딥러닝 입문 / 중간고사(1반) 10. 27. 2020

1. 이변수 함수

$$f(x,y) = xy$$

에 대하여 learning rate $\eta = 1$과 momentum 계수 $\alpha = 1$로 NAG를 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,2)$에서 출발하여 세 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$를 구하시오.

2. 이변수 함수

$$f(x,y) = xy$$

에 대하여 learning rate $\eta = 1$로 AdaGrad를 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,2)$에서 출발하여 두 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$를 구하시오.

3. 뉴런의 개수가 각각 784, 400, 300, 200, 100, 10인 5층 신경망을 생각하자. 각 $k = 1,2,3,4,5$에 대하여 $k-1$층에서 k층으로 가는 Affine 변환의 가중치 행렬을 W_k라 하자.
   (i) 활성화 함수(activation function)를 sigmoid함수로 잡고 gradient vanishing이나 표현력 제한이 일어나지 않도록 Xavier 초기값을 따라 초기값을 설정하려 한다. 초기 가중치 행렬 W_1, W_2, W_3, W_4, W_5를 각각 어떤 분포를 따라 랜덤하게 선택해야 하겠는가?
   (ii) 활성화 함수(activation function)를 ReLU함수로 잡고 gradient vanishing이나 표현력 제한이 일어나지 않도록 He 초기값을 따라 초기값을 설정하려 한다. 초기 가중치 행렬 W_1, W_2, W_3, W_4, W_5를 각각 어떤 분포를 따라 랜덤하게 선택해야 하겠는가?

4. 5개의 데이터를 묶은 행렬

$$\begin{pmatrix} -2 & 0 & -2 \\ -1 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 3 & 2 \\ 2 & 4 & 4 \end{pmatrix}$$

에 대하여 확대는 $\gamma = [\sqrt{2}, 2\sqrt{2}, 4]$, 이동은 $\beta = [2,4,6]$로써 배치 정규화 (batch normalization)를 적용한 결과를 구하시오.

5. 활성화 함수가 sigmoid인 이층 신경망이 dictionary

$$\{W_1 : \begin{bmatrix} -1 & -2 & -3 \\ 0 & 1 & 2 \end{bmatrix},\ b_1 : [0,0,0],\ W_2 : \begin{bmatrix} 3 & 3 \\ 5 & -5 \\ 9 & 0 \end{bmatrix},\ b_2 : [0,0]\}$$

로 주어져 있다. 데이터 $\mathbf{x} = [\log 2, 0]$의 라벨이 $[0,1]$이라 하자. 손실함수는 교차 엔트로피(cross entropy)에 L^2-규제 (L^2-regularization)가 가해져 있다. 가중치 감소의 세기가

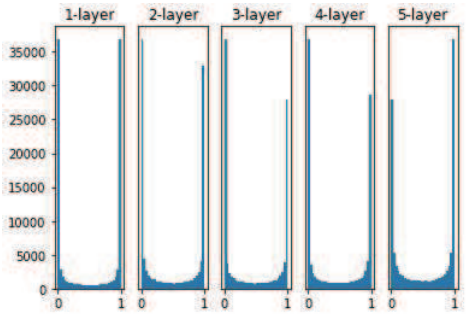
$$\lambda = \frac{1}{84}$$

일 때, 데이터 $\mathbf{x}$의 손실함수 값을 구하시오.

6. 밑줄 친 부분을 수식으로 바꾸고 이로부터 Adam의 점화식을 이끌어 내시오.

```
class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None
    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)
        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2**self.iter)
                / (1.0 - self.beta1**self.iter)
        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

7. 표준편차1로 가중치를 초기화 했을 때 다음과 같은 활성화층의 히스토그램을 얻는다. 빈 칸을 채우시오.




```
input_data = np.random.randn(1000, 100)
  node_num = 100
  hidden_layer_size = 5
  activations = {}
  x = input_data
  for i in range(hidden_layer_size):
      if i [      ]:
          x = activations[i-1]
      w = np.random.randn(node_num, node_num) * 1
      a = np.dot(x, w)
      z = sigmoid(a)
      activations[i] = z
  for i, a in activations.items():
      plt.[      ](1, len(activations), i+1)
      plt.title([      ] + "-layer")
      if [      ]: plt.yticks([], [])
      plt.[      ](a.flatten(), 30, range=(0,1))
  plt.show()
```

8. 다음은 배치 정규화(batch normalization)의 순전파와 역전파 코드이다. 빈 칸을 채우시오.

```
def forward(self, x, train_flg):
    if self.running_mean is None:
        N, D = x.shape
        self.running_mean = np.zeros(D)
        self.running_var = np.zeros(D)
    if train_flg:
        mu = [      ]
        xc = x - mu
        var = np.mean(xc**2, axis=0)
        std = np.sqrt(var + 10e-7)
        xn = [      ]
        self.batch_size = x.shape[0]
        self.xc = xc
        self.xn = xn
        self.std = std
        self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
        self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
    else:
        xc = x - self.running_mean
        xn = xc / ((np.sqrt(self.running_var + 10e-7)))
    out = self.gamma * xn + self.beta
    return out
def backward(self, dout):
    dbeta = [      ]
    dgamma = [      ]
    dxn = self.gamma * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = [      ] * dstd / self.std
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmu = np.sum(dxc, axis=0)
    dx = dxc - dmu / self.batch_size
    self.dgamma = dgamma
    self.dbeta = dbeta
    return dx
```

9. 다음은 심층 신경망 코드의 앞부분이다. 빈칸을 채우시오.

