مرکز آموزش‌های الکترونیکی دانشگاه شهید بهشتی

شبکه‌های عصبی مصنوعی

Artificial Neural Networks

Ali Katanforoush*

** Department of Data and Computer Sciences, Shahid Beheshti University*

تکنیک‌هایی برای بهبود کارآیی یادگیری

مبتنی بر بسته یابی کاهش گرادیان

«بسته یابی با استفاده از لکانه»

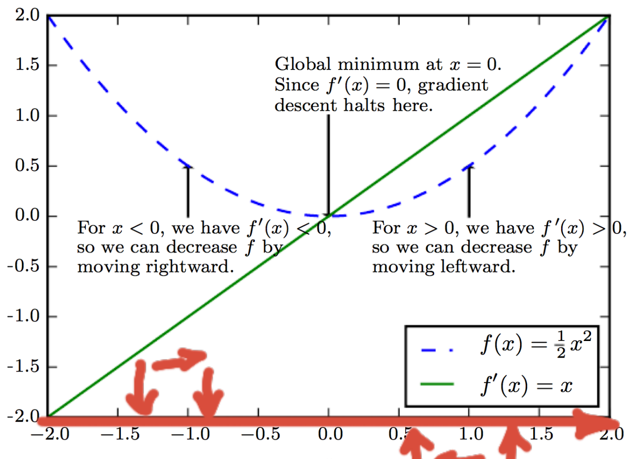
Topics

- Importance of Optimization in machine learning
- How learning differs from optimization
- Challenges in neural network optimization
- Basic Optimization Algorithms
  - SGD, Momentum, Nesterov Momentum
- Parameter initialization strategies
- Algorithms with adaptive learning rates
  - AdaGrad, RMSProp, Adam
- Approximate second-order methods
- Optimization strategies and meta-algorithms

1. Stochastic Gradient Descent

- Gradient descent follows gradient of entire training set downhill

$$\text{loss}(\text{mbatch}) = \sum_{\text{sample} \in \text{minibatch}} \text{loss}(\text{sample})$$



Criterion $f(x)$ minimized by moving from current solution in direction of the negative of gradient

$$w^{new} = w^{old} - \eta \nabla_w \text{loss}(\text{minibatch})$$

- $\boxed{\text{SGD:}}$ Accelerated by minibatches downhill
  - Wide use for ML in general and for deep learning
  - Average gradient on a minibatch is an estimate of the gradient

SGD follows gradient estimate downhill

Algorithm: SGD update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
end while

تابع زیان

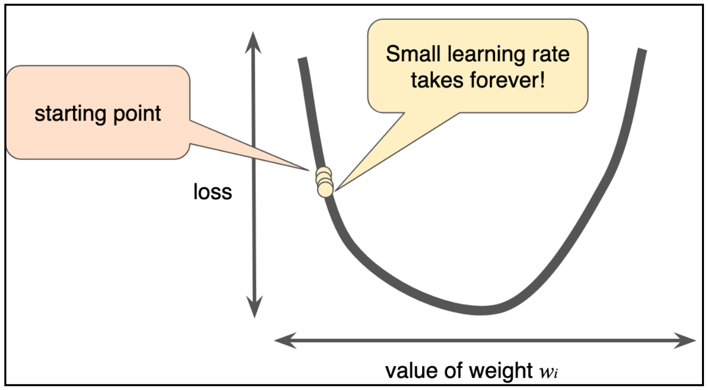
Desired output

Network output

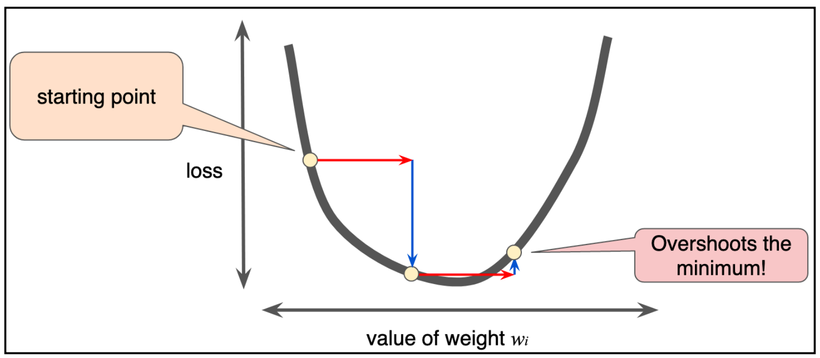
A crucial parameter is the learning rate ε

