ECE/CS 559 - Neural Networks Lecture Notes #3
Some example neural networks

Erdem Koyuncu

1 A concrete example

- We begin with a concrete example.
- We are given the points

$$\begin{aligned}\{-1,1\}^3 = \{&(-1,-1,-1),(-1,-1,1),(-1,1,-1),(-1,1,1),\\ &(1,-1,-1),(1,-1,1),(1,1,-1),(1,1,1)\}.\end{aligned}$$

- We wish to design the following classifier. Given the vector $(x_1, x_2, x_3) \in \{-1,1\}^3$, we wish to have the output $y = -1$ is the number of -1s in (x_1, x_2, x_3) is greater than the number of 1s. Otherwise, we want to have the output $y = 1$.
- We consider the signum function as our activation function:

$$\text{sgn}(x) = \begin{cases} 1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0. \end{cases}$$

- The following one-neuron "network" accomplishes this task:

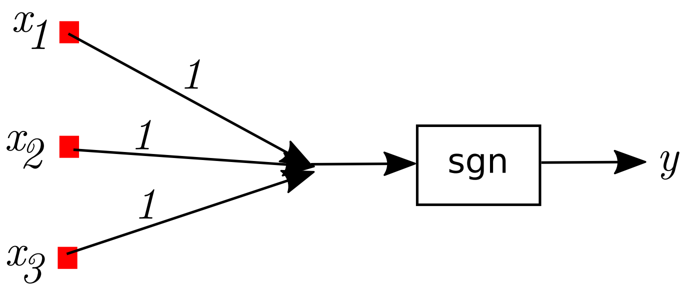

Figure 1: A one-neuron network for finding the component with the highest number occurrences.

- This can be generalized to any dimension of input vectors with components in $\{-1,1\}$ via $y = \text{sgn}(x_1 + \cdots + x_n)$. For even dimensions ($n$ even), the network will output a 0 when the number of occurrences of 1 equals the number of occurrences of -1. Otherwise, the network will output the component with the highest number of occurrences.

2 A neural network for pattern classification - Digit classifier

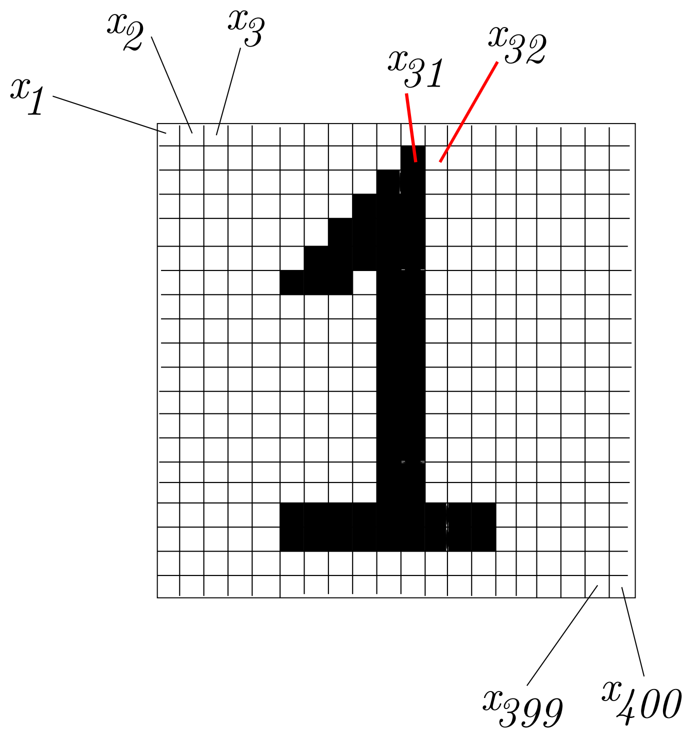

Figure 2: An image of a digit. Each pixel corresponds to one input neuron

- One may design a feedforward neural network for digit classification.
- We are given 20 pixel by 20 pixel images of handwritten digits. Our job is to find, for each given image, the actual digit corresponding to the image. Suppose that each pixel assumes binary values, i.e. the images are black and white only with no grayscale. One of the images may look like in Fig. 2.
- We can then consider a feedforward neural network with 400 nodes in the input layer (labeled as $x_1, x_2, \ldots, x_{400}$), some large number of nodes (say 1000 nodes) in a hidden layer, and 10 nodes in the output layer giving outputs $y_0, \ldots, y_9$. See Fig 3. Ignoring the biases, this fully-connected network has $400 \times 1000 + 1000 \times 10 = 410000$ parameters (synaptic weights) that we can optimize.

