1. (a) Word-sequence kernel function is the kernel function could deal with this task. Word-sequence kernel function is a upgrade version of string kernel that propose the use of sequences of words rather than characters. It inherits the advantage of string kernel that finite sequences of symbols that need not be of the same length. Thus, Word-sequence kernel could deal with the condition that each document is an arbitrary length sequence of words, taken from a vocabulary W.

The formula of Word-sequence kernel is:

$$\tilde{K}_n(X,Y) = \sum_{i=1}^{n} \mu^{1-i} \hat{K}_i(X,Y).$$

$$\phi_u(X) = \sum_{i:X[i]=u} \prod_{i_1 < j < i_n} \lambda_{x_j}$$

For computing the similarity between pairs of documents d_i and d_j, this kern function will find the more similar two sequences a and b are, the higher the value of a sequence kernel $K_n(a,b)$ will be. Also, this kernel function can convert the tuples of word counts into a sequence. Therefore, Word-sequence kernel function K_{ij} can computes the similariey between pairs of documents d_i and d_j.

Word-sequence Kernel

(b) We can use Multiple Kernel learning (function) to solve this problem. Multiple kernel learning uses a predefined set of kernels and learn an optimal linear or non-linear combination of kernels. So, for this case, we can make a combination of image kernel and string kernel to handle this problem.

The formula of Multible kernel is $K' = \sum_{i=1}^{n} \beta_i K_i$, where β is a vector of coefficients for each kernel.

image kernel:

$$K(X,Y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(X-s, Y-t)\ldots$$

String kernel:

$K(X,Y) = \varphi(x) \cdot \varphi(y)$ with φ mapping the arguments into an inner product space.

For this task, we may store the image kernel to deal with the image data, After this step, we can use the string kernel to handle the text data. Instead of creating a new kernel, multiple kernel algorithms can be used to combine kernels already established for each individual data source. So, multiple kernel function is the kernel function to solve this problem. To complete the similarity, multible kernel function will compare the similarity between two combinations of kernel functions. So, this should be a multible kernel function

Mutiple kernel function

(image kernel + string kernel)

2. the original Stochastic gradient descent function is

$$w_j \leftarrow -\lambda w_j + C(1-\lambda) y_i x_{ij} I[y_i x_i\, I[y_i(w \cdot x_i) + b < 1]$$

$$b \leftarrow b + C(1-\lambda) y_i I[y_i (w \cdot x_i) + b < 1]$$

For this case, we have two functions for each class

$$w_{j0} \leftarrow -\lambda w_{j0} + C_0 (1-\lambda) y_i x_{ij} I[y_i (w \cdot x_i) + b_0 < 1]$$

$$b_0 \leftarrow b_0 + C_0 (1-\lambda) y_i I[y_i (w \cdot x_i) + b_0 < 1]$$

$$w_{j1} \leftarrow -\lambda w_{j1} + C_1 (1-\lambda) y_i x_{ij} I(y_i (w \cdot x_i) + b_1 < 1]$$

$$b_1 \leftarrow b_1 + C_1 (1-\lambda) y_i I[y_i (w \cdot x_i) + b_0 < 1]$$

In original formula, both classes used same W and b, but right now the costs of misclassification of the two classes are unequal, that's why each class has their own function

To Balance the unequal cost, we can adjust bias and weight to make the margin becomes close. Since we have three different margin. (one for overall margin, one for C_0, and one for C_1). We can see the gap between each margin, and adjust the bias and weight to make the costs of misclassification for each case are close to each other. Finally, the cost will be same after many times of adjusting bias and weight, and we should use the overall margin to justify the cost. Above is how to make the costs of misclassification of two classes become equal.

Adjust the bias and weights separately for each class.

3. (a) The choice of the activation function Z_{jp} satisfy the requirments for universal function approximation theorm. $Z_{jp} = \frac{1}{\sigma^2 + h_{jp}^2}$ where σ is a constant and $h_{jp} = \sum_i w_{ji} x_{ip}$.

