ECE/CS 559 - Introduction to Supervised Learning

Erdem Koyuncu

1 Introduction

The output of the neural network will depend on the weights $\mathbf{w}$ of the network and the input signals/vector $\mathbf{x}$ (in addition to the activator function, network topology, etc). Nevertheless, we may simply write $\mathbf{y} = f(\mathbf{w}, \mathbf{x})$ as the network input-output relationship, where f is a certain non-linear function that depends on the parameters discussed above.

Our general goal is to find the appropriate weights such that the network can accomplish a certain desired task. One example is the perceptron that we discussed in the previous lectures: We used the perceptron training algorithm to find the appropriate weights so that the perceptron could classify two linearly separable classes. This was one example of learning (actually, supervised learning).

In general, learning is a systematic method to change/update the weights so that the network can accomplish the desired task after a certain number of learning steps.

In a learning task, usually we have a training set $\mathcal{S} = \{\mathbf{x}_i : i = 1, \ldots, |\mathcal{S}|\}$, and we begin with an initial choice of weights (e.g. picked randomly) $\mathbf{w}_1$. If we are extremely lucky, our random initial choice of weights will perform the desired task, and we do not need to do any learning. This is almost always not the case (for any nontrivial application), and we proceed to the first epoch of learning.

At Epoch 1, We begin with training set example $\mathbf{x}_1$, feed it to the network, we get the output $\mathbf{y}_1 = f(\mathbf{x}_1, \mathbf{w}_1)$. Based on the output $\mathbf{y}_1$ and our specific learning method, we change/update the weights to $\mathbf{w}_2 = \mathbf{w}_1 + \Delta\mathbf{w}_1$. We then proceed to the second training set element $\mathbf{x}_2$, we get the output with the updated weights $\mathbf{y}_2 = f(\mathbf{x}_2, \mathbf{w}_2)$ and find the new weights as $\mathbf{w}_3 = \mathbf{w}_2 + \Delta\mathbf{w}_2$. We keep updating weights until all members of the training set are exhausted.

The weights we obtain at the end of epoch 1 may be able to correctly perform the desired task. If this is the case, the learning may end. Otherwise, we continue to the second epoch of learning, and keep updating the weights in the same manner as described in Epoch 1.

The expectation is that after a certain number of epochs, the weights will converge and they will be able to perform our desired task. In some scenarios, such as the perceptron training algorithm, the learning method provably converges under the assumption of linearly separable classes. In general, the learning process may not converge (In fact, the performance we obtain after each epoch of learning may degrade), in which case we need to stop the learning/training. One can then retry with a different set of initial weights, a different learning method, or change the parameters (e.g. learning rate) of the learning process.

There are basically three different types of learning:

1. Supervised learning: For each pattern (member of the training set), we know what the correct output should be, and we update the weights accordingly.

2. Unsupervised learning: We do not have any correct output information. The learning algorithm finds (by itself) similarities between different training samples, and classifies them accordingly (usually this classification is called "clustering" when talking about unsupervised learning). In certain contexts, the process is also referred to as self-organization.

3. Reinforcement learning: We may think of $\mathbf{y}_1, \mathbf{y}_2, \ldots$ (the outputs) as the actions we choose (e.g. moves of a chess game). Each action/output will incur a different reward (some outputs may be good/some may be bad, chess piece taken good, queen lost bad), and a new different observation of the environment (the new chess board configuration after the opponent also makes his move - note that this may thus be random). Depending on the rewards and the new observation, we update the weights.

In terms of how we update the weights, there are basically two different types:

- Stepwise learning: One input is given, output is observed and the weights are updated. Then, for the next input, the last updated weights are used (as in the description above or the perceptron training algorithm). In this case, there is no need to store the weight increments. This is also called on-line learning.
- Batch learning: One input is given, necessary weight increments are found, but the weights are not updated. When the next input is given, previous weights are used, and again the necessary weight increments are found. This process is repeated till the end of an epoch. At the end of epoch, the weight increments are added to find the total increment, and added to the weights used within the epoch. This is sometimes called as off-line learning.

