NYU Paris **Spring 2019**

CSCI-UA 9473 Introduction to Machine Learning

Problem Set 5: Neural Networks

The exercises vary in difficulty. More advanced exercises are marked with one or two stars.

1. Perceptron learning rule

Recall that the simple perceptron is defined as

$$y(x) = \sigma(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + b) \tag{1}$$

The same model can be defined in the feature space as

$$y(x) = \sigma(\langle \boldsymbol{a}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b) \tag{2}$$

To train the perceptron on binary $+1/-1$ targets, we include the bias b in the weight vector, $\boldsymbol{w} = (\boldsymbol{a}, b)$, introduce the extended feature vector $\tilde{\phi}(\boldsymbol{x})$, and consider the criterion

$$E_P(\boldsymbol{w}) = -\sum_{n \in \text{misclassified}} \boldsymbol{w}^T \tilde{\boldsymbol{\phi}}(\boldsymbol{x}_n) t_n \tag{3}$$

To find the parameters $\boldsymbol{a}$ and b that minimize (3), we can then follow the update rule

$$\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k - \eta \nabla E_p^n(\boldsymbol{w}) \tag{4}$$

Where ∇E_p^n is the gradient (with respect to $\boldsymbol{w}$) of the error defined on a single training sample $\boldsymbol{x}_n$ (or $\phi(\boldsymbol{x}_n)$). We then loop over each of the samples from the training set that are misclassified,

$$\boldsymbol{w}^{k+1} \leftarrow \boldsymbol{w}^k - \eta \nabla E_p^n(\boldsymbol{w}), \quad n = 1, \ldots, N_{\text{misc.}} \tag{5}$$

And repeat the process until there are no misclassified points left. We can set the learning rate, $\eta = 1$ without loss of generality.

There are several choices for the activation function f. In the exercises below we take the "hardlim" function, $f(r) = +1$ if $r \geq 0$ and -1 if $r < 0$.

Note: when the targets are in $\{0, 1\}$ (as in some of the exercises below), we need to take a slightly different approach. In this case, we use e to denote the error $e = t - \sigma(\langle \boldsymbol{a}, \boldsymbol{x} \rangle + b)$ and we define the update as

$$\left(\boldsymbol{a}^{(k+1)}, b^{(k+1)}\right) \leftarrow \left(\boldsymbol{a}^{(k)}, b^{(k)}\right) + \begin{cases} ((\boldsymbol{\phi}(\boldsymbol{x}_j)), 1) & \text{if } e = \pm 1 \\ \boldsymbol{0} & \text{if } e = 0 \end{cases} \tag{6}$$

In other words, as before, we update the weights only with prototypes that are misclassified.

Exercise 1 (source: NN design [1]). *Consider the classification problem defined by the pairs $\{\boldsymbol{x}_i, t_i\}$, where $\boldsymbol{x}_1 = (-1, 1)$, $\boldsymbol{x}_2 = (0, 0)$, $\boldsymbol{x}_3 = (1, -1)$, $\boldsymbol{x}_4 = (1, 0)$ and $\boldsymbol{x}_5 = (0, 1)$ as well as $t_1 = 1$, $t_2 = 1$, $t_3 = 1$, $t_4 = 0$ and $t_5 = 0$.*

- *Draw a diagram of the single neuron perceptron you woul use to solve this problem.*
- *Draw a graph of the data points and color them according to their target. Is this problem solvable via the network you defined in item 1?*

Exercise 2 (source: NN design [1]). *Consider the following pairs $\{(-1, 1), 1\}$, $\{(-1, -1), 1\}$, $\{(0, 0), 0\}$, $\{(1, 0), 0\}$.*

- *Design a single-neuron perceptron to solve this problem. You can design the network graphically by choosing the weight vectors that are orthogonal to the decision boundaries.*

- *Double check your solution on the training samples. Then classify the four prototypes* $(-2,0)$, $(1,1)$, $(0,1)$, $(-1,-2)$
- *Which of the test vectors will always be classified the same way regardless of the parameters of your network? Why?*