```
class MultiLayerNet:
    def __init__(self, input_size, hidden_size_list, output_size, activation='relu',
            weight_init_std='relu', weight_decay_lambda=0):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.weight_decay_lambda = weight_decay_lambda
        self.params = {}
        self.__init_weight(weight_init_std)
        activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
        self.layers = OrderedDict()
        for idx in range(1, self.hidden_layer_num+1):
            self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                      self.params['b' + str(idx)])
            self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
```

```
        idx = self.hidden_layer_num + 1
        self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                  self.params['b' + str(idx)])
        self.last_layer = SoftmaxWithLoss()
    def __init_weight(self, weight_init_std):
        all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
        for idx in range(1, len(all_size_list)):
            scale = weight_init_std
            if str(weight_init_std).lower() in ('relu', 'he'):
                scale = [          ]
            elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
                scale = [          ]
            self.params['W' + str(idx)] = [          ]
            self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
    def loss(self, x, t):
        y = self.predict(x)
        weight_decay = 0
        for idx in range(1, self.hidden_layer_num + 2):
            W = self.params['W' + str(idx)]
            weight_decay += 0.5 * self.weight_decay_lambda * [          ]
        return self.last_layer.forward(y, t) + [          ]
```

10. 다음은 Drop Out 층의 순전파와 역전파를 구현하는 코드이다.

```
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None
    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)
    def backward(self, dout):
        return dout * self.mask
```

입력 데이터와 생성된 random 행렬이 각각

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad \begin{pmatrix} 0.7 & 0.3 & 0.5 \\ 0.1 & 0.6 & 0.1 \\ 0.9 & 0.8 & 0.2 \end{pmatrix}$$

이고 dropout ratio는 0.4라 하자. 훈련할 때와 테스트할 때 출력값이 각각 어떻게 되겠는가?

딥러닝 입문 / 중간고사(2반) 10. 28. 2020

1. 이변수 함수

$$f(x,y) = x^2 + xy$$

에 대하여 learning rate $\eta = 1$과 momentum 계수 $\alpha = 1$로 Momentum을 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,1)$에서 출발하여 세 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$를 구하시오.

2. 이변수 함수

$$f(x,y) = x^2 + xy$$

에 대하여 learning rate $\eta = 1$과 계수 $\beta_1 = \beta_2 = 1/2$로 Adam을 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,1)$에서 출발하여 두 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$를 구하시오.

3. 5개의 데이터를 묶은 행렬

$$\begin{pmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 2 & 3 & 2 \\ 1 & 4 & 2 \\ 4 & 5 & 0 \end{pmatrix}$$

에 대하여 확대는 $\gamma = [\sqrt{30}, 3\sqrt{2}, \sqrt{30}]$, 이동은 $\beta = [5,6,10]$로써 배치 정규화 (batch normalization)를 적용한 결과를 구하시오.

4. 활성화 함수가 sigmoid인 이층 신경망이 dictionary

$$\{W_1 : \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 \end{bmatrix},\ b_1 : [0,0,0,0,0],\ W_2 : \begin{bmatrix} 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 5 & 0 & 0 & 0 \end{bmatrix},\ b_2 : [0,0,0,0,0]\}$$

로 주어져 있다. 데이터 $\mathbf{x} = [\log 2, \log 3, \log 4, \log 5]$의 라벨이 $[0,0,1,0]$이라 하자. 손실함수는 교차 엔트로피(cross entropy)에 L^2-규제 (L^2-regularization)가 가해져 있다. 가중치 감소의 세기가 $\lambda = 1/21$ 일 때, 데이터 $\mathbf{x}$의 손실함수 값을 구하시오.

5. (4번 문제 계속) 교차 엔트로피(cross entropy)값을 L_c라 하고 L^2-규제 (L^2-regularization)까지 고려한 손실함수 값을 L이라 하자. 첫번째 Affine층에 흘러들어온 교차 엔트로피의 미분이

$$\frac{\partial L_c}{\partial \mathbf{y}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

으로 주어져 있다. 여기서, $\mathbf{y}$는 첫번째 Affine층의 출력값이다. L^2-규제 (L^2-regularization)까지 고려한 손실함수 L의 첫번째 가중치 W_1에 대한 미분

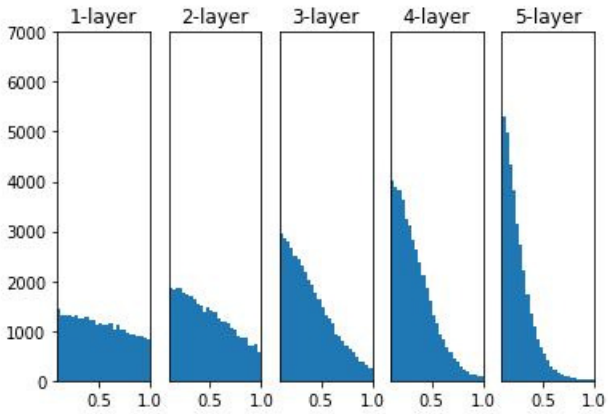
$$\frac{\partial L}{\partial W_1}$$

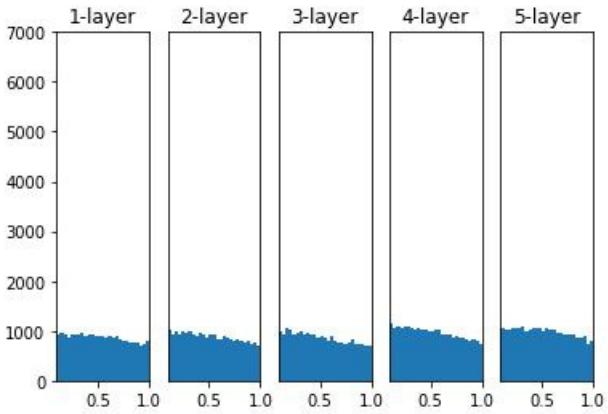
을 구하시오.

6. 다음은 옵티마이저 RMSProp을 구현하는 코드이다. 빈 칸을 채우시오.

