Advanced Deep Learning

Dr. **Ali Ghodsi**, University Of Waterloo

Collected By: Ardavan Modarres, K. N. Toosi University of Technology

October 19, 2023

Contents

List of Figures

1 Preface

The notes are from STAT 940, Advanced Deep Learning course, taught in Fall 2023 at the University of Waterloo by Professor Ali Ghodsi.

2 Lecture 2

2.1 Perceptron

Perceptron is the main building-block of Neural Networks. A single perceptron is depicted in figure 1.

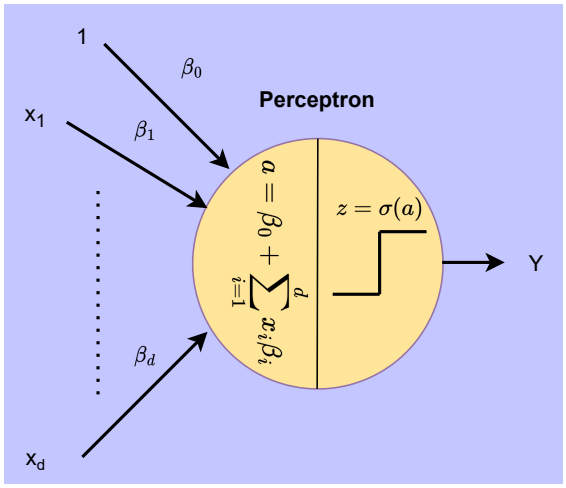

Figure 1: Perecptron

Where the input $\underline{x} = [x_1, x_2, ..., x_d]^T \in \mathbb{R}^d$ and Y comes from a finite set ($Y \in \{-1, +1\}$) and the task is classification, the label of one class is -1 and the label of the other class is $+1$. We aim to find vector $\underline{\beta} = [\beta_1, \beta_2, ..., \beta_d]^T$ such that operation:

$$y = \sigma(\underline{x}^T \underline{\beta} + \beta_0) \tag{1}$$

maps each input sample x^i to the corresponding label. Equation 1 defines a hyper-plane in d-dimensional space. Let's look at the geometry and some or properties of this hyper-pane which will be handy later. Assume we have a dataset $D = \{(x_i, y_i)\}_{i=1}^n$ comprising n points with corresponding labels. For better intuition, assume that $\underline{x} \in \mathbb{R}^2$ and we are in 2-dimensional space. The hyper-plane and 3 arbitrary points on this hyper-plane are depicted in figure 2.

1. $\underline{\beta}$ is perpendicular to the hyper-plane.

$$\underline{\beta}^T \underline{x}_1 + \beta_0 = \underline{\beta}^T \underline{x}_2^T + \beta_0 = 0 \Rightarrow \underline{\beta}(\underline{x}_1 - \underline{x}_2) = 0 \Rightarrow \underline{\beta}^T \perp (\underline{x}_1 - \underline{x}_2) \tag{2}$$

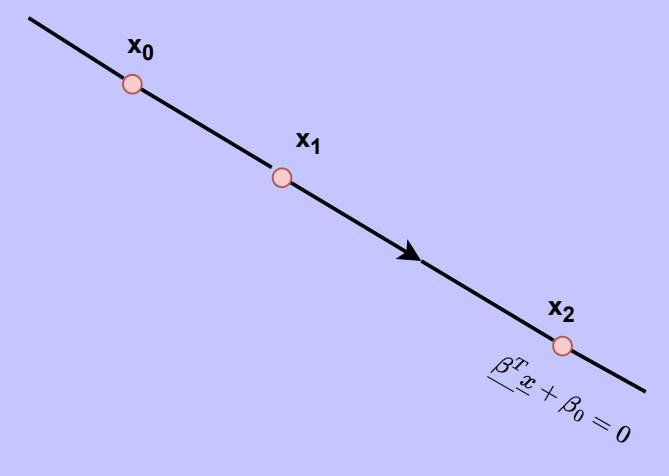

Figure 2: The hyper-plane in 2-dimensional space

2. For any arbitrary x_0 point on the hyper-plane:

$$\underline{\beta}^T \underline{x}_0 + \beta_0 = 0 \Rightarrow \beta_0 = -\underline{\beta}^T \underline{x}_0 \tag{3}$$

