CS 189 Spring 2014 Introduction to Machine Learning Final

- You have 3 hours for the exam.
- The exam is closed book, closed notes except your one-page crib sheet.
- Please use non-programmable calculators only.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation.
- For true/false questions, fill in the *True/False* bubble.
- For multiple-choice questions, fill in the bubbles for **ALL** CORRECT CHOICES (in some cases, there may be more than one). We have introduced a negative penalty for false positives for the multiple choice questions such that the expected value of randomly guessing is 0. Don't worry, for this section, your score will be the maximum of your score and 0, thus you cannot incur a negative score for this section.

First name	
Last name	
SID	
First and last name of student to your left	
First and last name of student to your right	

For staff use only:

Q1.	True or False	/9
Q2.	Multiple Choice	/24
Q3.	Softmax regression	/11
Q4.	PCA and least squares	/10
Q5.	Mixture of linear regressions	/10
Q6.	Training set augmentation	/10
Q7.	Kernel PCA	/12
Q8.	Autoencoder	/14
	Total	/100

Q1. [9 pts] True or False

(a) [1 pt] The singular value decomposition of a real matrix is unique.

○ True ● False

(b) [1 pt] A multiple-layer neural network with linear activation functions is equivalent to one single-layer perceptron that uses the same error function on the output layer and has the same number of inputs.

● True ○ False

(c) [1 pt] The maximum likelihood estimator for the parameter θ of a uniform distribution over $[0, \theta]$ is unbiased.

○ True ● False

(d) [1 pt] The k-means algorithm for clustering is guaranteed to converge to a local optimum.

● True ○ False

(e) [1 pt] Increasing the depth of a decision tree cannot increase its training error.

● True ○ False

(f) [1 pt] There exists a one-to-one feature mapping ϕ for every valid kernel k.

○ True ● False

(g) [1 pt] For high-dimensional data, k-d trees can be slower than brute force nearest neighbor search.

● True ○ False

(h) [1 pt] If we had infinite data and infinitely fast computers, kNN would be the only algorithm we would study in CS 189.

● True ○ False

(i) [1 pt] For datasets with high label noise (many data points with incorrect labels), random forests would generally perform better than boosted decision trees.

● True ○ False

Q2. [24 pts] Multiple Choice

(a) [2 pts] In Homework 4, you fit a logistic regression model on spam and ham data for a Kaggle Competition. Assume you had a very good score on the public test set, but when the GSIs ran your model on a private test set, your score dropped a lot. This is likely because you overfitted by submitting multiple times and changing the following between submissions:

- ● λ, your penalty term
- ● ϵ, your convergence criterion
- ● η, your step size
- ● Fixing a random bug

(b) [2 pts] Given d-dimensional data $\{\mathbf{x}_i\}_{i=1}^N$, you run principle component analysis and pick P principle components. Can you always reconstruct any data point $\mathbf{x}_i$ for $i \in \{1 \dots N\}$ from the P principle components with zero reconstruction error?

- ○ Yes, if $P < d$
- ● Yes, if $P = d$
- ○ Yes, if $P < n$
- ○ No, always

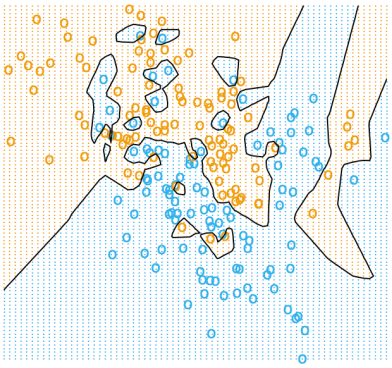
(c) [2 pts] Putting a standard Gaussian prior on the weights for linear regression ($\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$) will result in what type of posterior distribution on the weights?

- ○ Laplace
- ○ Uniform
- ○ Poisson
- ● None of the above

(d) [2 pts] Suppose we have N instances of d-dimensional data. Let h be the amount of data storage necessary for a histogram with a fixed number of ticks per axis, and let k be the amount of data storage necessary for kernel density estimation. Which of the following is true about h and k?

