Neural Network Optimization 2

CS 519 Deep Learning, Winter 2016

Fuxin Li
Tricks of the trade

• As you might have seen, it’s a bit difficult to make NN work well!
• Many tricks needed
• Some are optimization-related
  • Momentum
  • RMSprop
  • Adagrad
  • Adam, etc.
• Some are regularization-related
  • Dropout
  • Batch normalization
  • Particular regularizing network structures, etc.
Nesterov Momentum vs. Momentum

- Perform the “inertia” step first before taking gradient
- Better theoretical guarantees in convex optimization

Normal momentum

\[
\begin{align*}
D_0 &= 0 \\
D_{t+1} &= \mu D_t - \alpha \nabla f (W_t) \\
W_{t+1} &= W_t + D_{t+1}
\end{align*}
\]

Nesterov momentum

\[
\begin{align*}
D_0 &= 0 \\
D_{t+1} &= \mu D_t - \alpha \nabla f (W_t + \mu D_t) \\
W_{t+1} &= W_t + D_{t+1}
\end{align*}
\]

Nesterov Momentum vs. Momentum

- **N**: Nesterov
- **M**: Momentum
- **Number**: $\mu_{max}$
- **Schedule**: 

\[
\mu_t = \min(1 - 2^{-1-\log_2(|t/250|+1)}, \mu_{max})
\]

<table>
<thead>
<tr>
<th>task</th>
<th>0 (SGD)</th>
<th>0.9N</th>
<th>0.99N</th>
<th>0.995N</th>
<th>0.999N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curves</td>
<td>0.48</td>
<td>0.16</td>
<td>0.096</td>
<td>0.091</td>
<td>0.074</td>
</tr>
<tr>
<td>Mnist</td>
<td>2.1</td>
<td>1.0</td>
<td>0.73</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>Faces</td>
<td>36.4</td>
<td>14.2</td>
<td>8.5</td>
<td>7.8</td>
<td>7.7</td>
</tr>
</tbody>
</table>

Trick #2: Adaptive learning rates

• Recall, learning rate should go to zero for convergence
• Is fixed learning rate a great idea?
• Simple learning rate decay:
  • Reduce the learning rate after every few epochs
  • Seems to work relatively well if all the weights are quite balanced
The intuition behind separate adaptive learning rates

• Learning rates can be set per weight or layer
  • Gradient magnitudes differ across layers
  • Fan-in of a unit determines the size of the “overshoot” effects

• Use a global learning rate times local gain

Gradients can get very small in the early layers of very deep nets.

The fan-in often varies widely between layers.
AdaGrad (Duchi 2011)

• If the gradient is large, stepsize should be small
• Use square-root of accumulated gradient norm to be step size

\[
\alpha_k = \frac{\alpha_0}{\sqrt{r_{Tk}}}
\]

\[
r_{Tk} = \sum_{i=1}^{T} \|G_{ik}\|^2 = r_{T-1,k} + \|G_{ik}\|^2
\]

Try:

\[
\min_{w_1,w_2} (w_1 - 1)^2 + 100(w_2 - 1)^2
\]
AdaGrad

**Algorithm 8.4** The AdaGrad algorithm

**Require:** Global learning rate $\epsilon$

**Require:** Initial parameter $\theta$

**Require:** Small constant $\delta$, perhaps $10^{-7}$, for numerical stability

Initialize gradient accumulation variable $r = 0$

**while** stopping criterion not met **do**

Sample a minibatch of $m$ examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.

Compute gradient: $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i} L(f(x^{(i)}; \theta), y^{(i)})$

Accumulate squared gradient: $r \leftarrow r + g \odot g$

Compute update: $\Delta \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$. (Division and square root applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta \theta$

**end while**
RMSprop

• AdaGrad does not work well for non-convex problems
• Remembered too much history
• In stochastic non-convex optimization, history might be not very useful

$$r_{T_k} = \rho r_{T-1,k} + (1 - \rho) \|G_{T_k}\|^2$$

e.g. $\rho = 0.9$
RMSprop

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\theta$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.
Initialize accumulation variables $r = 0$

while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $g \leftarrow \frac{1}{m} \nabla \theta \sum_{i} L(f(x^{(i)}; \theta), y^{(i)})$
    Accumulate squared gradient: $r \leftarrow \rho r + (1 - \rho) g \odot g$
    Compute parameter update: $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$. ($\frac{1}{\sqrt{\delta + r}}$ applied element-wise)
    Apply update: $\theta \leftarrow \theta + \Delta \theta$
end while
Combining RMSProp with Nesterov Momentum (Sutskever 2013)

- Use RMSprop on the gradient part of the momentum
- Somehow it doesn’t seem to work well with normal momentum

\[
D_0 = 0 \\
D_{t+1} = \mu D_t - \alpha \nabla f(W_t + \mu D_t) \\
W_{t+1} = W_t + D_{t+1}
\]
Adam

- Running average of momentum estimates and RMS estimates are both biased
- Take RMSprop estimate at stationary state:

\[
r_{T+k} = \rho r_{T-1,k} + (1 - \rho) \|G_{T,k}\|^2
\]

\[
= (1 - \rho) \sum_{i=1}^{T} \rho^{T-i} \|G_{T,k}\|^2
\]

\[
E(r_{T,k}) = E(\|G_k\|^2) (1 - \rho) \sum_{i=1}^{T} \rho^{T-i}
\]

\[
= E(\|G_k\|^2) (1 - \rho^T)
\]
Adam

- Robust choice of step size
- Two moment estimates:
  - Bias-correcting momentum (T here is not transpose, is T-th power!), e.g. $\rho_1 = 0.99$
    \[ D_T = \frac{\rho_1}{1 - \rho_1^T} D_{T-1} + \frac{1 - \rho_1}{1 - \rho_1^T} G_T \]
  - Bias-correcting RMS estimate (average gradient norm), e.g. $\rho_2 = 0.999$
    \[ r_{Tk} = \frac{\rho_2}{1 - \rho_2^T} r_{T-1,k} + \frac{1 - \rho_2}{1 - \rho_2^T} \|G_{Tk}\|^2 \]
  - Final update:
    \[ W_k = W_k - \epsilon \frac{D_{Tk}}{\sqrt{r_{Tk}}} \]
Polyak averaging

• $\overline{W}_T = \alpha \overline{W}_{T-1} + (1 - \alpha) W_T$

• Sometimes used only at the end of optimization to create a “momentum”-like effect for the final model
When to use these things?

• Adam is usually good if you want to avoid tuning learning rate
• However, sometimes fixed learning rate + some manual decreases work just fine
• RMSprop + Nesterov momentum also works well
• Usually Adagrad is not used in deep learning
• Polyak averaging is a bit redundant with momentum, hence mostly used at the end (testing time)