COMP4211

Machine Learning

Larry128

A summary notes for revision

Fall 2024-2025

Contents

1 Linear Regression

1.1 Basic Ideas of Regression

1. Given a training set $S = \{(x^{(l)}, y^{(l)})\}_{l=1}^{N}$ of N labelled examples of input-output pairs.
2. A **Regression Function** $f(\mathbf{x}; \mathbf{w})$ uses S such that the predicted output $f(\mathbf{x^{(l)}}; \mathbf{w})$ for each input $\mathbf{x}^l$ such that $f(\mathbf{x}^{(l)}; \mathbf{w}) \approx \mathbf{y}^l$.
3. (multi-output regression) When the output $\mathbf{y}$ is a vector, it's a multi-output regression.
4. We denote the output by y if the output is univariate.
5. The input $\mathbf{x} = (x_1, \cdots, x_d)^T$ is d-dimensional.

1.2 Linear Regression Function

1. If the regression function is linear, then

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}) &= w_0 + w_1 x_1 + \cdots + w_d x_d \\ &= \begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \qquad = \begin{bmatrix} 1 & x_1 & \cdots & x_d \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \\ &= \mathbf{w}^T \tilde{\mathbf{x}} \qquad\qquad\qquad\qquad\qquad = \tilde{\mathbf{x}}^T \mathbf{w} \end{aligned}$$

2. w_0 is the *bias* term which serves as an offset.
3. The learning problem is to find the best $\mathbf{w}$ according to performance measure on S.

1.3 Loss Function

1. A common way to learn the parameter $\mathbf{w}$ of $f(\mathbf{x}; \mathbf{w})$ is to define a loss function $L(\mathbf{w}; S)$
2. The most common loss function is the **squared loss**

$$\begin{aligned} L(\mathbf{w}; S) &= \sum_{l=1}^{N} (f(\mathbf{x}^{(l)}; \mathbf{w}) - \mathbf{y}^{(l)})^2 \\ &= \sum_{l=1}^{N} (w_0 + w_1 x_1^{(l)} + \cdots + w_d x_d^{(l)} - y^{(l)})^2 \end{aligned}$$

3. We may also define the loss function by **mean** rather than the sum

$$L(\mathbf{w}; S) = \frac{1}{N} \sum_{l=1}^{N} (f(\mathbf{x}^{(l)}; \mathbf{w}) - \mathbf{y}^{(l)})^2$$

4. A special case ($d = 1$)
   Squared loss:

$$L(\mathbf{w}; S) = \sum_{l=1}^{N} (w_0 + w_1 x_1^{(l)} - y^{(l)})^2$$

   We can find the unique optimal solution $\tilde{\mathbf{w}} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ that minimizes $L(\mathbf{w}; S)$ using the method of least squares.

First, we take the derivatives of $L(\mathbf{w}; S)$ with respect to w_0 and w_1 and set them to 0.

$$\frac{\partial L}{\partial w_0} = 2\sum_{l=1}^{N}(w_0 + w_1 x_1^{(l)} - y^{(l)}) = 0 \iff \sum_{l=1}^{N}(w_0 + w_1 x_1^{(l)}) = \sum_{l=1}^{N} y^{(l)} \iff Nw_0 + \sum_{l=1}^{N} w_1 x_1^{(l)} = \sum_{l=1}^{N} y^{(l)}$$

$$\frac{\partial L}{\partial w_1} = 2\sum_{l=1}^{N}(w_0 + w_1 x_1^{(l)} - y^{(l)})x_1^{(l)} = 0 \iff w_0\sum_{l=1}^{N} x_1^{(l)} + w_1\sum_{l=1}^{N}(x_1^{(l)})^2 = \sum_{l=1}^{N} x_1^{(l)} y^{(l)}$$

Then, we have a system of linear equations of two unknown w_0, w_1. We can write it in matrix form.

