C.M. Bishop's PRML: Chapter 5; Neural Networks

Vasil Khalidov & Miles Hansard

Introduction

The aim is, as before, to find useful decompositions of the target variable;

$$t(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) + \epsilon(\mathbf{x}) \tag{3.7}$$

- $t(\mathbf{x}_n)$ and $\mathbf{x}_n$ are the observations, $n = 1, \ldots, N$.
- $\epsilon(\mathbf{x})$ is the residual error.

Linear Models

For example, recall the (Generalized) Linear Model:

$$y(\mathbf{x}, \mathbf{w}) = \overset{\text{비선형함수}}{\overset{\downarrow}{f}} \left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x}) \right) \tag{5.1}$$

- $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_M)^\top$ is the fixed model basis.
- $\mathbf{w} = (w_0, \ldots, w_M)^\top$ are the model coefficients.
- For regression: $f(\cdot)$ is the identity.
- For classification: $f(\cdot)$ maps to a posterior probability.

Feed-Forward Networks

Feed-forward Neural Networks generalize the linear model

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right) \qquad \text{(5.1 again)}$$

- The basis itself, as well as the coefficients w_j, will be adapted.
- Roughly: the principle of (5.1) will be used twice; once to define the basis, and once to obtain the output.

Activations

Construct M linear combinations of the inputs $x_1, \ldots, x_D$:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{5.2}$$

- a_j are the activations, $j = 1, \ldots, M$.
- $w_{ji}^{(1)}$ are the layer one weights, $i = 1 \ldots D$.
- $w_{j0}^{(1)}$ are the layer one biases.

Each linear combination a_j is transformed by a (nonlinear, differentiable) activation function:

$$z_j = h(a_j) \tag{5.3}$$

Output Activations

The hidden outputs $z_j = h(a_j)$ are linearly combined in layer two:

$$a_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j \tag{5.4}$$

- a_k are the output activations, $k = 1, \ldots, K$.
- $w_{kj}^{(2)}$ are the layer two weights, $j = 1 \ldots D$.
- $w_{k0}^{(2)}$ are the layer two biases.

The output activations a_k are transformed by the output activation function:

$$y_k = \sigma(a_k) \tag{5.5}$$

- y_k are the final outputs.
- $\sigma(\cdot)$ is, like $h(\cdot)$, a sigmoidal function.

The Complete Two-Layer Model

The model $y_k = \sigma(a_k)$ is, after substituting the definitions of a_j and a_k:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\underbrace{\sum_{j=0}^{M} w_{kj}^{(2)} \, h \left(\underbrace{\sum_{i=0}^{D} w_{ji}^{(1)} x_i}_{a_j} \right)}_{a_k} \right) \tag{5.9}$$

- $h(\cdot)$ and $\sigma(\cdot)$ are a sigmoidal functions, e.g. the logistic function.

$$s(a) = \frac{1}{1 + \exp(-a)} \qquad s(a) \in [0, 1]$$

- If $\sigma(\cdot)$ is the identity, then a regression model is obtained.
- Evaluation of (5.9) is called forward propagation.

Network Diagram

The approximation process can be represented by a network:

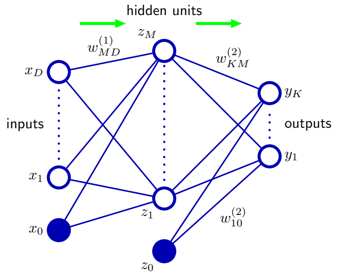

Figure: 5.1

- Nodes are input, hidden and output units. Links are corresponding weights.
- Information propagates 'forwards' from the explanatory variable $\mathbf{x}$ to the estimated response $y_k(\mathbf{x}, \mathbf{w})$.

Properties & Generalizations

- Typically $K \leq D \leq M$, which means that the network is redundant if all $h(\cdot)$ are linear.
- There may be more than one layer of hidden units. cf ResNet
  Skip Gram
- Individual units need not be fully connected to the next layer.
- Individual links may skip over one or more subsequent layers.
- Networks with two (cf. 5.9) or more layers are universal approximators.
- Any continuous function can be uniformly approximated to arbitrary accuracy, given enough hidden units. 중요!!
- This is true for many definitions of $h(\cdot)$, but excluding polynomials.
- There may be symmetries in the weight space, meaning that different choices of $\mathbf{w}$ may define the same mapping from input to output.

