732A99/TDDE01/732A68 Machine Learning

Lecture 3c Block 1: Neural Networks I

Jose M. Peña
IDA, Linköping University, Sweden

Contents and Literature

- Content
  - Neural Networks
  - Backpropagation Algorithm
  - Summary
- Literature
  - Lindholm, A., Wahlström, N., Lindsten, F. and Schön, T. B. *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. Chapter 6-6.2.

Neural Networks

- Predictions via a linear regression model.

$$\widehat{y} = W_1 x_1 + W_2 x_2 + \cdots + W_p x_p + b, \tag{6.2}$$

Neural Networks

- Predictions via a linear regression model.

$$\widehat{y} = W_1x_1 + W_2x_2 + \cdots + W_px_p + b, \tag{6.2}$$

- Predictions via a generalized linear regression model (i.e., non-linear relationship between $\boldsymbol{x}$ and y).

$$\widehat{y} = h\big(W_1x_1 + W_2x_2 + \cdots + W_px_p + b\big). \tag{6.3}$$

Neural Networks

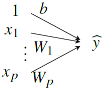
- Predictions via a linear regression model.

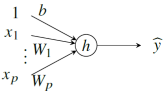
$$\widehat{y} = W_1 x_1 + W_2 x_2 + \cdots + W_p x_p + b, \tag{6.2}$$

- Predictions via a generalized linear regression model (i.e., non-linear relationship between $\boldsymbol{x}$ and y).

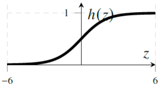
$$\widehat{y} = h\big(W_1 x_1 + W_2 x_2 + \cdots + W_p x_p + b\big). \tag{6.3}$$

- Graphically,

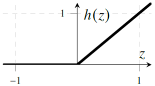




where the activation function $h(\cdot)$ may be



Logistic: $h(z) = \frac{1}{1+e^{-z}}$



ReLU: $h(z) = \max(0, z)$

Neural Networks

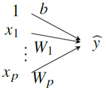
- Predictions via a linear regression model.

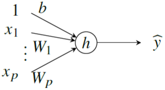
$$\widehat{y} = W_1 x_1 + W_2 x_2 + \cdots + W_p x_p + b, \tag{6.2}$$

- Predictions via a generalized linear regression model (i.e., non-linear relationship between $\boldsymbol{x}$ and y).

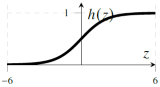
$$\widehat{y} = h\big(W_1 x_1 + W_2 x_2 + \cdots + W_p x_p + b\big). \tag{6.3}$$

- Graphically,

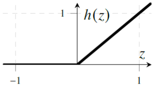




where the activation function $h(\cdot)$ may be



Logistic: $h(z) = \frac{1}{1+e^{-z}}$



ReLU: $h(z) = \max(0, z)$

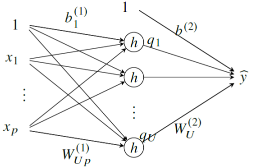
- The generalized linear regression model can be improved by creating a layer with several such models in parallel and, then, stacking these layers in a sequence.

Neural Networks

- A two-layer NN consists of U generalized linear regression models that are combined via a linear regression model. The output of each generalized linear regression model is called **hidden unit**, as it is not the final output.

$$
\begin{aligned}
q_1 &= h\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + \cdots + W_{1p}^{(1)}x_p + b_1^{(1)}\right),\\
q_2 &= h\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + \cdots + W_{2p}^{(1)}x_p + b_2^{(1)}\right),\\
&\vdots\\
q_U &= h\left(W_{U1}^{(1)}x_1 + W_{U2}^{(1)}x_2 + \cdots + W_{Up}^{(1)}x_p + b_U^{(1)}\right),
\end{aligned}
\tag{6.6a}
$$

$$
\widehat{y} = W_1^{(2)}q_1 + W_2^{(2)}q_2 + \cdots + W_U^{(2)}q_U + b^{(2)}. \tag{6.6b}
$$

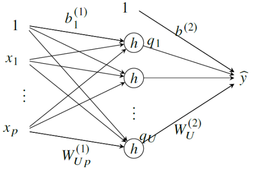


Neural Networks

- A two-layer NN consists of U generalized linear regression models that are combined via a linear regression model. The output of each generalized linear regression model is called **hidden unit**, as it is not the final output.

$$
\begin{aligned}
q_1 &= h\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + \cdots + W_{1p}^{(1)}x_p + b_1^{(1)}\right),\\
q_2 &= h\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + \cdots + W_{2p}^{(1)}x_p + b_2^{(1)}\right),\\
&\vdots\\
q_U &= h\left(W_{U1}^{(1)}x_1 + W_{U2}^{(1)}x_2 + \cdots + W_{Up}^{(1)}x_p + b_U^{(1)}\right),
\end{aligned}
\tag{6.6a}
$$

$$
\widehat{y} = W_1^{(2)}q_1 + W_2^{(2)}q_2 + \cdots + W_U^{(2)}q_U + b^{(2)}. \tag{6.6b}
$$



- More compactly,

$$
\mathbf{W}^{(1)} = \begin{bmatrix} W_{11}^{(1)} & \dots & W_{1p}^{(1)} \\ \vdots & & \vdots \\ W_{U1}^{(1)} & \dots & W_{Up}^{(1)} \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_U^{(1)} \end{bmatrix}, \quad \mathbf{W}^{(2)} = \begin{bmatrix} W_1^{(2)} & \dots & W_U^{(2)} \end{bmatrix}, \quad \mathbf{b}^{(2)} = \begin{bmatrix} b^{(2)} \end{bmatrix}. \tag{6.7}
$$

$$
\mathbf{q} = h\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right), \tag{6.8a}
$$

$$
\widehat{y} = \mathbf{W}^{(2)}\mathbf{q} + \mathbf{b}^{(2)}, \tag{6.8b}
$$

Neural Networks

- For a large variety of activation functions, the two-layer NN can uniformly approximate any continuous function to **arbitrary accuracy** provided enough hidden units. Easy to fit the parameters? Overfitting?!

