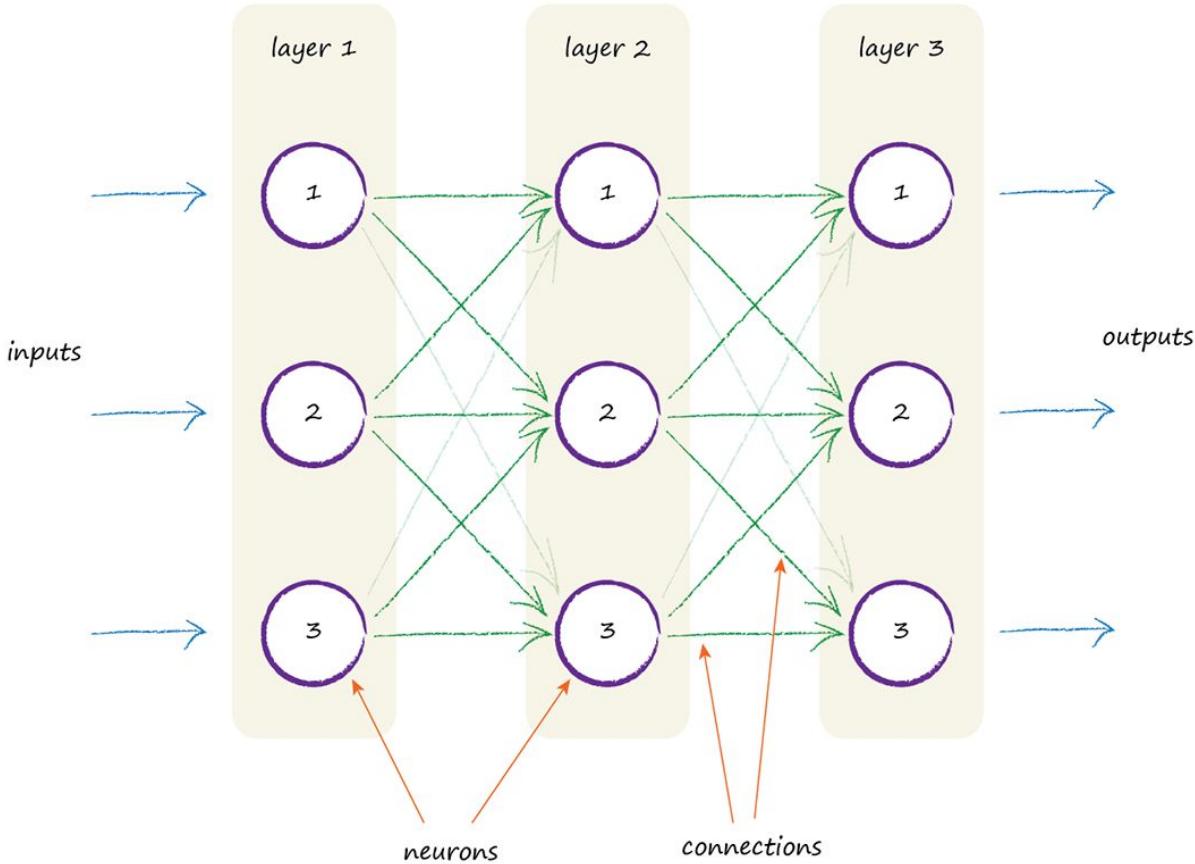
- Gives four valid solutions:

$$\mathbf{A}_1 = \begin{pmatrix} -1 & -1 \\ 0 & -1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} -1 & -1 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{A}_3 = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \quad \mathbf{A}_4 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

- Nonlinear models give much greater flexibility for describing data
- Tradeoff is that they are harder to solve

Nonlinear functions at the basis of neural networks

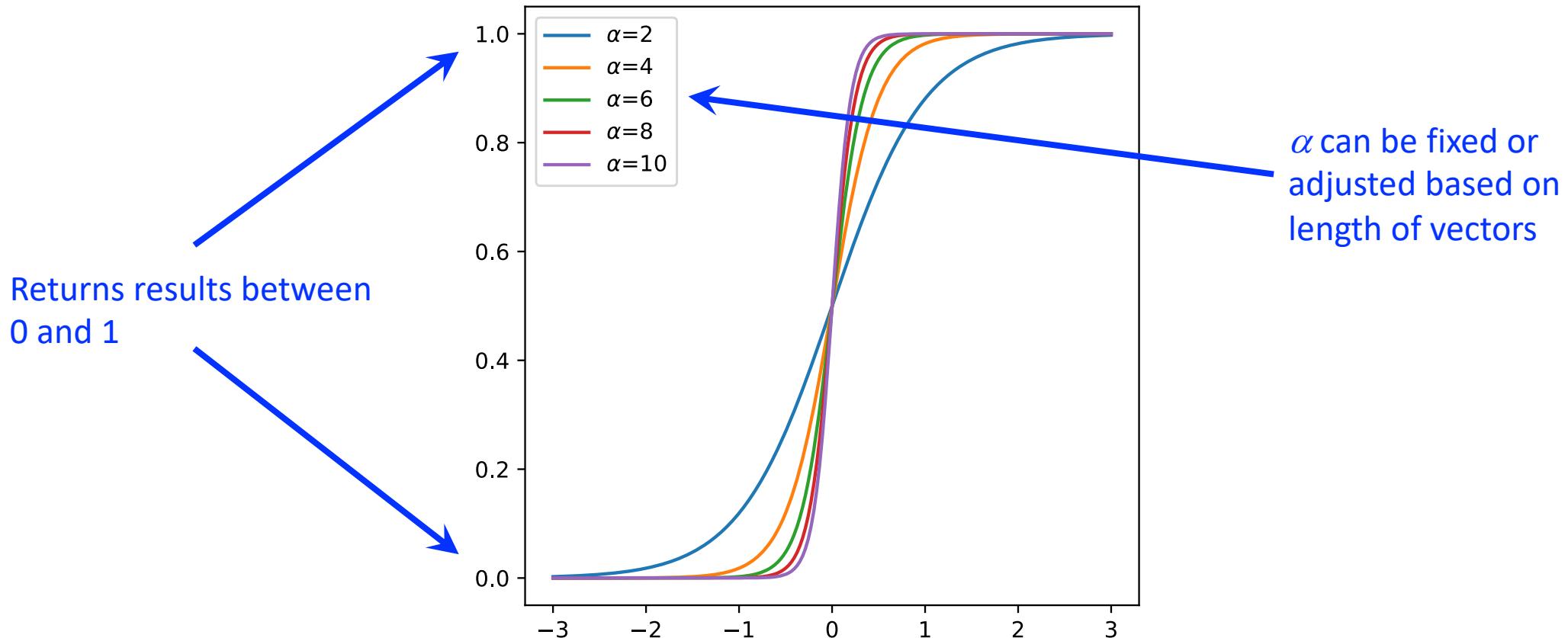
- Neural networks are divided into *layers*
 - Input layer accepts the input
 - Output layer outputs results
- Each layer has neurons (or nodes)
 - For input, one node for each input variable
 - Every node in the first layer connects to every node in the next layer
- Weight associated with the connection can be adjusted
 - These are the matrix elements
- Operations at neurons given by nonlinear activation function



Make Your Own Neural Network, Tariq Rashid

The sigmoid function for the nonlinear model

- What do we want from the nonlinear function?
 - For simplicity we will require that outputs are in the range (0,1)
 - We will need a function that is continuous and differentiable



