PHY604 Lecture 25

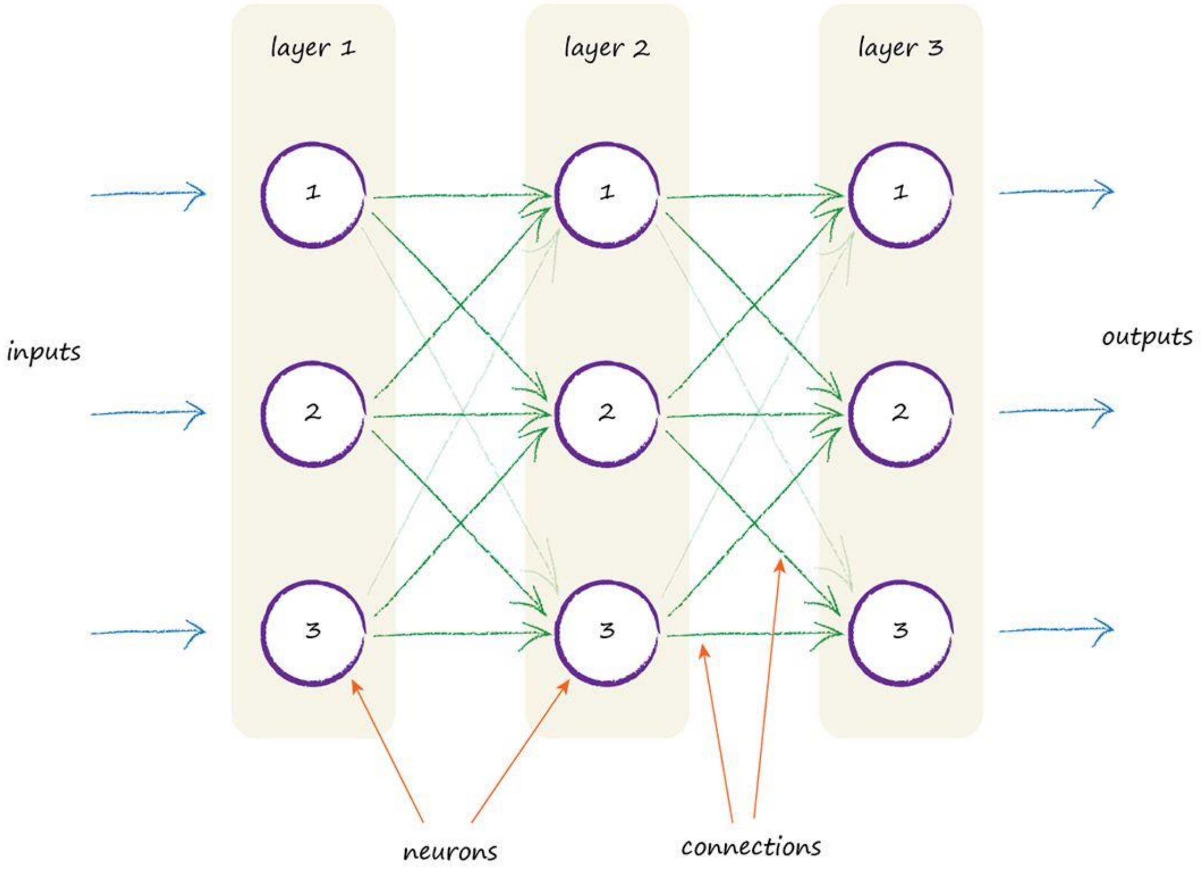
November 20, 2025

Today’s lecture: Neural networks

- Neural networks

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

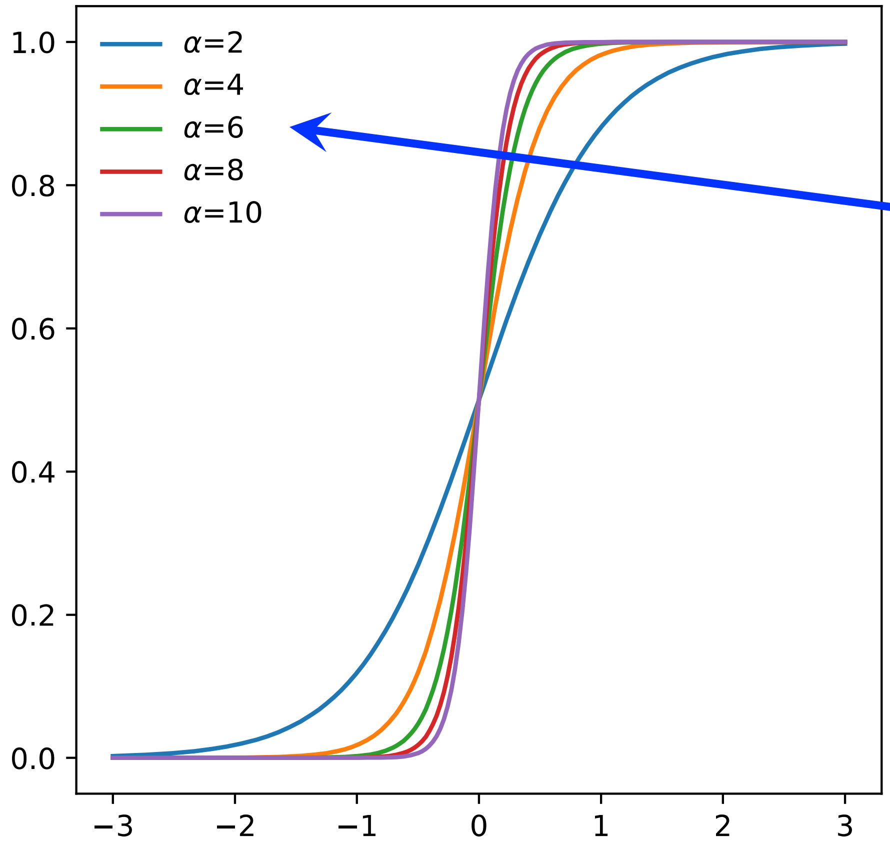


Make Your Own Neural Network, Tariq Rashid

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

Returns results between 0 and 1



α can be fixed or adjusted based on length of vectors

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 **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 *y*

- Start with random entries in **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 12), iteratively update entries in **A** according to:

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

Gradient of minimization function

- Writing out the function explicitly:

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

- 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 **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 **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