3. For an arbitrary point x which is not on the hyper-plane, the Signed Distance of x to the hyper-plane is calculated by finding the distance of x to an arbitrary point x_0 on the hyper-plane and then projecting the distance to the direction of $\underline{\beta}$:

$$d_s = \underline{\beta}^T (\underline{x} - \underline{x}_0) = \underline{\beta}^T \underline{x} - \underline{\beta}^T \underline{x}_0 = \underline{\beta}^T \underline{x} + \beta_0 \tag{4}$$

4. The Unsigned Distance of x to the hyper-plane is calculated by multiplying the Signed Distance by the corresponding label of the point x:

$$d_{us} = y_i (\underline{\beta}^T \underline{x}_i + \beta_0) \tag{5}$$

The properties 1 and 3 are briefly depicted in figure 3.

The next step is defining an appropriate Scoring Function (Loss Function). The first choice might be finding $(\underline{\beta}, \beta_0)$ such the number of misclassified samples is minimized.

$$Err_1(\underline{\beta}, \beta_0) : \text{The number of misclassified samples} \tag{6}$$

But since Err_1 is a non-derivative function, this choice is going to be problematic later and we have to make sure that the scoring function we define is Differentiable. The traditional choice for loss function of perceptron is summation of the distance of all the misclassified samples to the hyper-plane. The loss function is depicted in equation .

$$Err(\underline{\beta}, \beta_0) = \sum_{i \in M} -y_i (\underline{\beta}^T \underline{x}_i + \beta_0) \tag{7}$$

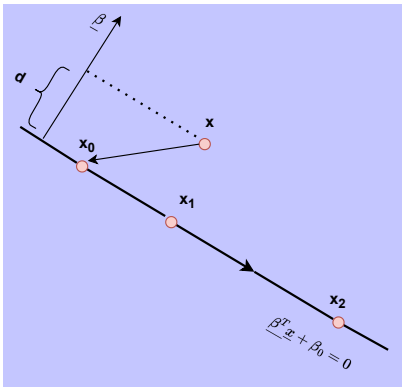

Figure 3: The hyper-plane in 2-dimensional space and properties 1 and 3

And M is the set of all misclassified samples. The intuition behind multiplying the loss function depicted in equation 7 by -1 is that the predicted label for a misclassified sample is the opposite of the ground truth and the distance $(\underline{\beta}^T \underline{x}_i + \beta_0)$ is going to be negative, so by multiplying the distance by -1 and minimizing the distance of misclassified samples, we are placing the hyper-plane such that misclassified samples are pushed to the other side of it. We can find the optimal β using Gradient Descent algorithm which is depicted in equations 8-9.

$$\underline{\beta}^{new} \leftarrow \underline{\beta}^{old} - \rho \frac{\partial Err}{\partial \underline{\beta}} \tag{8}$$

$$\beta_0^{new} \leftarrow \beta_0^{old} - \rho \frac{\partial Err}{\partial \beta_0} \tag{9}$$

Where ρ is the Learning Raye in equations 8-9. Notably, ρ is a hyper-parameter that should be adjusted and if $\underline{\beta}$ and β_0 are close to a minimum, ρ should be small and if $\underline{\beta}$ and β_0 are far from a minimum, ρ should be large. The partial derivatives of Err with respect to each of $\underline{\beta}$ and β_0 are depicted in equations 10-11.

$$\frac{\partial Err}{\partial \underline{\beta}} = \sum_{i \in M} -y_i x_i \tag{10}$$

$$\frac{\partial Err}{\partial \beta_0} = \sum_{i \in M} -y_i \tag{11}$$

And Gradient Decent algorithm can be written as depicted in equation 12.

