Efficient Optimization Methods for Deep Learning

Recitation 3

Rohit P Barnwal, Advait Gadhikar

- Review of Optimization
- Optimization methods
- Training tips in PyTorch

Mini Batch Gradient Descent

1. What is it?
    a. Performs update on every mini batch of data
2. Why mini batch?
    a. Batch gradient descent that uses the whole dataset for one update, slow and intractable for large datasets to fit into memory.
    b. Stochastic gradient descent that updates parameters for each data point: high variance updates, takes time to converge
    c. Trade Off: Take mini batches of data and compute the gradient over the mini batch for every update.

Mini Batch Gradient Descent (contd.)

- 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:

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

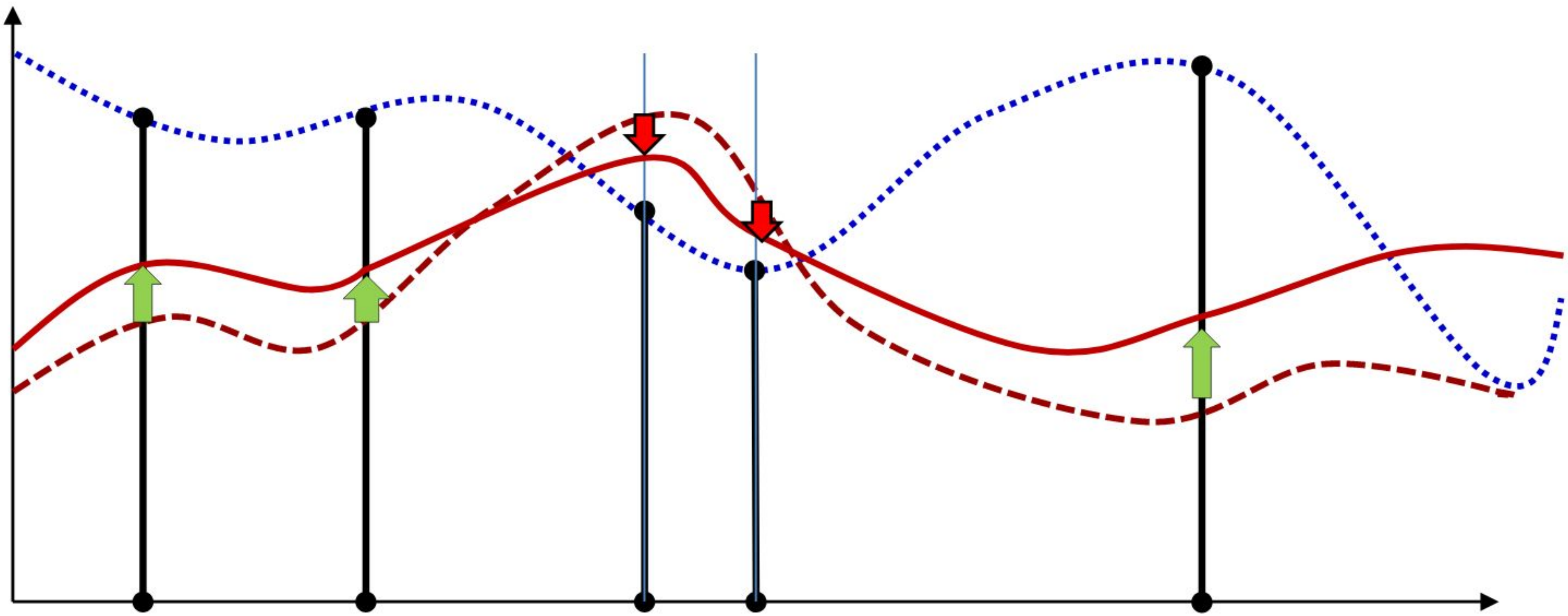
  - Let η be the learning rate, compute the update:

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

Mini Batch Gradient Descent (contd.)