  In other words, in offline learning, at Epoch 1, We begin with training set example $\mathbf{x}_1$, feed it to the network, we get the output $\mathbf{y}_1 = f(\mathbf{x}_1, \mathbf{w}_1)$. We then proceed to the second training set element $\mathbf{x}_2$, we get the output with the original weights $\mathbf{y}_2 = f(\mathbf{x}_2, \mathbf{w}_1)$ and so on until all members of the training set are exhausted, i.e. $\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{w}_1)$, $i = 1, \ldots, |\mathcal{S}|$. We now update the weights $\mathbf{w}_1$ to $\mathbf{w}_2$ depending on our observations $\mathbf{y}_i$s and the inputs $\mathbf{x}_i$s. Usually, the update is of the form $\mathbf{w}_2 = \mathbf{w}_1 + \sum_{i=1}^{n} \delta(\mathbf{x}_i, \mathbf{y}_i)$, where δ is some function that depends on the specific learning method. We then proceed to a new Epoch, and so on.

First, we focus on supervised learning.

2 Supervised Learning

2.1 Formulation

As we have mentioned before in Section 3 of Lecture 3, we may think of learning as an approximation problem, where we wish to approximate the function $h : \mathbb{R}^n \to \mathbb{R}^m$. In supervised learning we have the training set $\mathcal{S} = \{\mathbf{x}_i : i = 1, \ldots, |\mathcal{S}|\}$, and the samples (desired outputs) of the function $\{h(\mathbf{x}_i) : i = 1, \ldots, |\mathcal{S}|\}$ are available. The learning problem is then to find the weights $\mathbf{w}$ such that $f(\mathbf{w}, \mathbf{x})$ (the output of our neural network) is as close to $h(\mathbf{x})$ as possible, at least over the training set.

The mathematical way to describe closeness is via the notion of a **metric**.

A metric ρ on a set X is a function $\rho : X \times X \to [0, \infty)$ that satisfies the following properties for every $x, y, z \in X$:

1. Non-negativity: $\rho(x, y) \geq 0$.
2. Identity of indiscernibles: $\rho(x, y) = 0 \implies x = y$.
3. Symmetry: $\rho(x, y) = \rho(y, x)$.
4. Subadditivity/Triangle inequality: $\rho(x, y) \leq \rho(x, z) + \rho(y, z)$.

A metric can be thought as a distance function. For $X = \mathbb{R}^n$, with $\mathbf{y} = [y_1 \cdots y_n]^T$ and $\mathbf{x} = [x_1 \cdots x_n]^T$, some examples include:

- Euclidean metric: $\rho(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$.
- The L^p-metric, where $p \geq 1$: $\rho(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$.
- The L^∞-metric: $\rho(\mathbf{x}, \mathbf{y}) = \max_{i \in \{1, \ldots, n\}} |x_i - y_i|$.

A popular choice is the Euclidean metric as it is often analytically tractable with nice properties. Note that then, $\rho^2(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} (x_i - y_i)^2 = (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})$.

In any case, the learning problem is to find an optimal vector / matrix of weights $\mathbf{w}^\star$ such that $\rho(h(\mathbf{x}), f(\mathbf{x}, \mathbf{w}^\star)) \leq \rho(h(\mathbf{x}), f(\mathbf{x}, \mathbf{w}))$ for every $\mathbf{x} \in \mathcal{S}$ and $\mathbf{w}$.

Often, the errors are averaged out over the training samples so that defining (for the Euclidean metric)

$$E(\mathbf{w}) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \|f(\mathbf{x}, \mathbf{w}) - h(\mathbf{x})\|^2$$

the goal of learning is to find an optimal vector / matrix of weights $\mathbf{w}^\star$ such that $E(\mathbf{w}^\star) \leq E(\mathbf{w})$ for every $\mathbf{w}$.

