

# Advanced Machine Learning Subsidiary Notes

## Lecture 14: Stochastic Gradient Descent

Adam Prügel-Bennett

February 9, 2024

## 1 Keywords

- SGD, momentum, step size, ADAM

## 2 Main Points

### 2.1 Stochastic Gradient Descent

- One can estimate the gradient from a mini-batch  $\mathcal{B} \subset \mathcal{D}$

$$\nabla L_{\mathcal{B}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x}, y) \in \mathcal{B}} L(\mathbf{x}, y | \mathbf{w})$$

where  $L(\mathbf{x}, y | \mathbf{w})$  is the loss for example  $(\mathbf{x}, y)$  with weights  $\mathbf{w}$

- If  $|\mathcal{B}| \ll |\mathcal{D}|$  this is massively faster than computing the full gradient
- This allows us to make relatively small steps very quickly
- By making lots of steps we average out the random errors
- **Comparison to 2nd order methods**
  - Newton and Quasi-Newton methods converge faster
  - But you only care when you are close to a minimum
  - Away from a minimum 2nd order methods can lead you astray
  - When using ReLUs the Loss landscape does not have a continuous first derivative so second order methods might not work
  - We want to minimise the generalisation error so reaching the minimum of the training error is perhaps not that important
  - In high dimensions 2nd order methods are impractical
- Automatic differentiation allow us to compute gradients for complicated loss functions for free (this is often a game changer)
- However, you still need to decide on the step size and you can diverge

### 2.2 Momentum

- By using "momentum" we remember our earlier movements
  - Allows us to take large steps in directions with small curvature
  - Cancels zig-zagging in directions with high curvature

- Introduce a "velocity"

$$\begin{aligned} \mathbf{v}^{(t+1)} &= (1 - \gamma) \mathbf{v}^{(t)} - \gamma r \nabla L_{\mathcal{B}}(\mathbf{w}^{(t)}) \\ \mathbf{w}^{(t+1)} &= \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)} \end{aligned}$$

- $\gamma$  might be small 0.1
- $r$  is the usual step size

## 2.3 Adaptive Methods

- The difficulty of high dimensional optimisation is there are different curvatures
  - Where there is high curvature we want to make small steps
  - Where there is low curvature we want to make large steps
- In adaptive methods we change our step size for each variables
- We could measure the curvature in different directions

$$\frac{\partial^2 L(\mathbf{w})}{\partial w_i^2}$$

but most adaptive algorithms don't do this

- **AdaDelta**

- AdaDelta is an algorithm that estimates the curvature by computing a running mean squared gradient

$$S_i^{g(t+1)} = (1 - \gamma) S_i^{g(t)} + \gamma \left( \frac{\partial L_{\mathcal{B}}(\mathbf{w}^{(t)})}{w_i^{(t)}} \right)^2$$

\* This is a running average (it slowly forgets the past)

- We also compute a running average of the squared weight

$$S_i^{w(t+1)} = (1 - \gamma) S_i^{w(t)} + \gamma (w_i^{(t)})^2$$

- It then updates each weight according to

$$w_i^{(t+1)} = w_i^{(t)} - \eta \sqrt{\frac{S_i^w(t+1) + \epsilon}{S_i^g(t+1) + \epsilon}} \frac{\partial L_{\mathcal{B}}(\mathbf{w}^{(t)})}{\partial w_i^{(t)}}$$

- This ensures invariance in two ways
  - \* If we multiply our weights by a factor we get the same relative change
  - \* If we multiply our gradients by a factor we get the same change

- **ADAM**

- AdaDelta doesn't use momentum
- Adaptive Moment Estimation (ADAM) does both adaptive step-size per feature and it uses momentum
- It computes a running average momentum and squared gradient

$$M_i^{(t+1)} = (1 - \beta) M_i^{(t)} + \beta \frac{\partial L_{\mathcal{B}}(\mathbf{w}^{(t)})}{\partial w_i^{(t)}}$$

$$S_i^{(t+1)} = (1 - \gamma) S_i^{(t)} + \gamma \left( \frac{\partial L_{\mathcal{B}}(\mathbf{w}^{(t)})}{\partial w_i^{(t)}} \right)^2$$

- Running averages suffer from time-lag (it takes time for them to build-up)
- In ADAM we remove the time lag

$$\hat{M}_i^{(t+1)} = \frac{M_i^{(t+1)}}{1 - (1 - \beta)^t} \qquad \hat{S}_i^{(t+1)} = \frac{S^{(t+1)}}{1 - (1 - \gamma)^t}$$

