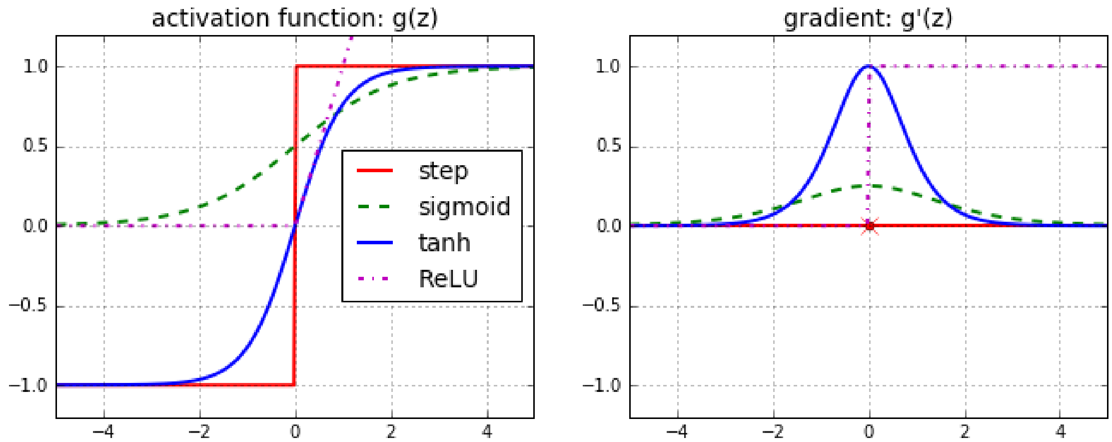
Assignment2

October 9, 2018

1. [활성함수] 다음 코드는 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (4점)

In [1]:
```
# Python 2, Python 3 지원
from __future__ import division, print_function, unicode_literals

# Library related
import os
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt


# sigmoid function
def logit(z):
    return 1 /(1 + np.exp(-z))

# ReLU Function
def relu(z):
    return np.maximum(0,z)

# deriavtive formula
# 미분 공식
def derivative(f, z, eps=0.000001):
    return (f(z + eps) - f(z - eps))/(2*eps)

# sample data
# numpyt linspace's reference
# https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/-
# numpy.linspace.html
# this function return evenly spaced numbers over a specificed interval
z = np.linspace(-5, 5, 200)

# figure function's reference is
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.figure.html
# figsize width, height in inches
plt.figure(figsize=(11,4))
```

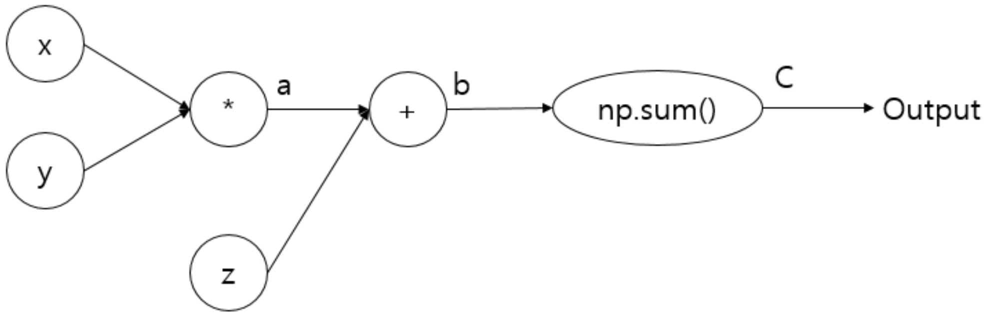
```python
# The left figure is functions like sign, logit(sigmoid), tanh, relu
# activation function figure
# the maplot's reference is
# https://matplotlib.org/api/_as_gen/matplotlib.pyplot.subplot.html
# the matplot's subplot function
# i.e. Either a 3-digit integer or three separate integers
# describing the position of the subplot.
# the three integers are nrow, ncol, and index in order.
plt.subplot(121)
plt.plot(z, np.sign(z), "r-", linewidth=2, label="step")
plt.plot(z, logit(z), "g--", linewidth=2, label="sigmoid")
plt.plot(z, np.tanh(z), "b-", linewidth=2, label="tanh")
plt.plot(z, relu(z), "m-.", linewidth=2, label="ReLU")
plt.grid(True)
plt.legend(loc="center right", fontsize=14)
plt.title("activation function: g(z)", fontsize=14)
plt.axis([-5, 5, -1.2, 1.2])

# The right figure is derivative function of each function above
# 각 activation function's 기울기 함수(미분함수)
plt.subplot(122)
plt.plot(z, derivative(np.sign, z), "r-", linewidth=2, label="step")
plt.plot(0, 0, "ro", markersize=5)
plt.plot(0, 0, "rx", markersize=10)
plt.plot(z, derivative(logit, z),  "g--", linewidth=2,  label="sigmoid")
plt.plot(z, derivative(np.tanh, z), "b-", linewidth=2, label="tanh")
plt.plot(z, derivative(relu, z), "m-.", linewidth=2, label="ReLU")
plt.grid(True)
plt.title("gradient: g'(z)", fontsize=14)
plt.axis([-5, 5, -1.2, 1.2])

plt.show()
# (1) 화면 출력 확인 및 각 활성함수의 특징을 비교 서술
# step function: 계단 함수로 항상 기울기는 0, 단 원점에 미분 불가능하지만
#                여기서 원점의 미분값은 0으로 표시
# sigmoid : 0 ~ 1 사이 값을 출력을 하지만 미분 값은 원점에 최대
#           양쪽 끝(음수,양수)으로 갈수록 미분은 값은 0이다.
# tanh : sigmoid function가 비슷하지만, 출력값이 -1 ~ 1 사이가 다르다.
#        하지만 양쪽 끝(음수, 양수)으로 갈수록 미분값은 0인것은 같다.
# ReLu : 양수에서는 기울기가 1이고 음수에서는 출력값이외에 기울기도 0이다.
# 각 activation function은 비선형적인 특징을 가지고 있다. 하지만
# gradient가 다르고 포화되는 위치 gradient의 크기가 제 각각이다.
```



2. [오류 역전파] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (4점)
(코드의 해서과 결과의 의미를 작성하세요.)

```
In [2]: import numpy as np
        # numpy's random seed
        # https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html
        np.random.seed(0)

        N, D = 3, 4

        # numpy's random rand
        # https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.randn.html
        x = np.random.randn(N, D)
        y = np.random.randn(N, D)
        z = np.random.randn(N, D)

        # in numpy, the sign like * + is elementwise
        a = x * y
        b = a + z
        c = np.sum(b)
        # (1) 해당 연산망의 그래프 연산을 손으로 작성
        # 아래의 그림처럼 단순 입력 값을 연산 순서에 맞게 위의 코드는
        # 계산을 한다.
```

```
In [3]: grad_c = 1.0
        grad_b = grad_c * np.ones((N,D))
        grad_a = grad_b.copy()
        grad_z = grad_b.copy()
        grad_x = grad_a*y
        grad_y = grad_a*x
        # (2) grad_c, grad_b, grad_a, grad_z, grad_x, grad_y 출력 확인
```




```
# 아래의 경우 error에 대한 backpropagation을 위한  gradient를 계산한 결과이다.
print("grad_c:\n{}".format(grad_c))
print("grad_b:\n{}".format(grad_b))
print("grad_a:\n{}".format(grad_a))
print("grad_z:\n{}".format(grad_z))
print("grad_x:\n{}".format(grad_x))
print("grad_y:\n{}".format(grad_y))
```

```
grad_c:
1.0
grad_b:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
grad_a:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
grad_z:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
grad_x:
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186   0.8644362  -0.74216502]]
grad_y:
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
 [-0.10321885  0.4105985   0.14404357  1.45427351]]
```

```
In [4]: import torch

        # The Reference of torch.randn(N, D) is normal distribution
```

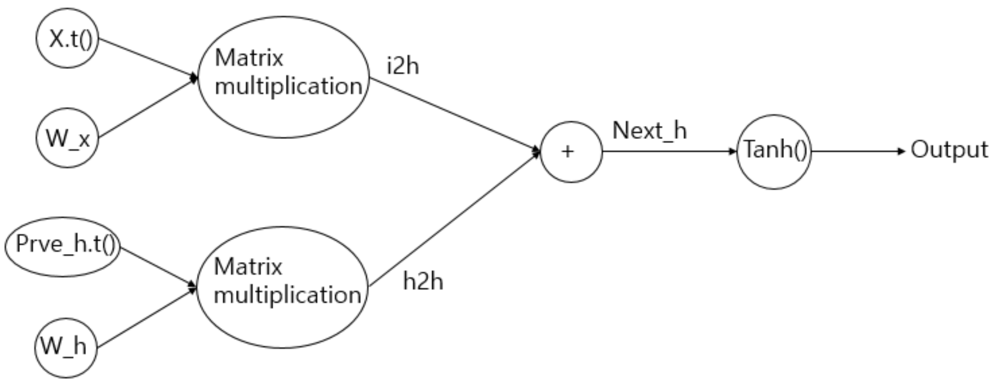
```
    # about mean equal to 0, variance 1.
    # https://pytorch.org/docs/stable/torch.html#torch.randn
    x = torch.randn(N, D, requires_grad = True)
    y = torch.randn(N, D, requires_grad = True)
    z = torch.randn(N, D)

    a = x * y
    b = a + z
    c = torch.sum(b)

    c.backward()
    # (3) grad_x, grad_y 출력확인
    print("grad_x:\n{}".format(x.grad))
    print("grad_y:\n{}".format(y.grad))

    # 아래의 출력결과는 torch를 활용한 gradient 결과이다.
```

```
grad_x:
tensor([[ 1.0741,  1.9732,  1.4140, -1.0282],
        [-1.1895, -1.1022, -0.5007, -1.9635],
        [-0.2747, -1.3813,  0.5262,  1.0943]])
grad_y:
tensor([[ 0.4448, -0.1613, -0.2329, -0.4336],
        [ 0.5539,  1.0458,  0.5737,  0.9508],
        [ 2.2725, -0.7400, -0.6197,  1.4848]])
```

3. [오류 역전파] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (4점)
   (코드의 해석과 결과의 의미를 작성하세요.)

```
In [5]: import torch

        x = torch.randn(1, 10)
        prev_h = torch.randn(1, 20)
        w_h = torch.randn(20, 20,requires_grad = True)
        w_x = torch.randn(20, 10, requires_grad = True)

        # for debugging
        #print("x:\n{}".format(x))
        #print("prev_h:\n{}".format(prev_h))
        #print("w_h:\n{}".format(w_h))
        #print("w_x:\n{}".format(w_x))

        # The Reference of torch.mm is
        # https://pytorch.org/docs/stable/torch.html?highlight=torch%20mm#torch.mm
        # it means performing a matrix multiiplication of the matrixes, torch.mm(mat1, mat2)
        i2h = torch.mm(w_x, x.t())
        h2h = torch.mm(w_h, prev_h.t())
```

```
next_h = i2h + h2h
next_h = next_h.tanh()
# (1) 해당 신경망의 그래프를 연산을 손으로 작성

# 아래의 그림을 벡터화 연산을 단순한 연산 그래프로 표현을 한 것이다.
```




```
In [6]: loss = next_h.sum()

        # This backpropagation.
        loss.backward()
        # (2) loss 출력확인
        print("loss:\n{}".format(loss))

        # 위의 연산 그래프를 기반으로 첫번째 연산 결과의 값을 출력을 하고 있다.
```