2.2 An exact solution

The linear case can be solved exactly: Suppose we have m input neurons and n output neurons without any bias, and the activator function is identity. Let $\mathbf{W} \in \mathbb{R}^{n \times m}$ denote the weight matrix with the weight going to output neuron i from neuron j denoted by w_{ij}. Then, given input $\mathbf{x} \in \mathbb{R}^{m \times 1}$, the output is $\mathbf{y} = \mathbf{W}\mathbf{x}$. So, if the desired outputs are $\mathbf{d}_i$, $i = 1, \ldots, |\mathcal{S}|$ given patterns $\mathbf{x}_i$, $i = 1, \ldots, |\mathcal{S}|$, we have to minimize

$$E(\mathbf{W}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \|\mathbf{d}_i - \mathbf{W}\mathbf{x}_i\|^2$$

or letting $\mathbf{D} = [\mathbf{d}_1 \cdots \mathbf{d}_{|\mathcal{S}|}]$, $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_{|\mathcal{S}|}]$, we have

$$E(\mathbf{W}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \|\mathbf{d}_i - \mathbf{W}\mathbf{x}_i\|^2 = \frac{1}{|\mathcal{S}|} \|\mathbf{D} - \mathbf{W}\mathbf{X}\|^2$$

Definition: Let X be a real $m \times q$ matrix. There exists a unique $q \times m$ matrix X^+ with the following properties: (i) $XX^+X = X$, (ii) $X^+XX^+ = X^+$, and (iii) X^+X and XX^+ are (Hermitian) symmetric matrices. The matrix X^+ is called the pseudo-inverse, or the Moore-Penrose pseudo-inverse of X.

If X has linearly independent columns, then $A^+ = (A^TA)^{-1}A^T$.

If X has linearly independent rows, then $A^+ = A^T(AA^T)^{-1}$.

For complex case, replace all transposes with Hermitian transpose, and all symmetrixes with Hermitian symmetrixes.

Proposition: Let $X \in \mathbb{R}^{m \times q}$, $Y \in \mathbb{R}^{n \times q}$. The $n \times m$ minimizer of $\|Y - WX\|^2$ is YX^+.

Proof: For any matrix A, note that $\|A\|^2 = \mathrm{tr}(AA^T)$. We have

$$\begin{aligned}
\|YX^+X - WX\|^2 &= \mathrm{tr}\Big[(YX^+X - WX)(YX^+X - WX)^T\Big] \\
&= \mathrm{tr}\Big[YX^+X(X^+X)^TY^T \\
&\quad - YX^+XX^TW^T \\
&\quad - WX(X^+X)^TY^T \\
&\quad + WXX^TW^T\Big]
\end{aligned}$$

Use the fact that XX^+ is symmetric for the first term so that $(X^+X)^T = X^+X$, giving $X^+X(X^+X)^T = X^+XX^+X = X^+X$, the last equality is from (ii) of the properties of pseudoinverse. For the second term, use the fact that $X^+XX^T = (X^+X)^TX^T = (XX^+X)^T = X^T$, the last equality is from property (i) of pseudoinv. For the third term, we have $X(X^+X)^T = XX^+X = X$. Thus,

$$\begin{aligned}
&\|YX^+X - WX\|^2 \\
&= \mathrm{tr}\Big[YX^+XY^T - YX^TW^T - WXY^T + WXX^TW^T\Big].
\end{aligned}$$

On the other hand, again using $\|A\|^2 = \mathrm{tr}(AA^T)$, we obtain

$$\|Y - WX\|^2 = \mathrm{tr}\Big[YY^T - YX^TW^T - WXY^T + WXX^TW^T\Big]$$

The two equalities give

$$\|YX^+X - WX\|^2 = \|Y - WX\|^2 + \mu$$

where μ depends only on X and Y. Hence the optimal solution that minimizes $\|Y - WX\|^2$ should also minimize $\|YX^+X - WX\|^2 = \|(YX^+ - W)X\|^2$. One solution is clearly $W = YX^+$. ■

Hence, one minimizer of $E(\mathbf{W})$ in this case is given by $\mathbf{DX}^+$.