Neural network

- If we write the matrix-vector multiplication as:

$$(\mathbf{A}x)_i = \sum_{j=1}^n A_{ij}x_j$$

- Then the action of our neural network is:

$$z_i = g[(\mathbf{A}x)_i] = g \left[\sum_{j=1}^n A_{ij}x_j \right]$$

- Would like the elements of $\mathbf{A}x$ to run over the nonlinear range of the sigmoid function. Choose for α :

$$\alpha = \frac{10}{n \max|x_i|}$$

Training our neural network

- Now we need to find the coefficients A_{ij}
- Assume we have some “training data” inputs x and outputs z
- Start with random entries in \mathbf{A} in the range [-1,1]
- Minimize the difference between $g(\mathbf{A}x_j)$ and z_j
 - Function to be minimized:
$$f(A_{ij}) = |g(\mathbf{A}x_j) - z_j|^2$$
- We will minimize this function with the steepest descent method (see Lecture 11), iteratively update entries in \mathbf{A} according to:

$$A_{ij} = A_{ij} - \eta \frac{\partial f}{\partial A_{ij}}$$

Gradient of minimization function

- Writing out the function explicitly:

$$f(A_{ij}) = \sum_{i=1}^m \left[g \left(\sum_{j=1}^n A_{ij} x_j \right) - z_i \right]^2$$

- Define:
- $$b_i \equiv \sum_{j=1}^n A_{ij} x_j, \quad z_i \equiv g(b_i)$$

- Then:
- $$f(A_{ij}) = \sum_{i=1}^m (z_i - y_i)^2$$

- And:
- $$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^m 2(z_i - y_i) \frac{\partial z_i}{\partial A_{pq}}$$

Gradient of minimization function

$$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^m 2(z_i - y_i) \frac{\partial z_i}{\partial A_{pq}}$$

- Where:

$$\frac{\partial z_i}{\partial A_{pq}} = g'(b_i) \frac{\partial b_i}{\partial A_{pq}}$$

- And:

$$\frac{\partial b_i}{\partial A_{pq}} = \sum_{j=1}^n \frac{\partial A_{ij}}{\partial A_{pq}} x_j = \sum_{j=1}^n \delta_{ip} \delta_{jq} x_j = \delta_{ip} x_q$$

Gradient of minimization function

- Because of our form of g , we have:

$$g'(p) = \frac{\alpha e^{-\alpha p}}{(1 + e^{-\alpha p})^2} = \alpha g(p)[1 - g(p)]$$

- So: $\frac{\partial z_i}{\partial A_{pq}} = \alpha g(b_i)[1 - g(b_i)]\delta_{ip}x_q = \alpha z_i(1 - z_i)\delta_{ip}x_q$

- And:

$$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^m 2(z_i - y_i)\alpha z_i(1 - z_i)\delta_{ip}x_q = 2\alpha(z_p - y_p)z_p(1 - z_p)x_q$$

Comments on using the neural network

- Once we have trained \mathbf{A} , then we can use our neural net on some input w for which we don't know the output by calculating $g(\mathbf{A}w)$
- Have to set a value of α prior to training (using the max of all of the input data)
- Note that once the matrix \mathbf{A} has been adapted for a given input/output pair, it will generally not work anymore for the previous pairs. To get around this:
 - Generate t sets of input/output training data
 - Repeat the sets Nt times, and run them through at random

Procedure for doing “Machine Learning” with neural network

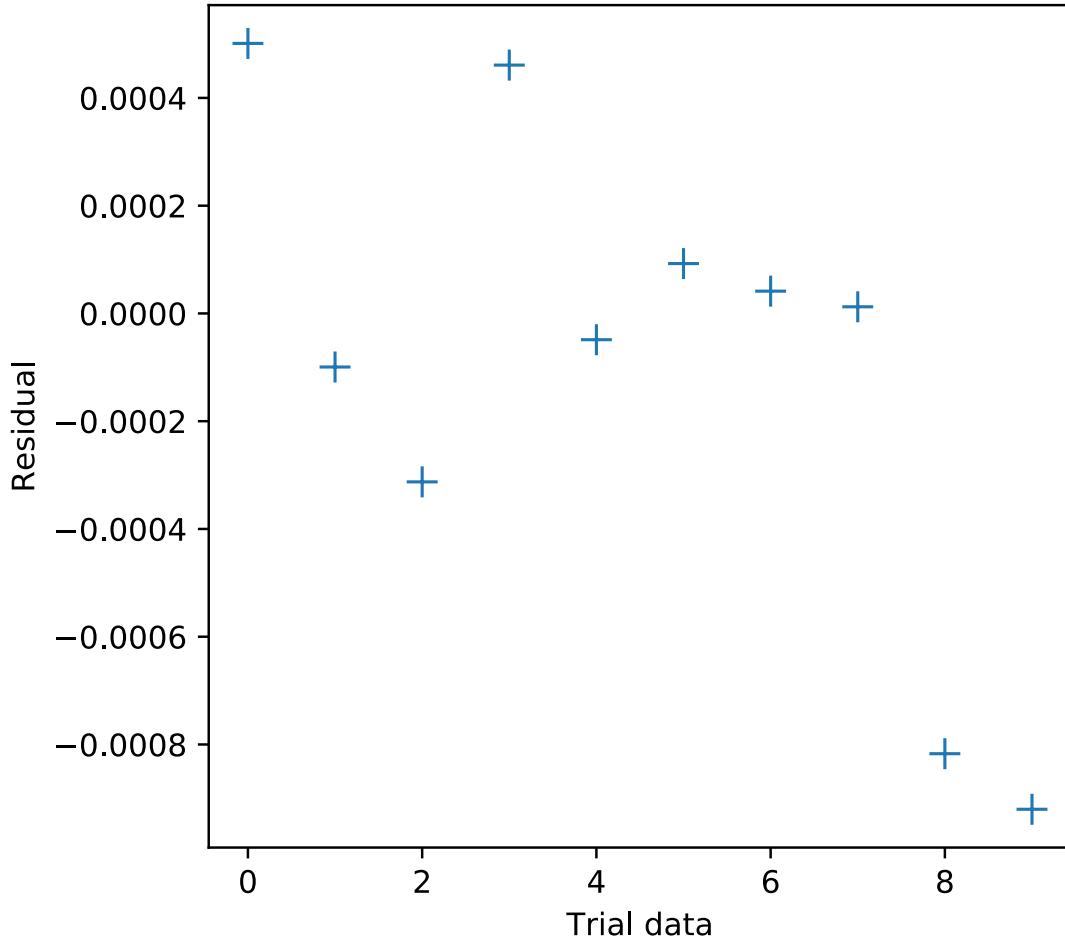
- 1. Choose a nonlinear activation function (in our case, find α)
- 2. Choose/generate t input/output pairs for training
- 3. Repeat the set from step 2 N times (epochs) to get a training set of $T=Nt$ pairs
- 4. Run the training set through the neural net at random, performing the steepest descent minimization for each
- 5. To test the training in step 4, run the t examples through and calculate the residual:
$$g(\mathbf{A}x_j) - z_j$$
- 6. Use the neural net on some new data