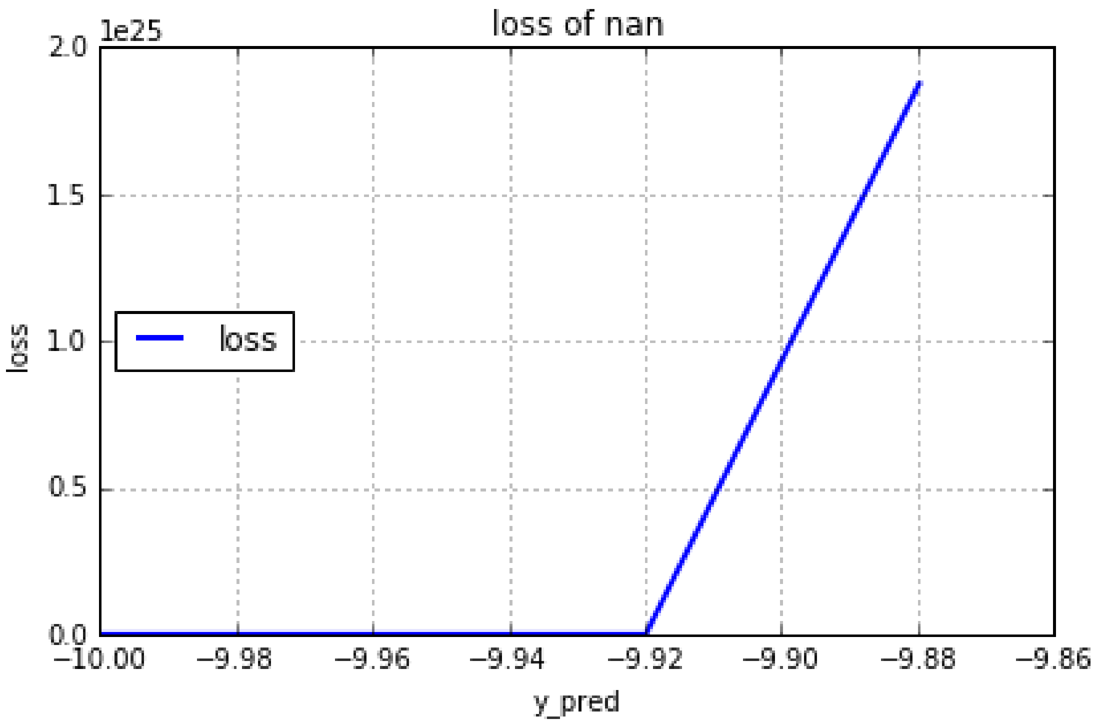
```
loss:
3.5939688682556152
```

4. [신경망 학습] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요.(4점) (코드의 해석화 결과의 의미를 작성하세요.)

```
In [7]: import torch
        # (1)의 그래프를 위한 library
        import numpy as np
        %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt

        N, D_in, H, D_out = 64, 1000, 100, 10

        x = torch.randn(N, D_in)
```

```python
# In the case below, I fixed it from D to N, wrong letter D in original assignment.
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H)
w2 = torch.randn(H, D_out)

learning_rate = 10e-6

loss1 = []
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()
    # (1)의 출력을 위해
    loss1.append(loss)

    grad_y_pred = 2.0*(y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h <0] = 0
    grad_w1 = x.t().mm(grad_h)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

# If you want to know how to use matplot
# reference is
# https://matplotlib.org/tutorials/introductory/usage.html-
# #sphx-glr-tutorials-introductory-usage-py
# (1) y_pred에 따른 loss(accuracy) 변화를 화면 출력확인 (plot)

# x_array is a range of y_pred
x_array = np.linspace(-10, 10, len(loss1))

loss_array = np.asarray(loss1)
plt.plot(x_array, loss_array, "b-", linewidth=2, label="loss")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss of nan")

plt.legend(loc="center left")
plt.tight_layout()

plt.show()
```

```
print(loss_array[:10])
# (2) 해당 학습이 적절히 진행되고 있는지 서술
# 그래프 아래의 loss일부분을 출력 결과들에서 볼 수 있듯이
# loss의 값은 inf에서 nan으로 되는 것을 확인할 수있다.
# 즉, 제대로 학습이 진행이 되지 않고 loss가 NAN(not a number) 되고 있다.
```

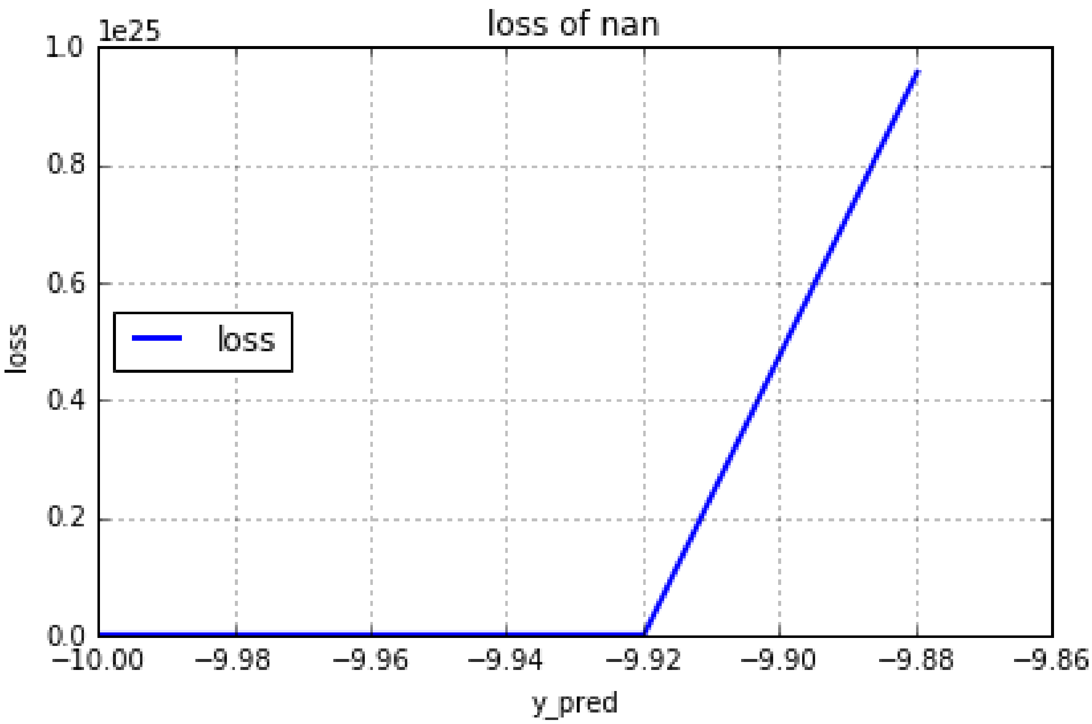



```
[3.7456420e+07 4.3933655e+09 2.1102670e+13 1.8754189e+25             inf
             nan             nan             nan             nan             nan]
```

5. [신경망 학습] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요.(4점) (코드의 해서과 결과의 의미를 작성하세요.)

In [8]:
```
import torch
# (1)의 그래프를 위한 library
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt


N, D_in, H, D_out = 64, 1000, 100, 10

x = torch.randn(N, D_in)
```

```python
# In the case below, I fixed it from D to N, wrong letter D in original assignment.
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad = True)
w2 = torch.randn(H, D_out, requires_grad = True)


loss2 = []
learning_rate = 10e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred-y).pow(2).sum()
    loss2.append(loss)
    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()


# (1) 매 t마다 y_pred, loss 변화를 화면 출력확인(plot)
# x_array is a range of y_pred
x_array = np.linspace(-10, 10, len(loss2))

loss_array = np.asarray(loss2)
plt.plot(x_array, loss_array, "b-", linewidth=2, label="loss")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss of nan")

plt.legend(loc="center left")
plt.tight_layout()

plt.show()
print(loss_array[:10])
# (2) 앞 문제의 코드와 비교
# 이경우에도 앞의 코드와 같은 INF(큰값)을 출력하고
# 결국 하드웨어가 표현 할 수 있는 숫자의 범위를 넘어서고 있어서
# NAN(not a nnumber)로 제대로 학습이 되지 않는다.
```



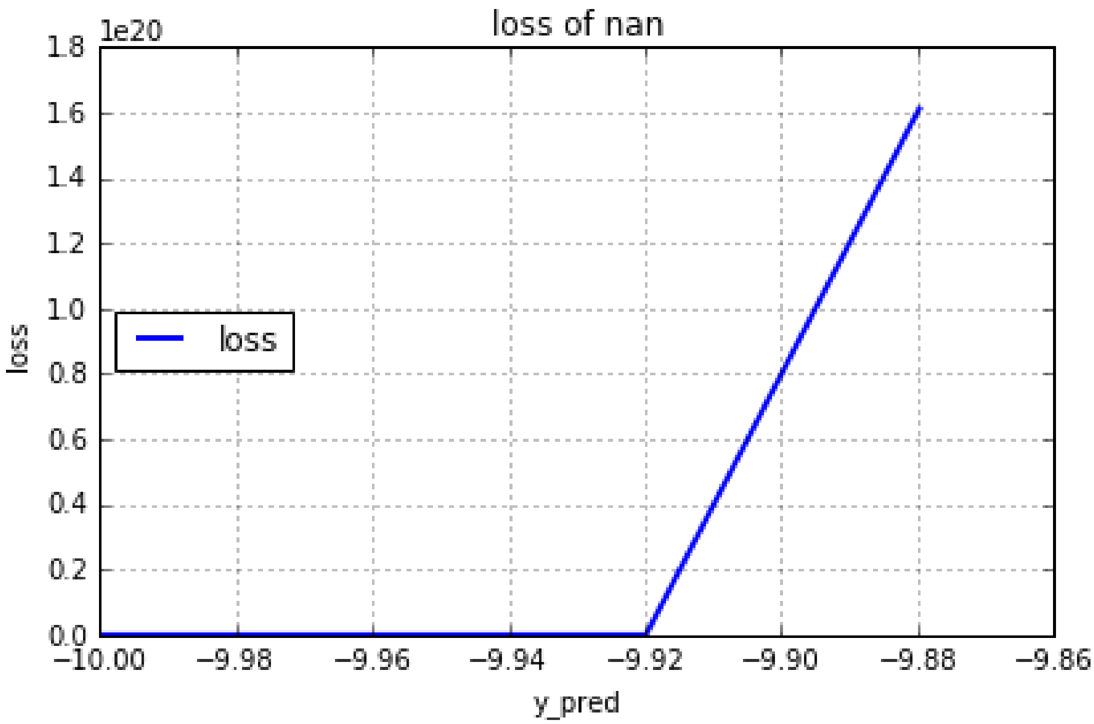

```
[tensor(35326296., grad_fn=<SumBackward0>)
 tensor(3963422720., grad_fn=<SumBackward0>)
 tensor(16727616258048., grad_fn=<SumBackward0>)
 tensor(95750851533539020636160000., grad_fn=<SumBackward0>)
 tensor(inf, grad_fn=<SumBackward0>) tensor(nan, grad_fn=<SumBackward0>)
 tensor(nan, grad_fn=<SumBackward0>) tensor(nan, grad_fn=<SumBackward0>)
 tensor(nan, grad_fn=<SumBackward0>) tensor(nan, grad_fn=<SumBackward0>)]
```

6. [신경망 학습] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (4 점) (코드의 해석과 결과의 의미를 작성하세요.)

In [9]:
```
# if you want to kwon processing below
# refer to https://pytorch.org/tutorials/beginner/-
# pytorch_with_examples.html#pytorch-tensors-and-autograd

import torch
# (1)의 그래프를 위한 library
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

```python
class MyReLU(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x):
        ctx.save_for_backward(x)
        return x.clamp(min=0)

    @staticmethod
    def backward(ctx, grad_y):
        x, = ctx.saved_tensors
        grad_input = grad_y.clone()
        grad_input[x<0] = 0
        return grad_input

def my_relu(x):
    return MyReLU.apply(x)

N, D_in, H, D_out = 64, 1000, 100, 10

x = torch.randn(N, D_in)
# In the case below, I fixed it from D to N, wrong letter D in original assignment.
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad = True)
w2 = torch.randn(H, D_out, requires_grad = True)

learning_rate = 10e-6


loss3 = []
for t in range(500):
    y_pred = my_relu(x.mm(w1)).mm(w2)
    loss = (y_pred - y).pow(2).sum()
    loss3.append(loss)

    # Use autograd to compute the backward pass. This call will compute the
    # gragident of loss with respect to all Tensors with requires_grad=True.
    # After this call w1.grad and w2.grad will be Tensors holding the gradient
    # of the loss with respect to w1 and w1 respectively.
    loss.backward()

    # Manually update weights using gradient descent. Wrap in torch.no_grad()
    # because weights hvae requires_grad=True, but we don't need to track this
    # in autograd
    # An alternaive way is to operate on weight.data and weight.grad.data.
    # Recall that tensor.data gives a tensor that shares the storage with
    # tensor, but doesn't track history.
    # You can also use torch.optim.SGD to achieve this.
    with torch.no_grad():
```

```
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad

        # Munually zero the gradient after updating weights.
        w1.grad.zero_()
        w2.grad.zero_()

# (1) 매 t마다 y_pred, loss 변화를 화면 출력 확인(Plot)
# x_array is a range of y_pred
x_array = np.linspace(-10, 10, len(loss3))

loss_array = np.asarray(loss3)
plt.plot(x_array, loss_array, "b-", linewidth=2, label="loss")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss of nan")

plt.legend(loc="center left")
plt.tight_layout()

plt.show()
print(loss_array[:10])
# (2) 앞 문제의 코드와 비교
# 이 문제도 앞의 코드와 같이
# NAN(not a number) 문제가 발생하여
# 제대로 학습이 되지 않는다.
```




```
[tensor(28107224., grad_fn=<SumBackward0>)
 tensor(1546043264., grad_fn=<SumBackward0>)
 tensor(481758478336., grad_fn=<SumBackward0>)
 tensor(161188738883496443904., grad_fn=<SumBackward0>)
 tensor(inf, grad_fn=<SumBackward0>) tensor(inf, grad_fn=<SumBackward0>)
 tensor(nan, grad_fn=<SumBackward0>) tensor(nan, grad_fn=<SumBackward0>)
 tensor(nan, grad_fn=<SumBackward0>) tensor(nan, grad_fn=<SumBackward0>)]
```

7. [신경망 학습] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요.(4점) (코드의 해석과 결과의 의미를 작성하세요.)