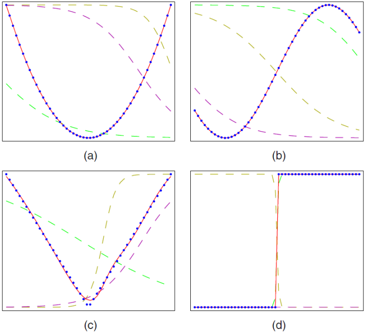


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

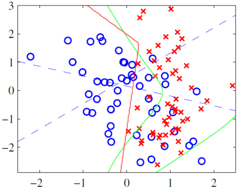
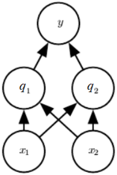
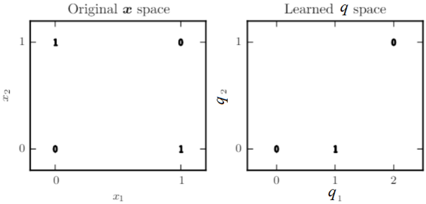


Figure 5.4 Example of the solution of a simple two-class classification problem involving synthetic data using a neural network having two inputs, two hidden units with 'tanh' activation functions, and a single output having a logistic sigmoid activation function. The dashed blue lines show the $z = 0.5$ contours for each of the hidden units, and the red line shows the $y = 0.5$ decision surface for the network. For comparison, the green line denotes the optimal decision boundary computed from the distributions used to generate the data.

Neural Networks

- Solving the XOR problem with a two-layer NN.
- No line shatters the points in the original space.
- The NN represents a **mapping from the input space to a feature space** where a line can shatter the points. Note that the points (0,1) and (1,0) are mapped both to the point (1,0).
- It resembles SVMs to some extent.





$$W_{11}^{(1)} = W_{12}^{(1)} = W_{21}^{(1)} = W_{22}^{(1)} = 1$$
$$b_1^{(1)} = 0,\ b_2^{(1)} = -1$$
$$q_i^{(1)} = h(W_{i1}^{(1)}x_1 + W_{i2}^{(1)}x_2 + b_i^{(1)}) = \max\{0, W_{i1}^{(1)}x_1 + W_{i2}^{(1)}x_2 + b_i^{(1)}\} \text{ with } i = 1, 2$$
$$W_1^{(2)} = 1,\ W_2^{(2)} = -2$$
$$b^{(2)} = 0$$
$$y = W_1^{(2)}q_1 + W_2^{(2)}q_2 + b^{(2)}$$

Neural Networks

- The two-layer NN can be generalized to L layers.

$$\begin{aligned}
\mathbf{q}^{(1)} &= h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}),\\
\mathbf{q}^{(2)} &= h(\mathbf{W}^{(2)}\mathbf{q}^{(1)} + \mathbf{b}^{(2)}),\\
&\vdots\\
\mathbf{q}^{(L-1)} &= h(\mathbf{W}^{(L-1)}\mathbf{q}^{(L-2)} + \mathbf{b}^{(L-1)}),\\
\widehat{y} &= \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}.
\end{aligned} \tag{6.11}$$

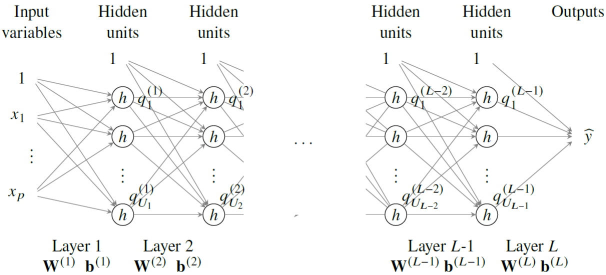


Figure 6.4: A deep neural network with L layers. Each layer l is parameterized by $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$.

Neural Networks

- NNs can also be used for M-class classification.

$$\mathbf{q}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \tag{6.16a}$$

$$\vdots$$

$$\mathbf{q}^{(L-1)} = h(\mathbf{W}^{(L-1)}\mathbf{q}^{(L-2)} + \mathbf{b}^{(1)}), \tag{6.16b}$$

$$\mathbf{z} = \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \tag{6.16c}$$

$$\mathbf{g} = \mathrm{softmax}(\mathbf{z}). \tag{6.16d}$$

where

$$\mathrm{softmax}(\mathbf{z}) \triangleq \frac{1}{\sum_{j=1}^{M} e^{z_j}} \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \vdots \\ e^{z_M} \end{bmatrix}. \tag{6.15}$$

and note that $\boldsymbol{W}^{(L)}$ and $\boldsymbol{b}^{(L)}$ must have dimensions $M \times U_{L-1}$ and M, respectively.

Neural Networks

- NNs can also be used for M-class classification.

$$
\begin{aligned}
\mathbf{q}^{(1)} &= h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), && (6.16a)\\
&\vdots \\
\mathbf{q}^{(L-1)} &= h(\mathbf{W}^{(L-1)}\mathbf{q}^{(L-2)} + \mathbf{b}^{(1)}), && (6.16b)\\
\mathbf{z} &= \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, && (6.16c)\\
\mathbf{g} &= \operatorname{softmax}(\mathbf{z}). && (6.16d)
\end{aligned}
$$

  where

$$
\operatorname{softmax}(\mathbf{z}) \triangleq \frac{1}{\sum_{j=1}^{M} e^{z_j}} \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \vdots \\ e^{z_M} \end{bmatrix}. \qquad (6.15)
$$

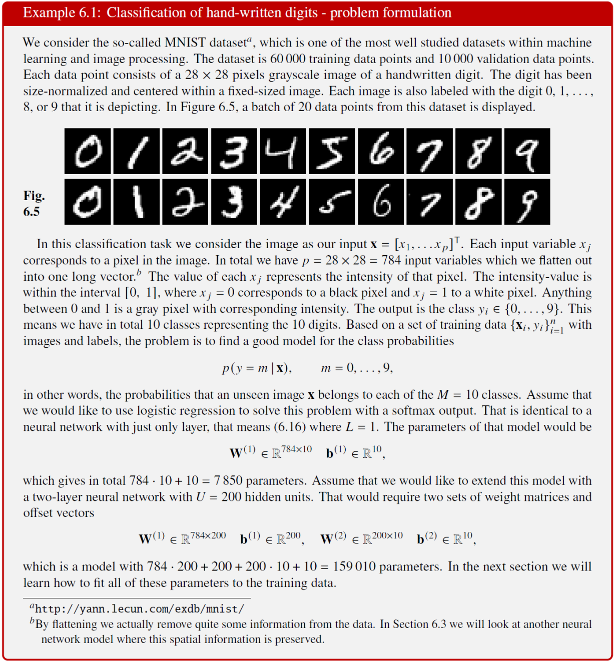
  and note that $\boldsymbol{W}^{(L)}$ and $\boldsymbol{b}^{(L)}$ must have dimensions $M \times U_{L-1}$ and M, respectively.