$$\begin{bmatrix} \underline{\beta}^{new} \\ \\ \beta_0^{new} \end{bmatrix} = \begin{bmatrix} \underline{\beta}^{old} \\ \\ \beta_0^{old} \end{bmatrix} + \rho \begin{bmatrix} \sum_{i \in M} y_i x_i \\ \\ \sum_{i \in M} y_i \end{bmatrix} \tag{12}$$

But in practice, we usually perform Stochastic Gradient Descent as depicted in equation 13.

$$\begin{bmatrix} \underline{\beta}^{new} \\ \beta_0^{new} \end{bmatrix} = \begin{bmatrix} \underline{\beta}^{old} \\ \beta_0^{old} \end{bmatrix} + \rho \begin{bmatrix} y_i x_i \\ y_i \end{bmatrix} \tag{13}$$

Equation 13 is performed until it converges, and convergence means there is no significant difference between β^{new} and β^{old}. The number of required eterations for convergence depends on the learning rate and the gap between cluster of two classes. Notably perceptron algorithm only works for two class classification and if two classes are linearly separable equation 13 converges and if two classes are not linearly separable equation 13 does not converge. Also the solution is not unique. Linear separability is mathematically formulated in equation .

$$\forall i \ \exists \gamma \geq 0 \text{ such that } y_i(\underline{\beta}^T \underline{x} + \beta_0) \geq \gamma \tag{14}$$

Where γ in equation 14 is a positive number which depicts the well separability of the data.

2.2 Backpropagation

Three consecutive layers of a deep neural network is depicted in figure 4. We aim to find neural networks parameters such that an arbitrary loss function such as Mean Square Error ($E(y, \hat{y}) = (y - \hat{y})^2$) is minimized.

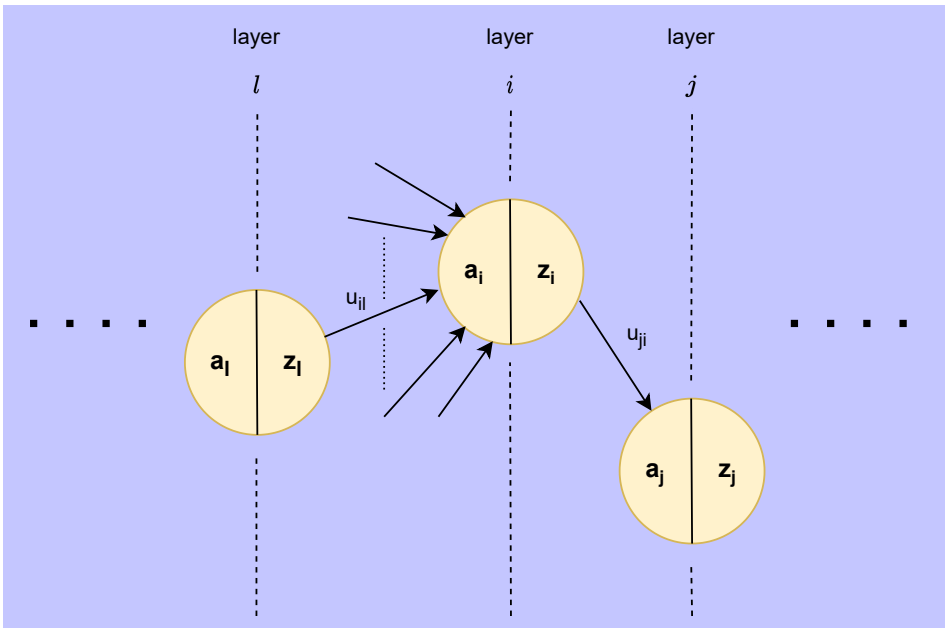

Figure 4: Three consecetive layers of a deep neural network

Assume we want to find $\frac{\partial E}{\partial u_{il}}$; by applying chain-rule:

$$\frac{\partial E}{\partial u_{il}} = \frac{\partial E}{\partial a_i}.\frac{\partial a_i}{\partial u_{il}} = \frac{\partial E}{\partial a_i}.\frac{\partial \sum_L z_l u_{il}}{\partial u_{il}} = \delta_i.z_l \tag{15}$$

