Efficient Deep Learning Optimization Methods

Recitation 3

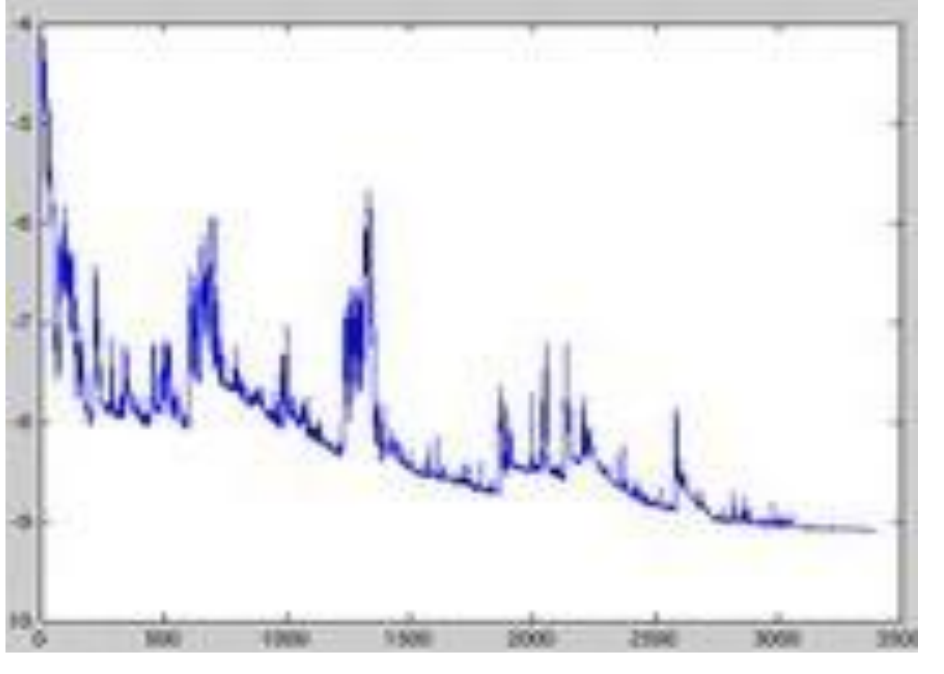
Aishwarya Reganti, Hanna Moazam

Outline

- 1 Review of optimization
- 2 Optimization practice
- 3 Training tips in PyTorch

1.1 Mini-batch gradient descent

- What is it?
  - Performs update for every mini-batch of data.
- Why mini-batch?
  - Batch gradient descent that uses the whole dataset for one update: slow and intractable for large datasets to fit into memory.
  - Stochastic gradient descent that updates for each data: high variance updates.



SGD fluctuation (Source: wiki)

1.1 Mini-batch gradient descent (continue)

- Update equation
  - Let F be our model, and θ is the parameter: $\hat{y} = F(x;\theta)$
  - The loss function is L, minimize the loss on the dataset:

$$g = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

  - Let η be the learning rate, compute the update:

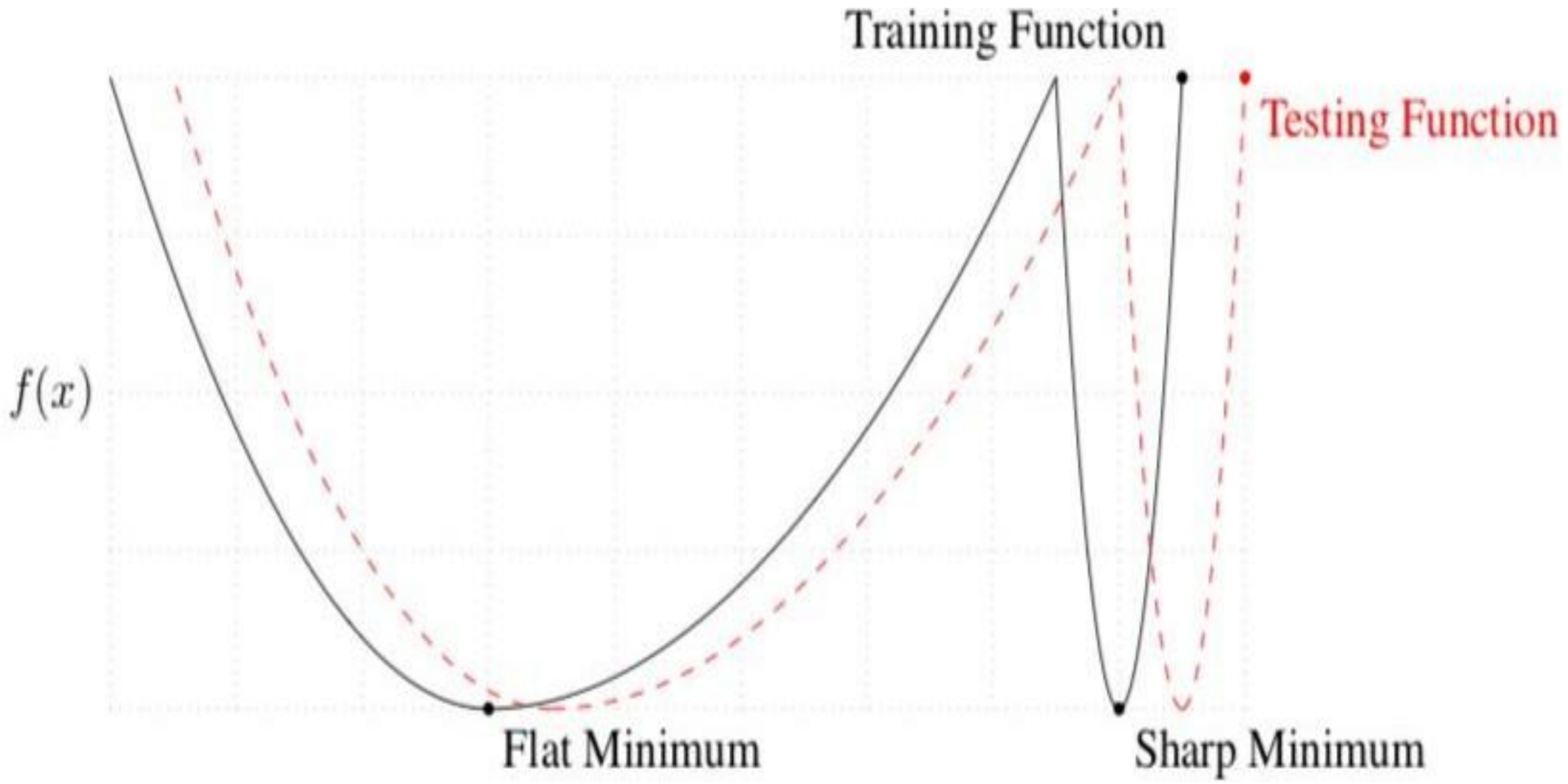
$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(y_i, \hat{y}_i), \quad \theta = \theta - \eta \cdot \hat{g}$$

1.1 Mini-batch gradient descent (Continue)

- The good things of mini-batch gradient descent
  - Reduces variance of updates
  - Matrix multiplication is faster
- Have to decide mini-batch size now!
- The common mini-batch size are 32-256.
- Too small: Slow and high variance,
- Too big: Harder to escape from local minima. Decay in generalization (paper link).