Exercise 3 (source: NN design [1])**.** *Solve the classification problem of exercise 2, by using the perceptron learning rule with initial parameters* $\boldsymbol{w}^0 = (0,0)$ *and* $b^0 = 0$

Exercise 4 (source: NN design [1])**.** *We have two categories of prototypes*

$$\mathcal{C}_1 = \{(0,0),(-1,0),(0,1)\} \tag{7}$$

$$\mathcal{C}_2 = \{(-1,1),(0,2),(-2,0)\} \tag{8}$$

- *Design a single neuron perceptron that can discriminate between those two classes*
- *Draw the diagram*
- *Sketch the decision boundary*
- *Consider the following additional prototype,* $(-3,0)$*, will your network classify it correctly?*

Exercise 5 (source: NN design [1])**.** *We again want to train a perceptron with the following pairs:* $\{(-1,-1),0\}$, $\{(0,0),0\}$ *and* $\{(-1,1),1\}$*. We consider the initialization given by* $\boldsymbol{\beta}_1^0 = (1,0)$ *and* $\beta_0^0 = 0.5$.

- *Plot the initial decision boundary, weight vector and input patterns*
- *Train the network with the perceptron learning rule. Present each vector once, in the order given.*
- *Plot the final decision boundary and demonstrate graphically which patterns are correctly classified*
- *Will the perceptron learning rule always learn to correctly classify the patterns in this training set, no matter what initial weights are used? explain.*

2. Neural networks, backpropagation

Training Neural Networks, part I.

As an illustration, we consider a one hidden layer network which outputs K values (those can be K binary values encoding a class for example), the k component of the final output, y_k is then defined as

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)}\sigma^{(1)}\left(\sum_{i=1}^{D} w_{ji}^{(1)}x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{9}$$

When training a neural network, we will minimize the "loss" function

$$\sum_{n=1}^{N}\sum_{k=1}^{K}(t_{n,k} - y_k(x_n))^2 \tag{10}$$

In (10), t_{nk} is the value of the k^{th} entry in the n^{th} training samlple. Think of it as binary vector

$$t_n = \underbrace{(0,1,0,\dots,0)}_{K\text{entries}} \tag{11}$$

We then want the k^{th} output of y to match the k^{th} value in t_n.
In most application we will minimize on one (or a few) training samples at a time, so we can drop the sum over n

$$\text{Loss} = E = \sum_{k=1}^{K}(t_k - y_k(x))^2 \tag{12}$$

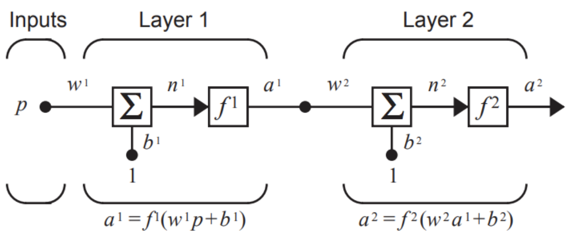

FIGURE 1. illustration for Exercise 6. The figure is taken from the book *Neural Network Design* by Hagan, Demuth, Beale and De Jesús.

Training Neural Networks, part II: Backpropagation.
When training the neural network, we will be interested in applying the gradient descent algorithm to minimize the function (10) with respect to the parameters (weights) of the networks. We thus get iterates of the form

$$\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta \frac{\partial (t_k - y_k(\boldsymbol{x}_n))^2}{\partial \boldsymbol{w}} \tag{13}$$

To compute the derivative with respect to each weight, we first introduce the notations

$$a_k^{(\ell)} = \sum_j w_{k,j}^{(\ell)} z_j^{(\ell)} = \sum_j w_{k,j}\sigma(a_j^{\ell-1}) \tag{14}$$

We will then use the "backpropagation" relation

$$\frac{\partial E}{\partial a_j^{\ell-1}} = \sum_{k=1}^{D} \frac{\partial E}{\partial a_k^{\ell}} \left(\frac{\partial a_k^{\ell}}{\partial a_j^{\ell-1}} \right) \tag{15}$$