```
class RMSprop:
    def __init__(self, lr=0.01, decay_rate = 0.99):
        self.lr = lr
        self.decay_rate = decay_rate
        self.h = None
    def update(self, params, grads):
        if self.h is None:
        self.h = {}
        for key, val in params.items():
            self.h[key] = np.zeros_like(val)
        for key in params.keys():
            self.h[key] [      ] self.decay_rate
            self.h[key] += [      ] * grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / ([      ] + 1e-7)
```

7. 다음 코드를 실행하면 좌측의 히스토그램을 얻는다. 오른쪽 히스트그램이 나오도록 코드를 수정하시오.






```
input_data = np.random.randn(1000, 100)
node_num = 100
hidden_layer_size = 5
activations = {}
x = input_data
for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]
    w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
    a = np.dot(x, w)
    z = ReLU(a)
    activations[i] = z
for i, a in activations.items():
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    if i != 0: plt.yticks([], [])
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```

8. 다음은 배치 정규화(batch normalization)의 순전파와 역전파 코드이다. 빈 칸을 채우시오.

```
def forward(self, x, train_flg):
    if self.running_mean is None:
        N, D = x.shape
        self.running_mean = np.zeros(D)
        self.running_var = np.zeros(D)
    if train_flg:
        mu = x.mean(axis=0)
        xc = x - mu
        var = [          ]
        std = np.sqrt(var + 10e-7)
        xn = xc / std
        self.batch_size = x.shape[0]
        self.xc = xc
        self.xn = xn
        self.std = std
        self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
        self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
    else:
        xc = x - self.running_mean
        xn = xc / ((np.sqrt(self.running_var + 10e-7)))
    out =self.gamma * [          ] + self.beta
    return out

def backward(self, dout):
    dbeta = [          ]
    dgamma = np.sum(self.xn * dout, axis=0)
    dxn = [          ] * dout
    dxc = dxn / self.std
    dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
    dvar = 0.5 * dstd / [          ]
    dxc += (2.0 / self.batch_size) * self.xc * dvar
    dmu = np.sum(dxc, axis=0)
    dx = dxc - dmu / self.batch_size
    self.dgamma = dgamma
    self.dbeta = dbeta
    return dx
```

9. 다음은 심층 신경망 코드의 앞부분이다. 빈칸을 채우시오.

```
class MultiLayerNet:
    def __init__(self, input_size, hidden_size_list, output_size, activation='relu',
            weight_init_std='relu', weight_decay_lambda=0):
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size_list = hidden_size_list
        self.hidden_layer_num = len(hidden_size_list)
        self.weight_decay_lambda = weight_decay_lambda
        self.params = {}
        self.__init_weight(weight_init_std)
        activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
```

```
        self.layers = OrderedDict()
        for idx in range(1, self.hidden_layer_num+1):
            self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                      self.params['b' + str(idx)])
            self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
        idx = self.hidden_layer_num + 1
        self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                  self.params['b' + str(idx)])
        self.last_layer = SoftmaxWithLoss()
    def __init_weight(self, weight_init_std):
        all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
        for idx in range(1, len(all_size_list)):
            scale = weight_init_std
            if str(weight_init_std).lower() in ('relu', 'he'):
                scale = [          ]
            elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
                scale = [          ]
            self.params['W' + str(idx)] = [          ]
            self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
    def loss(self, x, t):
        y = self.predict(x)
        weight_decay = 0
        for idx in range(1, self.hidden_layer_num + 2):
            W = self.params['W' + str(idx)]
            weight_decay += 0.5 * self.weight_decay_lambda * [          ]
        return self.last_layer.forward(y, t) + [          ]
```

10. 다음은 Drop Out 층의 순전파와 역전파를 구현하는 코드이다.

```
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None
    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)
    def backward(self, dout):
        return dout * self.mask
```

입력 데이터와 생성된 random 행렬이 각각

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad \begin{pmatrix} 0.4 & 0.8 & 0.7 \\ 0.2 & 0.6 & 0.6 \\ 0.1 & 0.1 & 0.9 \end{pmatrix}$$

이고 dropout ratio는 0.3라 하자. 훈련할 때와 테스트할 때 출력값이 각각 어떻게 되겠는가?

딥러닝 입문 / 중간고사(3반) 10. 30. 2020

1. 이변수 함수

$$f(x,y) = xy$$

에 대하여 learning rate $\eta = 1$과 momentum 계수 $\alpha = 1$로 Momentum을 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,2)$에서 출발하여 세 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$를 구하시오.

2. 이변수 함수

$$f(x,y) = xy$$

에 대하여 learning rate $\eta = 1$과 forgetting factor $\gamma = 3/4$으로 RMSProp을 적용하려 한다. 초기 위치 $\mathbf{x}_0 = (1,2)$에서 출발하여 두 발자국 걸어갈때, $\mathbf{x}_1$, $\mathbf{x}_2$를 구하시오.

3. 5개의 데이터를 묶은 행렬

$$\begin{pmatrix} 2 & -2 & -2 \\ 0 & -1 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$$

에 대하여 확대는 $\gamma = [4\sqrt{5}, 2\sqrt{2}, 4]$, 이동은 $\beta = [10,4,6]$로써 배치 정규화 (batch normalization)를 적용한 결과를 구하시오.

4. (3번 문제 계속) batch normalization층으로 흘러 들어온 미분이

$$\begin{pmatrix} 1 & -\sqrt{2} & -2 \\ 1 & -\sqrt{2} & 2 \\ 1 & 2 & -2 \\ 1 & \sqrt{2} & 2 \\ 1 & \sqrt{2} & 2 \end{pmatrix}$$

일 때, 미분 $\frac{\partial L}{\partial \beta}$와 $\frac{\partial L}{\partial \gamma}$의 값을 구하시오.