Simple example of a neural net

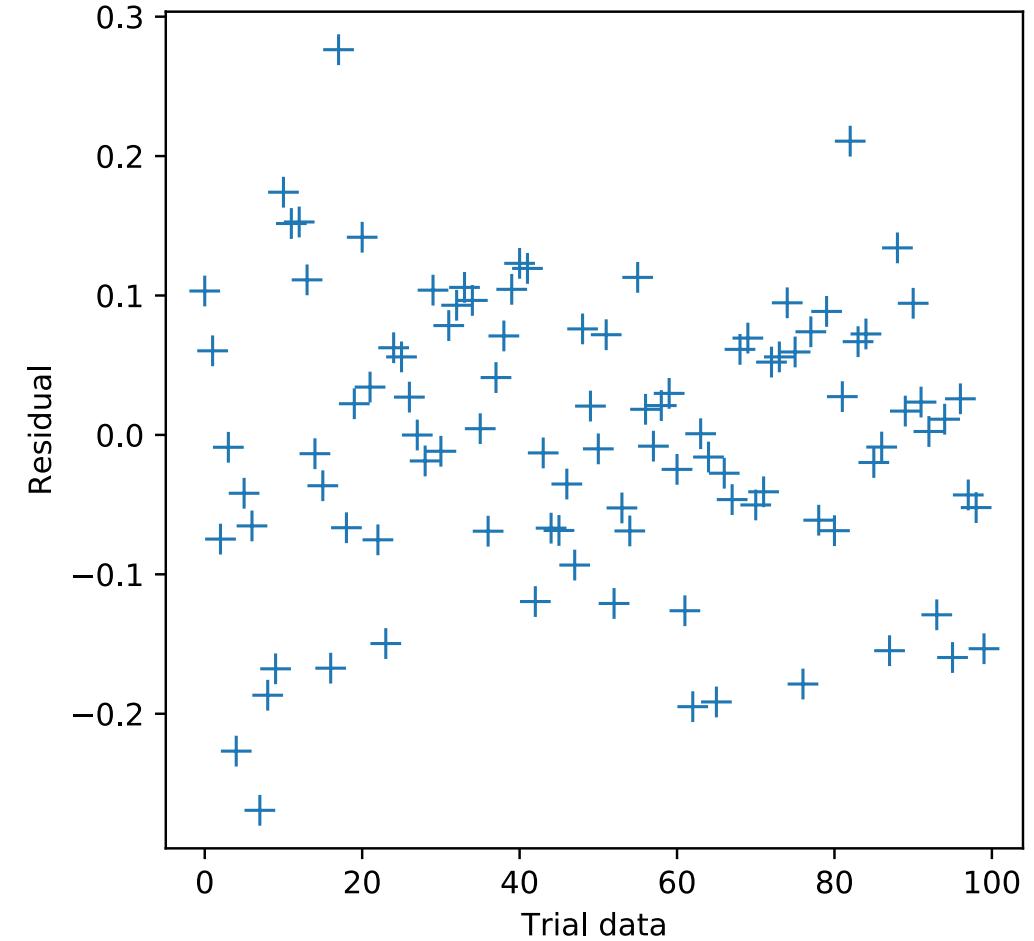
- Input data: Ten randomly chosen numbers from a set
- Output data: The tenth number in the set
- Make a training set of 10 input/output pairs
- Run it through randomly 100 times to train A

Results from our neural net

Applied to training data



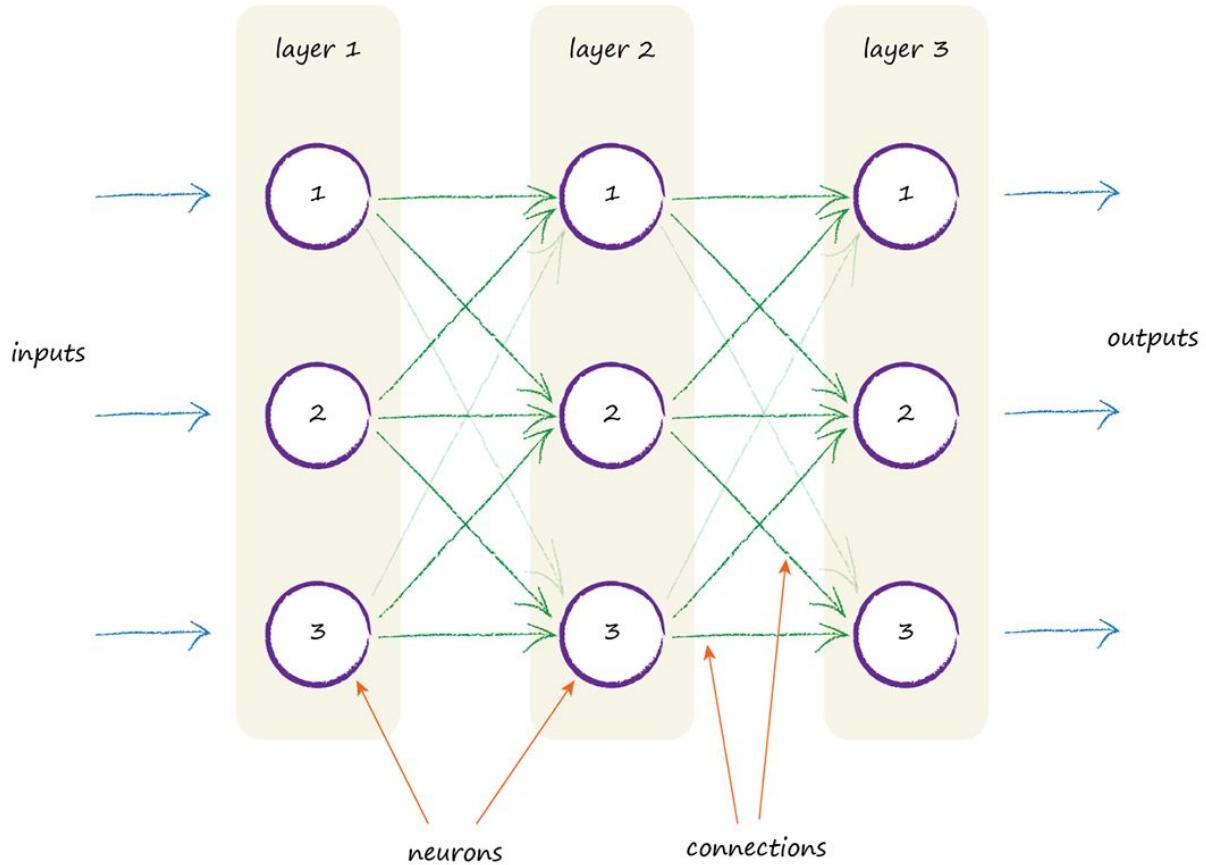
Applied to new data



$$\mathbf{A} = [-0.04 \quad 0.42 \quad 0.15 \quad -0.23 \quad 0.13 \quad 0.06 \quad 0.19 \quad -0.42 \quad 0.48 \quad 4.45]$$

Adding additional degrees of freedom

- In the previous example, the number of adjustable parameters is constrained by the size of the input and output
- To overcome this limitation, we can add **hidden layers** to our neural net
- Will need an additional matrix and an additional evaluation of our nonlinear function



Make Your Own Neural Network, Tariq Rashid

Hidden layers

- Take as input a vector x of length n
- Take as input a vector z of length m
- Consider a $k \times n$ matrix \mathbf{B} and a $m \times k$ matrix \mathbf{A}
- Construct the output as:

$$\tilde{z} = g(\mathbf{B}x) - \frac{1}{2}, \quad z = \tilde{g}(\mathbf{A}\tilde{z})$$

- Note that we differentiate the applications of g because they may have different α 's
- Extra shift of $\frac{1}{2}$ is to recenter the data around 0 to put it in the nonlinear range of g
- Key: k is independent of the size of input/output!
 - Can train $k(m+n)$ total elements

Implementing the hidden layer

- We still want to minimize our cost function f :

$$f(A_{rs}, B_{ij}) = \sum_{r=1}^m (z_r - y_r)^2$$

- Now we have to do two interrelated steepest descent minimizations:

$$A_{pq} = A_{pq} - \eta \frac{\partial f}{\partial A_{pq}}, \quad B_{pq} = B_{pq} - \eta \frac{\partial f}{\partial B_{pq}}$$

- Where:

$$\frac{\partial f}{\partial A_{pq}} = 2\tilde{\alpha}(z_p - y_p)z_p(1 - z_p)\tilde{z}_q \equiv \sigma_p \tilde{z}_q$$

$$\frac{\partial f}{\partial B_{pq}} = \sum_{r=1}^m \sigma_r A_{rp} \alpha \left(\frac{1}{2} + \tilde{z}_p \right) \left(\frac{1}{2} - \tilde{z}_p \right) x_q$$

