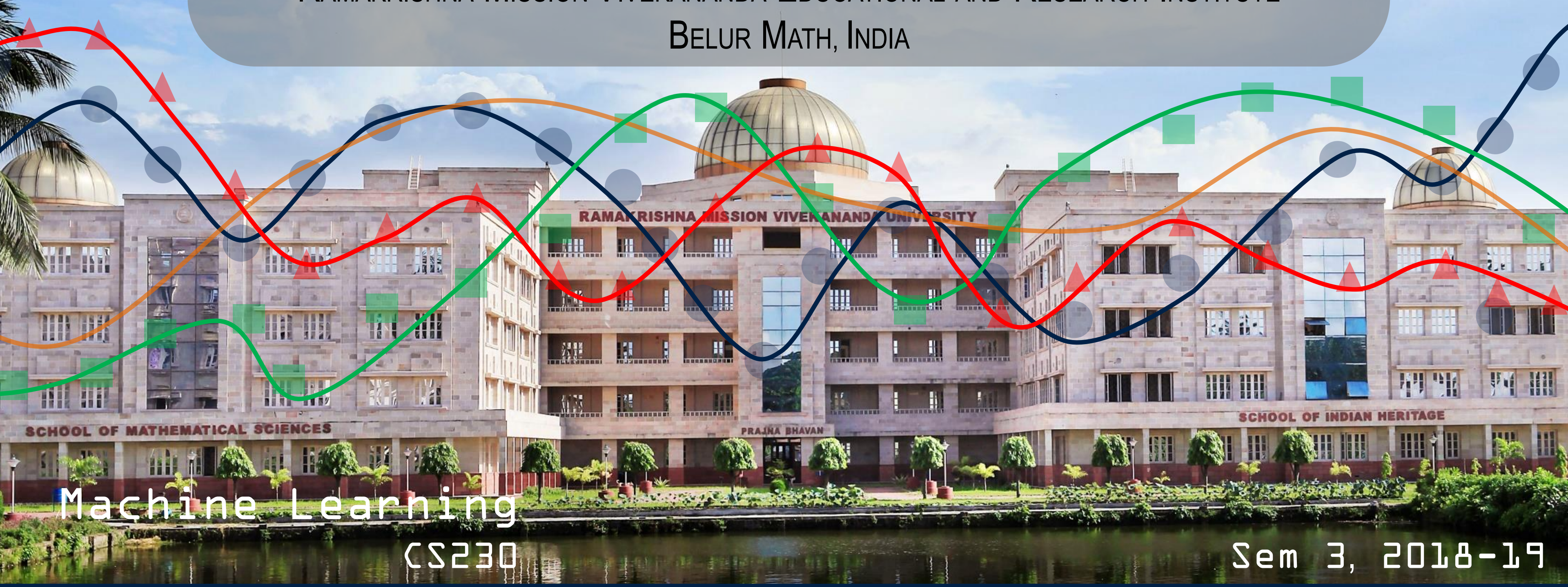
Optimization in Neural Networks

Dripta Mj
Department of Mathematics
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, India



Machine Learning
CS230
Sem 3, 2018-19

Gradient based optimization

- Gradient descent:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \xi \sum_{n=1}^{N} \nabla_{\mathbf{w}^{(t)}} L\big(y^{(n)}, y^{*(n)}(\mathbf{w}^{(t)})\big)$$

  where ξ is the learning rate.

- Frequency of updates:
  - Batch gradient descent: Updates after evaluating the loss gradient w.r.t. all training examples.
  - Stochastic gradient descent: Updates after evaluating the loss gradient w.r.t. every training example.
  - Mini-batch gradient descent: Updates after evaluating the loss gradient w.r.t. a subset of the training dataset.

- Type of updates:
  - Fixed learning rate
  - With momentum
  - Adaptive learning rate
  - Adaptive learning rate + Momentum

Ravines

- Stochastic gradient descent has difficulty navigating ravines.

