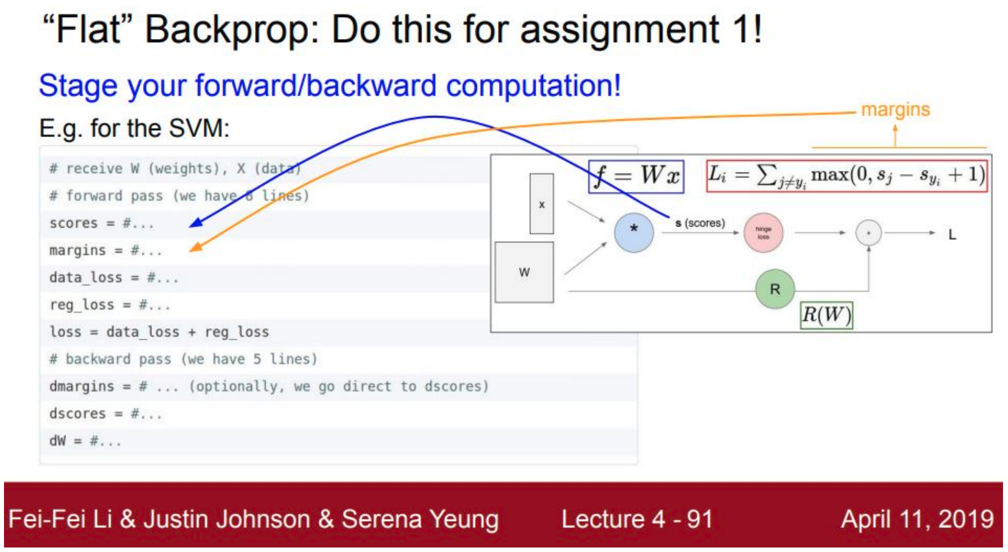
Assignment1-2-3

October 21, 2019

2. 2장 loss functions and optimizations 자료 참고하여 아래의 문제 (HW 표시 됨)를 해결하세요

a. what happens to loss if car scores change a bit (5점)



“Flat” Backprop: Do this for assignment 1!

Stage your forward/backward computation!

