COMP4211 2023-24 Spring
Machine Learning

T. Yeung
scyeungaf@connect.ust.hk

April 5, 2024

1. **Linear Regression**

   (a) We want to minimize the loss

$$\mathscr{L}(\mathbf{w};\mathcal{S}) = \sum_{\ell=1}^{N}(w_0 x_0^{(\ell)} + w_1 x_1^{(\ell)} + \ldots + w_d x_d^{(\ell)} - y^{(\ell)})^2$$

   , where $x_0^{(\ell)} = 1$ for all ℓ. Differentiating the loss with respect to w_i, we have

$$\frac{\partial L}{\partial w_i} = \sum_{\ell=1}^{N} 2(w_0 x_0^{(\ell)} + w_1 x_1^{(\ell)} + \ldots + w_d x_d^{(\ell)} - y^{(\ell)}) x_i^{(l)}$$

   . Setting $\frac{\partial L}{\partial w_i} = 0$, we have

$$\left(\sum_{\ell=1}^{N} x_0^{(\ell)} x_i^{(\ell)}\right) w_0 + \left(\sum_{\ell=1}^{N} x_1^{(\ell)} x_i^{(\ell)}\right) w_1 + \ldots + \left(\sum_{\ell=1}^{N} x_d^{(\ell)} x_i^{(\ell)}\right) w_d = \sum_{\ell=1}^{N} y^{(\ell)} x_i^{(\ell)}$$

   If we let $\mathbf{x}_0 = [x_0^{(1)} x_0^{(2)} \ldots x_0^{(N)}]^T \in \mathbf{R}^{n+1}$ and $\mathbf{w} = [w_0 w_1 \ldots w_d]^T \in \mathbf{R}^{d+1}$, $\mathbf{y} \in \mathbf{R}^{n+1} = [y_0 y_1 ... y_n]^T$, then the above expression can be rewritten as

$$\begin{bmatrix}\mathbf{x}_0 \cdot \mathbf{x}_i & \mathbf{x}_1 \cdot \mathbf{x}_i & \ldots & \mathbf{x}_d \cdot \mathbf{x}_i\end{bmatrix} \mathbf{w} = \sum_{\ell=1}^{N} y^{(\ell)} x_i^{(\ell)}$$

   To put $\frac{\partial L}{\partial w_i} = 0$, where i is from 0 to d. We get

$$\begin{bmatrix}\mathbf{x}_0 \cdot \mathbf{x}_0 & \mathbf{x}_1 \cdot \mathbf{x}_0 & \ldots & \mathbf{x}_d \cdot \mathbf{x}_0 \\ \mathbf{x}_0 \cdot \mathbf{x}_1 & \mathbf{x}_1 \cdot \mathbf{x}_1 & \ldots & \mathbf{x}_d \cdot \mathbf{x}_1 \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_0 \cdot \mathbf{x}_d & \mathbf{x}_1 \cdot \mathbf{x}_d & \ldots & \mathbf{x}_d \cdot \mathbf{x}_d\end{bmatrix} \mathbf{w} = \begin{bmatrix}\mathbf{y} \cdot \mathbf{x}_0 \\ \mathbf{y} \cdot \mathbf{x}_1 \\ \vdots \\ \mathbf{y} \cdot \mathbf{x}_d\end{bmatrix} \tag{1}$$

Let

$$\mathbf{X} = \begin{bmatrix} | & \dots & | \\ \mathbf{x}_0 & \dots & \mathbf{x}_d \\ | & \dots & | \end{bmatrix}$$

. (1) can be rewritten as $\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$. If $\mathbf{X}^T\mathbf{X}$ is invertible, we have $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

(b) The weight update rule is

$$w_i \leftarrow w_i - \mu \frac{\partial L}{\partial w_i}$$

From the previous derivation this is

$$w_i \leftarrow w_i - \mu \sum_{\ell=1}^{N} 2(w_0 x_0^{(\ell)} + w_1 x_1^{(\ell)} + \ldots + w_d x_d^{(\ell)} - y^{(\ell)}) x_i^{(l)}$$

(c) R^2 score is found by

$$R^2 = 1 - \frac{\sum_{\ell=1}^{N} (f(\mathbf{x}^{(\ell)}; \mathbf{w}) - y^{(\ell)})^2}{\sum_{\ell=1}^{N} (\bar{y} - y^{(\ell)})^2}$$

When the squared loss is minimized, i.e. $= 0$ (due to all squared terms). The R^2 score will be $1 - 0 = 1$, which is maximised because the second term in the R^2 score is always positive (due to all squared terms).