```
In [10]: # -*- coding: utf-8 -*-
         import torch


         # (1)의 그래프를 위한 library
         import numpy as np
         %matplotlib inline
         import matplotlib
         import matplotlib.pyplot as plt

         class TwoLayerNet(torch.nn.Module):
             def __init__(self, D_in, H, D_out):
```

```
        """
        In the constructor we instantiate two nn.Linear modules and assign them as
        member variables.
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Tensor of input data and we must return
        a Tensor of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Tensors.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred


# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold inputs and outputs
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Construct our model by instantiating the class defined above
model = TwoLayerNet(D_in, H, D_out)

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)

loss4 = []
for t in range(500):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x)

    # Compute and print loss
    loss = criterion(y_pred, y)
    loss4.append(loss)
    print(t, loss.item())

    # optimizer.zero_grad()
```

```
        # Backward pass: compute gradient of the loss with respect to model
        # parameters
        loss.backward()

        # Calling the step function on an Optimizer makes an update to its
        # parameters
        optimizer.step()
        # This is code error
        # Before the backward pass, use the optimizer object to zero all of the
        # gradients for the variables it will update (which are the learnable
        # weights of the model). This is because by default, gradients are
        # accumulated in buffers( i.e, not overwritten) whenever .backward()
        # is called. Checkout docs of torch.autograd.backward for more details.
        optimizer.zero_grad()
```

```
0 752.291259765625
1 697.1152954101562
2 649.825439453125
3 608.62841796875
4 572.4631958007812
5 540.3025512695312
6 511.09442138671875
7 484.2495422363281
8 459.5650634765625
9 436.6903076171875
10 415.0898742675781
11 394.4296875
12 374.9518737792969
13 356.33203125
14 338.6272888183594
15 321.73406982421875
16 305.5383605957031
17 290.00274658203125
18 275.1201171875
19 260.8442077636719
20 247.1610870361328
21 234.00474548339844
22 221.4241943359375
23 209.39483642578125
24 197.90155029296875
25 186.92022705078125
26 176.44859313964844
27 166.4690399169922
28 156.9864044189453
29 147.99073791503906
30 139.47186279296875
31 131.36688232421875
```

32 123.66256713867188
33 116.37864685058594
34 109.49337768554688
35 102.98854064941406
36 96.83961486816406
37 91.03842163085938
38 85.57056427001953
39 80.41991424560547
40 75.56986236572266
41 70.99664306640625
42 66.68790435791016
43 62.64469909667969
44 58.84621047973633
45 55.281681060791016
46 51.93621063232422
47 48.79524230957031
48 45.84355163574219
49 43.068058013916016
50 40.46259307861328
51 38.01936340332031
52 35.72951126098633
53 33.58113479614258
54 31.563623428344727
55 29.669754028320312
56 27.89153289794922
57 26.223913192749023
58 24.660503387451172
59 23.193754196166992
60 21.816938400268555
61 20.527132034301758
62 19.316198348999023
63 18.180282592773438
64 17.114545822143555
65 16.116180419921875
66 15.178633689880371
67 14.297553062438965
68 13.472288131713867
69 12.698366165161133
70 11.971505165100098
71 11.288501739501953
72 10.647683143615723
73 10.04599380493164
74 9.480554580688477
75 8.949334144592285
76 8.450098037719727
77 7.9803547859191895
78 7.538464069366455
79 7.122349739074707

80 6.730738639831543
81 6.362541675567627
82 6.01649284362793
83 5.690637111663818
84 5.383955478668213
85 5.094628810882568
86 4.821960926055908
87 4.565117835998535
88 4.323069095611572
89 4.094974040985107
90 3.879796028137207
91 3.6769142150878906
92 3.4854280948638916
93 3.304979085922241
94 3.1346538066864014
95 2.973694086074829
96 2.821730852127075
97 2.678217649459839
98 2.542536735534668
99 2.4146065711975098
100 2.2938883304595947
101 2.1795334815979004
102 2.071465492248535
103 1.96928870677948
104 1.8726366758346558
105 1.7810378074645996
106 1.69365656375885
107 1.6112138032913208
108 1.533058524131775
109 1.4590877294540405
110 1.3890352249145508
111 1.322710633277893
112 1.2598209381103516
113 1.2003252506256104
114 1.1438922882080078
115 1.090371012687683
116 1.0396183729171753
117 0.9913878440856934
118 0.9456302523612976
119 0.9022039771080017
120 0.8609762191772461
121 0.8217774629592896
122 0.7845546007156372
123 0.7491856217384338
124 0.7155372500419617
125 0.6835476160049438
126 0.653113842010498
127 0.6241589784622192

128 0.5966277122497559
129 0.5704209804534912
130 0.545486569404602
131 0.5217181444168091
132 0.49908098578453064
133 0.4775305986404419
134 0.4570139944553375
135 0.4374428987503052
136 0.4187891185283661
137 0.40093451738357544
138 0.38387614488601685
139 0.36761897802352905
140 0.3521140515804291
141 0.337334543466568
142 0.32322362065315247
143 0.3097476661205292
144 0.2968858778476715
145 0.28460222482681274
146 0.2728690505027771
147 0.26161983609199524
148 0.25086548924446106
149 0.24059626460075378
150 0.23078703880310059
151 0.2214154154062271
152 0.21244877576828003
153 0.2038876861333847
154 0.19571097195148468
155 0.18786969780921936
156 0.18036597967147827
157 0.17319488525390625
158 0.16633135080337524
159 0.15975889563560486
160 0.15346774458885193
161 0.14744260907173157
162 0.14166943728923798
163 0.13614171743392944
164 0.130851149559021
165 0.12577968835830688
166 0.120914988219738
167 0.11625484377145767
168 0.11179511249065399
169 0.10751646012067795
170 0.10341358929872513
171 0.09947247058153152
172 0.09569189697504044
173 0.09206648916006088
174 0.08859425038099289
175 0.08525854349136353

176	0.08206245303153992
177	0.0789947509765625
178	0.07605184614658356
179	0.07322555035352707
180	0.07051275670528412
181	0.06791072338819504
182	0.06541197746992111
183	0.06301163136959076
184	0.06070190668106079
185	0.05848217010498047
186	0.056349512189626694
187	0.05429946631193161
188	0.05232948064804077
189	0.05043783783912659
190	0.04861617088317871
191	0.046864841133356094
192	0.04518112167716026
193	0.04356203228235245
194	0.04200228676199913
195	0.040503326803445816
196	0.039059825241565704
197	0.03767260164022446
198	0.03633783385157585
199	0.03505109250545502
200	0.0338127575814724
201	0.03262173384428024
202	0.03147327899932861
203	0.030368724837899208
204	0.029306208714842796
205	0.028281748294830322
206	0.027294719591736794
207	0.02634439244866371
208	0.02542918175458908
209	0.024547602981328964
210	0.023697974160313606
211	0.022879736497998238
212	0.022090628743171692
213	0.021331502124667168
214	0.02059965953230858
215	0.019894061610102654
216	0.019213277846574783
217	0.018556758761405945
218	0.017923861742019653
219	0.017313187941908836
220	0.016724292188882828
221	0.016156407073140144
222	0.015609163790941238
223	0.015081255696713924

224	0.014571702107787132
225	0.014080161228775978
226	0.013605794869363308
227	0.013148442842066288
228	0.012707145884633064
229	0.012281342409551144
230	0.01187104918062687
231	0.011474324390292168
232	0.011091461405158043
233	0.010721806436777115
234	0.010365155525505543
235	0.010020757094025612
236	0.009688123129308224
237	0.009367326274514198
238	0.009057246148586273
239	0.008758004754781723
240	0.008469011634588242
241	0.008189859800040722
242	0.00792046170681715
243	0.007660719100385904
244	0.007409293204545975
245	0.007166366558521986
246	0.006931617856025696
247	0.00670536607503891
248	0.006486467085778713
249	0.006274798419326544
250	0.006070316769182682
251	0.005872903857380152
252	0.005681970622390509
253	0.005497493781149387
254	0.005319148767739534
255	0.005146829877048731
256	0.00498028052970767
257	0.004819334018975496
258	0.004663754720240831
259	0.004513339605182409
260	0.004367979709059
261	0.004227399360388517
262	0.004091412294656038
263	0.003960027825087309
264	0.003833097405731678
265	0.0037105721421539783
266	0.0035917649511247873
267	0.0034768888726830482
268	0.003365803277119994
269	0.003258421318605542
270	0.003154542064294219
271	0.0030540882144123316

272	0.002956940094009042
273	0.0028630036395043135
274	0.0027721081860363483
275	0.002684163162484765
276	0.0025992023292928934
277	0.002516944659873843
278	0.0024373552296310663
279	0.0023603213485330343
280	0.0022858362644910812
281	0.002213814062997699
282	0.0021440591663122177
283	0.0020765981171280146
284	0.0020113822538405657
285	0.0019481619819998741
286	0.001886995742097497
287	0.001827788189984858
288	0.0017704913625493646
289	0.001715065911412239
290	0.001661372371017933
291	0.0016094220336526632
292	0.0015591736882925034
293	0.0015104946214705706
294	0.0014633937971666455
295	0.0014177658595144749
296	0.0013735952088609338
297	0.0013308963971212506
298	0.0012895024847239256
299	0.001249415334314108
300	0.0012107063084840775
301	0.001173154334537685
302	0.0011367989936843514
303	0.0011016631033271551
304	0.001067548873834312
305	0.0010345338378101587
306	0.0010025834199041128
307	0.0009716147324070334
308	0.0009416014654561877
309	0.0009125578799284995
310	0.0008844258263707161
311	0.0008571870275773108
312	0.0008308066753670573
313	0.0008052538032643497
314	0.0007805046625435352
315	0.0007565301493741572
316	0.0007333060493692756
317	0.0007108046556822956
318	0.0006890103104524314
319	0.0006679128273390234

320 0.0006474782130680978
321 0.0006276818457990885
322 0.0006085109198465943
323 0.0005899785901419818
324 0.000571983284316957
325 0.0005545387393794954
326 0.0005376457702368498
327 0.000521293084602803
328 0.0005054227658547461
329 0.0004900556523352861
330 0.00047514867037534714
331 0.0004607096780091524
332 0.00044672650983557105
333 0.00043317090603522956
334 0.0004200369294267148
335 0.00040730112232267857
336 0.00039496703539043665
337 0.0003830149071291089
338 0.0003714062040671706
339 0.0003601711068768054
340 0.00034929311368614435
341 0.000338727084454149
342 0.00032850296702235937
343 0.0003185929963365197
344 0.00030900989077053964
345 0.00029969087336212397
346 0.0002906531735789031
347 0.00028189513250254095
348 0.00027340836822986603
349 0.0002651837421581149
350 0.0002572030934970826
351 0.0002494784421287477
352 0.00024197388847824186
353 0.00023470990709029138
354 0.0002276556333526969
355 0.00022083417570684105
356 0.0002142011362593621
357 0.0002077867538901046
358 0.00020156106620561332
359 0.0001955278276000172
360 0.0001896794856293127
361 0.00018400164844933897
362 0.00017849694995675236
363 0.0001731569936964661
364 0.00016798521392047405
365 0.0001629735779715702
366 0.00015810449258424342
367 0.00015338975936174393

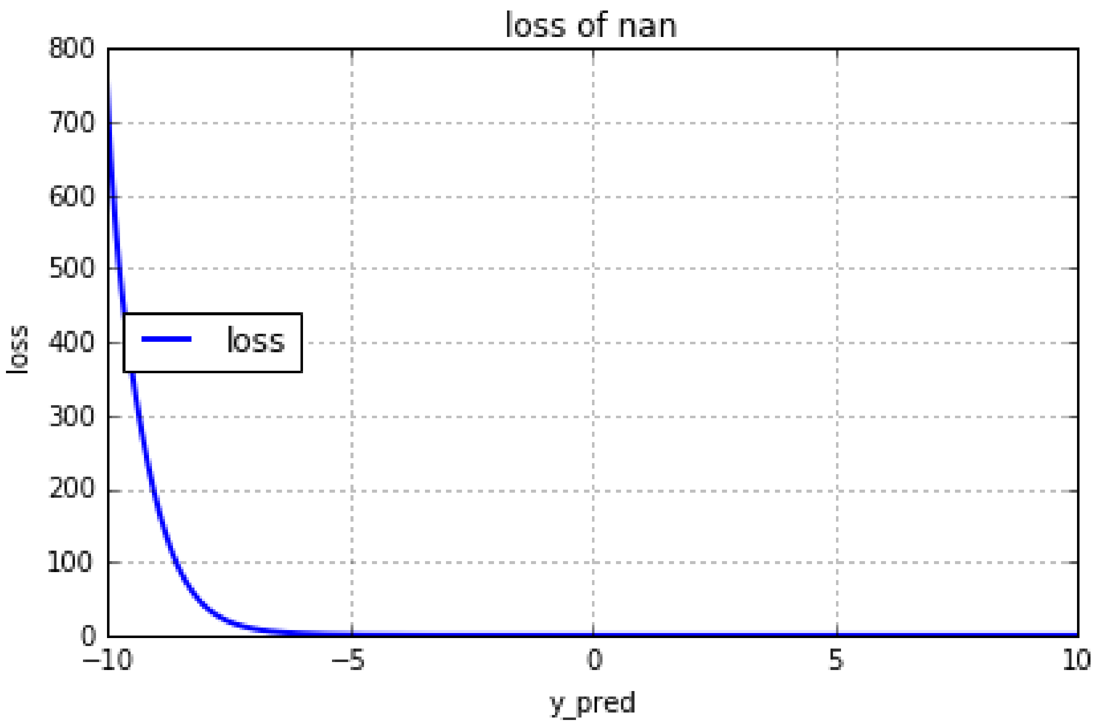
368 0.00014880861272104084
369 0.00014437608479056507
370 0.0001400706241838634
371 0.00013589489390142262
372 0.0001318538998020813
373 0.00012792422785423696
374 0.00012411409988999367
375 0.00012042245361953974
376 0.00011684449418680742
377 0.00011336674651829526
378 0.00010999780351994559
379 0.00010673738142941147
380 0.0001035651657730341
381 0.00010049381671706215
382 9.750755998538807e-05
383 9.46121581364423e-05
384 9.180873166769743e-05
385 8.908753079595044e-05
386 8.645085472380742e-05
387 8.389361755689606e-05
388 8.141212310874835e-05
389 7.90011472417973e-05
390 7.666576857445762e-05
391 7.439984619850293e-05
392 7.219740655273199e-05
393 7.006641681073233e-05
394 6.799779657740146e-05
395 6.598584150196984e-05
396 6.403742008842528e-05
397 6.215082976268604e-05
398 6.0319496697047725e-05
399 5.853941183886491e-05
400 5.681374022969976e-05
401 5.51393604837358e-05
402 5.35161052539479e-05
403 5.193746619625017e-05
404 5.040840187575668e-05
405 4.892657671007328e-05
406 4.748861465486698e-05
407 4.6087996452115476e-05
408 4.473387889447622e-05
409 4.3421863665571436e-05
410 4.214596629026346e-05
411 4.091026130481623e-05
412 3.970490433857776e-05
413 3.854452006635256e-05
414 3.741215914487839e-05
415 3.63133403880056e-05

416 3.524952262523584e-05
417 3.42150051437784e-05
418 3.321314579807222e-05
419 3.2238836865872145e-05
420 3.1296469387598336e-05
421 3.0380642783711664e-05
422 2.9490913220797665e-05
423 2.8627842766582035e-05
424 2.779205715341959e-05
425 2.6979801987181418e-05
426 2.619032602524385e-05
427 2.5423916667932644e-05
428 2.4682985895196907e-05
429 2.3962091290741228e-05
430 2.3261993192136288e-05
431 2.258350832562428e-05
432 2.1923840904491954e-05
433 2.128439700754825e-05
434 2.066389970423188e-05
435 2.0058692825841717e-05
436 1.9475692170090042e-05
437 1.8907956473412924e-05
438 1.8356042346567847e-05
439 1.7821563233155757e-05
440 1.7303620552411303e-05
441 1.6800342564238235e-05
442 1.6311725630657747e-05
443 1.583642551850062e-05
444 1.5376779629150406e-05
445 1.4927341908332892e-05
446 1.4494133210973814e-05
447 1.4072324120206758e-05
448 1.3663968275068328e-05
449 1.3267183931020554e-05
450 1.288118710363051e-05
451 1.2508214240369853e-05
452 1.2144530955993105e-05
453 1.1790480130002834e-05
454 1.1450625606812537e-05
455 1.1118075235572178e-05
456 1.0795685739140026e-05
457 1.0482966899871826e-05
458 1.0179225682804827e-05
459 9.885231520456728e-06
460 9.599159966455773e-06
461 9.321239303972106e-06
462 9.051499546330888e-06
463 8.787827937339898e-06