The requirments for UFAT is a (1) non-constant (2) non-linear (3) monotone (4) continuous function. For (1), Z_{jp} is clearly a non-constant because the value of Z_{jp} is always changing depending on σ and h_{jp}. For (2), obviously, Z_{jp} is not a linear function because there are σ^2 and h_{jp}^2. For (3), Z_{jp} is a monotone function as well because it depends on σ^2 and h_{jp}^2 that without change a trend of line immediately. For (4), This is obviously a continuous function. Thus, the activation function $Z_{jp} = \frac{1}{\sigma^2 + h_{jp}^2}$ satisfy all the requirments for UFAT

Yes, it satisfies the requirments of UFAT

(b) $E_a = \frac{1}{2} \sum_{p=1}^{P} (d_p - O_p)^2 = \frac{1}{2} \sum_{p=1}^{P} \left[d_p - \sum_j u_j \left(\frac{1}{\sigma^2 + (\sum_i w_{ji} x_{ip})^2} \right) \right]^2$

To update w_{ji}, we need the update equation for input to hidden units.

$$\frac{\partial E_p}{\partial w_{ji}} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial O_p} \frac{\partial O_p}{\partial w_{ji}} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial O_p} \frac{\partial O_p}{\partial Z_p} \cdot \frac{\partial Z_{jp}}{\partial h_{jp}} \cdot \frac{\partial h_{jp}}{\partial w_{ji}}$$

$$= \sum_{p=1}^{P} \frac{\partial}{\partial O_p} \left[\frac{1}{2} \sum_{l=1}^{P} (d_{lp} - O_{lp})^2 \right] (u_j)(Z_{jp})(1 - Z_{jp})(x_{ip})$$

$$= - \sum_{p=1}^{P} (d_p - O_p)(u_j)(Z_{jp})(1 - Z_{jp})(x_{ip})$$

$$= - \underbrace{\left(\sum_{p=1}^{P} \delta_p (u_j) \left(\frac{1}{\sigma^2 + (\sum_i w_{ji} x_{ip})^2} \right) \left(1 - \frac{1}{\sigma^2 + (\sum_i w_{ji} x_{ip})^2} \right) \right.}_{\delta_{jp}} (x_{ip})$$

$$= - \delta_{jp} x_{ip} \qquad w_{ji} \leftarrow w_{ji} + \eta \delta_{jp} x_{ip}$$

Above is the update equation for w_{ji} that minimize E_a.

Now see the update equation for u_j that Hidden-to-output

$$\frac{\partial E_p}{\partial u_j} = \frac{\partial E_p}{\partial h_{jp}} \cdot \frac{\partial h_{jp}}{\partial u_j} \qquad \frac{\partial h_{jp}}{\partial u_j} = Z_{jp}$$

$$\frac{\partial E_p}{\partial h_{jp}} = \frac{\partial E_p}{\partial O_p} \cdot \frac{\partial O_p}{\partial h_{jp}} = -(d_p - O_p)(1)$$

$$u_j \leftarrow u_j - \eta \frac{\partial E_p}{\partial u_j} = u_j + (d_p - O_p) Z_{jp} = u_j + \delta_p Z_{jp}$$

$\boxed{u_j \leftarrow u_j + \delta_p \left(\frac{1}{\sigma^2 (\Sigma_i w_{ji} x_{ip})^2} \right)}$, where δ can be found from the Previous part that update equation for w_{ji}.

Above is the update equation for u_j that minimize E_a.

4. (a) Since $E_b = \sum_{p=1}^{P} \sum_{i=0}^{N} \left(\frac{\partial E_a}{\partial X_{ip}}\right)^2$, the error function $E = \lambda E_a + (1-\lambda) E_b$ where $0 \leq \lambda \leq 1$ will never be decreased. To the impact of minimizing E_b on the sensitivity of the network output to relatively small amounts of noise in the input sample, the sensitivity to the noise will be small, which means the network output is insensitive to noise because E_b makes the weight is not that accurate for real results. In other words, E_b makes the increase of the robustness of this neural network. Overall, the sensitivity to small amounts of noise is small because of the impact of E_b.

the tendency of the network to over-fit the training data will be decreased (slow) as well because of a similar reason that E_b make the weight is not that accurate for desired output. In other words, it generalized the capability of the network, and it slows down the tendency of the network to over-fit the training data (avoid the risk of over-fitting)

Above are the impact of minimizing E_b on the sensitivity of the network and the tendency of the network to over-fit the training data.