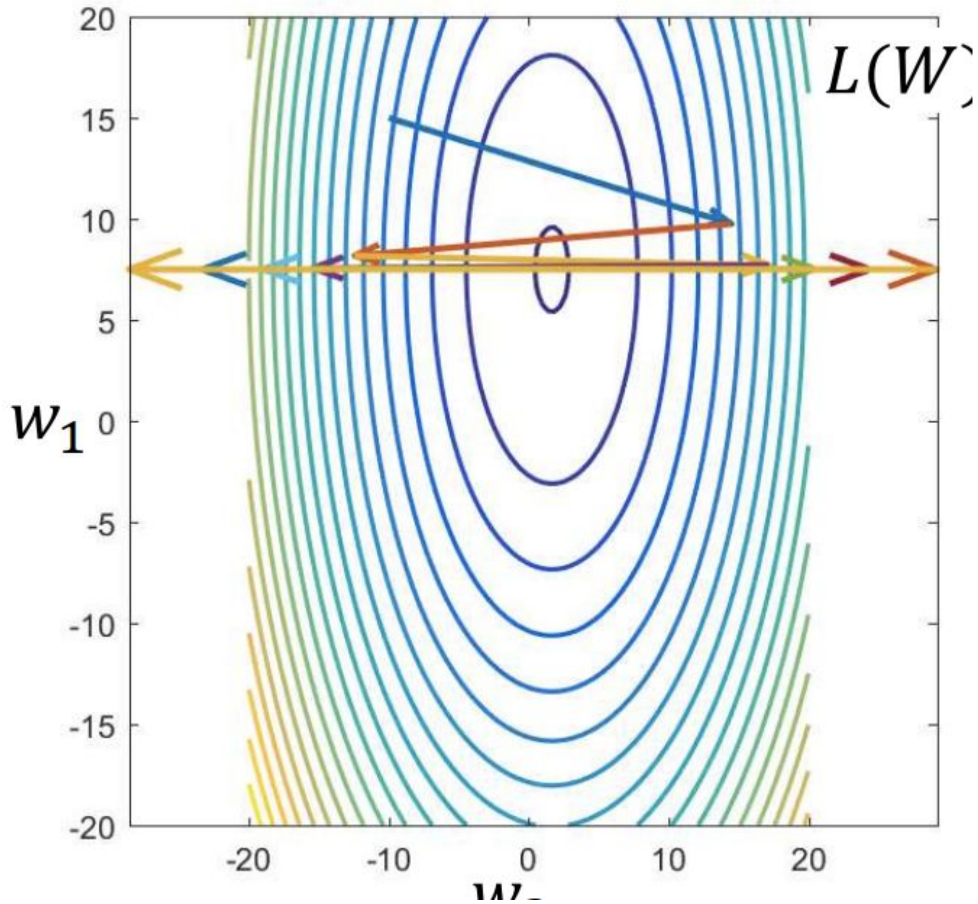
1.1 Mini-batch gradient descent (Continue)

The figure shows why big batch size is not OK:



Y-axis: value of the loss.   X-axis :the parameters.

Issues with Gradient Descent



- The loss is a function of many weights (and biases) – Has different eccentricities w.r.t different weights
- A fixed step size for all weights in the network can result in the convergence of one weight, while causing a divergence of another

Solutions:

Try to normalize curvature in all directions

- Second order methods, e.g. Newton's method
- Too expensive: require inversion of a giant Hessian

Treat each dimension independently:

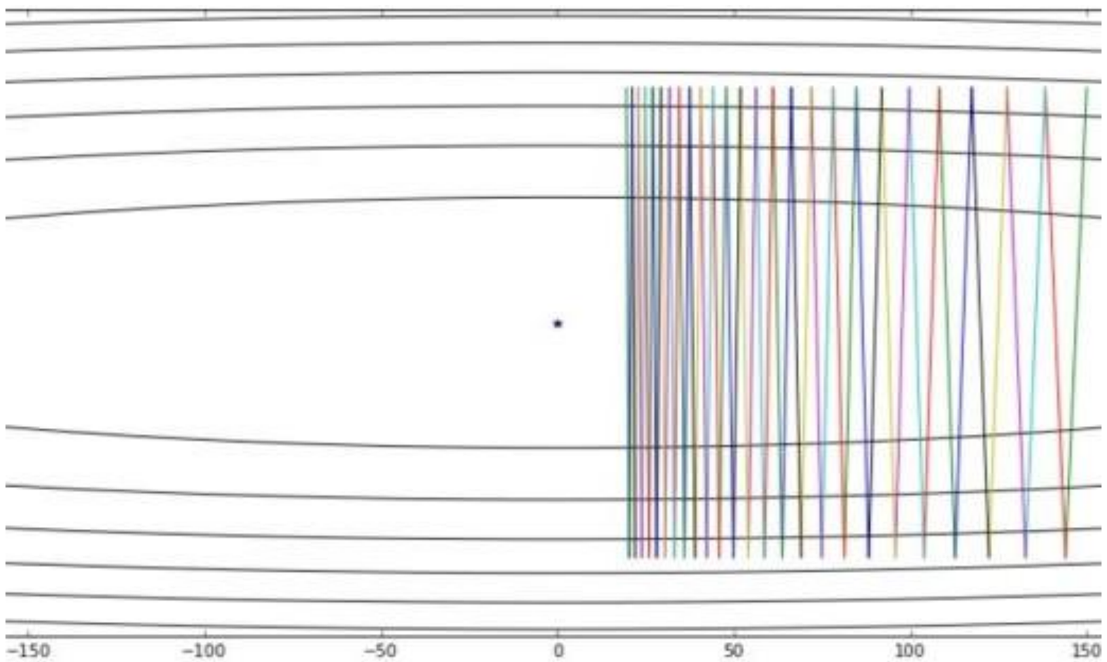
- Rprop, quickprop
- Pros/Cons (Ignores dependencies between dimensions, unexpected behaviour)

