Deep Learning II: Optimization

CSci 5525: Machine Learning

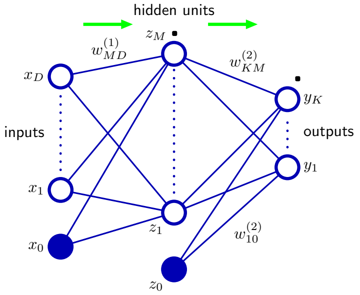
Instructor: Nicholas Johnson

October 15, 2020

Announcements

- HW2 is due tonight 11:59 PM CDT
- Exam 1 will be posted on Oct 20 (due 48 hours later)
- Course feedback form
- Project proposal grades/feedback posted

The Picture: General



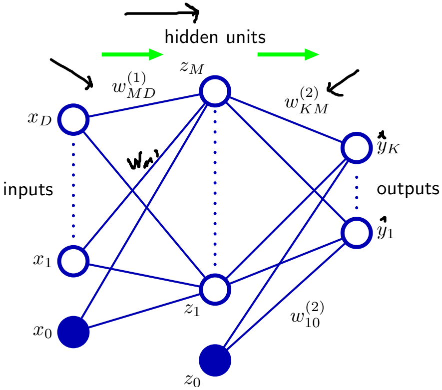
- Between input and hidden layers

$$w^{(1)}_{ji} \qquad a_j = \sum_{i=0}^{D} w_{ji} x_i \qquad z_j = h(a_j)$$

- Between hidden and output layers

$$w^{(2)}_{kj} \qquad a_k = \sum_{j=0}^{M} w_{kj} z_j \qquad \hat{y}_k = h(a_k)$$

The Picture: General



$$\hat{y}_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w^{(2)}_{kj}\, h\left(\sum_{i=0}^{D} w^{(1)}_{ji} x_i\right)\right)$$

Training

- Consider ERM problem with squared loss

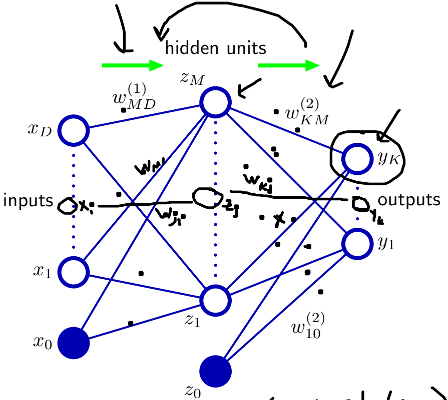
$$\text{argmin}_\theta \frac{c}{n} \sum_{i=1}^{n} \|\mathbf{y_i} - \hat{\mathbf{y}}_i\|_2^2$$

  where $\hat{\mathbf{y}}_i = f(\mathbf{x}_i, \theta)$ is the predicted output of the network and $\mathbf{y}_i$ is the target vector and $c > 0$ is a constant
- For the ith sample, the error is $E_i = \frac{1}{2}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2$ where $\hat{y}_k$ is the kth output node value and y_k is the kth target value

Training

- The process of sending a sample through the neural network to compute the prediction is called the forward propagation step
- To train the network, we use an algorithm called backpropagation (backprop)
- Backprop is a message-passing algorithm which attributes credit of the error to different nodes depending on their values
  - Nodes which have higher activation contribute more and are assigned higher credit for the error
- Backprop is an algorithm that computes the chain rule of calculus
  - Chain rule is used to compute derivatives of functions formed by composing other functions whose derivatives are known

The Picture: General



$\sigma(a_k) = h(a_k)$

$$\hat{y}_k(\mathbf{x}, \mathbf{w}) = \overset{h:}{\sigma}\left(\sum_{j=0}^{M} w^{(2)}_{kj}\, h\left(\sum_{i=0}^{D} w^{(1)}_{ji} x_i\right)\right)$$

Backpropagation Derivation

The squared error on a single example is $E = \frac{1}{2}\sum_k (y_k - \hat{y}_k)^2$

Compute gradients to propagate errors back through second layer:

$\hat{y}_k = h(a_k)$

$$
\begin{aligned}
\frac{\partial E}{\partial w_{k,j}} &= -\underline{(y_k - \hat{y}_k)}\frac{\partial \hat{y}_k}{\partial w_{k,j}} \\
&= -(y_k - \hat{y}_k)\frac{\partial h(a_k)}{\partial w_{k,j}} \\
&= -\underline{(y_k - \hat{y}_k)}\,\underline{h'(a_k)}\,\underline{\frac{\partial a_k}{\partial w_{k,j}}} \\
&= -(y_k - \hat{y}_k)h'(a_k)\frac{\partial}{\partial w_{k,j}}\left(\sum_j w_{k,j} z_j\right) \\
&= -(y_k - \hat{y}_k)h'(a_k)z_j \\
&\boxed{= -z_j \times \Delta_k}
\end{aligned}
$$

where $\Delta_k = (y_k - \hat{y}_k) \times h'(a_k)$ and z_j is jth hidden unit

Backprop: Output Layer Update Equations

- $\frac{\partial E}{\partial w_{k,j}} = -z_j \times \Delta_k$
- Use gradients in gradient descent: $w_{k,j} \leftarrow w_{k,j} - \alpha \frac{\partial E}{\partial w_{k,j}}$
- Backpropagate error to output layer:

$$w_{k,j} \leftarrow w_{k,j} + \alpha \times z_j \times \Delta_k$$

where $\Delta_k = (y_k - \hat{y}_k) \times h'(a_k)$ and α is learning rate

Backpropagation Derivation (cont.)

$$
\begin{aligned}
\frac{\partial E}{\partial w_{j,i}} &= -\sum_k (y_k - \hat{y}_k)\frac{\partial \hat{y}_k}{\partial w_{j,i}} = -\sum_k (y_k - \hat{y}_k)\frac{\partial h(a_k)}{\partial w_{j,i}} \\
&= -\sum_k \underbrace{(y_k - \hat{y}_k)h'(a_k)}_{\Delta_k}\frac{\partial a_k}{\partial w_{j,i}} = -\sum_k \Delta_k \frac{\partial}{\partial w_{j,i}}\left(\sum_j w_{k,j} z_j\right) \\
&= -\sum_k \Delta_k w_{k,j}\frac{\partial z_j}{\partial w_{j,i}} = -\sum_k \Delta_k w_{k,j}\frac{\partial h(a_j)}{\partial w_{j,i}} \\
&= -\sum_k \Delta_k w_{k,j} h'(a_j)\frac{\partial a_j}{\partial w_{j,i}} \\
&= -\sum_k \Delta_k w_{k,j} h'(a_j)\frac{\partial}{\partial w_{j,i}}\left(\sum_i w_{j,i} x_i\right) \\
&= -\sum_k \Delta_k w_{k,j} h'(a_j) x_i = \boxed{-x_i \times \Delta_j}
\end{aligned}
$$

a_k =

hidden units

where $\Delta_j = h'(a_j)\sum_k w_{k,j}\Delta_k$

Backprop: Hidden Layer Update Equations

- $\frac{\partial E}{\partial w_{j,i}} = -x_i \times \Delta_j$
- Use gradients in gradient descent: $w_{j,i} \leftarrow w_{j,i} - \alpha \frac{\partial E}{\partial w_{j,i}}$
- Backpropagate error from the output layer to hidden layer:

$$w_{j,i} \leftarrow w_{j,i} + \alpha \times x_i \times \Delta_j$$

where $\Delta_j = h'(a_j) \sum_k w_{k,j} \Delta_k$

Challenges in Deep Learning Optimization

- Local minima, weight space symmetry
- Plateaus, saddle points, flat regions
- Cliffs and exploding gradients
- Long term dependence
- Inexact gradients
- Choice of step size
- Scalability, first-order and second-order methods

Cliffs and Exploding Gradients

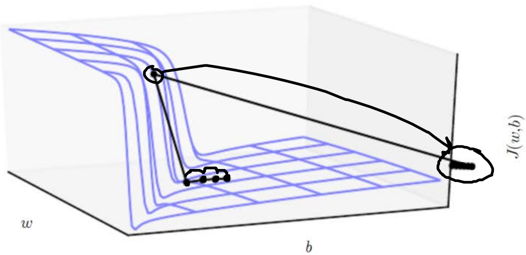


Figure 8.3: The objective function for highly nonlinear deep neural networks or for recurrent neural networks often contains sharp nonlinearities in parameter space resulting from the multiplication of several parameters. These nonlinearities give rise to very high derivatives in some places. When the parameters get close to such a cliff region, a gradient descent update can catapult the parameters very far, possibly losing most of the optimization work that had been done. Figure adapted with permission from Pascanu *et al.* (2013).