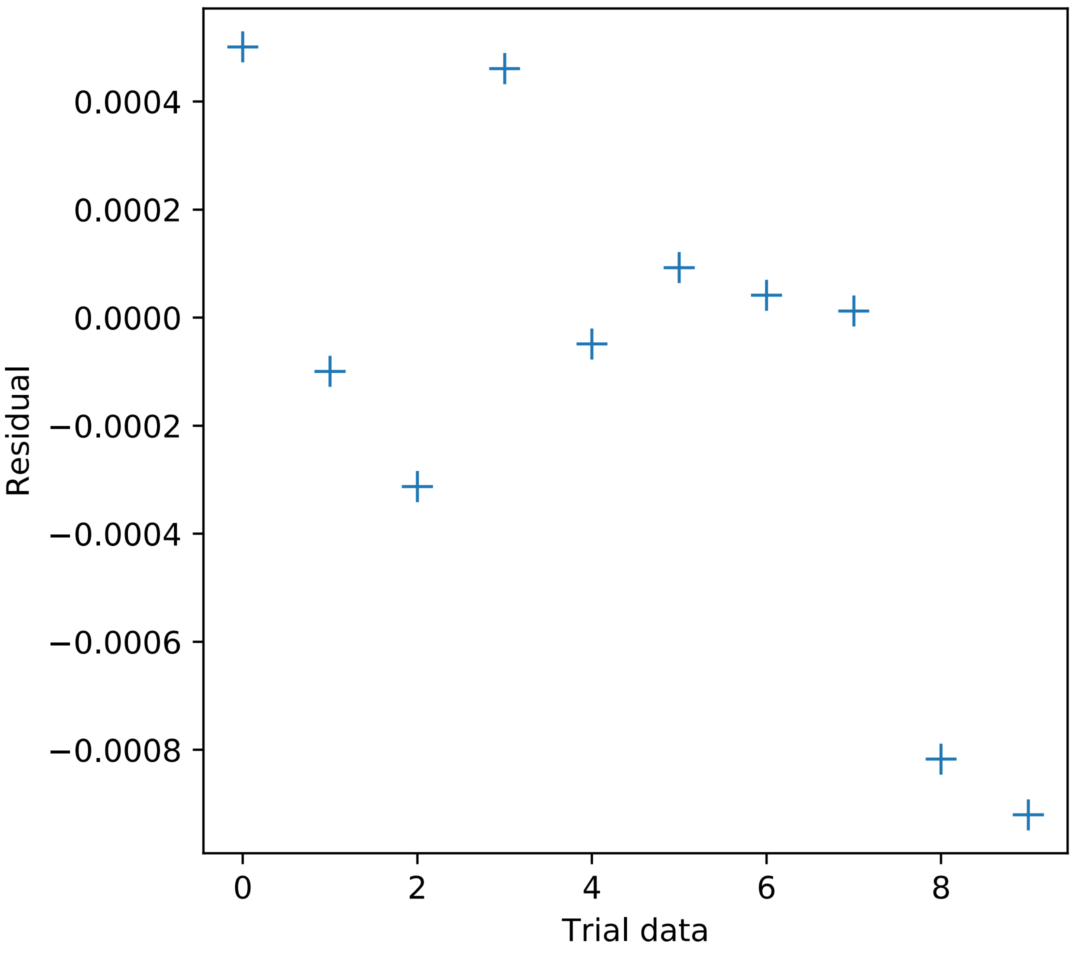
Simple example of a neural net

- 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**
- Expect something like:

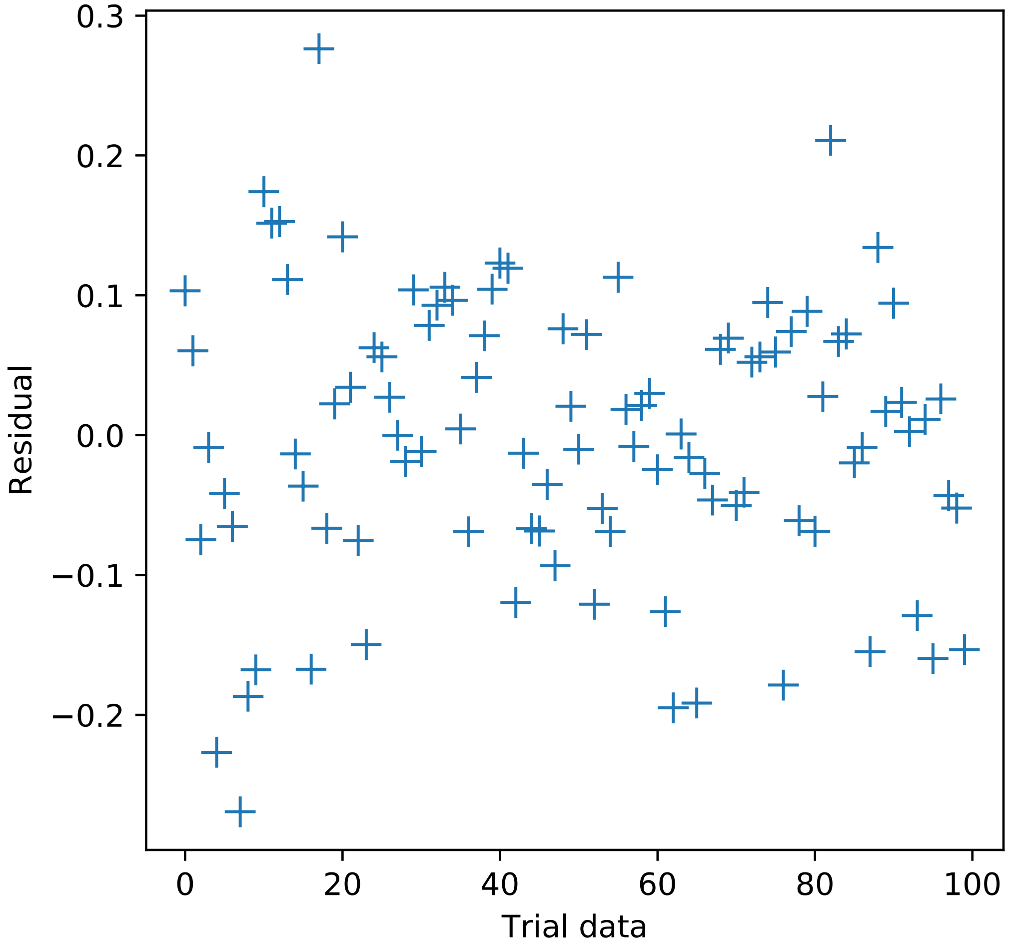
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Results from our neural net

