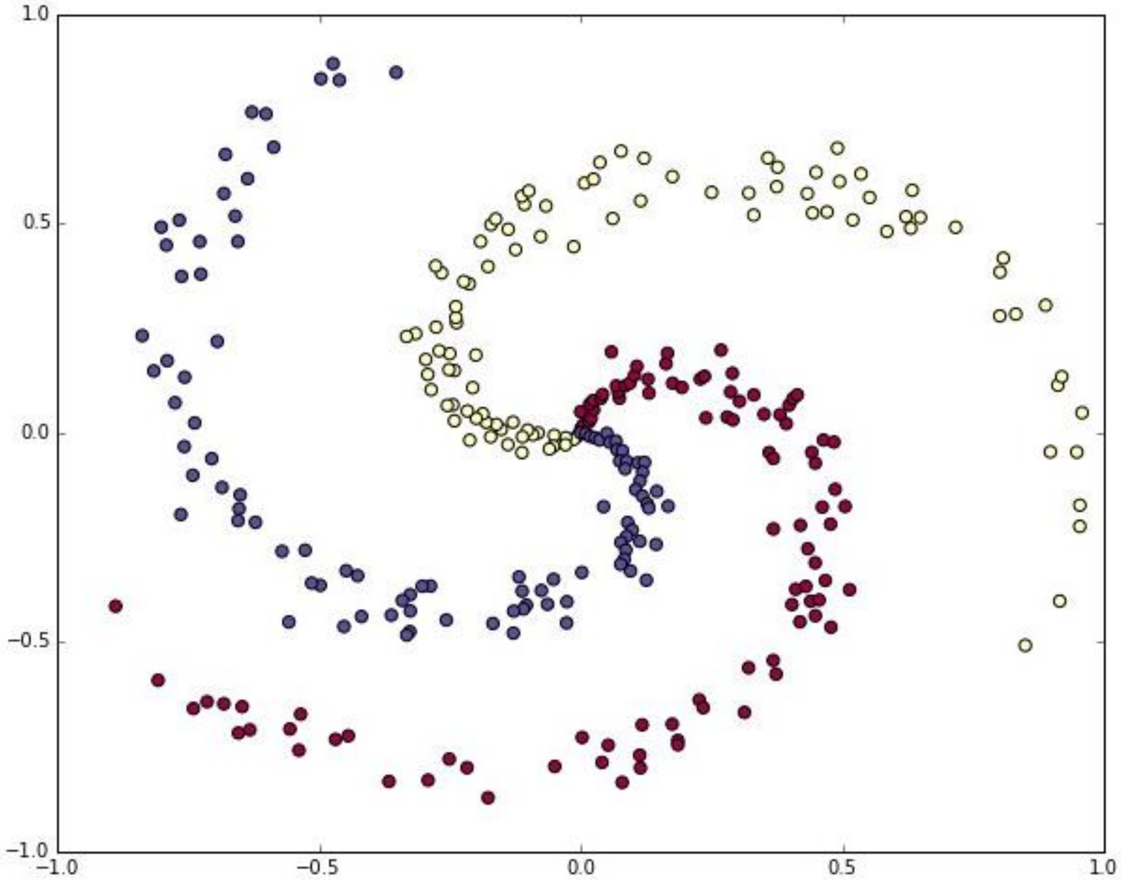
9강: 2D 장난감 데이터에 대한 작은 신경망 구현

과제3 관련 샘플

hidden_size	learning_rate	lr_decay	num_epochs	reg	훈련정확도	검증정확도
50	1.00E-04	0.95	5	0		
200	1.00E-04	0.95	5	0	0.392	0.41
500	1.00E-04	0.95	5	0	0.387	0.403
1000	1.00E-04	0.95	5	0	0.414	0.417
1500	1.00E-04	0.95	5	0	0.429	0.422
2000	1.00E-04	0.95	5	0	0.404	0.414
1500	1.00E-04	0.95	5	1.00E-04	0.43	0.421
1500	1.00E-04	0.95	5	1.00E-03	0.434	0.445
1500	1.00E-04	0.95	5	1.00E-02	0.422	0.43
1500	1.00E-04	0.95	5	1.00E-01	0.424	0.41
1500	1.00E-04	0.95	5	1.00E+00	0.432	0.433
1500	1.00E-04	0.95	10	1.00E+00	0.485	0.456
1500	1.00E-03	0.95	5	1.00E-03	0.371	0.383
1500	1.00E-05	0.95	5	1.00E-03	0.32	0.33
2000	1.00E-04	0.95	5	1.00E-03	0.427	0.446
1000	1.00E-04	0.95	5	1.00E-03	0.411	0.422
3000	1.00E-04	0.95	5	1.00E-03	0.457	0.435
3000	1.00E-04	0.95	20	1.00E-03	0.532	0.49