- The NN notation can be extended to predict the output for several input points simultaneously. For the two-layer NN, for instance, we have that

$$
\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{bmatrix}, \quad \widehat{\mathbf{y}} = \begin{bmatrix} \widehat{y}_1 \\ \vdots \\ \widehat{y}_n \end{bmatrix}, \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{q}_n^{\mathsf{T}} \end{bmatrix}. \qquad (6.13)
$$

$$
\begin{aligned}
\mathbf{Q} &= h(\mathbf{X}\mathbf{W}^{(1)\mathsf{T}} + \mathbf{b}^{(1)\mathsf{T}}), && (6.14a)\\
\widehat{\mathbf{y}} &= \mathbf{Q}\mathbf{W}^{(2)\mathsf{T}} + \mathbf{b}^{(2)\mathsf{T}}, && (6.14b)
\end{aligned}
$$

Neural Networks



Example 6.1: Classification of hand-written digits - problem formulation

We consider the so-called MNIST dataset[a], which is one of the most well studied datasets within machine learning and image processing. The dataset is 60 000 training data points and 10 000 validation data points. Each data point consists of a 28 × 28 pixels grayscale image of a handwritten digit. The digit has been size-normalized and centered within a fixed-sized image. Each image is also labeled with the digit 0, 1, . . . , 8, or 9 that it is depicting. In Figure 6.5, a batch of 20 data points from this dataset is displayed.

Fig. 6.5

In this classification task we consider the image as our input $\mathbf{x} = [x_1, \dots x_p]^\mathsf{T}$. Each input variable x_j corresponds to a pixel in the image. In total we have $p = 28 \times 28 = 784$ input variables which we flatten out into one long vector.[b] The value of each x_j represents the intensity of that pixel. The intensity-value is within the interval $[0,\ 1]$, where $x_j = 0$ corresponds to a black pixel and $x_j = 1$ to a white pixel. Anything between 0 and 1 is a gray pixel with corresponding intensity. The output is the class $y_i \in \{0, \dots, 9\}$. This means we have in total 10 classes representing the 10 digits. Based on a set of training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$ with images and labels, the problem is to find a good model for the class probabilities

$$p(y = m \,|\, \mathbf{x}), \qquad m = 0, \dots, 9,$$

in other words, the probabilities that an unseen image $\mathbf{x}$ belongs to each of the $M = 10$ classes. Assume that we would like to use logistic regression to solve this problem with a softmax output. That is identical to a neural network with just only layer, that means (6.16) where $L = 1$. The parameters of that model would be

$$\mathbf{W}^{(1)} \in \mathbb{R}^{784 \times 10} \quad \mathbf{b}^{(1)} \in \mathbb{R}^{10},$$

which gives in total $784 \cdot 10 + 10 = 7\,850$ parameters. Assume that we would like to extend this model with a two-layer neural network with $U = 200$ hidden units. That would require two sets of weight matrices and offset vectors

$$\mathbf{W}^{(1)} \in \mathbb{R}^{784 \times 200} \quad \mathbf{b}^{(1)} \in \mathbb{R}^{200}, \quad \mathbf{W}^{(2)} \in \mathbb{R}^{200 \times 10} \quad \mathbf{b}^{(2)} \in \mathbb{R}^{10},$$

which is a model with $784 \cdot 200 + 200 + 200 \cdot 10 + 10 = 159\,010$ parameters. In the next section we will learn how to fit all of these parameters to the training data.

[a] http://yann.lecun.com/exdb/mnist/

[b] By flattening we actually remove quite some information from the data. In Section 6.3 we will look at another neural network model where this spatial information is preserved.

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^{\mathsf{T}} \quad \mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^{\mathsf{T}} \quad \mathbf{b}^{(L)\mathsf{T}}\right]^{\mathsf{T}} \tag{6.17}$$

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \begin{bmatrix} \operatorname{vec}(\mathbf{W}^{(1)})^{\mathsf{T}} & \mathbf{b}^{(1)\mathsf{T}} & \cdots & \operatorname{vec}(\mathbf{W}^{(L)})^{\mathsf{T}} & \mathbf{b}^{(L)\mathsf{T}} \end{bmatrix}^{\mathsf{T}} \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\operatorname{vec}(\mathbf{W}^{(1)})^\top \quad \mathbf{b}^{(1)\top} \quad \cdots \quad \operatorname{vec}(\mathbf{W}^{(L)})^\top \quad \mathbf{b}^{(L)\top}\right]^\top \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \tag{6.19}$$

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^\intercal \quad \mathbf{b}^{(1)\intercal} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^\intercal \quad \mathbf{b}^{(L)\intercal}\right]^\intercal \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \tag{6.19}$$

- For classification, we use the cross-entropy loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^{M} e^{z_j}, \tag{6.20}$$

where $z_j = f_j(\mathbf{x}; \boldsymbol{\theta})$ is the j^{th} logit, that is the j^{th} output of the last layer before the softmax function $\mathbf{g}(\mathbf{z})$.

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^\mathsf{T} \quad \mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^\mathsf{T} \quad \mathbf{b}^{(L)\mathsf{T}}\right]^\mathsf{T} \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \tag{6.19}$$

- For classification, we use the cross-entropy loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^{M} e^{z_j}, \tag{6.20}$$

where $z_j = f_j(\mathbf{x}; \boldsymbol{\theta})$ is the j^{th} logit, that is the j^{th} output of the last layer before the softmax function $\mathbf{g}(\mathbf{z})$.