Applied to training data



Applied to new data



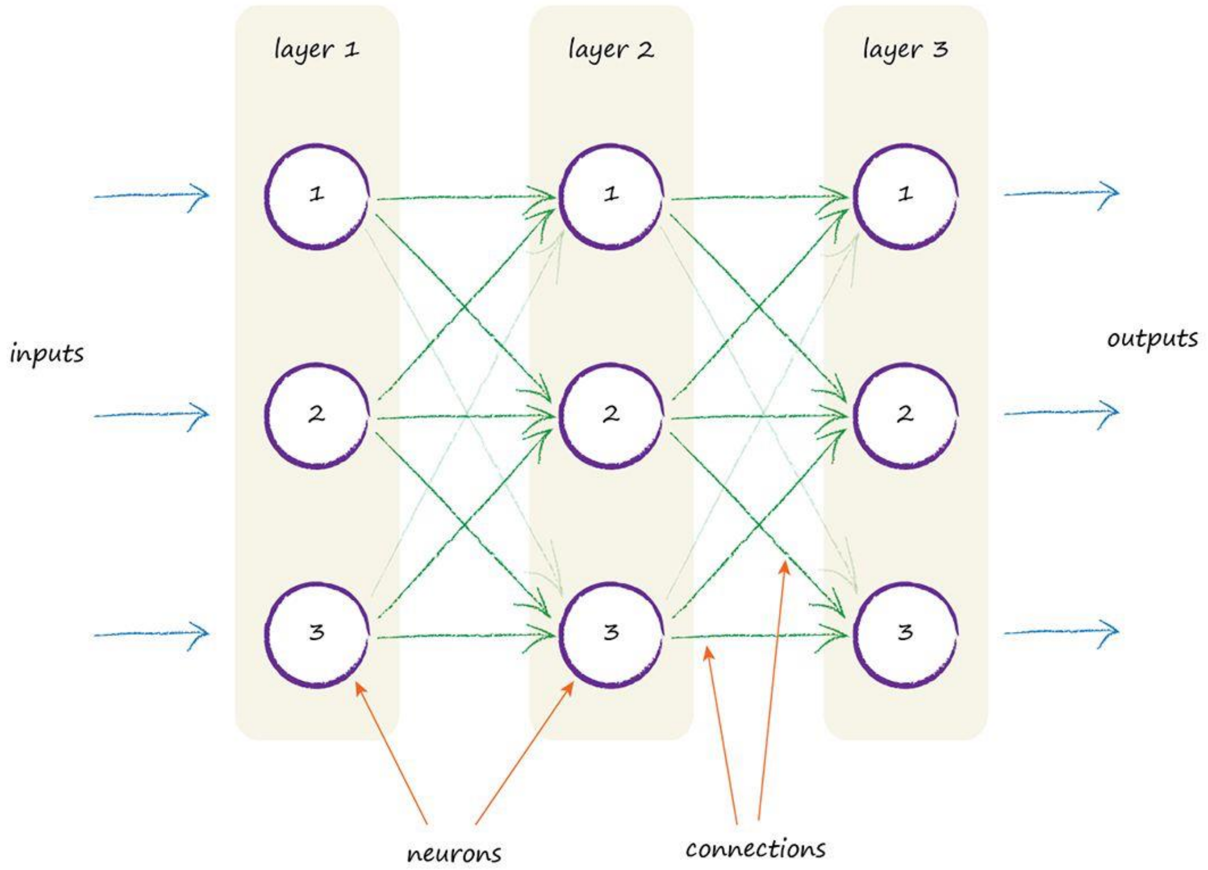
$$\mathbf{A} = \begin{bmatrix} -0.04 & 0.42 & 0.15 & -0.23 & 0.13 & 0.06 & 0.19 & -0.42 & 0.48 & 4.45 \end{bmatrix}$$

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 out 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* x *n* matrix **B** and a *m* x *k* matrix **A**
- Construct the output as:

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

- Note that we differentiate the applications of *g* because they may have different α's
- Extra shift of ½ 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:

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

- Where:

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

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

Back propagation

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

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

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

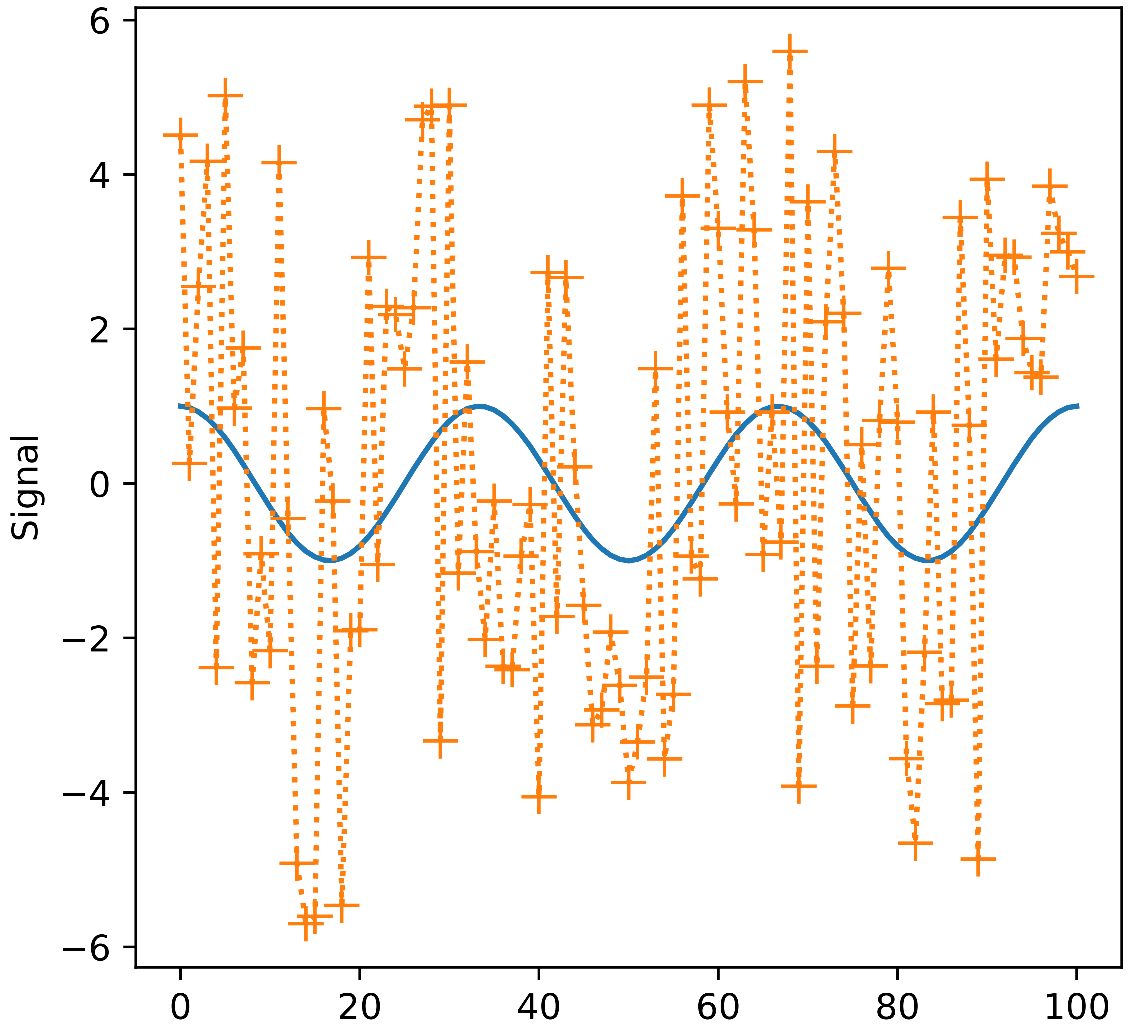
- So, the errors are “backpropagated” through the output and hidden layers

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:

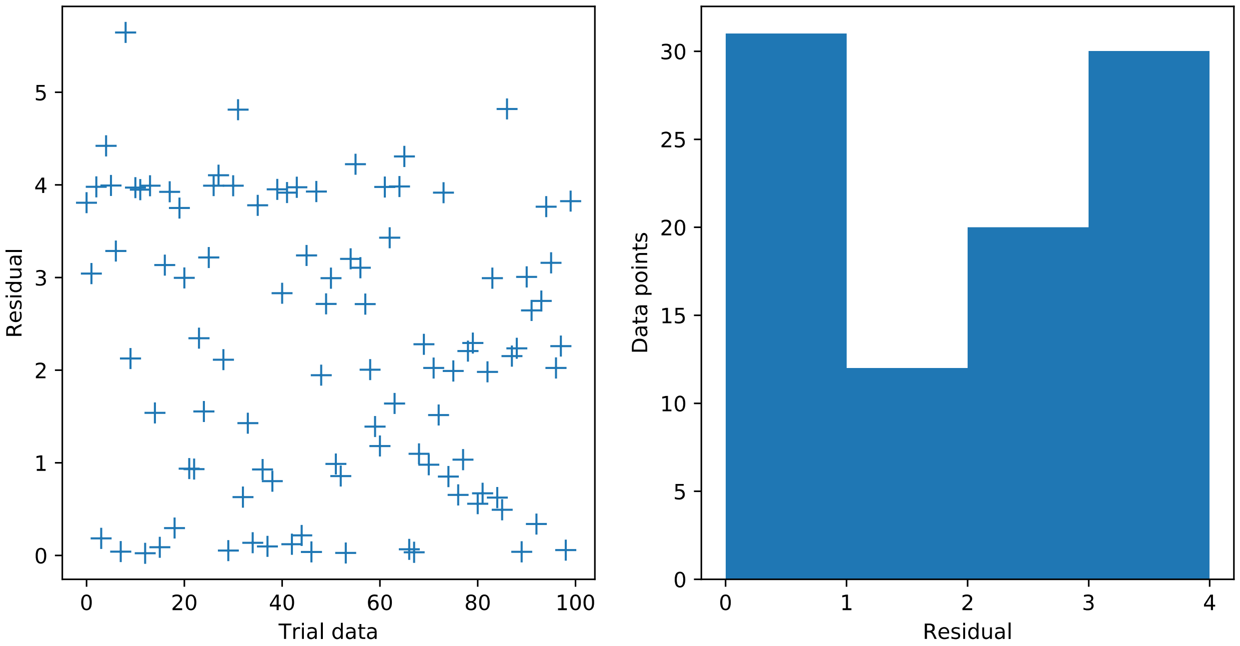
$$s(t) = \cos(2\pi f t) + 5\xi$$

  - ξ is a random number in [-1,1]
- Can we identify the frequency?

