ECE 595: Homework 2

Rahul Deshmukh, PUID: 0030004932

(Fall 2019)

1 Problem 1: Definitions

(a) Regularization is defined as any modification to the learning algorithm intended to reduce the generalization error but not the training error.

(b) Why Regularization : The goal of regularization is to take a model from the over-fitting regime to a true data-generating process regime.

(c) One possible way of regularization can be grouped sparsity where instead of taking a norm penalty on the entire set of weights we can rather regularize only a group of features, with some prior knowledge to make the groups. The regularization function will be:

$$\Omega(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}_g||_2$$

This method is a generalization of sparse regularizer on columns which will be discussed in Problem 4(c)

2 Problem 2: L^2 Regularization

Given:

$$\begin{aligned} \tilde{J}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) &= \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) \\ \nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) &= \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) \\ &\Rightarrow \boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) \end{aligned}$$

(a) Quadratic Approximation: We assume a quadratic approximation of the un-regularized loss in the neighborhood of $\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}}\, J(\boldsymbol{w})$ given by:

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^T \cancelto{0}{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*)} + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*)$$

Where $\boldsymbol{H}$ is the hessian of the un-regularized objective function evaluated at $\boldsymbol{w}^*$ and is positive-semi-definite matrix. Therefore the gradient of the regularized objective function will be given by:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}) = 0$$

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^*) + \alpha \boldsymbol{w}$$

$$= \alpha \boldsymbol{w} + \cancelto{0}{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*)} + \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) = 0$$

$$\Rightarrow \boldsymbol{w} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H} \boldsymbol{w}^*$$

Now since $\mathbf{H}\boldsymbol{H}$ is positive semi-definite, ie it can be decomposed to: $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$

Where $\boldsymbol{Q}$ is orthogonal matrix ie $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$

$$\Rightarrow \boldsymbol{w} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T + \alpha \boldsymbol{I})^{-1} \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^*$$

$$= (\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{Q}^T)^{-1} \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^*$$

$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^* \tag{1}$$

From the above solution we can interpret that the optimum solution is a projection of $\boldsymbol{w}^*$ onto the matrix $\boldsymbol{Q}^T$, where $\boldsymbol{Q}$ is the matrix of eigen vectors of the hessian $\boldsymbol{H}$. The i^{th} projections then get scaled by the a factor of $\frac{\lambda_i}{\lambda_i + \alpha_i}$ and again projected onto $\boldsymbol{Q}$. We can consider two different cases:

1. $\lambda_i >> \alpha$: Then the scaling factor is ≈ 1 ie the weights are not affected when the the eigen values are high
2. $\lambda_i << \alpha$: Then the scaling factor is ≈ 0 ie the weights get shrunk nearly to zero.

This means that L^2 Regularization would preserve only those directions along which the parameters contribute significantly (ie large λ_i) and in other directions with small eigen values the corresponding components of weight vector are decayed away.

(b) Effect of condition number of $\boldsymbol{H}$ on the performance of the L^2 Regularizer:

The condition number (ρ) of any matrix with eigen values λ_i is given by: $\rho = \frac{\lambda_{max}}{\lambda_{min}}$

For a large condition number ($\rho >> 1$) of the $\boldsymbol{H}$, we would have $\lambda_{max} >> \lambda_{min}$. We can comment on two aspects of the performance of regularization for this case:

1. **Numerical Stability**: From Eq 1 we can observe that the regularizer perturbs the eigen values of the Hessian matrix and thus makes it invertible which helps in numerical stability.
2. **Effect on Solution ($\boldsymbol{w}$):** The weights corresponding to the large eigen value (λ_{max}) will not be affected, however for small eigen values (λ_{min}) the weights get shrunk to zero.

3 Problem 2: L^1 Regularization

Given: The data is pre-processed such that no correlation exists between the input features (using PCA). This means that we can assume that the Hessian matrix ($\boldsymbol{H}$) for our quadratic approxima-

tion can be assumed as a diagonal matrix such that $\boldsymbol{H} = \sum_i H_{i,i} \boldsymbol{e}_i \otimes \boldsymbol{e}_i$ where each $H_{i,i} > 0$.