Now let's find $\delta_i = \frac{\partial E}{\partial a_i}$ by relating it to a_j and using recursion.

$$\delta_i = \frac{\partial E}{\partial a_i} = \sum_J \frac{\partial E}{\partial a_j}.\frac{\partial a_j}{\partial a_i} = \sum_J \frac{\partial E}{\partial a_j}.\frac{\partial \sum_J u_{ji}\sigma(a_i)}{\partial a_i} = \dot{\sigma}(a_i)\sum_J \delta_j u_{ji} \tag{16}$$

In conclusion we have:

$$\frac{\partial E}{\partial u_{il}} = \delta_i.z_l \tag{17}$$

$$\delta_i = \dot{\sigma}(a_i)\sum_J \delta_j u_{ji} \tag{18}$$

We can add an additional layer with linear activation function on top of our neural network. Such a network with additional layer is depicted in figure 5. For this additional linear layer we can compute δ_k as depicted in equation

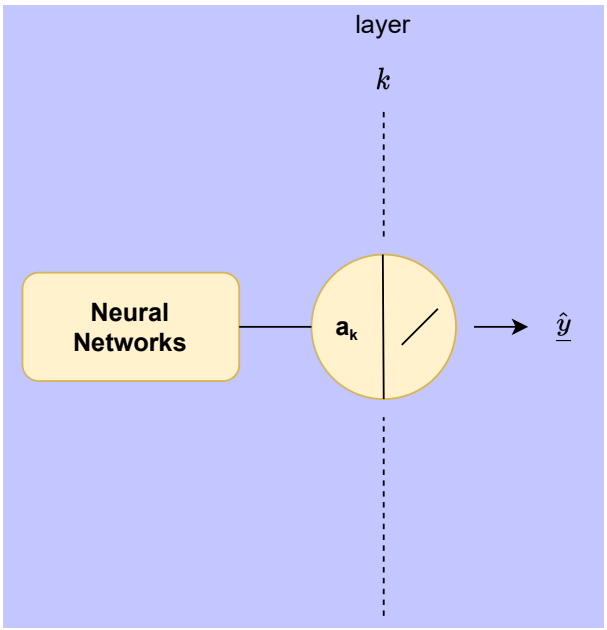

Figure 5: Additional layer with linear activation function on top of the neural network

19.

$$\delta_k = \frac{\partial E}{\partial a_k} = \frac{\partial (y-\hat{y})^2}{\partial \hat{y}} = 2(y-\hat{y}) \tag{19}$$

And we can compute the δ for previous layers using equation 18 and update all trainable parameters of the neural network using gradient descent:

$$u_{il}^{new} \leftarrow u_{il}^{old} - \rho \frac{\partial E}{\partial u_{il}} \tag{20}$$

3 Lecture 3

3.1 Stochastic Gradient Descent (SGD)

In practice we usually do SGD since GD is computationally expensive and SGD is more cost-effective. Assume $Q(w)$ is total cost function on a batch of data and $Q(w)$ is the sum of individual cost of each sample as depicted in equation 21.

$$Q(\underline{w}) = \sum_{i}^{n} Q_i(\underline{w}) \tag{21}$$

Gradient descent update is depicted in equation 22.

$$\underline{w}^{t+1} = \underline{w}^{t} - \rho \nabla_w Q(\underline{w}) \tag{22}$$

And stochastic gradient descent update is depicted in equation 25.