At iteration k it is ε_k

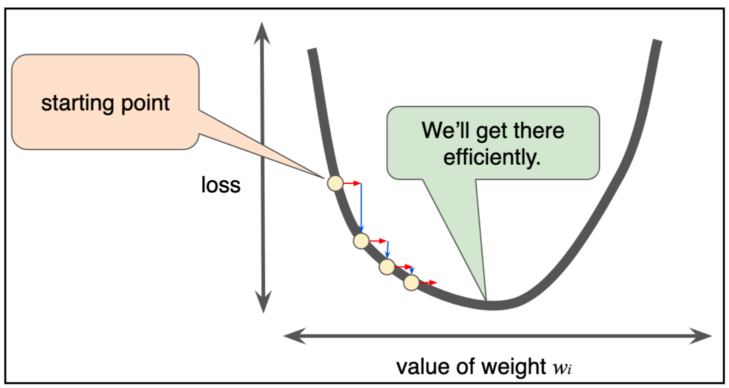
Choice of learning rate



Too small learning rate will take too long



Too large, the next point will perpetually bounce haphazardly across the bottom of the well



If gradient is small then you can safely try a larger learning rate, which compensates for the small gradient and results in a larger step size

Learning rate in Keras

- Keras provides SGD class to implement SGD optimizer with learning rate and momentum
  - The default learning rate is 0.01 and no momentum used

```
from keras.optimizers import SGD
...
opt = SGD()
model.compile(..., optimizer=opt)
```

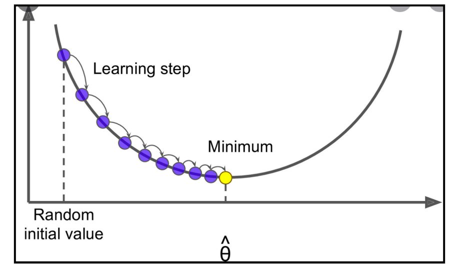
Need for decreasing learning rate

- True gradient of total cost function
  - Becomes small and then 0
    - We can use a fixed learning rate
- But SGD has a source of noise
  - Random sampling of m training samples
    - Gradient does not vanish even when arrive at a minimum
  - Sufficient condition for SGD convergence

$\lim_{k\to\infty} \varepsilon_k = 0$ AND $\boxed{\sum_{k=1}^{\infty} \varepsilon_k = \infty \text{ AND } \sum_{k=1}^{\infty} \varepsilon_k^2 < \infty}$

  - Common to decay learning rate linearly until iteration τ: $\varepsilon_k = (1-\alpha)\varepsilon_0 + \alpha\varepsilon_\tau$ with $\alpha = k/\tau$
  - After iteration τ, it is common to leave ε constant
    - Often a small positive value in the range 0.0 to 1.0

Learning Rate Decay



- Decay learning rate

  τ: $\varepsilon_k=(1-\alpha)\varepsilon_0+\alpha\varepsilon_\tau$ with $\alpha=k/\tau$

- Learning rate is calculated at each update
  - (e.g. end of each mini-batch) as follows:

```
lrate = initial_lrate * (1 / (1 + decay * iteration))
```

  - Where *lrate* is learning rate for current epoch
  - *initial_lrate* is specified as an argument to SGD
  - *decay* is the decay rate which is greater than zero and
  - *iteration* is the current update number

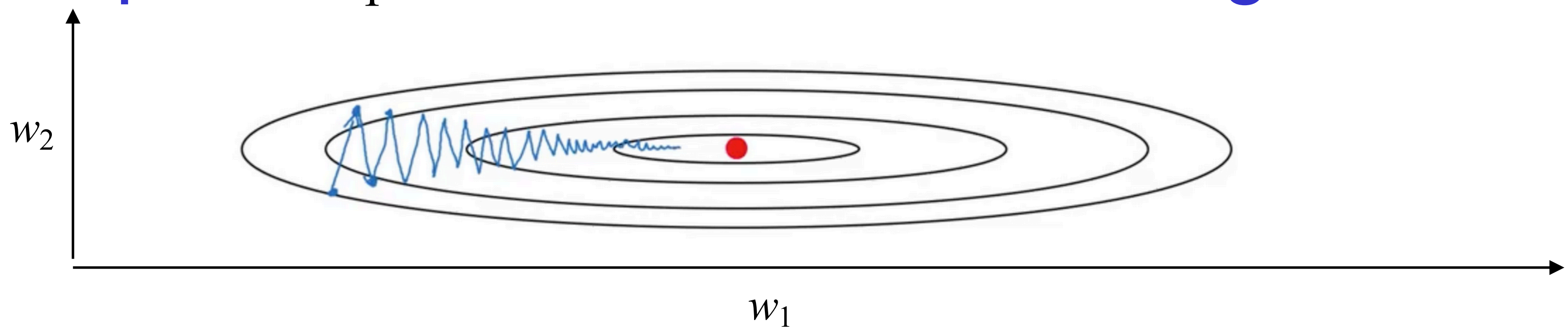
```
from keras.optimizers import SGD
...
opt = SGD(lr=0.01, momentum=0.9, decay=0.01)
model.compile(..., optimizer=opt)
```

