# Pattern Recognition and Machine Learning

## 5. Neural Networks

Seungyong Moon

Seoul National University

April 27th, 2018

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Linear models for the regression and classification

- linear combination of basis functions

$$y(\mathbf{x}, \mathbf{w}) = f(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x}))$$

$$\mathbf{x} : \text{an input vector}$$

$$\mathbf{w} : \text{a set of parameters}$$

$$\{\phi_j\} : \text{a set of basis function}$$

$$f : \text{nonlinear function}$$

# Neural Network

- In neural networks, each basis function is a nonlinear function of a linear combination of input.

$$\begin{aligned} \mathbf{y} &= f(W\mathbf{x} + \mathbf{b}) \\ \mathbf{x} &: \text{an input vector with dimension D} \\ W &: \text{a M} \times \text{D matrix} \\ \mathbf{b} &: \text{a bias vector with dimension D} \\ f &: \text{an activation function} \end{aligned}$$

# Neural Network

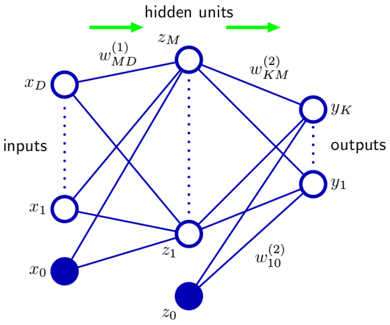


Figure: Network diagram for two-layer neural network

- Neural networks is a series of the transformations(layers).

# Neural Network as Universal Approximator

- Neural networks can be expressed as below.

$$\mathbf{y} = f_n(...f_2(W_2(f_1(W_1\mathbf{x} + \mathbf{b_1})) + \mathbf{b_2})... + \mathbf{b_n})$$

- Our purpose: Given $\{\mathbf{x}_i, \mathbf{y}_i\}_i$, find a function that maps $\mathbf{x}_i$ to $\mathbf{y}_i$ approximately.
- Can we do that using neural network?

- It is proved by George Cybenko in 1989 for sigmoid activation function.

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Weight-space Symmetries

- In neural network, multiple distinct choices for the parameters can give the same mapping function.
- It means that the dimension of parameter(weight) space is extremely high.

# Weight-space Symmetries

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Loss Function of Neural Network

- The loss(error) function of neural network is defined as follows.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

$\mathbf{x}_n$ : an input vector

$\mathbf{y}(\cdot, \cdot)$ : an output of neural network

$\mathbf{w}$ : a parameter vector

$\mathbf{t_n}$ : the real output corresponding to $\mathbf{x}_n$

# Probabilistic Interpretation

## one dimensional case

- Suppose **t** has a univariate normal distribution.

$$\mathbf{t}|_{\mathbf{x},\mathbf{w}} \sim N(y(\mathbf{x},\mathbf{w}), \beta^{-1})$$

- If we apply MLE, we obtain the error function.

$$\frac{\beta}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n\|^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

- Once we have found the optimal $\mathbf{w}$(denote $\mathbf{w}_{ML}$), we can calculate $\beta$.

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}_{ML}) - \mathbf{t}_n\|^2$$

# Probabilistic Interpretation

## K-dimensional case

- Suppose $\mathbf{t}$ has a multivariate normal distribution.

$$\mathbf{t}|_{\mathbf{x},\mathbf{w}} \sim N(y(\mathbf{x},\mathbf{w}), \beta^{-1}I)$$

- If we apply MLE, the error function is also propotional to.

$$\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n\|^2$$

- and $\beta$ is given by

$$\beta_{ML}^{-1} I = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n)(\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n)^T$$

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n\|^2$$

# Binary Classifier

- Think about a binary classifier in which we have a single target variable $t$ such that $t = 1$ denotes class 1 and $t = 0$ denotes class 2.
- How to train the binary classifier using neural network?

# Binary Classifier

- Consider a neural network whose the last activation function is sigmoid function.
- And, suppose t has a Bernoulli distribution of the form

$$t|_{\mathbf{x},\mathbf{w}} \sim B(1, y(\mathbf{x}, \mathbf{w}))$$

- After applying MLE, we get a loss function as follows.