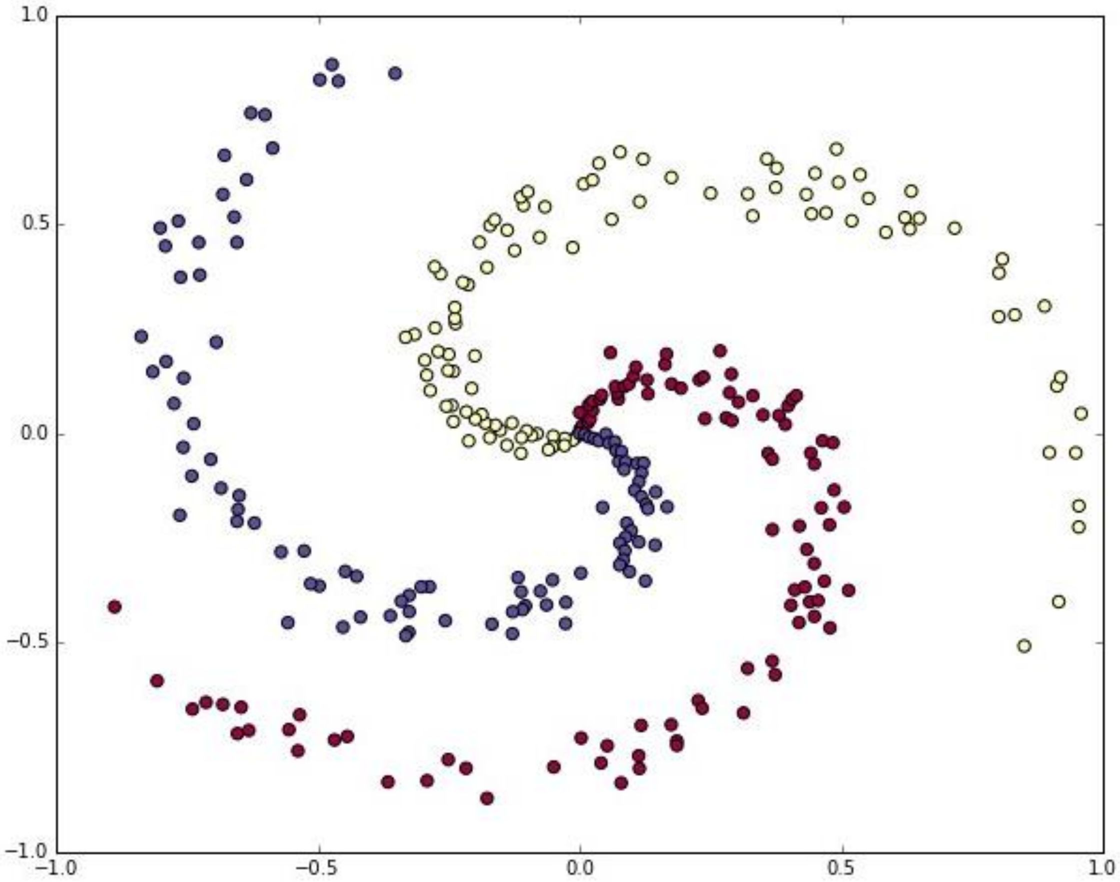
2차원에서의 장난감 신경망(toy neural network) 구현

- 2차원에서의 장난감 신경망 구현
- 먼저, 간단한 선형분류기를 구현한 후 2계층 신경망으로 확장



2D 장난감 데이터셋

```python
N = 100 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
# lets visualize the data:
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
plt.show()
```



소프트맥스 선형분류기 학습

- 소프트맥스 분류기 학습
  - 점수함수:

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

  - 손실:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

소프트맥스 선형분류기 학습

- 파라미터 초기화

```
# initialize parameters randomly
W = 0.01 * np.random.randn(D,K)
b = np.zeros((1,K))
```