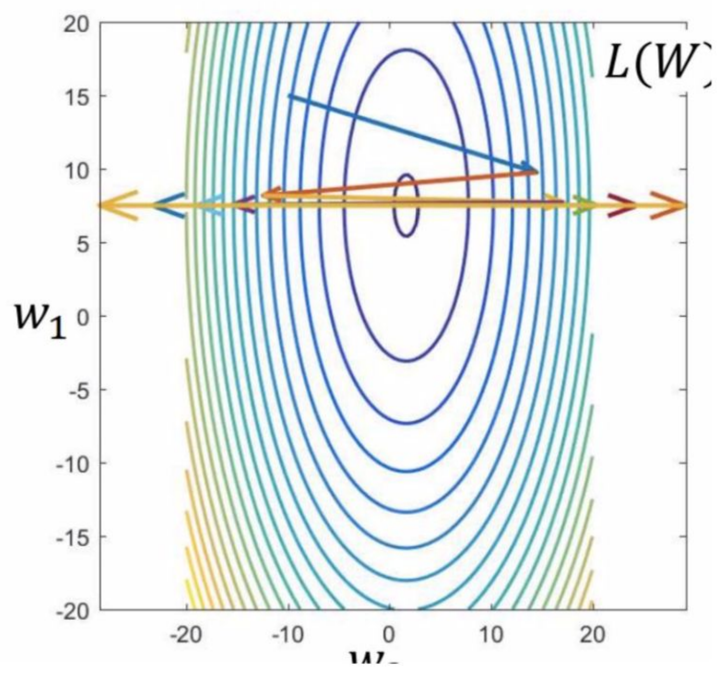
1. The good things about mini batch gradient descent
   a. Reduce variance between each update
   b. Computation is faster
2. How to decide the size of the mini-batch
   a. Mini batch sizes usually vary from 32 to 256
   b. Small batches: Slow convergence and High variance
   c. Large batches: Harder to escape from local minima

Empirical Risk Minimization with SGD



Issues with Gradient Descent

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

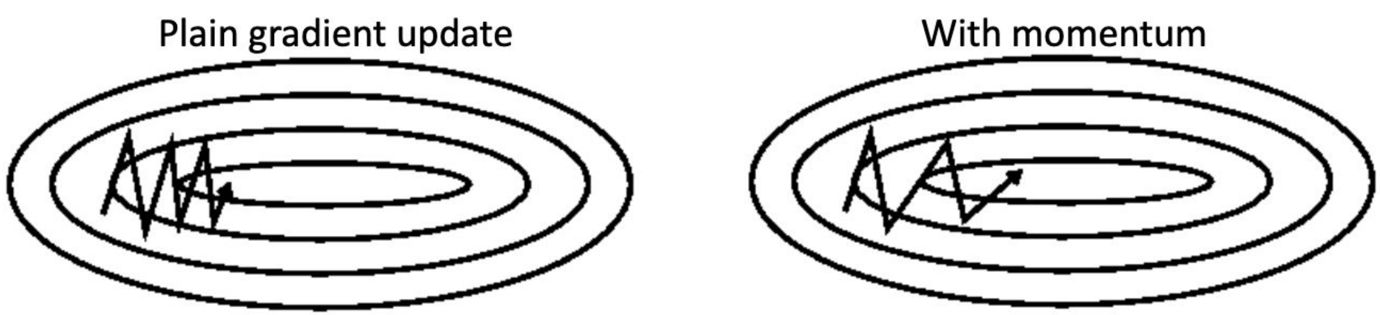


Solutions

1. Try to normalize curvature in all directions
   a. Second order methods like Newton's Method
   b. However, second order methods are computationally infeasible, require inversion of the Hessian matrix
2. Treat each dimension independently
   a. Rprop, Quickprop
   b. Ignores dependencies between dimensions

Momentum

1. Maintain running average of all past gradients(steps)
    a. In directions in which the convergence is smooth, the average will have a large value
    b. In directions in which the estimate swings, the positive and negative swings will cancel out in the average
    c. Update with the running average, rather than the current gradient



Momentum

1. Reduces updates for dimensions whose gradients change
2. Increases updates for dimensions whose gradients point in the same directions