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln (1 - y_n)\}$$

- We call it cross-entropy loss function

# Multi-class Classifier

- Consider the case where each input is assigned to one of K mutually exclusive classes.
- Suppose $\mathbf{t}(= \{t_k\})$ is one-hot encoding vector indicating the class, and the activation function of the output layer is softmax.
- Also, suppose that t has a distribution as follows.

$$p(t_k = 1|\mathbf{x}, \mathbf{w}) = y_k(\mathbf{x}, \mathbf{w})$$

$y_k$ : $k$th element of the output

# Multi-class Classifier

- After calculating the negative log likelihood of the distribution. we can get the loss function.

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}(\mathbf{t}_n)_k \ln y(\mathbf{x}_n, \mathbf{w})$$

# Parameter Optimisation

- Our goal: Given $E(\mathbf{w})$, find the optimal $\mathbf{w}^*$
- How to find?
  - Analytic method
  - Iterative method

# Outline

1 Feed-forward Network Functions
- Neural Network
- Weight-space Symmetries

2 Network Training
- Parameter Optimization
- Descent Optimization

3 Error Backpropagation
- Evaluation of Error-function Derivatives

4 The Hessian Matrix
- Approximation of Hessian
- Exact Evaluation of the Hessian
- Fast Multiplication by the Hessian

# Descent Method

- Descent method

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + t^{(k)}\Delta\mathbf{w}^{(k)}$$

$$t^{(k)} : \text{step size at time k}$$

$$\Delta\mathbf{w}^{(k)} : \text{search direction}$$

# Gradient Descent Method

- General descent method with $\Delta \mathbf{w}^{(k)} = -\nabla E(\mathbf{w}^{(k)})$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - t^{(k)} \nabla E(\mathbf{w}^{(k)})$$

$$t^{(k)} : \text{step size at time k}$$

# Taylor expansion

- The error function can be approximated by

$$E(\mathbf{w}+\mathbf{v}) \approx E(\mathbf{w}) + \nabla E(\mathbf{w})\mathbf{v} + \frac{1}{2}\mathbf{v}^T \nabla^2 E(\mathbf{w})\mathbf{v}$$

- At a local minimum, the Hessian matrix is positive definite.

# Taylor Expansion

- Find the optimal **v** that minimize the quadratic approximation.
- After differentiating the quadratic function with respect to **v**, we can get the following result.

$$
\begin{aligned}
\nabla E(\mathbf{w}) &= -\nabla^2 E(\mathbf{w}) v \\
v &= -(\nabla^2 E(\mathbf{w}))^{-1} \nabla E(\mathbf{w})
\end{aligned}
$$

# Newton's Method

- General descent method with $\Delta \mathbf{w}^{(k)} = -(\nabla^2 E(\mathbf{w}^{(k)}))^{-1} \nabla E(\mathbf{w}^{(k)})$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - t^{(k)} (\nabla^2 E(\mathbf{w}^{(k)}))^{-1} \nabla E(\mathbf{w}^{(k)}))$$

$t^{(k)}$ : step size at time k

# Stochastic Gradient Method

- Suppose that the error function has a form as below.

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{x}_n, \mathbf{w})$$

- Then, the gradient of the error function is

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} \nabla E_n(\mathbf{x}_n, \mathbf{w})$$

- If the number of terms is large, it is very expensive to compute the gradient of the error function at each step.

# Stochastic Gradient Method

- Just pick one data point randomly, and make an update to the weight vector based on the point at each step.

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - t^{(k)} \nabla E_n(\mathbf{x}_n, \mathbf{w}^{(k)})$$

- or splits the data into small batches and compute the gradient on each (mini)batch.