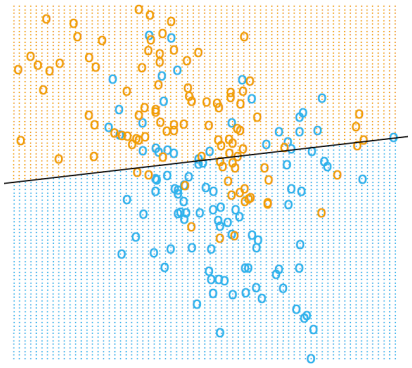
- ○ h and k grow linearly with N
- ● h grows exponentially with d, and k grows linearly N
- ○ h and k grow exponentially with d
- ○ h grows linearly with N, and k grows exponentially with d

(e) [2 pts] Which of the these classifiers could have generated this decision boundary?



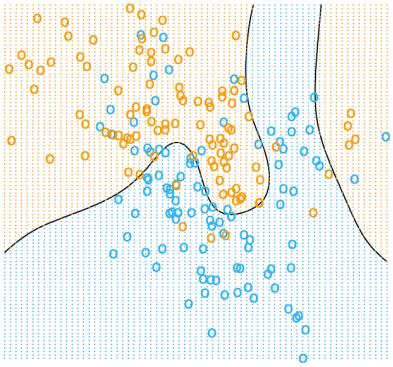
- ○ Linear SVM
- ● 1-NN
- ○ Logistic regression
- ○ None of the above

(f) [2 pts] Which of the these classifiers could have generated this decision boundary?



- [x] Linear SVM
- [ ] 1-NN
- [x] Logistic regression
- [ ] None of the above

(g) [2 pts] Which of the these classifiers could have generated this decision boundary?



- [ ] Linear SVM
- [ ] 1-NN
- [ ] Logistic regression
- [x] None of the above

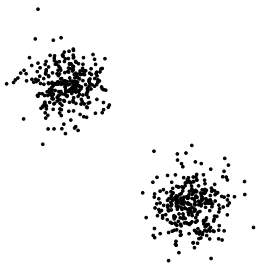
(h) [2 pts] You want to cluster this data into 2 clusters. Which of the these algorithms would work well?

- [ ] K-means
- [x] GMM clustering
- [ ] Mean shift clustering

(i) [2 pts] You want to cluster this data into 2 clusters. Which of the these algorithms would work well?

○ K-means ○ GMM clustering ● Mean shift clustering

(j) [2 pts] You want to cluster this data into 2 clusters. Which of the these algorithms would work well?



● K-means ● GMM clustering ● Mean shift clustering

The following questions are about how to help CS 189 TA Jonathan Snow to solve the homework.

(k) [2 pts] Jonathan just trained a decision tree for a digit recognition. He notices an extremely low training error, but an abnormally large test error. He also notices that an SVM with a linear kernel performs much better than his tree. What could be the cause of his problem?

● Decision tree is too deep

● Decision tree is overfitting

○ Learning rate too high

○ There is too much training data

(l) [2 pts] Jonathan has now switched to multilayer neural networks and notices that the training error is going down and converges to a local minimum. Then when he tests on the new data, the test error is abnormally high. What is probably going wrong and what do you recommend him to do?

● The training data size is not large enough. Collect a larger training data and retrain it.

● Play with learning rate and add regularization term to the objective function.

● Use a different initialization and train the network several times. Use the average of predictions from all nets to predict test data.

○ Use the same training data but add two more hidden layers.

Q3. [11 pts] Softmax regression

Recall the setup of logistic regression: We assume that the posterior probability is of the form

$$p(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^\top \mathbf{x}}}$$

This assumes that $Y|\mathbf{X}$ is a Bernoulli random variable. We now turn to the case where $Y|\mathbf{X}$ is a multinomial random variable over K outcomes. This is called softmax regression, because the posterior probability is of the form

$$p(Y = k|\mathbf{x}) = \mu_k(\mathbf{x}) = \frac{e^{\boldsymbol{\beta}_k^\top \mathbf{x}}}{\sum_{j=1}^K e^{\boldsymbol{\beta}_j^\top \mathbf{x}}}$$

which is called the softmax function. Assume we have observed data $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$. Our goal is to learn the weight vectors $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$.