- Since there is no closed form solution, gradient descent is typically used:

1. Pick an initialization $\boldsymbol{\theta}_0$.
2. Update the parameters as $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ for $t = 0, 1, 2, \ldots$. (6.21)
3. Terminate when some criterion is fulfilled, and take the last $\boldsymbol{\theta}_t$ as $\widehat{\boldsymbol{\theta}}$.

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^{\mathsf{T}} \quad \mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^{\mathsf{T}} \quad \mathbf{b}^{(L)\mathsf{T}}\right]^{\mathsf{T}} \qquad (6.17)$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \qquad (6.18)$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \qquad (6.19)$$

- For classification, we use the cross-entropy loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^{M} e^{z_j}, \qquad (6.20)$$

  where $z_j = f_j(\mathbf{x}; \boldsymbol{\theta})$ is the j^{th} logit, that is the j^{th} output of the last layer before the softmax function $\mathbf{g}(\mathbf{z})$.

- Since there is no closed form solution, gradient descent is typically used:

  1. Pick an initialization $\boldsymbol{\theta}_0$.
  2. Update the parameters as $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ for $t = 0, 1, 2, \ldots$. (6.21)
  3. Terminate when some criterion is fulfilled, and take the last $\boldsymbol{\theta}_t$ as $\widehat{\boldsymbol{\theta}}$.

- **Challenges**:
  - The parameter space is highly multimodal. Thus, training is sensitive to the initial parameter values. These are typically small random values.

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^\mathsf{T} \quad \mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^\mathsf{T} \quad \mathbf{b}^{(L)\mathsf{T}}\right]^\mathsf{T} \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \tag{6.19}$$

- For classification, we use the cross-entropy loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^{M} e^{z_j}, \tag{6.20}$$

where $z_j = f_j(\mathbf{x}; \boldsymbol{\theta})$ is the j^{th} logit, that is the j^{th} output of the last layer before the softmax function $\mathbf{g}(\mathbf{z})$.

- Since there is no closed form solution, gradient descent is typically used:

1. Pick an initialization $\boldsymbol{\theta}_0$.
2. Update the parameters as $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ for $t = 0, 1, 2, \ldots$. (6.21)
3. Terminate when some criterion is fulfilled, and take the last $\boldsymbol{\theta}_t$ as $\widehat{\boldsymbol{\theta}}$.

- **Challenges**:
  - The parameter space is highly multimodal. Thus, training is sensitive to the initial parameter values. These are typically small random values.
  - Computing the gradient is costly when *n* is large. Solution: Stochastic gradient descent, i.e. use mini-batches.

Backpropagation Algorithm

- A NN is a parametric model with parameters

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^\mathsf{T} \quad \mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(L)})^\mathsf{T} \quad \mathbf{b}^{(L)\mathsf{T}}\right]^\mathsf{T} \tag{6.17}$$

- To find the best parameter values, we solve the optimization problem

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad \text{where } J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, y_i, \boldsymbol{\theta}). \tag{6.18}$$

- For regression, we use the squared error loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = (y - f(\mathbf{x}; \boldsymbol{\theta}))^2, \tag{6.19}$$

- For classification, we use the cross-entropy loss function

$$L(\mathbf{x}, y, \boldsymbol{\theta}) = -\ln g_y(\mathbf{f}(\mathbf{x}; \boldsymbol{\theta})) = -z_y + \ln \sum_{j=1}^{M} e^{z_j}, \tag{6.20}$$

where $z_j = f_j(\mathbf{x}; \boldsymbol{\theta})$ is the j^{th} logit, that is the j^{th} output of the last layer before the softmax function $\mathbf{g}(\mathbf{z})$.

- Since there is no closed form solution, gradient descent is typically used:

1. Pick an initialization $\boldsymbol{\theta}_0$.
2. Update the parameters as $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$ for $t = 0, 1, 2, \dots$. (6.21)
3. Terminate when some criterion is fulfilled, and take the last $\boldsymbol{\theta}_t$ as $\widehat{\boldsymbol{\theta}}$.

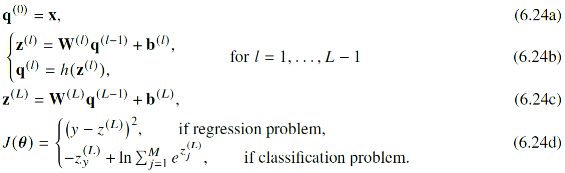
- **Challenges**:
  - The parameter space is highly multimodal. Thus, training is sensitive to the initial parameter values. These are typically small random values.
  - Computing the gradient is costly when n is large. Solution: Stochastic gradient descent, i.e. use mini-batches.
  - Computing the gradient is costly when there are many parameters. Solution: Use the backpropagation algorithm, which is based on the chain rule of derivatives, i.e. if $h(x) = f(g(x))$ then $h'(x) = f'(g(x))g'(x)$.

Backpropagation Algorithm

- The backpropagation algorithm consists of two steps, **forward and backward propagation**.

Backpropagation Algorithm

- The backpropagation algorithm consists of two steps, **forward and backward propagation**.
- Forward propagation, i.e. evaluate the cost function for a training data point $\{\boldsymbol{x}, y\}$.



$$\mathbf{q}^{(0)} = \mathbf{x}, \tag{6.24a}$$

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \\ \mathbf{q}^{(l)} = h(\mathbf{z}^{(l)}), \end{cases} \quad \text{for } l = 1, \ldots, L-1 \tag{6.24b}$$

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \tag{6.24c}$$

$$J(\boldsymbol{\theta}) = \begin{cases} \left(y - z^{(L)}\right)^2, & \text{if regression problem,} \\ -z_y^{(L)} + \ln \sum_{j=1}^{M} e^{z_j^{(L)}}, & \text{if classification problem.} \end{cases} \tag{6.24d}$$

Backpropagation Algorithm

- The backpropagation algorithm consists of two steps, **forward and backward propagation**.
- Forward propagation, i.e. evaluate the cost function for a training data point $\{\boldsymbol{x}, y\}$.

$$\mathbf{q}^{(0)} = \mathbf{x}, \tag{6.24a}$$

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \\ \mathbf{q}^{(l)} = h(\mathbf{z}^{(l)}), \end{cases} \qquad \text{for } l = 1, \dots, L-1 \tag{6.24b}$$