Exploding and Vanishing Gradients

- For an L layer neural network

$$\begin{aligned} f &= f_L \circ f_{L-1} \ldots \circ f_1 \\ f(x) &= f_L(f_{L-1}(\cdots f_1(x) \cdots)) \end{aligned}$$

- Gradient (Jacobian) of f with respect input x is the product

$$f' = f'_L f'_{L-1} \ldots f'_1$$

- Consequence: Exploding or vanishing gradients

Optimization

data

- Mini-batch updates: Sample m examples $\{x_i\}$

$$\hat{g}_t \leftarrow \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i; \theta_t), y_i)$$

$1 < m << n$

$$\theta_{t+1} \leftarrow \theta_t - \eta_t \hat{g}_t$$

$O\left(\frac{m}{\varepsilon}\right)$

- Momentum methods
- Adaptive gradient descent
- Combinations of such ideas

Momentum Method

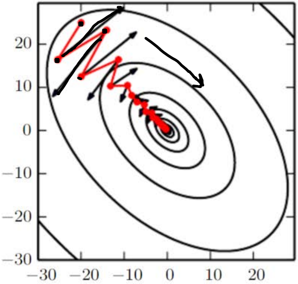


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also figure 4.6, which shows the behavior of gradient descent without momentum.

SGD with Momentum

- Velocity variable v
  - Direction + ~~velocity~~ speed at which the parameters move through space
  - Dynamical system perspective
- SGD/Mini-batch update with momentum

$$\hat{g}_t \leftarrow \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i; \theta_t), y_i)$$

$$v_{t+1} \leftarrow \alpha_t v_t - \eta_t \hat{g}_t$$

$$\theta_{t+1} \leftarrow \theta_t + v_{t+1}$$

$$\theta_{t+1} = \theta_t - \eta_t \hat{g}_t$$

- Parameter $\alpha \in [0, 1)$ determines decay rate of gradient
- Dynamics of α: start small, increase over iterations

SGD with Nesterov Momentum

- Inspired by Nesterov's accelerated gradient descent
- Evaluate gradient at $\theta_t + v_t$ instead of at θ_t
- SGD with Nesterov momentum

$$
\begin{aligned}
\tilde{\theta}_t &\leftarrow \theta_t + \alpha_t v_t \\
\hat{g}_t &\leftarrow \nabla_{\tilde{\theta}} \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i; \tilde{\theta}_t), y_i) \\
v_{t+1} &\leftarrow \alpha_t v_t - \eta_t \hat{g}_t \\
\theta_{t+1} &\leftarrow \theta_t + v_{t+1}
\end{aligned}
$$

Adaptive Learning Rate

- Adapt the learning rate over t

$$\theta_{t+1} = \theta_t - \underline{\eta_t} \nabla \ell(\theta_t)$$

init. LR

- Basic GD/SGD have decreasing learning rate: $\eta_t = \frac{\eta_0}{\sqrt{\gamma t}}$, $\eta_t = \frac{\eta_0}{\gamma t}$, etc.
- Limitation: Exact same learning rate across all dimensions, slow convergence
- Goal: different/adaptive learning rates for different parameters

Adaptive Learning Rate: Adagrad

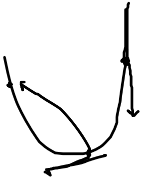
- Gradient of the ith parameter θ^i at iteration t:

$$g_t^i = \nabla_{\theta^i} \ell(\theta_t)$$

$O\left(\frac{1}{\sqrt{t}}\right)$

- The adaptive learning rate used by Adagrad

$$\eta_t^i = \frac{\eta_0}{\sqrt{\sum_{s=0}^{t} (g_s^i)^2}}$$



- SGD/minibatch with Adagrad

$$\hat{g}_t \leftarrow \nabla_{\tilde{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}_i; \tilde{\theta}_t), y_i)$$

$$r_t \leftarrow r_{t-1} + \hat{g}_t \odot \hat{g}_t$$

small value

$$\Delta\theta \leftarrow -\frac{\eta_0}{\delta + \sqrt{r_t}} \odot \hat{g}_t$$

$$\theta_{t+1} \leftarrow \theta_t + \Delta\theta$$

Adagrad: Advantages

- Parameters with large gradients $\Rightarrow$ small learning rate and
  small gradients $\Rightarrow$ large learning rate.
- Suitable for deep learning
  - Scale of the gradient for different θ^i often different by several orders of magnitude
  - Adjust step size based on scale
- Empirical results on large-scale problems are good

Adagrad: Disadvantages

- If initial gradients are large, subsequent training will be slow
- Learning rate monotonically decreases
  - May lead to early training termination
  - Maybe problematic for deep learning, e.g., exploding gradients
- Hyper-parameter η_0 needs to be chosen