1.2 Momentum

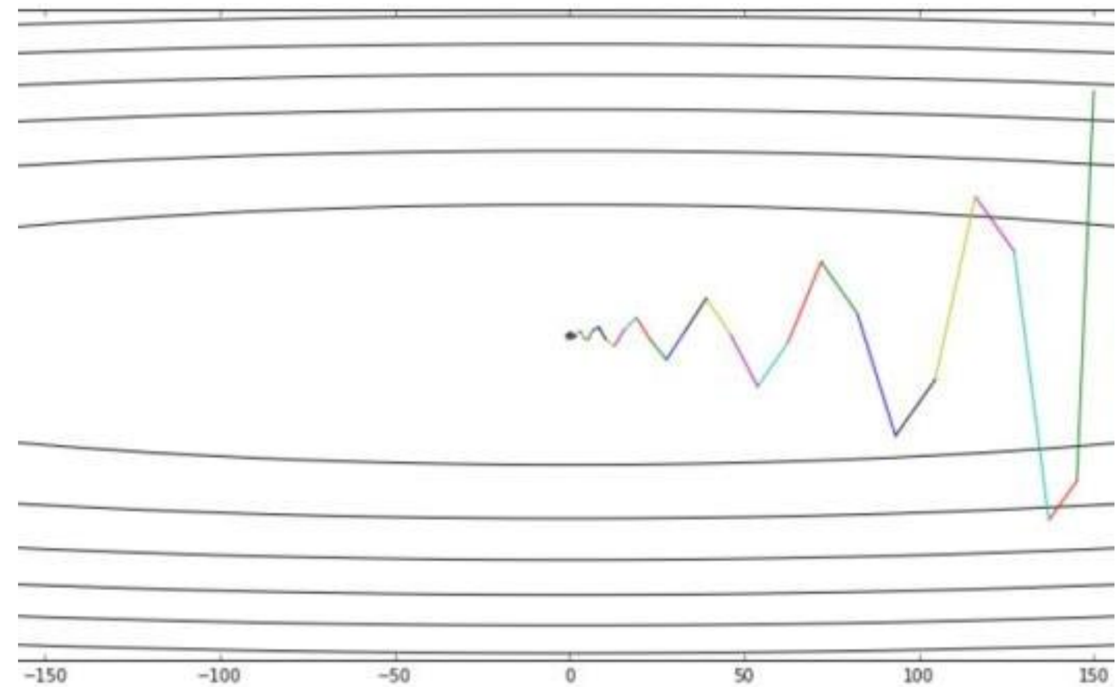
Maintain a running average of all past steps

- In directions in which the convergence is smooth, the average will have a large value
- In directions in which the estimate swings, the positive and negative swings will cancel out in the average
- Update with the running average, rather than the current gradient

1.2 Momentum



Use GD



GD + Momentum

- Reduces updates for dimensions whose gradients change directions.

- Increases updates for dimensions whose gradients point in the same directions.

