Slide04 (supplemental)

Haykin Chapter 4 (both 2nd and 3rd ed): Multi-Layer Perceptrons

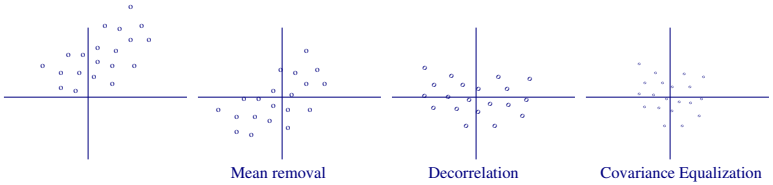
CPSC 636-600

Instructor: Yoonsuck Choe

Heuristic for Making Backprop Perform Better

1. **Sequential vs. batch update**: for large and highly redundant data sets, sequential update works well.
2. **Maximization of information content**: Every training sample should be chosen to maximize information content (we want to search more of the weight space).
   - Use examples that result in the largest error.
   - Use examples that are radically different from those previously used.
   - Randomize (shuffle) input presentation order.
   - Use an *emphasizing* scheme: present more difficult inputs. Issues: input distribution is distorted, and outlier or mislabeled input can cause serious problems.

Heuristic for Backprop (cont'd)

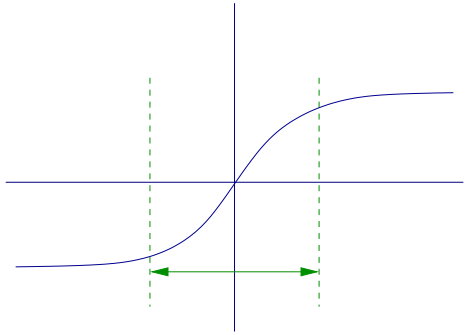


3. **Activation function**: Usually learning is faster with *antisymmetric* activation functions. $\phi(-v) = -\phi(v)$. A popular example is $\phi(v) = a\tanh(bv)$. Note: $\phi'(v) = ab(1 - \tanh^2(bv))$.
4. **Target values**: target values should be within the range of the sigmoid activation function.
5. **Normalizing the inputs**: *preprocessing* to make certain statistical properties hold over the entire training set is important.

Heuristic for Backprop (cont'd)

6. **Initialization**: Small initial weight values are usually good, since they prevent saturation of activity, but it is not good either to have too small weights. One suggestion is to initialize the weights to random values from a uniformly random distribution with zero mean and standard deviation of $\sigma_w = m^{-1/2}$, where m is the number of connections for that neuron.
7. **Learning from hints**: Include prior information about the function to be learned $f(\cdot)$, e.g., invariance properties, symmetries, etc.
8. **Learning rates**: hidden layer should have higher learning rate that output layer (output layer tends to have larger local gradients). Neurons with many inputs should have smaller learning rates.

Weight Initialization



- Given an input distribution (zero mean, unit variance), what is the distribution of the *induced local field* (the weighted sum) that gets plugged into the sigmoid $\phi(\cdot)$?
- We want this value v to be within the ramp region of the sigmoid.
- For this, what should the weight distribution look like?

Weight Initialization (cont'd)

- **Input**: $\mu_y = E[y_i] = 0, \sigma_y^2 = E[(y_i - \mu_y)^2] = E[y^2] = 1.$ Also assume $E[y_i y_k] = 1$ if $k = i$ and 0 if $k \neq i$ (inputs are uncorrelated).
- **Weights**: $\mu_w = E[w_{ji}] = 0$, and $\sigma_w = E[(w_{ji} - \mu_w)^2] = E[w_{ji}^2]$.
- **Induced local field**:

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^{m} w_{ji} y_i\right] = \sum_{i=1}^{m} E[w_{ji}]E[y_i] = 0.$$

$$\sigma_v^2 = E\left[(v_j - \mu_v)^2\right] = E[v_j^2] = E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji} w_{jk} y_i y_k\right]$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{m} E[w_{ji} w_{jk}]E[y_i y_k] = \sum_{i=1}^{m} E[w_{ji}^2] = m\sigma_w^2.$$

We want to adjust $\sigma_v = m^{1/2}\sigma_w$ to fit within the the ramp of $\phi(\cdot)$.