Adaptive Learning Rate: RMSprop

- Need to recover (increase) step size over 'flat' regions
- Uses an exponentially decaying average for r (squared gradient)
- RMSProp algorithm

momentum

adaptive LR

$$
\begin{aligned}
\hat{g}_t &\leftarrow \nabla_{\tilde{\theta}} \frac{1}{m} \sum_{i=1}^{m} \ell(f(\mathbf{x}_i; \tilde{\theta}_t), y_i) \\
r_t &\leftarrow \rho r_{t-1} + (1-\rho)\hat{g}_t \odot \hat{g}_t \\
\Delta\theta &\leftarrow -\frac{\eta_0}{\sqrt{\delta + r_t}} \odot \hat{g}_t \\
\theta_{t+1} &\leftarrow \theta_t + \Delta\theta
\end{aligned}
$$

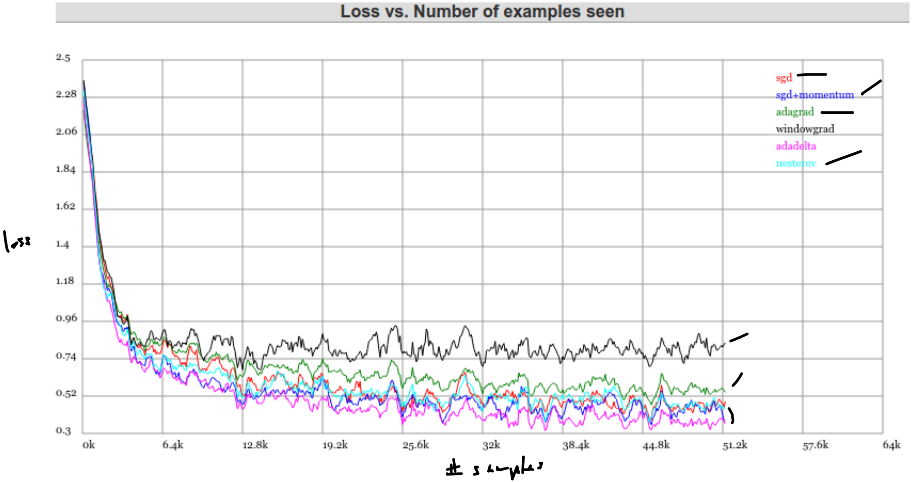
1e-5

- Extension to Nesterov momentum: gradient at $\tilde{\theta}_t = \theta_t + \alpha_t v_t$

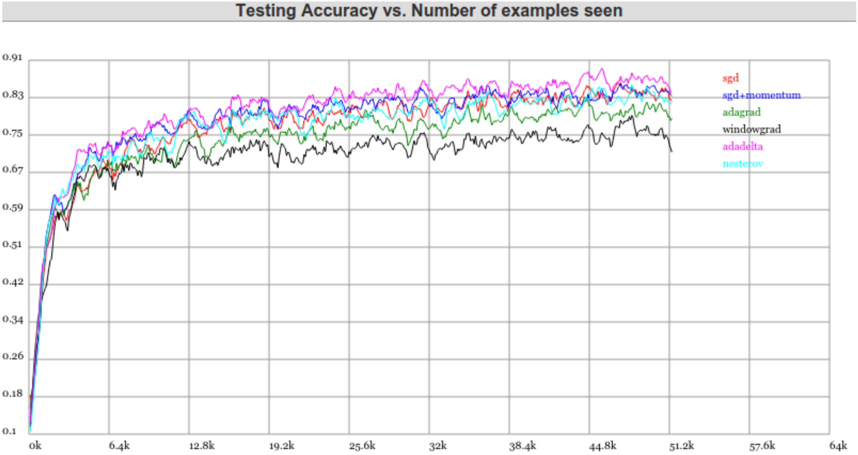
- Idea: RMSprop + first and second order momentum + bias correction
- Adam algorithm

$$
\begin{aligned}
\hat{g}_t &\leftarrow \nabla_{\tilde{\theta}} \frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}_i; \tilde{\theta}_t), y_i) \\
s_t &\leftarrow \rho_1 s_{t-1} + (1-\rho_1)\hat{g}_t \\
r_t &\leftarrow \rho_2 r_{t-1} + (1-\rho_2)\hat{g}_t \odot \hat{g}_t \\
\hat{s}_t &\leftarrow \frac{s_t}{1-\rho_1^t} \\
\hat{r}_t &\leftarrow \frac{r_t}{1-\rho_2^t} \\
\Delta\theta &\leftarrow -\frac{\eta_0}{\delta + \sqrt{\hat{r}_t}} \odot \hat{s}_t \\
\theta_{t+1} &\leftarrow \theta_t + \Delta\theta
\end{aligned}
$$

Algorithm Comparison on MNIST Dataset



Algorithm Comparison on MNIST Dataset



Algorithm Comparison on MNIST Dataset