$$\underline{w}^{t+1} = \underline{w}^{t} - \rho \nabla_w Q_i(\underline{w}) \tag{23}$$

By taking expectation from equation 25:

$$\mathbb{E}[\underline{w}^{t+1}] = \mathbb{E}[\underline{w}^{t}] - \mathbb{E}[\rho \nabla_w Q_i(\underline{w})] \tag{24}$$

Let's calculate the expectation of second term in equation 25:

$$\mathbb{E}[\rho \nabla_w Q_i(\underline{w})] = \rho\, \mathbb{E}[\nabla_w Q_i(\underline{w})] = \rho \nabla_w\, \mathbb{E}[Q_i(\underline{w})] = \rho \nabla_w \sum_{i=1}^{n} Q_i(\underline{w}) = \rho \nabla_w Q(\underline{w}) \tag{25}$$

Equation 25 states that on expectation, SGD updates the network parameters in the same direction that GD does.

There are lot's techniques that aim to boost the performance of vanilla gradient descent and there are lot's of optimizers in any framework for training neural networks. One popular technique is Momentum. Momentum update rules are depicted in equations 26 and 27 respectively.

$$\underline{v}^{t+1} = \beta \underline{v}^{t} + (1-\beta) \nabla_w Q(\underline{w}^{t}) \tag{26}$$

$$\underline{w}^{t+1} = \underline{w}^{t} - \rho \underline{v}^{t+1} \tag{27}$$

Where β is called Momentum Coefficient or Discount Rate and has a range between $(0.9 - 0.99)$ and is usually is a number close to 1. v^t is the velocity

term which is a weighted running sum of all previous gradient vectors. Let's compute the momentum update rule for the first three iterations.
For $t = 0$:

$$\underline{v}^1 = \beta \times 0 + (1-\beta)\nabla_w Q(\underline{w}^0) \tag{28}$$

$$\underline{w}^1 = \underline{w}^0 - \rho(1-\beta)\nabla_w Q(\underline{w}^0) \tag{29}$$

Note that $\underline{v}^0 = (0, 0, ..., 0)$. For $t = 1$:

$$\underline{v}^2 = \beta(1-\beta)\nabla_w Q(\underline{w}^0) + (1-\beta)\nabla_w Q(\underline{w}^1) \tag{30}$$

$$\underline{w}^2 = \underline{w}^1 - \rho(1-\beta)[\beta\nabla_w Q(\underline{w}^0) + \nabla_w Q(\underline{w}^1)] \tag{31}$$

For $t = 2$:

$$\underline{v}^3 = \beta^2(1-\beta)\nabla_w Q(\underline{w}^0) + \beta(1-\beta)\nabla_w Q(\underline{w}^1) + (1-\beta)\nabla_w Q(\underline{w}^2) \tag{32}$$

$$\underline{w}^3 = \underline{w}^2 - \rho(1-\beta)[\beta^2\nabla_w Q(\underline{w}^0) + \beta\nabla_w Q(\underline{w}^1) + \nabla_w Q(\underline{w}^2)] \tag{33}$$

3.2 Sten's Unbiased Risk Estimator (SURE)

Assume we have a training dataset $T = \{(x_i, y_i)\}$ and there is a True Function $f(x)$ that map each sample x to the corresponding label and we aim to estimate a mapping $\hat{y} = \hat{f}(x)$ as close as possible to $f(x)$. We assume:

$$\text{True Function } f_i = f_i(x) = f_i(.) \tag{34}$$

$$\text{Estimated Function } \hat{f}_i = \hat{f}_i(x) = \hat{f}_i(.) \tag{35}$$

$$y_i = f(x_i) + \epsilon_i;\ \epsilon_i \sim \mathcal{N}(0,\, \sigma^2) \tag{36}$$

$$\text{True Error (Err)} \ = (f - \hat{f})^2 \tag{37}$$

$$\text{Empirical Error (err)} \ = (y - \hat{y})^2 \tag{38}$$

We like to calculate the True Error $(f - \hat{f})^2$ but we can not, so we aim to find the relation between the Empirical Error $(y - \hat{y})^2$ that we cant compute to the True Error.