Momentum based gradient descent

- Builds up speed in directions with gentle and consistent gradient.
- Damps oscillations in direction of high curvature.
- The effect of the gradient is to increment the previous velocity. The velocity also decay by a factor β which is slightly less than one.
- Running average makes the gradient less dependent on its current value, and rely more on the general behaviour of the gradient in the past updates.
- More interested in the expected value of the gradient rather on the particular gradient value at a particular iteration.

$$\mathbf{v}^{(t)} = \beta \mathbf{v}^{(t-1)} + (1-\beta)\nabla_{\mathbf{w}^{(t)}} L$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \xi \mathbf{v}^{(t)}$$

Momentum based gradient descent: updates

$$\mathbf{v}^{(0)} = 0$$

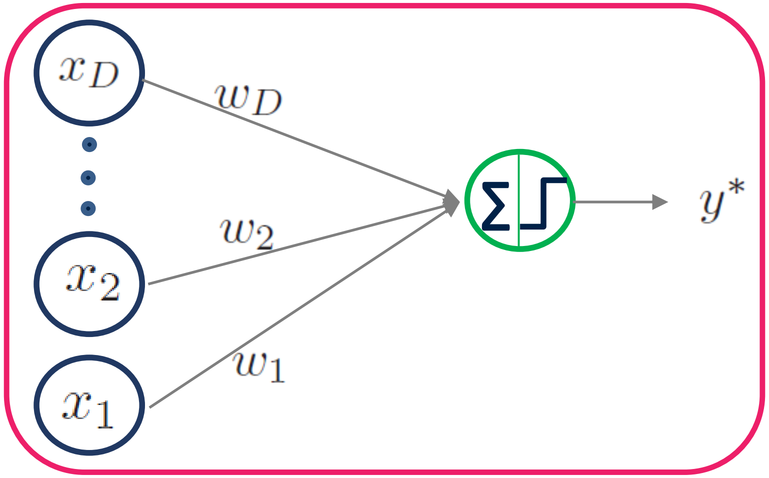
- Update 1: $\mathbf{v}^{(1)} = \beta \mathbf{v}^{(0)} + (1-\beta)\nabla_{\mathbf{w}^{(1)}} L$
$$= (1-\beta)\nabla_{\mathbf{w}^{(1)}} L$$

- Update 2: $\mathbf{v}^{(2)} = \beta \mathbf{v}^{(1)} + (1-\beta)\nabla_{\mathbf{w}^{(2)}} L$
$$= \beta(1-\beta)\nabla_{\mathbf{w}^{(1)}} L + (1-\beta)\nabla_{\mathbf{w}^{(2)}} L$$

- Update 3: $\mathbf{v}^{(3)} = \beta \mathbf{v}^{(2)} + (1-\beta)\nabla_{\mathbf{w}^{(3)}} L$
$$= \beta\big(\beta(1-\beta)\nabla_{\mathbf{w}^{(1)}} L + (1-\beta)\nabla_{\mathbf{w}^{(2)}} L\big) + (1-\beta)\nabla_{\mathbf{w}^{(3)}} L$$
$$= (1-\beta)\big[\beta^2\nabla_{\mathbf{w}^{(1)}} L + \beta^1\nabla_{\mathbf{w}^{(2)}} L + \nabla_{\mathbf{w}^{(3)}} L\big]$$

- Update t: $\mathbf{v}^{(t)} = (1-\beta)\big[\beta^{(t-1)}\nabla_{\mathbf{w}^{(1)}} L + \beta^{(t-2)}\nabla_{\mathbf{w}^{(2)}} L + ... + \nabla_{\mathbf{w}^{(t)}} L\big]$

Shortcomings



- Binary classification problem
- Binary cross-entropy loss function
- Update: $w_i := w_i - \xi \frac{\partial L}{\partial w_i}$
- The gradient can be computed as $\frac{\partial L}{\partial w_i} = (y^* - y)x_i$
- Suppose the i-th feature is sparse, i.e. $x_i = 0$ for most of the examples.
- In that case the weight w_i associated with x_i will have few updates as the gradient is 0 in most cases.
- Our results can be seriously impacted if x_i happens to be a very important feature.
- Want an algorithm that gives higher learning rate to sparse features.

