

کنسلی برای امور کارائی یادگیری

مستنی بر پایه یادابی کا حسی مرادمان

« زخ یادگیری تصبیعی »

Topics

- Importance of Optimization in machine learning
 1. How learning differs from optimization
 2. Challenges in neural network optimization
 3. Basic Optimization Algorithms
 - SGD, Momentum, Nesterov Momentum
 4. Parameter initialization strategies
 5. Algorithms with adaptive learning rates
 1. AdaGrad
 2. RMSProp
 3. Adam
 4. Choosing the right optimization algorithm
 6. Approximate second-order methods
 7. Optimization strategies and meta-algorithms

Learning Rate is Crucial

- Learning rate: most difficult hyperparam to set
- It significantly affects model performance
- Cost is highly sensitive to some directions in parameter space and insensitive to others

– Momentum helps but introduces another hyperparameter

– Is there another way?

- If direction of sensitivity is axis aligned, separate learning rate for each parameter and adjust them throughout learning

$$g = \left\langle \frac{\partial \text{loss}}{\partial w_i} \right\rangle \quad w = w - \eta \cdot g \quad i = 1, \dots, N$$

Heuristic Approaches

• Delta-bar-delta Algorithm

– Applicable to only full batch optimization (GD)

– Method:

$$w_i = w_i - \eta_0 \text{sign}(g_i)$$

- If partial derivative of the loss wrt to a parameter remains the same sign, the learning rate should increase
- If that partial derivative changes sign, the learning rate should decrease

• Recent Incremental mini-batch methods (SGD)

– To adapt learning rates of model parameters

1. AdaGrad
2. RMSProp
3. Adam





Keras Adaptive Optimizers

- RMSprop
- Adagrad
- Adadelta
- Adam
- Adamax
- Nadam

```
keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9)
```

```
keras.optimizers.Adagrad(learning_rate=0.01)
```

```
keras.optimizers.Adadelta(learning_rate=1.0, rho=0.95)
```

```
keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
```

```
keras.optimizers.Adamax(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
```

```
keras.optimizers.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
```

AdaGrad

- Individually adapts learning rates of all params
 - By scaling them inversely proportional to the sum of the historical squared values of the gradient

- The AdaGrad Algorithm:

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , 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 $\{x^{(1)}, \dots, x^{(m)}\}$ 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

$$\eta_i^{(t)} = \frac{1 \epsilon}{(g_i^{(1)2} + g_i^{(2)2} + \dots + g_i^{(t)2})^{1/2} + \delta}$$

$$\eta_i^{(t)} \xrightarrow{t \rightarrow \infty} 0 \quad ?$$

$$\sum_{t=1}^{\infty} \eta_i^{(t)} = \infty \quad ?$$

ضرب مولفه به مولفه $\langle g_1^2, \dots, g_i^2, \dots, g_n^2 \rangle$

Performs well for some but not all deep learning

RMSProp

- Modifies AdaGrad for a nonconvex setting
 - Change gradient accumulation into exponentially weighted moving average
 - Converges rapidly when applied to convex function

The RMSProp Algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , 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 $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

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

Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

Compute parameter update: $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta + \mathbf{r}}}$ applied element-wise)

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

end while

$$r^{(T)} = (1-\rho) \sum_{t=1}^T \rho^{t-1} g^{(t)2}$$

RMSProp combined with Nesterov

Algorithm: RMSProp with Nesterov momentum

Require: Global learning rate ϵ , decay rate ρ , momentum coefficient α .
Require: Initial parameter θ , initial velocity v , $v=0$
Initialize accumulation variable $r = 0$,
while stopping criterion not met **do**
 Sample a minibatch of m examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
 Compute interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$
 Compute gradient: $g \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$
 Accumulate gradient: $r \leftarrow \rho r + (1 - \rho) g \odot g$
 Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot g$. ($\frac{1}{\sqrt{r}}$ applied element-wise)
 Apply update: $\theta \leftarrow \theta + v$
end while



RMSProp is popular

- RMSProp is an effective practical optimization algorithm
- Go-to optimization method for deep learning practitioners



Adam: Adaptive Moments

- Yet another adaptive learning rate optimization algorithm
- Variant of RMSProp with momentum
- Generally robust to the choice of hyperparameters



The Adam Optimizer

The Adam Algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $\underline{s = 0}, \underline{r = 0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

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

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

$t \leftarrow t + 1$

 Update biased first moment estimate: $\underline{s} \leftarrow \rho_1 \underline{s} + (1 - \rho_1) \underline{g}$

 Update biased second moment estimate: $\underline{r} \leftarrow \rho_2 \underline{r} + (1 - \rho_2) \underline{g} \odot \underline{g}$

 Correct bias in first moment: $\hat{\underline{s}} \leftarrow \frac{\underline{s}}{1 - \rho_1^t}$