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \tag{6.24c}$$

$$J(\boldsymbol{\theta}) = \begin{cases} \left(y - z^{(L)}\right)^2, & \text{if regression problem,} \\ -z_y^{(L)} + \ln \sum_{j=1}^{M} e^{z_j^{(L)}}, & \text{if classification problem.} \end{cases} \tag{6.24d}$$

- Backward propagation, i.e. compute the gradient of the cost function $J(\cdot)$ wrt the hidden units $\boldsymbol{z}^{(l)}$ and $\boldsymbol{q}^{(l)}$ for $l = L, \dots, 1$.

$$d\mathbf{z}^{(l)} \triangleq \nabla_{\mathbf{z}^{(l)}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial z_1^{(l)}} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial z_{U^{(l)}}^{(l)}} \end{bmatrix}, \quad \text{and} \quad d\mathbf{q}^{(l)} \triangleq \nabla_{\mathbf{q}^{(l)}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial q_1^{(l)}} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial q_{U^{(l)}}^{(l)}} \end{bmatrix}. \tag{6.25}$$

Backpropagation Algorithm

- The backpropagation algorithm consists of two steps, **forward and backward propagation**.
- Forward propagation, i.e. evaluate the cost function for a training data point $\{\boldsymbol{x}, y\}$.

$$\mathbf{q}^{(0)} = \mathbf{x}, \tag{6.24a}$$

$$\begin{cases} \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \\ \mathbf{q}^{(l)} = h(\mathbf{z}^{(l)}), \end{cases} \quad \text{for } l = 1, \dots, L-1 \tag{6.24b}$$

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \tag{6.24c}$$

$$J(\boldsymbol{\theta}) = \begin{cases} \left(y - z^{(L)}\right)^2, & \text{if regression problem,} \\ -z_y^{(L)} + \ln \sum_{j=1}^{M} e^{z_j^{(L)}}, & \text{if classification problem.} \end{cases} \tag{6.24d}$$

- Backward propagation, i.e. compute the gradient of the cost function $J(\cdot)$ wrt the hidden units $\boldsymbol{z}^{(l)}$ and $\boldsymbol{q}^{(l)}$ for $l = L, \dots, 1$.

$$d\mathbf{z}^{(l)} \triangleq \nabla_{\mathbf{z}^{(l)}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial z_1^{(l)}} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial z_{U^{(l)}}^{(l)}} \end{bmatrix}, \quad \text{and} \quad d\mathbf{q}^{(l)} \triangleq \nabla_{\mathbf{q}^{(l)}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial q_1^{(l)}} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial q_{U^{(l)}}^{(l)}} \end{bmatrix}. \tag{6.25}$$

- Note that while forward propagation visits the layers from 1 to L, backward propagation visits them from L to 1.

Backpropagation Algorithm

- Specifically, backward propagation starts computing

$$dz^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z^{(L)}} = \frac{\partial}{\partial z^{(L)}} \left(y - z^{(L)}\right)^2 = -2\left(y - z^{(L)}\right). \tag{6.26a}$$

  for regression, or

$$dz_j^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(l)}} \left(-z_y^{(L)} + \ln \sum_{k=1}^{M} e^{z_k^{(L)}}\right) = -\mathbb{I}\{y = j\} + \frac{e^{z_j^{(L)}}}{\sum_{k=1}^{M} e^{z_k^{(L)}}}. \tag{6.26b}$$

  for classification.

Backpropagation Algorithm

- Specifically, backward propagation starts computing

$$dz^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z^{(L)}} = \frac{\partial}{\partial z^{(L)}}\left(y - z^{(L)}\right)^2 = -2\left(y - z^{(L)}\right). \tag{6.26a}$$

  for regression, or

$$dz_j^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(l)}}\left(-z_y^{(L)} + \ln \sum_{k=1}^{M} e^{z_k^{(L)}}\right) = -\mathbb{I}\{y = j\} + \frac{e^{z_j^{(L)}}}{\sum_{k=1}^{M} e^{z_k^{(L)}}}. \tag{6.26b}$$

  for classification.

- Backward propagation then continues with the following recursions.

$$d\mathbf{z}^{(l)} = d\mathbf{q}^{(l)} \odot h'(\mathbf{z}^{(l)}), \tag{6.27a}$$

$$d\mathbf{q}^{(l-1)} = \mathbf{W}^{(l)\mathsf{T}} d\mathbf{z}^{(l)}, \tag{6.27b}$$

  where $\odot$ denotes the element-wise product and where $h'(z)$ is the derivative of the activation function $h(z)$. Similar to the notation in (6.24b), $h'(\mathbf{z})$ acts element-wise on the vector $\mathbf{z}$. Note that the first line (6.27a) is not executed for layer L since that layer does not have an activation function, see (6.24c).

Backpropagation Algorithm

- Specifically, backward propagation starts computing

$$dz^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z^{(L)}} = \frac{\partial}{\partial z^{(L)}} \left(y - z^{(L)}\right)^2 = -2\left(y - z^{(L)}\right). \tag{6.26a}$$

  for regression, or

$$dz_j^{(L)} = \frac{\partial J(\boldsymbol{\theta})}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(l)}} \left(-z_y^{(L)} + \ln \sum_{k=1}^{M} e^{z_k^{(L)}}\right) = -\mathbb{I}\{y = j\} + \frac{e^{z_j^{(L)}}}{\sum_{k=1}^{M} e^{z_k^{(L)}}}. \tag{6.26b}$$

  for classification.

- Backward propagation then continues with the following recursions.

$$d\mathbf{z}^{(l)} = d\mathbf{q}^{(l)} \odot h'(\mathbf{z}^{(l)}), \tag{6.27a}$$

$$d\mathbf{q}^{(l-1)} = \mathbf{W}^{(l)\mathsf{T}} d\mathbf{z}^{(l)}, \tag{6.27b}$$

  where $\odot$ denotes the element-wise product and where $h'(z)$ is the derivative of the activation function $h(z)$. Similar to the notation in (6.24b), $h'(\mathbf{z})$ acts element-wise on the vector $\mathbf{z}$. Note that the first line (6.27a) is not executed for layer L since that layer does not have an activation function, see (6.24c).