$$\mathbf{Aw} = \begin{bmatrix} N & \sum_l x_1^l \\ \sum_l x_1^l & \sum_l (x_1^l)^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} \sum_l y^{(l)} \\ \sum_l x_1^{(l)} y^{(l)} \end{bmatrix} = \mathbf{b}$$

Assuming $\mathbf{A}$ is invertible, the least squares estimate is

$$\tilde{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{b}$$

5. General case ($d \geq 1$)

   (a) (First approach) We express the input and output of N examples as follows

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^l & \cdots & x_d^1 \\ 1 & x_1^2 & \cdots & x_d^2 \\ \vdots & & & \\ 1 & x_1^N & \cdots & x_d^N \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

   Then we can express the matrix form as follows (proof skipped)

$$\mathbf{Aw} = \mathbf{X^T X w} = \mathbf{X^T y} = \mathbf{b}$$

   Therefore, the least squares estimate is

$$\tilde{\mathbf{w}} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}$$

   , assuming $\mathbf{X^T X}$ is invertible

   (b) (Second approach) First write $\mathbf{Xw} - \mathbf{y}$ as

$$\begin{aligned} \mathbf{Xw} - \mathbf{y} &= \begin{bmatrix} 1 & x_1^l & \cdots & x_d^1 \\ 1 & x_1^2 & \cdots & x_d^2 \\ \vdots & & & \\ 1 & x_1^N & \cdots & x_d^N \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ y_d \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} \\ &= \begin{bmatrix} w_0 + w_1 x_1^{(1)} + \cdots + w_d x_d^1 - y^{(1)} \\ w_0 + w_1 x_1^{(2)} + \cdots + w_d x_d^2 - y^{(2)} \\ \vdots \\ w_0 + w_1 x_1^{(N)} + \cdots + w_d x_d^N - y^{(N)} \end{bmatrix} \end{aligned}$$

   Then the squared loss is just the square of **L-2 norm** of $\mathbf{Xw} - \mathbf{y}$

$$L(\mathbf{w}; S) = ||\mathbf{Xw} - \mathbf{y}||^2$$

   We can further write the squared loss as

$$\begin{aligned} L(\mathbf{w}; S) &= ||\mathbf{Xw} - \mathbf{y}||^2 \\ &= (\mathbf{Xw} - \mathbf{y})^T(\mathbf{Xw} - \mathbf{y}) \\ &= (\mathbf{w^T X^T} - \mathbf{y^T})(\mathbf{Xw} - \mathbf{y}) \\ &= \mathbf{w^T X^T X w} - \mathbf{2 y^T X w} + \mathbf{y^T y} \end{aligned}$$

After that, we can take the derivative with respect to $\mathbf{w}$

$$\begin{aligned} & \frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{X^T X w} - 2\mathbf{X^T y} = 0 \\ \iff & \mathbf{X^T X w} = \mathbf{X^T y} \\ \iff & \tilde{\mathbf{w}} = (\mathbf{X^T X})^{-1}\mathbf{X^T y} \end{aligned}$$

(c) Complexity considerations
To compute $\tilde{\mathbf{w}}$, we need to invert $\mathbf{X^T X} \in \mathbb{R}^{(d+1)\times(d+1)}$. *LeGall* is the fastest algorithm to compute that with $O(n^{2.3728639})$, instead of $O(n^3)$ for Cholesky, LU, Gaussian elimination.

1.4 Non-linear Extensions

(a) For solving more complicated problems, non-linear regression function are needed.