Supervised Learning as an Optimization Problem

- Supervised training of MLP as a problem of *numerical optimization* of $\mathcal{E}_{\text{av}}(\mathbf{w})$ averaged over all input samples.
- Taking the Taylor series expansion:

$$\begin{aligned}\mathcal{E}_{\text{av}}(\mathbf{w}(n) + \Delta\mathbf{w}(n)) &= \mathcal{E}_{\text{av}}(\mathbf{w}(n)) + \mathbf{g}^T(n)\Delta\mathbf{w}(n) \\ &+ \tfrac{1}{2}\Delta\mathbf{w}^T(n)\mathbf{H}(n)\Delta\mathbf{w}(n) + HOT,\end{aligned}$$

where

$$\mathbf{g}(n) = \left.\frac{\partial \mathcal{E}_{\text{av}}(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(n)} \text{ and } \mathbf{H}(n) = \left.\frac{\partial^2 \mathcal{E}_{\text{av}}(\mathbf{w})}{\partial \mathbf{w}^2}\right|_{\mathbf{w}=\mathbf{w}(n)}$$

- First-order method: gradient descent (linear approx)

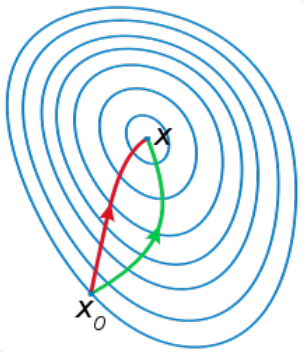
$$\Delta\mathbf{w}(n) = -\eta\mathbf{g}(n)$$

- Second-order method: *Newton's method* (quadratic approx)[a]

$$\Delta\mathbf{w}^*(n) = -\mathbf{H}^{-1}(n)\mathbf{g}(n)$$

[a] Note: typo in this eq in the book (need to add "-")

Newton's Methods vs. Grad. Descent



source: wikipedia

- Gradient descent (green): linear search
- Newton's method (red): quadratic search.

Conjugate Gradient Method: Introduction

- Newton's method, using the inverse of the Hessian $\mathbf{H}^{-1}(n)$, has limitations:
  - Computation of the Hessian can be expensive,
  - The Hessian needs to be nonsingular (for the inverse), which is rarely the case.
  - When the cost function is non-quadratic, convergence is not guaranteed.
- **Conjugate gradient method** overcomes the above problems while accelerating convergence (it's a second-order method).

Quadratic Function Minimization

- Consider minimizing the quadratic function ($\mathbf{A}$ is sym. pos. def.):

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c$$

which is minimized for

$$\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}.$$

So, the minimization problem turns into solving a system of linear equations: $\mathbf{A}\mathbf{x}^* = \mathbf{b}$.

- **Conjugate vectors**: Given an matrix $\mathbf{A}$, nonzero vectors $\mathbf{s}(0), \mathbf{s}(1), ...\mathbf{s}(W-1)$ is $\mathbf{A}$-conjugate if

$$\mathbf{s}^T(n)\mathbf{A}\mathbf{s}(j) = 0 \text{ for all } n \neq j.$$

When $\mathbf{A} = \mathbf{I}$, the meaning is clear: The vectors are orthogonal.

Conjugate Vectors: Properties

- The square root of the matrix $\mathbf{A}$ is defined as: $\mathbf{A} = \mathbf{A}^{1/2}\mathbf{A}^{1/2}$ (existence and uniqueness is guaranteed for pos. semidefinite matrices).
- Since $\mathbf{A}$ is **symmetric** positive definite,

$$\mathbf{A}^T = \left(\mathbf{A}^{1/2}\mathbf{A}^{1/2}\right)^T = (\mathbf{A}^{1/2})^T(\mathbf{A}^{1/2})^T$$

Thus, we get $\mathbf{A}^{1/2} = (\mathbf{A}^{1/2})^T$.

