CSCE 790: Neural Networks and Their Applications

AIISC and Dept. Computer Science and Engineering
Email: vignar@sc.edu

September 14, 2023

Projects

Your overall final course letter grade will be determined by your grades on the following assessments.

- Tentative title due 28th September
- Abstract

Homework 1 - Due 28-**Sep**

Algorithm (Generic optimization algorithm)

At each iteration k,

- $\theta^{k+1} = \theta^k + \alpha^k d^k$
- *If* $\nabla f(\theta^k) \neq 0$, *then the direction* d^k *is chosen so that* $(\nabla f(\theta^k))' d^k < 0$.
- *The step size* $\alpha^k > 0$ *is chosen such that* $f(\theta^k + \alpha^k d^k) < f(\theta^k)$.
- *Principal example:*

$$\theta^{k+1} = \theta^k - \alpha^k D^k \nabla f(\theta^k), \quad d^k = -D^k \nabla f(\theta^k),$$

  - $D^k \succ 0$.
  - $(\nabla f(\theta^k))' \cdot d^k = (\nabla f(\theta^k))'(-D^k \nabla f(\theta^k)) < 0$.

Convergence issues

- Only convergence to stationary points (critical points, $\nabla f(\theta) = 0$) can be guaranteed.
- Even convergence to a single limit is difficult to guarantee.
- There is a danger of non-convergence if the directions d^k tend to be orthogonal to $\nabla f(\theta^k)$, i.e., $\left\langle \nabla f(\theta^k), d^k \right\rangle = 0$.

Step size selection

There are in principle two types of step size selections, called exact and inexact line search.

Definition (Exact line search)

Exactly find the best step size for the current iteration, i.e.,

$$\alpha^k = \arg\min_{\alpha \geq 0} \underbrace{f(\theta^k + \alpha d^k)}_{h(\alpha)}.$$

Examples include *minimization rule* and *bisection rule*.

Definition (Inexact line search)

Approximately find a step size with essential reduction in the function value.

Step size selection

x in the figure should be replaced with θ for consistency.

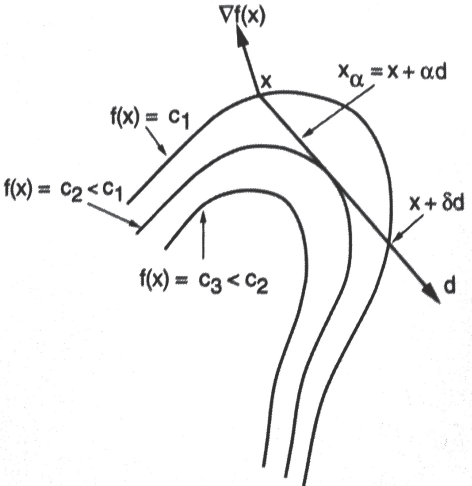

Figure: Orthogonality of gradient to level sets.

Step size selection

There are a number of exact and inexact rules for choosing the step size α^k in a gradient method. We list some that are used widely in practice.

1. Minimization Rule: $f(\theta^{k+1}) = f(\theta^k + \alpha^k d^k)$,

$$\alpha^k = \arg\min_{\alpha} f(\theta^k + \alpha d^k)$$

   α^k is such that the cost function f is minimized along the direction d^k

2. Limited Minimization Rule (because sometimes you are not allowed to choose α too large)

$$f(\theta^k + \alpha^k d^k) = \min_{\alpha \in [0,s]} f(\theta^k + \alpha d^k), \quad s > 0, \quad \text{fixed}$$

Step size selection

3. Constant Stepsize: $\alpha^k = s$, $k = 0, 1, 2, \ldots$ (very simple to implement)
   - 3a. If s is too large, divergence will occur. If s is too small, the rate of convergence may be very slow
   - 3b. Useful only for problems where an appropriate constant stepsize value is known or can be determined fairly easily
4. Diminishing step size