Back propagation

- Note that we are optimizing simultaneously **A** and **B**:

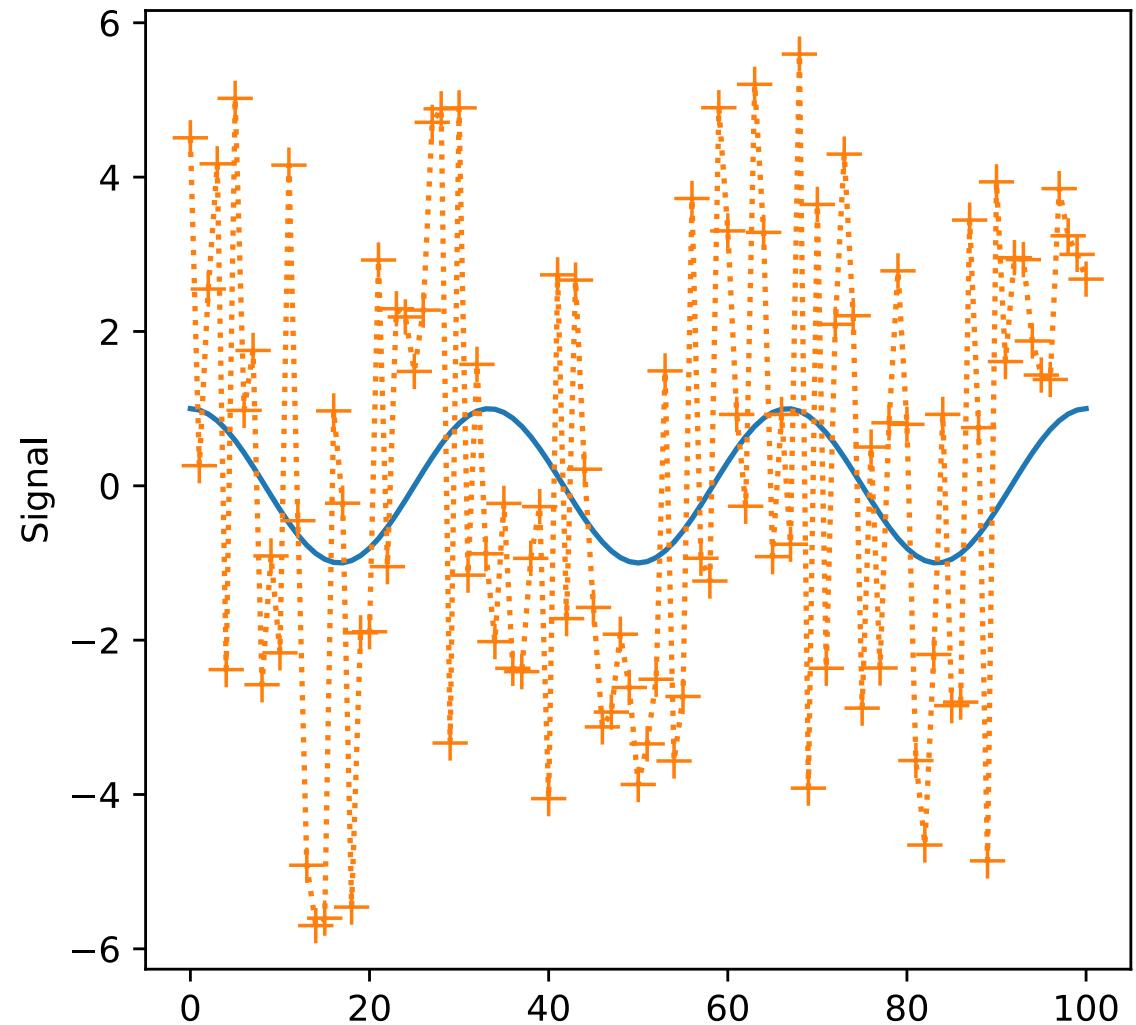
$$\frac{\partial f}{\partial A_{pq}} = 2\tilde{\alpha}(z_p - y_p)z_p(1 - z_p)\tilde{z}_q \equiv \sigma_p \tilde{z}_q$$

$$\frac{\partial f}{\partial B_{pq}} = \sum_{r=1}^m \sigma_r A_{rp} \alpha \left(\frac{1}{2} + \tilde{z}_p \right) \left(\frac{1}{2} - \tilde{z}_p \right) x_q$$

- So, the errors are “backpropagated” through the output and hidden layers

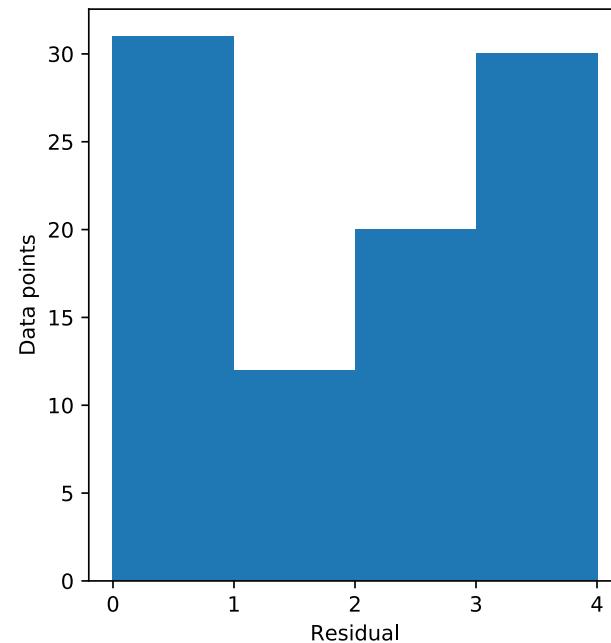
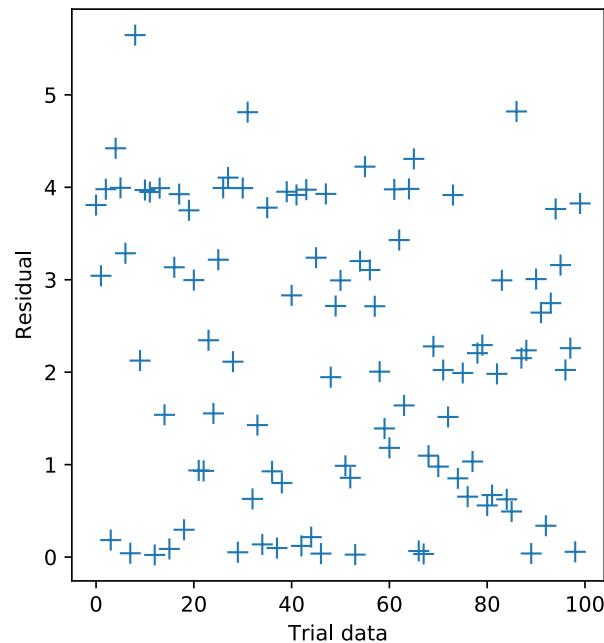
Example: Signal analysis

- Given a noisy signal expected to be one of four frequencies
 - $f = \{1, 2, 3, 4\}$ Hz
- Noise is significantly larger than the underlying signal:
$$s(t) = \cos(2\pi ft) + 5\xi$$
 - ξ is a random number in [-1,1]
- Can we identify the frequency?