- We then update the weights

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{\hat{S}_i^{(t+1)} + \epsilon}} \hat{M}_i^{(t+1)}$$

- ADAM and its variants are very successful: often giving state-of-the-art performance

- **Covariance**

- The adaptive schemes works independently on each coordinate
- Covariance properties of vectors
  - \* If we act on vectors using standard operations
    - scalar multiplication
    - addition
    - matrix multiplication
 then the results are invariant of the coordinate system we use
  - \* In particular they will be translation and rotation invariant
  - \* When we do elementwise multiplication this invariance is lost
  - \* More generally this is true for tensors
  - \* In machine learning although we call multi-dimensional arrays tensors we usually apply elementwise operations rather than proper tensor operations (we loose invariance to coordinate transformations)
- Because the adaptive schemes are elementwise they are not invariant to rotation
- If  $e_i$  is the direction of increasing weight  $w_i$  the if two weights interact we could have high curvature in a direction  $e_i + e_j$  and low curvature in a direction  $e_i - e_j$ . We cannot adapt the weights individually to equalise the curvature.

## 2.4 Loss Landscapes

- In modern machine learning we are often perform minimisation of the loss function in a massive search space
- Unless the search space has a simple structure (e.g. is convex) there are likely to be many local optima
- There is no algorithm that is guaranteed to find the global minimum
- In such large spaces we might never get near to a minimum
- **Symmetries**
  - The loss landscape will typically have many symmetries
  - If we permute the nodes of an MLP or feature maps of a CNN we get the same solution
  - There may also be continuous symmetries
  - Most directions might not change the loss at all
  - Symmetries complicated the loss landscape
    - \* If you have two local minima there will be a saddle-point in between
  - Adding skip connections removes permutation symmetries which seems to make optimisation simpler

## 3 Exercises

### 3.1 Removing Lag

- Consider a running average

$$a^{(t+1)} = (1 - \gamma) a^{(t)} + \gamma x^{(t)}$$

- Assume  $x^{(t)} = x$  (i.e. constant)

- Calculate  $a^{(t)}$  if  $a^{(0)} = 0$  as a sum
- Using the fact that the sum of a geometric series can be written as

$$\sum_{i=0}^{t-1} z^i = 1 + z + \dots + z^{t-1} = \frac{1 - z^t}{1 - z}$$

write  $a^{(t)}$  in closed form

- Compute the correction to the running mean so that the corrected running mean equals  $x$  for all  $t$

### 3.2 Gradient Descent in a Quadratic Minimum

- Consider minimising a quadratic function

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^\top \mathbf{Q}(\mathbf{x} - \mathbf{x}^*)$$

- Using gradient descent

$$\mathbf{x}(t+1) = \mathbf{x}(t) - r \nabla f(\mathbf{x})$$

- Using the definition of  $f(\mathbf{x})$  write down the update equation
- Subtract  $\mathbf{x}^*$  from both sides of the equation and write down an update equation for  $\mathbf{x}(t) - \mathbf{x}^*$
- Write a closed form solution for  $\mathbf{x}(t) - \mathbf{x}^*$
- Using the eigen-decomposition  $\mathbf{Q} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top$  rewrite the closed form solution
- Defining  $\mathbf{u}(t) = \mathbf{V}^\top(\mathbf{x}(t) - \mathbf{x}^*)$  write the update equation for the components  $u_i(t)$  and explain what happens when  $\lambda_i > 2/r$

## 4 Experiments

### 4.1 Gradient Descent

- Write a Matlab/Octave or python programme
- Compute a random  $5 \times 4$  matrix  $\mathbf{X}$
- Let  $\mathbf{M} = \mathbf{X}^\top \mathbf{X}$
- Consider minimising  $f(\mathbf{w}) = \frac{1}{2} \mathbf{w}^\top \mathbf{M} \mathbf{w}$

- Find the Hessian of  $f(\mathbf{w})$
- Compute the eigenvalues of the Hessian
- Compute the gradient of  $f(\mathbf{x})$
- From a random starting point  $\mathbf{x}^{(0)}$  follow the negative gradient

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - r \nabla f(\mathbf{x}^{(t)})$$

5. For what value of  $r$  do you converge?
6. Repeat this using momentum

$$\begin{aligned} \mathbf{v}^{(t+1)} &= (1 - \gamma) \mathbf{v}^{(t)} - \gamma r \nabla f(\mathbf{x}^{(t)}) \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} + \mathbf{v}^{(t+1)} \end{aligned}$$