Only critical points are identified in ideal case

- Minimizers or saddle points
- Local vs Global minimizers
- Region of convergence

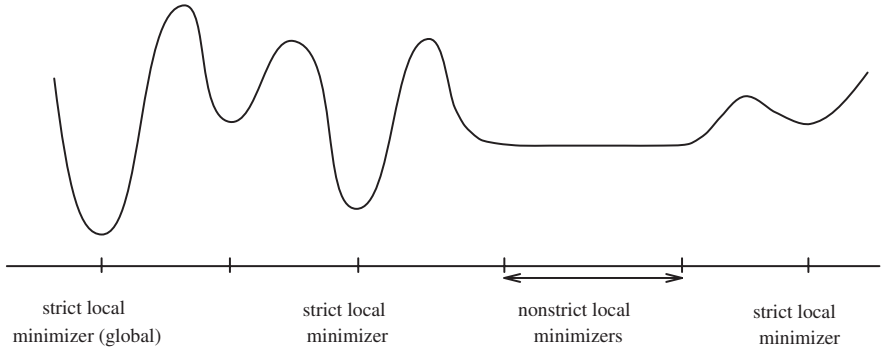

Figure: Examples of local minimizers

Neural network weight selection and training

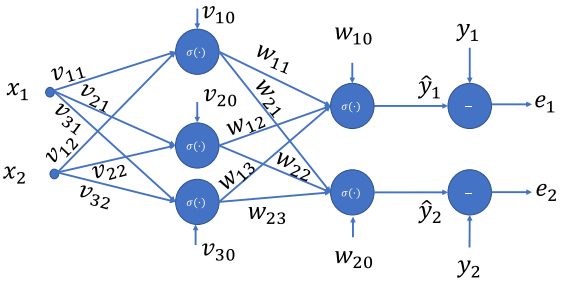

Figure: Error-credit assignment problem

- For a NN to function as desired, their weights and biases need to be selected appropriately
- It was for many years unknown, how to use the error to tune the weights of each layer - 'error-credit assignment problem'

Lemma: Chain rule

Proposition

Let $f : \mathbb{R}^k \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^n$ be smooth, i.e., C^∞. Let $h : \mathbb{R}^k \to \mathbb{R}^n$ be defined by $h(\theta) = g(f(\theta))$. Then

$$\nabla h(\theta) = \nabla f(\theta) \nabla g(f(\theta)), \quad \forall \theta \in \mathbb{R}^k.$$

Back-propagation - Training a 1-layer NN - Forward pass

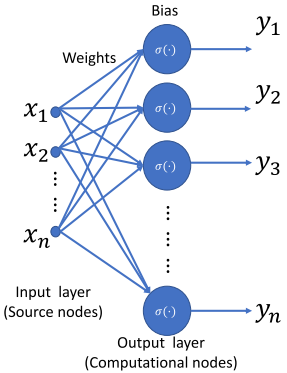

Figure: Single layer NN

$$y_i = \sigma\left(\sum_{j=1}^{n} v_{ij}x_j + v_{i0}\right), \quad i = 1, 2, \ldots, n.$$

$$y = \sigma(Vx),$$

$$x = \begin{pmatrix} 1 \\ \vdots \\ x_n \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix},$$

$$V = \begin{pmatrix} v_{10} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n0} & \cdots & v_{nn} \end{pmatrix}$$

Back-propagation - Training a 1-layer NN - Backward pass

- Let the target outputs for each output neuron be denoted as t_i (note there is a change in notation from our previous lectures)
- Then the error at each output neuron can be defined as $e_i = t_i - y_i$
- Define the loss function as $L(e)$, where $e = (e_1, \ldots, e_n)'$.
- Then, we have

$$e_i = t_i - \sigma\left(\sum_{j=1}^{n} v_{ij} x_j + v_{i0}\right), \quad i = 1, 2, \ldots, n.$$

$$e = t - \sigma\left(Vx\right),$$

where $t = (t_1, \ldots, t_n)'$.