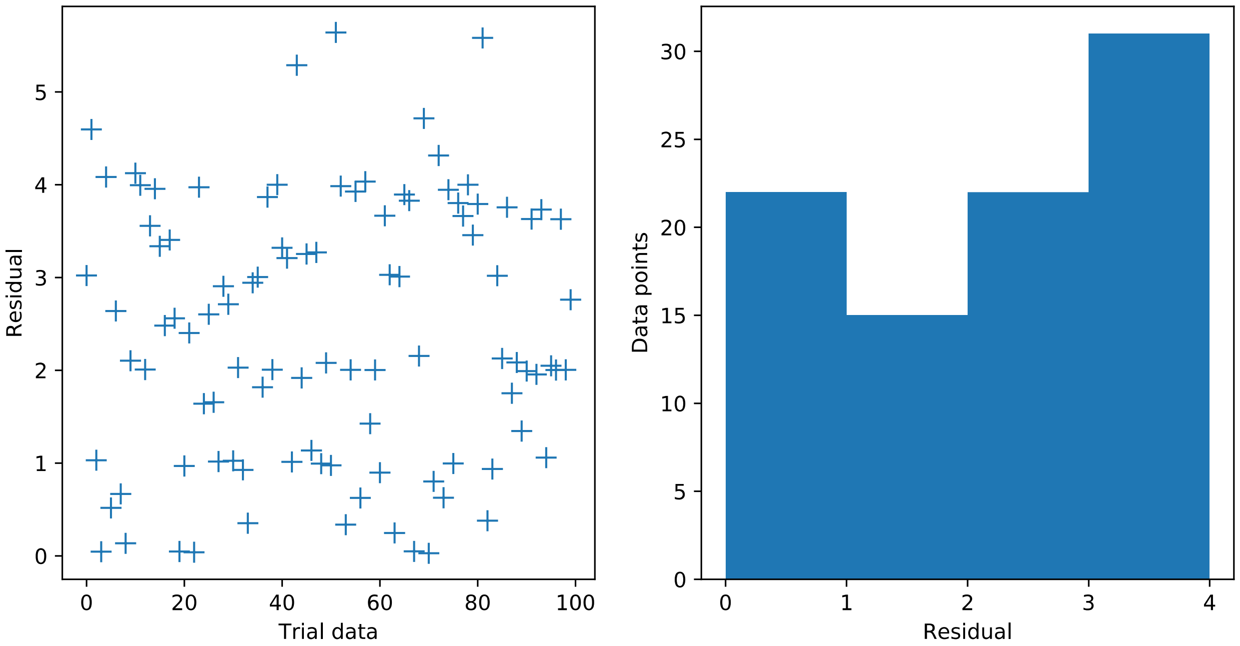


Signal analysis: Hidden layers size 2

Test on the training set

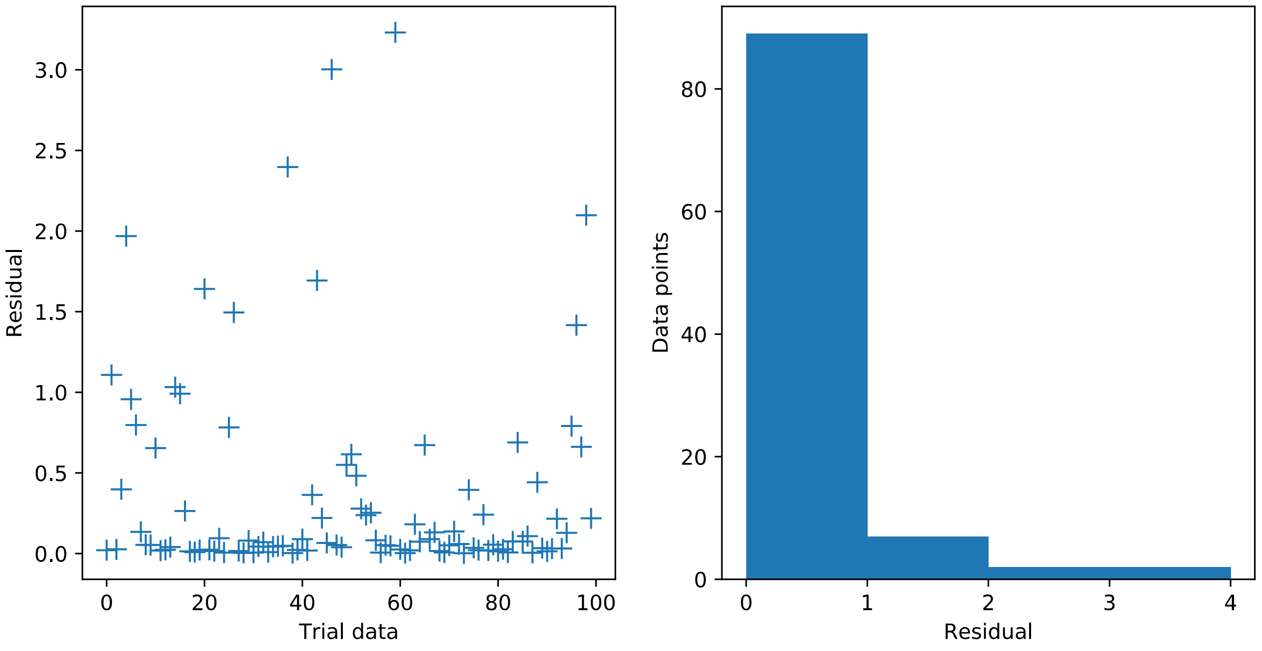


Test on new data

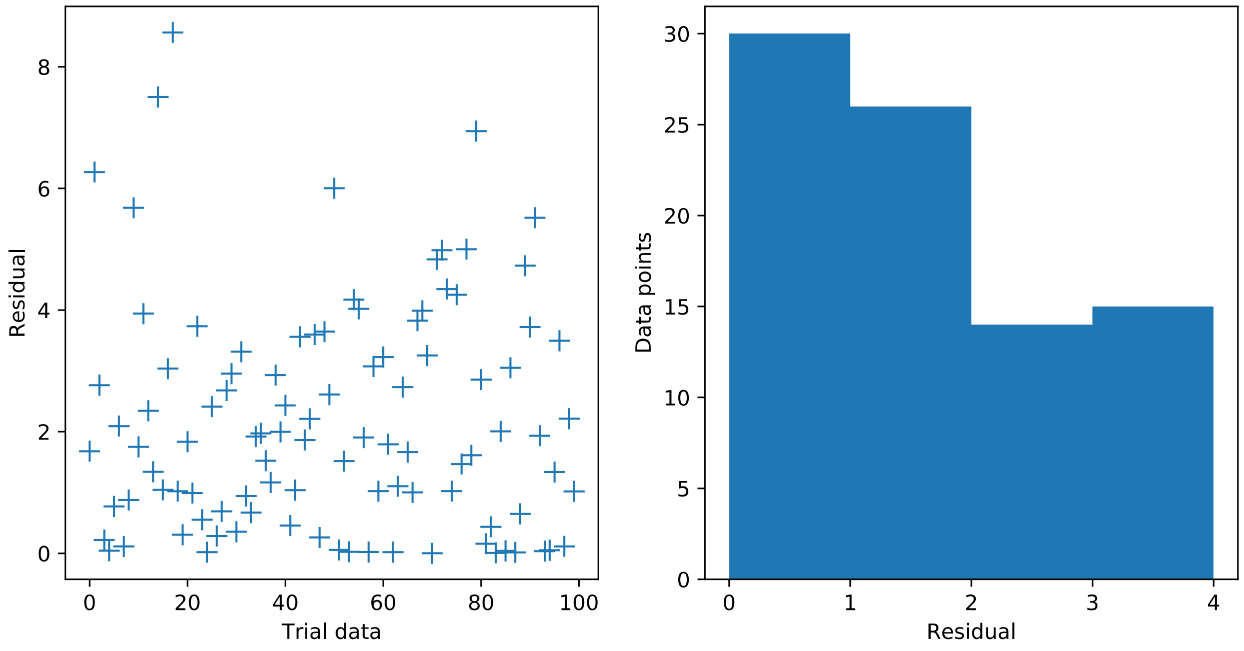


Signal analysis: Hidden layers size 3

Test on the training set