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

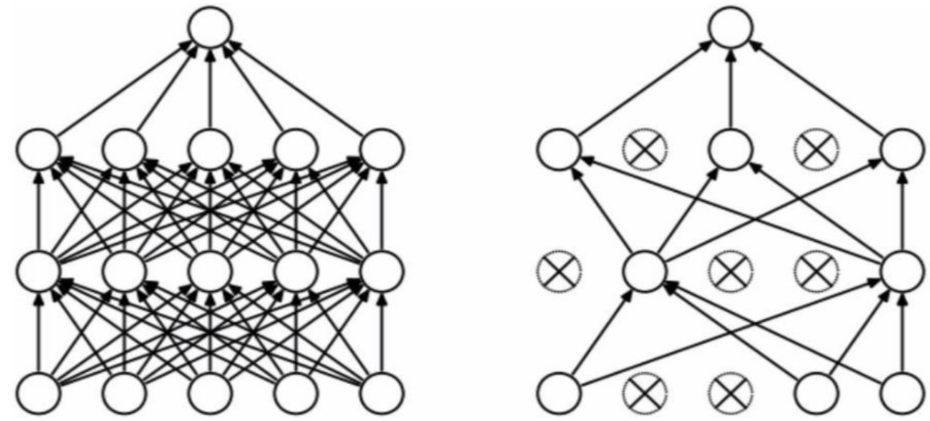
$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

More Recent Methods

Variance Normalized 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
- Interesting visualizations here.

Random Dropout



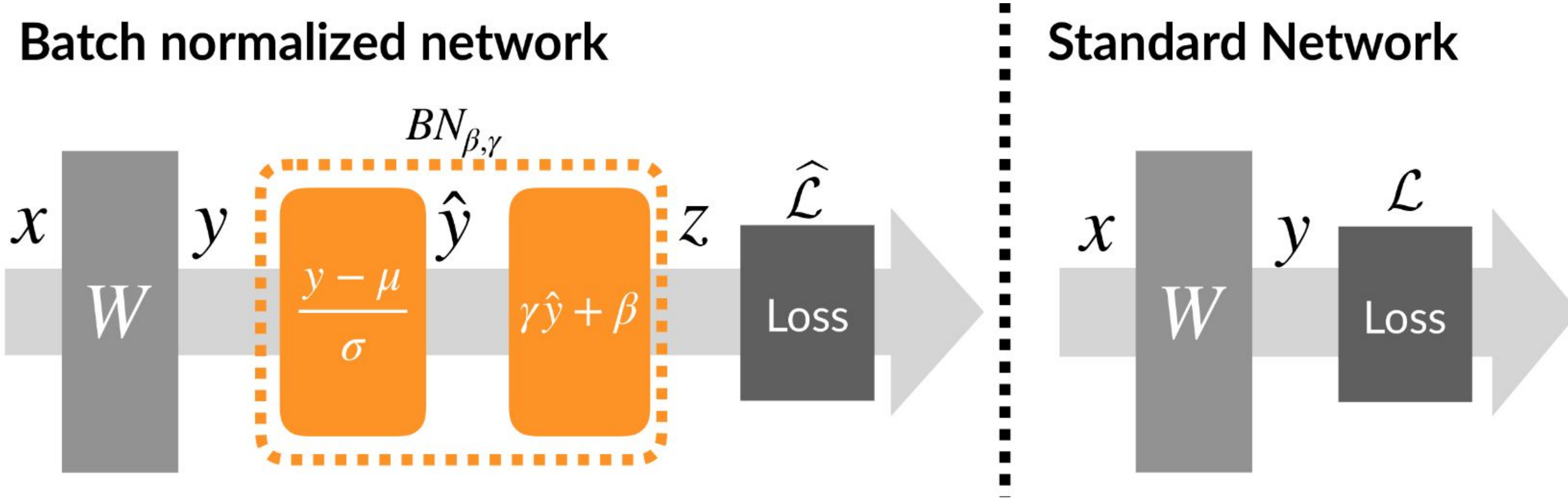
(a) Standard Neural Net (b) After applying dropout.