- Given any conjugate vectors $\mathbf{x}_i$ and $\mathbf{x}_j$,

$$\begin{aligned}\mathbf{x}_i A\mathbf{x}_j &= \mathbf{x}_i^T\left(\mathbf{A}^{1/2}\mathbf{A}^{1/2}\right)\mathbf{x}_j = \mathbf{x}_i^T\left((\mathbf{A}^{1/2})^T\mathbf{A}^{1/2}\right)\mathbf{x}_j \\ &= \mathbf{x}_i^T(\mathbf{A}^{1/2})^T\mathbf{A}^{1/2}\mathbf{x}_j = \left(\mathbf{A}^{1/2}\mathbf{x}_i\right)^T\mathbf{A}^{1/2}\mathbf{x}_j = 0.\end{aligned}$$

- So, transforming any (nonzero) vector $\mathbf{x}_i$ into:

$$\mathbf{v}_i = \mathbf{A}^{1/2}\mathbf{x}_i$$

results in vectors $\mathbf{v}_i$ that are mutually orthogonal.

Conjugate Vectors: Properties (Lin. Indep.)

The conjugate vectors are *linearly independent*, so they can form a basis to span the vector space.

- **Proof**: Assume they are not linearly independent:

$$\mathbf{s}(0) = \sum_{j=1}^{W-1}\alpha_j\mathbf{s}(j).$$

Multiplying both sides by $\mathbf{A}\mathbf{s}(0)$, we get:

$$\mathbf{s}^T(0)\mathbf{A}\mathbf{s}(0) = \sum_{j=1}^{W-1}\alpha_j\mathbf{s}^T(j)\mathbf{A}\mathbf{s}(0).$$

But, this is 0, since $\mathbf{s}^T(0)\mathbf{A}\mathbf{s}(0) = 0$ since these are conjugate vectors. However, $\mathbf{A}$ is positive definite and the vectors are nonzero, the sum cannot be 0, a contradiction. Thus, the conj. vectors are not linearly dependent.

Conjugate Direction Method

- For a quadratic error function $f(\mathbf{x})$

  $$\mathbf{x}(n+1) = \mathbf{x}(n) + \eta(n)\mathbf{s}(n), n = 0, 1, ..., W-1,$$

  where $\mathbf{x}(0)$ is an arbitrary initial vector and $\eta(n)$ is defined by

  $$f(\mathbf{x}(n) + \eta(n)\mathbf{s}(n)) = \min_{\eta} f(\mathbf{x}(n) + \eta\mathbf{s}(n)).$$

  The part where $\eta(n)$ is picked is called a *line search*.

- So, in sum, both the *direction* and the *distance* are determined.

Conjugate Direction Method (cont'd)

Observations

- Since the conjugate vectors are linearly independent, they form a basis that spans the vector space of $\mathbf{x}$.
- The line search results in the following formula for η:

  $$\eta(n) = -\frac{\mathbf{s}^T(n)\mathbf{A}\epsilon(n)}{\mathbf{s}^T(n)\mathbf{A}\mathbf{s}(n)},$$

  where $\epsilon(n) = \mathbf{x}(n) - \mathbf{x}^*$ is the error vector.
  - But $\mathbf{x}^*$ is unknown!
  - $\mathbf{A}(\mathbf{x} - \mathbf{x}^*) = (\mathbf{A}\mathbf{x} - \mathbf{b}) - \underbrace{(\mathbf{A}\mathbf{x}^* - \mathbf{b})}_{\text{This is 0.}} = (\mathbf{A}\mathbf{x} - \mathbf{b}).$
- Starting from an arbitrary $\mathbf{x}(0)$, the method reaches the minimum in at most W iterations.