```
464 8.532768333679996e-06
465 8.287450327770784e-06
466 8.047923984122463e-06
467 7.814242962922435e-06
468 7.5888387982558925e-06
469 7.369790182565339e-06
470 7.156863375712419e-06
471 6.950516308279475e-06
472 6.747801307938062e-06
473 6.5534600253158715e-06
474 6.364960427163169e-06
475 6.180860509630293e-06
476 6.002323516440811e-06
477 5.829500423715217e-06
478 5.660027454723604e-06
479 5.498181053553708e-06
480 5.339671133697266e-06
481 5.185972440813202e-06
482 5.036808033764828e-06
483 4.891069238510681e-06
484 4.749660092784325e-06
485 4.612293651007349e-06
486 4.480062216316583e-06
487 4.350998096924741e-06
488 4.226360033499077e-06
489 4.104018898942741e-06
490 3.9862511584942695e-06
491 3.871176886605099e-06
492 3.7601739677484147e-06
493 3.6523726976156468e-06
494 3.546994321368402e-06
495 3.4443355616531335e-06
496 3.3463309137005126e-06
497 3.251097041356843e-06
498 3.156185584884952e-06
499 3.0662204153486528e-06
```

In [11]:
```python
# (1) 매 t마다 y_pred, loss  변화를 화면 출력확인(plot)
# x_array is a range of y_pred
x_array = np.linspace(-10, 10, len(loss4))

loss_array = np.asarray(loss4)
plt.plot(x_array, loss_array, "b-", linewidth=2, label="loss")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)
```

```python
plt.title("loss of nan")

plt.legend(loc="center left")
plt.tight_layout()

plt.show()
print(loss_array[:10])

# (2) 앞 문제의 코드와 비교
# 앞의 코드와 비교했을 때 loss는 계속
# 감소하는 방향으로 제대로 학습이 되고 있다.
```



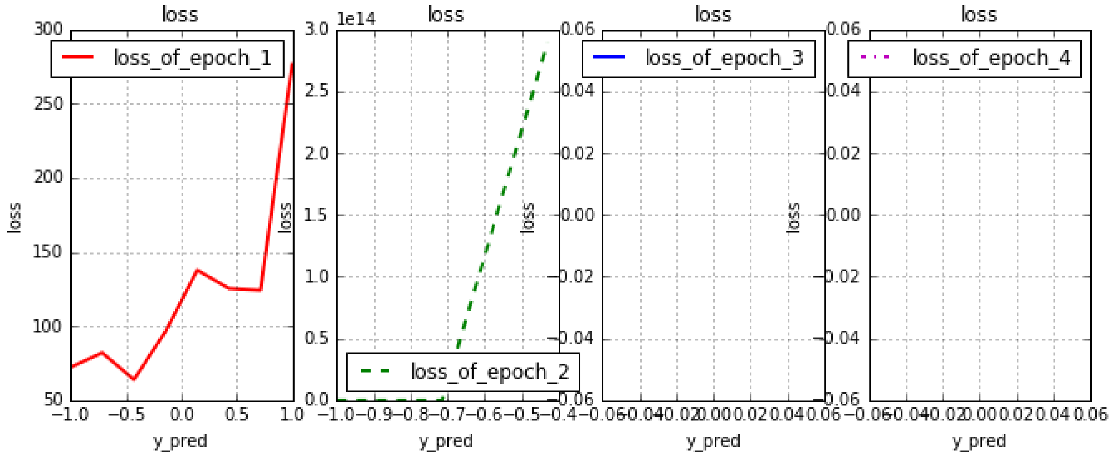

```
[tensor(752.2913, grad_fn=<MseLossBackward>)
 tensor(697.1153, grad_fn=<MseLossBackward>)
 tensor(649.8254, grad_fn=<MseLossBackward>)
 tensor(608.6284, grad_fn=<MseLossBackward>)
 tensor(572.4632, grad_fn=<MseLossBackward>)
 tensor(540.3026, grad_fn=<MseLossBackward>)
 tensor(511.0944, grad_fn=<MseLossBackward>)
 tensor(484.2495, grad_fn=<MseLossBackward>)
 tensor(459.5651, grad_fn=<MseLossBackward>)
 tensor(436.6903, grad_fn=<MseLossBackward>)]
```

8. [데이터 전처리] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요.(4점)
(코드의 해석과 결과의 의미를 작성하세요.)

In [12]:
```
import torch
from torch.utils.data import TensorDataset, DataLoader



# (1)의 그래프를 위한 library
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt


N, D_in, H, D_out = 64, 1000, 100, 10

x = torch.randn(N, D_in)
# In the case below, I fixed it from D to N, wrong letter D in original assignment.
y = torch.randn(N, D_out)

loader = DataLoader(TensorDataset(x,y), batch_size=8)
model = TwoLayerNet(D_in, H, D_out)

# mean squared error of nn module
loss_fn = torch.nn.MSELoss(reduction='sum')

# Stochastic gradient descent of nn module
optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)


loss5 = []
for epoch in range(20):
    loss5.append([])
    for x_batch, y_batch in loader:
        y_pred = model(x_batch)
        loss = loss_fn(y_pred, y_batch)
        loss5[epoch].append(loss)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()


# (1) 매 세대(epoch)마다 y_pred, loss 변화를 화면 출력 확인(plot)
# x_array is a range of y_pred
plt.figure(figsize=(11,4))

x_array = np.linspace(-1, 1, len(loss5[0]))
```

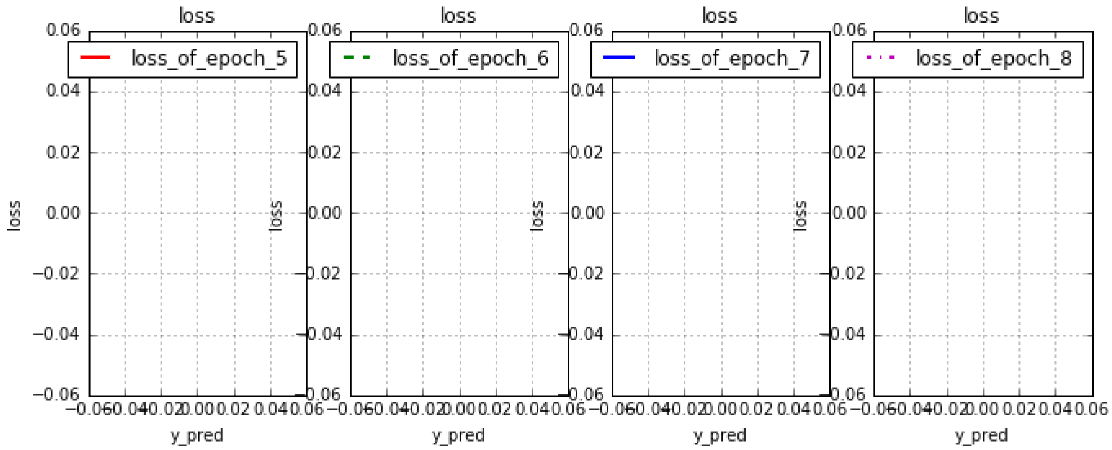
```
lines = ["r-", "g--", "b-", "m-."]

plt.subplot(141)
loss_array = np.asarray(loss5[0])
plt.plot(x_array, loss_array, lines[0], linewidth=2, label="loss_of_epoch_1")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")
plt.legend(loc="best")



plt.subplot(142)
loss_array = np.asarray(loss5[1])
plt.plot(x_array, loss_array, lines[1], linewidth=2, label="loss_of_epoch_2")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")



plt.subplot(143)
loss_array = np.asarray(loss5[2])
plt.plot(x_array, loss_array, lines[2], linewidth=2, label="loss_of_epoch_3")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")


plt.subplot(144)
loss_array = np.asarray(loss5[3])

plt.plot(x_array, loss_array, lines[3], linewidth=2, label="loss_of_epoch_4")
```

```
plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")

plt.show()
###### epoch 1 ~ 4 is done
```



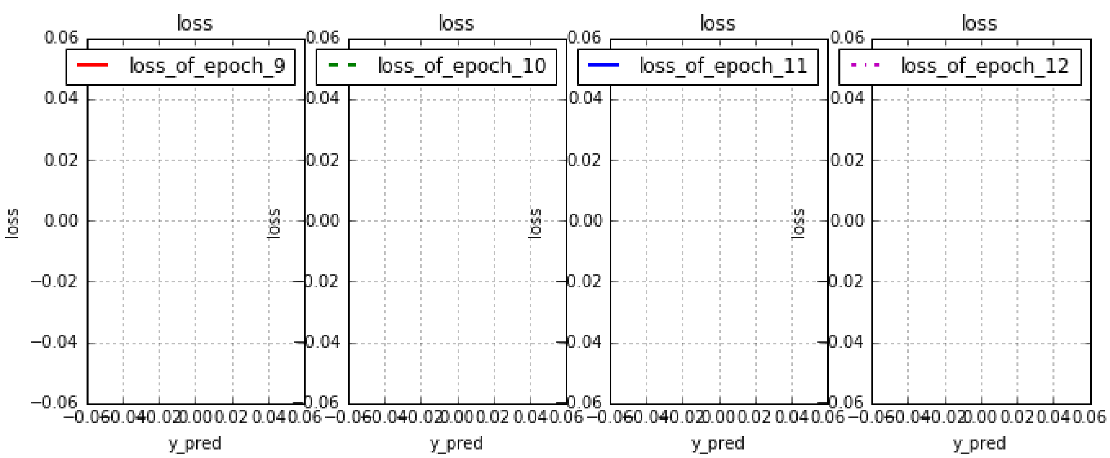