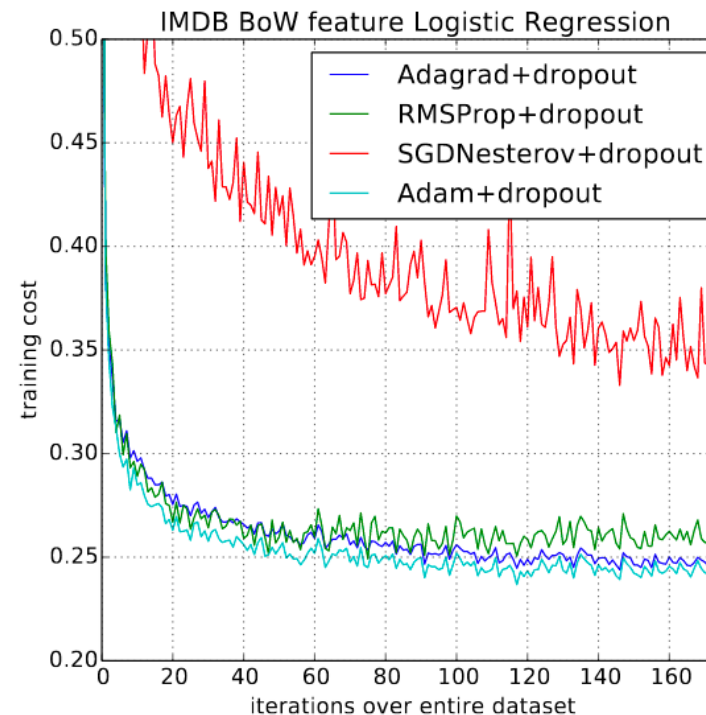
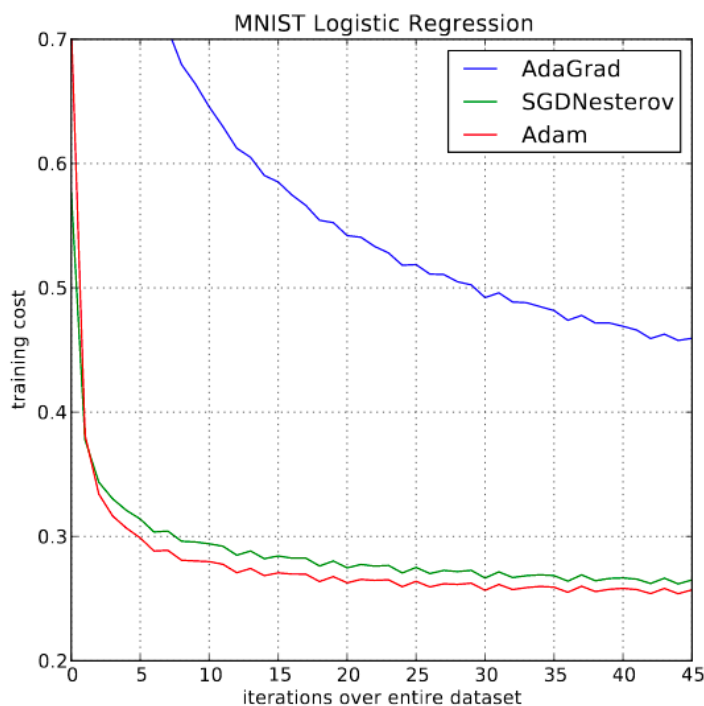
 Correct bias in second moment: $\hat{\underline{r}} \leftarrow \frac{\underline{r}}{1 - \rho_2^t}$

 Compute update: $\Delta \theta = -\epsilon \frac{\hat{\underline{s}}}{\sqrt{\hat{\underline{r}} + \delta}}$ (operations applied element-wise)

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

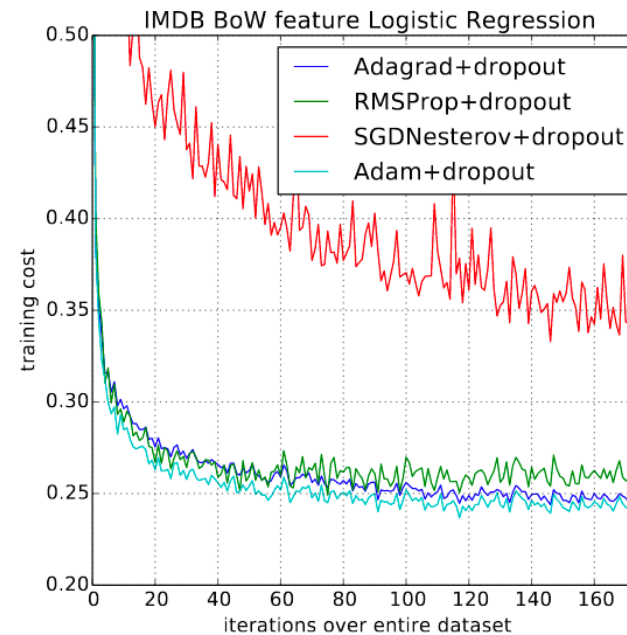
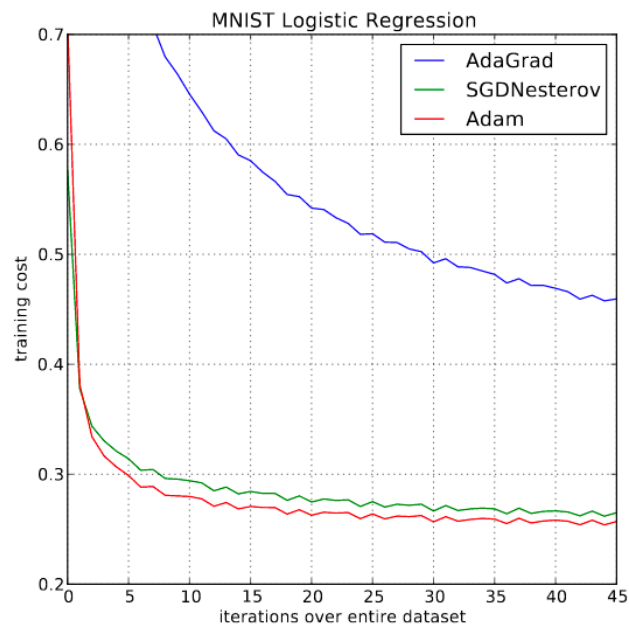
end while