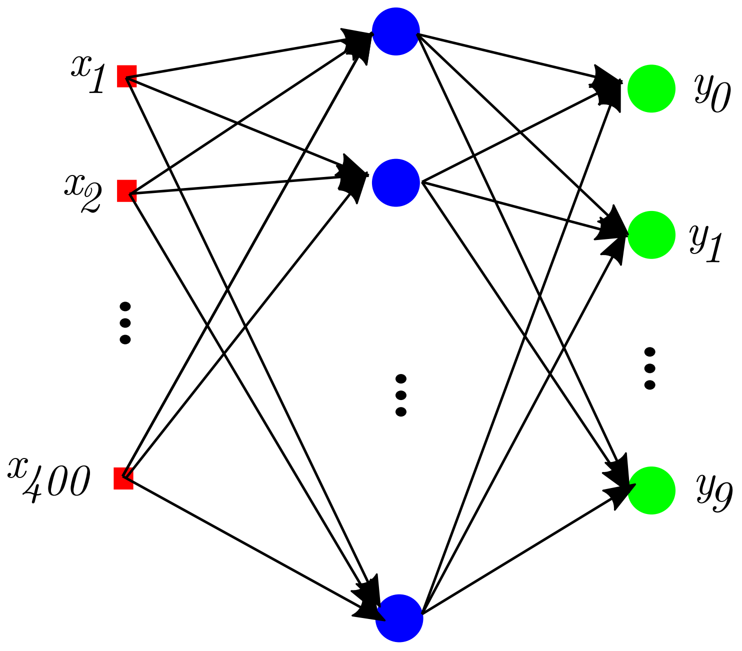

Figure 3: The 2-layer feedforward neural network for digit classification

- Each node in the input layer will correspond to one pixel in the image, as illustrated in Fig. 2. For example, if the image in Fig. 2 is to be fed to the network as input, the inputs would be $x_0 = 0$, $x_1 = 0, \ldots, x_{30} = 0, x_{31} = 1, x_{32} = 0, \ldots, x_{399} = 0, x_{400} = 0$, i.e. we use 0 to represent a white pixel, and we use 1 to represent a black square.
- It is possible to design the network (choose the weights) such that when some image of digit 0 is fed to the network, the output will be $y_0 = 1, y_1 = \ldots, y_9 = 0$. Likewise, for any $i \in \{0, \ldots, 9\}$, when some image of digit i is fed to the network the output will be $y_i = 1$ and $y_j = 0$, $\forall j \neq i$.
- This is usually done via supervised learning. We prepare (or usually acquire!) a large number of 20×20 training images each of which looks like Fig. 2 but of course with different digits, and differently written versions of the same digit. See Fig. 4 for some sample portion of such a training set. Each training image has a label that specifies the actual digit corresponding to the image. For example, in Fig. 4, the images in the first row will be labeled as 0, the second row as 1, and so on.

