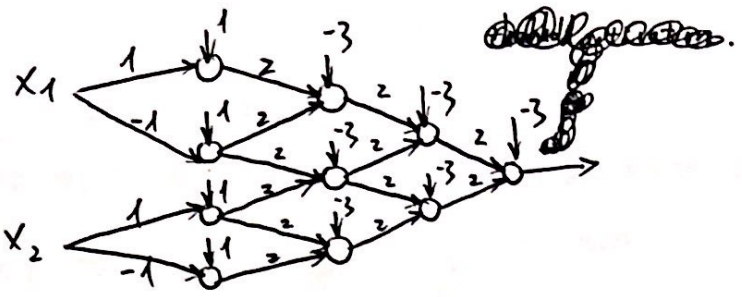
Ex 1. the basical idea of this NN is about to simulate logical gate by besigning NN architecture.

$x_1 + 1 > 0$

$-x_1 + 1 > 0$

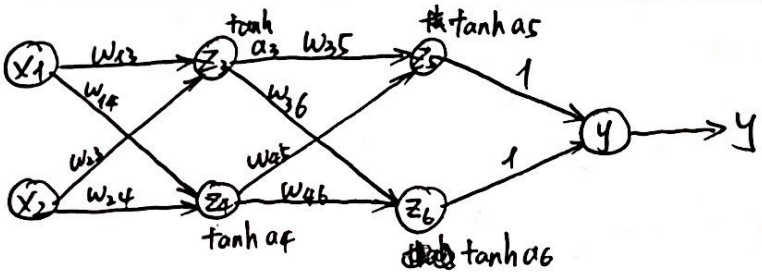
$x_2 + 1 > 0$

$-x_2 + 1 > 0$



the first layer solve the representation of four inequalities respectivly and the last three layers did and operation each two output of the first layer. Then only four inequalities hold, the output of this NN will be greater than 0. ~~The~~ each layers we used threshlold activation. so if inqualities hold output will be 1 for class A and 0 for class B.

Ex 2. a).



b). $\frac{\partial y}{\partial w_{13}}$

$$\frac{\partial y}{\partial z_5} = \frac{\partial y}{\partial a_5}\frac{\partial a_5}{\partial z_5} = 1-(\tanh(z_5))^2$$

$$\frac{\partial y}{\partial z_6} = \frac{\partial y}{\partial a_6}\frac{\partial a_6}{\partial z_6} = 1-(\tanh(z_6))^2$$

$$\frac{\partial y}{\partial a_3} = \left(\frac{\partial y}{\partial z_5}\frac{\partial z_5}{\partial a_3} + \frac{\partial y}{\partial z_6}\frac{\partial z_6}{\partial a_3}\right) = w_{35}\left(1-(\tanh(z_5))^2\right) + \left(1-(\tanh(z_6))^2\right)w_{36}$$

$$\frac{\partial y}{\partial z_3} = \frac{\partial y}{\partial a_3}\frac{\partial a_3}{\partial z_3} = \frac{\partial y}{\partial a_3}\cdot\left(1-(\tanh(z_3))^2\right)$$

$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\cdot\frac{\partial z_3}{\partial w_{13}} = x_1\left(1-(\tanh(z_3))^2\right)\left[w_{35}\left(1-(\tanh(z_5))^2\right) + \left(1-(\tanh(z_6))^2\right)\cdot w_{36}\right]$$

Ex 3.

a) $J(w) = E_{\hat{p}}\left[\frac{1}{2}(w^T x - v^T x - \varepsilon)^2\right]$

$$\frac{\partial J(w)}{\partial w} = E_{\hat{p}}\left[(w^T x - v^T x - \varepsilon)x\right] = E_{\hat{p}}\left[w^T xx - v^T xx - \varepsilon x\right]$$

$$\frac{\partial^2 J(w)}{\partial w \, \partial w^T} = E_{\hat{p}}\left[xx^T\right] \quad \text{in expectation } x \sim N(\mu, \sigma^2 I)$$

$$= \begin{bmatrix} \mu_1^2+\sigma^2 & & \\ & \ddots & \\ & & \mu_d^2+\sigma^2 \end{bmatrix} = \text{diag}^2(\mu) + \sigma^2 \cdot I^{d\times d}$$

b). because Hessian is diagonal Matrix and is positive definite.

then. $\frac{\lambda_1}{\lambda_d} = \frac{\max_i (\mu_i^2+\sigma^2)}{\min_j (\mu_j^2+\sigma^2)} = \frac{\mu_1^2+\cdots+\mu_d^2+\sigma^2}{\sigma^2} = 1 + \frac{\|\mu\|^2}{\sigma^2}$

c). Advantage: It will change the condition of x data distribution. So from a) and b) it will improve the condition number of Hessian, the condition number will approach 1. So each iteration the gradient will point to local minimum, the convergence process will be more stable and free from viberation. The NN will be easier to train. So the NN can naturlich a better performance, i.e. lower prediction error.