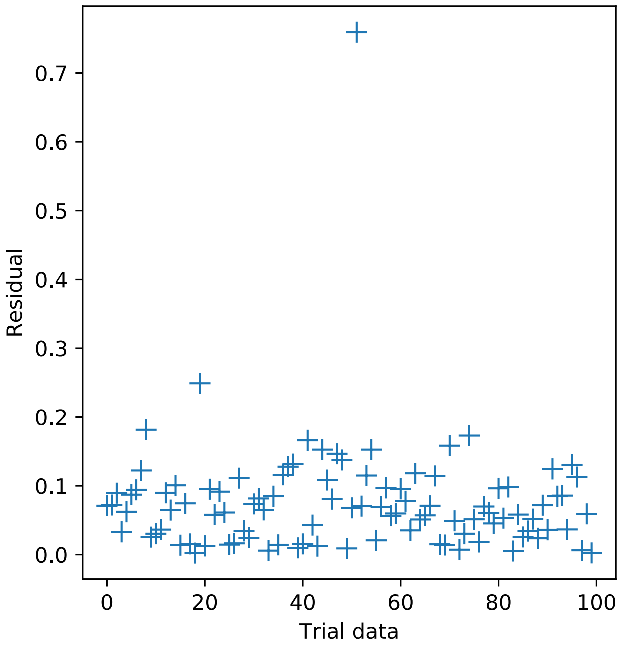


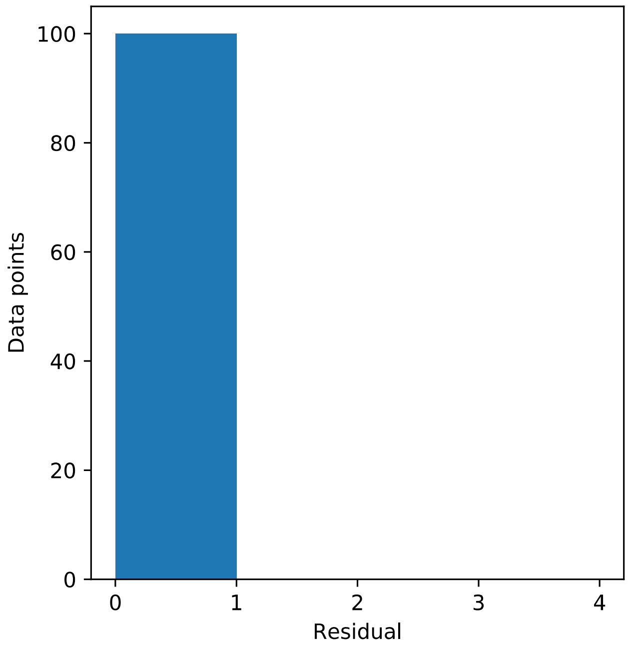
Test on new data



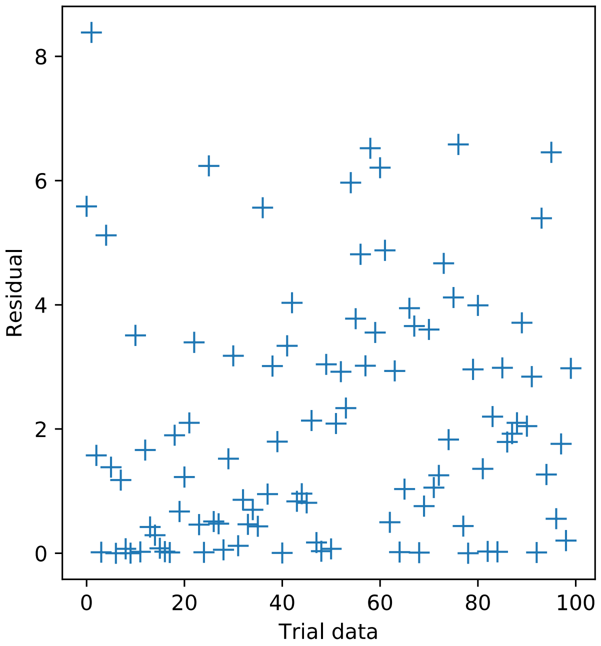
Signal analysis: Hidden layers size 4

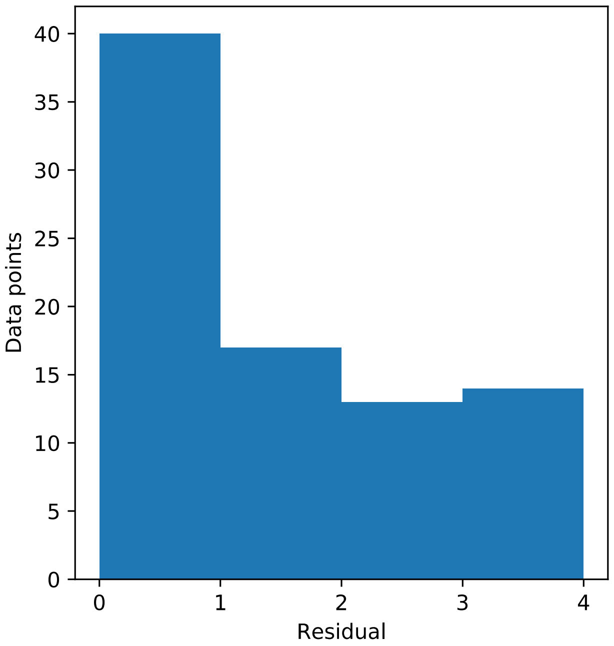
Test on the training set