- With $d\boldsymbol{z}^{(l)}$, the gradient of the parameters can be computed and the parameters can be updated.

$$d\mathbf{W}^{(l)} = d\mathbf{z}^{(l)}\mathbf{q}^{(l-1)\mathsf{T}}, \tag{6.27c}$$

$$d\mathbf{b}^{(l)} = d\mathbf{z}^{(l)}. \tag{6.27d}$$

$$\mathbf{W}_{t+1}^{(l)} \leftarrow \mathbf{W}_t^{(l)} - \gamma d\mathbf{W}_t^{(l)}, \tag{6.23a}$$

$$\mathbf{b}_{t+1}^{(l)} \leftarrow \mathbf{b}_t^{(l)} - \gamma d\mathbf{b}_t^{(l)}. \tag{6.23b}$$

Backpropagation Algorithm

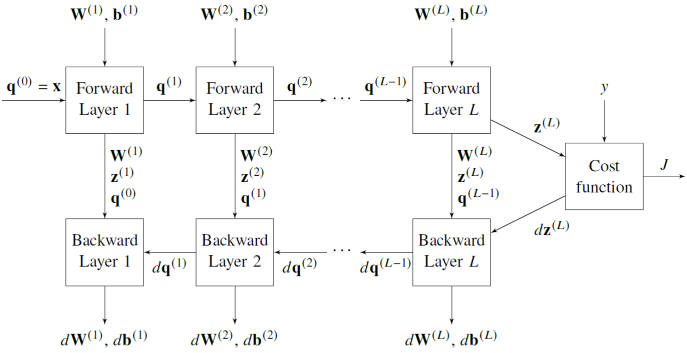

$\mathbf{W}^{(1)}, \mathbf{b}^{(1)}$
$\mathbf{W}^{(2)}, \mathbf{b}^{(2)}$
$\mathbf{W}^{(L)}, \mathbf{b}^{(L)}$
$\mathbf{q}^{(0)} = \mathbf{x}$
Forward Layer 1
$\mathbf{q}^{(1)}$
Forward Layer 2
$\mathbf{q}^{(2)}$
$\cdots$
$\mathbf{q}^{(L-1)}$
Forward Layer L
y
$\mathbf{z}^{(L)}$
Cost function
J
$\mathbf{W}^{(1)}$ $\mathbf{z}^{(1)}$ $\mathbf{q}^{(0)}$
$\mathbf{W}^{(2)}$ $\mathbf{z}^{(2)}$ $\mathbf{q}^{(1)}$
$\mathbf{W}^{(L)}$ $\mathbf{z}^{(L)}$ $\mathbf{q}^{(L-1)}$
$d\mathbf{z}^{(L)}$
Backward Layer 1
$d\mathbf{q}^{(1)}$
Backward Layer 2
$d\mathbf{q}^{(2)}$
$\cdots$
$d\mathbf{q}^{(L-1)}$
Backward Layer L
$d\mathbf{W}^{(1)}, d\mathbf{b}^{(1)}$
$d\mathbf{W}^{(2)}, d\mathbf{b}^{(2)}$
$d\mathbf{W}^{(L)}, d\mathbf{b}^{(L)}$

Figure 6.6: A computational graph of the backpropagation algorithm. We start with the input in the upper left corner and propagate forward and evaluate the cost function. Along the way we also cache the values for the hidden units $\mathbf{q}^{(l)}$ and $\mathbf{z}^{(l)}$. We then propagate the gradients $d\mathbf{q}^{(l)}$ and $d\mathbf{z}^{(l)}$ backwards and compute the gradients for the parameters $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$. The equations behind this computational graph are given in (6.24), (6.26) and (6.27).

Backpropagation Algorithm

- Backpropagation algorithm for the two-layer NN for regression. Iterate the following steps until the stopping criterion is met.

Backpropagation Algorithm

- Backpropagation algorithm for the two-layer NN for regression. Iterate the following steps until the stopping criterion is met.
  - Forward propagation.

$$
\begin{aligned}
\boldsymbol{q}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{z}^{(1)} &= \boldsymbol{W}^{(1)}\boldsymbol{q}^{(0)} + \boldsymbol{b}^{(1)} \\
\boldsymbol{q}^{(1)} &= h(\boldsymbol{z}^{(1)}) \\
z^{(2)} &= \boldsymbol{W}^{(2)}\boldsymbol{q}^{(1)} + b^{(2)} \\
J(\boldsymbol{\theta}) &= (y - z^{(2)})^2
\end{aligned}
$$

Backpropagation Algorithm

- Backpropagation algorithm for the two-layer NN for regression. Iterate the following steps until the stopping criterion is met.
  - Forward propagation.

$$\begin{aligned}
\boldsymbol{q}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{z}^{(1)} &= \boldsymbol{W}^{(1)}\boldsymbol{q}^{(0)} + \boldsymbol{b}^{(1)} \\
\boldsymbol{q}^{(1)} &= h(\boldsymbol{z}^{(1)}) \\
z^{(2)} &= \boldsymbol{W}^{(2)}\boldsymbol{q}^{(1)} + b^{(2)} \\
J(\boldsymbol{\theta}) &= (y - z^{(2)})^2
\end{aligned}$$

  - Backward propagation.

$$\begin{aligned}
dz^{(2)} &= -2(y - z^{(2)}) \\
d\boldsymbol{q}^{(1)} &= \boldsymbol{W}^{(2)T} dz^{(2)} \\
d\boldsymbol{z}^{(1)} &= d\boldsymbol{q}^{(1)} \odot h'(\boldsymbol{z}^{(1)}) \\
d\boldsymbol{W}^{(2)} &= dz^{(2)}\boldsymbol{q}^{(1)T} \\
db^{(2)} &= dz^{(2)} \\
d\boldsymbol{W}^{(1)} &= d\boldsymbol{z}^{(1)}\boldsymbol{q}^{(0)T} \\
d\boldsymbol{b}^{(1)} &= d\boldsymbol{z}^{(1)}
\end{aligned}$$

Backpropagation Algorithm

- Backpropagation algorithm for the two-layer NN for regression. Iterate the following steps until the stopping criterion is met.
  - Forward propagation.