- 클래스 점수 계산
  - f=xW+b

```
# compute class scores for a linear classifier
scores = np.dot(X, W) + b
```

소프트맥스 선형분류기 학습

- 손실 계산
  - 소프트맥스 분류기와 관련된 크로스-엔트로피 손실 사용

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

  - 소프트맥스 분류기는 f의 모든 성분들을 세 개의 클래스에 대한 정규화되지 않은 로그 확률로 해석
  - 이들을 지수화하고 정규화하여 확률을 얻음
  - L_i 표현식은 올바른 클래스의 확률이 높을 때 낮고, 확률이 낮을 때 높음

소프트맥스 선형분류기 학습

- 손실 계산
  - 손실의 최종 형태는 다음과 같음

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2}\lambda \sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}}$$

```
num_examples = X.shape[0]
# get unnormalized probabilities
exp_scores = np.exp(scores)
# normalize them for each example
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
```

소프트맥스 선형분류기 학습

- 손실 계산
  - 올바른 클래스에 대한 확률

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

```
correct_logprobs = -np.log(probs[range(num_examples),y])
```

  - 전체 손실 계산

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2}\lambda \sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}}$$

```
# compute the loss: average cross-entropy loss and regularization
data_loss = np.sum(correct_logprobs)/num_examples
reg_loss = 0.5*reg*np.sum(W*W)
loss = data_loss + reg_loss
```

소프트맥스 선형분류기 학습

- 역전파
  - 경사하강법을 통해 손실을 최소화
  - p: (정규화된) 확률 벡터

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad\qquad L_i = -\log(p_{y_i})$$

  - 그래디언트 계산

$$\frac{\partial L_i}{\partial f_k} = p_k - 1(y_i = k)$$

  - 예시
    - p=[0.2,0.3,0.5]에 대해 올바른 클래스가 중간 클래스일 때, 그래디언트는 [0.2,-0.7,0.5]
    - 직관에 들어맞는 걸 알 수 있음

소프트맥스 선형분류기 학습

- 역전파
  - 점수에 대한 그래디언트

$$\frac{\partial L_i}{\partial f_k} = p_k - 1(y_i = k)$$

```
dscores = probs
dscores[range(num_examples),y] -= 1
dscores /= num_examples
```

  - W, b에 대한 그래디언트
    - scores = np.dot(X, W) + b

```
dW = np.dot(X.T, dscores)
db = np.sum(dscores, axis=0, keepdims=True)
dW += reg*W # don't forget the regularization gradient
```

소프트맥스 선형분류기 학습

- 파라미터 업데이트 수행
  - 손실 감소를 위해 음의 그래디언트 방향으로 파라미터 업데이트 수행

```
# perform a parameter update
W += -step_size * dW
b += -step_size * db
```

소프트맥스 선형분류기 학습

```python
#Train a Linear Classifier

# initialize parameters randomly
W = 0.01 * np.random.randn(D,K)
b = np.zeros((1,K))

# some hyperparameters
step_size = 1e-0
reg = 1e-3 # regularization strength

# gradient descent loop
num_examples = X.shape[0]
for i in range(200):

  # evaluate class scores, [N x K]
  scores = np.dot(X, W) + b

  # compute the class probabilities
  exp_scores = np.exp(scores)
  probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

  # compute the loss: average cross-entropy loss and regularization
  correct_logprobs = -np.log(probs[range(num_examples),y])
  data_loss = np.sum(correct_logprobs)/num_examples
  reg_loss = 0.5*reg*np.sum(W*W)
  loss = data_loss + reg_loss
  if i % 10 == 0:
    print "iteration %d: loss %f" % (i, loss)
```