2.3 Nonlinear, nonconvex optimization

Unfortunately, most realistic problems require a non-linear non-convex approach, i.e. in general $E(\mathbf{W})$ is non-linear non-convex in $\mathbf{W}$ (e.g. when we utilize a non-linear activator function). Our general goal is thus to find the global minima of a certain energy function $E(\cdot)$ that may be non-linear, non-convex etc, say, for a single-layer network with a non-linear activation function

$$E(\mathbf{W}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \|\mathbf{d}_i - \phi(\mathbf{W}\mathbf{x}_i)\|^2$$

with the understanding that $\phi(\cdot)$ is applied component-wise. We visit some well-known solution methods in this context.

2.4 Gradient descent

We ask ourselves, if we are currently at position $\mathbf{w}$ with energy $E(\mathbf{w})$, which position we have to move so that the energy $E(\cdot)$ is decreased. To answer this, define the gradient operator

$$\nabla \triangleq \left[\frac{\partial}{\partial w_1} \cdots \frac{\partial}{\partial w_n}\right]^T$$

so that letting $\mathbf{g} \triangleq \nabla E(\mathbf{w})$, we obtain via Taylor series

$$E(\mathbf{w} + \Delta\mathbf{w}) = E(\mathbf{w}) + \mathbf{g}^T \Delta\mathbf{w} + O(\|\Delta\mathbf{w}\|^2).$$

Hence, if we move just a little, i.e. if $\Delta\mathbf{w}$ is small the new energy is approximately,

$$E(\mathbf{w} + \Delta\mathbf{w}) \simeq E(\mathbf{w}) + \mathbf{g}^T \Delta\mathbf{w}.$$

Choosing $\Delta\mathbf{w} = -\eta\mathbf{g}$ for some constant $\eta > 0$, we obtain

$$E(\mathbf{w} + \Delta\mathbf{w}) \simeq E(\mathbf{w}) - \eta\|\mathbf{g}\|^2$$

so that the new position $\mathbf{w} + \delta\mathbf{w}$ indeed results in a lesser energy, provided that η is small. The update $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla E(\mathbf{w})$ is called the gradient descent.

Intuitively, the gradient $\mathbf{g}$ describes the direction where $E(\mathbf{w})$ increases the fastest. Hence, we move in the negative direction, which is the direction where $E(\mathbf{w})$ decreases the fastest.

The method of steepest descent converges to the optimal solution slowly, as the step size decreases as one approaches the optimal solution. Also, if η is small, the transient response of the algorithm is overdamped, in that the trajectory traced by $\mathbf{w}$ follows a smooth path (although if η is too small, then the convergence may take many iterations).

When η is large, the transient response of the algorithm is underdamped, in that the trajectory of $\mathbf{w}$ follows a zigzagging (oscillatory) path.

When η exceeds a certain critical value, the algorithm may become unstable, i.e. $\mathbf{w}$ diverges to infinity.

2.5 Newton's method

Define the Hessian

$$\nabla^2 \triangleq \begin{bmatrix} \frac{\partial^2 E(\mathbf{w})}{\partial w_1^2} & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_1 \partial w_n} \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_2^2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\mathbf{w})}{\partial w_n \partial w_1} & \frac{\partial^2 E(\mathbf{w})}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 E(\mathbf{w})}{\partial w_n^2} \end{bmatrix},$$

so that letting $\mathbf{g} \triangleq \nabla E(\mathbf{w})$ and $\mathbf{H} \triangleq \nabla^2 E(\mathbf{w})$, we obtain, again by Taylor series,

$$\begin{aligned} E(\mathbf{w} + \Delta\mathbf{w}) &= E(\mathbf{w}) + \mathbf{g}^T \Delta\mathbf{w} + \frac{1}{2}(\Delta\mathbf{w})^T \mathbf{H} \Delta\mathbf{w} + O(\|\Delta\mathbf{w}\|^3) \\ &\simeq E(\mathbf{w}) + \mathbf{g}^T \Delta\mathbf{w} + \frac{1}{2}(\Delta\mathbf{w})^T \mathbf{H} \Delta\mathbf{w}. \end{aligned}$$