Back-propagation - Training a 1-layer NN - Backward pass

- $e_i = t_i - \sigma\left(\sum_{j=1}^{n} v_{ij}x_j + v_{i0}\right), \quad i = 1, 2, \ldots, n.$
- $e = t - \sigma\left(Vx\right)$
- Loss function - $L(e)$
- Optimization problem -

$$\min_{v_{ij}\in\mathbb{R}, i=1,\ldots,n, j=1,\ldots,n+1} L(e) \tag{1}$$

$$\min_{V\in\mathbb{R}^{n\times n+1}} L(e) \tag{2}$$

Back-propagation - Training a 1-layer NN - Backward pass

Algorithm (Gradient algorithm)

At each iteration k,

- $\min_\theta f(\theta)$
- $\theta^{k+1} = \theta^k + \alpha^k d^k$
- *If $\nabla f(\theta^k) \neq 0$, then the direction d^k is chosen so that $(\nabla f(\theta^k))' d^k < 0$.*
- *The step size $\alpha^k > 0$ is chosen such that $f(\theta^k + \alpha^k d^k) < f(\theta^k)$.*
- *Principal example:*

$$\theta^{k+1} = \theta^k - \alpha^k D^k \nabla f(\theta^k),$$

- $d^k = -D^k \nabla f(\theta^k)$

Algorithm (Training algorithm)

At each iteration k,

- $\min_{V \in \mathbb{R}^{n \times n+1}} L(e)$
- $V^{k+1} = V^k + \alpha^k d^k$
- $V^{k+1} = V^k - \alpha^k \nabla L(V^k)$,
- $d^k = -D^k \nabla L(V^k)$, $D^k = I$

Algorithm (Training algorithm)

At each iteration k,

- $v_{ij}^{k+1} = v_{ij}^k - \alpha^k \nabla L(v_{ij}^k)$,

Back-propagation - Training a 1-layer NN - Example

- $e_i = t_i - \sigma\left(\sum_{j=1}^{n} v_{ij}x_j + v_{i0}\right), \quad i = 1, 2, \ldots, n.$
- Let $\sum_{j=1}^{n} v_{ij}x_j + v_{i0} = z$
- $v_{ij}^{k+1} = v_{ij}^{k} - \alpha^k \nabla L(v_{ij}^{k})$, L is the loss function
- Chain rule

$$\nabla L(v_{ij}^{k}) = \frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial e_i}\frac{\partial e_i}{\partial v_{ij}} = \frac{\partial L}{\partial e_i}\frac{\partial e_i}{\partial y_i}\frac{\partial y_i}{\partial v_{ij}}$$
$$= \frac{\partial L}{\partial e_i}\frac{\partial e_i}{\partial y_i}\frac{\partial y_i}{\partial \sigma}\frac{\partial \sigma}{\partial v_{ij}} = \frac{\partial L}{\partial e_i}\frac{\partial e_i}{\partial y_i}\frac{\partial y_i}{\partial \sigma}\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial v_{ij}} \tag{3}$$

Two-layer feedforward NN - Forward pass

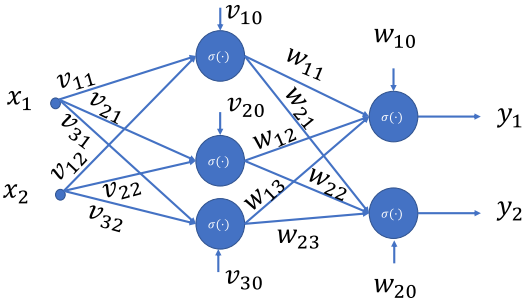

Figure: Two-layer feedforward NN

- Let the output of the hidden layer be $\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$
- $$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \sigma \begin{pmatrix} v_{10} + v_{11}x_1 + v_{12}x_2 \\ v_{20} + v_{21}x_1 + v_{22}x_2 \\ v_{30} + v_{31}x_1 + v_{32}x_2 \end{pmatrix} = \sigma \left[\begin{pmatrix} v_{10} & v_{11} & v_{12} \\ v_{20} & v_{21} & v_{22} \\ v_{30} & v_{31} & v_{32} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \right] = \sigma(Vx)$$
- $z_l = \sigma(\sum_{j=0}^{2} v_{lj} x_j), \quad l = 1, 2, 3.$