$$
\begin{aligned}
\boldsymbol{q}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{z}^{(1)} &= \boldsymbol{W}^{(1)}\boldsymbol{q}^{(0)} + \boldsymbol{b}^{(1)} \\
\boldsymbol{q}^{(1)} &= h(\boldsymbol{z}^{(1)}) \\
z^{(2)} &= \boldsymbol{W}^{(2)}\boldsymbol{q}^{(1)} + b^{(2)} \\
J(\boldsymbol{\theta}) &= (y - z^{(2)})^2
\end{aligned}
$$

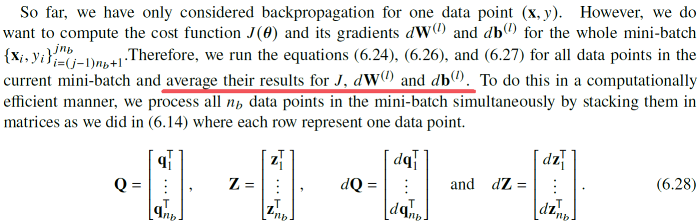
  - Backward propagation.

$$
\begin{aligned}
dz^{(2)} &= -2(y - z^{(2)}) \\
d\boldsymbol{q}^{(1)} &= \boldsymbol{W}^{(2)\,T} dz^{(2)} \\
d\boldsymbol{z}^{(1)} &= d\boldsymbol{q}^{(1)} \odot h'(\boldsymbol{z}^{(1)}) \\
d\boldsymbol{W}^{(2)} &= dz^{(2)}\boldsymbol{q}^{(1)\,T} \\
db^{(2)} &= dz^{(2)} \\
d\boldsymbol{W}^{(1)} &= d\boldsymbol{z}^{(1)}\boldsymbol{q}^{(0)\,T} \\
d\boldsymbol{b}^{(1)} &= d\boldsymbol{z}^{(1)}
\end{aligned}
$$

  - Parameter updating.

$$
\begin{aligned}
\boldsymbol{W}_{t+1}^{(2)} &= \boldsymbol{W}_{t}^{(2)} - \gamma d\boldsymbol{W}_{t}^{(2)} \\
\boldsymbol{b}_{t+1}^{(2)} &= \boldsymbol{b}_{t}^{(2)} - \gamma d\boldsymbol{b}_{t}^{(2)} \\
\boldsymbol{W}_{t+1}^{(1)} &= \boldsymbol{W}_{t}^{(1)} - \gamma d\boldsymbol{W}_{t}^{(1)} \\
b_{t+1}^{(1)} &= b_{t}^{(1)} - \gamma db_{t}^{(1)}
\end{aligned}
$$

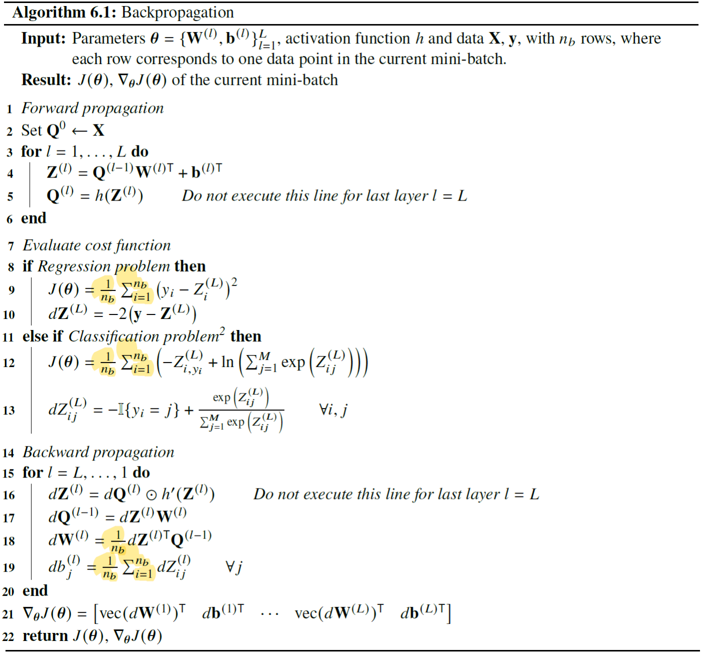
Backpropagation Algorithm



So far, we have only considered backpropagation for one data point $(\mathbf{x}, y)$. However, we do want to compute the cost function $J(\boldsymbol{\theta})$ and its gradients $d\mathbf{W}^{(l)}$ and $d\mathbf{b}^{(l)}$ for the whole mini-batch $\{\mathbf{x}_i, y_i\}_{i=(j-1)n_b+1}^{jn_b}$.Therefore, we run the equations (6.24), (6.26), and (6.27) for all data points in the current mini-batch and average their results for J, $d\mathbf{W}^{(l)}$ and $d\mathbf{b}^{(l)}$. To do this in a computationally efficient manner, we process all n_b data points in the mini-batch simultaneously by stacking them in matrices as we did in (6.14) where each row represent one data point.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1^\mathsf{T} \\ \vdots \\ \mathbf{q}_{n_b}^\mathsf{T} \end{bmatrix}, \quad \mathbf{Z} = \begin{bmatrix} \mathbf{z}_1^\mathsf{T} \\ \vdots \\ \mathbf{z}_{n_b}^\mathsf{T} \end{bmatrix}, \quad d\mathbf{Q} = \begin{bmatrix} d\mathbf{q}_1^\mathsf{T} \\ \vdots \\ d\mathbf{q}_{n_b}^\mathsf{T} \end{bmatrix} \quad \text{and} \quad d\mathbf{Z} = \begin{bmatrix} d\mathbf{z}_1^\mathsf{T} \\ \vdots \\ d\mathbf{z}_{n_b}^\mathsf{T} \end{bmatrix}. \tag{6.28}$$

Backpropagation Algorithm



Algorithm 6.1: Backpropagation

Input: Parameters $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L}$, activation function h and data $\mathbf{X}$, $\mathbf{y}$, with n_b rows, where each row corresponds to one data point in the current mini-batch.