We now minimize RHS over all possible choices of $\Delta\mathbf{w}$. The function is of the form $f(x) = x^T Q x + c^T x$. We have

$$\frac{1}{2} x^T Q x + c^T x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i$$

so, taking the derivatives, we obtain

$$\frac{\partial}{\partial x_k} \left(\frac{1}{2} x^T Q x + c^T x \right) = q_{kk} x_k + \frac{1}{2} \sum_{i=1,\, i \neq k}^{n} q_{ik} x_i + \frac{1}{2} \sum_{i=1,\, i \neq k}^{n} q_{ki} x_i + c_k = \sum_{i=1}^{n} q_{ik} x_i$$

and thus $\frac{\partial}{\partial x} \left(\frac{1}{2} x^T Q x + c^T x \right) = Qx + c$. Letting $Qx + c = 0$, we obtain the critical point $x = -Q^{-1}c$, or $\Delta\mathbf{w} = -\mathbf{H}^{-1}\mathbf{g}$ for the original problem. Now, note that the Hessian of our objective function is Q. When the Hessian Q is positive definite, the critical points of the function are local minimum. The only critical point is $x = -Q^{-1}c$. A continuous, twice differentiable function of several variables is convex on a convex set if and only if its Hessian matrix is positive semidefinite on the interior of the convex set. Hence, our function is also convex. Since the function is convex (as shown below), the critical point, which is a local minimum, is also a global minimum.

Usually a learning parameter is also appended, i.e. $\Delta\mathbf{w} = -\eta \mathbf{H}^{-1}\mathbf{g}$. We then obtain

$$\begin{aligned} E(\mathbf{w} + \Delta\mathbf{w}) - E(\mathbf{w}) &\simeq -\eta \mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} + \frac{\eta^2}{2} (\mathbf{H}^{-1}\mathbf{g})^T \mathbf{H} \mathbf{H}^{-1} \mathbf{g} \\ &= -\eta \mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} + \frac{\eta^2}{2} \mathbf{g}^T (\mathbf{H}^{-1})^T \mathbf{g} \\ &= -\eta \mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} + \frac{\eta^2}{2} \left(\mathbf{g}^T (\mathbf{H}^{-1})^T \mathbf{g} \right)^T \\ &= -\left(\eta + \frac{\eta^2}{2} \right) \mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} \\ &\simeq -\eta \mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} \end{aligned}$$

The last expression is greater than or equal to 0 if $\mathbf{H}^{-1}$ is positive definite, so that our method is indeed a descent, if η is also small.

If $\mathbf{H}$ is not positive-definite, usually the following modification is applied $\Delta\mathbf{w} = -\eta\beta(\beta\mathbf{I} + \mathbf{H})^{-1}\mathbf{g}$. As β gets large the "new Hessian" $\beta\mathbf{I} + \mathbf{H}$ is bound to be positive definite. In fact, as $\beta \to \infty$, this type of iteration becomes the same as Gradient descent.

The geometric interpretation of Newton's method is that at each iteration one approximates $E(w)$ by a quadratic function around w_n, and then takes a step towards the maximum/minimum of that quadratic function (in higher dimensions, this may also be a saddle point). Note that if $E(w)$ happens to be a quadratic function, then the exact extremum is found in one step. Indeed if $E(w) = \frac{1}{2}w^TQw + c^Tw$, then gradient is $Qw + c$, and hessian is Q so that starting from $w = b$, we go to the point $b - H^{-1}(Hb + c) = -H^{-1}c$, but this new point is the global minimum.

Generally speaking, Newton's method converges quickly asymptotically and does not exhibit the zigzagging behavior that sometimes characterizes the method of steepest descent. In any case, one limitation of Newton's method is the computational complexity: To implement, the method, we need to calculate Hessian and take its inverse - both are difficult tasks when the dimensionality is large.