The Loss function for L^1 regularizer is given by : $\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \alpha \|\boldsymbol{w}\|_1$

(a) Quadratic Approximation :

$$\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^T \cancelto{0}{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*)} + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) + \alpha \|\boldsymbol{w}\|_1$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}) = \cancelto{0}{\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*)} + \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) + \alpha \operatorname{sign}(\boldsymbol{w}) = 0$$

$$\Rightarrow \sum_i (H_{i,i}(w_i - w_i^*) + \alpha \operatorname{sign}(w_i)) = 0$$

$$\Rightarrow w_i = \operatorname{sign}(w_i^*) \max\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\} \tag{2}$$

From Eq 2 we can observe the effect of the eigen values ($H_{i,i}$) of the Hessian matrix on the weights. We can consider two cases:

1. Case 1: $w_i^* > 0$
   (a) $w_i^* \leq \frac{\alpha}{H_{i,i}}$: Then the corresponding weight component $w_i = 0$. This happens when the contribution of the un-regularized loss ($H_{i,i} w_i^*$) gets overwhelmed by the L^1 regularizer term (α).
   (b) $w_i^* > \frac{\alpha}{H_{i,i}}$: Then the corresponding weight component w_i gets reduced by $\frac{\alpha}{Hi,i}$. This happens when the contribution of the un-regularized loss ($H_{i,i} w_i^*$) is not overwhelmed by the L^1 regularizer term (α).

   Therefore smaller the eigen value $H_{i,i}$, more chances that the corresponding weight component gets a value of zero

2. Case 2: $w_i^* < 0$
   (a) $|w_i|^* \leq \frac{\alpha}{H_{i,i}}$: Then the corresponding weight component $w_i = 0$.
   (b) $|w_i|^* > \frac{\alpha}{H_{i,i}}$: Then the corresponding weight component w_i gets increased by $\frac{\alpha}{Hi,i}$.

   Therefore larger the eigen value $H_{i,i}$, more chances that the corresponding weight component gets a value of zero

(b) Feature Selection: As we saw in part (a) the L^1 regularizer assigns a value of zero to certain weight components, this mans that the regularizer decided that the feature corresponding to the weight component is not a critical a critical feature and hence helps in selection of a subset of features which are useful. This is why the L^1 regularizer is also called as the LASSO (Least absolute shrinkage and selection operator).

4 Problem 4: Constrained Optimization

(a) The regularized loss function is given by:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta})$$

Where, for norm penalties the regularization function ($\Omega(\boldsymbol{\theta})$) is a Norm of the model parameters ($\boldsymbol{\theta}$). The above objective function can also be viewed as a generalized Lagrange function with the norm penalty term as a constraint.

The norm penalty (centered around origin) terms, in general, would tend to push the solution ($\boldsymbol{\theta}$) close to origin, ie to small values of $\boldsymbol{\theta}$ which would be a local minima for the problem. Also, the penalties can cause non-convex optimization procedures to get stuck in this local minima. This results in "dead units" while training which can be locally optimal even if it is possible to significantly reduce J by making the weights larger.

(b) When using explicit constraints, the optimization strategy is such that we descend a step-down the un-regularized cost function to get the new solution and then re-project this solution to the explicit constraint boundary $\Omega(\boldsymbol{\theta}) < k$. As this strategy does not encourage the weights to approach to origin, we can explore more and don't get stuck in a local minima.

When using high learning rates, it is possible to encounter a positive feedback loop in which large weights induces large gradients, which then induces a large update to the weights. If these updates consistently increase the size of the weights, then $\boldsymbol{\theta}$ rapidly moves away from the origin. A norm penalty in this case would simply push the weights to small value and will prevent from learning a better solution, however, explicit constraints with reprojection would prevent the feedback loop from continuing to increase the magnitude without bound and would allow for more exploration.

(c) In 2012 Hinton *et al.* recommended a strategy of constraining the norm of each column of the weight matrix of a neural net layer, rather than constraining the Frobenius norm of the entire weight matrix. Constraining the norm of each column separately prevents any one hidden unit from having very large weights. This means that the hidden units will now have weights of similar order and thus would not allow for dead units any longer. Which means we will be able to learn for the units which became dead earlier in the case of Frobenius norm strategy.

5 Problem 5: Regularization and Under-Constrained Problems

(a) Regularization helps in stabilizing of under-constrained problems by perturbing the eigen values of a non invertible matrix to make it invertible. For example in the case of logistic regression, where we have an analytical solution, we need to invert the matrix $\boldsymbol{X}^T\boldsymbol{X}$. This matrix is composed of only available training samples, and it is not guaranteed that the matrix will be invertible. For

instance the matrix can be singular whenever the data-generating distribution has no variance in some direction, r when no variance is observed in some direction because there are fewer examples than the input features. However, when we use L^2 regularizer, we then need to invert the matrix $(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})$ which is guaranteed to be invertible. Therefore regularization helps in stabilization of solution.