Two-layer feedforward NN - Forward pass

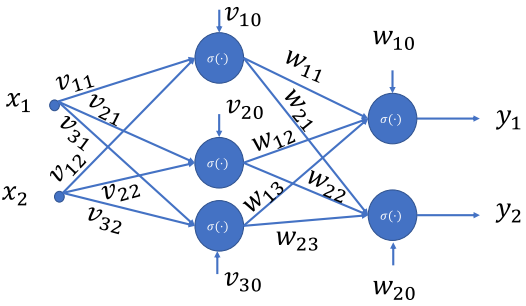

Figure: Two-layer feedforward NN

- Let the output of the NN be $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$
- $$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \sigma \begin{pmatrix} w_{10} + w_{11}z_1 + w_{12}z_2 + w_{13}z_3 \\ w_{20} + w_{21}z_1 + w_{22}z_2 + w_{23}z_3 \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} 1 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} \right] = \sigma(Wz)$$
- $y_i = \sigma(\sum_{l=0}^{3} w_{il} z_l), \quad i = 1, 2.$

Two-layer feedforward NN - Forward pass - Simplified notations

- $z_l = \sigma(\sum_{j=0}^{2} v_{lj} x_j), \quad l = 1, 2, 3.$
- $y_i = \sigma(\sum_{l=0}^{3} w_{il} z_l), \quad i = 1, 2.$
- $u_i^2 = \sum_{l=0}^{3} w_{il} z_l$
- $u_l^1 = \sum_{j=0}^{2} v_{lj} x_j$
- $y_i = \sigma(u_i^2)$ and $z_l = \sigma(u_l^1)$

Two-layer feedforward NN - Derivatives

- $\frac{\partial y_i}{\partial w_{il}} = \sigma'(u_i^2) z_l$
- $\frac{\partial y_i}{\partial z_l} = \sigma'(u_i^2) w_{il}$
- $\frac{\partial z_l}{\partial v_{lj}} = \sigma'(u_l^1) x_j$
- $\frac{\partial z_l}{\partial x_j} = \sigma'(u_i^2) v_{lj}$

Example activation function : Sigmoid activation

$$\sigma(s) = \frac{1}{1 + \exp^{-s}}$$

Derivative:

$$\sigma'(s) = \sigma(s)(1 - \sigma(s))$$

Two-layer feedforward NN - Backward pass

Back-propagation rules (dropping iteration index k for simplicity)

- $w_{il} = w_{il} - \alpha \frac{\partial L}{\partial w_{il}}$
- $v_{lj} = v_{lj} - \alpha \frac{\partial L}{\partial v_{lj}}$

L here is the loss/cost function

Two-layer feedforward NN - Backward pass - Example - Quadratic error

Consider the following expression for cost

- $L = \frac{1}{2} e'e = \frac{1}{2} \sum_{i=1}^{2} e_i^2$
- $e_i = t_i - y_i$

t_i is the target/desired output

Two-layer feedforward NN - Backward pass - Example - Required gradients

- $\frac{\partial L}{\partial w_{il}} = \frac{\partial L}{\partial u_i^2} \frac{\partial u_i^2}{\partial w_{il}} = [\frac{\partial L}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial u_i^2}] \frac{\partial u_i^2}{\partial w_{il}}$
- $\frac{\partial L}{\partial u_i^2} = -\sigma'(u_i^2) e_i$
- $\frac{\partial L}{\partial w_{il}} = -z_l[\sigma'(u_i^2) e_i]$

