Chapter 12: Variations on Backpropagation

Brandon Morgan

1/22/2021

E12.7

We are given the following quadratic function from E12.3

$$F(X) = \frac{1}{2}X^T \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} X + \begin{bmatrix} 4 & 4 \end{bmatrix} X$$

With gradient:

$$\nabla F(X) = AX + D = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} X + \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

We want to perform three iterations of variable learning rate with initial guess $x_0^T = [-1, -2.5]$, $\alpha = 0.4$, $\gamma = 0.1$, $\eta = 1.5$, $\rho = 0.5$, and $\xi = 5\%$.

Iteration 1

Our first step is to evaluate the function at the initial guess:

$$F(x_0) = F(\begin{bmatrix} -1 \\ -2.5 \end{bmatrix}) = \frac{1}{2} \begin{bmatrix} -1 \\ -2.5 \end{bmatrix}^T \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \begin{bmatrix} -1 \\ -2.5 \end{bmatrix} + \begin{bmatrix} 4 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ -2.5 \end{bmatrix} = 7.25$$

Next, we evaluate our gradient at the initial point:

$$g_0 = \nabla F(\begin{bmatrix} -1 \\ -2.5 \end{bmatrix}) = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \begin{bmatrix} -1 \\ -2.5 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 9 \\ -15 \end{bmatrix}$$

With a learning rate of $\alpha = 0.4$,

$$\Delta x_0 = \gamma \Delta x_{-1} - (1-\gamma)\alpha g_0 = 0.1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} - (1-0.1)(0.4) \begin{bmatrix} 9 \\ -15 \end{bmatrix} = \begin{bmatrix} -3.24 \\ 5.4 \end{bmatrix}$$

thus,

$$x_1^T = x_0 + \Delta x_0 = \begin{bmatrix} -1 \\ -2.5 \end{bmatrix} + \begin{bmatrix} -3.24 \\ 5.4 \end{bmatrix} = \begin{bmatrix} -4.24 \\ 2.9 \end{bmatrix}$$

Now, we have to verify that this is a valid step by seeing if the function of this new point is less than the previous:

$$F(\begin{bmatrix} -4.24 \\ 2.9 \end{bmatrix}) = 200.354$$

Because this value is more than $\xi = 5\%$ larger than the value at the previous point, we reject this step, reduce the learning rate and set the momentum coefficient to zero:

$$x_2 = x_1, F(x_2) = F(x_1) = 7.25, \alpha = \rho\alpha = 0.4(0.5) = 0.2, \gamma = 0$$