```
In [13]: plt.figure(figsize=(11,4))

         x_array = np.linspace(-1, 1, len(loss5[0]))

         lines = ["r-", "g--", "b-", "m-."]

         plt.subplot(141)
         loss_array = np.asarray(loss5[4])
         plt.plot(x_array, loss_array, lines[0], linewidth=2, label="loss_of_epoch_5")

         plt.xlabel("y_pred")
         plt.ylabel("loss")
         plt.grid(True)

         plt.title("loss")
         plt.legend(loc="best")
```

```
plt.subplot(142)
loss_array = np.asarray(loss5[5])
plt.plot(x_array, loss_array, lines[1], linewidth=2, label="loss_of_epoch_6")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")



plt.subplot(143)
loss_array = np.asarray(loss5[6])
plt.plot(x_array, loss_array, lines[2], linewidth=2, label="loss_of_epoch_7")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")


plt.subplot(144)
loss_array = np.asarray(loss5[7])

plt.plot(x_array, loss_array, lines[3], linewidth=2, label="loss_of_epoch_8")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")

plt.show()
###### epoch 5 ~ 8 is done
```



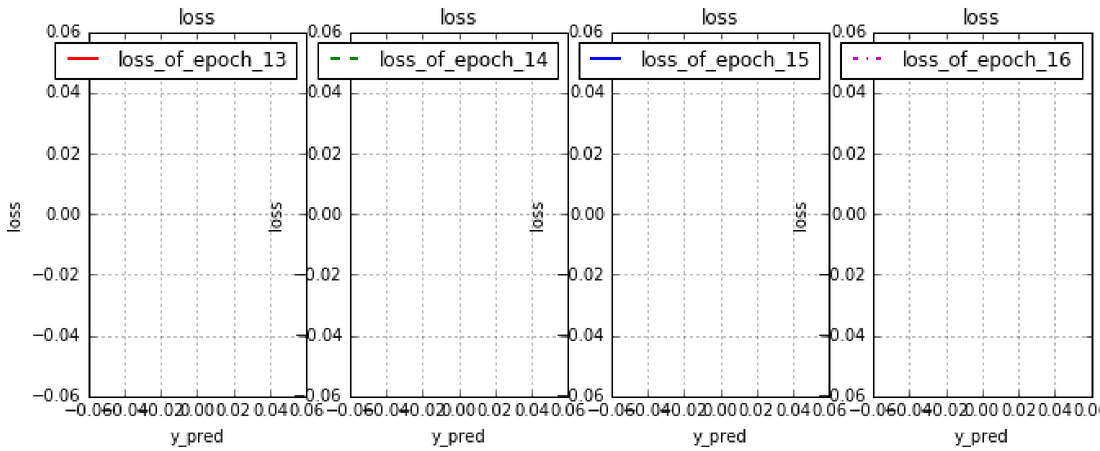

```
In [14]: plt.figure(figsize=(11,4))

         x_array = np.linspace(-1, 1, len(loss5[0]))

         lines = ["r-", "g--", "b-", "m-."]

         plt.subplot(141)
         loss_array = np.asarray(loss5[8])
         plt.plot(x_array, loss_array, lines[0], linewidth=2, label="loss_of_epoch_9")

         plt.xlabel("y_pred")
         plt.ylabel("loss")
         plt.grid(True)

         plt.title("loss")
         plt.legend(loc="best")



         plt.subplot(142)
         loss_array = np.asarray(loss5[9])
         plt.plot(x_array, loss_array, lines[1], linewidth=2, label="loss_of_epoch_10")

         plt.xlabel("y_pred")
         plt.ylabel("loss")
         plt.grid(True)

         plt.title("loss")

         plt.legend(loc="best")
```

```
plt.subplot(143)
loss_array = np.asarray(loss5[10])
plt.plot(x_array, loss_array, lines[2], linewidth=2, label="loss_of_epoch_11")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")


plt.subplot(144)
loss_array = np.asarray(loss5[11])

plt.plot(x_array, loss_array, lines[3], linewidth=2, label="loss_of_epoch_12")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")

plt.show()
###### epoch 9 ~ 12 is done
```



In [15]:
```
plt.figure(figsize=(11,4))

x_array = np.linspace(-1, 1, len(loss5[0]))

lines = ["r-", "g--", "b-", "m-."]

plt.subplot(141)
loss_array = np.asarray(loss5[12])
plt.plot(x_array, loss_array, lines[0], linewidth=2, label="loss_of_epoch_13")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")
plt.legend(loc="best")



plt.subplot(142)
loss_array = np.asarray(loss5[13])
plt.plot(x_array, loss_array, lines[1], linewidth=2, label="loss_of_epoch_14")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")



plt.subplot(143)
loss_array = np.asarray(loss5[14])
plt.plot(x_array, loss_array, lines[2], linewidth=2, label="loss_of_epoch_15")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")


plt.subplot(144)
```

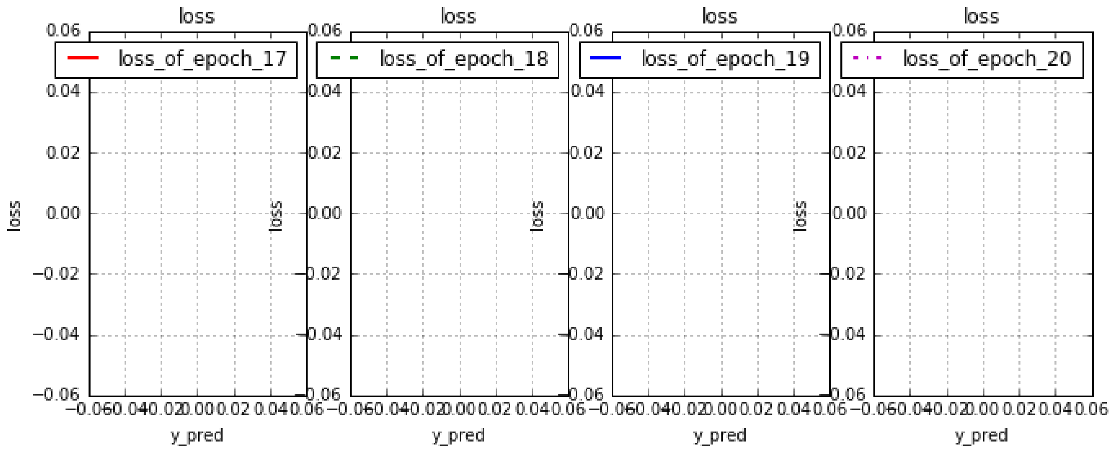
```
loss_array = np.asarray(loss5[15])

plt.plot(x_array, loss_array, lines[3], linewidth=2, label="loss_of_epoch_16")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")

plt.show()
###### epoch 13 ~ 16 is done
```




```
In [16]: plt.figure(figsize=(11,4))

         x_array = np.linspace(-1, 1, len(loss5[0]))

         lines = ["r-", "g--", "b-", "m-."]

         plt.subplot(141)
         loss_array = np.asarray(loss5[16])
         plt.plot(x_array, loss_array, lines[0], linewidth=2, label="loss_of_epoch_17")

         plt.xlabel("y_pred")
         plt.ylabel("loss")
         plt.grid(True)

         plt.title("loss")
```

```
plt.legend(loc="best")



plt.subplot(142)
loss_array = np.asarray(loss5[17])
plt.plot(x_array, loss_array, lines[1], linewidth=2, label="loss_of_epoch_18")

plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")



plt.subplot(143)
loss_array = np.asarray(loss5[18])
plt.plot(x_array, loss_array, lines[2], linewidth=2, label="loss_of_epoch_19")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")


plt.subplot(144)
loss_array = np.asarray(loss5[19])

plt.plot(x_array, loss_array, lines[3], linewidth=2, label="loss_of_epoch_20")


plt.xlabel("y_pred")
plt.ylabel("loss")
plt.grid(True)

plt.title("loss")

plt.legend(loc="best")

plt.show()
###### epoch 17 ~ 20 is done
```

```
print(loss_array[:10])

# (2) 앞 문제의 코드와 비교
# 앞의 문제의 코드와 비교 했을때,
# epoch이 증가 할때마다 loss가 증가하고
# inf(무한대) 값을 거처 NAN 값으로 변하기 때문에
# 학습이 제대로 이루어지지 않고 있다.
```

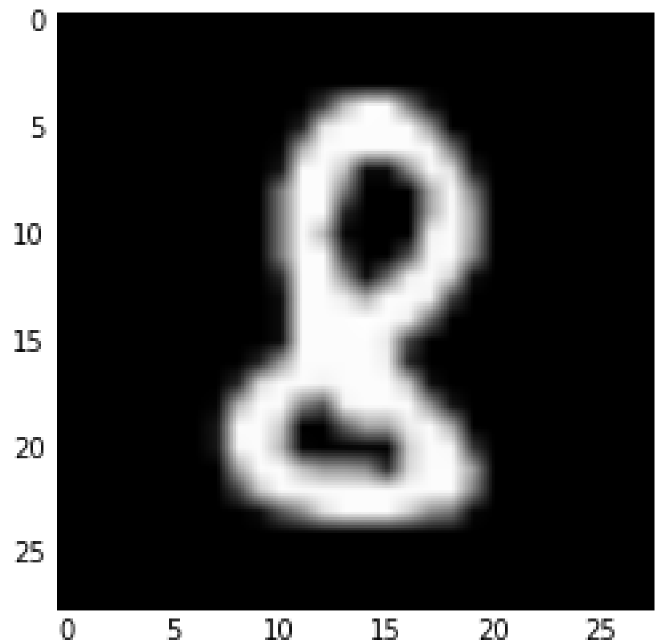



```
[tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)
 tensor(nan, grad_fn=<MseLossBackward>)]
```

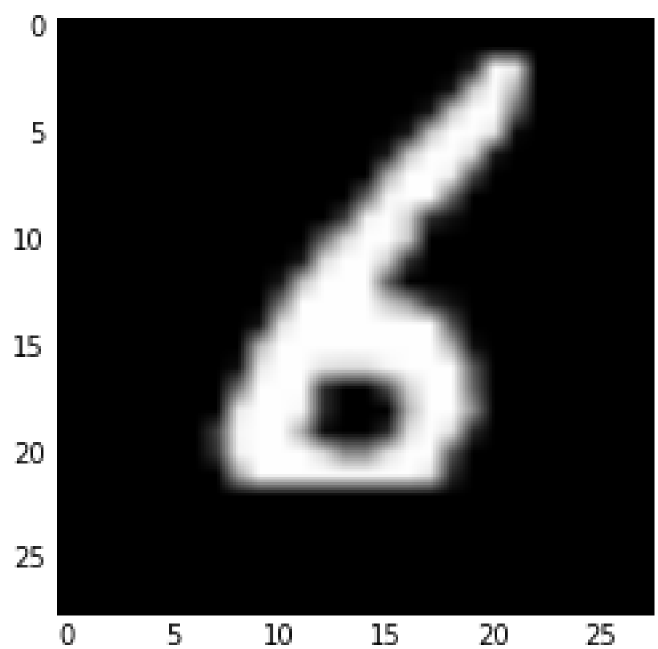
9. [영상 인식] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요.(12 점) (코드의 해석과 결과의 의미를 작성하세요.)

In [17]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.autograd import Variable
import matplotlib.pyplot as plt
%matplotlib inline

is_cuda = False
```

```python
#if torch.cuda.is_available():
#    is_cuda = True


# Load data
# reference is
# https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
transformation = transforms.Compose([transforms.ToTensor(),
                                     transforms.Normalize((0.1307,),(0.3081,))])

# MNIST data set
train_dataset = datasets.MNIST('data/', train=True,
                               transform=transformation, download=True)
test_dataset = datasets.MNIST('data/', train=False,
                              transform=transformation, download=True)


train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=True)


# first data for sample
sample_data = next(iter(train_loader))


def plot_img(image):
    image = image.numpy()[0]
    mean = 0.1307
    std = 0.3081
    image = ((mean * image ) + std)
    plt.imshow(image, cmap='gray')


# first data set's image
plot_img(sample_data[0][2])
# (1) 화면 출력 확인
```




```
In [18]: plot_img(sample_data[0][1])
         # (2) 화면 출력 확인
```



```
In [19]: # Convoluttion neural network

         class Net(nn.Module):
             def __init__(self):
                 super().__init__()
                 self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
                 self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
                 # Dropout for optimization to regularize
                 self.conv2_drop = nn.Dropout2d()
                 self.fc1 = nn.Linear(320, 50)
                 self.fc2 = nn.Linear(50, 10)

             def forward(self, x):
                 x = F.relu(F.max_pool2d(self.conv1(x), 2))
                 x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
                 x = x.view(-1, 320)
                 x = F.relu(self.fc1(x))
                 #x = F.dropout(x, p=0.1, training = self.training)
                 x = self.fc2(x)
                 return F.log_softmax(x, dim=1)

         model = Net()

         is_cuda = False

         if is_cuda:
             model.cuda()

         optimizer = optim.SGD(model.parameters(), lr=0.01)

         data, target = next(iter(train_loader))

         output = model(Variable(data))

         # (3) output.size() 출력확인
         # predicted output tensor of log_softmax
         print("output.size():\n{}".format(output.size()))
         # (4) target.size() 출력확인
         # actual label
         print("\ntarget.size():\n{}".format(target.size()))
```

```
output.size():
torch.Size([32, 10])

target.size():
torch.Size([32])
```

```
In [20]: # For training of model of Net()
```

```python
def fit(epoch, model, data_loader, phase="training", volatile=False):
    if phase == 'training':
        model.train()
    if phase == 'validation':
        model.eval()
        volatile = True
    running_loss = 0.0
    running_correct = 0
    for batch_idx, (data, target) in enumerate(data_loader):
        #if is_cuda:
        #    data, target = data.cuda(), target.cuda()
        data, target = Variable(data, volatile), Variable(target)
        if phase == 'training':
            optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)

        running_loss += F.nll_loss(output, target, size_average=False).data[0]
        preds = output.data.max(dim=1, keepdim=True)[1]
        running_correct += preds.eq(target.data.view_as(preds)).cpu().sum()

        if phase == 'training':
            loss.backward()
            optimizer.step()

    loss = running_loss/len(data_loader.dataset)
    accuracy = 100. * running_correct/len(data_loader.dataset)

    print("{} loss is {} and {} accuracy is {}/{} -> {})".format(phase, loss, phase,
                                                            running_correct,
                                                        len(data_loader.dataset),
                                                            accuracy))
    return loss, accuracy

train_losses, train_accuracy = [], []
val_losses, val_accuracy = [], []

for epoch in range(1, 20):
    epoch_loss, epoch_accuracy = fit(epoch, model, train_loader, phase='training')
    val_epoch_loss, val_epoch_accuracy = fit(epoch, model,
                                        test_loader, phase='validation')
    train_losses.append(epoch_loss)
    train_accuracy.append(epoch_accuracy)
    val_losses.append(val_epoch_loss)
    val_accuracy.append(val_epoch_accuracy)
    # (5) 화면 출력 확인
    # Cross validation
```