(b) Different approaches for non-linear extension are:

i. Explicitly adding more input dimensions (which depend non-linearly on the original input dimensions) and applying linear regress to the expanded input. Here is an example

$$f(\mathbf{x}; \mathbf{w}) = w_0 \cdot 1 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_{d+1} x_1^2 x_8^3 + w_{d+2} x_{10} x_{19}$$

In this case,

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1^2 x_8^3 \\ x_{10} x_{19} \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \\ w_{d+1} \\ w_{d+2} \end{bmatrix}$$

ii. Applying an explicit defined non-linear regression function to the original input. For example

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 x_2^2 + w_2 x_3 x_5^8 + \cdots$$

iii. Applying an implicitly defined non-linear transformation to the original input and then a linear model to the transformed input

$$U \mapsto V, \mathbf{x} \in U, \mathbf{z} \in V, f(\mathbf{z}; \mathbf{w})$$

(c) We will consider the first approach here and leave the other two for some later topics

(d) Advantage of the first approach is that linear regression can still be used and the weights in a linear regression model have **text interpretability** (the larger the magnitude of a weight, the more significant is the corresponding input feature).

(e) (Polynomial Regression). One common approach is to introduce *higher-order terms* as additional input dimensions, e.g., $x_i^2, x_i x_j, x_i x_j^2 x_k$

$$\begin{aligned} f(\mathbf{x}; \mathbf{w}) &= w_0 + w_1 x + \cdots + w_m x^m \\ &= \begin{bmatrix} w_0 & w_1 & \cdots & w_m \end{bmatrix} \begin{bmatrix} 1 \\ x \\ \vdots \\ x^m \end{bmatrix} \\ &= \mathbf{w}^T \tilde{\mathbf{x}} \end{aligned}$$

Remark: Although f is non-linear in $\tilde{\mathbf{x}}$, it is linear in the optimization variable $\mathbf{w}$.
Very often, **feature engineering** that uses domain knowledge to define application-specific features is applied, two of the original features are body weight and body height, we may define the body mass index (BMI)

1.5 Model Over-fitting

1. If the training set $S = \{(\mathbf{x}^l, \mathbf{y}^l)\}_{l=1}^N$ is small compared to the number of parameters in the linear regression function $f(\mathbf{x}; \mathbf{w})$, overfitting may occur.

2. When overfitting occurs, it is common to find large magnitudes in at least some of the parameters. This is because a large search space is needed for the (overly complex) model to fit the data exactly.

3. One common solution to the overfitting problem is to prevent the parameters from growing excessively large in magnitude.

$$\begin{aligned} \text{Overfitting} &\implies \text{Large magnitudes in some weights} \\ \equiv \text{Not large magnitudes in some weights} &\implies \text{Not overfitting} \end{aligned}$$

   To attain so, we will do **Regularization**.

1.6 Regularization

1. Regularization is an approach which modifies the original loss function by adding one or more penalty terms, called regularizers, that penalize large parameter magnitudes.

2. For example, Regularized loss function based on L_2 regularization (a.k.a. Tikhonov regularization)

$$\begin{aligned} L_\lambda(\mathbf{w}; S) &= L(\mathbf{w}; S) + \lambda||\mathbf{w}||^2 \\ &= ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda||\mathbf{w}||^2 \\ &= (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}^T\mathbf{w} \\ &= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{y}^T\mathbf{X}\mathbf{w} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w} \end{aligned}$$

   where $\lambda > 0$ is called the regularization parameter which controls how strong the regularization is. Its value can be determined as part of the validation process.

3. In practice, not regularizing the biase term w_0 usually gives better result since overfitting is caused by the data.

4. To compute the closed-form solution with L_2 Regularization, we will follow a few steps.

   (a) Differentiate $L_\lambda(\mathbf{w}; S)$ with respect to $\mathbf{w}$ and set the derivative to $\vec{0}$

$$\begin{aligned} 2\mathbf{X^T}\mathbf{X}\mathbf{w} - 2\mathbf{X^T}\mathbf{y} + 2\lambda\mathbf{w} &= 0 \\ (\mathbf{X^T}\mathbf{X} + \lambda\mathbf{I})\mathbf{w} &= \mathbf{X^T}\mathbf{y} \end{aligned}$$

   (b) The least squares estimate can also be obtained in closed form:

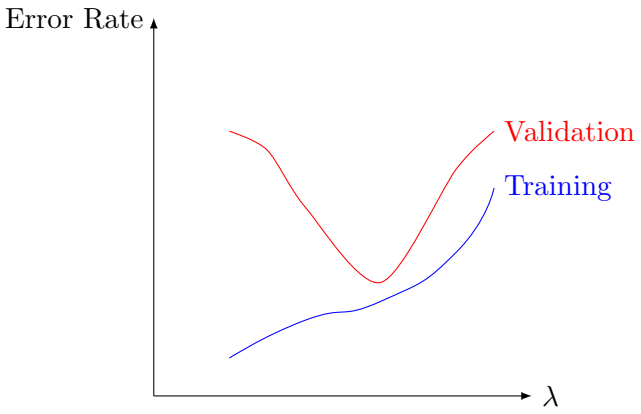
$$\tilde{\mathbf{w}} = (\mathbf{X^T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X^T}\mathbf{y}$$

   (c) Note that $\mathbf{X^T}\mathbf{X}$ is positive semi-definite and $\mathbf{X^T}\mathbf{X} + \lambda\mathbf{I}$ is positive definite. Therefore, $\mathbf{X^T}\mathbf{X} + \lambda\mathbf{I}$ is always invertible for any $\lambda > 0$.

   (d) Remark: linear regression with L_2 regularization degenerates to the ordinary linear regression (without regularization) when $\lambda = 0$.

5. (Choice of λ). Although not the only method, **cross validation** (or, more correctly, called holdout validation) is commonly used by training a model on a `training set` and validating the trained model on a separate `validation set` (which mimics the `test set`).

6. Typical Training and validation error curves



The validation process is to look for the sweet spot in the validation error curve.

7. Other Regularizers
   (a) Many other regularizers can also be defined.
   (b) For Example, instead of using L_2 norm, the L_p norm for some other value of p has also been used.
   (c) Linear regression with L_1 regularization, also called **LASSO** (least absolute shrinkage and selection operator), favors sparse solutions with all but a small number of dimensions equal to 0.
   (d) Although the L_1 norm is also convex like L_2 norm, there is no closed-form solution for LASSO since it's not differentiable everywhere. Iterative algorithms are needed for estimating the parameters.

8. Mean Squared Error
   (a) A common performance metric for regression problems is the mean square error (MSE)

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

   , which is similar to the squared loss but with two differences:
      i. MSE can be used for the validation set and test set in addition to the training set.
      ii. MSE measures the mean over all the examples in the set, not the sum.
   (b) Instead of MSE, it is more common to use the root mean squared error (RMSE).

9. R^2 Score
   (a) Another commonly used performance metric for regression is the coefficient of determination or R^2 score

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

   , where $\bar{y} = \frac{1}{N}\sum_{l=1}^{N} y^{(l)}$
   (b) The best possible R^2 score is 1 when the corresponding MSE is 0.
   (c) When the model always predicts the mean value of y, the R^2 score will be equal to 0.
   (d) Negative values are also possible because the model can have arbitrarily large MSE.

2 Feedforward Neural Network

2.1 Artificial Neural Network

1. Early research in artificial neural networks was inspired by findings from **neuronscience**, but subsequent development has mostly been guided by mathematical and computational considerations.

2. Machine learning researchers and practitioners regard artificial neural networks as computational models for machine learning.

3. There are **two** types of artificial neural networks:

   (a) Feedforward neural networks: networks without loops

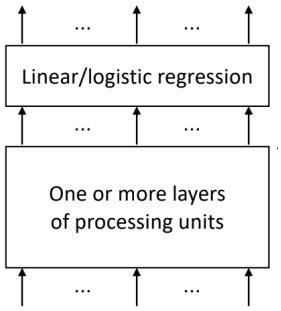
   (b) Recurrent neural networks: networks with loops

2.2 Layered Extension of Linear or Logistic Regression

1. A feedforward neural network (a.k.a. multi-layer perceptron MLP), may be considered as an extension of linear or logistic regression.

2. The input is transformed by one or more layers of processing units (a.k.a. neurons) before it is fed into the linear or logistic regression model which corresponds to the output layer of the feedforward neural network.