Which we write compactly by letting $\delta_j^{(\ell)}$ to denote the derivative $\frac{\partial E}{\partial a_j^{\ell-1}}$ also known as "sensitivity", we thus have the backpropagation rule

$$\delta_j^{(\ell-1)} = \sum_{k=1}^{D} \delta_k^{(\ell)} \sigma^{\ell\prime}(a_j^{\ell-1}) \tag{16}$$

We start the chain with

$$\delta_k^L = \frac{\partial L}{\partial a_k^L} = \tag{17}$$

and then compute all the $\delta_i^{(\ell)}$. Once those $\delta_i^{(\ell)}$ have been computed, the derivatives are simply given by

$$\frac{\partial E}{\partial w_{ji}^{(\ell-1)}} = \frac{\partial E}{\partial a_j^{\ell-1}} \frac{\partial a_j^{(\ell-1)}}{\partial w_{ij}^{(\ell-1)}} \tag{18}$$

$$= \delta_j^{\ell} z_i^{\ell} \tag{19}$$

Exercise 6. *For the neural network shown in Fig. 1, the initial weights and biases are chosen to be*

$$w^1(0) = 1, b^1(0) = -2, w^2(0) = 1, b^2(0) = 1 \tag{20}$$

The network transfer functions are

$$f^1(x) = x^2, \quad f^2(x) = \frac{1}{x} \tag{21}$$

We consider the input/ target pair given by $\{p = 1, t = 1\}$. Perform one iteration of backpropagation with $\alpha = 1$.

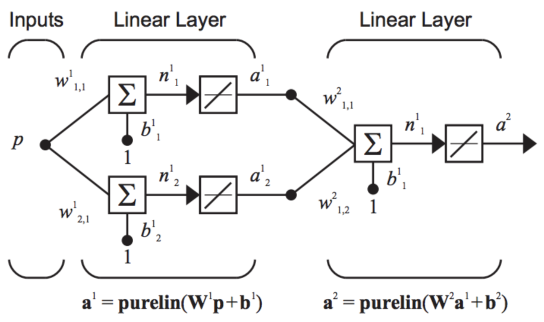
Inputs
Linear Layer
Linear Layer
$w^1_{1,1}$
$w^1_{2,1}$
p
n^1_1
a^1_1
b^1_1
n^1_2
a^1_2
b^1_2
1
$w^2_{1,1}$
$w^2_{1,2}$
n^1_1
a^2
b^1_1
$\mathbf{a}^1 = \mathbf{purelin}(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$
$\mathbf{a}^2 = \mathbf{purelin}(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$

Figure 2. Exercise 7, source: NN design

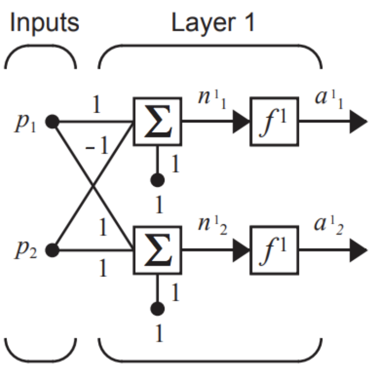

Figure 3. Material for Exercise 8, source: Neural Network design [1]

Exercise 7 (source: NN design)**.** *Consider the neural network of Fig. 2. With the following input and target, $\{p = 1, t = 2\}$. The initial weights and biases are given by $\boldsymbol{W}^1(0) = (1, -1)$, $\boldsymbol{W}^2(0) = (-1, 1)$, $\boldsymbol{b}^1(0) = (2, 1)$ and $\boldsymbol{b}^2(0) = 3$.*

- *Apply the input to the network and make one pass forward through the network to compute the output and the error.*
- *Compute the sensitivities*[1] *by backpropagating through the network*
- *Compute the derivative $\frac{\partial e^2}{\partial w^1_{1,1}}$ using the result of item 2. (here $e^2 = (t - a^2)^2$)*

Exercise 8 (source: NN design)**.** *For the network shown in Fig. 3, with $f^1(n) = n^2$, and the training data $\boldsymbol{p} = [1, 1]$, $\boldsymbol{t} = [8, 2]$, perform one step of backpropagation.*