Regularization is connected to the Occam's razor, as regularizer puts constraints on the number of allowable hypothesis. For instance, in the scenario when $\boldsymbol{w}, 2\boldsymbol{w}, 3\boldsymbol{w}, ...$ all provide consistent hypothesis then we can have an infinite set of possible hypothesis. However, with the addition of regularization the weights get penalized and are not allowed to increase without a bound, this reduces the number of possible hypothesis and thus we now choose from a less complex hypothesis space.

6 Problem 6: Data Augmentation and Noise Robustness

(a) Dataset augmentation for object recognition tasks helps in reducing the generalization error. For object detection we have to design a classifier which can detect the same object when placed at different positions or orientations (in plane rotations) in the same image. Which means we want our model to learn rigid body motion invariant features, we can do this by augmenting the training dataset with new images by slightly translating (ie using pixel shifts), rotating (ie using a homography) or scaling the object in the images and then train with the augmented dataset. Identifying possible types of transformations depends on the dataset and classification task.

(b) For images of letters and numbers we cannot allow a horizontal flip of 180° rotations as these can change the true label of the image. For instance, for letters "b" and "d" and numbers "6" and "9" cannot allow for horizontal flips or 180° rotations as these transformations will change the correct class of the image.

(c) Given : Data set $(\boldsymbol{x}, y)$, a model which estimates the label for the samples using the function $\hat{y}(\boldsymbol{x})$.. The squared loss is given by:

$$J = \mathbb{E}_{p(x,y)}[(\hat{y}(\boldsymbol{x}) - y)^2]$$

Now we add random noise $\epsilon_{\boldsymbol{w}} \sim \mathcal{N}(0, \eta\boldsymbol{I})$ to the weights of the model. The new loss is given by:

$$\begin{aligned}\tilde{J} &= \mathbb{E}_{p(x,y)}[(\hat{y}_\epsilon(\boldsymbol{x}) - y)^2] \\ &= \mathbb{E}_{p(x,y)}[(\hat{y}_\epsilon(\boldsymbol{x})^2 + y^2 - 2\hat{y}_\epsilon(\boldsymbol{x})y)]\end{aligned}$$

Using Taylor's expansion we can say: $\hat{y}_\epsilon(\boldsymbol{x}) \approx \hat{y}(\boldsymbol{x}) + (\nabla_{\boldsymbol{w}}\hat{y})^T\epsilon$

and the fact: $\mathbb{E}[\epsilon] = 0$

$$\begin{aligned}\Rightarrow \tilde{J} &= \mathbb{E}_{p(x,y),\epsilon}[(\hat{y}(\boldsymbol{x}) + (\nabla_{\boldsymbol{w}}\hat{y})^T\epsilon)^2 + y^2 - 2(\hat{y}(\boldsymbol{x}) + (\nabla_{\boldsymbol{w}}\hat{y})^T\epsilon)y] \\ &= J + \mathbb{E}_{p(x,y),\epsilon}[(\nabla_{\boldsymbol{w}}\hat{y})^T\epsilon \otimes \epsilon(\nabla_{\boldsymbol{w}}\hat{y})] + \\ &\quad \cancelto{0}{\mathbb{E}_{p(x,y),\epsilon}[2\hat{y}(\boldsymbol{w})\nabla_{\boldsymbol{w}}\hat{y}\epsilon]} \quad + \cancelto{0}{\mathbb{E}_{p(x,y),\epsilon}[-2\hat{y}(\boldsymbol{w})\nabla_{\boldsymbol{w}}\hat{y}\epsilon y]} \\ &= J + \eta\mathbb{E}_{p(x,y)}[\|\nabla_{\boldsymbol{w}}\hat{y}\|^2]\end{aligned}$$

From the above solution we can see that adding random noise to the weights of the network boils down to adding a regularization term f the form $\eta\mathbb{E}_{p(x,y)}[\|\nabla_{\boldsymbol{w}}\hat{y}\|]$. Such a regularization term encourages the parameters to go to regions of parameter space where small perturbations of weights have a relatively small influence on the output. Thus the optimal solution will be the one where the model is relatively insensitive to small variations in the weights, ie finding points that are not merely minima but a minima surrounded by flat regions.

(d) Label smoothing is done to account for mistakes in the y labels. We account for these mistakes by assuming that for some small constant ϵ, the training set label is correct with a probability $1 - \epsilon$, and otherwise any other possible label might be correct. This assumption is then easily incorporated into the cost function analytically, rather than by explicitly drawing noise samples where instead of hard targets for class labels we now have soft targets.

