Exploring Adaptive Learning Methods for Convex Optimization

Bhavdeep Sethi
Rhea Goel

Context (Backprop)

- Most popular method for training neural networks
- Uses Stochastic Gradient Descent
- But it is too sensitive to learning rate

    too low: slow convergence

    too high: divergence

- Fixed learning step cannot handle shape of typical multi dimensional error functions (lots of local minima)

Why Adaptive Learning?

- Large eigenvalue of H→ steep curve → need small learning rate
- To learn all weights reasonably well, learning rate should be inverse Hessian
- But we don’t know the Hessian, so we either approximate or use adaptive learning

- Let’s look at some methods.

Adaptive learning rates!

- What all is out there:
    - Momentum
    - NAG
    - ADAGRAD
    - ADADELTA

Stochastic Gradient Descent

- Updates each weight by subtracting the gradient of loss (wrt the weight), scaled by learning rate

- Highly dependent on learning rate

- Update Rule:

$$\theta_{t+1} = \theta_t - \eta \nabla l(\theta_t)$$

Momentum

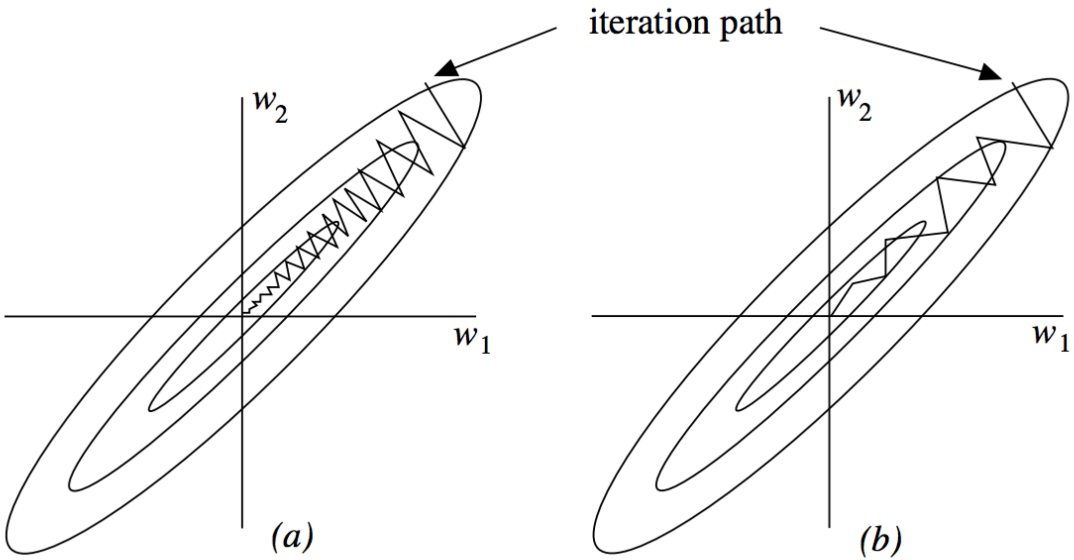
- Re-using the gradient value from the previous iteration, scaled by a momentum hyperparameter μ, as follows:

$$v_{t+1} = \mu v_t - \eta \nabla l(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

- Instead of following the negative gradient direction, a weighted average of the current gradient and previous direction is computed

Momentum (contd.)