Two-layer feedforward NN - Backward pass - Example - Required gradients

- $\frac{\partial L}{\partial v_{lj}} = \frac{\partial L}{\partial u_l^1} \frac{\partial u_l^1}{\partial v_{lj}} = \left[\sum_{i=1}^{2} \frac{\partial L}{\partial u_i^2} \frac{\partial u_i^2}{\partial z_l} \frac{\partial z_l}{\partial u_l^1} \right] \frac{\partial u_l^1}{\partial v_{lj}}$
- $\frac{\partial L}{\partial u_l^1} = -\sigma'(u_l^1) \sum_{i=1}^{2} w_{il} [\sigma'(u_i^2) e_i]$
- $\frac{\partial L}{\partial v_{lj}} = -x_j [\sigma'(u_l^1) \sum_{i=1}^{2} w_{il} [\sigma'(u_i^2) e_i]]$

Two-layer feedforward NN - Backward pass - Example - Simplification

- These equations can be considerably simplified by introducing the notion of a backward recursion through the network
- Define the backpropagated error for layers 2 and 1 respectively as
  - $\delta_i^2 = -\frac{\partial L}{\partial u_i^2} = \sigma'(u_i^2)e_i$
  - $\delta_l^1 = \frac{\partial L}{\partial u_l^1} = \sigma'(u_l^1)\sum_{i=1}^{2} w_{il}\delta_i^2$
- Assuming sigmoid activation function at each layer, we have
  - $\delta_i^2 = y_i(1-y_i)e_i$
  - $\delta_l^1 = z_l(1-z_l)\sum_{i=1}^{2} w_{il}\delta_i^2$

Improvements to gradient descent?

- Several improvements can be made to correct deficiencies in gradient descent NN training algorithms
- These can be applied at each layer of a multilayer NN when using backpropagation tuning
- Deficiencies are:
  1. Gradient-based minimization algorithms provide only a local minimum
  2. Verification that gradient descent decreases the cost function is based on an approximation
- Improvements in performance are given by:
  1. selecting better initial conditions
  2. using learning with *momentum*
  3. using an adaptive learning rate α.

Learning with momentum

- An improved version of gradient descent is given by the *Momentum Gradient Algorithm*
- Example:

$$V(k+1) = \beta V(k) - \alpha(1-\beta)\frac{\partial L(k)}{\partial V(k)}$$

- $\beta < 1$ is a positive momentum parameter and $\alpha < 1$ is the positive learning rate
- β is generally selected near 1 (e.g. 0.95)
- This corresponds in discrete-time dynamical system terms to moving the system pole from $z = 1$ to the interior of the unit circle, and adds stability in a manner similar to friction effects in mechanical systems

Challenges in training NN (see Ch. 7 and 8 in DL, 2016))

- Optimization in general is an extremely difficult task
- In ML, the difficulty in optimization is generally avoided by carefully designing the objective function and constraints to ensure that the optimization problem is convex
- When training neural networks, we must confront the general nonconvex case

Ill conditioning

- Some challenges arise even when optimizing convex functions
- Of these, the most prominent is ill-conditioning of the Hessian matrix H

$$f(\theta - \alpha \nabla f(\theta)) \approx f(\theta) - \alpha [\nabla f(\theta)]' \nabla f(\theta) + \alpha^2 [\nabla f(\theta)]' \frac{H(\theta)}{2} \nabla f(\theta)$$

- Ill-conditioning of the gradient becomes a problem when the net sum to the cost is dominated by the term with Hessian
- One can monitor the squared gradient norm $[\nabla f(\theta)]' \nabla f(\theta)$ and the $[\nabla f(\theta)]' \frac{H(\theta)}{2} \nabla f(\theta)$ term.
- In many cases, the gradient norm does not shrink significantly throughout learning, but the second term grows by more than an order of magnitude.
- The result is that learning becomes very slow despite the presence of a strong gradient because the learning rate must be shrunk to compensate for even stronger curvature