1.3 More recent methods

- Variance normalised methods:
  - RMS-Prop
    - Updates are by parameter
    - The mean squared derivative is a running estimate of the average squared derivative
  - Adam
    - RMS-Prop considers only second moment (Variance)
    - Adam = RMS-Prop + momentum

  - Other variants
    - Adagrad
    - AdaDelta
    - AdaMax

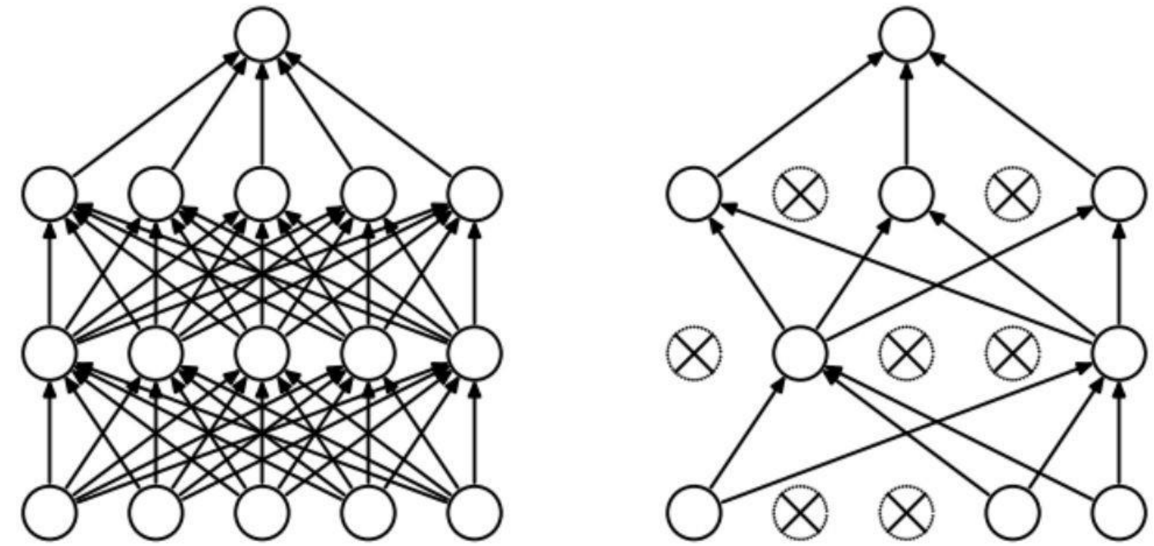
2.1 Parameter Initialization

- Can we start with zero initial weights?
  - The derivative with respect to loss function is same for every w , thus all weights have same value in subsequent iterations
- Can we have equal initial weights?
  - Harms stochastic optimisation techniques like GD
  - All the neurons will follow the same gradient, and will always end up doing the same thing as one another
- Methods to initialize
  - Random (typically gaussian)
  - Xavier
  - Pretraining

2.2 Annealing the learning rate

- Usually helpful to anneal the learning rate over time
- High learning rates can cause the parameter vector to bounce around chaotically, unable to settle down into deeper, but narrower parts of the loss function
- **Step decay**: Reduce the learning rate by some factor after some number of epochs (i.e. reduce by a half every 5 epochs, or by 0.1 every 20 epochs).
- **Plateau decay**: Watch the validation error or loss while training with a fixed learning rate, and reduce the learning rate by a constant factor whenever the validation performance stops improving
- **Exponential decay**: It has the mathematical form lr = lr0 * e^(−kt), where lr0, k are hyperparameters and t is the iteration number

2.3 Random Dropout


(a) Standard Neural Net (b) After applying dropout.

- Implementation
  - Dropout each unit with probability p
  - No parameters dropped at test time
- Results
  - Network is forced to learn a distributed representation
    Improves generalization by eliminating neuron
  - co-dependencies within a layer
- Typical dropout probability is around 0.1 to 0.5

2.4 Batch Normalization

- We assume that every mini-batch is statistically similar to every other batch
- In reality, they can be very far apart- Covariance Shift
- Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
  - Then move the entire collection to the appropriate location
- Applied on affine combination of inputs
- BN aggregates the statistics over a minibatch and normalizes the batch by them

2.5 Others:

- Scheduler
- Shuffle the dataset

  If not shuffle, the network will remember the data order!

  In hw1p2, it is a frame-level task, so you need to shuffle in frames.
- Weight decay:

  L2 regularization for (not) overfitting:

$$loss = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \frac{1}{2} w\theta^2$$

$$\theta = \theta - \Delta\theta - w\theta$$

- Early Stopping for (not) overfitting