1. For SVR, prove that $w_0 = y_j - \epsilon - \sum_{i\in\mathcal{S}} \kappa(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathcal{S}$ is the set of indices of support vectors and $\mathbf{x}_j$ is a support vector at the *upper edge* of the ϵ-tube. (10 pts)

2. For the following classification problem, design a single-layer perceptron, by using the multiclass Perceptron update rule. (20 pts)

$$\mathcal{D}_{\omega_1} = \left\{ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$
$$\mathcal{D}_{\omega_2} = \left\{ \mathbf{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}$$
$$\mathcal{D}_{\omega_3} = \left\{ \mathbf{x}_3 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \right\}$$

Use one-hot coding for classes, for example, ω_3 should be represented using the following vector

$$\mathbf{y}_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

(a) Start with $\mathbf{W}(0) = \mathbf{0}_{2\times 3}$ and choose $\eta(i) = 0.5$ in $\mathbf{W}(i+1) = \mathbf{W}(i) + \eta(i)\mathbf{x}(i)\mathbf{e}^T$. Do not use the augmented space and assume that the biases are always zero (no update for biases). Showmultiple steps of your algorithm. Does it converge? Why? It is alright if you use a computer or calculator to perform the matrix calculations, but you should write down all the steps, and should not write a computer program to yield the final results.

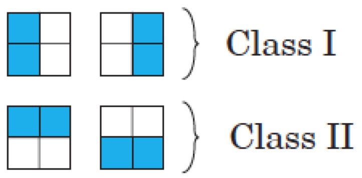
(b) Now redo the previous procedure, but this time deal with each of the columns of $\mathbf{W}$ as one perceptron, i.e. update each column (the weight associated with a linear discriminant) separately, for example the first iteration becomes:

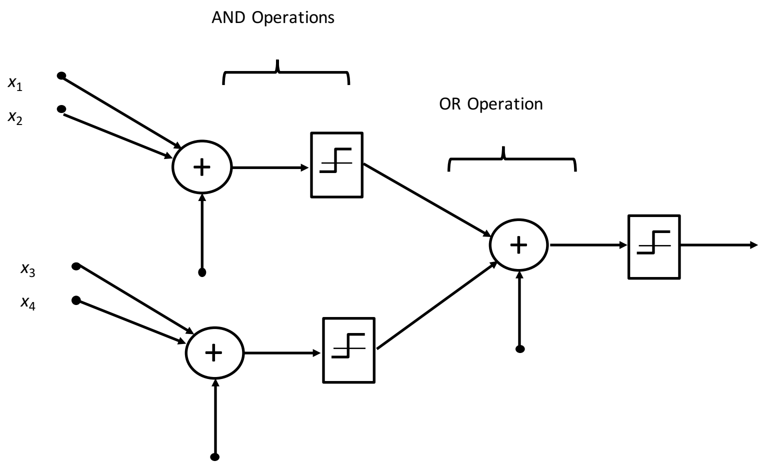
$$\mathbf{W}(0) = [\mathbf{w}_1(0)\ \mathbf{w}_2(0)\ \mathbf{w}_3(0)]$$
$$\mathbf{w}_1(1) = \mathbf{w}_1(0) + \eta e_1 \mathbf{x}(1)$$
$$\mathbf{w}_2(1) = \mathbf{w}_2(0) + \eta e_2 \mathbf{x}(1)$$
$$\mathbf{w}_3(1) = \mathbf{w}_3(0) + \eta e_3 \mathbf{x}(1)$$

where $e_i = y_{i1} - \text{sign}(\mathbf{w}_i(0)^T\mathbf{x}(1))$ is the difference between the i^{th} element of $\mathbf{y}_1$ (the target vector for $\mathbf{x}_1$) and the output of the i^{th} neuron/linear discriminant $\mathbf{w}_i(1)$.

Perform two epochs only. This is essentially to make you observe that a multicategory Perceptron algorithm is based on multiple binary problems.

3. Consider the two classes of patterns that are shown in the figure below. Design a multilayer neural network with the following architecture to distinguish these categories. (30 pts)





4. **Programming Assignment: Parkinsons Telemonitoring**

   (a) Download the Parkinsons Telemonitoring Data Set from: `http://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring`. Choose 70% of the data randomly as the training set.

   (b) Use metric learning with Gaussian kernels[1] to estimate each of the outputs motor UPDRS and total UPDRS from the features. As metric learning uses a low dimensional transformation of the features except the non-predictive feature subject#, use 5-fold cross-validation to decide the number of components form $M = 5, 10, 15, p$, where p is the number of all predictive features you can use. Initialize the linear transformation with PCA features for $M = 5, 10, 15$ and with original features for $M = p$. This corresponds to setting init as (default='auto'). Remember to standardize your features. Report the R^2 on training and test sets for each of the outputs. (30 pts)

   (c) Use sklearn's neural network implementation to train a neural network with two outputs that predicts motor UPDRS and total UPDRS. Use a single layer. You are responsible to determine other architectural parameters of the network, including the number of neurons in the hidden and output layers, method of optimization, type of activation functions, and the L2 "regularization" parameter etc. You should determine the design parameters via trial and error, by testing your trained network on the test set and choosing the architecture that yields the smallest test error. For this part, set early-stopping=False. Remember to standardize your features. Report your R^2 on both training and test sets. (20 pts)

[1] `http://contrib.scikit-learn.org/metric-learn/generated/metric_learn.MLKR.html#metric-learn-mlkr`

(d) Use the design parameters that you chose in the first part and train a neural network, but this time set early-stopping=True. Research what early stopping is, and compare the performance of your network on the test set with the previous network. You can leave the validation-fraction as the default (0.1) or change it to see whether you can obtain a better model. Remember to standardize your features. Report your R^2 on both training and test sets. (10 pts)

Note: there are a lot of design parameters in a neural network. If you are not sure how they work, just set them as the default of sklearn, but if you use them masterfully, you can have better models.