Provides “inertia” to avoid excessive oscillations in narrow valleys



BachProp without Momentum

BachProp with Momentum

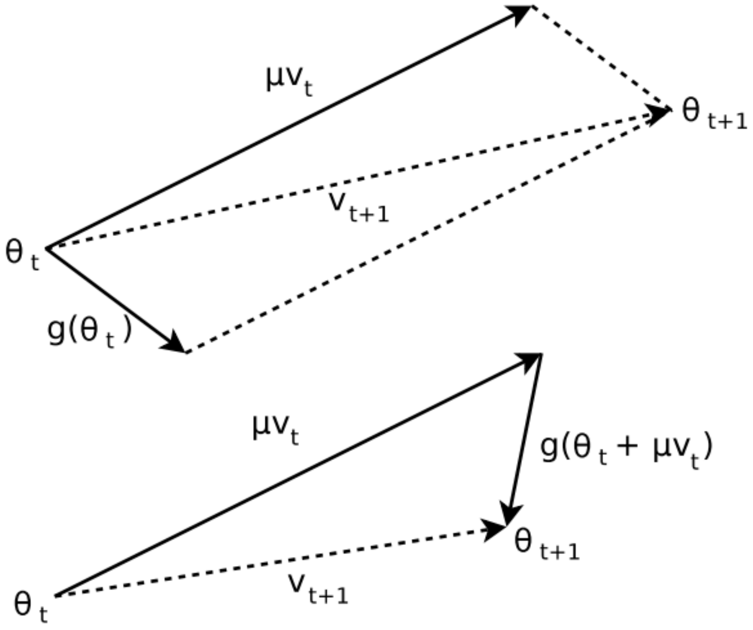
Nestorov’s Accelerated Gradient

- The gradient of the loss at each step is computed at $\theta_t + \mu v_t$ instead of θ_t
- Computes the gradient at the new parameter location but without considering the gradient term
- NAG behaves more stably than regular momentum in many situations.

Update rule: $v_{t+1} = \mu v_t - \eta \nabla l(\theta_t + \mu v_t)$ $\theta_{t+1} = \theta_t + v_{t+1}$

NAG (contnd.)

Quicker and more responsive way of changing v



Adagrad update rule

The learning rate is adapted component-wise, and is given by the square root of sum of squares of the historical, component-wise gradient.

AdaGrad alters this(SGD) update to adapt based on historical information, so that frequently occurring features in the gradients get small learning rates and infrequent features get higher ones.

AdaGrad (contd.)

$$g_{t+1} = g_t + \nabla l(\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta \nabla l(\theta_t)}{\sqrt{g_{t+1} + \varepsilon}}$$

- Each feature dimension has it’s own learning rate!
  - Adapts with t
  - Takes geometry of the past observations into account
  - Primary role of η is determining rate the first time a feature is encountered

AdaDelta

It was derived from ADAGRAD in order to improve upon the two main drawbacks of the method:

- the continual decay of learning rates throughout training
- the need for a manually selected global learning rate.

AdaDelta (contd.)

$$g_{t+1} = \gamma g_t + (1-\gamma)\nabla l(\theta_t)^2 \qquad x_{t+1} = \gamma x_t + (1-\gamma)\nu_{t+1}^2$$

$$\nu_{t+1} = -\frac{\sqrt{x_t + \varepsilon}}{\sqrt{g_{t+1} + \varepsilon}}\nabla l(\theta_t) \qquad \theta_{t+1} = \theta_t + \nu_{t+1}$$

- Using a window instead of time t, the denominator of ADAGRAD cannot accumulate to infinity and instead becomes a local estimate using recent gradients.

- Some approximation to the Hessian is made, but costs only one gradient computation per iteration by leveraging information from past updates.

AdaDelta (contnd.)

- No manual setting of a learning rate
- Insensitive to hyperparameters.
- Separate dynamic learning rate per-dimension
- Minimal computation over gradient descent.
- Robust to large gradients, noise and architecture choice.
- Applicable in both local or distributed environments.

Our Setup

- **Framework: Theano**
- **Architecture : Deep NNET (1 ReLU Hidden Layer, 200 neurons)**
- **Machine:** g2.2xlarge (26 ECUs, 8 vCPUs, 2.6 GHz, Intel Xeon E5-2670, 15 GiB memory, 1 x 60 GiB Storage Capacity)
- **Data:** Each datapoint is a 8x8 image of a digit.