2.6 Gauss-Newton method

Used to solve non-linear least-squares problems, i.e. suppose $E(\mathbf{w})$ is of the form $E(\mathbf{w}) = \sum_{i=1}^q e_i^2(\mathbf{w})$ for some functions e_i. Suppose that the dimensionality of $\mathbf{w}$ is n. Then, we have

$$E(\mathbf{w}) = \left\| \begin{bmatrix} e_1(\mathbf{w}) \\ \vdots \\ e_q(\mathbf{w}) \end{bmatrix} \right\|^2$$

so that

$$E(\mathbf{w} + \Delta\mathbf{w}) = \left\| \begin{bmatrix} e_1(\mathbf{w} + \Delta\mathbf{w}) \\ \vdots \\ e_q(\mathbf{w} + \Delta\mathbf{w}) \end{bmatrix} \right\|^2 \simeq \left\| \begin{bmatrix} e_1(\mathbf{w}) + (\nabla e_1(\mathbf{w}))^T\Delta\mathbf{w} \\ \vdots \\ e_q(\mathbf{w}) + (\nabla e_q(\mathbf{w}))^T\Delta\mathbf{w} \end{bmatrix} \right\|^2 \triangleq \|\mathbf{e}(\mathbf{w}) + \mathbf{J}\Delta\mathbf{w}\|^2,$$

where $\mathbf{e}(\mathbf{w}) \triangleq \begin{bmatrix} e_1(\mathbf{w}) \\ \vdots \\ e_q(\mathbf{w}) \end{bmatrix}$, and $\mathbf{J} \triangleq \begin{bmatrix} (\nabla e_1(\mathbf{w}))^T \\ \vdots \\ (\nabla e_q(\mathbf{w}))^T \end{bmatrix}$ is the $q \times n$ Jacobian.

Now recall that the minimizer of $\|\mathbf{D} - \mathbf{W}\mathbf{X}\|^2$ was $\mathbf{D}\mathbf{X}^+$. Equivalently, the minimizer of $\|\mathbf{D}^T - \mathbf{X}^T\mathbf{W}^T\|^2$ is $\mathbf{D}\mathbf{X}^+$. Equivalently, the minimizer of $\|\mathbf{D} - \mathbf{X}\mathbf{W}^T\|^2$ is $\mathbf{D}^T(\mathbf{X}^T)^+$. One can easily show that $(\mathbf{X}^T)^+ = (\mathbf{X}^+)^T$. Hence, the minimizer of $\|\mathbf{D} - \mathbf{X}\mathbf{W}\|^2$ is $(\mathbf{D}^T(\mathbf{X}^T)^+)^T = \mathbf{X}^+\mathbf{D}$

Using this result, minimizing the linear approximation of $E(\mathbf{w})$ over all possible $\Delta\mathbf{w}$, we obtain the optimal descent $(\Delta\mathbf{w})_\star = (-\mathbf{J})^+\mathbf{e} = -\mathbf{J}^+\mathbf{e}(\mathbf{w})$. Note that usually q (the number of observations) is much larger than n, so that $\mathbf{J} \in \mathbb{R}^{q\times n}$ (often) has linearly independent columns. In such a scenario, we have $\mathbf{J}^+ = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$, and the updates are given by $(\Delta\mathbf{w})_\star = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{e}$.

In certain cases, $\mathbf{J}$ may still not have linearly-independent columns, in which case one can use the update $(\Delta\mathbf{w})_\star = -(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\mathbf{e}(\mathbf{w})$, where δ is "regularization parameter." As $\delta \to \infty$, we obtain gradient descent over each component function $e_i(\mathbf{w})$.

Remark - 1: Note that the solution $(\Delta\mathbf{w})_\star = -(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\mathbf{e}(\mathbf{w})$ is the minimizer of the regularized mean squared problem

$$\|\mathbf{e}(\mathbf{w}) + \mathbf{J}\Delta\mathbf{w}\|^2 + \lambda\|\Delta\mathbf{w}\|^2.$$

To see this, we can write the objective function of the regularized mean-squares problem as

$$\left\| \begin{bmatrix} e(\mathbf{w})_{q\times 1} \\ \mathbf{0}_{n\times 1} \end{bmatrix} + \begin{bmatrix} \mathbf{J}_{q\times n} \\ \sqrt{\lambda}\mathbf{I}_{n\times n} \end{bmatrix} \Delta\mathbf{w}_{n\times 1} \right\|^2$$

The solution can be shown to be (using the Proposition in Section 3.2):

$$\left(\begin{bmatrix} \mathbf{J} \\ \sqrt{\lambda}\mathbf{I} \end{bmatrix}^T \begin{bmatrix} \mathbf{J} \\ \sqrt{\lambda}\mathbf{I} \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{J} \\ \sqrt{\lambda}\mathbf{I} \end{bmatrix}^T \begin{bmatrix} e(\mathbf{w}) \\ \mathbf{0} \end{bmatrix} = (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T e(\mathbf{w}).$$

The regularization ensures that the next step is not too far from the previous step (when λ is large).