5. **Optional Programming Assignment: (Deep) CNNs for Image Colorization. This part will not be graded.**

(a) This assignment uses a convolutional neural network for image colorization which turns a grayscale image to a colored image.[2] By converting an image to grayscale, we loose color information, so converting a grayscale image back to a colored version is not an easy job. We will use the CIFAR-10 dataset. Downolad the dataset from `http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`.

(b) From the train and test dataset, extract the class birds. We will focus on this class, which has 6000 members.

(c) Those 6000 images have $6000 \times 32 \times 32$ pixels. Choose at least 10% of the pixels randomly. It is strongly recommended that you choose a large number or all of the pixels. You will have between $P = 614400$ and $P = 6144000$ pixels. Each pixel is an RGB vector with three elements.

(d) Run k-means clustering on the P vectors using $k = 4$. The centers of the clusters will be your main colors. Convert the colored images to k-color images by converting each pixel's value to the closest main color in terms of Euclidean distance. These are the outputs of your network, whose each pixel falls in one of those k classes.[3]

(e) Use any tool (e.g., openCV or scikit-learn) to obtain grayscale $32 \times 32 \times 1$ images from the original $32 \times 32 \times 3$ images. The grayscale images are inputs of your network.

(f) Set up a deep convolutional neural network with two convolution layers (or more) and two (or more) MLP layers. Use 5×5 filters and a softmax output layer. Determine the number of filters, strides, and whether or not to use padding yourself. Use a minimum of one max pooling layer. Use a classification scheme, which means your output must determine one of the $k = 4$ color classes for each pixel in

[2]MATLAB seems to have an easy to use CNN library. `https://www.mathworks.com/help/nnet/examples/train-a-convolutional-neural-network-for-regression.html`

[3]Centers of clusters have been reported too close previously, so the resultant tetra-chrome images will be very close to grayscale. In case you would like to see colorful images, repeat the exercise with colors you select from `https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/` or `https://www.rapidtables.com/web/color/RGB_Color.html`. A suggestion would be Navy = (0,0,128), Red =(230, 25, 75), Mint = (170, 255, 195), and White = (255, 255, 255).

your grayscale image. Your input is a grayscale version of an image ($32 \times 32 \times 1$) and the output is $32 \times 32 \times 4$. The output assigns one of the $k = 4$ colors to each of the 32×32 pixels; therefore, each of the pixels is classified into one of the classes [1 0 0 0], [0 1 0 0], [0 0 1 0], [0 0 0 1]. After each pixel is classified into one of the main colors, the RGB code of that color can be assigned to the pixel. For example, if the third *main color*[4] is [255 255 255] and pixel (32,32) of an image has the one-hot encoded class [0 0 1 0], i.e it was classified as the third color, the (32,32) place in the output can be associated with [255 255 255]. The size of the output of the convolutional part, $c_1 \times c_2$ depends on the size of the convolutional layers you choose and is a feature map, which is a matrix. That matrix must be *flattened* or *reshaped*, i.e. must be turned into a vector of size $c_1c_2 \times 1$, before it is fed to the MLP part. Choose the number of neurons in the first layer of the MLP (and any other hidden layers, if you are willing to have more than one hidden layer) yourself, but the last layer must have $32 \times 32 \times 4 = 4096$ neurons, each of which represents a pixel being in one of the $k = 4$ classes. Add a softmax layer[5] which will choose the highest value out of its $k = 4$ inputs for each of the 1024 pixels; therefore, the output of the MLP has to be reshaped into a $32 \times 32 \times 4$ matrix, and to get the colored image, the RGB vector of each of the $k = 4$ classes has to be converted to the RGB vector, so an output image will be $32 \times 32 \times 3$. Train at least for 5 epochs (30 epochs is strongly recommended). Plot training, (validation), and test errors in each epoch. Report the train and test errors and visually compare the artificially colored versions of the first 10 images in the test set with the original images.[6]

(g) Extra Practice: Repeat the whole exercise with $k = 16, 24, 32$ colors if your computer can handle the computations.

[4]Do not use the original CIFAR-10 images as the output. You must use the tetrachrome images you created as your output.

[5]Compile the network with `loss = cross_entropy` .

[6]If you are using matplotlib, you may get a floating point error because to print an image, matplotlib either expects ints in range 0-255 or floats in range 0-1. You might be having, for example, 153.0 representation of 153 in your array and this is what makes matplotlib think that you are sending floats.

Wrap your array into `np.uint8()`. It will convert 153.0 into 153. NOTE that you cannot use `np.round` or `np.int` etc because matplotlib's requirement is unsigned int of 8 bit (think 0-255).