$$\begin{aligned}
\mathbb{E}[(\hat{y}_i - y_i)^2] &= \mathbb{E}[(\hat{f}_i - f_i - \epsilon_i)^2] = \mathbb{E}[(\hat{f}_i - f_i)^2 + \epsilon_i^2 - 2\epsilon_i(\hat{f}_i - f_i)] \\
&= \mathbb{E}[(\hat{f}_i - f_i)^2] + \mathbb{E}[\epsilon_i^2] - 2\,\mathbb{E}[\epsilon_i(\hat{f}_i - f_i)] \\
&= \mathbb{E}[(\hat{f}_i - f_i)^2] + \sigma^2 - 2\,\mathbb{E}[\epsilon_i(\hat{f}_i - f_i)]
\end{aligned} \tag{39}$$

Let's investigate the term $-2\,\mathbb{E}[\epsilon_i(\hat{f}_i - f_i)]$ more. We need to learn about Stein's Lemma at first which is depicted in equation 40. For $x \sim \mathcal{N}(\theta,\, \sigma^2)$ and differentiable function $g(.)$:

$$\mathbb{E}[g(x)(x-\theta)] = 2\sigma^2[\frac{\partial g(x)}{\partial x}] \tag{40}$$

Case 1: $(x_i, y_i) \notin T$
By applying Stein's Lemma on the last term of equation 39 we can rewrite it as:

$$\begin{aligned}\mathbb{E}[\epsilon_i(\hat{f}_i - f_i)] &= 2\sigma^2[\frac{\partial(\hat{f}_i - f_i)}{\partial \epsilon_i}] \\ &= 2\sigma^2[\frac{\partial \hat{f}_i}{\partial \epsilon_i} - \frac{\partial f_i}{\partial \epsilon_i}]\end{aligned} \tag{41}$$

Any perturbation on ϵ_i has no effect on the True Function $(f(.))$, so $\frac{\partial f_i}{\partial \epsilon_i} = 0$. Since $(x_i, y_i) \notin T$ and $\hat{f}$ is estimated based on samples within the train set, also any perturbation on ϵ_i has no effect on the Estimated Function $(\hat{f}(.))$ as well and $\frac{\partial \hat{f}_i}{\partial \epsilon_i} = 0$ too. So we can rewrite equation 39 as depicted in equation 42-45.

$$\mathbb{E}[(\hat{y}_i - y_i)^2] = \mathbb{E}[(\hat{f}_i - f_i)^2] + \sigma^2 \tag{42}$$

$$\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \sum_{i=1}^{m}(\hat{f}_i - f_i)^2 + m\sigma^2 \tag{43}$$

$$err = Err + m\sigma^2 \Rightarrow Err = err - m\sigma^2 \tag{44}$$

Case 2: $(x_i, y_i) \in T$
Since $(x_i, y_i) \in T$ and $\hat{f}$ is estimated based on samples within the train set, perturbation on ϵ_i has effect on the Estimated Function $(\hat{f}(.))$ and $\frac{\partial \hat{f}_i}{\partial \epsilon_i} \neq 0$. So we can rewrite equation 39 as depicted in equation 45-47.

$$\mathbb{E}[(\hat{y}_i - y_i)^2] = \mathbb{E}[(\hat{f}_i - f_i)^2] + \sigma^2 - 2\sigma^2\frac{\partial \hat{f}_i}{\partial y_i} \tag{45}$$

$$\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \sum_{i=1}^{n}(\hat{f}_i - f_i)^2 + m\sigma^2 - 2\sigma^2\sum_{i=1}^{n}\frac{\partial \hat{f}_i}{\partial y_i} \tag{46}$$

$$err = Err + n\sigma^2 - 2\sigma^2\sum_{i=1}^{n}\frac{\partial \hat{f}_i}{\partial y_i} \Rightarrow Err = err - n\sigma^2 + 2\sigma^2\sum_{i=1}^{n}\frac{\partial \hat{f}_i}{\partial y_i} \tag{47}$$