3. Consequently, feedforward neural networks can be regarded as nonlinear generalizations



2.3 Universal Approximation

1. Analogous to **Turing machine** as a universal mathematical model of computation for today's digital computers, one would also like to know the universal mathematical model for feedforward neural networks.

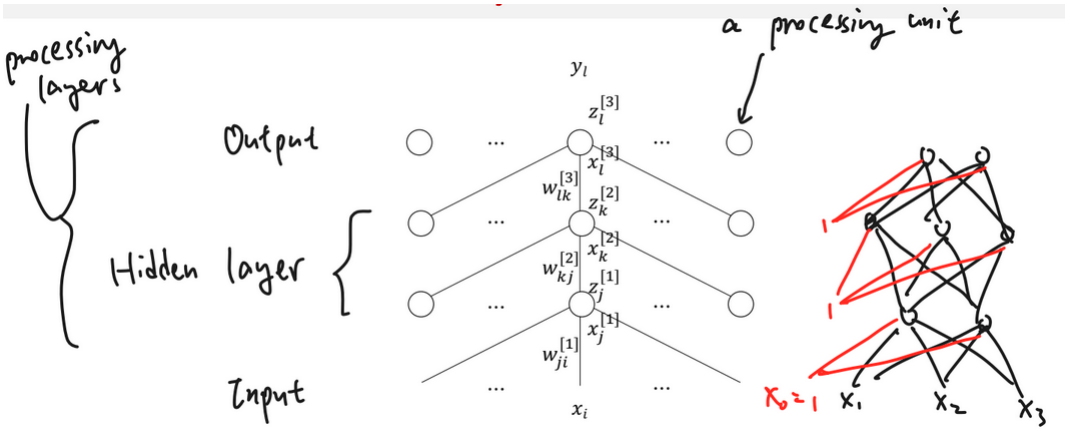
2. An informal way of stating the universal approximation theorem (proved in the late 1980s) is that, a feedforward neural network with **sufficiently many sigmoid hidden units in only one layer** can approximate any well-behaved function to arbitrary precision. However, this theorem might have misled people not to put too much effort into exploring deeper neural networks.

3. Nevertheless, using more than one hidden layer may give a network that can approximate the same function using (exponentially) **fewer parameters** due to the high non-linearly that a deeper network can induce. This higher parameter efficiency also makes deeper networks much **faster to train**.

4. Deeper networks also mimic better the **hierarchical organization** of data in many real-world applications.

2.4 An illustrative example: a 3-layer network

1. Each processing unit is shown as a circle. No processing is done in the input layer so no processing units are needed.



2. There are **bias terms** for all the processing layers.

3. We consider here an illustrative example with 2 hidden layers to simply the notation. However, we still call it a 3-layer network as the output layer also count as a layer with processing units.

4. The superscripts, e.g., $[1], [2]$, refer to the corresponding network layers.

5. For each processing unit,

   (a) its (summed) input is denoted by $x^{[*]}$;
   e.g., $x_j^{[1]}$ means the input to j-th unit in the first hidden layer.

   (b) its output is denoted by $z^{[*]}$;
   e.g., $z_j^{[1]}$ means the output from j-th unit in the first hidden layer.

6. The input layer may also be denoted using the superscript $[0]$.

2.5 Activation Functions

1. The function relating the input and output of a processing unit is called an activation function of the unit, e.g.,
$$z_j^{[1]} = g_j^{[1]}(x_j^{[1]})$$

2. All the activation functions are **non-linear**, except for those in the output layer.

3. Note that if the activation functions of the 2 hidden layers are **linear**, then these 2 layers can be merge into one.