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - t^{(k)} \sum_{i=1}^{M} \nabla E_{n_i}(\mathbf{x}_{n_i}, \mathbf{w}^{(k)})$$

where $\{\mathbf{x}_{n_1}, \mathbf{x}_{n_2}, \ldots, \mathbf{x}_{n_M}\}$ is a batch

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Evaluation of Error-function Derivatives

- Forward propagation of 'value'
- Back propagation of 'gradient'
  - By chain rule

# Matrix Differentiation Revisited

- Suppose $f(\mathbf{x}) = W\mathbf{x}$ where $W$ is a $m \times n$ matrix.
- Then, the gradient of $f_k$ ($k$th element of $f$) with respect to $W_{ij}$ is

$$\frac{\partial f_k}{\partial W_{ij}} = \delta_{ik}\mathbf{x}_j$$

# BackProp in Fully-connected Layers

# Outline

1 Feed-forward Network Functions
- Neural Network
- Weight-space Symmetries

2 Network Training
- Parameter Optimization
- Descent Optimization

3 Error Backpropagation
- Evaluation of Error-function Derivatives

4 The Hessian Matrix
- Approximation of Hessian
- Exact Evaluation of the Hessian
- Fast Multiplication by the Hessian

# Diagonal approximation

- We need the inverse of the Hessian, rather than the Hessian itself.
- Construct diagonal approximation to the Hessian so that we can easily calculate the inverse of the Hessian.

# Diagonal approximation

- Consider a layer $\mathbf{y} = W\mathbf{x}$ where $\mathbf{x}$ is the output of the previous layer.(Don't think about an activation function now)
- Also, consider that an error function that consists of a sum of terms, one for each data point, so that $E = \sum_n E_n$
- Then, the gradient of $E_n$ with respect to $W$ is

$$\frac{\partial E_n}{\partial W_{ij}} = \frac{\partial E_n}{\partial \mathbf{y}_i}\mathbf{x}_j$$

- and the diagonal elements of the Hessian is

$$\frac{\partial^2 E_n}{\partial W_{ij}^2} = \frac{\partial^2 E_n}{\partial \mathbf{y}_i^2}\mathbf{x}_j^2$$

# Diagonal approximation

- $\frac{\partial^2 E_n}{\partial \mathbf{y}_i^2}$ can be found recursively.
- Suppose $f$ is an activation function whose input is $\mathbf{y}$, and $\mathbf{z} = \tilde{W} f(\mathbf{y})$ be the next layer.
- Then, the gradient of $E_n$ with respect to $\mathbf{y}$ is

$$\frac{\partial E_n}{\partial \mathbf{y}_i} = f'(\mathbf{y}_i) \sum_k \frac{\partial E_n}{\partial \mathbf{z}_k} \tilde{W}_{ki}$$

- the diagonal elements of the Hessian is

$$\frac{\partial^2 E_n}{\partial \mathbf{y}_i^2} = f'(\mathbf{y}_i)^2 \sum_k \frac{\partial E_n}{\partial \mathbf{z}_k} \tilde{W}_{ki} + f''(\mathbf{y}_i) \sum_{k'} \sum_k \frac{\partial^2 E_n}{\partial \mathbf{z}_{k'} \partial \mathbf{z}_k} \tilde{W}_{k'i} \tilde{W}_{ki}$$

# Diagonal approximation

- If we neglect off-diagonal elements, we obtain

$$\frac{\partial^2 E_n}{\partial \mathbf{y}_i^2} = f'(\mathbf{y}_i)^2 \sum_k \frac{\partial E_n}{\partial \mathbf{z}_k} \tilde{W}_{ki} + f''(\mathbf{y}_i) \sum_k \frac{\partial^2 E_n}{\partial \mathbf{z}_k^2} \tilde{W}_{ki}^2$$

# Diagonal approximation

- Time complexity: $O(W)$
  - where $W$ is the number of parameters
- In practice, the Hessian is typically non-diagonal.

# Outer Product Approximation

- Suppose a sum-of-squares of error function

$$E = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2$$

($y_n$ and $t_n$ is a scalar value)

- Then, the Hessian of the error function is

$$\nabla^2 E = \sum_{n=1}^{N}\nabla y_n \nabla y_n^T + \sum_{n=1}^{N}(y_n - t_n)\nabla^2 y_n$$

- If the network has been trained on the data set, then $y_n$ will be very close to $t_n$, and the second term will be small.

# Outer Product Approximation

- Therefore the Hessian can be expressed as

$$\nabla^2 E \approx \sum_{n=1}^{N} \nabla y_n \nabla y_n^T$$

- $\nabla y_n$ can be evaluated by back propagation.