Note that by applying chain rule we can rewrite $\frac{\partial \hat{f}_i}{\partial \epsilon_i}$ as depicted in equation 48.

$$\frac{\partial \hat{f}_i}{\partial \epsilon_i} = \frac{\partial \hat{f}_i}{\partial y_i}\frac{\partial y_i}{\partial \epsilon_i} = \frac{\partial \hat{f}_i}{\partial y_i} \tag{48}$$

The term $\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{f}_i}{\partial y_i}$ is not easy to compute and increases with the capacity of $\hat{f}$. We add some terms as regularization terms to the loss function on behalf of this term to prevent over-fitting.

4 Lecture 4

In many cases we add a term to the original loss function as a regularization term which is increased as the capacity of the model increases.

$$\tilde{J}(\underline{w}, X, y) = J(\underline{w}, X, y) + \alpha\Omega(\underline{w}) \tag{49}$$

Where $\Omega(\underline{w})$ is the regularization term. Taylor expansion is a useful tool for mathematical analysis of the effect of the regularization terms, so let's do a quick recap of Taylor Expansion. Taylor Expansion of an arbitrary function $f(.)$ around point a can be found in equation 50.

$$f(x) = f(a) + \frac{\dot{f}(a)}{1!}(x-a) + \frac{\ddot{f}(a)}{2!}(x-a)^2 + \cdots \tag{50}$$

An arbitrary cost function $J(\underline{w})$ can be approximated around the neighbourhood of an arbitrary optimum $\underline{w}^*$ as stated in equation 51.

$$\begin{aligned} \hat{J}(\underline{w}) &= J(\underline{w}^*) + \nabla_{\underline{w}} J(\underline{w}^*) + \frac{1}{2}(\underline{w} - \underline{w}^*)H(\underline{w} - \underline{w}^*)^T \\ &= J(\underline{w}^*) + \frac{1}{2}(\underline{w} - \underline{w}^*)H(\underline{w} - \underline{w}^*)^T \end{aligned} \tag{51}$$

Note that $\nabla_{\underline{w}} J(\underline{w}^*) = 0$ since $\underline{w}^*$ is an optimum.

4.1 Effect of Weight Decay as regularization term

The loss function in this case can be written as:

$$\tilde{J}(\underline{w}) = J(\underline{w}) + \frac{1}{2}\alpha \| \underline{w} \|^2 \tag{52}$$

The Taylor Expansion of this regularized loss function around optimum $\underline{w}^*$ is:

$$\hat{\tilde{J}}(\underline{w}) = J(\underline{w}^*) + \frac{1}{2}(\underline{w} - \underline{w}^*)H(\underline{w} - \underline{w}^*)^T + \frac{1}{2}\alpha \| \underline{w} \|^2 \tag{53}$$

To find the optimum of $\hat{\tilde{J}}(\underline{w})$ with respect to $\underline{w}^*$, we should solve $\nabla_{\underline{w}} \hat{\tilde{J}}(\underline{w}) = 0$.

$$\nabla_{\underline{w}} \hat{\tilde{J}}(\underline{w}) = H(\underline{w} - \underline{w}^*) + \alpha\underline{w} \tag{54}$$

$$\begin{aligned} H(\underline{w} - \underline{w}^*) + \alpha\underline{w} = H\underline{w} - H\underline{w}^* + \alpha\underline{w} = 0 \\ \Rightarrow (H + \alpha I)\underline{w} = H\underline{w}^* \\ \Rightarrow \underline{w} = (H + \alpha I)^{-1} H\underline{w}^* \end{aligned} \tag{55}$$