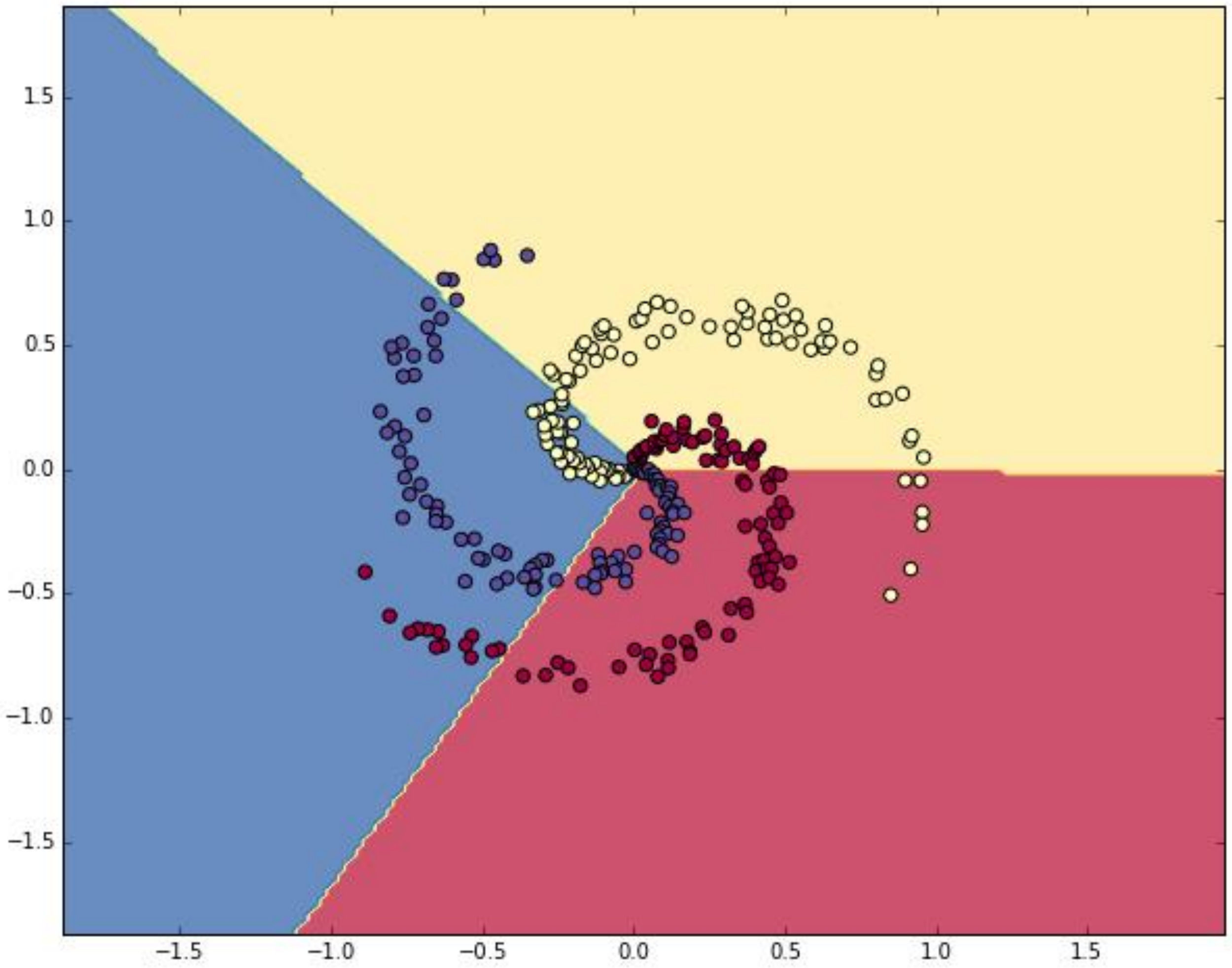
```python
  if i % 10 == 0:
    print "iteration %d: loss %f" % (i, loss)

  # compute the gradient on scores
  dscores = probs
  dscores[range(num_examples),y] -= 1
  dscores /= num_examples

  # backpropate the gradient to the parameters (W,b)
  dW = np.dot(X.T, dscores)
  db = np.sum(dscores, axis=0, keepdims=True)

  dW += reg*W # regularization gradient

  # perform a parameter update
  W += -step_size * dW
  b += -step_size * db
```

소프트맥스 선형분류기 학습 결과

```
iteration 0: loss 1.096956
iteration 10: loss 0.917265
iteration 20: loss 0.851503
iteration 30: loss 0.822336
iteration 40: loss 0.807586
iteration 50: loss 0.799448
iteration 60: loss 0.794681
iteration 70: loss 0.791764
iteration 80: loss 0.789920
iteration 90: loss 0.788726
iteration 100: loss 0.787938
iteration 110: loss 0.787409
iteration 120: loss 0.787049
iteration 130: loss 0.786803
iteration 140: loss 0.786633
iteration 150: loss 0.786514
iteration 160: loss 0.786431
iteration 170: loss 0.786373
iteration 180: loss 0.786331
iteration 190: loss 0.786302
```

```
# evaluate training set accuracy
scores = np.dot(X, W) + b
predicted_class = np.argmax(scores, axis=1)
print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```

소프트맥스 선형분류기 학습 결과



신경망 학습

- 선형분류기 대신 신경망을 사용해 분류
- 장난감 데이터는 1개의 추가적 은닉계층으로 충분
- 파라미터 초기화

