Neural Network Basics

0100_kNN

- What is Image Classification
- Nearest Neighbor Classifier
- Validation Set
  - Hyperparam tuning
  - Cross validation

0200_Linear CLassification

- Overview
- 线性分类器的直观理解
- Loss
  - SVM Loss（hinge loss)
  - Regularization
  - Softmax (cross entropy loss)
    - 为了计算的稳定性，通常先减去最大值

0300_Optimization

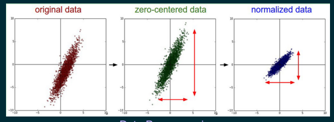
- Strategy
  - 1. Random Search
  - 2. Random Local Search
  - 3. Gradient
- Compute the gradient
  - Numerical
    - 用于梯度检查 — 用相对误差比较两种梯度
    - centered difference formula: $\frac{df(x)}{dx} = \frac{f(x+h)-f(x-h)}{2h}$
  - Analytical
- Gradient descent
  - Vanilla version（单样本更新)
  - Mini-batch

0400_Backpropagation

0500_NN1

- Intro to Neural Network
- Activation functions
  - Sigmoid
  - Tanh
  - ReLU
  - Leaky ReLU
- Neural Network architectures

0600_NN2



- Data Preprocessing
  - Mean subtraction — X -= np.mean(X, axis=0)
  - Normalization — 先中心化，然后除以标准差
  - PCA、Whitening — 神经网络中不用
- Weight Initialization
  - w = np.random.randn(n) * sqrt(2.0/n)
  - b = 0
- Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Regularization
  - L2 regularization — $\frac{1}{2}\lambda W^2$
  - L1 regularization — $\lambda|W|$
  - Dropout

```
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask.
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask.
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3
```

例如p=0.5，则U1和H1与之前的方法相比都扩大了两倍

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

- 各种Loss

0700_NN3

- Parameter updates — 这些参数更新的方法是对W中所有参数使用同样的学习率

```
# Vanilla update
x += - learning_rate * dx
```

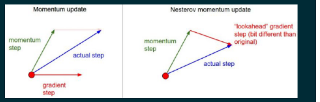
Vanilla update

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Momentum update

```
x_ahead = x + mu * v
# evaluate dx_ahead (the gradient at x_ahead instead of at x)
v = mu * v - learning_rate * dx_ahead
x += v
```

Nesterov Momentum



- Annealing learning rate
- Per-parameter adaptive learning rate

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

Adagrad — 单纯梯度的累加，可能导致学习率下降过快

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

RMSprop — 增加了一个衰减系数，会遗忘一些之前累加的东西

t是迭代次数

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

m_bias = m / (1 - beta1**t)
v_bias = v / (1 - beta2**t)
x += - learning_rate * m_bias / (np.sqrt(v_bias) + eps)

Adam — 与RMSprop相比，把dx换成了一个更平滑的项 m 和 v可以看成是梯度的一阶和二阶矩估计

- Master and worker
- Model ensemble