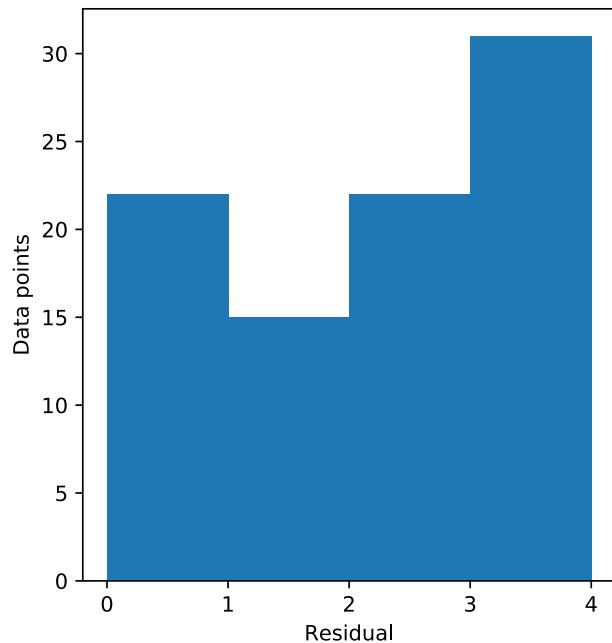
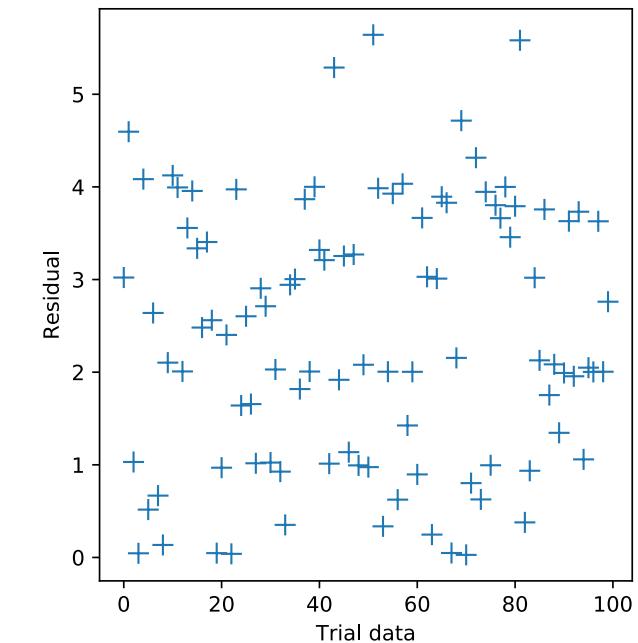


Signal analysis: Hidden layers size 2

Test on the training set

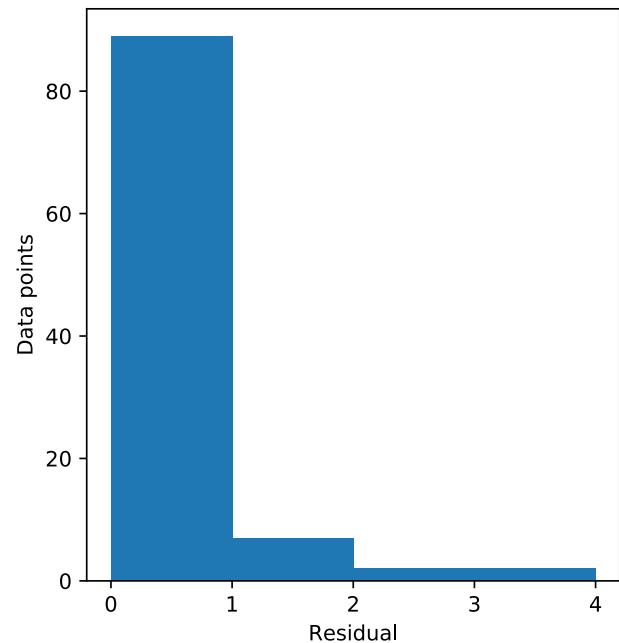
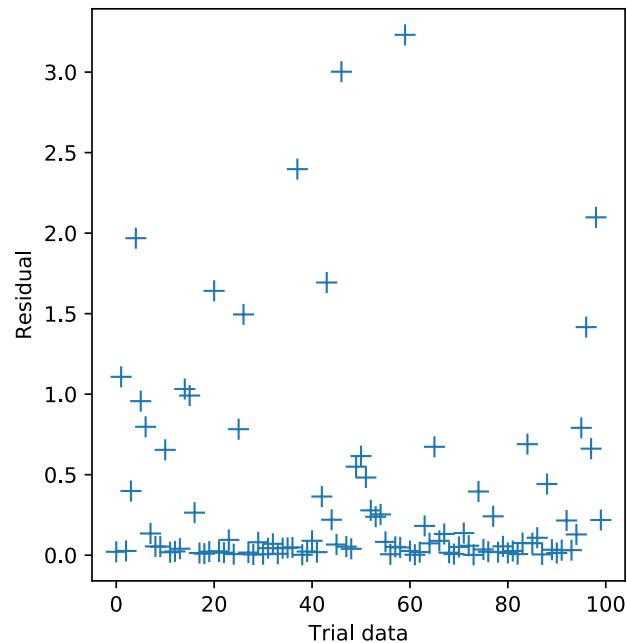


Test on new data

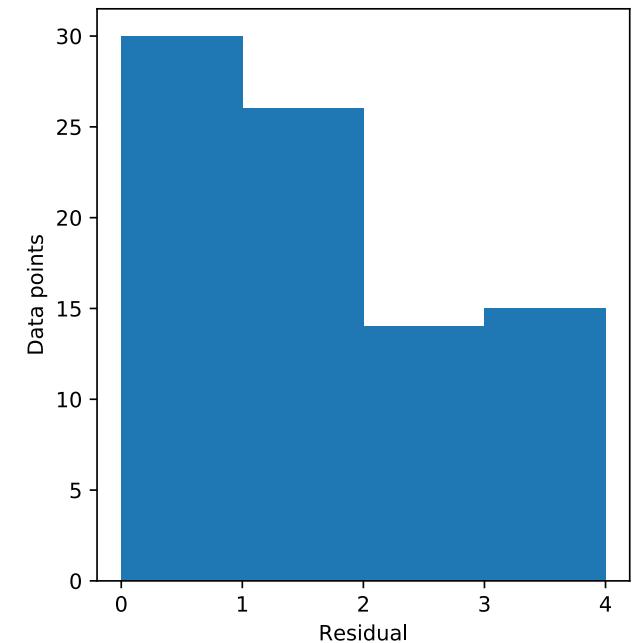
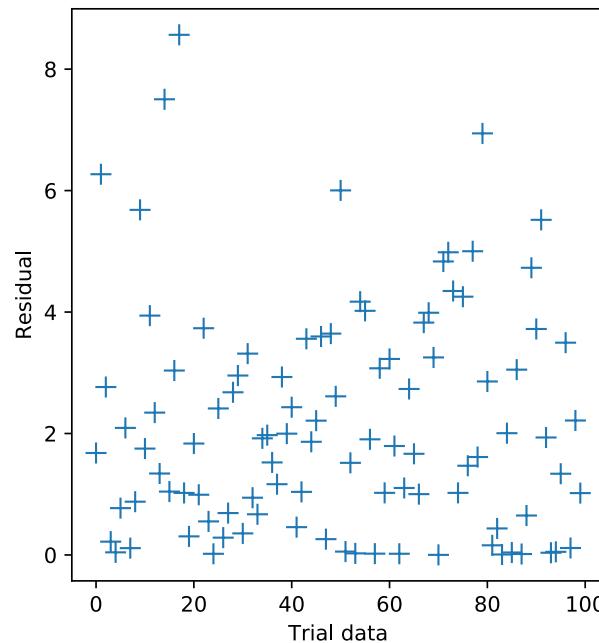