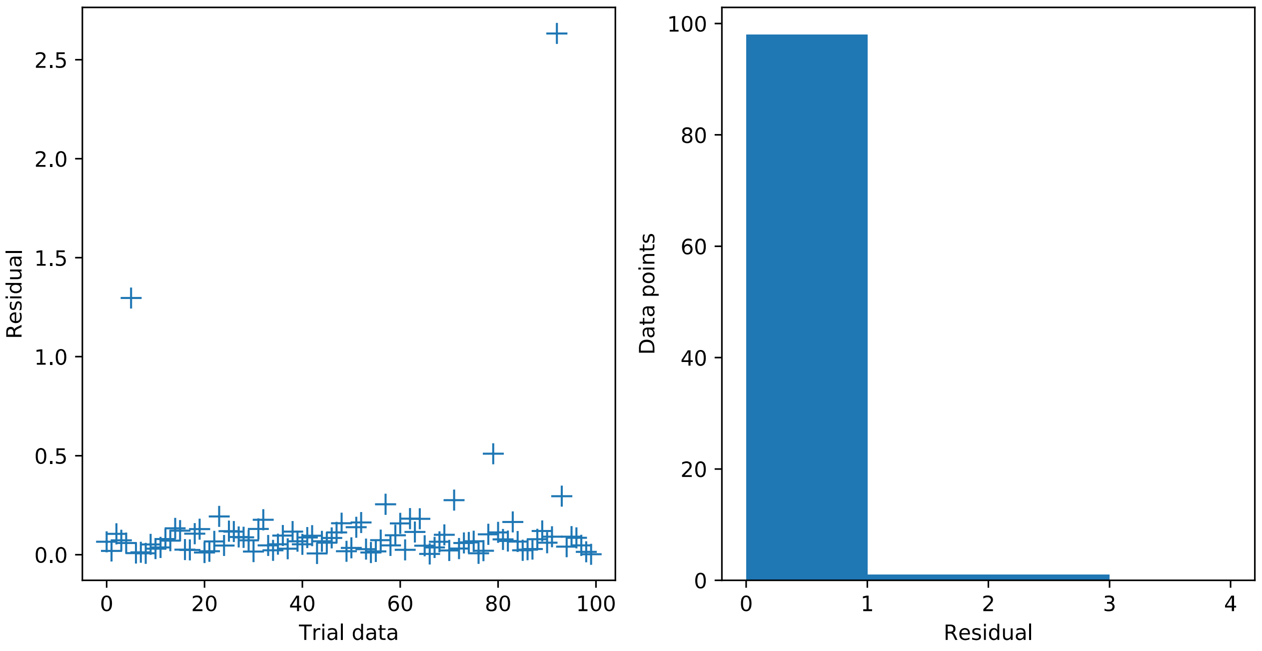
Test on new data



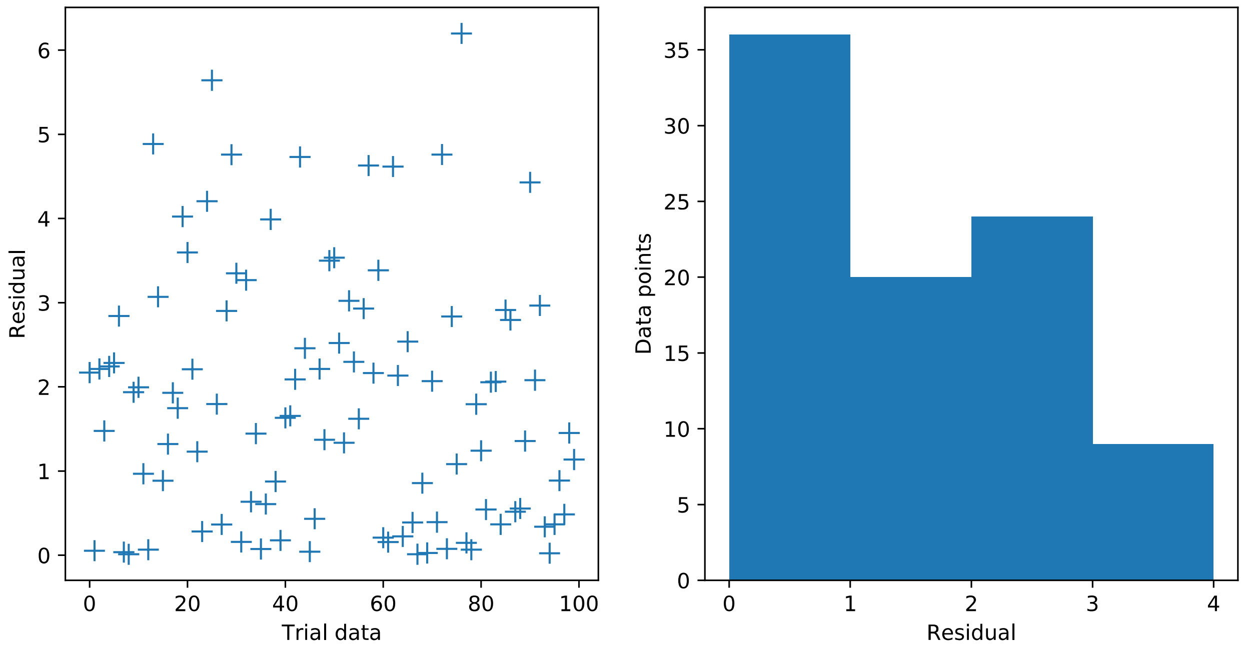


Signal analysis: Hidden layers size 8

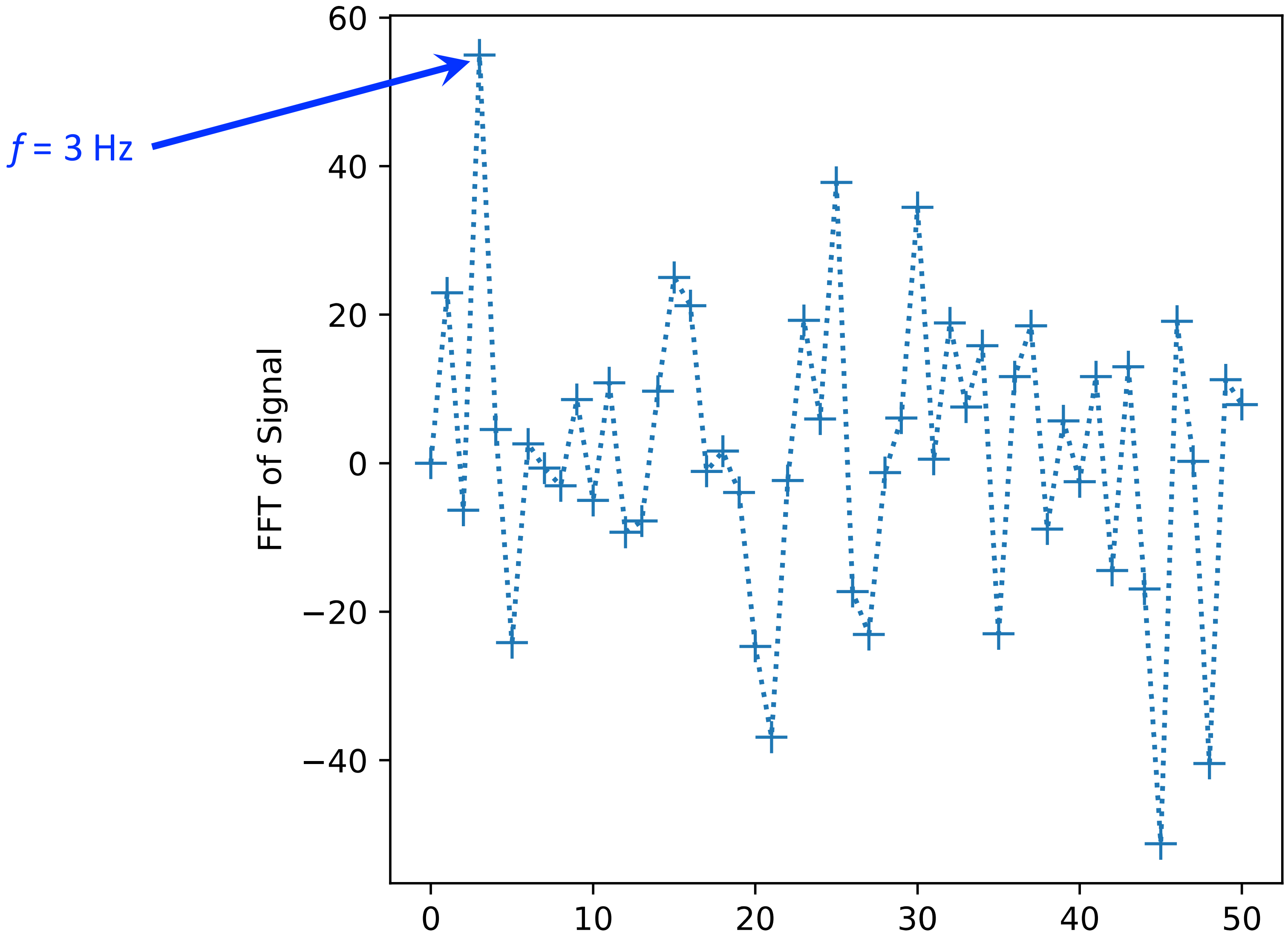
Test on the training set



Test on new data



Can we do the same with an FFT?