   (a) Input layer to first hidden layer:
   Input can be computed by
   $$x_j^{[1]} = \sum_i w_{ji}^{[1]} x_i$$
   or in matrix form
   $$\mathbf{x}^{[1]} = \mathbf{W}^{[1]}\mathbf{x}^{[0]}$$
   ; Output can be computed by
   $$z_j^{[1]} = g_j^{[1]}(x_j^{[1]})$$
   or in matrix form
   $$\mathbf{z}^{[1]} = g^{[1]}(\mathbf{x}^{[1]})$$

(b) First hidden layer to second hidden layer:
Input can be computed by

$$x_j^{[2]} = \sum_j w_{kj}^{[2]} z_j$$

or in matrix form

$$\mathbf{x}^{[2]} = \mathbf{W}^{[2]}\mathbf{z}^{[1]}$$

; Output can be computed by

$$z_k^{[2]} = g_k^{[2]}(x_k^{[2]})$$

or in matrix form

$$\mathbf{z}^{[2]} = g^{[2]}(\mathbf{x}^{[2]})$$

(c) Second hidden layer to output layer:
Input can be computed by

$$x_l^{[3]} = \sum_k w_{lk}^{[3]} z_k$$

or in matrix form

$$\mathbf{x}^{[3]} = \mathbf{W}^{[3]}\mathbf{z}^{[2]}$$

; Output can be computed by

$$z_l^{[3]} = g_l^{[3]}(x_l^{[3]})$$

or in matrix form

$$\mathbf{z}^{[3]} = g^{[3]}(\mathbf{x}^{[3]})$$

Sequentially, we can write

$$\begin{aligned}
\mathbf{z}^{[3]} &= g^{[3]}(\mathbf{x}^{[3]}) \\
&= g^{[3]}(\mathbf{W}^{[3]}\mathbf{z}^{[2]}) \\
&= g^{[3]}(\mathbf{W}^{[3]}g^{[2]}(\mathbf{x}^{[2]})) \\
&= g^{[3]}(\mathbf{W}^{[3]}g^{[2]}(\mathbf{W}^{[2]}\mathbf{z}^{[1]})) \\
&= g^{[3]}(\mathbf{W}^{[3]}g^{[2]}(\mathbf{W}^{[2]}g^{[1]}(\mathbf{x}^{[1]}))) \\
&= g^{[3]}(\mathbf{W}^{[3]}g^{[2]}(\mathbf{W}^{[2]}g^{[1]}(\mathbf{W}^{[1]}\mathbf{x}^{[0]})))
\end{aligned}$$

If $g^{[1]}$, $g^{[2]}$, and $g^{[3]}$ are linear, we can write

$$\mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathbf{x}^{[0]} = \mathbf{W}'\mathbf{x}^{[0]}$$

, that is, we can merge 3 layers in to one.

4. One exception is in an autoencoder in which a linear function may be used in the hidden units.
5. The logistic function for logistic regression can also seen as an activation function.

2.6 Loss Functions

Recall these loss functions:

1. Squared loss function for regression problems

$$L(\mathbf{W};\mathcal{S}) = \frac{1}{2}\sum_{q=1}^{N}\sum_l (z_l^{[3](q)} - y_l^{(q)})^2 = \frac{1}{2}\sum_{q=1}^{N}\sum_l (x_l^{[3](q)} - y_l^{(q)})^2$$

The constant $\frac{1}{2}$ is introduced to simplify the subsequent derivation.

2. Cross-entropy loss function for classification problems

$$L(\mathbf{W};\mathcal{S}) = -\sum_{q=1}^{N}\sum_l y_l^{(q)} \log z_l^{[3](q)} = -\sum_{q=1}^{N}\sum_l y_l^{(q)} \log \texttt{softmax}(x_l^{[3](q)})$$

2.7 Back-propagation Learning Algorithm

1. **Gradient descent** based on the gradients computed recursively in the backward direction starting from the output layer:

$$\Delta w_{lk}^{[3]} \propto -\frac{\partial L}{\partial w_{lk}^{[3]}}, \Delta w_{kj}^{[2]} \propto -\frac{\partial L}{\partial w_{kj}^{[2]}}, \Delta w_{ji}^{[1]} \propto -\frac{\partial L}{\partial w_{ji}^{[1]}}$$

2. This recursive way of gradient computation for gradient descent is called the back-propagation (BP) learning algorithm.