In case of softmax with multi-class (k classes), we set the soft targets to $(1 - \epsilon)$ for a hard label of 1 and $\frac{\epsilon}{k-1}$ for a hard label of 0. We then create new one hot labels with the soft targets and incorporate them in the softmax function and the cross-entropy loss function as follows:

$$p(y_i|x) = \frac{exp((1-\epsilon)z_i)}{\sum_{j\neq i} exp(\frac{\epsilon}{k-1}z_j) + exp((1-\epsilon)z_i)}$$

$$J = \sum_i^k y_i log(p(y_i|x))$$

$$y_i = \begin{cases} (1-\epsilon) & \text{if in } i^{th} \text{ class} \\ \frac{\epsilon}{k-1} & \text{otherwise} \end{cases}$$

7 Problem 7: Generative Training

(a) A purely discriminative training criterion is used in the case of supervised learning where we have the access to the samples and labels from the joint distribution $p(x, y)$. In such a case we can estimate $P(y|x)$ using the discriminative criterion $-log(P(y|x))$. Whereas, a purely generative criterion is used in the case of un-supervised learning where we have access to only un-labelled samples from the distribution $p(x)$ and we try to learn the representation of data or the true data generating distribution process. In such a case, we estimate $p(y|x)$ using a generative criterion such as $-log(p(x))$.

(b) Striking balance between generative and discriminative criterion.

(i) Semi-Supervised Learning: In this case we have both labelled and un-labelled data available for learning and the goal is to learn a representation for the data so that examples from the same class have similar representations. We then aggregate the losses for purely generative and discriminative processes using respective criterion and allow for parameter sharing between the two models. By controlling the of contribution of generative criterion in the total loss, would help in learning the best representation of data.

(ii) Multi-Task Learning: In this case we improve the generalization by learning parameters arising out of several tasks such that each tasks shares a subset of weights with the other task ie parameter sharing. For example for classification problem we can improve the generalization capability of the classifier if the classifier classifies examples from a generative representation. This can be done by multi-tasking a generative task (such as autoencoders) which will learn a representation of data and club it with the classifier which would distinguish between the samples. When both the task share parameters in the initial layers of the network then it can improve the generalization.

(iii) Parameter Sharing among multiple-tasks helps in improving the generalization capability of classifier/auto-encoder. Parameters among multiple tasks are shared during the initial layers of the network when the network is trying to find the best set of low-level features. Then individual tasks branch-out will have only task-specific parameters (in the final layers).

8 Problem 8: Early Stopping

(a) Validation set is subset of the actual training set such that its size is proportional to the testing set. Validation set is different from the testing set because we are allowed to use validation set to improve our training and also tune the model hyperparameters. The Validation set is mainly used to obtaining optimal model hyperparameters such as number of training steps, learning rate using early stopping algorithm.

(b) When using a model which has sufficient capacity to overfit the task, we observe that training

error decreases (mainly because the model would overfit to training data), but the validation set error initially decreases and then rises resulting in a U-shaped curve. In *early stopping* we store the model parameters for each update and keep a track of the best validation error, and we terminate training if the validation error does not improve for a fixed number ("patience") of time steps. Thus *early stopping* helps us in determining the optimal number of time steps required for training with the complete training set.

(c) After identifying the optimal number of training time-steps using *early stopping*, it is desirable to add the validation set to the training set because we have not learned from the samples in validation set. Once the validation set is combines with the training set we can retrain for the same number of steps as the early stopping procedure determined. However, as the size of training set is bigger than the size of training set for *early stopping*, there is not a good way of knowing whether to retrain for the same number of parameter updates or the same number of passes through the dataset. This because on the second round of training, each pass through the dataset will require mode parameter updates because the training set is bigger.

(d) Another method of using validation set is to resume training from the *early stopping* procedure by including the validation set into the training set. However, now we don't have a guide for when to stop in terms of number of steps. In this process we continue training till we get a lower validation loss than what was obtained during *early stopping*. This strategy avoids the high cost of retraining from scratch but is not as well behaved. For instance, we don't have any guarantee that the new validation error would drop below what was computed during *early stopping*.