2. The Momentum method

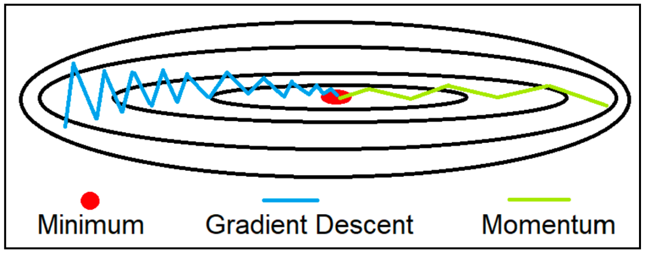
- SGD is a popular optimization strategy
- But it can be slow
- Momentum method accelerates learning, when:
  - Facing high curvature
  - Small but consistent gradients
  - Noisy gradients
- Algorithm accumulates moving average of past gradients and move in that direction, while exponentially decaying

Gradient Descent with momentum

- Gradient descent with momentum converges faster than standard gradient descent
- Taking large steps in w_2 direction and small steps in w_1 direction slows down algorithm



- Momentum reduces oscillation in w_2 direction



- We can set a higher learning rate

Momentum definition

- Introduce variable $\boldsymbol{v}$, or velocity
- It is the direction and speed at which parameters move through parameter space
- Momentum in physics is mass times velocity
- The momentum algorithm assumes unit mass
- A hyperparameter $\alpha \; \varepsilon \; [0,1)$ determines exponential decay

Momentum update rule

- The update rule is given by

$$\boxed{\begin{aligned} &\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \varepsilon \nabla_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m} L\left(f(x^{(i)};\theta), y^{(i)}\right)\right) \\ &\theta \leftarrow \theta + \boldsymbol{v} \end{aligned}}$$

- The velocity $\boldsymbol{v}$ accumulates the gradient elements $\boxed{\nabla_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m} L\left(f(x^{(i)};\theta), y^{(i)}\right)\right)}$
- The larger α is relative to ε, the more previous gradients affect the current direction
- The SGD algorithm with momentum is next

SGD algorithm with momentum

Algorithm: SGD with momentum

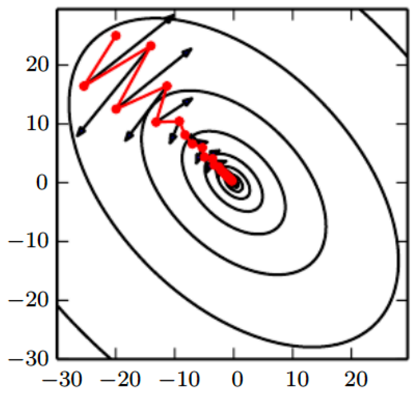
Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Keras: The learning rate can be specified via the *lr* argument and the momentum can be specified via the *momentum* argument.

```
from keras.optimizers import SGD
...
opt = SGD(lr=0.01, momentum=0.9)
model.compile(..., optimizer=opt)
```

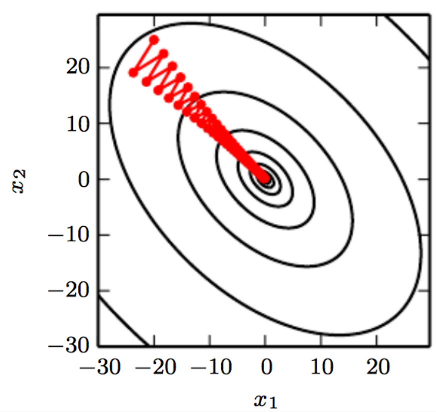
Momentum

- SGD with momentum



Contour lines depict a quadratic loss function
With a poorly conditioned Hessian matrix
Red path cutting across the contours depicts path followed by momentum learning rule as it minimizes this function

- Comparison to SGD without momentum



At each step we show path that would be taken by SGD at that step
Poorly conditioned quadratic objective
Looks like a long narrow valley
with steep sides
Wastes time

3. Nesterov Momentum

- A variant to accelerate gradient, with update

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \underbrace{\boldsymbol{\theta} + \alpha \boldsymbol{v}}_{\theta'}), \boldsymbol{y}^{(i)} \right) \right],$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

  - where parameters α and ε play a similar role as in the standard momentum method

– Difference between Nesterov and standard momentum is where gradient is evaluated.

  - Nesterov gradient is evaluated after the current velocity is applied.
  - Thus one can interpret Nesterov as attempting to add a correction factor to the standard method of momentum

SGD with Nesterov Momentum

- A variant of the momentum algorithm
  - Nesterov's accelerated gradient method
- Applies a correction factor to standard method

Algorithm: SGD with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

This line is added from plain momentum

Keras SGD with Nesterov Momentum

- An optimizer is one of the two arguments required for compiling a Keras model:

```
from keras import optimizers

model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

Arguments
learning_rate: float >= 0. Learning rate.
momentum: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
nesterov: boolean. Whether to apply Nesterov momentum.