2. **Logistic Regression**

(a) In this context, instead of taking the softmax function as the activation function over the output to get a probability distribution, we should take the sigmoid function as the activation function for each output to obtain a probability for each output.

Let $\mathbf{W} = [w_{ki}] \in \mathbf{R}^{K\times(1+d)}$, $\mathbf{r}^{(\ell)} = \sigma(\mathbf{W}\tilde{\mathbf{x}})$, where the sigmoid function is taken elementwise. We can define the likelihood function to be

$$\text{Likelihood} = \prod_{k=1}^{K}\prod_{\ell=1}^{N}(r_k^{(\ell)})^{y_k^{(\ell)}}(1-r_k^{\ell})^{1-y_k^{(\ell)}}$$

This is the product of the likelihood for the sigmoid for each output. The corresponding loss function is

$$\text{Loss} = -\sum_{k=1}^{K}\sum_{\ell=1}^{N} y_k^{(\ell)}\log r_k^{(\ell)} + (1-y_k^{(\ell)})\log(1-r_k^{(\ell)})$$

(b) There will be a total of 2^K classes. Each output in $\mathbf{r} \in \mathbb{R}^K$ can classified as 1 or 0. Therefore we have two possibility for each output. If we want to train it with a multiclass classification where only one class can be true, we need one class for each possibility, hence a total of 2^K classes.

3. **Binary Classification**

(a) When $\beta = 1$, $F_1 = \frac{(1+1)PR}{P+R} = \frac{2PR}{P+R}$

(b) i. $F_{0.5} = \frac{(1+0.5^2)PR}{(0.5)^2P+R} = 1.25\frac{PR}{0.25P+R}$. This put more emphasis on precision because when $P = R$, $F_{0.5}$ increases more when P gets increased rather than R. This means that $F_{0.5}$ penalizes false positive more than false negative.

ii. $F_2 = \frac{(1+2^2)PR}{2^2P+R} = 5\frac{PR}{4P+R}$. This, on other other hand, but more emphasis on recall because when $P = R$, F_2 increases more when R gets increased rather than P. This means that F_2 penalises false negative more than false positive.

(c) i. $\lim_{\beta\to 0} F_\beta = \frac{PR}{R} = P$. It degenerates to precision.

ii. $\lim_{\beta\to\infty} F_\beta = \frac{(1+\beta^2)PR}{\beta^2P+R} = \frac{PR}{P} = R$. It degenerates to recall.

4. **Feedforward Neural Network**

(a) Hyperbolic tangent function has a range of $(-1, 1)$ while sigmoid function has a range of $(1, 0)$. Hyperbolic tangent can make use of the representation power of floating-point value more and won't let negative range go to waste.

(b) $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - 1 = 2\sigma(2x) - 1$

(c) $\frac{d}{dx}(2\sigma(2x) - 1) = 2\frac{d}{dx}\sigma(d(2x)) = 2\frac{d\sigma(2x)}{2x}\frac{d(2x)}{dx} = 4\sigma(2x)(1 - \sigma(2x))$

(d) The weight update rule state that the weight update at the last layer is $\Delta w_{\ell k}^{[3]} = -\mu\frac{\partial L}{\partial w_{\ell k}^{[3]}} = \mu\sum_{q=1}^{N}\delta_{\ell}^{[3](q)}z_{k}^{[2](q)}$. However, since the weights are initialized to large magnitude, $\delta_{\ell}^{[3](q)} = y_{\ell}^{(q)} - z_{\ell}^{[3](q)}$ can be a very large negative number (due to $z_{\ell}^{[3](q)}$ being very larg)e, this leads to the exploding gradient problem at the last layer where the model can be unstable at training.

(e) The weight update rule state that the weight update at the last layer is $\Delta w_{\ell k}^{[3]} = -\mu\frac{\partial L}{\partial w_{\ell k}^{[3]}} = \mu\sum_{q=1}^{N}\delta_{\ell}^{[3](q)}z_{k}^{[2](q)}$. However, since the weights are initialized to zero, and we know that $\tanh(0) = 0$, it follows that $z_{k}^{[2](q)}$ is 0. This leads to vanishing gradient at the last layer during training.