Remark - 2: Derivation from Newton's method:

Recall that in Newton's method we had the iterations: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}\mathbf{g}$. In particular, for the function $E(\mathbf{w}) = \sum_{i=1}^{q} e_i^2(\mathbf{w})$, we can calculate

$$\mathbf{g} = \left[2\sum_{i=1}^{q} e_i \frac{\partial e_i}{\partial w_1} \quad \cdots \quad 2\sum_{i=1}^{q} e_i \frac{\partial e_i}{\partial w_n} \right]^T = 2\mathbf{J}^T\mathbf{e}(\mathbf{w}),$$

$$\mathbf{H}_{jk} = 2\sum_{i=1}^{q} \left(\frac{\partial e_i}{\partial w_k}\frac{\partial e_i}{\partial w_j} + e_i \frac{\partial e_i^2}{\partial w_j \partial w_k} \right) \simeq 2\sum_{i=1}^{q} \frac{\partial e_i}{\partial w_k}\frac{\partial e_i}{\partial w_j} = 2(\mathbf{J}^T\mathbf{J})_{j,k}$$

so the update rule is $\mathbf{w} \leftarrow \mathbf{w} - \left(2(\mathbf{J}^T\mathbf{J})\right)^{-1} 2\mathbf{J}^T\mathbf{e}(\mathbf{w})$ same as before.

The approximation "holds" if $\frac{\partial e_i}{\partial w_k}\frac{\partial e_i}{\partial w_j}$ is large relative to $e_i \frac{\partial e_i^2}{\partial w_j \partial w_k}$ in magnitude. Hence, the terms e_i should be small or/and the functions e_i should be close to being linear (When they are linear functions of $\mathbf{w}$, the approximation holds as the other terms are 0). In this context, if the function to be optimized is a quadratic function, that this rule will converge to the globally optimal solution in one iteration. Actually, then the $\mathbf{H}$ approximation becomes valid, the rule boils down to Newton's rule, which gives the optimal solution for quadratic functions in one step.

2.7 Supervised Learning Rules

We now have an arsenal of different optimization methods available at our disposal. Throughout the course, we shall mostly utilize however the simple gradient descent idea. Gradient descent applied to different scenarios are given different names including Delta Learning, Widrow-Hoff Rule, the LMS algorithm Backpropagation, etc. We begin with Widrow-Hoff learning.

2.8 Widrow-Hoff Learning/The LMS algorithm

Consider an identity activation function, one output neuron, m input neurons, and a training sequence of length $|\mathcal{S}|$ as usual. Given $\mathbf{w} \in \mathbb{R}^{m\times 1}$, We wish to minimize

$$E(\mathbf{w}) = \sum_{i=1}^{|\mathcal{S}|} (d_i - \mathbf{w}^T\mathbf{x}_i)^2$$

We already know the optimal $\mathbf{w}$ here. It is given by $\mathbf{w}_\star^T = \mathbf{d}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$, where $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_{|\mathcal{S}|}]$.

Alternatively, we may use gradient descent,

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -2\sum_{i=1}^{|\mathcal{S}|} (d_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

so that we shall utilize the update

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{i=1}^{|\mathcal{S}|} (d_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

In terms of learning, we thus begin with an initial weight. At epoch 1, We apply pattern $\mathbf{x}_1$, observe the output $\mathbf{w}^T\mathbf{x}_i$, do not change the weights, apply pattern $\mathbf{x}_2$, observe the outputs, do not change the weights, and so on until all training patterns are exhausted. We can then calculate the descent factor as above an update our weights accordingly at the end of the epoch (this has been called offline or batch learning remember).