  Classes: 10

  Samples per class: ~180

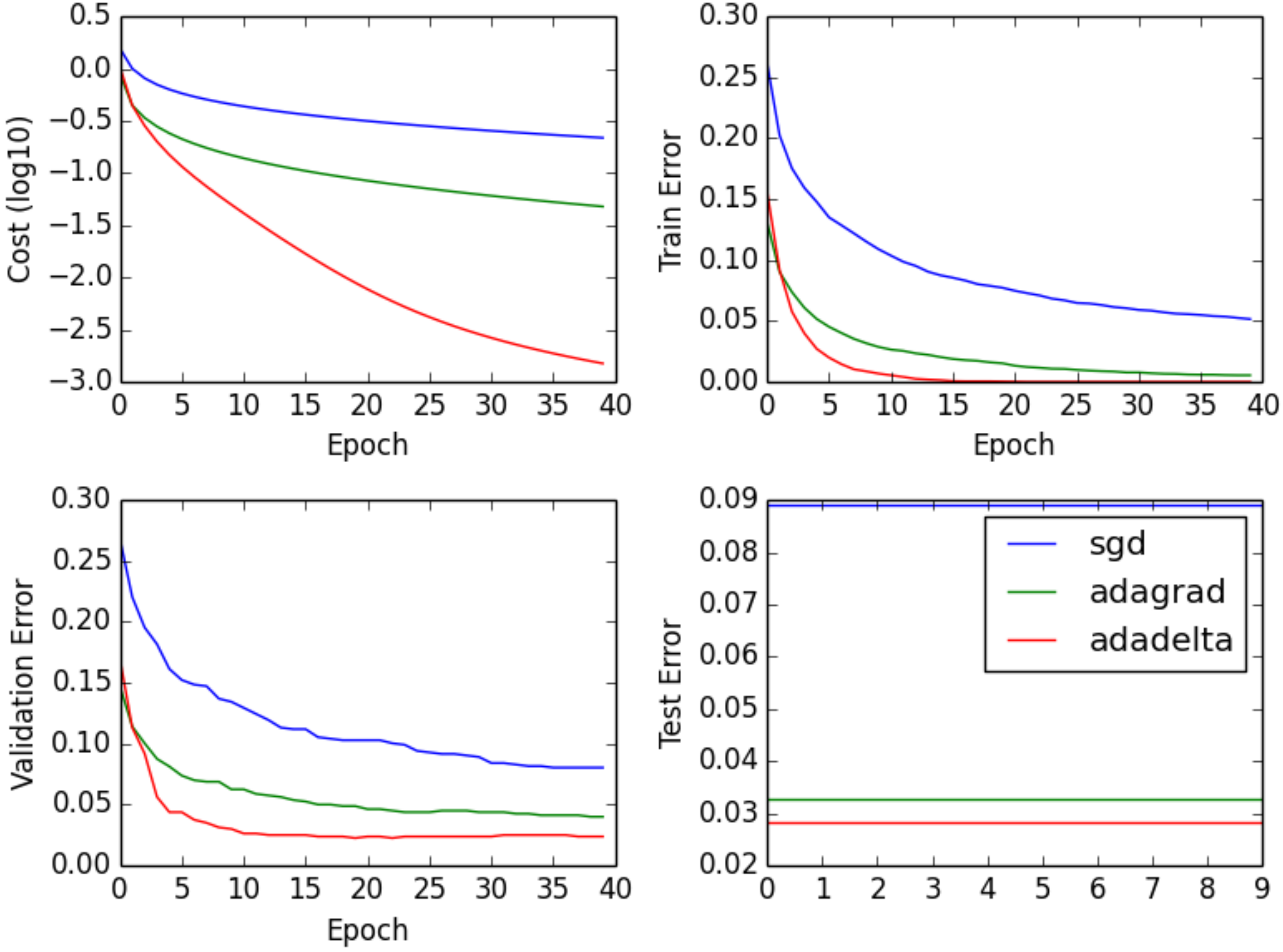
  Samples total: 1797

  Dimensionality: 64

  Features: integers (0-16)

NNET training: Stochastic Gradient

- Batch: accumulate gradient for all data points in the training set before updating weights
- Online: weights are updated immediately after seeing each data point (could be noisy)
- **Mini-batches (a compromise)**: weights are updated after every n data points



Cost (log10)
Epoch
Train Error
Epoch
Validation Error
Epoch
Test Error
sgd
adagrad
adadelta

Future Work

- More implementations
  - Momentum
  - NAG
  - Rprop
  - RMSProp (Aug, 2013)
  - AdaSecant (Dec 23, 2014)
  - ESGD (Feb, 2015)
- More Data sets:
  - MNIST
  - 20NewsGroup
  - Labeled Faces in the Wild (LFW) people