E.g. for the SVM:

```
# receive W (weights), X (data)
# forward pass (we have 6 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```

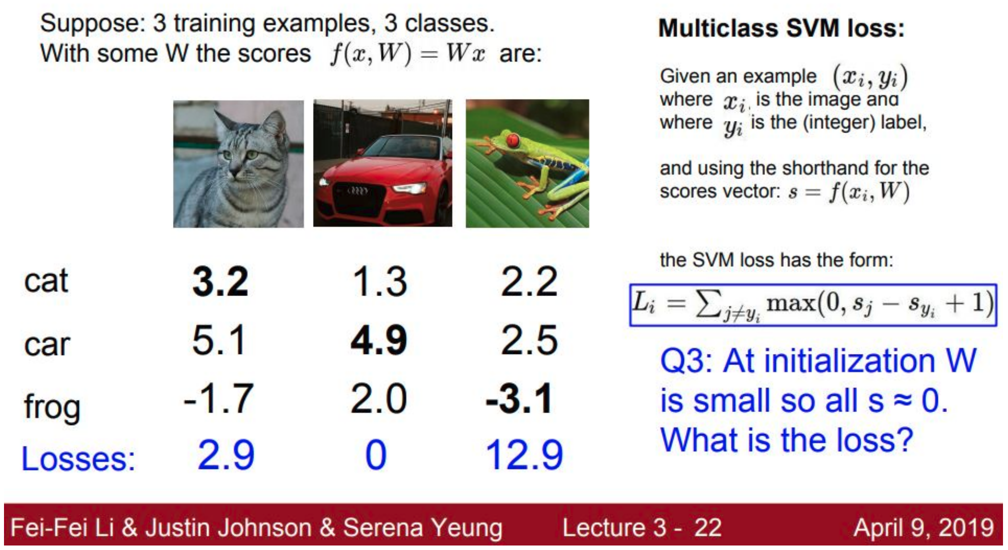
Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 4 - 91   April 11, 2019

answer> 현재 이 예제에서는 car scores이 다른 class보다 크 상태이기 때문에 car scores를 약간 변경해도 Loss은 크게 변경되지 않는다. 즉, 간단히 두번째 이미지에서 car score이미 다른 것보다 이미 margin 1이기 때문에 loss는 여전히 0이다. 다른 첫번째 및 세번째 이미지는 correct label이 이미 incorrect label(car score)보다 작기 때문에 이에 따른 loss도 살짝 변하기만 한다.

b. at initialization W is small so all s~0, What is the loss?(5점)

answer> SVM loss은 incorrect label보다 correct label 차이가 margin 만큼을 유지하도록 하는데 초기에 작은 수로 W를 initialization을 하면 각 label score도 0에 근사한 값을 가지 상태라면 현재 margin을 1로 두고 있기 때문에, C를 classes의 수로한다면 약간의 오차는 존재하지만 loss는 C-1(the number of incorrect labels)이다. 즉, 하나의 observation에서 loss는 incorrect labels의 수(C-1)가 된다.

c. What if the sum was over all classes? (including j = y_i) (5점)



Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

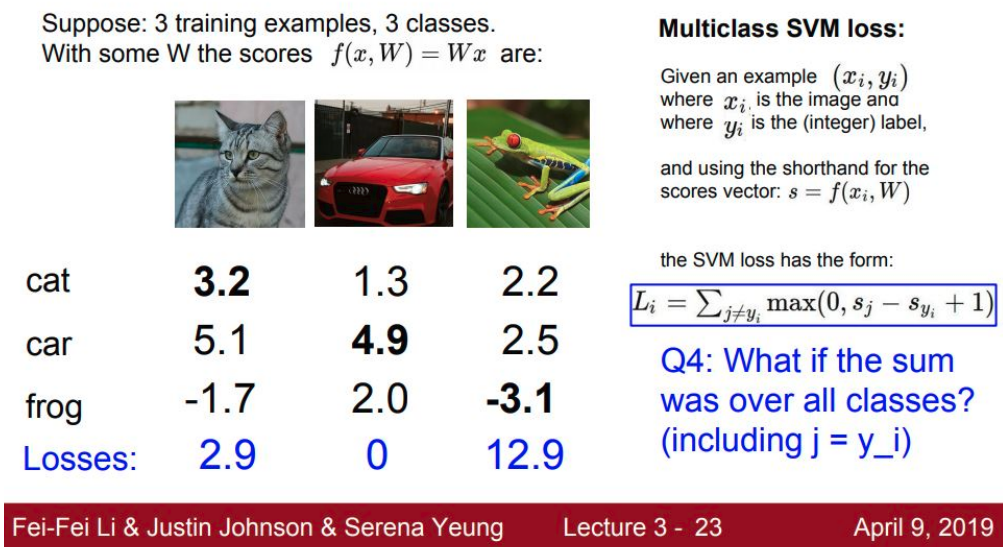
Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q3: At initialization W is small so all s ≈ 0.
What is the loss?



Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

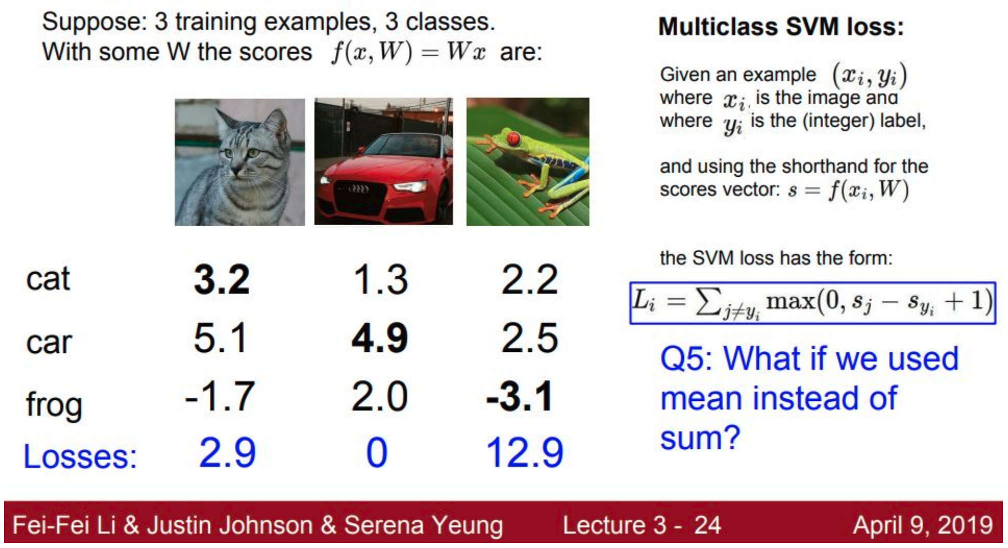