5. 활성화 함수가 Relu인 이층 신경망이 dictionary

$$\{W_1 : \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix},\ b_1 : [0,0,0,0,0],\ W_2 : \begin{bmatrix} 0 & 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \end{bmatrix},\ b_2 : [0,0,0,0,0]\}$$

로 주어져 있다. 데이터 $\mathbf{x} = [1,2,3,4,5]$의 라벨이 $[1,0,0,0,0]$이라 하자. 손실함수는 교차 엔트로피(cross entropy)에 L^2-규제 (L^2-regularization)가 가해져 있다. 가중치 감소의 세기가 $\lambda = \frac{1}{5}$ 일 때, 데이터 $\mathbf{x}$의 손실함수 값을 구하시오.

6. 밑줄 친 부분을 수식으로 바꾸고 이로부터 Adam의 점화식을 이끌어 내시오.

```
class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None
    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)
        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2**self.iter)
                / (1.0 - self.beta1**self.iter)
        for key in params.keys():
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])
            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)
```

7. 다음은 배치 정규화(batch normalization)의 순전파와 역전파 코드이다. 빈 칸을 채우시오.

```
def forward(self, x, train_flg):
    if self.running_mean is None:
        N, D = x.shape
        self.running_mean = np.zeros(D)
        self.running_var = np.zeros(D)
    if train_flg:
        mu = x.mean(axis=0)
        xc = x - mu
        var = np.mean(xc**2, axis=0)
        std = np.sqrt(var + 10e-7)
        xn = xc / std
        self.batch_size = x.shape[0]
        self.xc = xc
        self.xn = xn
        self.std = std
        self.running_mean = self.momentum * self.running_mean + (1-self.momentum) * mu
        self.running_var = self.momentum * self.running_var + (1-self.momentum) * var
    else:
        xc = [      ]
        xn = xc / ((np.sqrt([      ] + 10e-7)))
    out = self.gamma * xn + self.beta
    return out
def backward(self, dout):
```

```
        dbeta = [      ]
        dgamma = np.sum(self.xn * dout, axis=0)
        dxn = [      ] * dout
        dxc = dxn / self.std
        dstd = -np.sum((dxn * self.xc) / (self.std * self.std), axis=0)
        dvar = [      ] * dstd / self.std
        dxc += (2.0 / self.batch_size) * self.xc * dvar
        dmu = np.sum(dxc, axis=0)
        dx = dxc - dmu / self.batch_size
        self.dgamma = dgamma
        self.dbeta = dbeta
        return dx
```

8. 다음은 심층 신경망 코드의 앞부분이다. 빈칸을 채우시오.

```
    class MultiLayerNet:
        def __init__(self, input_size, hidden_size_list, output_size, activation='relu',
                weight_init_std='relu', weight_decay_lambda=0):
            self.input_size = input_size
            self.output_size = output_size
            self.hidden_size_list = hidden_size_list
            self.hidden_layer_num = len(hidden_size_list)
            self.weight_decay_lambda = weight_decay_lambda
            self.params = {}
            self.__init_weight(weight_init_std)
            activation_layer = {'sigmoid': Sigmoid, 'relu': Relu}
            self.layers = OrderedDict()
            for idx in range(1, self.hidden_layer_num+1):
                self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                          self.params['b' + str(idx)])
                self.layers['Activation_function' + str(idx)] = activation_layer[activation]()
            idx = self.hidden_layer_num + 1
            self.layers['Affine' + str(idx)] = Affine(self.params['W' + str(idx)],
                                                      self.params['b' + str(idx)])
            self.last_layer = SoftmaxWithLoss()
        def __init_weight(self, weight_init_std):
            all_size_list = [self.input_size] + self.hidden_size_list + [self.output_size]
            for idx in range(1, len(all_size_list)):
                scale = weight_init_std
                if str(weight_init_std).lower() in ('relu', 'he'):
                    scale = [      ]
                elif str(weight_init_std).lower() in ('sigmoid', 'xavier'):
                    scale = [      ]
                self.params['W' + str(idx)] = [      ]
                self.params['b' + str(idx)] = np.zeros(all_size_list[idx])
        def predict(self, x):
            for layer in self.layers.values():
                x = layer.forward(x)
```

```
        return x
    def loss(self, x, t):
        y = self.predict(x)
        weight_decay = 0
        for idx in range(1, self.hidden_layer_num + 2):
            W = self.params['W' + str(idx)]
            weight_decay += 0.5 * self.weight_decay_lambda * [      ]
        return self.last_layer.forward(y, t) + [      ]
```

9. (8번 계속) 다음은 심층 신경망 코드의 뒷부분이다. 빈칸을 채우시오.

```
    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.[      ](y, axis=1)
        accuracy = np.sum([      ]) / float(x.shape[0])
        return accuracy
    def gradient(self, x, t):
        self.loss(x, t)
        dout = 1
        dout = self.last_layer.backward(dout)
        layers = list(self.layers.values())
        layers.[      ]
        for layer in layers:
            dout = layer.backward(dout)
        grads = {}
        for idx in range(1, self.hidden_layer_num+2):
            grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dW
                                    + [      ] * [      ]
            grads['b' + str(idx)] = self.layers['Affine' + str(idx)].db
        return grads
```

10. 다음은 Drop Out 층의 순전파와 역전파를 구현하는 코드이다. 빈칸을 채우시오.

```
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None
    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.[      ] ([      ] x.shape) > self.dropout_ratio
            return x [      ]
        else:
            return x [      ]
    def backward(self, dout):
        return dout [      ]
```

딥러닝 입문 / 기말고사(1반) 12. 8. 2020

1. 데이터 X가 $\mathrm{N} \times \mathrm{C} \times \mathrm{H} \times \mathrm{W} = 2 \times 2 \times 3 \times 3$ 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} & \begin{pmatrix} 2 & 1 & 0 \\ 2 & 1 & 0 \\ 2 & 1 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} & \begin{pmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \end{pmatrix}$$

로, 필터 W가 $\mathrm{FN} \times \mathrm{C} \times \mathrm{FH} \times \mathrm{FW} = 2 \times 2 \times 2 \times 2$ 텐서

$$W = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. no padding과 stride=1로 합성곱한 값을 구하시오.