9 Problem 9: Early Stopping as an L^2 Regularizer

(a) Let us reiterate the result we had for L^2 regularization in Eq 1:

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \\ \Rightarrow Q^T\boldsymbol{w} &= (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \\ &= [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^T\boldsymbol{w}^* \end{aligned} \tag{3}$$

Now for Early stopping, with a quadratic approximation ($\hat{J}$) of the un-regularized loss function (with only weights $\boldsymbol{w}$ as parameters) around the point $\boldsymbol{w}^*$, we have:

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

The update equation then becomes:

$$\Rightarrow \boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)})$$

$$= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

$$\Rightarrow \boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

Using: $\boldsymbol{H} = \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q}$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

$$\Rightarrow \boldsymbol{Q}^T(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda}) \boldsymbol{Q}^T (\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*)$$

The above equation is a recursive equation (in τ) and can be simplifies using: $\boldsymbol{w}^0 = 0$

$$\Rightarrow \boldsymbol{Q}^T(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau \boldsymbol{Q}^T (-\boldsymbol{w}^*)$$

$$\Rightarrow \boldsymbol{Q}^T \boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau] \boldsymbol{Q}^T (\boldsymbol{w}^*) \tag{4}$$

On Comparing Eq 3 and Eq 4 we get:

$$(\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha$$

$$\Rightarrow \tau log(1 - \epsilon \lambda_i) = log(\frac{1}{\frac{\lambda_i}{\alpha} + 1})$$

Using Taylor's Series: $log(1 + x) \approx x$

assuming: $\epsilon \lambda_i << 1 \quad \frac{\lambda_i}{\alpha} << 1$

$$\tau(-\epsilon \lambda_i) = -(\frac{\lambda_i}{\alpha})$$

$$\tau = \frac{1}{\epsilon \alpha} \quad \text{or } \alpha = \frac{1}{\epsilon \tau}$$

Therefore from the above derivation we can see that early stopping is equivalent to L^2 regularization with an automatic setting of regularization parameter $\alpha = \frac{1}{\epsilon \tau}$.

10 Problem 10: Bootstrap Aggregating and Boosting

(a) Bagging is a type of ensemble method that allows same kind of model, training algorithm and objective function to be reused several times by constructing k different datasets by sampling with replacement from the original dataset and then using these datasets for train k different models.

Bagging can also be useful even when the training is done on the entire dataset, this is because we can have different learned models due to differences in random initialization, random selection

of mini-batches, differences in hyperparameters. Thus the different members of ensemble can make partially independent errors and will be useful in improving the generalization of model.

(b) In boosting we allow for the model to make decisions during training about training set selection, or model selection (ie adding or removing hidden units). Thus it be viewed as an ensemble as it can have a pool of different models for different decisions.

11 Problem 11: Dropout

(a) Dropout requires us to select a binary-mask, which results in hiding some units in the network, for every minibatch that we select from the training data. This means that for every minibatch we can have a different mask and thus we would be learning from an ensemble of exponentially many nueral networks. We use the word exponential because there can be 2^n, where n is the number of non-output units in total, possible masks from which we choose a single mask for one minibatch.

(b) The main differences between dropout and bagging are:

1. While in bagging we train each model separately using the entire training dataset, In dropout each model is trained using one or few minibatches (whose size is smaller than training set).
2. In bagging the models are learnt independent of each other ie the weights of one model do not affect the other, whereas in dropout the models share parameters.

12 Problem 12: Dropout Continued

(a) At inference, using the ensemble, we can estimate $p_{ensemble}$ by evaluating $p(y|x)$ in one model: the model with all units, but with the weights going out of unit i multiplied by the probability of including unit i,ie $p(\mu_i = 1)$. This approach is called weight scaling inference rule. The $p_{ensemble}$ can be approximated as:

$$\begin{aligned} p_{ensemble} &\propto \sqrt[2^n]{\prod_{d \in \{0,1\}^n} exp(\boldsymbol{W}_y^T(d \odot \boldsymbol{v}) + b_y)} \\ &= exp(\frac{1}{2}\boldsymbol{W}_y^T\boldsymbol{v} + b_y) \end{aligned}$$

Goodfellow *et al.* (2013) found experimentally that the weight scaling approximation can work better than Monte Carlo approximation of the ensemble predictor. This held true even when the Monte Carlo approximation was allowed to draw samples up-to 1000 sub-networks.

(b) Explanation of methods

(i) Dropout Boosting: Here we draw out exactly the same mask noise as traditional dropout but it lacks its regularization effect because we update the parameters which exist only in the

current realization. This method shows that the advantage of dropout is mainly due to the way training is done and not necessarily the binary mask.

(ii) DropConnect: This is a special case of dropout where each product between a single scalar weight and a single hidden unit sate is considered a unit that can be dropped.

(ii) Non-Sparse masks: This method draws the mask from a non-binary distribution $\mu \sim \mathcal{N}(1, \boldsymbol{I})$ and has been shown to outperform traditional dropout. Also, since $\mathbb{E}[\mu] = 1$, the standard network automatically implements approximate inference in the ensemble without needing any weight scaling.