For practical reasons, the update is also often done for each sample via

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(d_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

Note that this is shorthand notation for

$$\mathbf{w}(i+1) = \mathbf{w}(i) + \eta\left(d_i - [\mathbf{w}(i)]^T\mathbf{x}_i\right)\mathbf{x}_i,\ i = 1, \ldots, |\mathcal{S}|$$

with some initial condition $\mathbf{w}(1)$.

Note how this expression is very similar to the PTA. This is called the Widrow-Hoff learning rule, or the LMS algorithm. Note that this is not a true gradient descent, and may not converge to the globally optimal solution (unlike the true gradient descent does when η is taken sufficiently low). In fact, the solution will move randomly around the globally optimal solution (there is a nice theory that formalizes these ideas - but we will likely not discuss it in this course).

2.9 Delta Rule

This can be considered to be a generalization of Widrow-Hoff rule to an arbitrary (differentiable) activation function. Here, we wish to minimize

$$E(\mathbf{w}) = \sum_{i=1}^{|\mathcal{S}|}(d_i - \phi(\mathbf{w}^T\mathbf{x}_i))^2$$

Again, using the gradient descent idea, we obtain

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = -2\sum_{i=1}^{|\mathcal{S}|}(d_i - \phi(\mathbf{w}^T\mathbf{x}_i))\phi'(\mathbf{w}^T\mathbf{x}_i)x_{ij}$$

so that the gradient descent relies on the update so that we shall utilize the update

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\sum_{i=1}^{|\mathcal{S}|}(d_i - \phi(\mathbf{w}^T\mathbf{x}_i))\phi'(\mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

Again, for practical reasons, the update is usually done per-sample (online learning) via

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(d_i - \phi(\mathbf{w}^T\mathbf{x}_i))\phi'(\mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i$$

This is called the Delta rule. For a less cumbersome formula, if $v_i = \mathbf{w}^T\mathbf{x}_i$ denotes the induced local field with input $\mathbf{x}_i$, and $y_i = \phi(v_i)$ is the corresponding output, we have

$$\mathbf{w}(i+1) = \mathbf{w}(i) + \eta(d_i - y_i)\phi'(v_i)\mathbf{x}_i,\ i = 1,\ldots,|\mathcal{S}|.$$

Calculation of ϕ' may be problematic in general. But, for some certain cases, the calculation is easier. For example,

- Recall
$$\tanh(\alpha x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}}$$
.
$$\begin{aligned}\frac{\partial \beta\tanh(\alpha x)}{\partial x} &= \beta\frac{\alpha(e^{\alpha x} + e^{-\alpha x})^2 - \alpha(e^{\alpha x} - e^{-\alpha x})^2}{(e^{\alpha x} + e^{-\alpha x})^2} \\ &= \beta\alpha(1 - \tanh^2(\alpha x)) = \frac{\alpha}{\beta}(\beta^2 - \beta^2\tanh^2(\alpha x))\end{aligned}$$
so in this case, the update is simply (since $y_i = \beta\tanh(\alpha v_i)$ by definition)
$$\mathbf{w}(i+1) = \mathbf{w}(i) + \eta\frac{\alpha}{\beta}(d_i - y_i)(\beta^2 - y_i^2)\mathbf{x}_i,\ i = 1,\ldots,|\mathcal{S}|.$$

- Or let $y = \phi(x) = \frac{\beta}{1+e^{-\alpha x}}$.
$$\frac{\partial\phi(x)}{\partial x} = \frac{\alpha\beta e^{-\alpha x}}{(1+e^{-\alpha x})^2} = \frac{\alpha}{\beta}e^{-\alpha x}y^2 = \frac{\alpha}{\beta}\left(\frac{\beta}{y} - 1\right)y^2 = \frac{\alpha}{\beta}y(\beta - y)$$
so the update is
$$\mathbf{w}(i+1) = \mathbf{w}(i) + \eta\frac{\alpha}{\beta}(d_i - y_i)y_i(\beta - y_i)\mathbf{x}_i,\ i = 1,\ldots,|\mathcal{S}|.$$