[1]The sensitivities are defined as $\frac{\partial \hat{F}}{\partial n^m_i}$ where $\hat{F} = \sum_\ell (t^k(\ell) - a^k(\ell))^2$ is the squared error at iteration k

3. Regularization**

Regularization and Generalization.
Because of their representation power, neural networks are also prone to overfitting. For this reason, when training a neural network, we usually add various constraints on the elements of the network. Among such constraints, one can for example minimize the norm of the weights along with the error when training the network. We then end up with an objective of the form

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{22}$$

Where $E = (t_k - y_k(x))^2$ is the usual RSS.
An alternative is to let the gradient iterations go on for a while and then stop those iterations when the test error starts increasing. This second approach is known as early stopping

Exercise 9 (***Comparing Early stopping and Regularization. Source: NN design)**.** *In order to understand the connection between regularization and early stopping, we consider the simple (ADALINE) network with linear activation function shown in Fig. 4. The ouput of this network is given by $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{p} + \boldsymbol{b}$ and the i^{th} entry of this vector is thus given by $y_i = \boldsymbol{w}_i^T \boldsymbol{p} + b_i$.*

- *We start by considering the ADALINE network with a single neuron (S=1) and $\boldsymbol{w}$ is a vector. Show that the training error*

$$E = \sum_{i=1}^{N_{training}} (t_i - a(p_i))^2 \tag{23}$$

  can read as

$$c + \boldsymbol{d}^T \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \tag{24}$$

  for appropriate $\boldsymbol{A}$ and b and $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$

- *Let $\boldsymbol{x}^{ML} = -\boldsymbol{A}^{-1}\boldsymbol{d}$ denote the maximum likelihood solution for $\boldsymbol{x}$. Write one iteration of gradient descent on the error function (24) with a constant step size α and show that such an iteration can read as*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}^{ML}) \tag{25}$$

- *Equivalently show that the iterations of gradient descent on this function can read as $\boldsymbol{x}_{k+1} = \boldsymbol{M}\boldsymbol{x}_k + (\boldsymbol{I} - \boldsymbol{M})\boldsymbol{x}^{ML}$ for an appropriate matrix $\boldsymbol{M}$*
- *Start with $\boldsymbol{x}_0$ and show that after two iterations, we have*

$$\boldsymbol{x}_2 = \boldsymbol{M}^2 \boldsymbol{x}_0 + \left(\boldsymbol{I} - \boldsymbol{M}^2\right) \boldsymbol{x}^{ML} \tag{26}$$

- *Finally show that k iterations of gradient descent give*

$$\boldsymbol{x}_k = \boldsymbol{M}^k \boldsymbol{x}_0 + [\boldsymbol{I} - \boldsymbol{M}^k] \boldsymbol{x}^{ML} \tag{27}$$

We now consider traditional regularization around $\boldsymbol{x}_0$ (as opposed to early stopping). The loss we want to minimize has the form

$$E + \rho(\boldsymbol{x} - \boldsymbol{x}_0)^T (\boldsymbol{x} - \boldsymbol{x}_0) \tag{28}$$

Where E is as before given by $E = c + \boldsymbol{d}^T \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$.

- *Compute the gradient and set it to zero. Show that what you obtain is equivalent to the equation*

$$\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^{ML}) = -2\rho(\boldsymbol{x} - \boldsymbol{x}^{ML}) - 2\rho(\boldsymbol{x}^{ML} - \boldsymbol{x}_0) \tag{29}$$

  or equivalently

$$(\boldsymbol{A} + 2\rho \boldsymbol{I})(\boldsymbol{x} - \boldsymbol{x}^{ML}) = 2\rho(\boldsymbol{x}_0 - \boldsymbol{x}^{ML}) \tag{30}$$

