Softmax Classifier on CIFAR-10 dataset

主讲人：周展科
组员：唐彬 杨澍生 杨阳

华中科技大学
电子信息与通信学院

提纲

1. Softmax Classifier
    1. Implementation Details
    2. Loss Function
    3. Regularization
    4. Gradient Checking
    5. Weight Visulization
2. Improvement
3. Evaluation
4. Reference

1.Softmax Classifier

- Implementation Details
  - Weight Initialization

$$f(x_i, W, b) = Wx_i + b$$

$$f(x_i, W) = Wx_i$$

```
# initialize weight metric W
if self.W is None:
    # C :number of classes
    # D: dimension of each flattened image
    C, D = num_classes, 3072
    self.W = np.random.randn(C, D) * 0.001
```

  - Mini-Batch Gradient Descent

```
idx = np.random.choice(num_train, batch_size)
X_batch = X[idx, :]
y_batch = y[idx]
X_batch = X_batch.T

loss, grad = self.loss(X_batch, y_batch, reg)
loss_history.append(loss)
self.W -= learning_rate * grad
```

1.Softmax Classifier

- **Loss Function**
  - Muti-Class Cross-entropy

Let o_k denote the k-th node of the input layer of the following softmax layer. The calculation of softmax function is given as follows.

$$p_j = \frac{e^{o_j}}{\sum_k e^{o_k}} \tag{1}$$

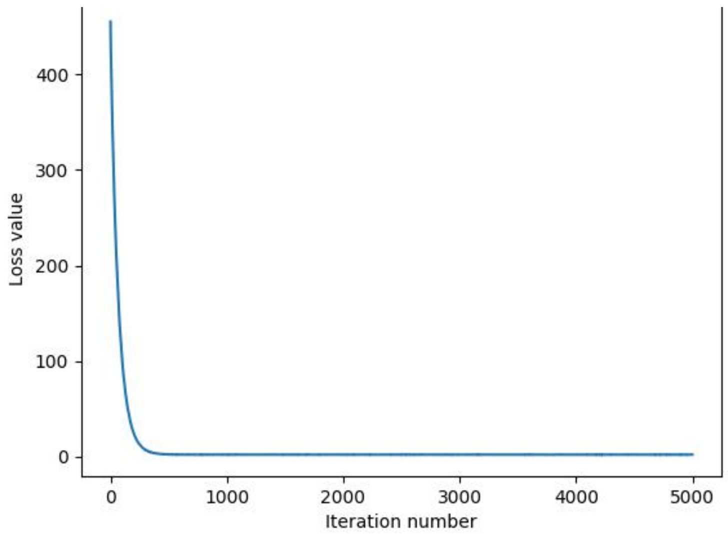
The standard cross entropy loss function L is given as follows.

$$L = -\sum_j y_j \log p_j, \tag{2}$$

Then the derivation of the softmax cross entropy loss is given as follows.

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

```python
# Calculation of loss
z = np.dot(self.W, x)
# Max trick for the softmax, preventing infinite values
z -= np.max(z, axis=0)
# Softmax function
p = np.exp(z) / np.sum(np.exp(z), axis=0)
# Cross-entropy loss
L = -1 / len(y) * np.sum(np.log(p[y, range(len(y))]))
# Regularization term
R = 0.5 * np.sum(np.multiply(self.W, self.W))
# Total loss
loss = L + R * reg
```



1.Softmax Classifier

- Regularization
  - L1 $\alpha||w||_1$
    - Sparse weight matrix
  - L2 $\alpha||w||_2{}^2$
    - Weight Decay
    - Alleviate overfitting

```
# Regularization term
R = 0.5 * np.sum(np.multiply(self.W, self.W))
# Total loss
loss = L + R * reg
self.loss_L_history.append(L)
self.loss_R_history.append(R)
# Calculation of dW
p[y, range(len(y))] -= 1
dW = 1 / len(y) * p.dot(x.T) + reg * self.W
return loss, dW
```

1.Softmax Classifier

- Gradient Check
  - Numerical Gradient

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - Analytical Gradient

$$\begin{aligned}
\frac{\partial L}{\partial o_i} &= -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} \\
&= -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial o_i} \\
&= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i) \\
&= -y_i(1-p_i) + \sum_{k \neq i} y_k(p_i) \\
&= -y_i + y_i p_i + \sum_{k \neq i} y_k(p_i) \\
&= p_i\left(\sum_k y_k\right) - y_i \\
&= p_i - y_i
\end{aligned}$$

Centered Difference Formula: $[f(x+h) - f(x-h)]/2h$

```
x[ix] += h  # Increment by h at dimension index ix
fxph = f(x)  # Evaluate f(x + h)
x[ix] -= 2 * h  # Decrement by h at dimension index ix
fxmh = f(x)  # Evaluate f(x - h)
x[ix] += h  # Reset x

grad_numerical = (fxph - fxmh) / (2 * h)
```

```
==> Gradient Checking:
numerical: -10.482636 analytic: -10.482636, relative error: 1.645301e-11
numerical: -14.302828 analytic: -14.302828, relative error: 3.230006e-09
numerical: -11.691458 analytic: -11.691458, relative error: 2.175477e-11
numerical: -9.659885 analytic: -9.659885, relative error: 3.774794e-11
numerical: -11.953463 analytic: -11.953463, relative error: 4.426308e-09
```