Result: $J(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of the current mini-batch

```
Forward propagation
Set $\mathbf{Q}^0 \leftarrow \mathbf{X}$
for $l = 1, \dots, L$ do
    $\mathbf{Z}^{(l)} = \mathbf{Q}^{(l-1)}\mathbf{W}^{(l)\mathsf{T}} + \mathbf{b}^{(l)\mathsf{T}}$
    $\mathbf{Q}^{(l)} = h(\mathbf{Z}^{(l)})$        Do not execute this line for last layer $l = L$
end

Evaluate cost function
if Regression problem then
    $J(\boldsymbol{\theta}) = \frac{1}{n_b}\sum_{i=1}^{n_b}\left(y_i - Z_i^{(L)}\right)^2$
    $d\mathbf{Z}^{(L)} = -2\left(\mathbf{y} - \mathbf{Z}^{(L)}\right)$
else if Classification problem[2] then
    $J(\boldsymbol{\theta}) = \frac{1}{n_b}\sum_{i=1}^{n_b}\left(-Z_{i,y_i}^{(L)} + \ln\left(\sum_{j=1}^{M}\exp\left(Z_{ij}^{(L)}\right)\right)\right)$
    $dZ_{ij}^{(L)} = -\mathbb{I}\{y_i = j\} + \frac{\exp\left(Z_{ij}^{(L)}\right)}{\sum_{j=1}^{M}\exp\left(Z_{ij}^{(L)}\right)}$        $\forall i, j$

Backward propagation
for $l = L, \dots, 1$ do
    $d\mathbf{Z}^{(l)} = d\mathbf{Q}^{(l)} \odot h'(\mathbf{Z}^{(l)})$        Do not execute this line for last layer $l = L$
    $d\mathbf{Q}^{(l-1)} = d\mathbf{Z}^{(l)}\mathbf{W}^{(l)}$
    $d\mathbf{W}^{(l)} = \frac{1}{n_b} d\mathbf{Z}^{(l)\mathsf{T}}\mathbf{Q}^{(l-1)}$
    $db_j^{(l)} = \frac{1}{n_b}\sum_{i=1}^{n_b} dZ_{ij}^{(l)}$        $\forall j$
end
$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left[\mathrm{vec}(d\mathbf{W}^{(1)})^{\mathsf{T}} \quad d\mathbf{b}^{(1)\mathsf{T}} \quad \cdots \quad \mathrm{vec}(d\mathbf{W}^{(L)})^{\mathsf{T}} \quad d\mathbf{b}^{(L)\mathsf{T}}\right]$
return $J(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
```

Backpropagation Algorithm

Example 6.2: Classification of hand-written digits - first attempt

We consider the example of classifying hand-written digits introduced in Example 6.1. Based on the presented data we train the two models mentioned in end of that example: a logistic regression model (or equivalently a one-layer neural network) and a two-layer neural network.

We use stochastic gradient descent explained in Algorithm 5.3 with the learning rate $\gamma = 0.5$ and a mini-batch size of $n_b = 100$ data points. Since we have $n = 60\,000$ training data points in total, one epoch is completed after $6\,000/100 = 600$ iterations. We run the algorithm for 15 epochs, i.e., 9 000 iterations.

As explained earlier, at each iteration of the algorithm the cost function is evaluated for the current mini-batch of the training data. The value for this cost function is illustrated in blue in the left part of Figure 6.7 and 6.8. In addition, we also compute the miscalssification rate for the 100 data points in the current mini-batch. This is illustrated in the right part of Figure 6.7 and 6.8. Since the current mini-batch consist of 100 randomly selected training data points, the performance fluctuates depending on which mini-batch that is considered.

However, the important measure is how we perform on data which has not been used during training. Therefore, at every 100$^{\text{th}}$ iteration we also evaluate the cost function and the misclassification error on the whole validation dataset of 10 000 data points. This performance is illustrated in red in Figure 6.7 and 6.8.

Logistic regression model

Figure 6.7 illustrates the training of the logistic regression model.

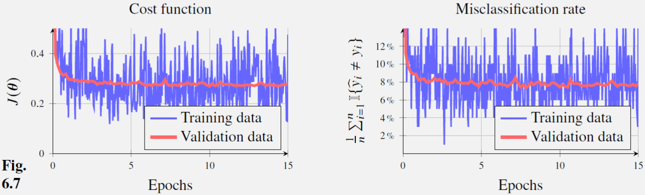
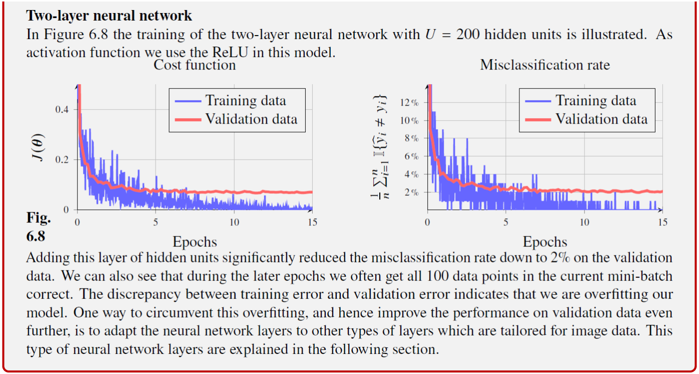


Fig. 6.7

We can see that the performance on validation data improves and already after a few epochs we do not see any additional improvements and we get a misclassification rate of approximately 8% on the validation data.

Backpropagation Algorithm



Two-layer neural network

In Figure 6.8 the training of the two-layer neural network with $U = 200$ hidden units is illustrated. As activation function we use the ReLU in this model.

Cost function
Training data
Validation data
$J(\theta)$
0.4
0.2
0
0
5
10
15
Epochs
Misclassification rate
Training data
Validation data
$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\{\widehat{y}_i \neq y_i\}$
12 %
10 %
8 %
6 %
4 %
2 %
0
5
10
15
Epochs

Fig. 6.8

Adding this layer of hidden units significantly reduced the misclassification rate down to 2% on the validation data. We can also see that during the later epochs we often get all 100 data points in the current mini-batch correct. The discrepancy between training error and validation error indicates that we are overfitting our model. One way to circumvent this overfitting, and hence improve the performance on validation data even further, is to adapt the neural network layers to other types of layers which are tailored for image data. This type of neural network layers are explained in the following section.

Summary

- Neural Networks
- Backpropagation Algorithm