We can rewrite equation 55 using singular-value decomposition of H:

$$\begin{aligned}\underline{w} &= (H+\alpha I)^{-1}H\underline{w}^* \\ &= (Q\Lambda Q^T + Q\alpha Q^T)^{-1}Q\Lambda Q^T\underline{w}^* \\ &= (Q(\Lambda+\alpha I)Q^T)^{-1}Q\Lambda Q^T\underline{w}^* \\ &= Q(\Lambda+\alpha I)^{-1}Q^{-1}Q\Lambda Q^T\underline{w}^* \\ &= Q(\Lambda+\alpha I)^{-1}\Lambda Q^T\underline{w}^*\end{aligned} \tag{56}$$

The $(\Lambda+\alpha I)^{-1}\Lambda$ matrix is a diagonal matrix and each entry on the main diagonal of this matrix has the form of $\frac{\lambda_i}{\lambda_i+\alpha}$. Let's consider the below two cases:

- $\lambda_i \gg \alpha : \frac{\lambda_i}{\lambda_i+\alpha} \to 1.$
- $\lambda_i \ll \alpha : \frac{\lambda_i}{\lambda_i+\alpha} \to 0.$

Which means only search alongside the significant directions is continued and search alongside insignificant direction is stopped.

4.2 Effect of Early Stopping as regularization term

The Taylor Expansion of the cost function is depicted in equation 57.

$$\hat{J}(\underline{w}) = J(\underline{w}^*) + \frac{1}{2}(\underline{w}-\underline{w}^*)H(\underline{w}-\underline{w}^*)^T \tag{57}$$

And the derivative of $\hat{J}(\underline{w})$ can be found in equation 58.

$$\nabla_{\underline{w}}\hat{J}(\underline{w}) = H(\underline{w}-\underline{w}^*) \tag{58}$$

We can write the Gradient Descent update rule based on the acquired gradient in equation 58.

$$\begin{aligned}\underline{w}^{t+1} &= \underline{w}^t - \rho\nabla_{\underline{w}}\hat{J}(\underline{w}^t) \\ &= \underline{w}^t - \rho H(\underline{w}^t - \underline{w}^*) \\ &= \underline{w}^t + \underline{w}^* - \underline{w}^* - \rho H(\underline{w}^t - \underline{w}^*) \\ &= (I-\rho H)(\underline{w}^t - \underline{w}^*) + \underline{w}^*\end{aligned} \tag{59}$$

After τ Gradient Descent update and using Singular-Value Decomposition, equation 59 can be rewritten as equation 60.

$$\begin{aligned}\underline{w}^\tau &= (I-\rho H)^\tau(\underline{w}^0 - \underline{w}^*) + \underline{w}^* \\ &= (QQ^T - \rho Q\lambda Q^T)^\tau(\underline{w}^0 - \underline{w}^*) + \underline{w}^* \\ &= (Q(I-\rho Q\lambda)Q^T)^\tau(\underline{w}^0 - \underline{w}^*) + \underline{w}^* \\ &= Q(I-\rho Q\lambda)^\tau Q^T(\underline{w}^0 - \underline{w}^*) + \underline{w}^*\end{aligned} \tag{60}$$

Let's investigate the matrix $(I - \rho Q\lambda)^\tau$ in more detail; each entry on the main diagonal of this matrix has form of $(1 - \rho\lambda_i)^\tau$. Let's consider the two below cases:

- $\rho\lambda_i$ is small with respect to $1 : (1 - \rho\lambda_i)^\tau \to 1$.
- $\rho\lambda_i$ is large with respect to $1 : (1 - \rho\lambda_i)^\tau \to 0$.

Very interesting! It means training the neural network for large number of epochs leads to searching only alongside insignificant directions in the optimization landscape while searching alongside significant directions is stopped and it means the model is learning sample-specific details which do not lead to better generalization and model is over-fitted.

References