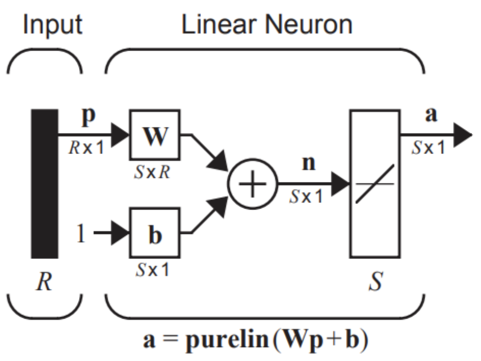

FIGURE 4. Simple linear network used in exercise 9, source: Neural Network design [1]

- *Solve for* $\boldsymbol{x}$ *and show that the solution for* $\boldsymbol{x}$ *can read as*

$$(\boldsymbol{x} - \boldsymbol{x}^{ML}) = \boldsymbol{M}_\rho(\boldsymbol{x}_0 - \boldsymbol{x}^{ML}) \tag{31}$$

  where $\boldsymbol{M}_\rho = 2\rho(\boldsymbol{A} + 2\rho\boldsymbol{I})^{-1}$
- *Finally, using your solution, show that we can write*

$$\boldsymbol{x}^{reg} = \boldsymbol{M}_\rho\boldsymbol{x}_0 + [\boldsymbol{I} - \boldsymbol{M}_\rho]\boldsymbol{x}^{ML} \tag{32}$$

  How does this solution compare to early stopping. Explain the connection between the two.

4. Designing Multi-layer networks*

Neural Networks Design. In this part, we will study how to design neural network to perform binary classification tasks by extending and repeating the separating hyperplane idea. Training a single perceptron is easy as it only implies determining the positioning of a single hyperplane. When considering data that is not linearly separable, we will need to combine several such hyperplanes. Consider the example below which is taken from [1]. We want to design a network that is able to discriminate between the black and the white dots in Fig. 5 below.

Example 4.1. Designing a multilayer perceptron.
In this example, we will design a network to separate the black dots from the white dots in Fig. 5. The first step is to design 11 separating planes that determine each of the boundaries separating the black region from the white one. One can show that each of those planes have equation $w_1x_1 + w_2x_2 + b$ where (w_1, w_2, b) are given by

$$(\boldsymbol{W}^{(1)})^T = \left[\begin{array}{ccccccccccc} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \end{array}\right], \tag{33}$$

$$\boldsymbol{b} = [-2,\ 3,\ 0.5,\ 0.5,\ -1.75,\ 2.25,\ -3.25,\ 3.75,\ 6.25,\ -5.75,\ -4.75] \tag{34}$$

The output of the system $\boldsymbol{W}^{(1)}\left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] + \boldsymbol{b}$ is now positive or negative depending on which side of the planes we stand. If we add a step function $\sigma(x) = 1$ if $x \geq 0$ and $\sigma(x) = -1$ if $x < 0$, then the output of $\sigma(\boldsymbol{Wx} + \boldsymbol{b})$ is now a 11 components vector of $+1$ and -1 where the entry k is $+1$ if we are above the k^{th} plane and -1 otherwise. We will add another layer to combine those $+1/-1$ values in order to define the region.

The second layer will be used to delimit each of the intermediate 4 Black regions of Fig. 7. We again define a matrix of 4 hyperplanes which take the 11 dimensional output of $\boldsymbol{y\sigma}(W^{(1)}\boldsymbol{x} + \boldsymbol{b})$.

The first plane should output a positive value only when were are inside region 1 (that is when the ouputs of planes 5, 6, 9 and 11 from Fig. 6 are positive), the second plane will output a positive value when we are inside the third region and finally the fourth plane will return a positive value when we are inside region 4. To do this, we let

$$\boldsymbol{W}^{(2)} = \left[\begin{array}{ccccccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right], \quad \boldsymbol{b} = \left[\begin{array}{c} -3 \\ -3 \\ -3 \\ -3 \end{array}\right] \tag{35}$$