(a) [3 pts] Find the negative log likelihood of the data $l(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K)$.

$$-\log \mathbb{P}(Y|X) = -\log \prod_{i=1}^N \mathbb{P}(y_i|x_i) = -\log \prod_{i=1}^N \prod_{k=1}^K \left(\frac{e^{\beta_k^T x_i}}{\sum_{j=1}^K e^{\beta_j^T x_i}} \right)^{1\{y_i = k\}}$$

$$= -\sum_{i=1}^N \sum_{k=1}^K 1\{y_i = k\} \left(\beta_k^T x_i - \log \left(\sum_{j=1}^K e^{\beta_j^T x_i} \right) \right)$$

$$= -\sum_{i=1}^N \sum_{k=1}^K 1\{y_i = k\} \beta_k^T x_i + \sum_{i=1}^N \log \left(\sum_{j=1}^K e^{\beta_j^T x_i} \right)$$

(b) [2 pts] We want to minimize the negative log likelihood. To combat overfitting, we put a regularizer on the objective function. Find the gradient w.r.t. $\boldsymbol{\beta}_k$ of the regularized objective

$$l(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K) + \lambda \sum_{k=1}^K \|\boldsymbol{\beta}_k\|^2$$

$$\nabla_{\beta_k} - \log \mathbb{P}(Y|X) = 2\lambda \beta_k - \sum_{i=1}^N 1\{y_i = k\} x_i + \sum_{i=1}^N \frac{e^{\beta_k^T x_i}}{\sum_{j=1}^K e^{\beta_j^T x_i}} x_i$$

Note that we can use the definition of $\mu_k(x_i)$ here to save a bunch of writing.

$$= 2\lambda \beta_k + \sum_{i=1}^N \left(\mu_k(x_i) - 1\{y_i = k\} \right) x_i$$

(c) [4 pts] State the gradient updates for both batch gradient descent and stochastic gradient descent.

Batch gradient descent:

$$\beta_k^{(t+1)} = \beta_k^{(t)} - \eta \left(2\lambda \beta_k^{(t)} + \sum_{i=1}^N \left(\mu_k(x_i) - 1\{y_i = k\} \right) x_i \right)$$

Stochastic gradient descent:

$$\beta_k^{(t+1)} = \beta_k^{(t)} - \eta \left(2\lambda \beta_k^{(t)} + \left(\mu_k(x_i) - 1\{y_i = k\} \right) x_i \right)$$

(d) [2 pts] There are times when we'd like to consider the multiclass case to be a 1-vs.-all scenario with K binary classifiers, and there are times when we'd like to attack the multiclass case with a multiclass classifier such as softmax regression.

When would you want to use a softmax regression as opposed to K 1-vs.-all logistic regressions?

- ● When the classes are mutually exclusive
- ○ When the classes are not linearly separable
- ○ When the classes are not mutually exclusive
- ○ Both work equally well

Q4. [10 pts] PCA and least squares

Recall that PCA transforms (zero-mean) data into low-dimensional reconstructions that lie in the span of the top k eigenvectors of the sample covariance matrix. Let $\mathbf{U}_k$ denote the $d \times k$ matrix of the top k eigenvectors of the covariance matrix ($\mathbf{U}_k$ is a truncated version of $\mathbf{U}$, which is the matrix of eigenvectors of the covariance matrix).

There are two approaches to computing the low-dimensional reconstruction $\mathbf{w} \in \mathbb{R}^k$ of a data point $\mathbf{x} \in \mathbb{R}^d$:

1. Solve a least squares problem to minimize the reconstruction error
2. Project $\mathbf{x}$ onto the span of the columns of $\mathbf{U}_k$

In this problem, you will show that these approaches are equivalent.

(a) [5 pts] Formulate the least squares problem in terms of $\mathbf{U}_k, \mathbf{x}$, and the variable $\mathbf{w}$.

