Training Neural Networks

Part 2

Hongzhi Liu

November 22nd, 2019

CONTENT

1. Gradient Descent (GD)

A one-dimensional Taylor series is an expansion of a real function about a point x=a is given by

$$f(x)=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n$$

with Peano remainder

$$f(x+\epsilon)\approx f(x)+f'(x)\epsilon+\mathcal{O}(\epsilon^2)$$

$$f(x+\epsilon)\approx f(x)+f'(x)\epsilon$$

$$f(x-\eta f'(x))\approx f(x)-\eta f'(x)^2 \quad \eta>0$$

$$x\leftarrow x-\eta f'(x)$$

2. Stochastic Gradient Descent (SGD)

SGD

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

SGD + Nesterov Momentum

```
v = 0
while True:
    dx = compute_gradient(x)
    old_v = v
    v = rho * v - learning_rate * dx
    x += -rho * old_v + (1 + rho) * v
```

Sutskever et al. On the importance of initialization and momentum in deep learning. In *ICML*, 2013
Nesterov. Introductory lectures on convex optimization: A basic course. 2004

3. AdaGrad and RMSProp

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7))
```

RMSProp: “Leaky AdaGrad”

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7))
```

4. Adam

Full form

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    grad_squared += decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Learning Rate Decay

Linear $$\alpha_t = \alpha_0(1 - t/T)$$

```
for t in range(1, epoch):
    learning_rate = learning_rate * (1 - t / epoch)
```

Inverse sqrt $$\alpha_t = \alpha_0/\sqrt{t}$$

```
for t in range(1, epoch):
    learning_rate = learning_rate / (sqrt(t))
```

Cosine $$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

```
for t in range(1, epoch):
    learning_rate = 0.5 * learning_rate * (1 + cos(pi * t / epoch))
```

Regularization

1. Dropout

```
p = ratio
def train_step(X):
    H_1 = np.maximum(0,np.dot(W1,X)+b_1)
    U_1 = np.random.rand(*H_1.shape) < p
    H_1 *= U_1
    H_2 = np.maximum(0,np.dot(W2,X)+b_2)
    U_2 = np.random.rand(*H_2.shape) < p
    H_2 *= U_2
    out = np.dot(W3, H2) + b3
```

Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

2. Batch Normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

1. It accelerates the training of deep neural nets.

2. It also acts as a regularizer, in some cases eliminating the need for Dropout.

3. It is helpful to reduce the over-fitting of the network and improve the accuracy rate as a whole.

Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.. In *ICML*, 2015.

3. Data Augmentation

Horizontal Flips

```
class torchvision.transforms.RandomHorizontalFlip(p=0.5)
```

Random crops

```
class torchvision.transforms.RandomCrop(size, padding=None,
                pad_if_needed=False, fill=0, padding_mode='constant')
```

Scales

```
class torchvision.transforms.Scale(size, interpolation=2)
```

ColorJitter

```
class torchvision.transforms.ColorJitter(brightness=0, contrast=0,
                saturation=0, hue=0)
```

https://pytorch.org/docs/stable/torchvision/transforms.html

Thanks