Adagrad

- Adapts the learning rate of the parameters.
  - Higher learning rate for sparse features.
  - Lower learning rate for dense features.
- More updates of a parameter indicate more decay of its learning rate.

$$\mathbf{s}^{(t)} = \mathbf{s}^{(t-1)} + \left(\nabla_{\mathbf{w}^{(t)}} L\right)^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\xi}{\sqrt{\mathbf{s}^{(t)} + \epsilon}} \odot \nabla_{\mathbf{w}^{(t)}} L$$

- Method appropriate for dealing with sparse datasets.

RMSProp

- Adagrad reduces the learning rates of parameters associated with dense features very fast. So the corresponding weight updates will be small.

- Adagrad: Sum of squares of the past gradients.

- RMSProp: Exponentially decaying (moving) average of the squares of past gradients.

$$\boxed{\begin{aligned} \mathbf{s}^{(t)} &= \beta \mathbf{s}^{(t-1)} + (1-\beta)\left(\nabla_{\mathbf{w}^{(t)}} L\right)^2 \\ \mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \frac{\xi}{\sqrt{\mathbf{s}^{(t)}} + \epsilon} \odot \nabla_{\mathbf{w}^{(t)}} L \end{aligned}}$$

- RMSProp addresses the learning rate decay problem of Adagrad.

ADAM

- Adaptive Moment Estimation (ADAM).
- Adaptive learning rate + Momentum
  - Keeps an exponentially decaying average of past gradients (like momentum).
  - Also keeps an exponentially decaying average of past squared gradients (like RMSProp).

$$\boxed{\begin{aligned}\mathbf{v}^{(t)} &= \beta_1 \mathbf{v}^{(t-1)} + (1-\beta_1)\nabla_{\mathbf{w}^{(t)}} L \\ \mathbf{s}^{(t)} &= \beta_2 \mathbf{s}^{(t-1)} + (1-\beta_2)\big(\nabla_{\mathbf{w}^{(t)}} L\big)^2\end{aligned}}$$

- $\mathbf{v}^{(t)}$ is the vector of first moment (mean) estimates of the mean of the gradients.
- $\mathbf{s}^{(t)}$ is the vector of second moment (uncentered variance) estimates of the gradients.
- Bias corrections: $\overline{\mathbf{v}}^{(t)} = \dfrac{\mathbf{v}^{(t)}}{1-\beta_1^t} \qquad \overline{\mathbf{s}}^{(t)} = \dfrac{\mathbf{s}^{(t)}}{1-\beta_2^t}$
- Update of weights: $\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} - \dfrac{\xi}{\sqrt{\overline{\mathbf{s}}^{(t)}} + \epsilon} \odot \overline{\mathbf{v}}^{(t)}$

ADAM: Bias correction

- We have seen that $\mathbf{v}^{(t)} = (1-\beta_1)\sum_{j=1}^{t}\beta^{t-j}\nabla_{\mathbf{w}^{(j)}}L$
- Taking Expectation on both sides yields

$$\begin{aligned}
\mathbb{E}\big[\mathbf{v}^{(t)}\big] &= \mathbb{E}\Big[(1-\beta_1)\sum_{j=1}^{t}\beta^{t-j}\nabla_{\mathbf{w}^{(j)}}L\Big] \\
&= (1-\beta_1)\mathbb{E}\Big[\sum_{j=1}^{t}\beta^{t-j}\nabla_{\mathbf{w}^{(j)}}L\Big] \\
&= (1-\beta_1)\sum_{j=1}^{t}\mathbb{E}\Big[\beta^{t-j}\nabla_{\mathbf{w}^{(j)}}L\Big] \\
&= (1-\beta_1)\sum_{j=1}^{t}\beta^{t-j}\mathbb{E}\Big[\nabla_{\mathbf{w}^{(j)}}L\Big]
\end{aligned}$$

- Assumption: All gradients come from the same distribution, i.e.

$$\nabla_{\mathbf{w}^{(1)}}L = \nabla_{\mathbf{w}^{(2)}}L = \ldots\ldots\ldots = \nabla_{\mathbf{w}^{(j)}}L = \nabla_{\mathbf{w}}L$$