and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q4: What if the sum was over all classes? (including j = y_i) answer> 이 때는 correct class도 포함을 하기 때문에 loss의 minimum은 1이고 다른 환경이 같다면 학습된 model은 같을 것이다. 왜냐하면 학습 목적 자체가 correct label과 incorrect label의 margin 만큼을 유지하는 것이기 때문이다. 즉, 관습적으로 loss가 0이라면 직관적으로 해석을 하기 편해서 y_i class를 빼고 하지만, 같은 환경에서는 같은 classifier를 얻는다.

d. what if we used to mean instead of sum?(5점)



Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q5: What if we used mean instead of sum?

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 24 April 9, 2019

answer> model의 결과는 변하지 않는다. 실험을 하기 전에 데이터에 따라 분류 클래스들은 고정되고 단지 loss가 상수만큼의 크기만큼 조정될 뿐이다. 그리고 모델을 학습을 할때 실제 loss 값 자체의 관심을 가지는 것이 아니라 현재 데이터, loss function, 그리고 model에 따른 상대적인 loss에 관심을 가지고 이를 최소화하는 방식으로 model을 학습을 하기 때문에 상수만큼 크기를 조절하는 것은 model의 결과에 영향을 주지 못한다.

e. what if we used L_i~(5점)

answer> loss function이 변하게 된다면 weight는 loss를 낮추기 위해 학습하기 때문에 parameter-loss를 축으로 하는 공간에서 목적함수가 다르기 때문에 기존의 제곱을 하지 않는 loss function과 다른 classifier을 얻게 된다. 즉 squared hinge loss는 margin이 1보다 멀어질수록 linearly하게 loss를 계산을 하는 것이 아니라 non-linearly하기 때문에 다른 claissifier를 얻는다.

3. 3장 introduction to neural networks 자료 참고하여 아래의 문제를 해결하세요.

a. flat backprop: stage your forward/ backward computation 자신의 연산 그래프 정의, 코드 포함 (10점)