2. (1번 문제 계속) 다음은 합성곱 클래스의 순전파 코드이다. 편향이 $b_1 = 1$, $b_2 = 2$로 주어져 있다. 밑줄 친 4개의 값을 구하시오.

```
def forward(self, x):
    FN, C, FH, FW = self.W.shape
    N, C, H, W = x.shape
    out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
    out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
    col = im2col(x, FH, FW, self.stride, self.pad)
    col_W = self.W.reshape(FN, -1).T
    out = np.dot(col, col_W) + self.b
    out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
    self.x = x
    self.col = col
    self.col_W = col_W
    return out
```

3. (1,2번 문제 계속) 주어진 합성곱층을 Affine층으로 바꾸시오. 즉 , 입력 노드, 출력 노드, 에지를 그리고 에지위에 가중치를 표시하고 가중치 행렬과 편향 벡터를 구하고 데이터를 나열하시오.

4. (1,2번 문제 계속) 다음은 합성곱 클래스의 역전파 코드이다. 흘러 들어온 미분이

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix}$$

일 때, 밑줄친 6개의 값을 구하시오.

```
def backward(self, dout):
    FN, C, FH, FW = self.W.shape
```

```
        dout = dout.transpose(0,2,3,1).reshape(-1, FN)
        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
        return dx
```

5. 데이터 X가 $\mathrm{N} \times \mathrm{C} \times \mathrm{H} \times \mathrm{W} = 2 \times 2 \times 3 \times 3$ 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 1 & 0 & 2 \\ 7 & 3 & 8 \\ 6 & 5 & 4 \end{pmatrix} & \begin{pmatrix} 8 & 0 & 1 \\ 2 & 7 & 3 \\ 4 & 5 & 6 \end{pmatrix} \\ \begin{pmatrix} 5 & 1 & 4 \\ 2 & 0 & 3 \\ 8 & 6 & 7 \end{pmatrix} & \begin{pmatrix} 5 & 6 & 7 \\ 3 & 8 & 4 \\ 0 & 1 & 2 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. PH=2, PW=2, padding=0, stride=1로 Max Pooling한 값을 구하시오.

6. (5번 문제 계속) 다음은 Max Pooling 클래스의 순전파 코드이다. 밑줄친 4개의 값을 구하시오.

```
    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)
        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
        self.x = x
        self.arg_max = arg_max
        return out
```

7. (5번 문제 계속) Max Pooling층으로 흘러들어온 미분이

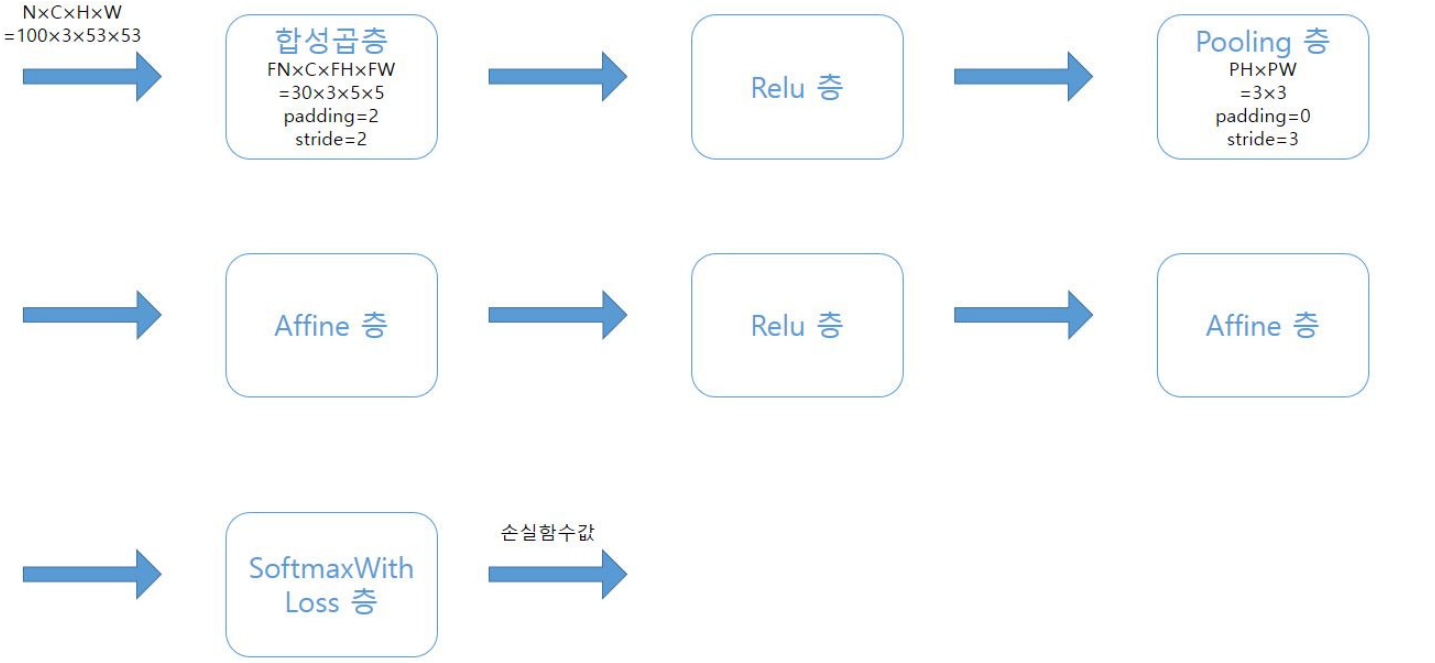
$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \end{pmatrix}$$

일 때, 데이터에 대한 미분

$$\frac{\partial L}{\partial X}$$

을 구하시오.