# Outer Product Approximation

- Consider the case of the cross-entropy error function for a network with logistic sigmoid activation function.

$$E = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln (1 - y_n)\}$$

- Let $a_n$ be the output vector without activation. That is,

$$y_n = \sigma(a_n) = \frac{1}{1 + \exp(-a_n)}$$

- Then, the error function can be expressed as

$$E = -\sum_{n=1}^{N} \left\{ t_n \ln \frac{1}{1 + \exp(-a_n)} + (1 - t_n) \ln (1 - \frac{1}{1 + \exp(-a_n)}) \right\}$$
$$= \sum_{n=1}^{N} \{\ln (1 + \exp(-a_n)) + (1 - t_n) a_n)\}$$

# Outer Product Approximation

- If we differentiate the equation with respect to W, then we can get the following.

$$
\begin{aligned}
\nabla E &= \sum_{n=1}^{N} \left\{ -\frac{\exp(-a_n)}{1+\exp(-a_n)} \nabla a_n + (1-t_n)\nabla a_n \right\} \\
&= \sum_{n=1}^{N} (y_n - t_n)\nabla a_n
\end{aligned}
$$

# Outer Product Approximation

- Them the Hessian of the error function is,

$$\nabla^2 E = \sum_{n=1}^{N} \frac{\partial y_n}{\partial a_n} \nabla a_n \nabla a_n + (y_n - t_n) \nabla^2 a_n$$

- If we neglect the last term and apply the property of logistic sigmoid function, $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, then we can get the following result.

$$\nabla^2 E \approx \sum_{n=1}^{N} y_n (1 - y_n) \nabla a_n \nabla a_n$$

# Inverse Hessian

- We can use the outer product approximation to approximate the inverse of the Hessian.
- Suppose the Hessian has a form as

$$H = \nabla^2 E \approx \sum_{n=1}^{N} \nabla y_n \nabla y_n^T$$

- and denote $H_k$ a partial sum of $H$

$$H_k = \sum_{n=1}^{k} \nabla y_n \nabla y_n^T$$

# Inverse Hessian

- Then we can get a recurrence relation

$$H_{k+1} = H_k + \nabla y_{k+1} \nabla y_{k+1}^T$$

- and if we apply the following matrix identity

$$(M + \mathbf{v}\mathbf{v}^T)^{-1} = M^{-1} - \frac{(M^{-1}\mathbf{v})(\mathbf{v}^T M^{-1})}{1 + \mathbf{v}^T M^{-1}\mathbf{v}}$$

  on the relation, we obtain

$$H_{k+1}^{-1} = H_k^{-1} - \frac{(H_k^{-1}\nabla y_{k+1})(\nabla y_{k+1}^T H_k^{-1})}{1 + \nabla y_{k+1}^T H_k^{-1} \nabla y_{k+1}}$$

# Inverse Hessian

- We call it Symmetric Rank 1(SR1) method.
- Usually we choose the initial matrix to be $\alpha I$.

# Quasi-Newton Method

- Quasi-Newton nonlinear optimization algorithms gradually build up an approximation to the inverse of the Hessian during training
  - SR1
  - BFGS
  - L-BFGS
- Usually works very well in full batch, but not transfer well to mini-batch setting.

# Finite Difference Method

- Evaluate the second derivative by using finite difference.

$$
\begin{aligned}
\frac{\partial^2 E}{\partial w_{ij} \partial w_{kl}} = \frac{1}{4\epsilon^2} \{ & E(w_{ij} + \epsilon, w_{kl} + \epsilon) - E(w_{ij} + \epsilon, w_{kl} - \epsilon) \\
& - E(w_{ij} - \epsilon, w_{kl} + \epsilon) + E(w_{ij} - \epsilon, w_{kl} - \epsilon)\} + O(\epsilon^2)
\end{aligned}
$$

# Finite Difference Method

- Needs $W^2$ perturbations
  - where $W$ is the number of parameters
- Needs O(W) operations for each forward propagation.
- Time complexity: $O(W^3)$

# Finite Difference Method

- A more efficient version of numerical differentiation can be found by applying central differences to the first derivatives of the error function.

$$\frac{\partial^2 E}{\partial w_{ij} \partial w_{kl}} = \frac{1}{2\epsilon}\left\{ \frac{\partial E}{\partial w_{ij}}(w_{kl}+\epsilon) - \frac{\partial E}{\partial w_{ij}}(w_{kl}-\epsilon) \right\} + O(\epsilon^2)$$