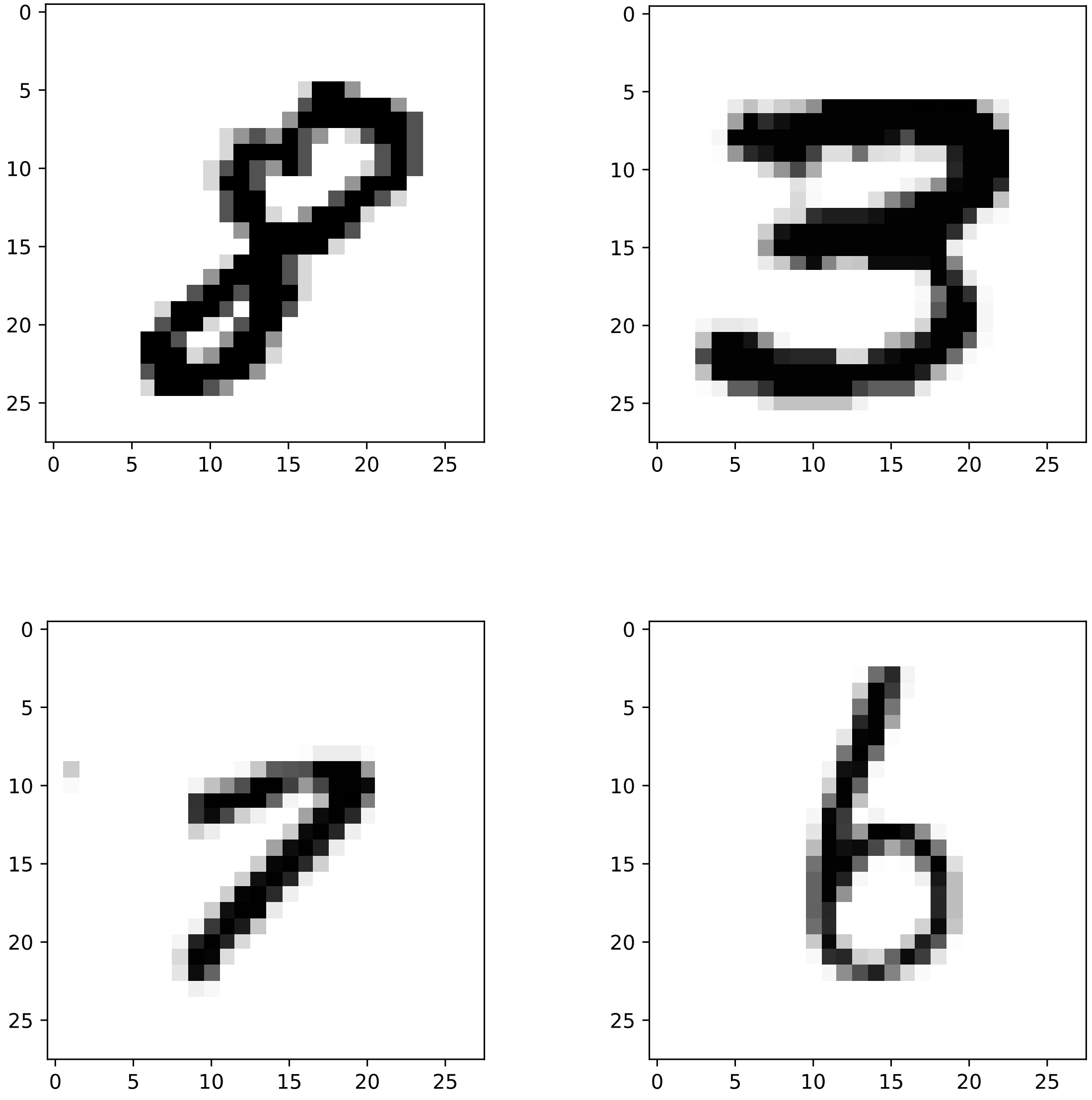


Another example: Recognizing written characters

- 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

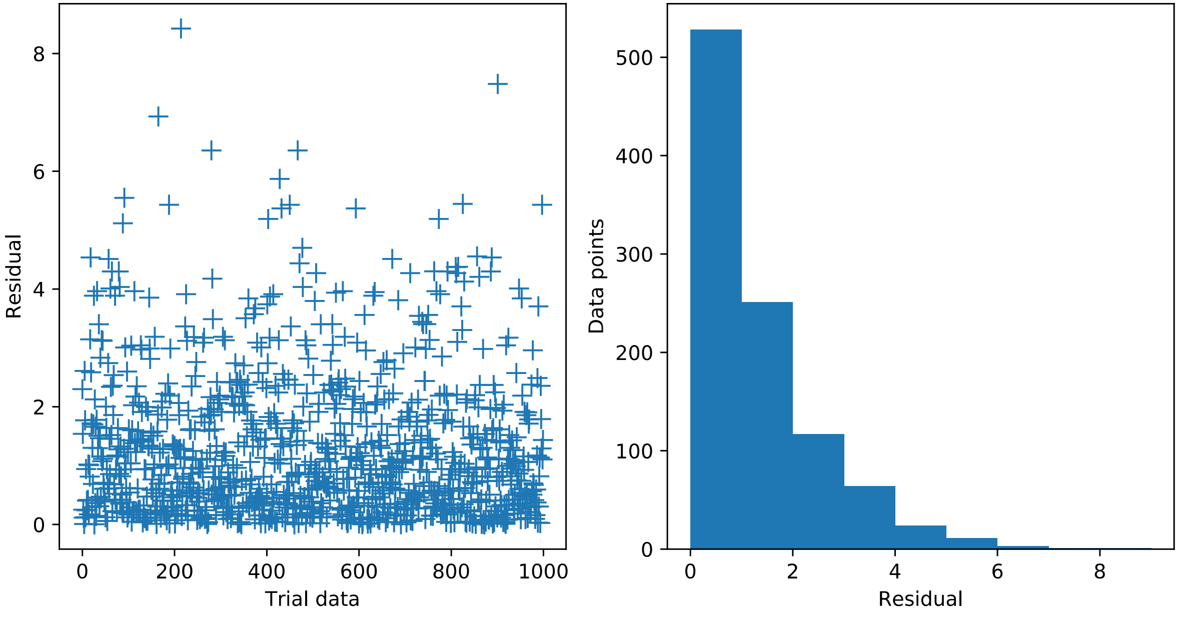
Another example: Recognizing written characters

- 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

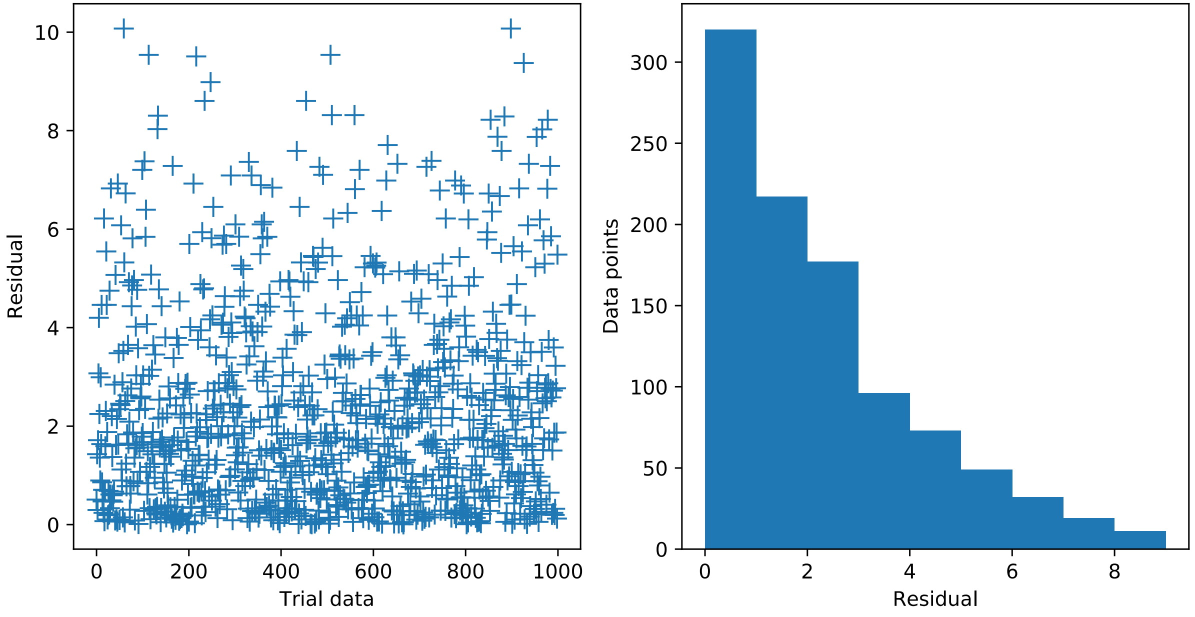


Another example: Recognizing written characters

Test on the training set



Test on new data



After class tasks

- HW4 graded, check your repositories!

- Readings:
  - *Computational Methods for Physics*, Joel Franklin, Chapter 14
  - *Make Your Own Neural Network*, Tariq Rashid
  - http://playground.tensorflow.org