Performance in Logistic Regression



Logistic regression training negative log likelihood on MNIST images and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors.

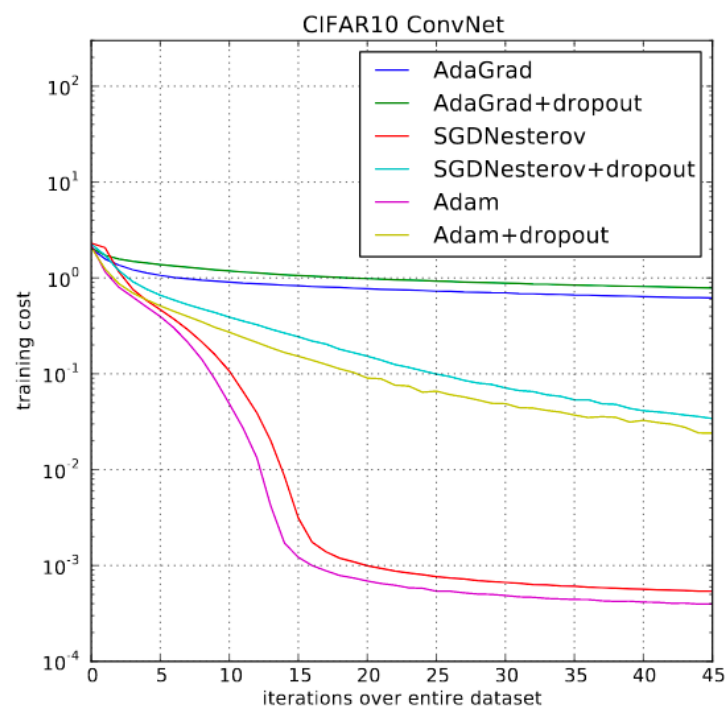
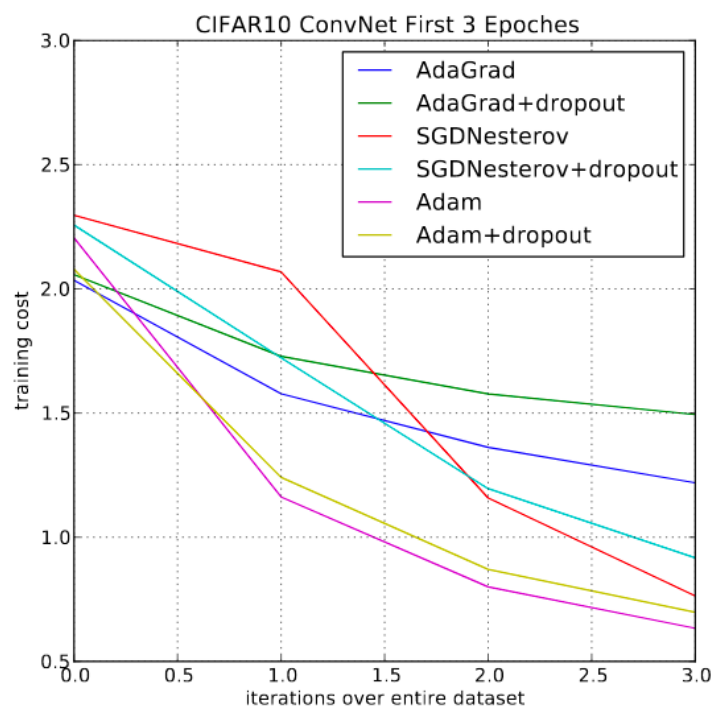
Performance on Multilayer NN



Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function.



Performance with CNN



Convolutional neural networks training cost.

(left) Training cost for the first three epochs.

(right) Training cost over 45 epochs.

CIFAR-10 with c64-c64-c128-1000 architecture.



Choosing the Right Optimizer

- We have discussed several methods of optimizing deep models by adapting the learning rate for each model parameter
- Which algorithm to choose?
 - There is no consensus
- Most popular algorithms actively in use:
 - SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam
 - Choice depends on user's familiarity with algorithm