```
In [1]: import numpy as np

        class Multiclass_SVM():
            def svm_loss(self, W, X, y, reg):
```

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

Q6: What if we used

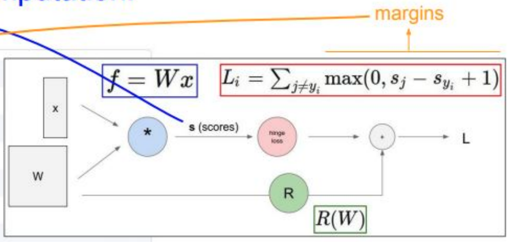
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

"Flat" Backprop: Do this for assignment 1!

Stage your forward/backward computation!

E.g. for the SVM:

```
# receive W (weights), X (data)
# forward pass (we have 6 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```

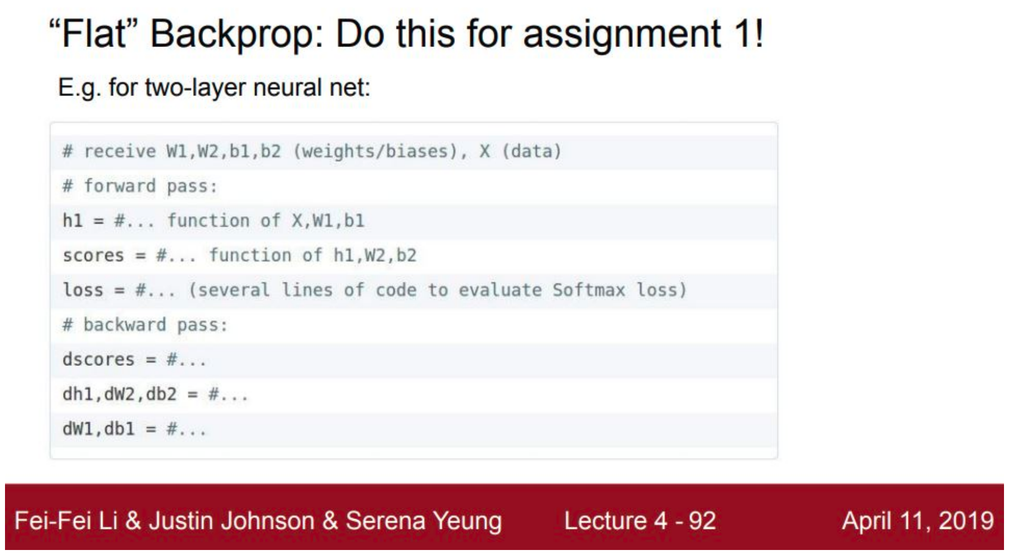


```
            """
            Structured SVM loss function, naive implementation (with loops).
            Inputs have dimension D, there are C classes, and we operate on minibatches
            of N examples.
            Inputs:
                - W: A numpy array of shape (D, C) containing weights.
                - X: A numpy array of shape (N, D) containing a minibatch of data.
                - y: A numpy array of shape (N,) containing training labels; y[i] = c mean
                     that X[i] has label c, where 0 <= c < C.
                - reg: (float) regularization strength
            Returns a tuple of:
                - loss as single float
                - gradient with respect to weights W; an array of same shape as W
            """
            dW = np.zeros(W.shape)  # initialize the gradient as zero

            # compute the loss and the gradient
            num_classes = W.shape[1]
            num_train = X.shape[0]
            loss = 0.0
            for i in range(num_train):
                scores = X[i].dot(W)
                correct_class_score = scores[y[i]]
                loss_contributors_count = 0
                for j in range(num_classes):
                    if j == y[i]:
                        continue
                    margin = scores[j] - correct_class_score + 1  # margin = 1
                    if margin > 0:
                        loss += margin
                        # incorrect class gradient part
                        dW[:, j] += X[i]
                        # correct class gradient part
                        dW[:, y[i]] -= X[i]

            # Right now the loss is a sum over all training examples, but we want it
            # to be an average instead so we divide by num_train.
            loss /= num_train
            dW /= num_train

            # Add regularization to the loss.
            loss += reg * np.sum(W * W)
            # Add regularization to the gradient
            dW += 2 * reg * W

            return loss, dW
```

```
In [2]: x = np.array([[0.2, 0.4],[0.2, 0.4]])
```

```
        input_N, input_D = x.shape
        y = [0,1]
        output_D = len(y)
        # generate a random SVM weight matrix of small numbers
        W = np.random.randn(input_D, output_D)
        linear_SVM = Multiclass_SVM()
        loss, grad = linear_SVM.svm_loss(W, x, y, 0.000005)
        print("==================")
        print("loss: {}".format(loss))
        print("grad: {}".format(grad))
```

```
==================
loss: 1.0000212658358887
grad: [[-1.31177059e-05 -1.43532899e-05]
 [ 5.55046919e-06 -4.05189734e-06]]
```



"Flat" Backprop: Do this for assignment 1!

E.g. for two-layer neural net:

```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```

Fei-Fei Li & Justin Johnson & Serena Yeung   Lecture 4 - 92   April 11, 2019

```
In [3]: import numpy as np

        class TwolayerNet():
            """
            A two layer fully-connected nerual network. The network has an input dimension of
            a hidden layer dimension of H, I train the network with activation fucntion of ReL
            and L2 Distance Loss funtion on the weight matrices.

            In other words, the network has the architecture like image above.

            input - fully-connected layer - ReLU - fully-connected layer - softmax
            """
```

```
    def __init__(self, input_size=2, hidden_size=2, output_size=2):
        """
        Initialized the model. Weights are initialzied to small random values
        and biases are initialized to zero. Weight and biases are stored
        in the variable self.params, which is a dictionary with the following keys:

        W1 = First layer weights; has a shape (D, H)
        b1: First layer biases; has shape (H,)
        W2 = Second layer weights; has a shape (H,C)
        b2: Second layer biases; has shape (C,)

        Inputs :
            - input_size: The dimension D of the input data.
            - hiddend_size: The number of neurons H in the hidden layer.
            - output_size: The number of data point's dimensionality.
        """
        self.params = {}
        self.params['W1'] = np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

    def loss(self, X, y):
        """
        Compute the loss and gradients for a two layer fully-connected neural network.