3. Strictly speaking BP is not a recursive algorithm in the usual programming language sense. They share the same spirit though, e.g., computing $n!$ requires doing something (multiplying by n) to the result of $(n-1)!$.

4. More advanced gradient-based learning algorithms may also make use of the gradients computed this way.

5. Algorithm Sketch for (Batch) BP Learning

Algorithm 1 Batch BP Learning

```
Initialize variables
repeat
    for each training example x do
        predicted-output ← neural-network-output(x)        ▷ Forward propagation
        actual-output ← label(x)
        Compute error terms at output units by a loss function

        Compute weights changes for last layers of weights (Δw[3])      ▷ Backward propagation
        Compute weights changes for second layers of weights (Δw[2])
        Compute weights changes for first layers of weights (Δw[1])
    end for
    Update network weights for all layers
until some stopping criterion is satisfied
```

6. Gradients computations
   We first let

$$L^{(q)} = \begin{cases} \frac{1}{2}\sum_l (z_l^{[3](q)} - y_l^{(q)})^2 \text{ if regression} \\ -\sum_l y_l^{(q)} \log z_l^{[3](q)} \text{ if classification} \end{cases}$$

   (a) Gradients of the last layer of weights $w_{lk}^{[3]}$

$$\begin{aligned} \frac{\partial L}{\partial w_{lk}^{[3]}} &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{lk}^{[3]}} \\ &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_l^{[3](q)}} \frac{\partial x^{[3](q)}}{\partial w_{lk}^{[3]}} \\ &= -\sum_{q=1}^{N} \delta^{[3](q)} \frac{\partial x^{[3](q)}}{\partial w_{lk}^{[3]}} \end{aligned}$$

   So, to compute $\frac{\partial L}{\partial w_{lk}^{[3]}}$, we need both $\delta^{[3](q)}$ and $\frac{\partial x^{[3](q)}}{\partial w_{lk}^{[3]}}$

   i. Compute $\delta^{[3](q)}$:

(for regression)

$$\begin{aligned}\delta^{[3](q)} &= -\frac{\partial L^{(q)}}{\partial x_l^{[3](q)}} \\ &= -\sum_m \frac{\partial L^{(q)}}{\partial z_m^{[3](q)}} \frac{\partial z_m^{[3](q)}}{\partial x_l^{[3](q)}} \\ &= -\frac{\partial L^{(q)}}{\partial z_l^{[3](q)}} \frac{\partial z_l^{[3](q)}}{\partial x_l^{[3](q)}} \\ &= y_l^{(q)} - z_l^{[3](q)}\end{aligned}$$

(for classification)

$$\begin{aligned}\delta^{[3](q)} &= -\frac{\partial L^{(q)}}{\partial x_l^{[3](q)}} \\ &= -\sum_m \frac{\partial L^{(q)}}{\partial z_m^{[3](q)}} \frac{\partial z_m^{[3](q)}}{\partial x_l^{[3](q)}} \\ &= \sum_m \frac{y_m^{(q)}}{z_m^{[3](q)}} z_m^{[3](q)} (\delta_{ml} - z_l^{[3](q)}) \\ &= \sum_m y_m^{(q)} (\delta_{ml} - z_l^{[3](q)}) \\ &= y_l^{(q)} - z_l^{[3](q)}\end{aligned}$$

Remarks: the terms $\delta_l^{[3](q)}$ can be regarded as the error terms computed at the output layer.

