DTE-2502: Neural Networks

Module02: Gradient descent

Basic definitions

Kalyan Ram Ayyalasomayajula, email: kay001@uit.no

1. Gradient descent

Basic Concept for single variable function:

- Function : $f(x)$ that we want to minimize
- Gradient : $\nabla f(x)$ = derivative/slope at point x
- Update Rule : $x_{new} = x_{old} - \alpha.\nabla f(x)$
- Learning Rate : α (step size parameter)

Extending to multiple variable function

- Function : $f(x_1, x_2, \cdots, x_n)$
- Gradient : $\nabla f = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}]$

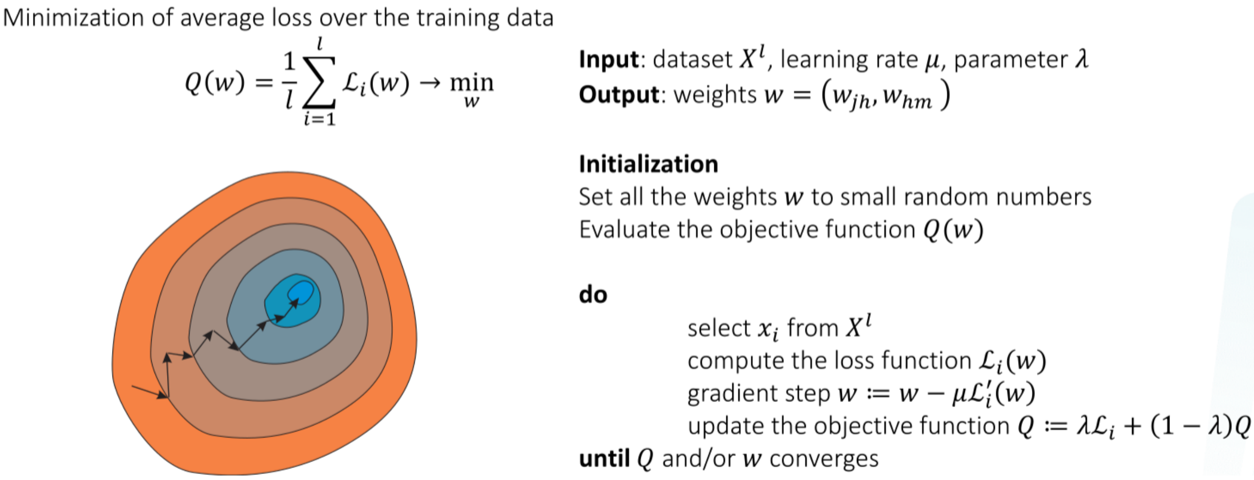
Algorithm 1 Gradient Descent Algorithm

```
Require: Training data, cost function Q(w)
Ensure: Optimized parameters (weights) w*
Initialize parameters w (randomly or zeros)
Set learning rate α
repeat
    Calculate gradient: ∇Q(w)
    Update parameters: w ← w − α.∇Q(w)
    Check convergence criteria
until convergence
return optimized parameters w
```

Traditional gradient descent (batch gradient descent) uses average gradient computed over the entire dataset for each update. This is a very simplified version (and not the entire picture) of what happens in gradient descent, but good enough to understand.

- Pros : Stable convergence, guaranteed to converge for convex functions
- Cons : Slow for large datasets, memory intensive

2. Stochastic gradient descent



3. Mini-batch gradient descent

Mini-batch gradient descent uses average of gradients computed over a mini-batch (small set) of dataset for each update.

- Pros : Balance between batch-GD and SGD
- Cons : Need to tune optimal batch size

4. Gradient Descent Optimizers

In all the below approaches $\nabla f(\theta)$ is the gradient w.r.t the parameter vector θ.

Definition 1: Momentum.
Key idea: As the descent direction is fixed add historical vector to accelerate convergence $v_t = v_{t-1} + \epsilon\nabla f(\theta_{t-1})$

Definition 2: Nesterov nomentum (NAG).
Key idea: In addition to the descent direction "Look ahead" by computing gradient at anticipated future position. $v_t = v_{t-1} + \epsilon\nabla f(\theta_{t-1} - \mu v_{t-1})$

Definition 3: AdaGrad.
Key idea: Adaptive learning rates - larger updates for infrequent parameters, smaller updates for frequent ones.

$$v_t = v_{t-1} + g^2, \text{where } g = \nabla f(\theta_{t-1})$$
$$\theta_{t+1} = \theta_t - \frac{\mu}{\sqrt{v_t} + \epsilon} g$$

- Pros
  - No manual learning rate tuning per parameter

- Great for sparse data
- Larger updates for rare features

- Cons
  - Learning rate decreases too aggressively, may stop learning

DEFINITION 4: RMSprop.
Key Idea : Fix AdaGrad's vanishing learning rate using exponential moving average.

$$v_t = \beta v_{t-1} + (1-\beta)g^2, \beta \approx 1$$
$$\theta_{t+1} = \theta_t - \frac{\mu}{\sqrt{v_t} + \epsilon} g$$

- Pros
  - Maintains adaptive learning rates without vanishing
  - Works well for non-stationary objectives
- Cons
  - Sensitive to choice of μ

DEFINITION 5: Adam (Adaptive Moment Estimation).
Key Idea : Combine momentum + RMSprop with bias correction.

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)g; \hat{v}_t = \frac{v_t}{1-\beta_1^t}$$
$$s_t = \beta_2 s_{t-1} + (1-\beta_2)g^2; \hat{s}_t = \frac{s_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\mu \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon} g$$

5. Practical issues in neural networks training

- Overfitting
  - Regularization
  - Dropout
- Vanishing or exploding gradients
  - Adaptive learning rate
  - Batch normalization
- Local optima
  - Pre-training