Signal analysis: Hidden layers size 3

Test on the training set

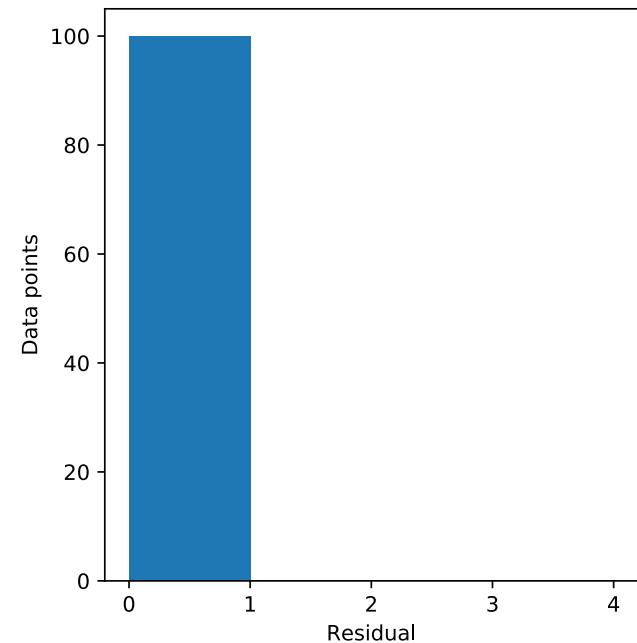
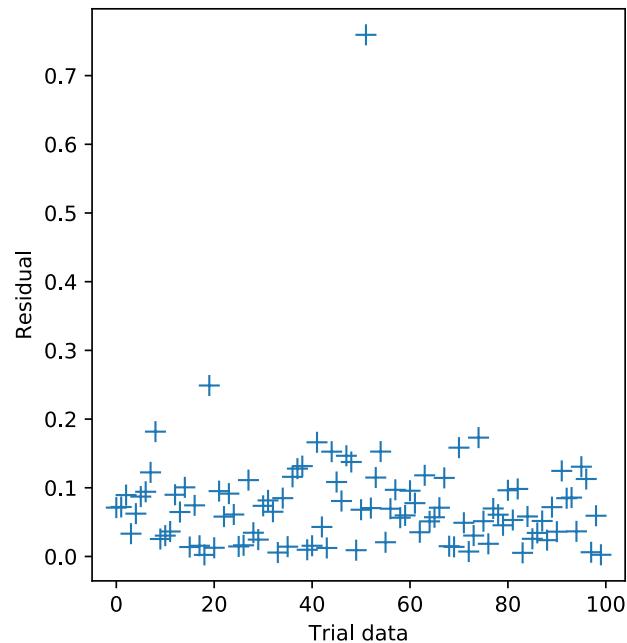


Test on new data

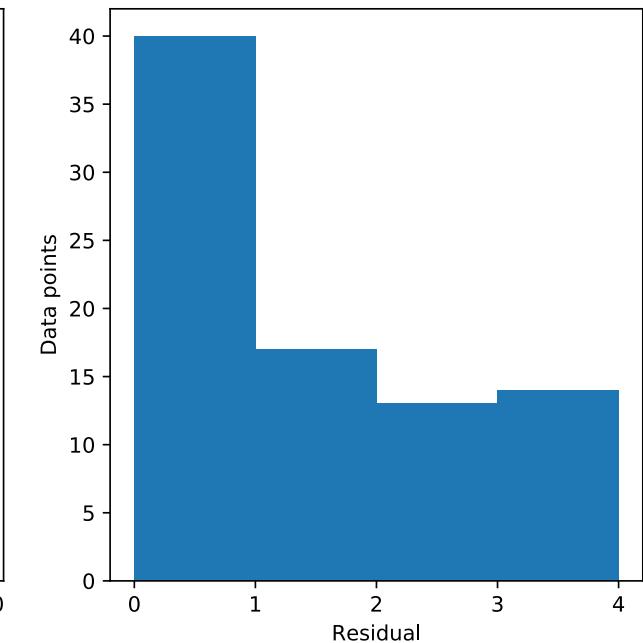
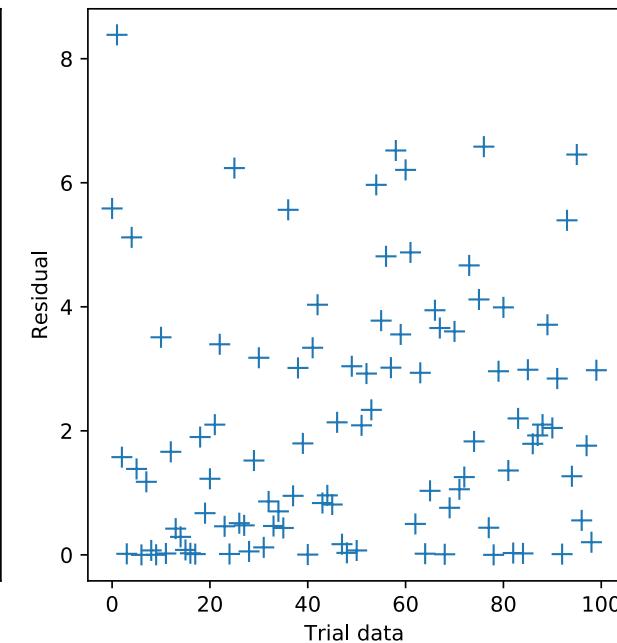