```
# initialize parameters randomly
h = 100 # size of hidden layer
W = 0.01 * np.random.randn(D,h)
b = np.zeros((1,h))
W2 = 0.01 * np.random.randn(h,K)
b2 = np.zeros((1,K))
```

- 점수 계산

```
# evaluate class scores with a 2-layer Neural Network
hidden_layer = np.maximum(0, np.dot(X, W) + b) # note, ReLU activation
scores = np.dot(hidden_layer, W2) + b2
```

- 역전파
  - 이전처럼 점수를 기반으로 손실 및 점수의 그래디언트 dscores 계산
  - dW2, db2 계산

```
# backpropate the gradient to the parameters
# first backprop into parameters W2 and b2
dW2 = np.dot(hidden_layer.T, dscores)
db2 = np.sum(dscores, axis=0, keepdims=True)
```

  - 은닉계층 출력의 그래디언트 dhidden 계산

```
dhidden = np.dot(dscores, W2.T)
```

신경망 학습

- 역전파
  - 은닉계층 출력에 대한 그래디언트 dhidden을 가진 상황
  - ReLU 함수를 역전파해야함
  - r=max(0,x) 함수에서 dr/dx=1(x>0)이 성립

```
# backprop the ReLU non-linearity
dhidden[hidden_layer <= 0] = 0
```

  - 첫 번째 계층의 가중치와 편향벡터로 역전파 진행

```
# finally into W,b
dW = np.dot(X.T, dhidden)
db = np.sum(dhidden, axis=0, keepdims=True)
```

신경망 학습 결과

```python
# initialize parameters randomly
h = 100 # size of hidden layer
W = 0.01 * np.random.randn(D,h)
b = np.zeros((1,h))
W2 = 0.01 * np.random.randn(h,K)
b2 = np.zeros((1,K))

# some hyperparameters
step_size = 1e-0
reg = 1e-3 # regularization strength

# gradient descent loop
num_examples = X.shape[0]
for i in range(10000):

  # evaluate class scores, [N x K]
  hidden_layer = np.maximum(0, np.dot(X, W) + b) # note, ReLU activation
  scores = np.dot(hidden_layer, W2) + b2

  # compute the class probabilities
  exp_scores = np.exp(scores)
  probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

  # compute the loss: average cross-entropy loss and regularization
  correct_logprobs = -np.log(probs[range(num_examples),y])
  data_loss = np.sum(correct_logprobs)/num_examples
  reg_loss = 0.5*reg*np.sum(W*W) + 0.5*reg*np.sum(W2*W2)
  loss = data_loss + reg_loss
  if i % 1000 == 0:
    print "iteration %d: loss %f" % (i, loss)

