Elective Course
Course Code: CS4103
Autumn 2025-26

Lecture #42

Artificial Intelligence for Data Science

Week-12:
MACHINE LEARNING (Part X)
Neural Network Learning

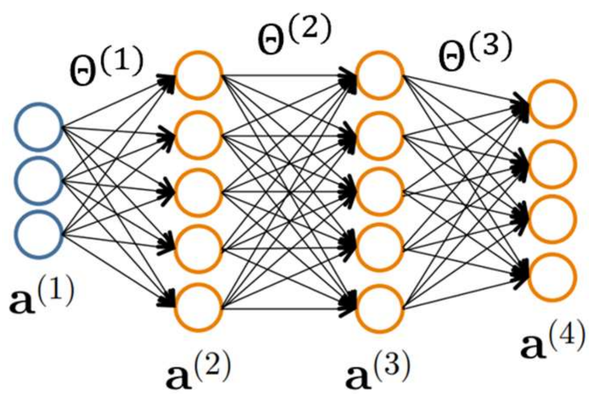
Course Instructor:
Dr. Monidipa Das
Assistant Professor
Department of Computational and Data Sciences
Indian Institute of Science Education and Research Kolkata, India 741246

Forward Propagation

- Given one labeled training instance $(\mathbf{x},\ y)$:

Forward Propagation

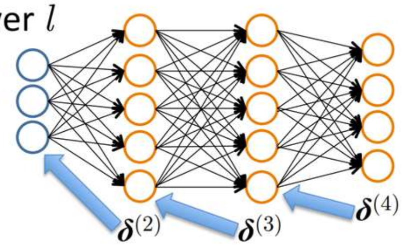
- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$



Backpropagation Intuition: Gradient Computation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

(#layers L = 4)



Element-wise product .*

Backpropagation

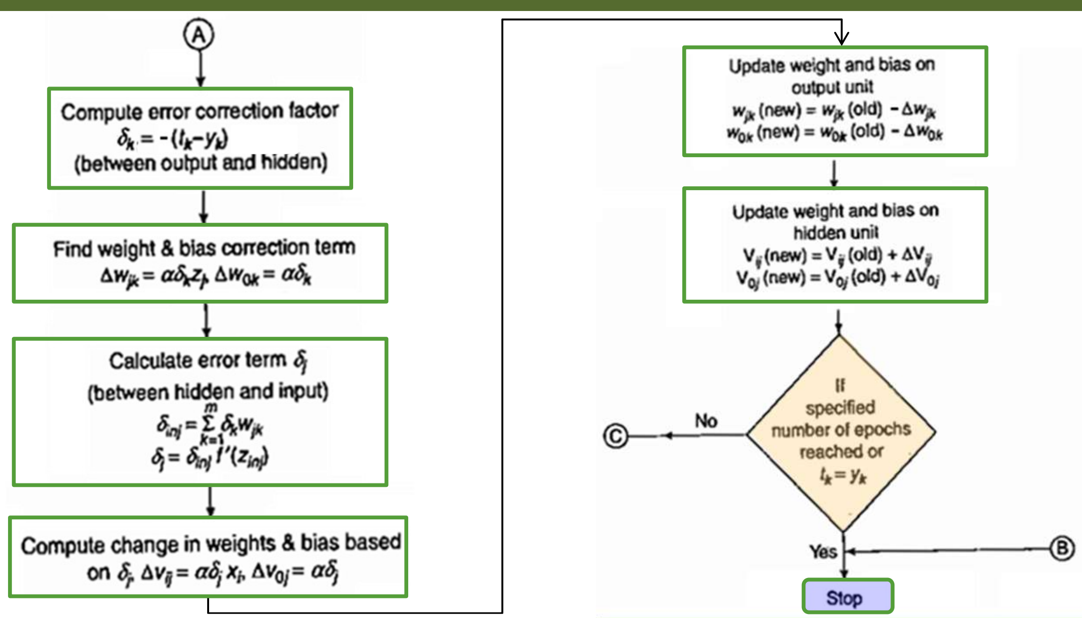
- $\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^T \boldsymbol{\delta}^{(4)} .* g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^T \boldsymbol{\delta}^{(3)} .* g'(\mathbf{z}^{(2)})$
- (No $\boldsymbol{\delta}^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)} .* (1 - \mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)} .* (1 - \mathbf{a}^{(2)})$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$ (ignoring λ; if $\lambda = 0$)

BPN Training



Training a Neural Network via Gradient Descent with Backprop

5

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
For each training instance $(\mathbf{x}^{(s)}, y^{(s)})$:
Set $\mathbf{a}^{(1)} = \mathbf{x}^{(s)}$
Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y^{(s)}$
Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

Backpropagation

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached

Optimizing the Neural Network

6

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1 - y_{ik})\log\Big(1 - (h_\Theta(\mathbf{x}_i))_k\Big)\right]$$
$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Solve via: $\min_{\Theta} J(\Theta)$

We can use Gradient Descent (GD)

Need code to compute:
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Training a Neural Network