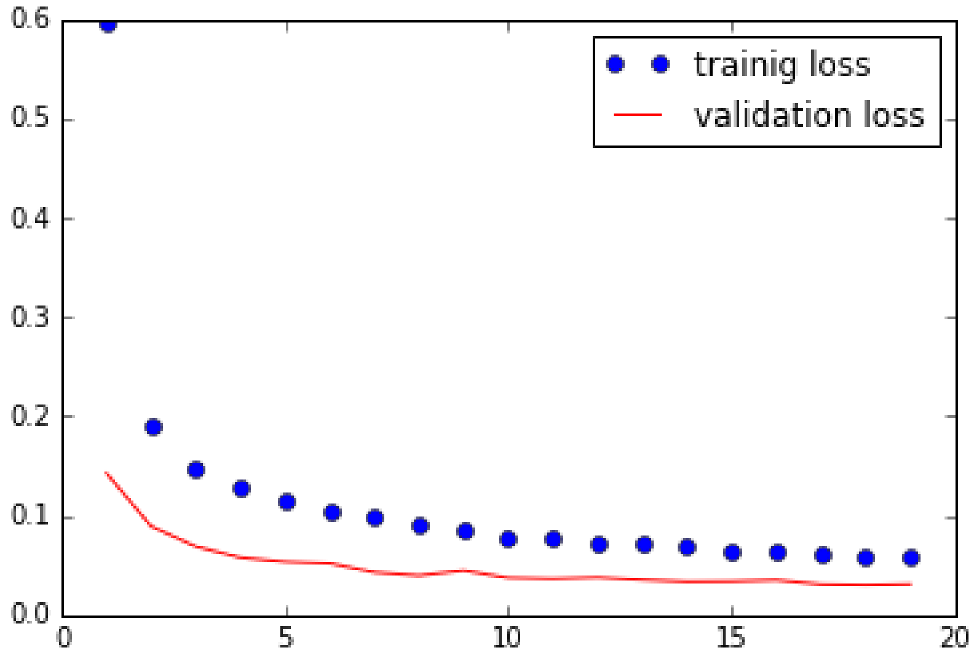
```
        # comparing training set with validation set
        # I can early stop
```

```
/home/hyunyoung2/.local/lib/python3.5/site-packages/torch/nn/functional.py:52: UserWarning: si
  warnings.warn(warning.format(ret))
/home/hyunyoung2/.local/lib/python3.5/site-packages/ipykernel_launcher.py:20: UserWarning: inva


training loss is 0.5956605672836304 and training accuracy is 49062/60000 -> 81)
validation loss is 0.1428748220205307 and validation accuracy is 9557/10000 -> 95)
training loss is 0.19141218066215515 and training accuracy is 56613/60000 -> 94)
validation loss is 0.08931832015514374 and validation accuracy is 9723/10000 -> 97)
training loss is 0.14799994230270386 and training accuracy is 57386/60000 -> 95)
validation loss is 0.06888460367918015 and validation accuracy is 9778/10000 -> 97)
training loss is 0.12740691006183624 and training accuracy is 57771/60000 -> 96)
validation loss is 0.057700544595718384 and validation accuracy is 9805/10000 -> 98)
training loss is 0.1137586161494255 and training accuracy is 57977/60000 -> 96)
validation loss is 0.0534110888838768 and validation accuracy is 9824/10000 -> 98)
training loss is 0.10471786558628082 and training accuracy is 58127/60000 -> 96)
validation loss is 0.05187523365020752 and validation accuracy is 9823/10000 -> 98)
training loss is 0.09773274511098862 and training accuracy is 58229/60000 -> 97)
validation loss is 0.042845770716667175 and validation accuracy is 9857/10000 -> 98)
training loss is 0.09033861011266708 and training accuracy is 58359/60000 -> 97)
validation loss is 0.0396590456366539 and validation accuracy is 9870/10000 -> 98)
training loss is 0.08578614890575409 and training accuracy is 58478/60000 -> 97)
validation loss is 0.04460400342941284 and validation accuracy is 9852/10000 -> 98)
training loss is 0.07841832935810089 and training accuracy is 58611/60000 -> 97)
validation loss is 0.037293486297130585 and validation accuracy is 9873/10000 -> 98)
training loss is 0.07716350257396698 and training accuracy is 58599/60000 -> 97)
validation loss is 0.03643083572387695 and validation accuracy is 9882/10000 -> 98)
training loss is 0.07127117365598679 and training accuracy is 58697/60000 -> 97)
validation loss is 0.03758731856942177 and validation accuracy is 9877/10000 -> 98)
training loss is 0.07082290202379227 and training accuracy is 58744/60000 -> 97)
validation loss is 0.035266581922769547 and validation accuracy is 9887/10000 -> 98)
training loss is 0.07003258168697357 and training accuracy is 58731/60000 -> 97)
validation loss is 0.0334736593067646 and validation accuracy is 9895/10000 -> 98)
training loss is 0.06351704150438309 and training accuracy is 58842/60000 -> 98)
validation loss is 0.03375457599759102 and validation accuracy is 9894/10000 -> 98)
training loss is 0.06294967234134674 and training accuracy is 58852/60000 -> 98)
validation loss is 0.03494734689593315 and validation accuracy is 9877/10000 -> 98)
training loss is 0.06090139225125313 and training accuracy is 58898/60000 -> 98)
validation loss is 0.030788660049438477 and validation accuracy is 9902/10000 -> 99)
training loss is 0.058314986526966095 and training accuracy is 58939/60000 -> 98)
validation loss is 0.03013336844742298 and validation accuracy is 9897/10000 -> 98)
training loss is 0.057965610176324844 and training accuracy is 58945/60000 -> 98)
validation loss is 0.030912548303604126 and validation accuracy is 9896/10000 -> 98)
```

```
In [21]: plt.plot(range(1,len(train_losses)+1), train_losses, 'bo', label = 'trainig loss')
```

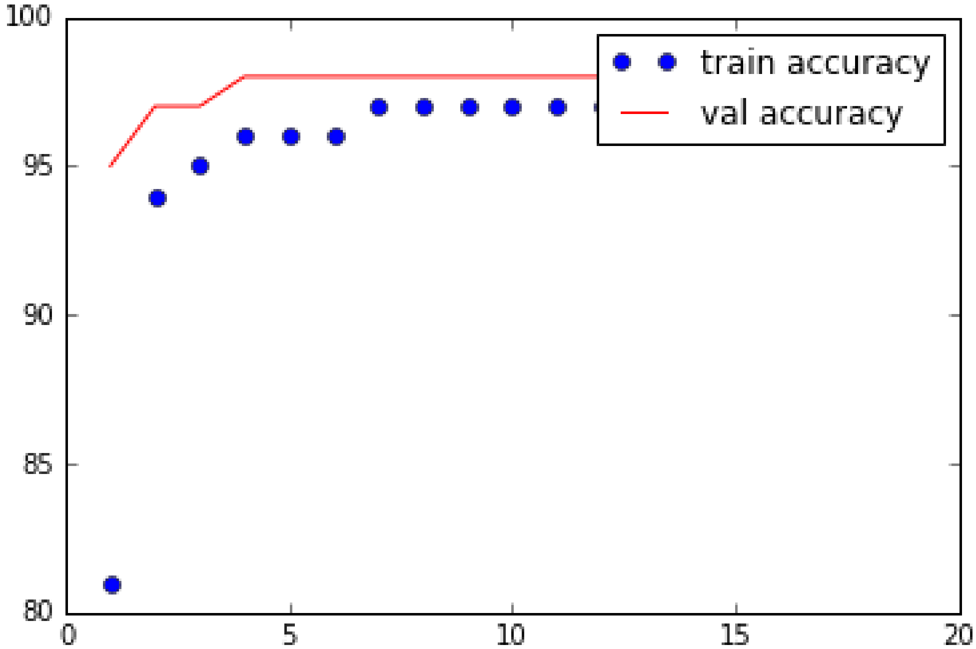
```
plt.plot(range(1, len(val_losses)+1), val_losses, 'r', label = 'validation loss')
plt.legend()
# (6) 화면 출력 확인
# comparing training loss and validation loss
```

Out[21]: <matplotlib.legend.Legend at 0x7f15a4263be0>



In [22]:
```
plt.plot(range(1, len(train_accuracy)+1), train_accuracy, 'bo', label = 'train accurac
plt.plot(range(1, len(val_accuracy)+1), val_accuracy, 'r', label = 'val accuracy')
plt.legend()
# (7) 화면 출력 확인
# comparing triang accuracy and validation loss
```

Out[22]: <matplotlib.legend.Legend at 0x7f15a421a160>



10. NOR 게이트와 AND 게이트의 동작을 데이터로 간주하면 다음과 같다. 이들을 100% 옳게 분류하는 퍼셉트론을 각각 제시하시오.

$$\text{NOR 분류}\begin{cases}\mathbf{x}_1=(0,0)^T, y_1 = 1\\ \mathbf{x}_2=(1,0)^T, y_2=-1\\ \mathbf{x}_3=(0,1)^T, y_3=-1\\ \mathbf{x}_4=(1,1)^T, y_4=-1\end{cases}\qquad \text{AND 분류}\begin{cases}\mathbf{x}_1=(0,0)^T, y_1=-1\\ \mathbf{x}_2=(1,0)^T, y_2=-1\\ \mathbf{x}_3=(0,1)^T, y_3=-1\\ \mathbf{x}_4=(1,1)^T, y_4=1\end{cases}$$

In [ ]:

figure1-1의 퍼셉트론은 NOR gate를 위한 것이다. NOR 게이트의 분류를 하는 것도 hpyer plane 하나로 가능하기 때문에 figure1의 그림처럼 hyper plane으로 영역을 두개로 구분하여 분류가 가능하다.

In [ ]:

figure2-1 퍼셉트론은 AND gate를 위한 것이다. AND 게이트의 경우에는 하나의 hyper plane을 통해 output 결과들을 figure2와 같이 분류할 수 있다. 그래서 충분히 하나의 perceptron으로 분류가 가능하다.