  # compute the gradient on scores
  dscores = probs
```

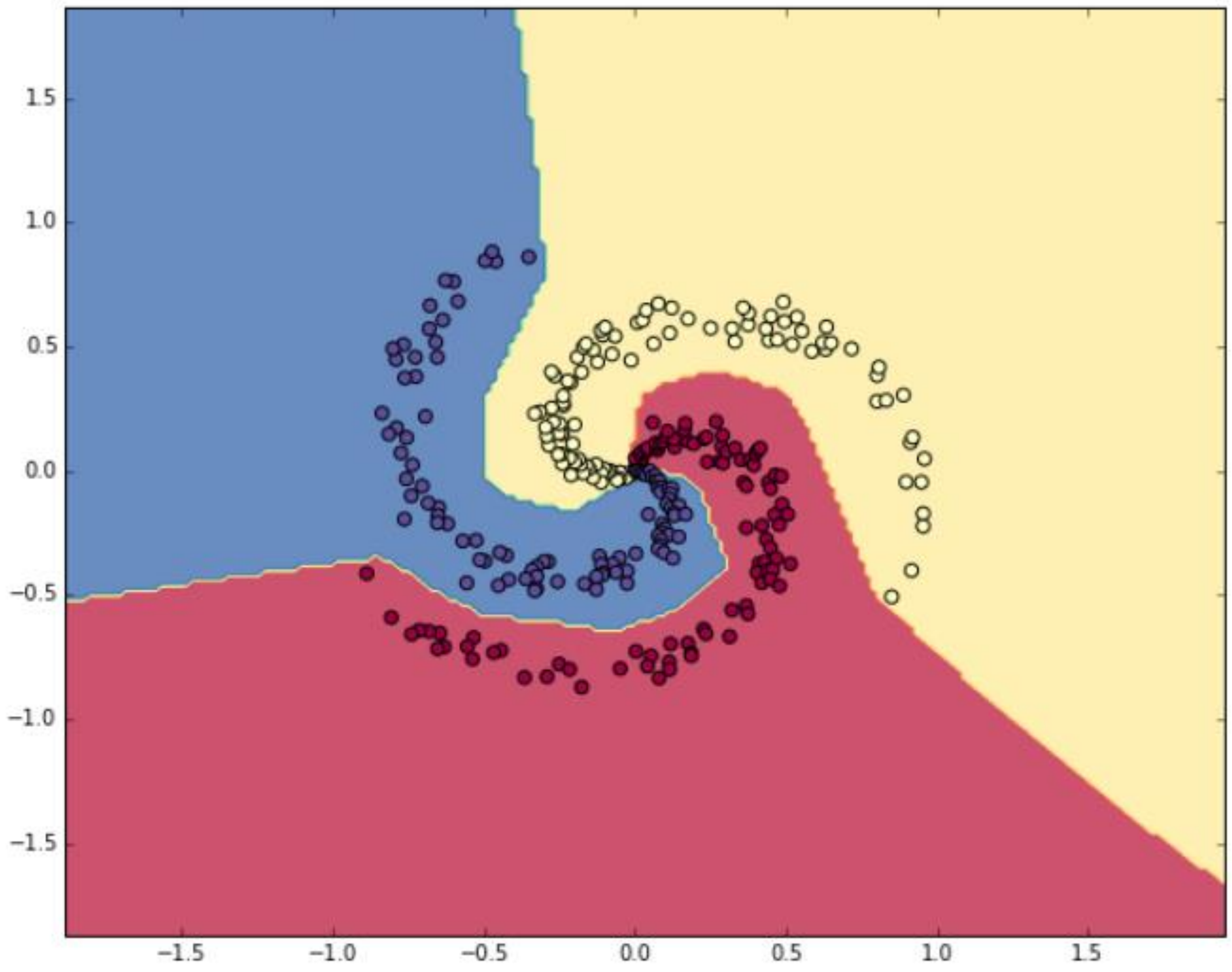
```python
# compute the gradient on scores
dscores = probs
dscores[range(num_examples),y] -= 1
dscores /= num_examples

# backpropate the gradient to the parameters
# first backprop into parameters W2 and b2
dW2 = np.dot(hidden_layer.T, dscores)
db2 = np.sum(dscores, axis=0, keepdims=True)
# next backprop into hidden layer
dhidden = np.dot(dscores, W2.T)
# backprop the ReLU non-linearity
dhidden[hidden_layer <= 0] = 0
# finally into W,b
dW = np.dot(X.T, dhidden)
db = np.sum(dhidden, axis=0, keepdims=True)

# add regularization gradient contribution
dW2 += reg * W2
dW += reg * W

# perform a parameter update
W += -step_size * dW
b += -step_size * db
W2 += -step_size * dW2
b2 += -step_size * db2
```

신경망 학습 결과

- 98%의 정확도!

```
iteration 0: loss 1.098744
iteration 1000: loss 0.294946
iteration 2000: loss 0.259301
iteration 3000: loss 0.248310
iteration 4000: loss 0.246170
iteration 5000: loss 0.245649
iteration 6000: loss 0.245491
iteration 7000: loss 0.245400
iteration 8000: loss 0.245335
iteration 9000: loss 0.245292
```

```
# evaluate training set accuracy
hidden_layer = np.maximum(0, np.dot(X, W) + b)
scores = np.dot(hidden_layer, W2) + b2
predicted_class = np.argmax(scores, axis=1)
print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5