        Input:
            - X: Input data of shape (N, D). Each X[i] is a training sample.

        Return:
            - loss: Loss (data loss and regularization loss) for this batch of trainin
            - grads: gradients of all variable in our network with respect to the loss
        """

        # unpack variables from the params dictionary
        W1, b1 = self.params["W1"], self.params["b1"]
        W2, b2 = self.params["W2"], self.params["b2"]
        num_data, dim_data = X.shape

        # forward pass
        H1 = np.dot(x, W1)+b1
        X2 = np.fmax(0, H1)
        H2 = np.dot(H1, W2) + b2
        scores = H2

        # compute loss
        scores -= scores.max()
        scores_exp = np.exp(scores)
```

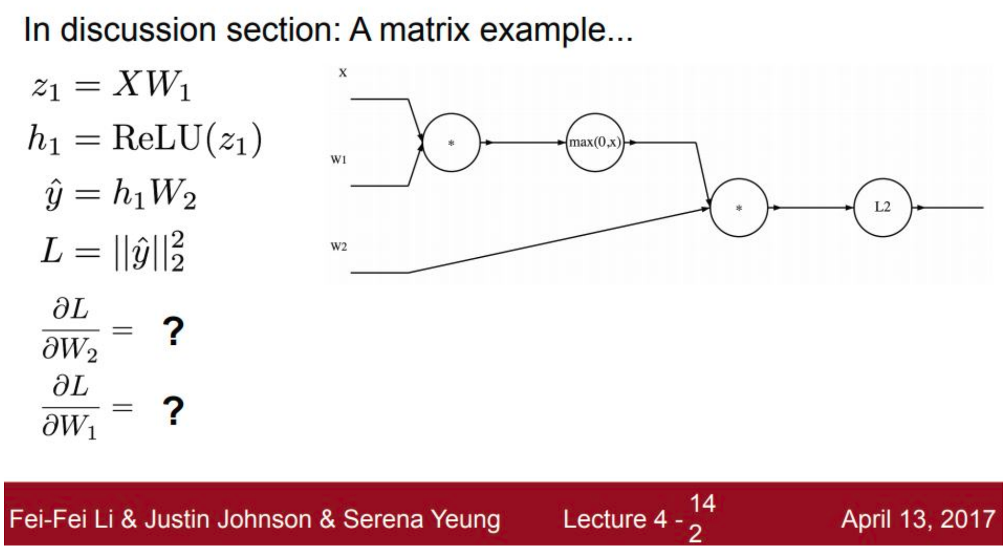
```
                softmax_matrix = scores_exp / np.sum(scores_exp, axis=1, keepdims=True)
                loss = softmax_matrix[np.arange(num_data), y]
                loss = -np.sum(np.log(loss))/ num_data

                # Backward
                grads = {}

                softmax_matrix[np.arange(num_data), y] -= 1 # (scores_sum - cors) / scores_sum

                softmax_matrix /= num_data

                # W2 gradient
                dW2 = X2.T.dot(softmax_matrix)

                # b2 gradient
                db2 = softmax_matrix.sum(axis=0)

                # W1 gradeint
                dW1 = softmax_matrix.dot(W2.T)
                dH1 = dW1 * (H1 > 0)
                dW1 = X.T.dot(dH1)

                # b1 gradient
                db1 = dH1.sum(axis=0)

                grads = {'W1':dW1, 'b1':db1, 'W2':dW2, 'b2':db2}
                return loss, grads
```