(Hint: This optimization problem should resemble linear regression.)

We want to find the weights such that when we weight the columns of U_k, we will minimize the residual error. Thus, the objective function is $\|U_k w - x_i\|^2$. Here is our least squares problem:

$$\min_w \|U_k w - x_i\|^2$$

(b) [5 pts] Show that the solution of the least squares problem is equal to $\mathbf{U}_k^\top \mathbf{x}$, which is the projection of $\mathbf{x}$ onto the span of the columns of $\mathbf{U}_k$.

Recall the normal equations (which is the most important equation in the class!). For some unconstrained least squares problem in the form

$$\min_x \|Ax - y\|^2$$

the solution of the minimizer is

$$x^* = (A^T A)^{-1} A^T y$$

only when A is full rank. This applies here since U_k is full rank by definition. Plugging in our least squares problem, we have

$$w^* = (U_k^T U_k)^{-1} U_k^T x_i$$

If you don't remember this you can easily derive this by taking the gradient of the objective function and setting it to 0. We note that $U_k^T U_k = I_k$, (I_k = the identity matrix in k dimensions) thus

$$w^* = (U_k^T U_k)^{-1} U_k^T x_i = (I_k)^{-1} U_k^T x_i = U_k^T x_i$$

since the inverse of I is simply I.

Q5. [10 pts] Mixture of linear regressions

In class, you learned that K-means partitions points into K clusters by minimizing an objective that encourages points to be close to their cluster centers. K-means minimizes this objective in a coordinate descent fashion, alternating between computing cluster assignments and cluster centers until convergence.

In this problem, you will devise an algorithm, in the spirit of K-means, that fits an entirely different kind of mixture model. You are given a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. You know that this dataset is a mixture of realizations of K different linear regression models

$$y = \mathbf{w}_k^\top \mathbf{x} + \mathcal{N}(0, \sigma^2)$$

parameterized by K weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_K \in \mathbb{R}^d$.

Your algorithm will jointly determine the following:

- A partition $S_1, \ldots, S_K$ of the dataset such that $(\mathbf{x}_i, y_i) \in S_k$ iff $(\mathbf{x}_i, y_i)$ comes from model k
- The model weights $\mathbf{w}_1, \ldots, \mathbf{w}_K$

(a) [4 pts] Write an objective function $f(S_1, \ldots, S_K, \mathbf{w}_1, \ldots, \mathbf{w}_K)$ to be minimized to solve this problem. Use the penalty $\|\mathbf{w}_k^\top \mathbf{x} - y\|^2$ if the point $(\mathbf{x}, y)$ is assigned to model k. Your objective should be a sum of N terms, and each data point should show up in exactly one of these terms.

$$f(S_1, \ldots, S_K, w_1, \ldots, w_K) = \sum_{k=1}^{K} \sum_{(x,y) \in S_k} \|w_k^\top x - y\|^2$$

(b) [3 pts] What is coordinate descent update for f with $\mathbf{w}_1, \ldots, \mathbf{w}_K$ fixed? In other words, to which of the K models should a point $(\mathbf{x}, y)$ be assigned?

Assign the point (x, y) to S_k if $k = \arg\min_k \|w_k^\top x - y\|^2$.

(c) [3 pts] Write the coordinate descent update for f with $S_1, \ldots, S_K$ fixed.

For a set S of $(\mathbf{x}, y)$ values, you should use the notation $\mathbf{X}_S$ to denote the design matrix whose rows are the $\mathbf{x}$-values of the elements of S, and $\mathbf{y}_S$ to denote the column vector of y-values of the elements of S.

$w_k = (X_{S_k}^\top X_{S_k})^{-1} X_{S_k}^\top y_{S_k}$

Q6. [10 pts] Training set augmentation

In class, you learned that one way to encourage invariance of a model to certain transformations is to augment the training set with extra examples perturbed according to those transformations. In this problem, you will examine the behavior of a certain type of input perturbation for a probabilistic linear regression setting.