In [ ]:

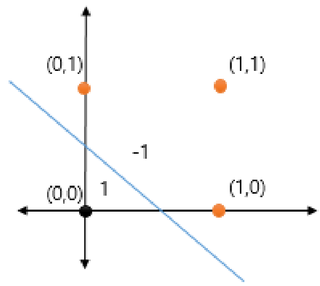


Figure1. NOR gate

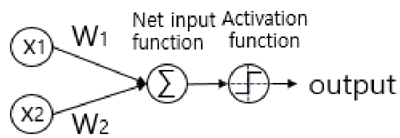


Figure1-1. NOR gate perceptron

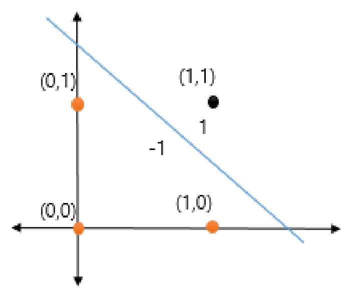


Figure2. AND gate

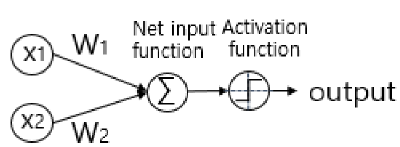
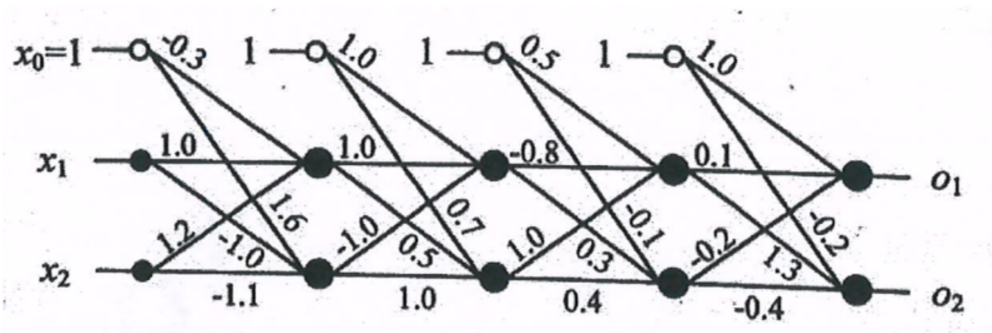


Figure2-1. AND gate perceptron

11. 다음은 은닉층이 3개인 DMLP이다. Hint 계산은 Matlab 또는 Python을 사용하시오.

In [ ]:



In [ ]:

(1) 가중치 행렬 U1, U2, U3, U4를 식(4.1)처럼 쓰시오.

In [23]:

```python
import numpy as np

U1 = np.array([[-0.3, 1.0, 1.2],
               [1.6, -1.0, -1.1]])

U2 = np.array([[1.0, 1.0, -1.0],
               [0.7, 0.5, 1.0]])

U3 = np.array([[0.5, -0.8, 1.0],
               [-0.1, 0.3, 0.4]])

U4 = np.array([[1.0, 0.1, -0.2],
               [-0.2, 1.3, -0.4]])

print("U1:\n{}".format(U1))
print("U2:\n{}".format(U2))
print("U3:\n{}".format(U3))
print("U4:\n{}".format(U4))
```

```
U1:
[[-0.3  1.   1.2]
 [ 1.6 -1.  -1.1]]
U2:
[[ 1.   1.  -1. ]
 [ 0.7  0.5  1. ]]
U3:
[[ 0.5 -0.8  1. ]
```

```
 [-0.1  0.3  0.4]]
U4:
[[ 1.   0.1 -0.2]
 [-0.2  1.3 -0.4]]
```

(2) x = (1,0)T 가 입력되었을 때 출력 O를 구하시오. 활성함수로 로지스틱 시그모이드를 사용하시오.

```
In [24]: def sigmoid(z):
             return 1/(1+np.exp(-z))

         x = np.array([1,1,0])
         print("x:{}\n{}".format(x.shape, x))
         x_t = x.T
         print("x.T:{}\n{}".format(x_t.shape, x_t))
         hidden_1 = sigmoid(U1.dot(x_t))
         print("hidden_1:{}\n{}".format(hidden_1.shape, hidden_1))

         # U1 is done

         x_1 = np.append([1], hidden_1)
         print("\nx_1:{}\n{}".format(x_1.shape, x_1))
         x_1_t = x_1.T
         print("x_1.T:{}\n{}".format(x_1_t.shape, x_1_t))
         hidden_2 = sigmoid(U2.dot(x_1_t))
         print("hidden_2:{}\n{}".format(hidden_2.shape, hidden_2))

         # U2 is done

         x_2 = np.append([1], hidden_2)
         print("\nx_2:{}\n{}".format(x_2.shape, x_2))
         x_2_t = x_2.T
         print("x_2.T:{}\n{}".format(x_2_t.shape, x_2_t))
         hidden_3 = sigmoid(U3.dot(x_2_t))
         print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

         # U3 is done

         x_3 = np.append([1], hidden_3)
         print("\nx_3:{}\n{}".format(x_3.shape, x_3))
         x_3_t = x_3.T
         print("x_3.T:{}\n{}".format(x_3_t.shape, x_3_t))
         hidden_3 = U4.dot(x_3_t)
         print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

         # U4 is done
```

```
        print("\ninput of sigmoid:\n{}".format(hidden_3))
        print("final output with logisitic sigmoid funtion")
        output_of_sigmoid = sigmoid(hidden_3)
        print(output_of_sigmoid)
```

```
x:(3,)
[1 1 0]
x.T:(3,)
[1 1 0]
hidden_1:(2,)
[0.66818777 0.64565631]

x_1:(3,)
[1.         0.66818777 0.64565631]
x_1.T:(3,)
[1.         0.66818777 0.64565631]
hidden_2:(2,)
[0.7354654  0.84287145]

x_2:(3,)
[1.         0.7354654  0.84287145]
x_2.T:(3,)
[1.         0.7354654  0.84287145]
hidden_3:(2,)
[0.68015824 0.61248935]

x_3:(3,)
[1.         0.68015824 0.61248935]
x_3.T:(3,)
[1.         0.68015824 0.61248935]
hidden_3:(2,)
[0.94551796 0.43920998]

input of sigmoid:
[0.94551796 0.43920998]
final output with logisitic sigmoid funtion
[0.72021291 0.60807077]
```

(3) x = (1,0)T 가 입력되었을 때 출력 O를 구하시오. 활성함수로 ReLU를 사용하시오.

sol> 위의 simoid function에서 활성함수를 ReLU로 바꾸면 바로 그 값이 되기때문에 활성함수만 만들고 바로 hidden_3의 값에 활성함수 ReLU를 사용하였다.

```
In [25]: def ReLU(z):
             return np.maximum(0, z)

         x = np.array([1,1,0])
         print("x:{}\n{}".format(x.shape, x))
```

```python
        x_t = x.T
        print("x.T:{}\n{}".format(x_t.shape, x_t))
        hidden_1 = ReLU(U1.dot(x_t))
        print("hidden_1:{}\n{}".format(hidden_1.shape, hidden_1))

        # U1 is done

        x_1 = np.append([1], hidden_1)
        print("\nx_1:{}\n{}".format(x_1.shape, x_1))
        x_1_t = x_1.T
        print("x_1.T:{}\n{}".format(x_1_t.shape, x_1_t))
        hidden_2 = ReLU(U2.dot(x_1_t))
        print("hidden_2:{}\n{}".format(hidden_2.shape, hidden_2))

        # U2 is done

        x_2 = np.append([1], hidden_2)
        print("\nx_2:{}\n{}".format(x_2.shape, x_2))
        x_2_t = x_2.T
        print("x_2.T:{}\n{}".format(x_2_t.shape, x_2_t))
        hidden_3 = ReLU(U3.dot(x_2_t))
        print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

        # U3 is done

        x_3 = np.append([1], hidden_3)
        print("\nx_3:{}\n{}".format(x_3.shape, x_3))
        x_3_t = x_3.T
        print("x_3.T:{}\n{}".format(x_3_t.shape, x_3_t))
        hidden_3 = U4.dot(x_3_t)
        print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

        # U4 is done

        print("input of ReLU:\n{}".format(hidden_3))
        print("final output with ReLU on hidden_3")
        output_of_ReLU = ReLU(hidden_3)
        print(output_of_ReLU)
```

```
x:(3,)
[1 1 0]
x.T:(3,)
[1 1 0]
hidden_1:(2,)
[0.7 0.6]

x_1:(3,)
[1.  0.7 0.6]
```

```
x_1.T:(3,)
[1.  0.7 0.6]
hidden_2:(2,)
[1.1  1.65]

x_2:(3,)
[1.   1.1  1.65]
x_2.T:(3,)
[1.   1.1  1.65]
hidden_3:(2,)
[1.27 0.89]

x_3:(3,)
[1.   1.27 0.89]
x_3.T:(3,)
[1.   1.27 0.89]
hidden_3:(2,)
[0.949 1.095]
input of ReLU:
[0.949 1.095]
final output with ReLU on hidden_3
[0.949 1.095]
```

(4) x = (1,0)T 의 기대출력 O = (0,1)T일 때, 현재 1.0인 u312 가중치를 0.9로 줄이면 오류에 어떤 영향을 미치는지 설명하시오.

sol> 오류 함수를 Mean Squared Error 함수로 설정하고, 활성함수는 logistic sigmoid 와 ReLU 두개로 하여 결과를 아래와 같이 뽑아 냈다. 그 결과 활성함수 logisitic sigmoid function과 함께 한 경우, Mean Squared Error는 증가하고, 반대로 활성함수 ReLU와 함께 한 경우는 Mean Sqared Error 가 감소하였다.

```
In [26]: expected = np.array([0,1], dtype=float)

         print("expected:\n{}".format(expected))

         def loss_function(output, expectation):
             return ((output-expectation)**2).mean()

         print("\nU321 0.9로 바꾸기 전 with logisitic sigmoid")
         prior_sigmoid = loss_function(output_of_sigmoid, expected)
         print(prior_sigmoid)

         U_3 = np.array([[0.5, -0.8, 0.9],
                         [-0.1, 0.3, 0.4]])

         print("\n U321 1.0 -> 0.9 바꾼후 다시 계산")
         x = np.array([1,1,0])
```

```
print("x:{}\n{}".format(x.shape, x))
x_t = x.T
print("x.T:{}\n{}".format(x_t.shape, x_t))
hidden_1 = sigmoid(U1.dot(x_t))
print("hidden_1:{}\n{}".format(hidden_1.shape, hidden_1))

# U1 is done

x_1 = np.append([1], hidden_1)
print("\nx_1:{}\n{}".format(x_1.shape, x_1))
x_1_t = x_1.T
print("x_1.T:{}\n{}".format(x_1_t.shape, x_1_t))
hidden_2 = sigmoid(U2.dot(x_1_t))
print("hidden_2:{}\n{}".format(hidden_2.shape, hidden_2))

# U2 is done

x_2 = np.append([1], hidden_2)
print("\nx_2:{}\n{}".format(x_2.shape, x_2))
x_2_t = x_2.T
print("x_2.T:{}\n{}".format(x_2_t.shape, x_2_t))
hidden_3 = sigmoid(U_3.dot(x_2_t))
print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

# U3 is done

x_3 = np.append([1], hidden_3)
print("\nx_3:{}\n{}".format(x_3.shape, x_3))
x_3_t = x_3.T
print("x_3.T:{}\n{}".format(x_3_t.shape, x_3_t))
hidden_3 = U4.dot(x_3_t)
print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

# U4 is done

print("\ninput of sigmoid:\n{}".format(hidden_3))
print("final output with logisitic sigmoid funtion")
output_of_sigmoid_after = sigmoid(hidden_3)
print(output_of_sigmoid_after)

print("\nU321 0.9로 바꾸기 후 with logisitic sigmoid funtion")
post_sigmoid = loss_function(output_of_sigmoid_after, expected)
print("Befor: {}".format(prior_sigmoid))
print("After: {}".format(post_sigmoid))
```

```
expected:
[0. 1.]
```

```
U321 0.9로 바꾸기 전 with logisitic sigmoid
0.33615757900101034

 U321 1.0 -> 0.9 바꾼후 다시 계산
x:(3,)
[1 1 0]
x.T:(3,)
[1 1 0]
hidden_1:(2,)
[0.66818777 0.64565631]

x_1:(3,)
[1.         0.66818777 0.64565631]
x_1.T:(3,)
[1.         0.66818777 0.64565631]
hidden_2:(2,)
[0.7354654  0.84287145]

x_2:(3,)
[1.         0.7354654  0.84287145]
x_2.T:(3,)
[1.         0.7354654  0.84287145]
hidden_3:(2,)
[0.66155062 0.61248935]

x_3:(3,)
[1.         0.66155062 0.61248935]
x_3.T:(3,)
[1.         0.66155062 0.61248935]
hidden_3:(2,)
[0.94365719 0.41502007]

input of sigmoid:
[0.94365719 0.41502007]
final output with logisitic sigmoid funtion
[0.7198378  0.60229099]

U321 0.9로 바꾸기 후 with logisitic sigmoid funtion
Befor: 0.33615757900101034
After: 0.3381694602679114
```