8. 합성곱 신경망이 다음과 같은 층으로 이루어져 있다. 은닉층의 뉴런수는 50개이고 출력층의 뉴런수는 10개이다. 53×53 해상도와 RGB 채널을 가지는 컬러사진 100장을 넣었을때 흘러가는 데이터의 shape을 화살표마다 순서대로 쓰시오. 첫번째 Affine층의 가중치 행렬의 shape을 구하시오.



9. 합성곱 신경망이

와 같이 주어져 있다. 합성곱층의 필터, 첫번째 Affine층의 가중치 행렬, 두번째 Affine층의 가중치 행렬이 각각

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 & \log 2 \\ \log 3 & 0 & 0 & 0 \\ 0 & \log 4 & 0 & 0 \\ 0 & 0 & \log 5 & 0 \end{pmatrix}$$

으로 주어져 있다. 편향은 모두 없다. 합성곱층에서는 stride=1, padding=0이고 Max Pooling층에서는 PH=PW=2, stride=2, padding=0이다. 데이터

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

의 라벨이 $[0, 0, 0, 1]$일 때, 손실함수 값을 구하시오.

10. transpose(0,3,1,2)의 역전파를 구하시오.

딥러닝 입문 / 기말고사(2반) 12. 9. 2020

1. 데이터 X가 N × C × H × W = 2 × 2 × 3 × 3 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 3 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 3 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{pmatrix}$$

로, 필터 W가 FN × C × FH × FW = 2 × 2 × 2 × 2 텐서

$$W = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. padding=0와 stride=1로 합성곱한 값을 구하시오.

2. (1번 문제 계속) 다음은 합성곱 클래스의 순전파 코드이다. 편향이 $b_1 = -1$, $b_2 = -2$로 주어져 있다. 밑줄친 4개의 값을 구하시오.

```
def forward(self, x):
    FN, C, FH, FW = self.W.shape
    N, C, H, W = x.shape
    out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
    out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
    col = im2col(x, FH, FW, self.stride, self.pad)
    col_W = self.W.reshape(FN, -1).T
    out = np.dot(col, col_W) + self.b
    out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
    self.x = x
    self.col = col
    self.col_W = col_W
    return out
```

3. (1,2번 문제 계속) 다음은 합성곱 클래스의 역전파 코드이다. 흘러 들어온 미분이

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{pmatrix}$$

일 때, 밑줄친 6개의 값을 구하시오.

```
def backward(self, dout):
    FN, C, FH, FW = self.W.shape
    dout = dout.transpose(0,2,3,1).reshape(-1, FN)
    self.db = np.sum(dout, axis=0)
```

```
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
        return dx
```

4. 데이터 X가 N × C × H × W = 2 × 2 × 3 × 3 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 5 & 6 & 7 \\ 3 & 8 & 4 \\ 0 & 1 & 2 \end{pmatrix} & \begin{pmatrix} 4 & 0 & 5 \\ 1 & 2 & 3 \\ 7 & 6 & 8 \end{pmatrix} \\ \begin{pmatrix} 2 & 0 & 1 \\ 8 & 7 & 3 \\ 4 & 5 & 6 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 8 \\ 7 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. PH=2, PW=2, no padding, stride=1로 Max Pooling한 값을 구하시오.

5. (4번 문제 계속) 다음은 Max Pooling 클래스의 순전파 코드이다. 밑줄친 4개의 값을 구하시오.

```
    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)
        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
        self.x = x
        self.arg_max = arg_max
        return out
```

6. (4번 문제 계속) Max Pooling층으로 흘러들어온 미분이

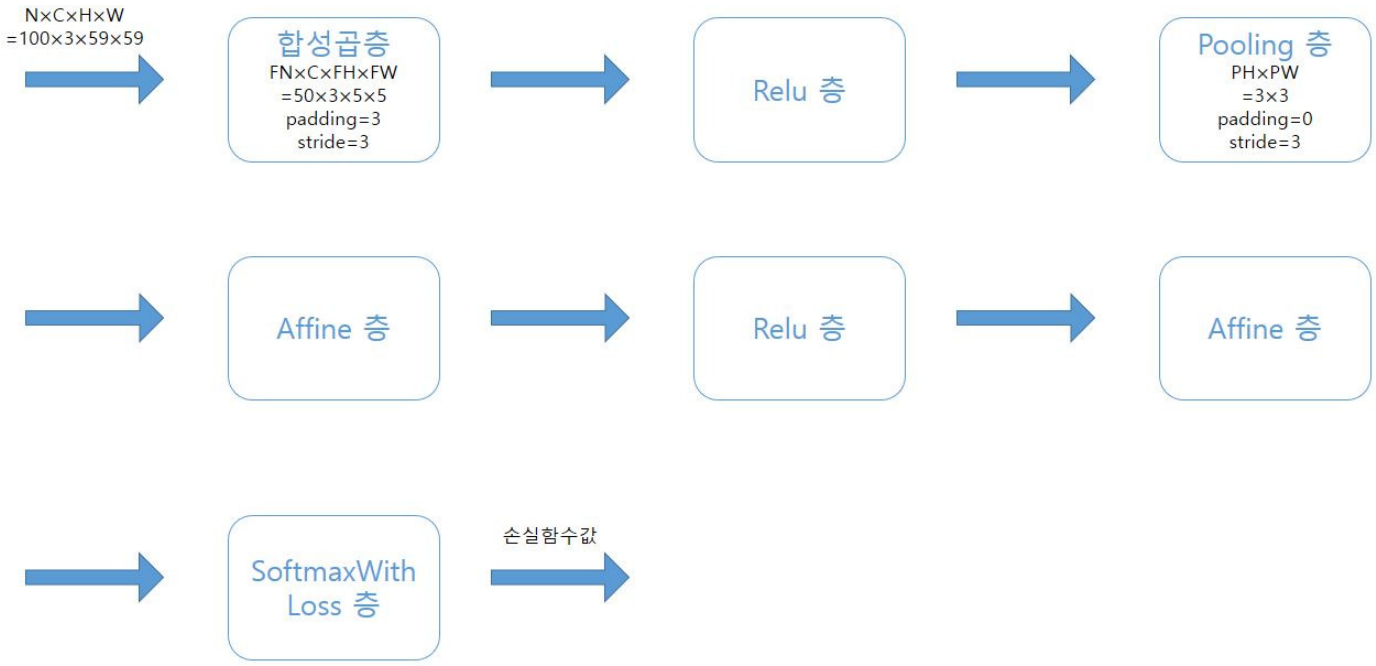
$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \end{pmatrix}$$