Signal analysis: Hidden layers size 4

Test on the training set

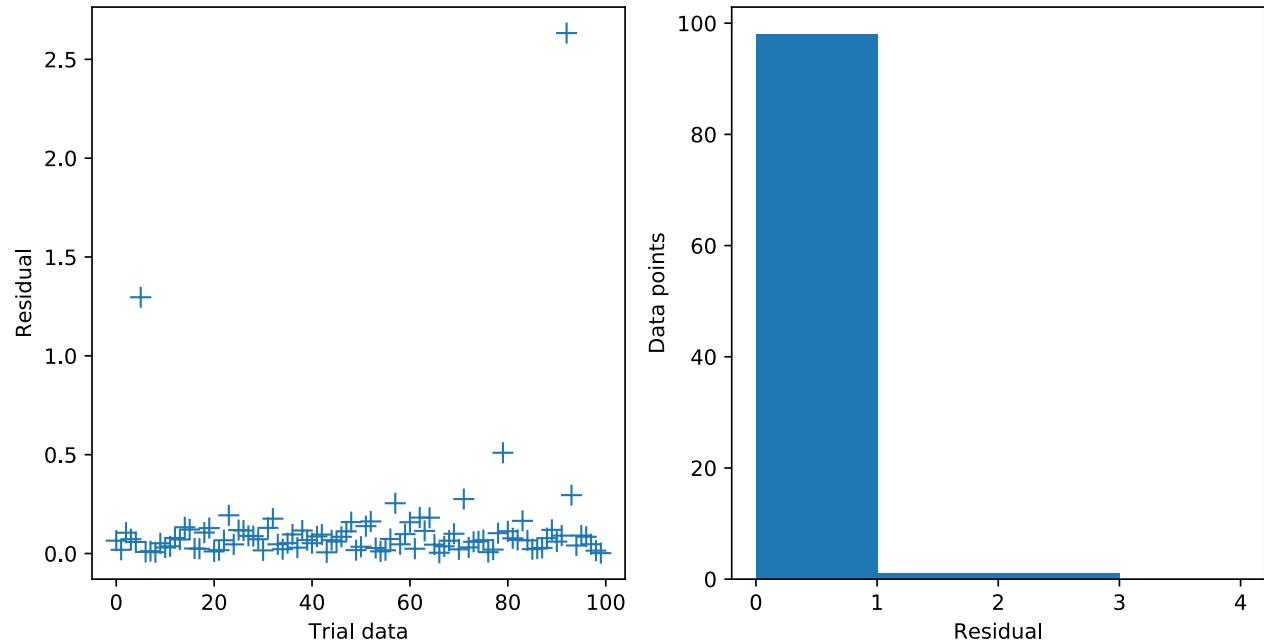


Test on new data

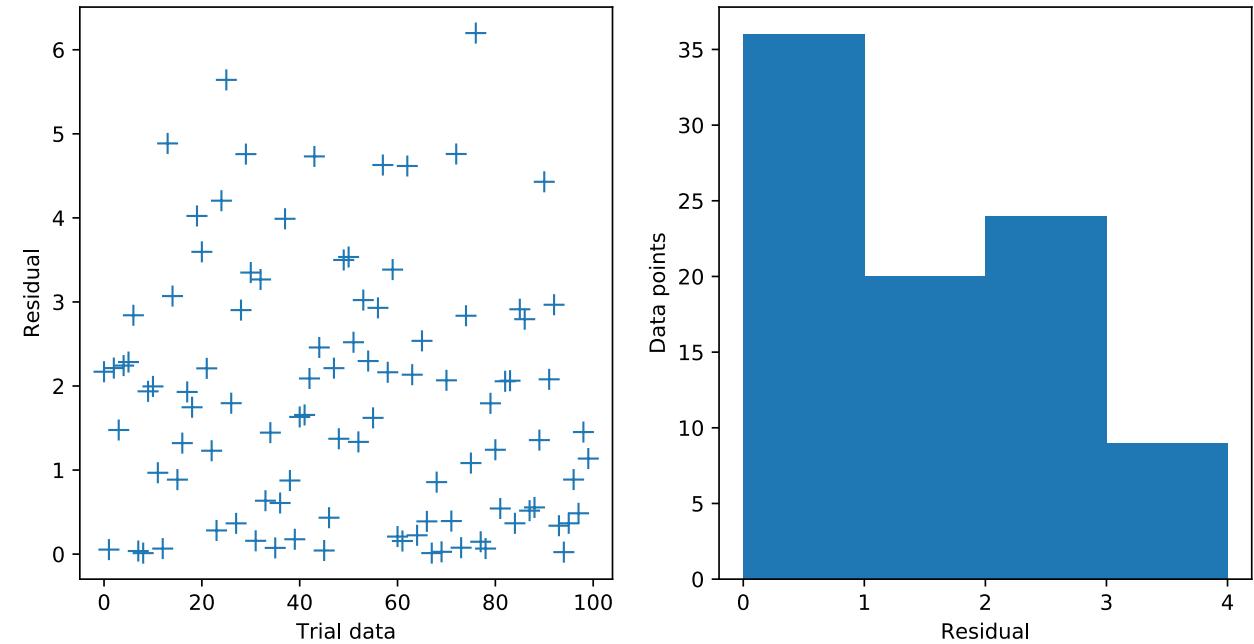


Signal analysis: Hidden layers size 8

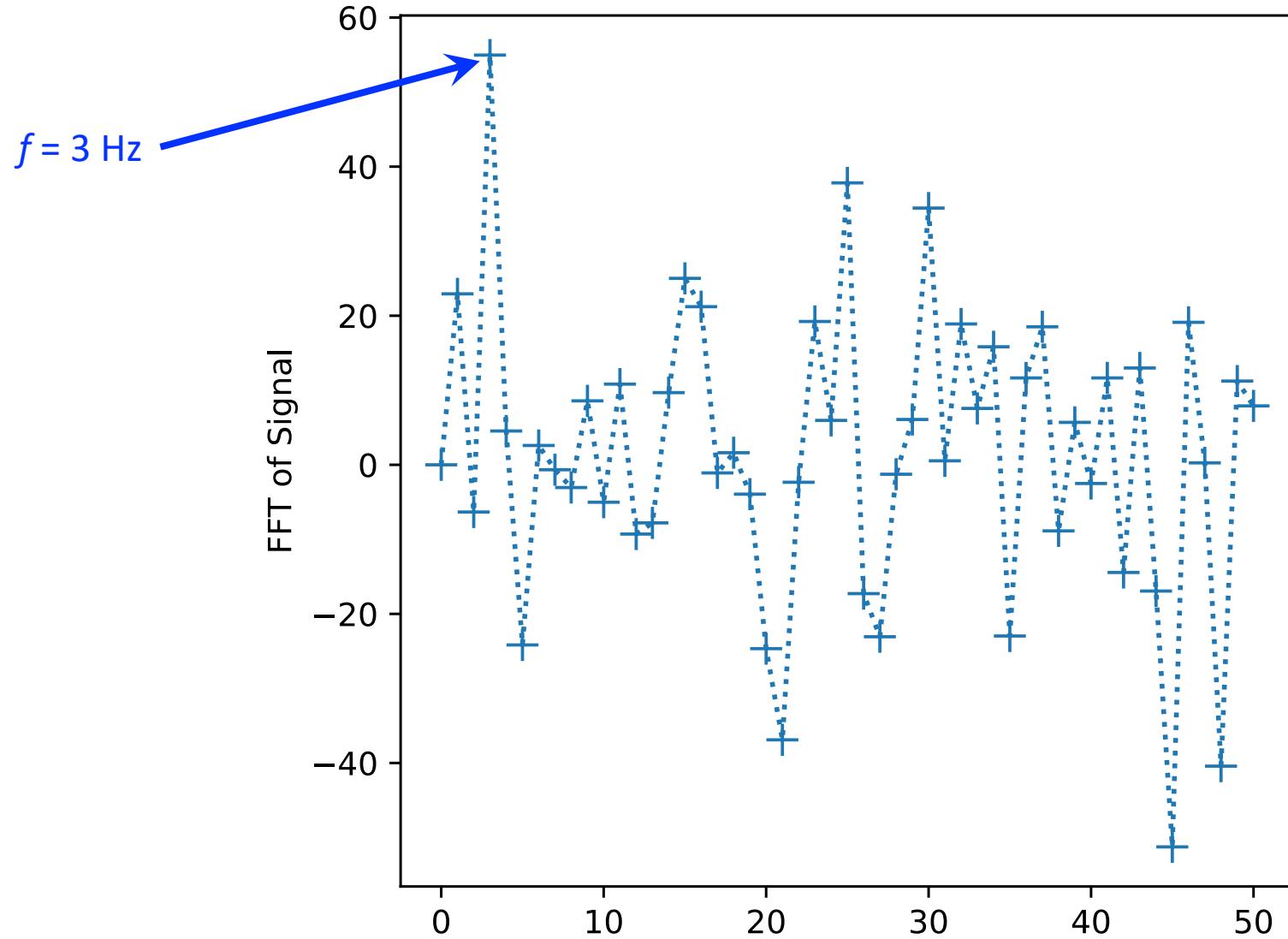
Test on the training set



Test on new data



Can we do the same with an FFT

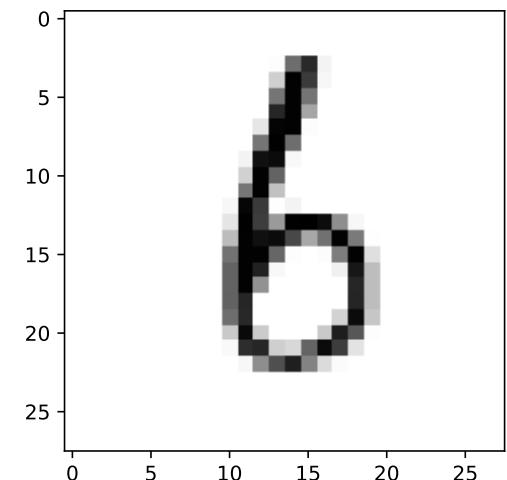
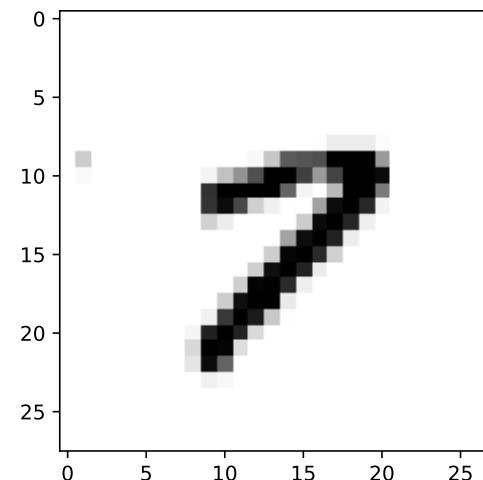
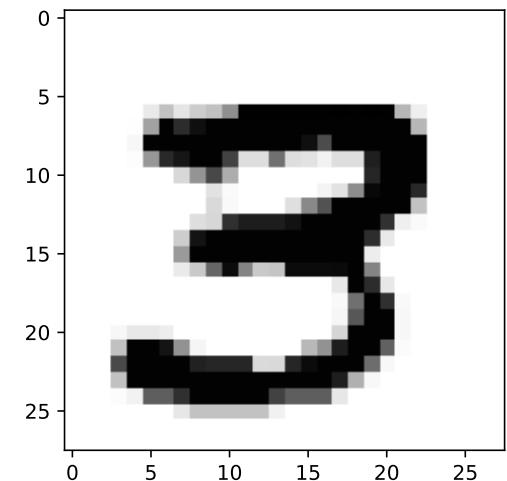
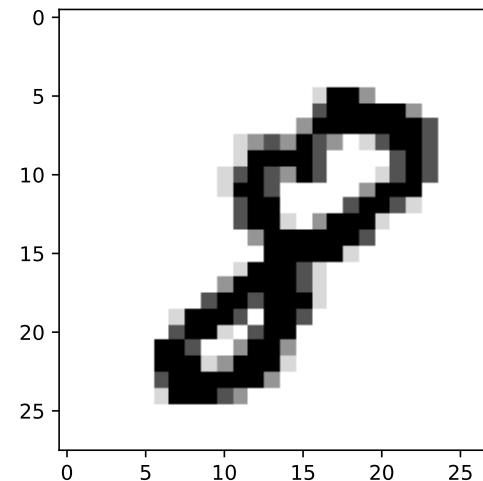


Another example: Recognizing written characters

- We'll try to recognize a digit (0 – 9) from an image of a handwritten digit.
- MNIST dataset (<http://yann.lecun.com/exdb/mnist/>)
 - Popular dataset for testing out machine learning techniques
 - Training set is 60,000 images
 - Approximately 250 different writers
 - Test set is 10,000 images
 - Correct answer is known for both sets so we can test our performance