In [27]:
```python
expected = np.array([0,1], dtype=float)

print("expected:\n{}".format(expected))

def loss_function(output, expectation):
    return ((output-expectation)**2).mean()
```

```python
print("\nU321 0.9로 바꾸기 전 with ReLU")
prior_ReLU = loss_function(output_of_ReLU, expected)
print(prior_ReLU)

U_3 = np.array([[0.5, -0.8, 0.9],
               [-0.1, 0.3, 0.4]])

print("\n U321 1.0 -> 0.9 바꾼후 다시 계산")
x = np.array([1,1,0])
print("x:{}\n{}".format(x.shape, x))
x_t = x.T
print("x.T:{}\n{}".format(x_t.shape, x_t))
hidden_1 = ReLU(U1.dot(x_t))
print("hidden_1:{}\n{}".format(hidden_1.shape, hidden_1))

# U1 is done

x_1 = np.append([1], hidden_1)
print("\nx_1:{}\n{}".format(x_1.shape, x_1))
x_1_t = x_1.T
print("x_1.T:{}\n{}".format(x_1_t.shape, x_1_t))
hidden_2 = ReLU(U2.dot(x_1_t))
print("hidden_2:{}\n{}".format(hidden_2.shape, hidden_2))

# U2 is done

x_2 = np.append([1], hidden_2)
print("\nx_2:{}\n{}".format(x_2.shape, x_2))
x_2_t = x_2.T
print("x_2.T:{}\n{}".format(x_2_t.shape, x_2_t))
hidden_3 = ReLU(U_3.dot(x_2_t))
print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

# U3 is done

x_3 = np.append([1], hidden_3)
print("\nx_3:{}\n{}".format(x_3.shape, x_3))
x_3_t = x_3.T
print("x_3.T:{}\n{}".format(x_3_t.shape, x_3_t))
hidden_3 = U4.dot(x_3_t)
print("hidden_3:{}\n{}".format(hidden_3.shape, hidden_3))

# U4 is done

print("\ninput of ReLU:\n{}".format(hidden_3))
print("final output with ReLU")
output_of_ReLU_after = ReLU(hidden_3)
```

```
        print(output_of_ReLU_after)

        print("\nU321 0.9로 바꾸기 후 with ReLU")
        post_ReLU = loss_function(output_of_ReLU_after, expected)
        print("Befor: {}".format(prior_ReLU))
        print("After: {}".format(post_ReLU))
```

```
expected:
[0. 1.]

U321 0.9로 바꾸기 전 with ReLU
0.4548129999999999

 U321 1.0 -> 0.9 바꾼후 다시 계산
x:(3,)
[1 1 0]
x.T:(3,)
[1 1 0]
hidden_1:(2,)
[0.7 0.6]

x_1:(3,)
[1.  0.7 0.6]
x_1.T:(3,)
[1.  0.7 0.6]
hidden_2:(2,)
[1.1  1.65]

x_2:(3,)
[1.   1.1  1.65]
x_2.T:(3,)
[1.   1.1  1.65]
hidden_3:(2,)
[1.105 0.89 ]

x_3:(3,)
[1.    1.105 0.89 ]
x_3.T:(3,)
[1.    1.105 0.89 ]
hidden_3:(2,)
[0.9325 0.8805]

input of ReLU:
[0.9325 0.8805]
final output with ReLU
[0.9325 0.8805]

U321 0.9로 바꾸기 후 with ReLU
```

```
Befor: 0.45481299999999999
After: 0.44191825
```

12. [그림 4-14]에서 특징 맵의 나머지 8개 값을 계산하시오.

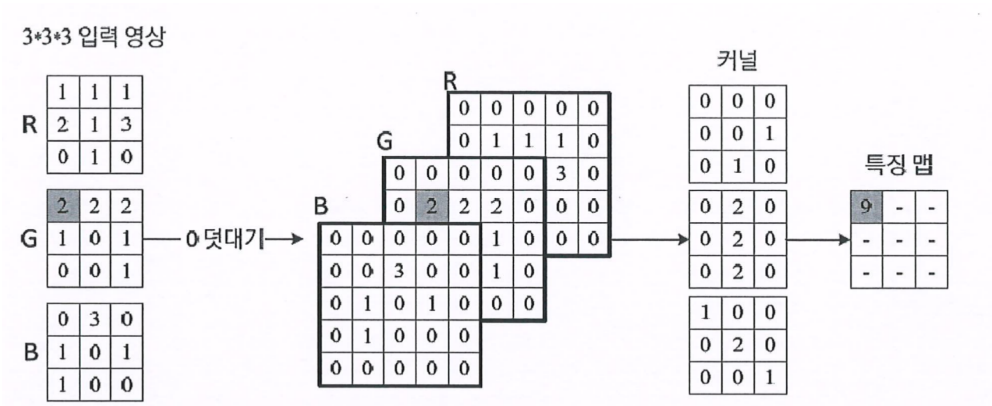


그림 4-14 텐서의 컨볼루션 연산(0 덧대기 적용)

```
In [28]: import numpy as np

         R = np.array([[0,0,0,0,0],
                       [0,1,1,1,0],
                       [0,2,1,3,0],
                       [0,0,1,0,0],
                       [0,0,0,0,0]])

         G = np.array([[0,0,0,0,0],
                       [0,2,2,2,0],
                       [0,1,0,1,0],
                       [0,0,0,1,0],
                       [0,0,0,0,0]])

         B = np.array([[0,0,0,0,0],
                       [0,0,3,0,0],
                       [0,1,0,1,0],
                       [0,1,0,0,0],
                       [0,0,0,0,0]])

         R_kernel =  np.array([[0,0,0],
                               [0,0,1],
                               [0,1,0]])
```

```
G_kernel =  np.array([[0,2,0],
                      [0,2,0],
                      [0,2,0]])

B_kernel =  np.array([[1,0,0],
                      [0,2,0],
                      [0,0,1]])


a11 = ((R[0:3, 0:3] * R_kernel) +
       (G[0:3, 0:3] * G_kernel) +
       (B[0:3, 0:3] * B_kernel))

a12 = ((R[0:3, 0+1:3+1]* R_kernel) +
       (G[0:3, 0+1:3+1] * G_kernel) +
       (B[0:3, 0+1:3+1] * B_kernel))

a13 = ((R[0:3, 0+2:3+2] * R_kernel) +
       (G[0:3, 0+2:3+2] * G_kernel) +
       (B[0:3, 0+2:3+2] * B_kernel))

a21 = ((R[0+1:3+1, 0:3] * R_kernel) +
       (G[0+1:3+1, 0:3] * G_kernel) +
       (B[0+1:3+1, 0:3] * B_kernel))

a22 = ((R[0+1:3+1, 0+1:3+1]* R_kernel) +
       (G[0+1:3+1, 0+1:3+1] * G_kernel) +
       (B[0+1:3+1, 0+1:3+1] * B_kernel))

a23 = ((R[0+1:3+1, 0+2:3+2] * R_kernel) +
       (G[0+1:3+1, 0+2:3+2] * G_kernel) +
       (B[0+1:3+1, 0+2:3+2] * B_kernel))

a31 = ((R[0+2:3+2, 0:3] * R_kernel) +
       (G[0+2:3+2, 0:3] * G_kernel) +
       (B[0+2:3+2, 0:3] * B_kernel))

a32 = ((R[0+2:3+2, 0+1:3+1]* R_kernel) +
       (G[0+2:3+2, 0+1:3+1] * G_kernel) +
       (B[0+2:3+2, 0+1:3+1] * B_kernel))

a33 = ((R[0+2:3+2, 0+2:3+2] * R_kernel) +
       (G[0+2:3+2, 0+2:3+2] * G_kernel) +
       (B[0+2:3+2, 0+2:3+2] * B_kernel))

feature_map =np.array([[a11.sum(), a12.sum(), a13.sum()],
                       [a21.sum(), a22.sum(), a23.sum()],
                       [a31.sum(), a32.sum(), a33.sum()]])
```

```python
        print("Feature map:\n{}".format(feature_map))
```

```
Feature map:
[[ 9 13  9]
 [ 9  8 13]
 [ 5  1  4]]
```

13. 컨볼루션 층의 입력 크기가 32*32*3이고, (a) 10개 5*5 필터들을 보폭 1과 덧대기 2로 적용하였을 때 출력의 크기와 매개변수의 수를 구하세요. (b) 동일한 입력에 64개 3*3필터들을 보폭 1과 덧대기 1로 적용하였을 때 출력의 크기와 매개변수의 수도 구하세요. (6점)

(a) 10개 5*5 필터들을 보폭 1과 덧대기 2로 적용하였을 때 출력의 크기와 매개변수의 수를 구하세요.

매개변수의 수는 필터 5 by 5 이고, 이러한 필터가 10개 이므로, 입력 층의 depth 3 이므로
필터마다 1 개의 bias를 가진다면
총 매개변수는 5 * 5 * 3 * 10 + 10 = 760개
출력의 크기는 W2 * H2 * D2

- W2 = (32-5+2*2)/1+1 = 24
- H2 = (32-5+2*2)/1+1 = 24
- D2 = k = 10

(b) 동일한 입력에 64개 3*3필터들을 보폭 1과 덧대기 1로 적용하였을 때 출력의 크기와 매개변수의 수도 구하세요.

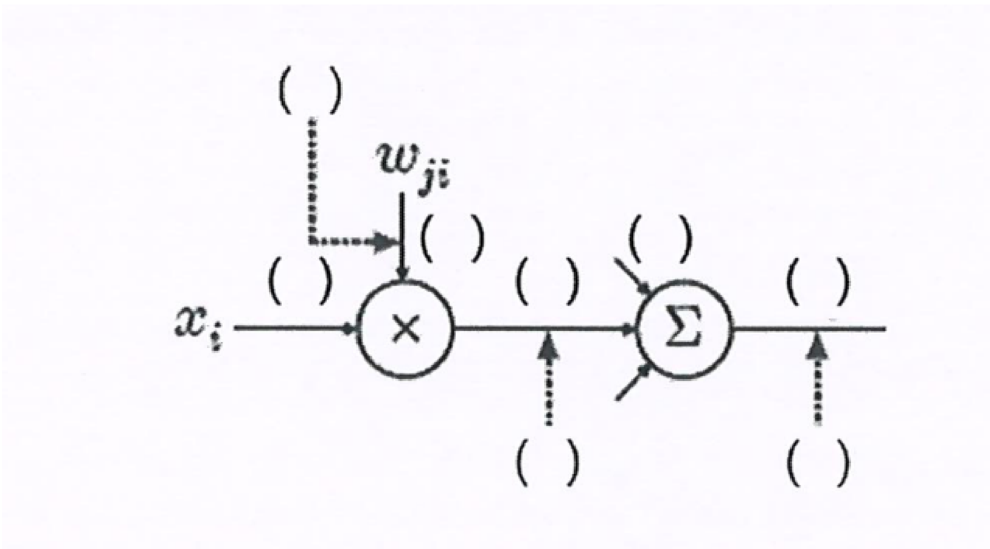
매개변수의 수는 필터 3 by 3 이고, 이러한 필터가 64개 이므로, 입력 층의 depth 3 이므로
필터마다 1 개의 bias를 가진다면
총 매개변수는 3 * 3 * 3 * 64 + 64 = 1792
출력의 크기는 W2 * H2 * D2

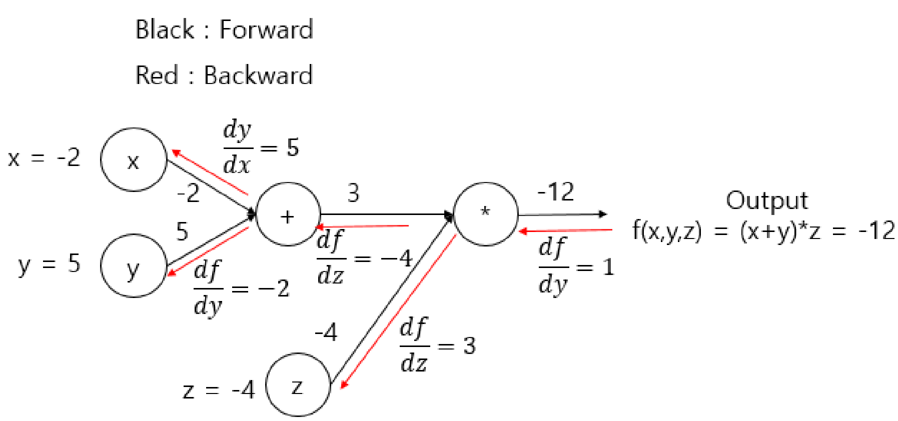
- W2 = (32-3+2*1)/1+1 = 28
- H2 = (32-3+2*1)/1+1 = 28
- D2 = k = 64

14. 아래 그림의 연산 그래프 예처럼 f(x,y,z) = (x+y)z 연산에 대한 연산 그래프를 새롭게 생성하고, x=-2, y=5, z=-4인 경우에 전방 전파와 이에 대응되는 오류 역전파를 각 가중치마다 계산하세요.

[예에 표시된 것처럼 전방 전파 연산 결과는 검은색 빈칸, 오류 역전파 연산 결과는 빨간색 빈칸으로 표시하여 구분하세요.]

sol>


( )
w_{ji}
( )
( )
( )
( )
( )
x_i
×
Σ
( )
( )


Black : Forward
Red : Backward
x = -2
y = 5
z = -4
x
y
z
+
*
-2
5
3
-4
-12
$\frac{dy}{dx} = 5$
$\frac{df}{dy} = -2$
$\frac{df}{dz} = -4$
$\frac{df}{dz} = 3$
$\frac{df}{dy} = 1$
Output
f(x,y,z) = (x+y)*z = -12

1 Reference

- pytorch docs
- Fequently Asked Quest on pytorch docs
- Automatic differentiation
- pytorch tutorial
- How to load data on pytorch