Consider the following general generative model for regression:

- $\mathbf{x} \sim p(\mathbf{x})$, where $p(\mathbf{x})$ is a distribution over input vectors $\mathbf{x} \in \mathbb{R}^d$
- $y|\mathbf{x} \sim p(y|\mathbf{x})$, where $p(y|\mathbf{x})$ is a distribution over output scalars $y \in \mathbb{R}$ given $\mathbf{x}$

Assume that the relationship between y and $\mathbf{x}$ is well-modeled by a linear function $y = \mathbf{w}^\top \mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^d$, so that in the infinite dataset limit, the objective to be minimized for this regression problem is:

$$\mathcal{L}_0(\mathbf{w}) = \mathbb{E}\left[(\mathbf{w}^\top \mathbf{x} - y)^2\right]$$

Now suppose the inputs are perturbed by zero-mean Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \lambda \mathbf{I})$, which is independent of the training data. The new objective is

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}\left[(\mathbf{w}^\top (\mathbf{x} + \boldsymbol{\epsilon}) - y)^2\right]$$

(a) [9 pts] Compute and simplify $\mathcal{L}(\mathbf{w})$. Show all your work in detail, and write your answer in terms of $\mathcal{L}_0$.

$$\begin{aligned}
\mathcal{L}(w) &= \mathbb{E}\left[(w^\top (x + \epsilon) - y)^2\right] \\
&= \mathbb{E}\left[((w^\top x - y) + w^\top \epsilon)^2\right] \\
&= \mathbb{E}\left[(w^\top x - y)^2 + 2(w^\top x - y) w^\top \epsilon + w^\top \epsilon \epsilon^\top w\right] \\
&= \mathbb{E}\left[(w^\top x - y)^2\right] + \mathbb{E}\left[2(w^\top x - y) w^\top\right] \mathbb{E}[\epsilon] + w^\top \mathbb{E}\left[\epsilon \epsilon^\top\right] w
\end{aligned}$$

Substituting $\mathbb{E}[\epsilon] = 0$ and $\mathbb{E}[\epsilon \epsilon^\top] = \lambda I$, this simplifies to

$$= \mathcal{L}_0(w) + \lambda \|w\|^2$$

(b) [1 pt] Is there a relationship between this particular type of input perturbation and some type of regularization? If so, what kind of regularizer is involved?

In this setting, regression assuming this type of input perturbation turns out to be equivalent to regression with an L_2 regularizer.

Q7. [12 pts] Kernel PCA

You are given d-dimensional real-valued data $\{\mathbf{x}_i\}_{i=1}^N$ and a feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^m$. In the following questions, you will investigate how to do PCA in feature space on the feature vectors $\{\phi(\mathbf{x}_i)\}_{i=1}^N$. Assume that the data is centered in feature space; that is, $\sum_{i=1}^N \phi(\mathbf{x}_i) = 0$.

In the following, $\mathbf{\Phi}$ is a design matrix whose i^{th} row is $\phi(\mathbf{x}_i)$.

(a) [1 pt] Recall that as a part of PCA, we must solve the eigenvalue problem $\mathbf{Sv} = \lambda\mathbf{v}$, where $\mathbf{S}$ proportional to the sample covariance matrix. For PCA in feature space, we have $\mathbf{S} = \sum_{i=1}^N \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top = \mathbf{\Phi}^\top\mathbf{\Phi}$. Why is this a problem if m is large?

Working in feature space directly is too expensive if m is large. The covariance matrix $\Phi^\top\Phi$ is $m \times m$, which is too large to compute and work with.

(b) [3 pts] Now, you are given a kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$. Define the kernel matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

Show that if $\lambda \neq 0$, then λ is an eigenvalue of $\mathbf{S}$ if and only if λ is also an eigenvalue of $\mathbf{K}$ (in other words, finding feature-space principal components can be done by finding eigenvectors of $\mathbf{K}$).

Notice that $K = \Phi\Phi^\top$. The nonzero eigenvalues of $K = \Phi\Phi^\top$ and $S = \Phi^\top\Phi$ are the same.