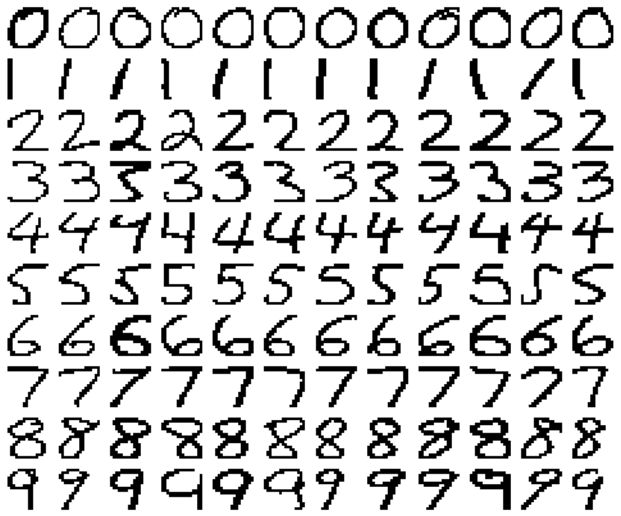

Figure 4: Part of a training set for digit classification

The supervised learning algorithm is then as follows. Usually, we begin with a random initialization of weights. We feed the first training image to the network with label, say, i (i.e. the actual digit that the image corresponds to). We then observe the outputs $y_0, \dots, y_9$. Note that the desired response for our training image is $y_i^{\text{desired}} = 1$ and $y_j^{\text{desired}} = 0$, $\forall j \neq i$. We update the weights according to how the outputs $y_0, \dots, y_9$ differ from the desired outputs that we wanted to have (There are different methods to update the weights, we will discuss these different methods later in the course). We repeat this process many times over all training images. After some point, the weights will converge and the network will be able to classify most of the digit images (even arbitrary images that are not in the set of training images) correctly.

3 Approximation of arbitrary functions

- The key idea of the previous section is that any kind of practical input (image, voice, etc) can be represented digitally as some point in a multidimensional space. For example, each 20×20 image is a point in a 400-dimensional space.
- One can then imagine the existence of a function, the ultimate correct classifier, that maps each point of this 400-dimensional space to the corresponding correct classification, or in general, to some arbitrary point of another multidimensional space. In general, it is extremely hard to determine exactly what the optimal classifier function should be. Instead, we approximate the optimal function via samples of the values assumed by the optimal function (our training set and their labels). Using these samples, we design a neural network that can approximate the highly complicated non-linear optimal classifier function.
- In fact, a neural network with even a single hidden layer can approximate any continuous function to arbitrary accuracy (provided that one can afford as many neurons as needed). This statement will be made more precise later on.
- The same ideas apply to any other application. For example, a sample spoken letter can be modeled as a mere point in another multidimensional space (if it is in digitized audio format). A neural network can then be designed to classify spoken letters. The exact same network model and the learning rules as in digit image classification can be applied to this (seemingly fundamentally different) problem as well.

- Hence, one interesting fact about neural networks is as follows: Our approach and solution to the problem is independent of the "specific nature of the problem." For example, we can use the same network topology and the same learning algorithms to solve both the digit image and spoken letter classification problems. On the other hand, if we were looking for "classical algorithmic" solutions, then an algorithm for digit image classification would be fundamentally different from an algorithm for spoken letter classification; the solution for one problem would not be applicable to the other.

4 Memory and restoration of patterns

- Recurrent networks can be used as memory.