일 때, 데이터에 대한 미분

$$\frac{\partial L}{\partial X}$$

을 구하시오.

7. 합성곱 신경망이 다음과 같은 층으로 이루어져 있다. 은닉층의 뉴런수는 100개이고 출력층의 뉴런수는 10개이다. 59×59 해상도와 RGB 채널을 가지는 컬러사진 100장을 넣었을때 흘러가는 데이터의 shape을 화살표마다 순서대로 쓰시오. 첫번째 Affine층의 가중치 행렬의 shape을 구하시오.



8. (7번 문제 계속) He 초기값을 따라 필터와 가중치 행렬을 초기화하려 한다. 합성곱층의 필터 W_1, 첫번째 Affine층의 가중치 행렬 W_2, 두번째 Affine층의 가중치 행렬 W_3를 각각 어떤 분포를 따라 랜덤하게 초기화해야 하는가?

9. 합성곱 신경망이

와 같이 주어져 있다. 합성곱층의 필터, 첫번째 Affine층의 가중치 행렬, 두번째 Affine층의 가중치 행렬이 각각

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 & \log 5 \\ 0 & 0 & \log 4 & 0 \\ 0 & \log 3 & 0 & 0 \\ \log 2 & 0 & 0 & 0 \end{pmatrix}$$

으로 주어져 있다. 편향은 모두 없다. 합성곱층에서는 stride=1, padding=0이고 Max Pooling층에서는 PH=PW=2, stride=2, padding=0이다. 데이터

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

의 라벨이 $[0, 1, 0, 0]$일 때, 손실함수 값을 구하시오.

10. 다음 코드를 실행했을때 출력되는 3개의 이미지를 순서대로 그리시오.

```
img=np.zeros((100,100))
img[:50,:50]=1.
plt.imshow(img, cmap=plt.cm.gray)
plt.show()

w1=np.array([[[[1,2,1],[0,0,0],[-1,-2,-1]]]])
b=0
conv_layer1 = Convolution(w1, b)
img = img.reshape(1, 1, *img.shape)
out = conv_layer1.forward(img)
out = out.reshape(out.shape[2], out.shape[3])
plt.imshow(out, cmap=plt.cm.gray)
plt.show()

w2=np.array([[[[1,0,-1],[2,0,-2],[1,0,-1]]]])
conv_layer2 = Convolution(w2, b)
out = conv_layer2.forward(img)
out = out.reshape(out.shape[2], out.shape[3])
plt.imshow(out, cmap=plt.cm.gray)
plt.show()
```

딥러닝 입문 / 기말고사(3반) 12. 11. 2020

1. 데이터 X가 N × C × H × W = 2 × 2 × 3 × 3 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 3 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 3 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 2 & 1 \end{pmatrix} \end{pmatrix}$$

로, 필터 W가 FN × C × FH × FW = 2 × 2 × 2 × 2 텐서

$$W = \begin{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. padding=0와 stride=1로 합성곱한 값을 구하시오.

2. (1번 문제 계속) 다음은 합성곱 클래스의 순전파 코드이다. 편향이 $b_1 = 2$, $b_2 = 1$로 주어져 있다. 밑줄 친 4개의 값을 구하시오.

```
def forward(self, x):
    FN, C, FH, FW = self.W.shape
    N, C, H, W = x.shape
    out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
    out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
    col = im2col(x, FH, FW, self.stride, self.pad)
    col_W = self.W.reshape(FN, -1).T
    out = np.dot(col, col_W) + self.b
    out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
    self.x = x
    self.col = col
    self.col_W = col_W
    return out
```

3. (1,2번 문제 계속) 다음은 합성곱 클래스의 역전파 코드이다. 흘러 들어온 미분이

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{pmatrix}$$

일 때, 밑줄친 6개의 값을 구하시오.

```
def backward(self, dout):
    FN, C, FH, FW = self.W.shape
    dout = dout.transpose(0,2,3,1).reshape(-1, FN)
    self.db = np.sum(dout, axis=0)
```

```
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
        dcol = np.dot(dout, self.col_W.T)
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
        return dx
```

4. 데이터 X가 N × C × H × W = 2 × 2 × 3 × 3 텐서

$$X = \begin{pmatrix} \begin{pmatrix} 7 & 0 & 3 \\ 5 & 2 & 1 \\ 6 & 4 & 8 \end{pmatrix} & \begin{pmatrix} 7 & 6 & 5 \\ 3 & 8 & 4 \\ 0 & 2 & 1 \end{pmatrix} \\ \begin{pmatrix} 2 & 0 & 6 \\ 8 & 7 & 5 \\ 4 & 3 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 6 & 3 \\ 1 & 2 & 5 \\ 4 & 8 & 7 \end{pmatrix} \end{pmatrix}$$

로 주어져 있다. PH=2, PW=2, no padding, stride=1로 Max Pooling한 값을 구하시오.