5. **Deep Neural Network**

(a) The rectifier activation function is just $\text{Relu}(x) = \max(x, 0)$, this can be done with a simple branching. Compares to sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ which requires expensive exponentiation and division of floating point number, the rectifier activation function is easier to compute for computers.

(b) i. Batch normalization is applied on a mini-batch. It independently normalizes each feature of a batch of given inputs, in contrast to ordinary data normalization where normalization happens across all features of the same input. To be more specific, if we are given a batch of input of size n with d dimensional features $X \in \mathbf{R}^{n \times d}$, the normalization is done column-wise for the whole batch, whereas for ordinary data normalization, the normalization is done row-wise independently for the whole dataset.

ii. In batch normalization, after making the distribution of each feature zero mean and unit variance, we will scale each feature i by γ_i and add a constant term β_i to it, where γ_i and β_i are learnable parameters. This allows the model to learn the identity function during normalization and prevent the loss of representation power. On the other hand, in ordinary data normalization, we only normalize but do not scale and add constant. There is no learnable parameters in ordinary data normalization.

6. **Feedforward Neural Networks and Convolutional Neural Networks**

   (a) i. The number of learnable parameters is $(784 \times 64 + 64) + (64 \times 64) + 64 + (64 \times 10) + 10 = 55050$

   ii. The number of parameters that are regularized is $784 \times 64 + 64 \times 64 + 64 \times 10 = 54912$

   (b) The number of learnable parameters for each layer is listed below:

$$\text{conv2d} : (3 \times 3 \times 3 + 1) \times 32 = 896$$
$$\text{conv2d\_1} : (3 \times 3 \times 32 + 1) \times 64 = 18496$$
$$\text{dense} : (2304 \times 10 + 10) = 23050$$

   The layers not listed above has 0 learnable parameters.

7. **Recurrent Neural Network**

(a) Here is the unfolded representation of the recurrent neural network:

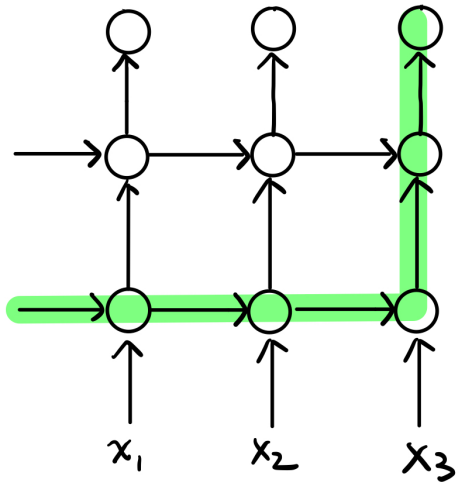

Figure 1: Unfolded representation of the recurrent neural network

(b) The highlighted path is the longest feedforward path. It has 5 layers of processing units.

8. **Principal Component Analysis**

(a) The mean vector $\mu = \begin{pmatrix} \frac{2+3+4+6+7+8}{6} \\ \frac{1+5+3+6+5+10}{6} \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$

(b) $\mathcal{S} = \left\{ \begin{pmatrix} -3 \\ -4 \end{pmatrix}, \begin{pmatrix} -2 \\ -0 \end{pmatrix}, \begin{pmatrix} -1 \\ -2 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 5 \end{pmatrix} \right\}$

(c)

$$\begin{aligned} \Sigma &= \frac{1}{6}\left(\begin{pmatrix} 9 & 12 \\ 12 & 16 \end{pmatrix} + \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \right. \\ &\quad \left. + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 9 & 15 \\ 15 & 25 \end{pmatrix} \right) \\ &= \frac{1}{6} \begin{pmatrix} 28 & 30 \\ 30 & 46 \end{pmatrix} \\ &= \frac{1}{3} \begin{pmatrix} 14 & 15 \\ 15 & 23 \end{pmatrix} \end{aligned}$$

(d) The characteristic equation for Σ is $(14-\lambda)(23-\lambda)-15^2 = 0$, solving gives $\lambda_1, \lambda_2 = \frac{37}{2} \pm 3\frac{\sqrt{109}}{2} = 2.84, 34.2$

(e) The PoV is given by $\frac{34.2}{2.84+34.2} = 0.92$