(b) $E = \lambda E_a + (1-\lambda) E_b$ where $0 \leq \lambda \leq 1$.

$$E_a = \frac{1}{2} \sum_{p=1}^{P} (d_p - O_p)^2 = \frac{1}{2} \sum_{p=1}^{P} \left[d_p - \Sigma_j u_j \frac{1}{(6^2 + (\Sigma_i W_{ji} X_{ip})^2)}\right]^2$$

$$E_b = \sum_{p=1}^{P} \sum_{i=0}^{N} \left(\frac{\partial E_a}{\partial X_{ip}}\right)^2$$

We will calculate the error function separately. For E_a, we have already derive the update equations in 3(b), so, here only compute the update equation for E_b.

$$\frac{\partial E_a}{\partial X_{ip}} = \sum_{p=1}^{P} \frac{(d_p - O_p)}{\partial X_{ip}} = \sum_{p=1}^{P} \left[d_p - \Sigma_j u_j \frac{1}{(6^2 + (\Sigma_i W_{ji} X_{ip})^2}\right]$$

$$E_b = \sum_{p=1}^{P} \sum_{i=0}^{N} \left[d_p - \Sigma_j u_j \frac{1}{(6^2 + (\Sigma_i W_{ji} X_{ip})^2}\right]^2$$

$$\frac{\partial E_b}{\partial x_n} = -2\lambda\left[d_p - \Sigma_j u_j \frac{1}{(\sigma^2 + \Sigma_i (w_{ji}))}\right]$$

Thus, $w_{ji} \leftarrow w_{ji} + \eta y_p\, z_n - 2\lambda\left[d_p - \Sigma_j u_j \frac{1}{(\sigma^2 + \Sigma_i (w_{ji}))}\right]$

and u_j is

$$u_j \leftarrow u_j - \eta\frac{\partial E_b}{\partial u_j} = u_j - 2\lambda\left[d_p - \Sigma_j u_j \frac{1}{\sigma^2 + \Sigma_i (w_{ji})}\right]$$

Above is the update equations for the parameters w_{ji} and u_j for E_b.

To minimize E, we can simplely add the update equations of E_a and E_b togther, and calculate the final error. However, the logic behind this is very similar to the 3(b), the update equations of E_a.

5.(a) Since the samples are approximately linearly separable, we can use perceptron learning algorithm to classify this task. Perceptron is an algorithm that attempts to fix all errors encountered in the training set, and divided the data into two-classes. For the data that is non-linear separable, we can use linear kernel to make them become linear-separable. However, I think Perceptron learning algorithm is the best choice for this task.

(b) Naive-Bayes classifier is the best choice for this task. Since the features are likely to be independent, Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorm with strong independence assumptions between the features. Thus, NB fits this requirment. Additionally, the dataset has a limited number of training examples, however NB can deal with this situation too.

Naive Bayes

(c) I recommend neural network is the best algorithm for this task. First, neural network can do that the features are numeric, second, the robustness of neural work is good, and introduce small amounts of noise in the weight update, can help neural network reduces the risk of over-fitting. Thus, I recommend neural network for this task.

neural network

(d) To solve this problem, we can chose the combination of sparse regularization, universal function approximation theorem, and, finally, neural network as our learning algorthim. Since the number of features far exceeds the number of training example, we should make the features become sparsed regularized, and select a couple of the most important features based on our prior knowledge of features. After this, the number of features becomes small, and we can use UFAT to compute all the eligible functions into neural network classifier. Thus, we will use neural network to do normal learning since we have already know prior thoughts to guide the selection of features to be used to train a neural network function approximator.

Sparse regularization + UFAT + Neural Network

(e) I recommend neural Network to do this task. First, we can divided the data into three subgroups based on their types (namely, ~~text~~, images). Since neural network will have a weight for each input, thus, we donot need to worry about the weight problem. Finally, neural networks will combine them into a Net input function, so neural network is the algorithm to handle the composed of three types of data.

Neural Network