Using  $\gamma = 0.1$  and  $r = 1$

```
X = rand(5,4)
M = X'*X           % This is the Hessian
eig(M)             % Eigenvalues of momentum

w = rand(4,1)      % x0
r = 0.01
for t = 1:10
    f = w'*M*w/2    % current function value
    w = w - r*M*w;  % gradient is M*w
endfor             % I use octave

%%% Experiment with different values of r
for r = 0.05:0.05:0.5
    w = rand(4,1);
    for t = 1:100
        w = w - r*M*w;
    endfor
    [r, w'*M*w/2]    % function value after 100 iterations
endfor

%%% Using Momentum
w = rand(4,1);
v = zeros(4,1);
f = []
gamma = 0.1
for t = 1:100
    v = (1-gamma)*v - gamma*M*w;
    w = w + v;
    f(end+1) = w'*M*w/2;
endfor
plot(1:100,f)
```

## 5 Solutions

### 5.1 Removing Lag

1. Writing  $a^{(t)}$  as a sum

$$\begin{aligned} a^{(1)} &= (1 - \gamma) a^{(0)} + \gamma x = \gamma x \\ a^{(2)} &= (1 - \gamma) a^{(1)} + \gamma x = (1 - \gamma) \gamma x + \gamma x \\ a^{(3)} &= (1 - \gamma) a^{(2)} + \gamma x = (1 - \gamma)^2 \gamma x + (1 - \gamma) \gamma x + \gamma x \\ a^{(t)} &= \gamma x \sum_{i=0}^{t-1} (1 - \gamma)^i \end{aligned}$$

2. • Geometric series

– As an aside we can prove the identity just multiply the geometric series by  $1 - z$

$$(1 - z)(1 + z + \dots + z^{t-1}) = (1 + z + \dots + z^{t-1}) - (z + z^2 + \dots + z^t) = 1 - z^t$$

– Dividing both sides by  $(1 - z)$  we obtain our identity

• Applying the identity to  $a^{(t)}$  we find

$$a^{(t)} = \gamma x \frac{1 - (1 - \gamma)^t}{1 - (1 - \gamma)} = x (1 - (1 - \gamma)^t)$$

Note that as  $t \rightarrow \infty$  then  $a^{(t)}$  approaches  $x$

3. Dividing through by  $1 - (1 - \gamma)^t$  i.e.

$$\bar{a}^{(t)} = \frac{a^{(t)}}{1 - (1 - \gamma)^t}$$

we lose the lag

## 5.2 Gradient Descent in a Quadratic Minimum

1.  $\mathbf{x}(t + 1) = \mathbf{x}(t) - r \mathbf{Q}(\mathbf{x}(t) - \mathbf{x}^*)$

2.  $\mathbf{x}(t + 1) - \mathbf{x}^* = (\mathbf{I} - r \mathbf{Q})(\mathbf{x}(t) - \mathbf{x}^*)$

3.  $\mathbf{x}(t) - \mathbf{x}^* = (\mathbf{I} - r \mathbf{Q})^t(\mathbf{x}(0) - \mathbf{x}^*)$

4.  $\mathbf{x}(t) - \mathbf{x}^* = \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda})^t \mathbf{V}^T(\mathbf{x}(0) - \mathbf{x}^*)$  see note below

5.  $\mathbf{u}(t) = (\mathbf{I} - r \mathbf{\Lambda})^t \mathbf{u}(0)$  or  $u_i(t) = (1 - r \lambda_i)^t u_i(0)$ . If  $\lambda_i > 2/r$  then  $u_i(t)$  diverges exponentially fast. That is any component in the direction of  $\mathbf{v}_i$  away from the optimum will diverge if the curvature (i.e. second derivative) in that direction exceeds  $2/r$ , where  $r$  is the step size.

Note that  $\mathbf{I} - r \mathbf{Q} = \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda}) \mathbf{V}^T$  so

$$(\mathbf{I} - r \mathbf{Q})^2 = \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda}) \mathbf{V}^T \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda}) \mathbf{V}^T$$

but  $\mathbf{V}^T \mathbf{V} = \mathbf{I}$  as they are orthogonal matrix thus

$$(\mathbf{I} - r \mathbf{Q})^2 = \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda})^2 \mathbf{V}^T$$

and similarly

$$(\mathbf{I} - r \mathbf{Q})^t = \mathbf{V}(\mathbf{I} - r \mathbf{\Lambda})^t \mathbf{V}^T.$$