Maximum Likelihood Parameters

The aim is to minimize the residual error between $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ and $\mathbf{t}_n$. Suppose that the target is a scalar-valued function, which is Normally distributed around the estimate:

$$p(t|\mathbf{x},\mathbf{w}) = \mathcal{N}\left(t \,\middle|\, y(\mathbf{x},\mathbf{w}), \beta^{-1}\right) \tag{5.12}$$

Then it will be appropriate to consider the sum of squared-errors

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\Big(y(\mathbf{x}_n,\mathbf{w}) - t_n\Big)^2 \tag{5.14}$$

The maximum-likelihood estimate of $\mathbf{w}$ can be obtained by (numerical) minimization:

$$\mathbf{w}_{\mathrm{ML}} = \min_{\mathbf{w}} \; E(\mathbf{w})$$

Maximum Likelihood Precision

Having obtained the ML parameter estimate $\mathbf{w}_{\mathrm{ML}}$, the precision, β can also be estimated. E.g. if the N observations are IID, then their joint probability is

$$p\Big(\{t_1, \ldots, t_N\}\Big|\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \mathbf{w}, \beta\Big) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$

The negative log-likelihood, in this case, is

$$-\log p = \beta E(\mathbf{w}_{\mathrm{ML}}) - \frac{N}{2}\log\beta + \frac{N}{2}\log 2\pi \qquad (5.13)$$

The derivative $\mathrm{d}/\mathrm{d}\beta$ is $E(\mathbf{w}_{\mathrm{ML}}) - \frac{N}{2\beta}$ and so

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}2E(\mathbf{w}_{\mathrm{ML}}) \qquad (5.15)$$

And $1/\beta_{\mathrm{ML}} = \frac{1}{NK}2E(\mathbf{w}_{\mathrm{ML}})$ for K target variables.

Error Surface

The residual error $E(\mathbf{w})$ can be visualized as a surface in the weight-space:

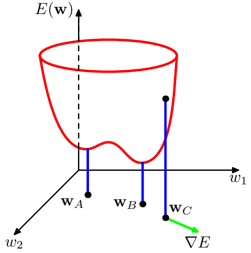

Figure: 5.5

- The error will, in practice, be highly nonlinear, with many minima, maxima and saddle-points.
- There will be inequivalent minima, determined by the particular data and model, as well as equivalent minima, corresponding to weight-space symmetries.

Parameter Optimization

Iterative search for a local minimum of the error:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{5.27}$$

- ∇E will be zero at a minimum of the error.
- τ is the time-step.
- $\Delta\mathbf{w}^{(\tau)}$ is the weight-vector update.
- The definition of the update depends on the choice of algorithm.

Local Quadratic Approximation

The truncated Taylor expansion of $E(\mathbf{w})$ around a weight-point $\hat{\mathbf{w}}$ is

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w}-\hat{\mathbf{w}})^\top \mathbf{b} + \frac{1}{2}(\mathbf{w}-\hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w}-\hat{\mathbf{w}}) \qquad (5.28)$$

- $\mathbf{b} = \nabla E|_{\mathbf{w}=\hat{\mathbf{w}}}$ is the gradient at $\hat{\mathbf{w}}$.
- $(\mathbf{H})_{ij} = \frac{\partial E}{\partial w_i \partial w_j}\Big|_{\mathbf{w}=\hat{\mathbf{w}}}$ is the Hessian $\nabla\nabla E$ at $\hat{\mathbf{w}}$.

The gradient of E can be approximated by the gradient of the quadratic model (5.28); if $\mathbf{w} \simeq \hat{\mathbf{w}}$ then

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w}-\hat{\mathbf{w}}) \qquad (5.31)$$

where $\frac{1}{2}\left((\mathbf{H}+\mathbf{H}^\top)\mathbf{w} - \mathbf{H}\hat{\mathbf{w}} - \mathbf{H}^\top\hat{\mathbf{w}}\right) = \mathbf{H}(\mathbf{w}-\hat{\mathbf{w}})$, as $\mathbf{H}^\top = \mathbf{H}$.

Approximation at a Minimum

Suppose that $\mathbf{w}^\star$ is at a minimum of E, so $\nabla E|_{\mathbf{w}=\mathbf{w}^\star}$ is zero, and

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^\top \mathbf{H}(\mathbf{w}-\mathbf{w}^\star) \qquad (5.32)$$

- $\mathbf{H} = \nabla\nabla E|_{\mathbf{w}=\mathbf{w}^\star}$ is the Hessian.
- The eigenvectors $\mathbf{H}\mathbf{u}_i = \lambda\mathbf{u}_i$ are orthonormal.
- $(\mathbf{w}-\mathbf{w}^\star)$ can be represented in $\mathbf{H}$-coordinates as $\sum_i \alpha_i \mathbf{u}_i$.

Hence the second term of (5.32) can be written

$$\frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^\top \mathbf{H}(\mathbf{w}-\mathbf{w}^\star) = \frac{1}{2}\Big(\sum_i \lambda_i\alpha_i\mathbf{u}_i\Big)^\top\Big(\sum_j \alpha_j\mathbf{u}_j\Big)$$

So that

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i\alpha_i^2 \qquad (5.36)$$

Characterization of a Minimum

The eigenvalues λ_i of $\mathbf{H}$ characterize the stationary point $\mathbf{w}^\star$.