$$\frac{dloss}{dw} = \frac{dloss}{dscores} \cdot \frac{dscores}{dw}$$

The estimation error is given by

$$R = \frac{-f^{(3)}(c)}{6} h^2,$$

Wikipedia: Numerical differentiation

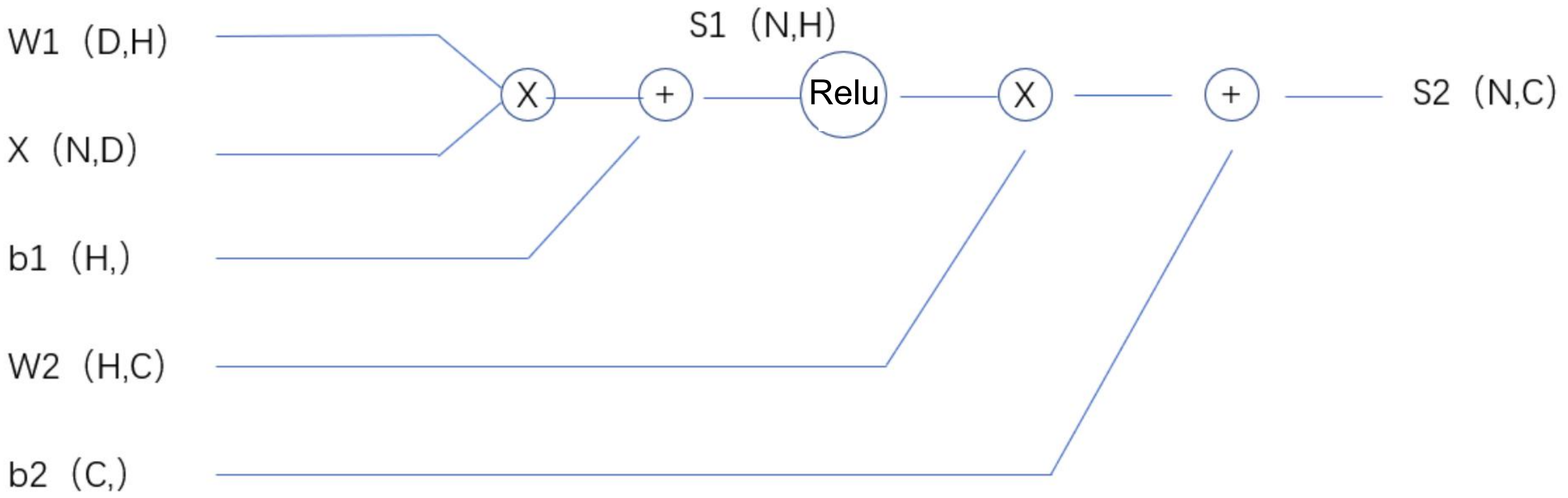
1.Softmax Classifier

- Weight Visulization
  - W.shape: 10*3072
  - Reshape to 10*32*32*3

- Interpretation
  - Classifier works as template matching
  - W corresponds to a template

2.Improvement

- Two-layer Softmax Classifier
  - With two Weight Matrixes (W1 W2)



2.Improvement

- Two-layer Softmax Classifier
  - Loss Calculation

其中$S_1 = XW_1 + b1$，$S_{1relu} = relu(S_1)$，$S_2 = S_{1relu}W_2 + b2$。求$\frac{\partial L}{\partial W_1}$、$\frac{\partial L}{\partial W_2}$、$\frac{\partial L}{\partial b_1}$、$\frac{\partial L}{\partial b_2}$

- $\frac{\partial L}{\partial W_2}$

$$\frac{\partial L}{\partial W_2} = S_{1relu}^T \frac{\partial L}{\partial S_2}$$

- $\frac{\partial L}{\partial b_2}$

$$\boxed{\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial S_2}\frac{\partial S_2}{\partial b_2} =} \sum_i \frac{\partial L}{\partial S_{2ij}}$$

- $\frac{\partial L}{\partial W_1}$

$$\frac{\partial L}{\partial W_1} = X^T \frac{\partial L}{\partial S_1}$$

其中，$\frac{\partial L}{\partial S_1} = \frac{\partial L}{\partial S_{1relu}} \frac{\partial S_1}{\partial S_{1relu}} = \frac{\partial L}{\partial S_2} W_2^T (S_1 > 0)$

$$\frac{\partial L}{\partial W_1} = X^T \frac{\partial L}{\partial S_2} W_2^T (S_1 > 0)$$

- $\frac{\partial L}{\partial b_1}$

$$\boxed{\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial S_1}\frac{\partial S_1}{\partial b_1} =} \sum_i \frac{\partial L}{\partial S_{1ij}}$$

3.Evaluation

- Global Configuration
  - Grid Search
    - Learning Rate
    - Regularization Strength
  - Data Processing
    - Normalization
    - Standardization
    - Data augmentation

Configuration	Features	Accuracy
Softmax	-	0.394
Softmax	HOG + HSV	0.462
Softmax +Relu	HOG + HSV	0.493
2-Layer Softmax +Relu	HOG + HSV	0.591

4.Reference

- CS231n notes
  - http://cs231n.github.io/linear-classify/
  - http://cs231n.github.io/optimization-1/

- Related Work
  - Deep Learning using Linear Support Vector Machines

	ConvNet+Softmax	ConvNet+SVM
Test error	14.0%	11.9%

  - Comparison of Support Vector Machine and Softmax Classifiers in Computer Vision

Q&A

Thank you for watching!