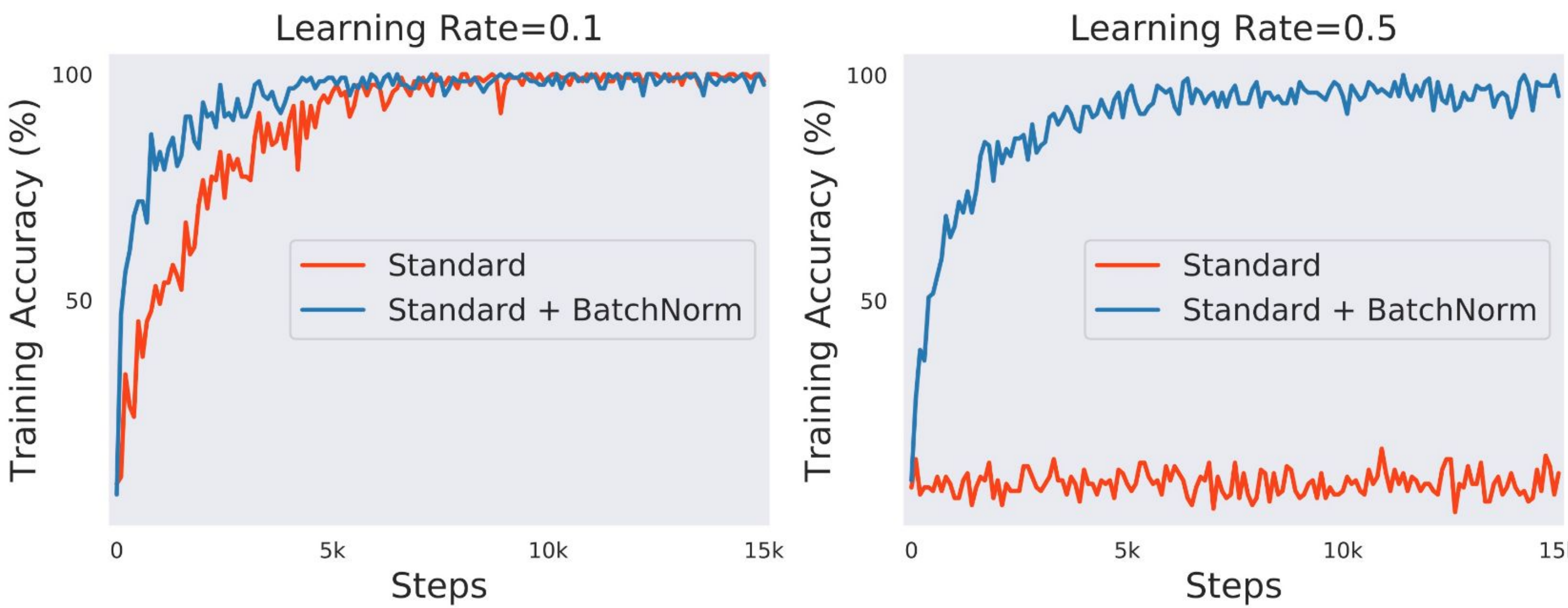
- Implementation
  - Dropout each unit with probability p
  - No parameters dropped at test time
- Results
  - Network is forced to learn a distributed representation

    Improves generalization by eliminating neuron
  - co-dependencies within a layer
- Typical dropout probability is around 0.1 to 0.5

Batch Normalization

1. Training assumes every batch to be statistically similar, but in practice this is not necessary.
2. BN shifts inputs at every layer to have zero mean and unit variance
3. Applied on affine combination of inputs
4. Allows training with larger learning rates by shifting the input to activations to be zero mean and unit variance
5. Prevents SGD to be stuck in a sharp local minima





Credits: http://gradientscience.org/batchnorm/

Learning Rate Annealing

- Usually helpful to anneal the learning rate over time
- High learning rates can cause the weight vector to bounce around and not settle into a narrower minima of the loss function
- Step Decay:
  - Reduce learning rate by some factor after every ‘n’ number of epochs (reduce LR by a factor of 10 every 5 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 performs stops improving, settles at a certain value
- Exponential Decay:
  - It modifies the learning rate as
    - $\eta = \eta_0(\exp(-kt))$
    - Here, k is a hyperparameter and t is the iteration number

Parameter Initialization

1. Can we start with zero initial weights?
   a. The derivative with respect to the loss will be the same for every W, thus all weights will have same values in subsequent iterations
2. What about equal initial weights?
   a. All neurons will follow the same gradient update
   b. Will not work well with SGD
3. Initialization methods
   a. Random(typically drawn from a Gaussian distribution)
   b. Xavier (inputs of each activation fall within its range)
   c. Pretraining

Other optimization methods

1. Weight Decay
   a. L2 regularization for not overfitting

$$loss = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{(}y_i)) + \frac{\lambda}{2}\mathbf{W}^2$$

$$w_i \leftarrow w_i - \eta\frac{\partial loss}{\partial w_i} - \eta\lambda w_i$$

   b. Early stopping to prevent overfitting
      i. Train loss decreases but val loss saturates

Other optimization methods

1. Shuffle the dataset
    a. If the data is not shuffled the network can remember the order in which data was presented
    b. We might get a cyclic behaviour in the model performance instead of a converging model
    c. Hence, important to randomly shuffle data