This completes the first iteration. Instead of performing the rest of the iterations by hand, we will use an algorithm:

```
# learning_rate = alpha
# momenum = gamma
# percentage = xi
# factor1 = eta
# factor2 = rho

variable_learning_rate = function(fun, gradient, init, learning_rate, momentum,
                                  percentage, factor1, factor2, maxIter, tol, mask) {

  x = init
  deltaX0 = matrix(0, ncol = 1, nrow=nrow(init))

  momentumOriginal = momentum

  for(i in 1:maxIter) {

    fx0 = fun(x)
    g = gradient(x)

    deltaX = momentum*deltaX0-(1-momentum)*learning_rate*g

    x1 = x + deltaX

    fx1 = fun(x1)

    if(!mask) {
      print(sprintf("    Iteration %d", i))
      print("F(x):")
      print(fx0)
      print("Gradient: ")
      print(g)
      print("Delta x")
      print(deltaX)
      print("New x1")
      print(x1)
      print("F(x1):")
      print(fx1)
    }

    if(norm(g, "F")<tol) {
      print("    ALGORITHM CONVERGED ")
      print(sprintf("Iterations taken %d", i))
```

```
    print("Stationary point found at")
    print(x)
    return(x)
  }


  if(fx0 > fx1) {
    if(!mask) {
        print("  Weight change accepted: fx0 > fx1")
    }


    learning_rate = learning_rate*factor1
    if(momentum == 0) {
      if(!mask) {
        print("Momentum set back to original value")
      }
      momentum = momentumOriginal
    }

    if(!mask) {
        print(sprintf("New Learning Rate: %f", learning_rate))
    }


    deltaX0 = deltaX
    x = x1
  }
  else if(((percentage/100)*fx0+fx0)>fx1) {

    if(!mask) {
       print("  Weight change accepted: (fx0*perent+fx0) > fx1 ")
    }

    if(momentum == 0) {
      if(!mask) {
        print("Momentum set back to original value")
      }
      momentum = momentumOriginal
    }
    deltaX0 = deltaX
    x = x1
  }
  else { # then (((percentage/100)*fx0)+fx0)<fx1

    learning_rate = learning_rate * factor2
    momentum = 0

    if(!mask) {
       print("  Weight change Rejected: Larger than percent of fx0")
       print("Momentum set to zero")
       print(sprintf("New Learning Rate: %f", learning_rate))
    }
```

```
    }
  }

  print("    MAXIMUM ITERATION REACHED")
  print("Current position: ")
  print(x)
  print("ERROR: ")
  print(norm(g, "F"))

}
```

```
fun = function(X) {
  A = matrix(c(10, -6, -6, 10), ncol=2)
  d = matrix(c(4, 4), nrow=1)
  0.5*t(X)%*%A%*%X+d%*%X
}
gradient = function(X) {
  A = matrix(c(10, -6, -6, 10), ncol=2)
  d = matrix(c(4, 4), nrow=1)
  A%*%X+t(d)
}
```

```
init = matrix(c(-1, -2.5), ncol=1)
min = variable_learning_rate(fun, gradient, init, 0.4, 0.1, 5, 1.5, 0.5, 3, 1e-10, 0)
```

```
## [1] "    Iteration 1"
## [1] "F(x):"
##      [,1]
## [1,] 7.25
## [1] "Gradient: "
##      [,1]
## [1,]    9
## [2,]  -15
## [1] "Delta x"
##       [,1]
## [1,] -3.24
## [2,]  5.40
## [1] "New x1"
##       [,1]
## [1,] -4.24
## [2,]  2.90
## [1] "F(x1):"
##         [,1]
## [1,] 200.354
## [1] "  Weight change Rejected: Larger than percent of fx0"
## [1] "Momentum set to zero"
## [1] "New Learning Rate: 0.200000"
## [1] "    Iteration 2"
## [1] "F(x):"
##      [,1]
## [1,] 7.25
## [1] "Gradient: "
##      [,1]
```

```
## [1,]    9
## [2,]  -15
## [1] "Delta x"
##      [,1]
## [1,] -1.8
## [2,]  3.0
## [1] "New x1"
##      [,1]
## [1,] -2.8
## [2,]  0.5
## [1] "F(x1):"
##       [,1]
## [1,] 39.65
## [1] "  Weight change Rejected: Larger than percent of fx0"
## [1] "Momentum set to zero"
## [1] "New Learning Rate: 0.100000"
## [1] "    Iteration 3"
## [1] "F(x):"
##      [,1]
## [1,] 7.25
## [1] "Gradient: "
##      [,1]
## [1,]    9
## [2,]  -15
## [1] "Delta x"
##      [,1]
## [1,] -0.9
## [2,]  1.5
## [1] "New x1"
##      [,1]
## [1,] -1.9
## [2,] -1.0
## [1] "F(x1):"
##      [,1]
## [1,] 0.05
## [1] "  Weight change accepted: fx0 > fx1"
## [1] "Momentum set back to original value"
## [1] "New Learning Rate: 0.150000"
## [1] "    MAXIMUM ITERATION REACHED"
## [1] "Current position: "
##      [,1]
## [1,] -1.9
## [2,] -1.0
## [1] "ERROR: "
## [1] 17.49286
```

Here we have our three iterations of variable learning rate. If we run our algorithm until converge, it will take 42 iterations with tolerane $1e-6$:

```r
init = matrix(c(-1, -2.5), ncol=1)
min = variable_learning_rate(fun, gradient, init, 0.4, 0.1, 5, 1.5, 0.5, 100, 1e-6, 1)
```

```
## [1] "    ALGORITHM CONVERGED "
```

```
## [1] "Iterations taken 42"
## [1] "Stationary point found at"
##      [,1]
## [1,]   -1
## [2,]   -1
```