(c) [4 pts] Let $\mathbf{v}$ be an eigenvector of $\mathbf{S}$ with nonzero eigenvalue λ. Show that $\mathbf{v}$ can be written as $\mathbf{v} = \mathbf{\Phi}^\top\boldsymbol{\alpha}_\mathbf{v}$, where $\boldsymbol{\alpha}_\mathbf{v}$ is an eigenvector of $\mathbf{K}$ with eigenvalue λ.

First, write $Sv = \Phi^\top\Phi v = \lambda v$. Multiplying this equation by Φ gives $K\alpha = \lambda\alpha$, where $\alpha = \Phi v$. Then, since $\Phi^\top\alpha = \Phi^\top\Phi v = Sv = \lambda v$, we have $v = \Phi^\top\alpha/\lambda$. Choosing $\alpha_v = \alpha/\lambda$ gives the desired result.

(d) [4 pts] You are given a new data point $\mathbf{x} \in \mathbb{R}^d$. Find the scalar projection of its feature representation $\phi(\mathbf{x})$ onto $\mathbf{v}/\|\mathbf{v}\|$ (with $\mathbf{v}$ defined as above).

Write your answer in terms of $\boldsymbol{\alpha}_\mathbf{v}$ and λ. Use the kernel k, and do not explicitly use ϕ. You should use the notation $\mathbf{k}_\mathbf{x} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}) & \cdots & k(\mathbf{x}_n, \mathbf{x}) \end{bmatrix}^\top$.

First, let's calculate the squared norm of v:

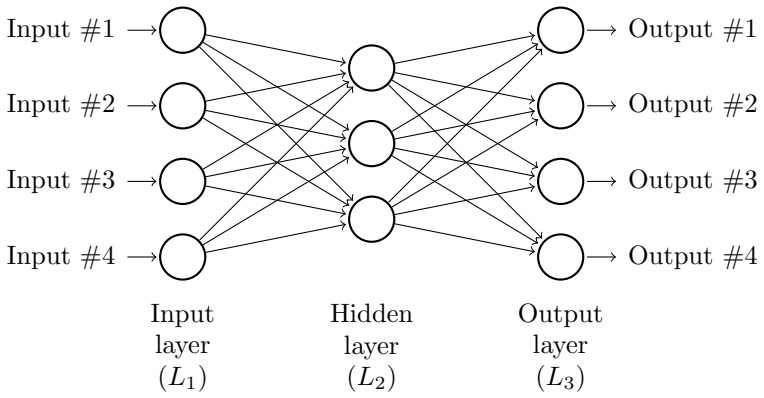
$$\|v\|^2 = v^\top v = \alpha_v^\top \Phi\Phi^\top \alpha_v = \alpha_v^\top K \alpha_v = \lambda\|\alpha_v\|^2$$

Then, the projection is:

$$\frac{v^\top\phi(x)}{\|v\|} = \frac{\alpha_v^\top\Phi\phi(x)}{\sqrt{\lambda}\|\alpha_v\|} = \frac{\alpha_v^\top k_x}{\sqrt{\lambda}\|\alpha_v\|}$$

Q8. [14 pts] Autoencoder

An autoencoder is a neural network designed to learn feature representations in an unsupervised manner. Unlike a standard multi-layer network, an autoencoder has the same number of nodes in its output layer as its input layer. An autoencoder is not trained to predict some target value y given input $\mathbf{x}$; rather, it is trained to reconstruct its own input $\mathbf{x}$, i.e. to minimize the reconstruction error. An autoencoder is shown below.