From this second set of planes, you see that all four planes of each region has to ouput a positive value otherwise the -3 in $\boldsymbol{b}$ leads to a negative. We then add the activation σ and get the two layers network

$$\sigma(\boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{p} + \boldsymbol{b}^1) + \boldsymbol{b}^2) \tag{36}$$

which outputs 4 values which are positive in each of the 4 regions of Fig. **??** and negative outside those regions. We are now left with grouping those four values into a last single value which will be positive only if the four region values are all positive. To achieve this result, we simply repeat the ideas developed above and introduce the plane and constant bias

$$\boldsymbol{W}^{(3)} = [1,\ 1,\ 1,\ 1], \quad b = -3 \tag{37}$$

The final network,

$$\sigma(\boldsymbol{W}^{(3)}\sigma(\boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{p} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}) + b^3) \tag{38}$$

outputs a $+1$ for any $\boldsymbol{p} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ on the black regions and -1 otherwise.

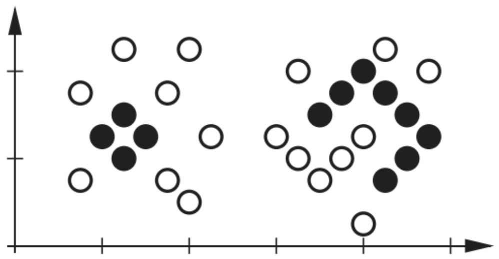

FIGURE 5. source: Neural Network Design [1]. Non linearly separable data. Illustration of example 4.1. (I)

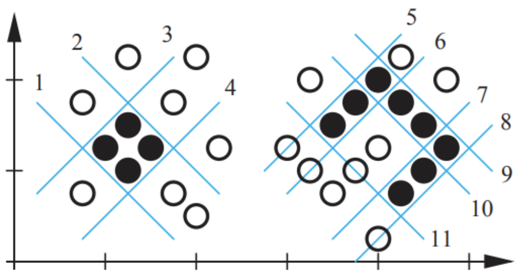

FIGURE 6. source: Neural Network Design [1]. Non linearly separable data and first layer of separating hyperplanes. Illustration of example 4.1. (II)

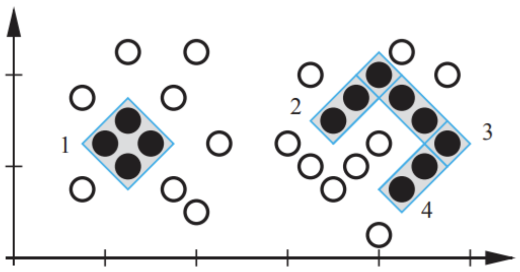

FIGURE 7. source: Neural Networks design [1]. Illustration of Example 4.1.

Exercise 10 (**source: NN design [1]). *Design multi-layer networks to perform the classifications illustrated in Fig. 8. The network should output a 1 whenever the input vector is in the shaded areas and −1 otherwise. Draw the network and give the weight matrices and bias vectors.*

References

[1] Martin T Hagan, Howard B Demuth, Mark H Beale, and Orlando De Jesús. *Neural network design*, volume 20. Pws Pub. Boston, 1996.

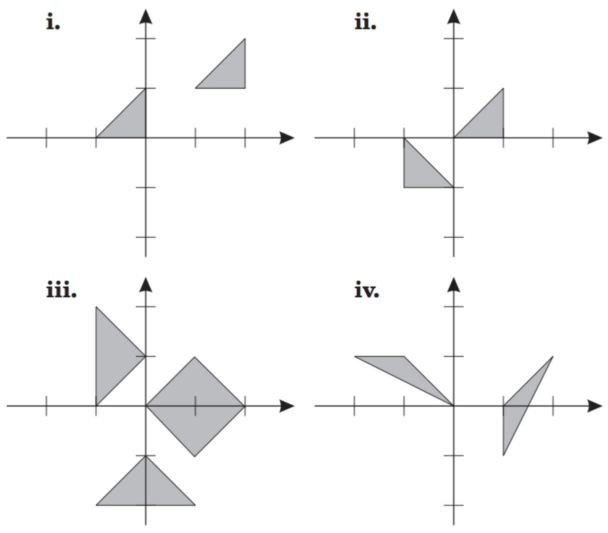

Figure 8. Material for exercise 10, the figure is taken from [1]