5. (4번 문제 계속) 다음은 Max Pooling 클래스의 순전파 코드이다. 밑줄친 4개의 값을 구하시오.

```
    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)
        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
        self.x = x
        self.arg_max = arg_max
        return out
```

6. (4번 문제 계속) Max Pooling층으로 흘러들어온 미분이

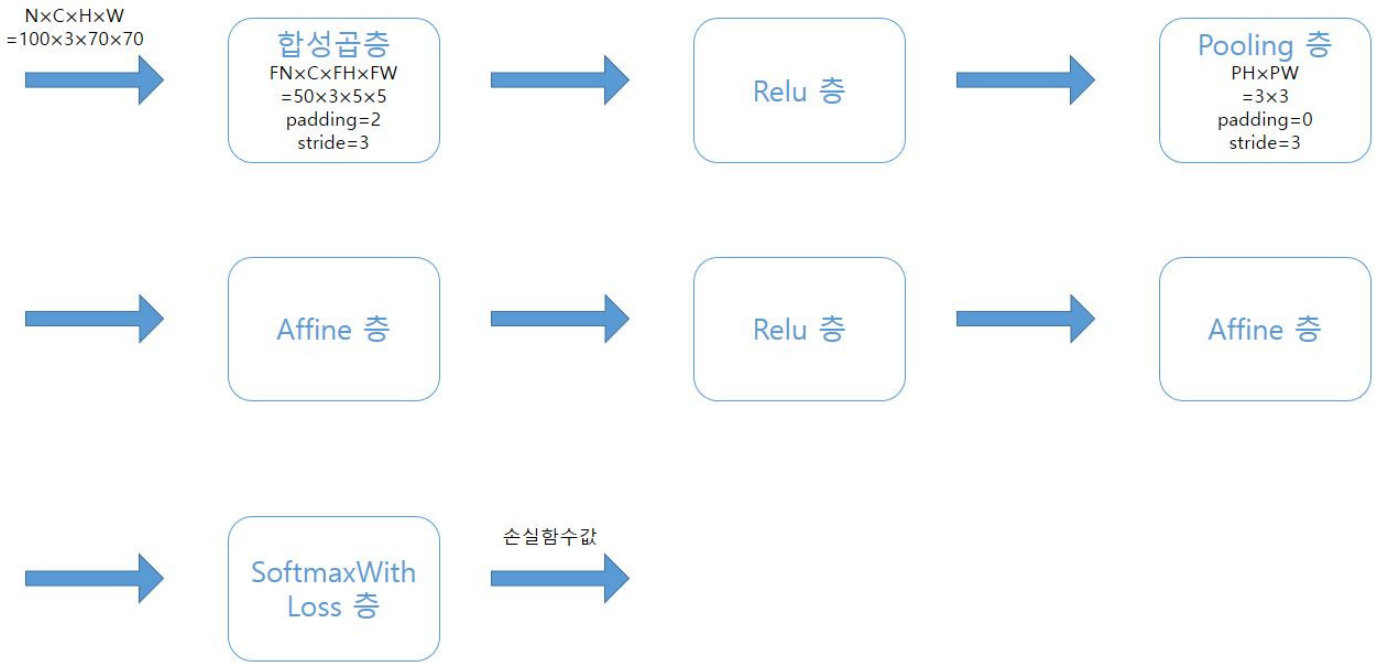
$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \end{pmatrix}$$

일 때, 데이터에 대한 미분

$$\frac{\partial L}{\partial X}$$

을 구하시오.

7. 합성곱 신경망이 다음과 같은 층으로 이루어져 있다. 은닉층의 뉴런수는 80개이고 출력층의 뉴런수는 10개이다. 70×70 해상도와 RGB 채널을 가지는 컬러사진 100장을 넣었을때 흘러가는 데이터의 shape을 화살표마다 순서대로 쓰시오. 첫번째 Affine층의 가중치 행렬의 shape을 구하시오.



8. (7번 문제 계속) He 초기값을 따라 필터와 가중치 행렬을 초기화하려 한다. 합성곱층의 필터 W_1, 첫번째 Affine층의 가중치 행렬 W_2, 두번째 Affine층의 가중치 행렬 W_3를 각각 어떤 분포를 따라 랜덤하게 초기화해야 하는가?

9. 합성곱 신경망이

와 같이 주어져 있다. 합성곱층의 필터 W_1, 첫번째 Affine층의 가중치 행렬 W_2, 두번째 Affine층의 가중치 행렬 W_3가 각각

$$W_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad W_2 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \qquad W_3 = \begin{pmatrix} 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

으로 주어져 있다. 편향은 모두 없다. 합성곱층에서는 stride=1, padding=0이고 Max Pooling층에서는 PH=PW=2, stride=1, padding=0이다. 데이터

$$\begin{pmatrix} 2 & 0 & 0 & 2 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix}$$

의 라벨이 $[0, 0, 0, 1]$일 때, 손실함수 값을 구하시오.

10. (9번 문제 계속) 역전파를 통하여 세 gradient

$$\frac{\partial L}{\partial W_3}, \qquad \frac{\partial L}{\partial W_2}, \qquad \frac{\partial L}{\partial W_1}$$

를 구하시오.