- Time complexity: $O(W^2)$

# Outline


1. Feed-forward Network Functions
   - Neural Network
   - Weight-space Symmetries
2. Network Training
   - Parameter Optimization
   - Descent Optimization
3. Error Backpropagation
   - Evaluation of Error-function Derivatives
4. The Hessian Matrix
   - Approximation of Hessian
   - Exact Evaluation of the Hessian
   - Fast Multiplication by the Hessian

# Exact Evaluation of the Hessian

- The Hessian can be evaluated exactly, using extension of the technique of back propagation.
- See more details in Bishop(1992)

# Outline


1 Feed-forward Network Functions
- Neural Network
- Weight-space Symmetries

2 Network Training
- Parameter Optimization
- Descent Optimization

3 Error Backpropagation
- Evaluation of Error-function Derivatives

4 The Hessian Matrix
- Approximation of Hessian
- Exact Evaluation of the Hessian
- Fast Multiplication by the Hessian

# Fast Multiplication by the Hessian

- The quantity of interest is not the Hessian matrix $H$ itself but the product of $H$ with some vector $\mathbf{v}$.
- Evaluate $\mathbf{v}^T H$ directly.

# Fast Multiplication by the Hessian

- $\mathbf{v}^T H = \mathbf{v}^T \nabla(\nabla E)$
- Consider $\nabla E$ as a function on the weight space, and $v^T \nabla$ as an operator on function space.
- Actually, $v^T \nabla$ is a differential operator.(Think about the definition of directional derivative)
- Let $\mathcal{R}\{\cdot\} = v^T \nabla$
- Trivially, $\mathcal{R}\{\mathbf{w}\} = \mathcal{R}\{I(\mathbf{w})\} = \mathbf{v}$

# Fast Multiplication by the Hessian

- Consider a neural network with 2 layers.

$$a_j = \sum_i w_{ji} x_i$$
$$z_j = h(a_j)$$
$$y_k = \sum_j w_{kj} z_j$$

- Now, we act on these equations using $\mathcal{R}\{\cdot\}$, and we can get the following.

$$\begin{aligned}
\mathcal{R}\{a_j\} &= \sum_i v_{ji} x_i \\
\mathcal{R}\{z_j\} &= h'(a_j)\mathcal{R}\{a_j\} \\
\mathcal{R}\{y_k\} &= \sum_j w_{kj}\mathcal{R}\{z_j\} + \sum_j v_{kj} x_j
\end{aligned} \tag{1}$$

where $v_{ji}$ is the element of the vector $v$ that corresponds to $w_{ji}$

# Fast Multiplication by the Hessian

- Suppose that the error function is a sum of squares function. Then the following holds.

$$\delta_k = \frac{\partial E}{\partial y_k} = y_k - t_k$$
$$\delta_j = \frac{\partial E}{\partial a_j} = h'(a_j) \sum_k w_{kj} \delta_k$$

- If we act on the equation with $\mathcal{R}\{\cdot\}$, we obtain

$$\begin{aligned} \mathcal{R}\{\delta_k\} &= \mathcal{R}\{y_k\} \\ \mathcal{R}\{\delta_j\} &= h''(a_j)\mathcal{R}\{a_j\} \sum_k w_{kj}\delta_k \\ &\quad + h'(a_j) \sum_k v_{kj}\delta_k + h'(a_j) \sum_k w_{kj}\mathcal{R}\{\delta_k\} \end{aligned}$$

# Fast Multiplication by the Hessian

- Finally if we acting on the equation below with the $\mathcal{R}\{\cdot\}$

$$\frac{\partial E}{\partial w_{kj}} = \delta_k z_j$$
$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i$$

we can get

$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{kj}}\right\} = \mathcal{R}\{\delta_k\}x_j + \delta_k\mathcal{R}\{x_j\}$$
$$\mathcal{R}\left\{\frac{\partial E}{\partial w_{ji}}\right\} = x_i\mathcal{R}\{\delta_j\}$$

each of which is corresponding to an element of $\mathbf{v}^T H$