```
In [4]: x = np.array([[0.2, 0.4],[0.2, 0.4]])
        input_N, input_D = x.shape
        y = [0,1]
        output_D = len(y)

        nn = TwolayerNet(input_size=input_D, hidden_size=2, output_size=output_D)
        loss, grads = nn.loss(x,y)
        print("==================")
        print("loss: {}".format(loss))
        print("grad_W2: {}".format(grads["W2"]))
        print("grad_b2: {}".format(grads["b2"]))
        print("grad_W1: {}".format(grads["W1"]))
        print("grad_b1: {}".format(grads["b1"]))
```

```
==================
loss: 0.6943877927038427
grad_W2: [[ 0.          0.        ]
 [-0.00336307  0.00336307]]
grad_b2: [-0.0248905   0.0248905]
grad_W1: [[ 0.         -0.00026769]
 [ 0.         -0.00053538]]
```

```
grad_b1: [ 0.          -0.00133845]
```

b. in discussion section: a Matrix example 손실 함수의 W_1, W_2 편미분을 구하세요. [초기값은 임의 설정하여 연산]



In discussion section: A matrix example...

$$z_1 = XW_1$$
$$h_1 = \mathrm{ReLU}(z_1)$$
$$\hat{y} = h_1 W_2$$
$$L = ||\hat{y}||_2^2$$
$$\frac{\partial L}{\partial W_2} = \ ?$$
$$\frac{\partial L}{\partial W_1} = \ ?$$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 4 - 142 April 13, 2017

```
In [5]: import numpy as np

        class TwolayerNet():
            """
            A two layer fully-connected nerual network. The network has an input dimension of
            a hidden layer dimension of H, I train the network with activation fucntion of ReL
            and L2 Distance Loss funtion on the weight matrices.

            In other words, the network has the architecture like image above.

            input - fully-connected layer - ReLU - fully-connected layer - L2 loss
            """
            def __init__(self, input_size=2, hidden_size=2, output_size=1):
                """
                Initialized the model. Weights are initialzied to small random values
                and biases are initialized to zero. Weight and biases are stored
                in the variable self.params, which is a dictionary with the following keys:

                W1 = First layer weights; has a shape (D, H)
                W2 = Second layer weights; has a shape (H,C)

                Inputs :
                    - input_size: The dimension D of the input data.
                    - hiddend_size: The number of neurons H in the hidden layer.
                    - output_size: The number of data point's dimensionality.
                """
                self.params = {}
                self.params["W1"] = np.random.randn(input_size, hidden_size) # np.array([[0.1,
                self.params["W2"] = np.random.randn(hidden_size, output_size)

            def loss(self, X):
                """
                Compute the loss and gradients for a two layer fully-connected neural network.

                Input:
                    - X: Input data of shape (N, D). Each X[i] is a training sample.

                Return:
                    - loss: Loss (data loss and regularization loss) for this batch of trainin
                    - grads: gradients of all variable in our network with respect to the loss
                """
                # unpack variables from the params dictionary
                W1 = self.params["W1"]
                W2 = self.params["W2"]

                # forward pass

                # first layer
                h1 = X.dot(W1)

                # reference
                # https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.fmax.html
                h1_activated = np.fmax(0, h1)

                # second layer
                h2 = h1_activated.dot(W2)

                loss = np.sum(np.square(h2), axis=1)

                ##################################################
                grads = {}

                grads["dloss"] = 2*h2

                grads["dh2"] = grads["dloss"].dot(W2.T)
                grads["dW2"] = h1_activated.T.dot(grads["dloss"])

                # Relu
                mask = np.zeros((X.shape[0], W1.shape[1]))
                mask[h1>0] = 1
                grads["dh1"] = mask*grads["dh2"]
```

```
                grads["dW1"] = X.T.dot(grads["dh1"])
                grads["dx"] = grads["dh1"].dot(W1.T)

                return loss, grads
```

```
In [6]: x = np.array([[0.2, 0.4]])

        input_N, input_D = x.shape

        nn = TwolayerNet(input_size=input_D, hidden_size=2, output_size=1)

        loss, grads = nn.loss(x)

        print("===== solution =====")
        print("grad_W1: {}".format(grads["dW1"]))
        print("grad_W2: {}".format(grads["dW2"]))
```

```
===== solution =====
grad_W1: [[ 0.48566648 -0.13484185]
 [ 0.97133297 -0.26968371]]
grad_W2: [[2.35375515]
 [0.72140824]]
```