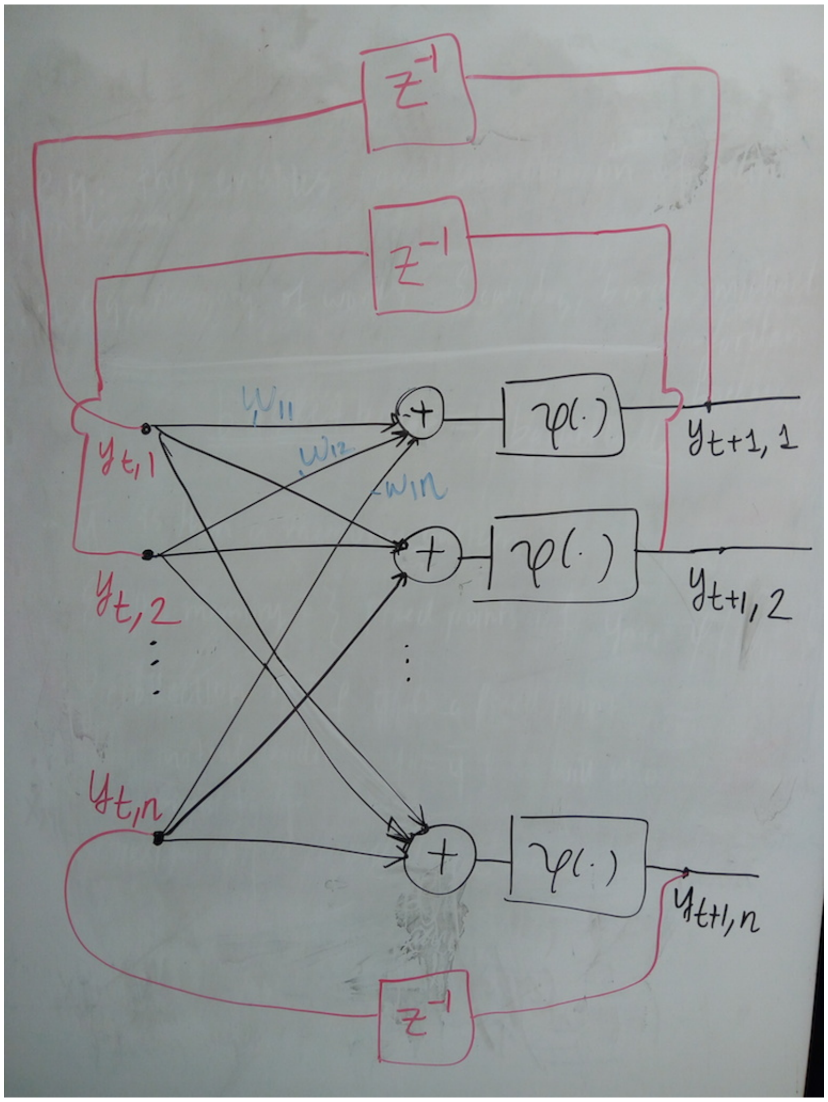

Figure 5: Recurrent network with no self-feedback loops and no hidden neurons.

- Call the outputs at time t as $\mathbf{y}_t = [y_{t,1} \cdots y_{t,n}]^T$, where $[\cdot]^T$ is the tranpose. The input-output relationships are

$$y_{t+1,i} = \phi\left(\sum_{j=1}^{n} w_{ij} y_{t,j}\right), \; i = 1, \ldots, n. \tag{1}$$

Let $\mathbf{W}$ be the $m \times m$ matrix whose element in the ith row, jth column is w_{ij}. We have the compact version of (1)

$$\mathbf{y}_{t+1} = \phi\left(\mathbf{W}\mathbf{y}_t\right).$$

in the sense that $\phi(\cdot)$ is to be applied component-wise.

- The fixed points of the relation $\mathbf{y}_{t+1} = \phi(\mathbf{W}\mathbf{y}_t)$ (i.e. points of the form $\mathbf{y} = \phi(\mathbf{W}\mathbf{y})$) are the memory patterns stored by the network.
- Under certain conditions on $\mathbf{W}$ and ϕ, any initial pattern will converge to a stored memory pattern.
- In particular, if $\mathbf{y}$ is a stored pattern (a fixed-point), then small initial patterns of the form $\mathbf{y} + \boldsymbol{\epsilon}$ will also converge to $\mathbf{y}$.
- Practical example: Your usual human memory!, i.e. one can imagine that all the valid English words are stored in our memory: E.g. cat, dog, precipitation, incomprehensibilities. We are able to restore the original words in spite of small perturbations (missing letters or typos), e.g.

  incoprehansibilities $\rightarrow$ incomprehensibilities.

  Micheel Jourrdan $\rightarrow$ Michael Jordan
- Example:

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

$$\mathbf{y}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

$$\mathbf{y}_1 = \phi(\mathbf{W}\mathbf{y}_0) = \phi\left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

$$\mathbf{y}_2 = \phi(\mathbf{W}\mathbf{y}_1) = \phi\left(\begin{bmatrix} 3 \\ 3 \\ 3 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \mathbf{y}_1$$

Hence $\mathbf{y}_1$ is a stored pattern. There may be other stored patterns, e.g. $-\mathbf{y}_1$ is also a stored pattern. With more neurons, they may be several other non-trivial stored patterns. Also, in general, convergence may not occur, e.g. oscillatory behavior:

$$\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \rightarrow \cdots$$

- One can use recurrent networks for digit classification:

    - Somehow install each digit as memories (fixed points of the non-linear recurrence relation), i.e. find an appropriate $\mathbf{W}$.
    - Any new image of a digit can be fed to the network, and hopefully will converge to one of the already-stored memories.

- The question of how to design $\mathbf{W}$ (so that we retain some specific memories) will be discussed later on.