- If all $\lambda_i > 0$, then $\mathbf{H}$ is positive definite ($\mathbf{v}^\top \mathbf{H} \mathbf{v} > 0$).
- This is analogous to the scalar condition $\left.\frac{\partial^2 E}{\partial w^2}\right|_{w^\star} > 0$.
- Zero gradient and positive principle curvatures mean that $E(\mathbf{w}^\star)$ is a minimum.

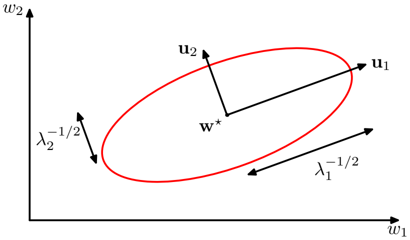

Figure: 5.6

Gradient Descent

The simplest approach is to update **w** by a displacement in the negative gradient direction.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\big(\mathbf{w}^{(\tau)}\big) \qquad (5.41)$$

- This is a steepest descent algorithm.
- η is the learning rate.
- This is a batch method, as evaluation of ∇E involves the entire data set.
- Conjugate gradient or quasi-Newton methods may, in practice, be preferred.
- A range of starting points $\{\mathbf{w}^{(0)}\}$ may be needed, in order to find a satisfactory minimum.

Optimization Scheme

An efficient method for the evaluation of $\nabla E(\mathbf{w})$ is needed.

- Each iteration of the descent algorithm has two stages:
- I. Evaluate derivatives of error with respect to weights (involving backpropagation of error though the network).
- II. Use derivatives to compute adjustments of the weights (e.g. steepest descent).

Backpropagation is a general principle, which can be applied to many types of network and error function.

Simple Backpropagation

The error function is, typically, a sum over the data points $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$. For example, consider a linear model

$$y_k = \sum_i w_{ki} x_i \tag{5.45}$$

The error function, for an individual input $\mathbf{x}_n$, is

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2, \quad \text{where} \quad y_{nk} = y_k(\mathbf{x}_n, \mathbf{w}). \tag{5.46}$$

The gradient with respect to a weight w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})\, x_{ni} \tag{5.47}$$

- w_{ji} is a particular link (x_i to y_j).
- x_{ni} is the input to the link (i-th component of $\mathbf{x}_n$).
- $(y_{nj} - t_{nj})$ is the error output by the link.

General Backpropagation

Recall that, in general, each unit computes a weighted sum:

$$a_j = \sum_i w_{ji} z_i \quad \text{with activation} \quad z_j = h(a_j). \qquad (5.48,5.49)$$

For each error-term: $$\frac{\partial E_n}{\partial w_{ji}} = \underbrace{\frac{\partial E_n}{\partial a_j}}_{\equiv \delta_j} \frac{\partial a_j}{\partial w_{ji}} \qquad (5.50)$$

So, from 5.48: $$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \qquad (5.53)$$

In the network: $$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \quad \text{where } j \to \{k\} \qquad (5.55)$$

Algorithm: $$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad \text{as} \quad \frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} \qquad (5.56)$$

Backpropagation Algorithm

The formula for the update of a given unit depends only on the 'later' (i.e. closer to the output) layers:

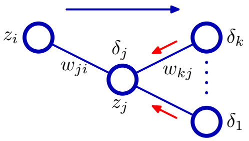

Figure: 5.7

Hence the backpropagation algorithm is:

- Apply input **x**, and forward propagate to find the hidden and output activations.
- Evaluate δ_k directly for the output units.
- Back propagate the δ's to obtain a δ_j for each hidden unit.
- Evaluate the derivatives $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$.

Computational Efficiency

The back-propagation algorithm is computationally more efficient than standard numerical minimization of E_n. Suppose that W is the total number of weights and biases in the network.

- Backpropagation: The evaluation is $O(W)$ for large W, as there are many more weights than units.
- Standard approach: Perturb each weight, and forward propagate to compute the change in E_n. This requires $W \times O(W)$ computations, so the total complexity is $O(W^2)$.

Jacobian Matrix

The properties of the network can be investigated via the Jacobian

$$J_{ki} = \frac{\partial y_k}{\partial x_i} \tag{5.70}$$

For example, (small) errors can be propagated through the trained network:

$$\Delta y_k \simeq \frac{\partial y_k}{\partial x_i} \Delta x_i \tag{5.72}$$

This is useful, but costly, as J_{ki} itself depends on $\mathbf{x}$. However, note that

$$\frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} = \sum_j w_{ji} \frac{\partial y_k}{\partial a_j} \tag{5.74}$$

The required derivatives $\partial y_k / \partial a_j$ can be efficiently computed by backpropagation.