- Image details:
 - 28 × 28 pixels, grayscale (0 – 255 intensity)
- We'll use a small subset

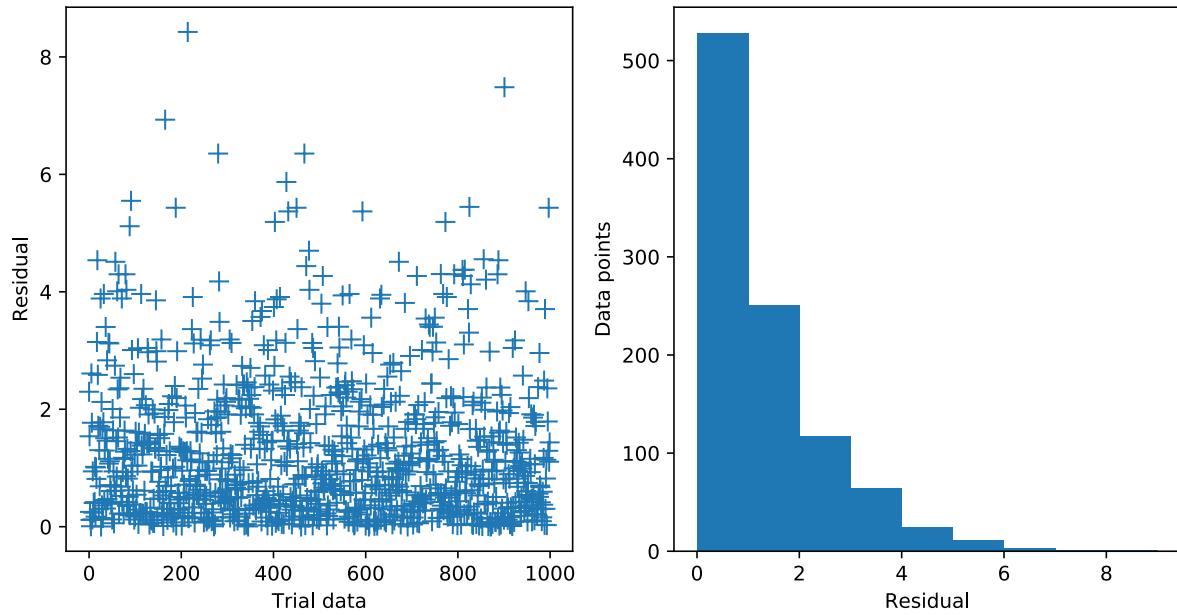
Another example: Recognizing written characters

- Input layer: 784 nodes (number of pixels)
- Output layer will be 10 nodes
 - Array with an entry for each possible digit
- Hidden layer size of 100
- 10 epochs
- We'll train on the training set, using 1000 images
- Rescale the input to be in $[0.01, 1]$
- We'll test on the test set of 1000 images

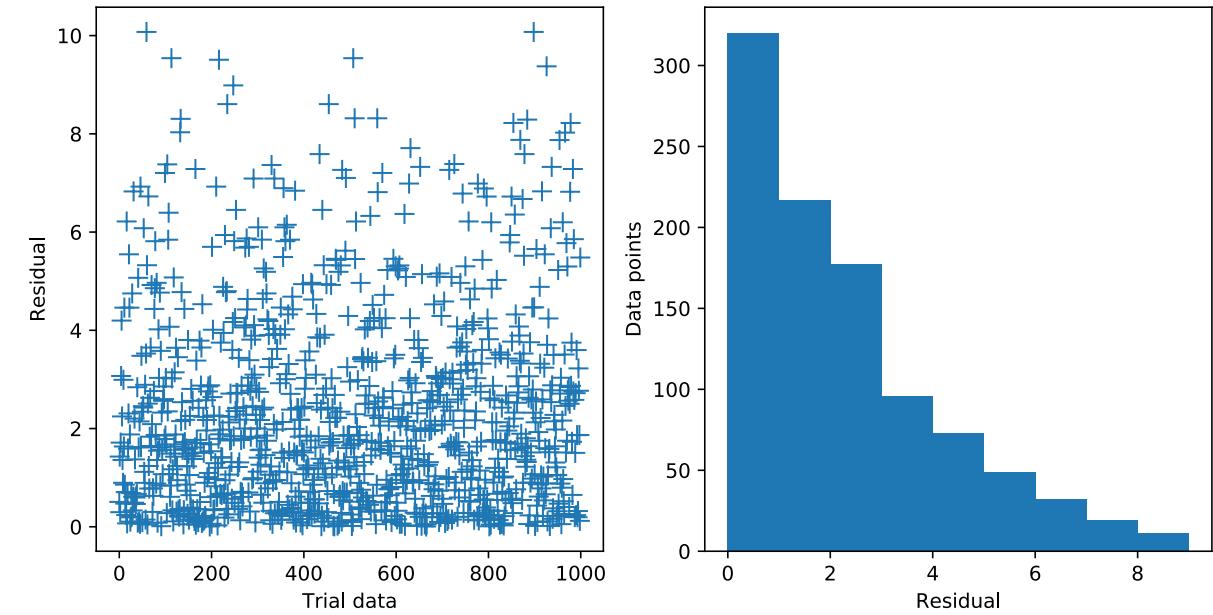


Another example: Recognizing written characters

Test on the training set



Test on new data



After class tasks

- Homework 5 due today (sorry for typos!)
- Homework 4 graded
- First draft of first two sections of writeup due Nov. 18
- Readings:
 - *Computational Methods for Physics*, Joel Franklin, Chapter 14
 - *Make Your Own Neural Network*, Tariq Rashid
 - <http://playground.tensorflow.org>