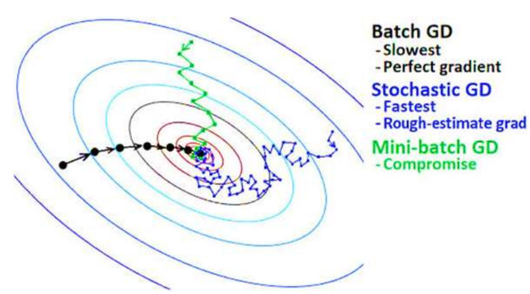
7

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(\mathbf{x}_i)$ for any instance $\mathbf{x}_i$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta^{(l)}_{jk}} J(\Theta)$ computed using backpropagation vs. the numerical gradient estimate.
   – Then, disable gradient checking code
6. Use gradient descent with backprop to fit the network

Momentum Method

8

- SGD is a popular optimization strategy but it can be slow
- Momentum method accelerates learning, when:
  – Facing high curvature
  – Small but consistent gradients
  – Noisy gradients

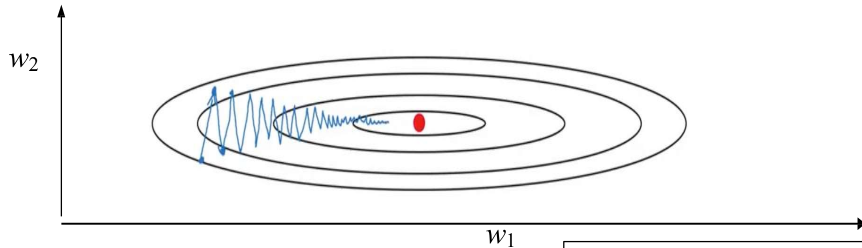


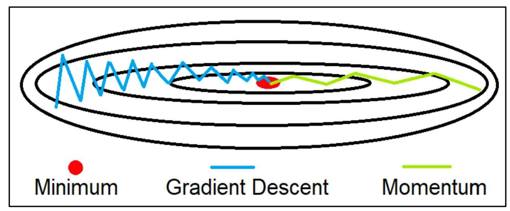
- It works by accumulating the moving average of past gradients and moves in that direction while exponentially decaying

Gradient Descent with Momentum

9

- Gradient descent with momentum converges faster than standard gradient descent
- Taking large steps in w_2 direction and small steps in w_1 direction slows down algorithm





- Momentum reduces oscillation in w_2 direction
- Now can set a higher learning rate

Momentum Definition

10

- Introduce velocity variable $\boldsymbol{v}$
- This is the direction and speed at which parameters move through parameter space
- Name momentum comes from physics and is mass times velocity
  - The momentum algorithm assumes unit mass
- A hyperparameter $\delta \in [0,1)$ determines exponential decay of $\boldsymbol{v}$

SGD Algorithm with Momentum

11

```
INPUT: cost function J(θ), learning rate α,  number of iterations: T,
 batch size B, initial velocity: v
INITIALIZE: random θ
FOR i = 1 to T DO
  Split the training examples into B mini-batches of size b:
  FOR j = 1 to number of mini-batches B DO
    Compute the gradient of J with respect to θ for a mini-batch
    of training examples:
    gradient = 1/b × ∇θ Σ(J(θ, x^j, y^j))
    Compute velocity update: v = δv − α × gradient
    Update the parameters θ:
    θ = θ + v
  END FOR
END FOR
OUTPUT: θ
```


Nesterov Momentum

12

- A variant to accelerate gradient, with update

$$v \leftarrow \delta v - \alpha \nabla_{\theta} \left[\frac{1}{n} \sum_{i=1}^{n} L\left(h_{\theta}(x^{(i)}; \theta + \delta v), y^{(i)} \right) \right],$$

$$\theta \leftarrow \theta + v,$$

- where parameters δ and α play a similar role as in the standard momentum method

– Difference between Nesterov and standard momentum is where gradient is evaluated.

- **Nesterov gradient is evaluated after the current velocity is applied.**

SGD with Nesterov Momentum

13

- A variant of the momentum algorithm
  - Nesterov's accelerated gradient method
- Applies a correction factor to standard method

INPUT: cost function J(θ), learning rate α, number of iterations: T, batch size B, initial velocity: v
INITIALIZE: random θ
FOR i = 1 to T DO
  Split the training examples into B mini-batches of size b:
  FOR j = 1 to number of mini-batches B DO
    Apply interim update: $\tilde{\theta} = \theta + \delta v$
    Compute the gradient of J with respect to θ for a mini-batch of training examples:
    gradient = 1/b × $\nabla\tilde{\theta}\ \Sigma(J(\tilde{\theta}, x^j, y^j))$
    Compute velocity update: $v = \delta v - \alpha \times gradient$
    Update the parameters θ:
    $\theta = \theta + v$
  END FOR
END FOR
OUTPUT: θ

Adam: Adaptive Moments

14

Adam Algorithm

Require: Step size α (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\alpha\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Choosing the Right Optimizer

15

- We have discussed several methods of optimizing deep models by adapting the learning rate for each model parameter
- Which algorithm to choose?
  - No consensus
- Most popular algorithms actively in use:
  - SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam
  - Choice depends on user's familiarity with algorithm

Neural Network: Advantages

16

- Handling complex relationships
- Feature extraction
- Scalability
- Processing unorganized data

Neural Network: Disadvantages

17

- Lack of transparency ("black box" nature)
- Computational expense
- Need for large datasets
- Overfitting
- Trial and error for architecture
- Data preparation

18

Questions?