Minimise your Loss

VLC = ∭ Vision Learning Control dVdLdC

Optimisation

Kate Farrahi (adapted from Jon Hare's lecture)

Vision, Learning and Control
University of Southampton

Reminder: Gradient Descent

- Define total loss as $\mathcal{L} = \sum_{(\boldsymbol{x},y)\in\boldsymbol{D}} \ell(g(\boldsymbol{x},\boldsymbol{\theta}), y)$ for some loss function ℓ, dataset $\boldsymbol{D}$ and model g with learnable parameters $\boldsymbol{\theta}$.
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Gradient Descent updates the parameters $\boldsymbol{\theta}$ by moving them in the direction of the negative gradient with respect to the **total loss** $\mathcal{L}$ by the learning rate η multiplied by the gradient:

```
for each Epoch:
```

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

Gradient Descent

- Fewer updates to the model means more computationally efficient.
- The decreased update frequency results in more stable error gradient and more stable convergence.
- The updates at the end of the training epoch require the additional complexity of accumulating prediction errors across all training examples. Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
- Model updates, and in turn training speed, may become very slow for large datasets.

Reminder: Stochastic Gradient Descent

- Define loss function ℓ, dataset $\boldsymbol{D}$ and model g with learnable parameters $\boldsymbol{\theta}$.
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Stochastic Gradient Descent updates the parameters $\boldsymbol{\theta}$ by moving them in the direction of the negative gradient with respect to the loss of a **single item** ℓ by the learning rate η multiplied by the gradient:

`for each Epoch:`
    `for each` $(\boldsymbol{x}, y) \in \boldsymbol{D}$:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ell$

Stochastic Gradient Descent

- The frequent updates immediately give an insight into the performance of the model.
- The noisy update process can allow the model to avoid local minima.
- The frequent updates can result in a noisy gradient signal, which may cause the model parameters and in turn the model error to jump around. This makes it hard for the algorithm to settle on an error minimum.
- Computationally inefficient (poor utilisation of resources - particularly with respect to vectorisation)

Mini-batch Stochastic Gradient Descent

- Define a batch size b
- Define batch loss as $\mathcal{L}_b = \sum_{(\boldsymbol{x},y)\in\boldsymbol{D}_b} \ell(g(\boldsymbol{x},\boldsymbol{\theta}), y)$ for some loss function ℓ and model g with learnable parameters $\boldsymbol{\theta}$. $\boldsymbol{D}_b$ is a subset of dataset $\boldsymbol{D}$ of cardinality b.
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Mini-batch Gradient Descent updates the parameters $\boldsymbol{\theta}$ by moving them in the direction of the negative gradient with respect to the loss of a **mini-batch** $\boldsymbol{D}_b$, $\mathcal{L}_b$ by the learning rate η multiplied by the gradient:

partition the dataset $\boldsymbol{D}$ into an array of subsets of size b

`for each Epoch:`

`for each` $\boldsymbol{D}_b \in \textit{partitioned}(\boldsymbol{D})$`:`

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_b$$

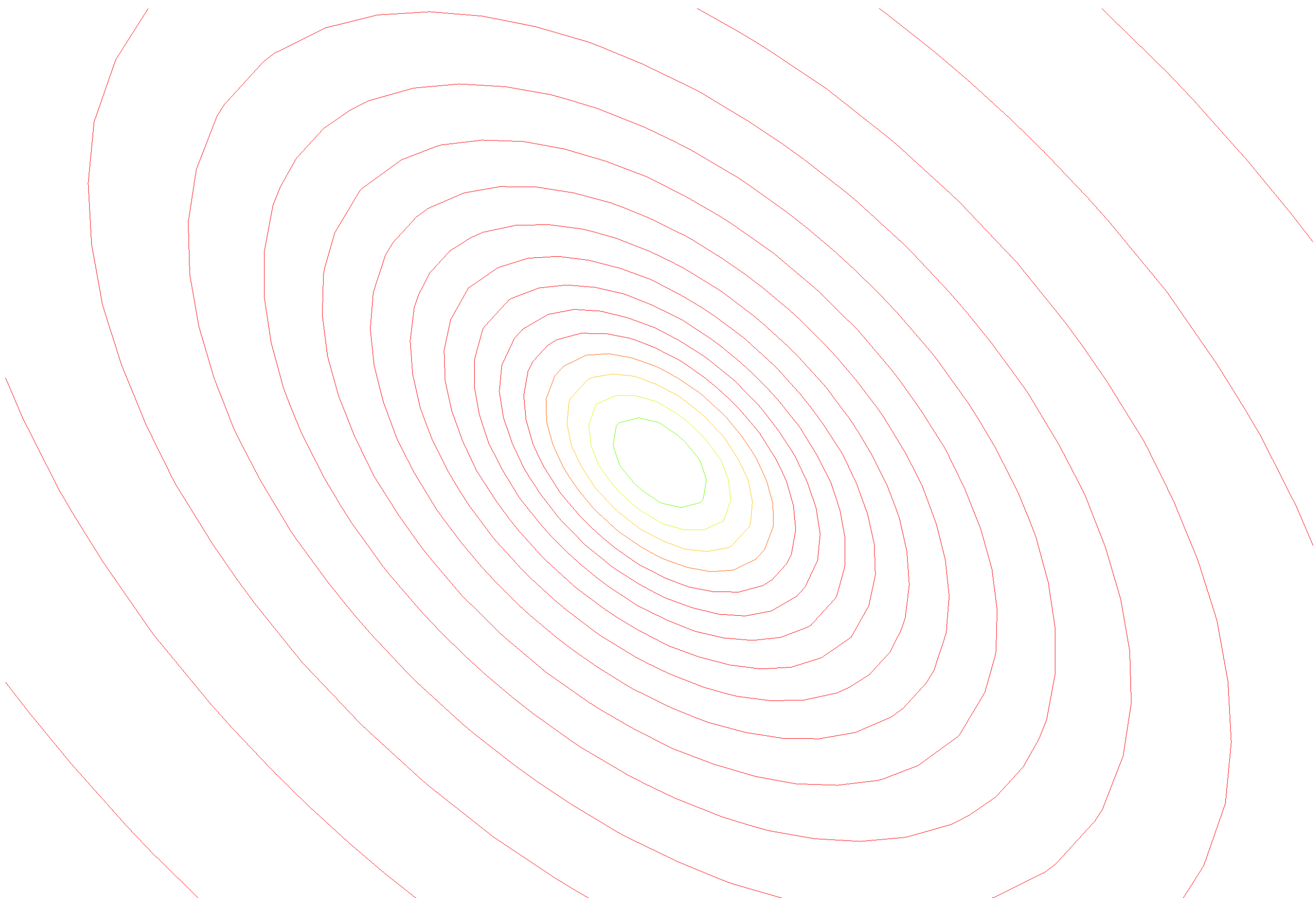
Mini-batch Stochastic Gradient Descent

- Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.
- The model update frequency is higher than gradient descent, which allows for a more robust convergence, avoiding local minima.
- Allows for computationally efficiency (good utilisation of resources)
- The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

So, what about the learning rate?

- Choice of learning rate is extremely important
- But we have to reason about the 'loss landscape'
    - Most convergence analysis of optimisation algorithms assumes a convex loss landscape
        - Easy to reason about
        - Can be shown that (S)GD will converge to the optimal solution for a variety of learning rates
        - Can give insights into potential problems in the non-convex case
    - Deep Learning is highly non-convex
        - Many local minima
        - Plateaus
        - Saddle points
        - Symmetries (permutation, etc)
        - Certainly no single global minima

*GD in the convex case: failure modes



Accelerated Gradient Methods

- Accelerated gradient methods use an exponentially weighted average of the gradient, rather than the instantaneous gradient estimate at each step.
- A physical analogy would be one of the momentum a ball picks up rolling down a hill...
- As you'll see, this helps address the *GD failure modes, but also helps avoid getting stuck in local minima

Exponentially Weighted Averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$

v_t is approximately average over $\approx \frac{1}{1-\beta}$ days

For example
$v_{100} = 0.9 v_{99} + 0.1\theta_{100}$
$v_{99} = 0.9 v_{98} + 0.1\theta_{99}$
$v_{98} = 0.9 v_{97} + 0.1\theta_{98}$
...

$v_{100} = 0.1\theta_{100} + 0.9[0.1\theta_{99} + 0.9[\dots]]]$

$v_{100} = 0.1\theta_{100} + 0.9 * 0.1 * \theta_{99} + 0.1 * (0.9)^2 * \theta_{98} + 0.1(0.9)^3\theta_{97} + \dots$

Momentum I

It's common for the weighted (or 'leaky') term (the 'velocity', v_t) to be a weighted average of the instantaneous gradient g_t and the past velocity[1]:

$$v_t = \beta v_{t-1} + g_t$$

where $\beta \in [0, 1]$ is the 'momentum'.

[1] There are quite a few variants of this; here we're following the PyTorch variant

Momentum II

- The momentum method allows to accumulate velocity in directions of low curvature that persist across multiple iterations
- This leads to accelerated progress in low curvature directions compared to gradient descent

MB-SGD with Momentum

Learning with momentum on iteration t (batch at t denoted by $b(t)$) is given by:

$$\begin{aligned}\boldsymbol{v}_t &\leftarrow \beta \boldsymbol{v}_{t-1} + \nabla_{\boldsymbol{\theta}} \mathcal{L}_{b(t)} \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \eta \boldsymbol{v}_t\end{aligned}$$

Note $\beta = 0.9$ is a good choice for the momentum parameter.

Learning rate schedules

- In practice you want to decay your learning rate over time
- Smaller steps will help you get closer to the minima
- But don't do it to early, else you might get stuck
- Something of an art form!
  - 'Grad Student Descent' or GDGS ('Gradient Descent by Grad Student')

Reduce LR on plateau

- Common Heuristic approach:
  - if the loss hasn't improved (within some tolerance) for k epochs
  - then drop the lr by a factor of 10
- Remarkably powerful!

Cyclic learning rates

- Worried about getting stuck in a non-optimal local minima?
- Cycle the learning rate up and down (possibly annealed), with a different lr on each batch
- See `https://arxiv.org/abs/1506.01186`

More advanced optimisers

- Adagrad
  - Decrease learning rate dynamically per weight.
  - Squared magnitude of the gradient used to adjust how quickly progress is made - weights with large gradients are compensated with a smaller learning rate.
  - Used for less complex models, too slow for deep learning.
- RMSProp
  - Modifies Adagrad to decouple learning rate from gradient magnitude scaling
  - Incorporates leaky averaging of squared gradient magnitudes
  - LR would typically follow a predefined schedule
- Adam
  - Essentially takes all the best ideas from RMSProp and SDG+Momentum
  - Shown that it might still diverge (or be non optimal, even in convex settings)...

Take-away messages

- The loss landscape of a deep network is complex to understand (and is far from convex)
- If you're in a hurry to get results use Adam
- If you have time (or a Grad Student at hand), then use SGD (with momentum) and work on tuning the learning rate
- If you're implementing something from a paper, then follow what they did!