Conjugate Direction Method (cont'd)

- For each iteration n, $\mathbf{x}(n+1)$ minimizes $f(\mathbf{x})$ over a linear vector space $\mathcal{D}_n$ that passes through an arbitrary point $\mathbf{x}(0)$, and is spanned by the $\mathbf{A}$-conjugate vectors:

  $$\mathbf{x}(n+1) = \arg\min_{\mathbf{x}\in\mathcal{D}_n} f(\mathbf{x}),$$

  where

  $$\mathcal{D}_n = \left\{ \mathbf{x}(n) \,\middle|\, \mathbf{x}(n) = \mathbf{x}(0) + \sum_{j=0}^{n} \eta(j)\mathbf{s}(j) \right\}.$$

- The whole scheme depends on the existence of the conjugate vectors $\mathbf{s}(n)$: Solution—*conjugate gradient method*.

Conjugate Gradient Method

Determine successive conjugate vectors $\{\mathbf{s}(n)\}$ in a sequential manner at successive steps.

- **Residual**: $\mathbf{r}(n) = \mathbf{b} - \mathbf{A}\mathbf{x}(n)$ (note: this is $-\partial f/\partial \mathbf{x}$).
- **Recursive step**: Find the next step $\mathbf{s}(n)$ as

  $$\mathbf{s}(n) = \mathbf{r}(n) + \beta(n)\mathbf{s}(n-1), n = 1, 2, ..., W-1,$$

  where the scaling factor $\beta(n)$ is determined as

  $$\beta(n) = -\frac{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{r}(n)}{\mathbf{s}^T(n-1)\mathbf{A}\mathbf{s}(n)}.$$

  This results in conjugate vectors $\mathbf{s}(n)$!
- **Problem**: we need to know $\mathbf{A}$, with respect to which the vectors are conjugate.

Conjugate Gradient Methods: Alternative $\beta(n)$s

We can evaluate the scaling factor $\beta(n)$ without an explicit knowledge of the matrix $\mathbf{A}$, based only on the residuals:

- Polak-Ribiére formula

$$\beta(n) = -\frac{\mathbf{r}^T(n)(\mathbf{r}(n) - \mathbf{r}(n-1))}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)}.$$

- Fletcher-Reeves formula

$$\beta(n) = -\frac{\mathbf{r}^T(n)\mathbf{r}(n)}{\mathbf{r}^T(n-1)\mathbf{r}(n-1)}.$$

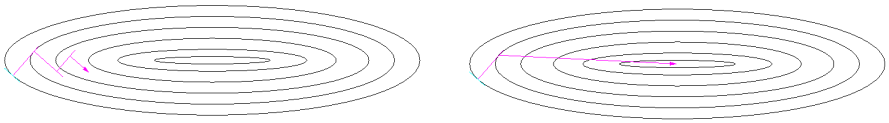
Polak-Ribiére form is superior for nonquadratic cost functions, and is guaranteed to converge if $\beta = \max(\beta_{\mathrm{PR}}, 0)$.

Conjugate Gradient Method: Estimating $\eta(n)$

Line search procedure estimates the proper $\eta(n)$:

- **Bracketing phase**: find a nontrivial interval that is known to contain the minimum.
- **Sectioning phase**: divide the bracket into sections, leading to successively narrower brackets.
- These two steps above can be achieve by *inverse parabolic approximation*.
- Repeated application of the two phases results in good estimation of the point of minimum.

Conjugate Gradient Method vs. Gradient Descent



- Gradient descent can oscillate badly.
- Conjugate gradient method can directly bring you to the minimum in one step (plus the initial step) if the cost function is quadratic (for a 2D case).

Figure from Gutierrez-Osuna's 636 notes.

Application of CGM to NN Learning

- Approximate the cost function $\mathcal{E}_{\mathrm{av}}(\mathbf{w})$ by a quadratic function (ignore HOT).
- Formulate the computation of coefficients $\beta(n)$ and $\eta(n)$ so as to only require gradient information: avoid calculating the Hessian.

Quadratic function $f(\mathbf{x})$	Cost function $\mathcal{E}_{\mathrm{av}}(\mathbf{w})$
Parameter vector $\mathbf{x}(n)$	Weight vector $\mathbf{w}(n)$
Gradient vector $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$	Gradient vector $\mathbf{g} = \frac{\partial \mathcal{E}_{\mathrm{av}}}{\partial \mathbf{w}}$
Matrix $\mathbf{A}$	Hessian matrix $\mathbf{H}$