ii. Compute $\frac{\partial x^{[3](q)}}{\partial w_{lk}^{[3]}}$:

$$\frac{\partial x^{[3](q)}}{\partial w_{lk}^{[3]}} = z_k^{[2](q)}$$

(b) Gradients of the second layer of weights $w_{kj}^{[2]}$

$$\begin{aligned}\frac{\partial L}{\partial w_{kj}^{[2]}} &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{kj}^{[2]}} \\ &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} \frac{\partial x_k^{[2](q)}}{\partial w_{kj}^{[2]}} \\ &= -\sum_{q=1}^{N} \delta_k^{[2](q)} \frac{\partial x_k^{[2](q)}}{\partial w_{kj}^{[2]}}\end{aligned}$$

, where

$$\begin{aligned}\delta_k^{[2](q)} &= -\frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} \\ &= -\sum_l \frac{\partial L^{(q)}}{\partial x_l^{[3](q)}} \frac{\partial x_l^{[3](q)}}{\partial z_k^{[2](q)}} \frac{\partial z_k^{[2](q)}}{\partial x_k^{[2](q)}} \\ &= \sum_l \delta_l^{[3](q)} w_{lk}^{[3]} g_k^{[2]\prime}(x_k^{[2](q)})\end{aligned}$$

Remarks: the error terms $\delta_k^{[2](q)}$ of the second hidden layer are computed based on $\delta_l^{[3](q)}$; and

$$\frac{\partial x^{[2](q)}}{\partial w_{kj}^{[2]}} = z_j^{[1](q)}$$

(c) Gradients of the first layer of weights $w_{ji}^{[1]}$

$$\begin{aligned}\frac{\partial L}{\partial w_{ji}^{[1]}} &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial w_{ji}^{[1]}} \\ &= \sum_{q=1}^{N} \frac{\partial L^{(q)}}{\partial x_j^{[1](q)}} \frac{\partial x_j^{[1](q)}}{\partial w_{ji}^{[1]}} \\ &= -\sum_{q=1}^{N} \delta_j^{[1](q)} \frac{\partial x_j^{[1](q)}}{\partial w_{ji}^{[1]}}\end{aligned}$$

, where

$$\begin{aligned}\delta_k^{[1](q)} &= -\frac{\partial L^{(q)}}{\partial x_j^{[1](q)}} \\ &= -\sum_l \frac{\partial L^{(q)}}{\partial x_k^{[2](q)}} \frac{\partial x_k^{[2](q)}}{\partial z_j^{[1](q)}} \frac{\partial z_j^{[1](q)}}{\partial x_j^{[1](q)}} \\ &= \sum_k \delta_k^{[2](q)} w_{kj}^{[2]} g_j^{[1]'}(x_j^{[1](q)})\end{aligned}$$

Remarks: the error terms $\delta_j^{[1](q)}$ of the first hidden layer are computed based on $\delta_k^{[2](q)}$; and

$$\frac{\partial x^{[1](q)}}{\partial w_{ji}^{[1]}} = x_i^{[1](q)}$$

7. Weights Update Rules

(a) Last layer:

$$\delta_l^{[3](q)} = y_l^{(q)} - z_l^{[3](q)}$$

$$\Delta w_{lk}^{[3]} = -\eta \frac{\partial L}{\partial w_{lk}^{[3]}} = \eta \sum_{q=1}^{N} \delta_l^{[3](q)} z_k^{[2](q)}$$

(b) Second layer:

$$\delta_k^{[2](q)} = \sum_l \delta_l^{[3](q)} w_{lk}^{[3]} g_k^{[2]'}(x_k^{[2](q)})$$

$$\Delta w_{kj}^{[2]} = -\eta \frac{\partial L}{\partial w_{kj}^{[2]}} = \eta \sum_{q=1}^{N} \delta_k^{[2](q)} z_j^{[1](q)}$$

(c) First layer:

$$\delta_j^{[1](q)} = \sum_k \delta_k^{[2](q)} w_{kj}^{[2]} g_j^{[1]'}(x_j^{[1](q)})$$

$$\Delta w_{ji}^{[1]} = -\eta \frac{\partial L}{\partial w_{ji}^{[1]}} = \eta \sum_{q=1}^{N} \delta_j^{[1](q)} x_i^{(q)}$$