ADAM: Bias correction

- So then we have

$$\begin{aligned}
\mathbb{E}\left[\mathbf{v}^{(t)}\right] &= (1-\beta_1)\sum_{j=1}^{t}\beta^{t-j}\mathbb{E}\left[\nabla_{\mathbf{w}}L\right] \\
&= (1-\beta_1)\mathbb{E}\left[\nabla_{\mathbf{w}}L\right]\sum_{j=1}^{t}\beta_1^{t-j} \\
&= (1-\beta_1)\mathbb{E}\left[\nabla_{\mathbf{w}}L\right]\left(\beta_1^{t-1}+\beta_1^{t-2}+....+\beta_1^{0}\right) \\
&= (1-\beta_1)\mathbb{E}\left[\nabla_{\mathbf{w}}L\right]\left(\frac{1-\beta_1^t}{1-\beta_1}\right) \\
&= \mathbb{E}\left[\nabla_{\mathbf{w}}L\right]\left(1-\beta_1^t\right)
\end{aligned}$$

Therefore

$$\mathbb{E}\left[\frac{\mathbf{v}^{(t)}}{1-\beta_1^t}\right] = \mathbb{E}\left[\nabla_{\mathbf{w}}L\right]$$

$$\mathbb{E}\left[\overline{\mathbf{v}}^{(t)}\right] = \mathbb{E}\left[\nabla_{\mathbf{w}}L\right]$$

Comparison

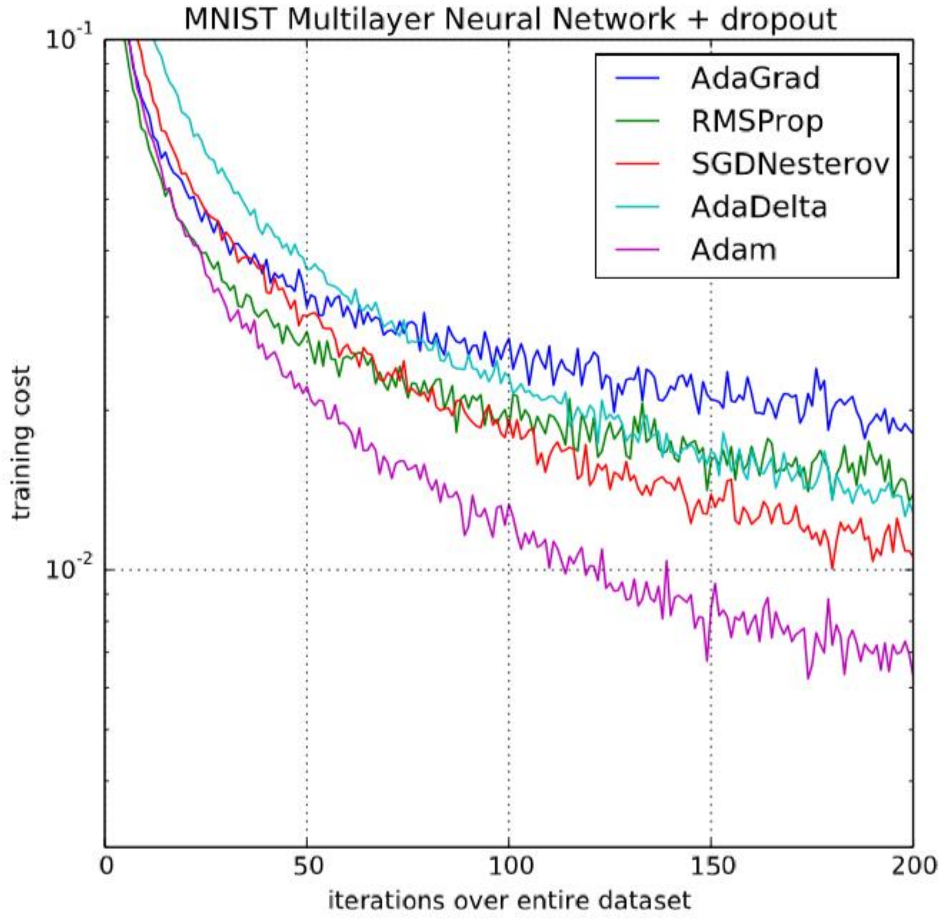


Figure source: Kingma and Lei Ba, ADAM: A method for stochastic optimization, 2015.