Suppose the input is a set of P-dimensional unlabeled data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$. Consider an autoencoder with H hidden units in L_2. We will use the following notation for this autoencoder:

- $\mathbf{W}^e$ denotes the $P \times H$ weight matrix between L_1 and L_2
- $\mathbf{W}^d$ denotes the $H \times P$ weight matrix between L_2 and L_3
- σ denotes the activation function for L_2 and L_3
- $s_j^{(i)} = \sum_{k=1}^{P} W_{kj}^e x_k^{(i)}$
- $z_j^{(i)} = \sigma\left(\sum_{k=1}^{P} W_{kj}^e x_k^{(i)}\right)$
- $t_j^{(i)} = \sum_{k=1}^{H} W_{kj}^d z_k^{(i)}$
- $\hat{x}_j^{(i)} = \sigma\left(\sum_{k=1}^{H} W_{kj}^d z_k^{(i)}\right)$
- $J(\mathbf{W}^e, \mathbf{W}^d)^{(i)} = \|\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}\|_2^2 = \sum_{j=1}^{P} (x_j^{(i)} - \hat{x}_j^{(i)})^2$ is the reconstruction error for example $\mathbf{x}^{(i)}$
- $J(\mathbf{W}^e, \mathbf{W}^d) = \sum_{i=1}^{N} J(\mathbf{W}^e, \mathbf{W}^d)^{(i)}$ is the total reconstruction error

(We add element 1 to the input layer and hidden layer so that no bias term has to be considered.)

(a) [8 pts] Fill in the following derivative equations for $\mathbf{W}^e$ and $\mathbf{W}^d$. Use the notation defined above; there should be no new notation needed.

$$\frac{\partial J^{(i)}}{\partial W^d_{kl}} = \sum_{j=1}^{P} \left(\boxed{2(\hat{x}^{(i)}_j - x^{(i)}_j)} \cdot \frac{\partial \hat{x}^{(i)}_j}{\partial W^d_{kl}} \right)$$

$$\frac{\partial \hat{x}^{(i)}_j}{\partial W^d_{kl}} = \sigma' \left(\sum_{k=1}^{P} W^e_{kj} x^{(i)}_k \right) \cdot \boxed{z^{(i)}_k}$$

$$\frac{\partial J^{(i)}}{\partial W^e_{kl}} = \frac{\partial J^{(i)}}{\partial s^{(i)}_j} \cdot \boxed{\frac{\partial s^{(i)}_j}{\partial W^e_{kl}} = x^{(i)}_k}$$

$$\frac{\partial J^{(i)}}{\partial s^{(i)}_j} = \sum_{k=1}^{H} \left(\frac{\partial J^{(i)}}{\partial t^{(i)}_k} \cdot \boxed{W^d_{jk}} \cdot \sigma'(s^{(i)}_j) \right)$$

(b) [4 pts] To limit the number of activated hidden units, we add a sparsity penalty to the problem. The reconstruction error is formulated as

$$J_{sparse}(\mathbf{W}^e, \mathbf{W}^d) = J(\mathbf{W}^e, \mathbf{W}^d) + \beta \sum_{j=1}^{H} \mathrm{KL}(\rho \| \hat{\rho}_j)$$

where $\hat{\rho}_j = \frac{1}{N} \sum_{i=1}^{N} z^{(i)}_j$, and ρ and β are hyperparameters. KL divergence is defined as

$$\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \left(\frac{\rho}{\hat{\rho}_j} \right) + (1 - \rho) \log \left(\frac{1 - \rho}{1 - \hat{\rho}_j} \right)$$

Write the following derivative updates for $\mathbf{W}^e$ and $\mathbf{W}^d$.

$$\frac{\partial J_{sparse}}{\partial W^d_{kl}} = \frac{\partial J}{\partial W^d_{kl}} + \boxed{0}$$

$$\frac{\partial J_{sparse}}{\partial W^e_{kl}} = \frac{\partial J}{\partial W^e_{kl}} + \beta \cdot \sum_{j=1}^{H} \boxed{\left(-\frac{\rho}{\rho_j} + \frac{1-\rho}{1-\rho_j}\right) \cdot \frac{1}{N} \sum_{i=1}^{N} \left(x^{(i)}_k \sigma' \left(\sum_{s=1}^{P} W^e_{sj} x^{(i)}_k\right)\right)}$$

(c) [2 pts] State some relations between autoencoders and PCA.

They are both feature representation learning methods. PCA is only linear transformation to the subspace while autoencoder is nonlinear transformation to the hidden units. If the autoencoder's activation functions are linear, it is very similar to PCA method.