Disadvantage:

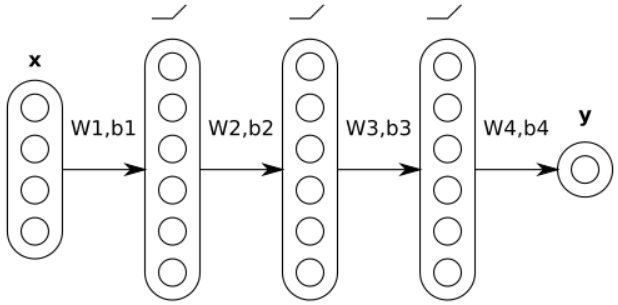
Limitation of Memeory, this will require a reletive large number of Data. For batch training, this will be the main disadvantage.

sheet11

February 1, 2021

1 Error Backpropagation

In this homework, our goal is to test two approaches to implement backpropagation in neural networks. The neural network we consider is depicted below:



1.1 Exercise 1: Implementing backpropagation (20 P)

The following code loads the data and current parameters, applies the neural network forward pass, and computes the error. Pre-activations at each layer are stored in a list so that they can be reused for the backward pass.

[1]:
```
import numpy,utils

# 1. Get the data and parameters

X,T = utils.getdata()
W,B = utils.getparams()

# 2. Run the forward pass
Z1 = X.dot(W[0])+B[0]
A1 = numpy.maximum(0,Z1)
Z2 = A1.dot(W[1])+B[1]
A2 = numpy.maximum(0,Z2)
Z3 = A2.dot(W[2])+B[2]
A3 = numpy.maximum(0,Z3)
Y  = A3.dot(W[3])+B[3];
```

```
# 3. Compute the error

err = ((Y-T)**2).mean()
```

Here, you are asked to implement the backward pass, and obtain the gradient with respect to the weight and bias parameters.

Task:

- **Write code that computes the gradient (and format it in the same way as the parameters themselves, i.e. as lists of arrays).**

[4]:
```
# ----------------------------------------
# TODO: Replace by your code
# ----------------------------------------
def d_relu(aa):
    grad = aa.copy()
    grad[grad>0] = 1
    return grad

delta_4 = 2*(Y-T)
dw_3 = delta_4*A3.T
db_3 = delta_4
delta_3 = delta_4@W[3].T*d_relu(A3)
dw_2 = delta_3*numpy.tile(A2.T,(1,W[2].shape[1]))
db_2 = delta_3
delta_2 = delta_3@W[2].T*d_relu(A2)
dw_1 = delta_2*numpy.tile(A1.T,(1,W[1].shape[1]))
db_1 = delta_2
delta_1 = delta_2@W[1].T*d_relu(A1)
dw_0 = delta_1*numpy.tile(X.T,(1,W[0].shape[1]))
db_0 = delta_1
# form the DW, DB
DW = []
DW.append(dw_0)
DW.append(dw_1)
DW.append(dw_2)
DW.append(dw_3)
DB = []
DB.append(db_0)
DB.append(db_1)
DB.append(db_2)
DB.append(db_3)

# ----------------------------------------
```

To test the implementation, we print the gradient w.r.t. the first parameter in the first layer.

[5]:
```
print(numpy.linalg.norm(DW[0][0,0]))
```

```
1.5422821523392451
```

1.2 Exercise 2: Using Automatic Differentiation (10 P)

Because manual computation of gradients can be tedious and error-prone, it is now more common to use libraries that perform automatic differentiation. In this exercise, we make use of the PyTorch library. You are then asked to compute the error of the neural network within that framework, and this error can then be automatically differentiated.

[6]:
```
import torch
import torch.nn as nn

# 1. Get the data and parameters

X,T = utils.getdata()
W,B = utils.getparams()

# 2. Convert to PyTorch objects

X = torch.Tensor(X)
T = torch.Tensor(T)
W = [nn.Parameter(torch.Tensor(w)) for w in W]
B = [nn.Parameter(torch.Tensor(b)) for b in B]
```

Task:

- **Write code that computes the forward pass and the error in a way that can be differentiated automatically by PyTorch.**

[7]:
```
# ---------------------------------------
# TODO: Replace by your code
# ---------------------------------------
Z1 = torch.matmul(X,W[0]) + B[0]
A1 = torch.relu(Z1)
Z2 = torch.matmul(A1,W[1]) + B[1]
A2 = torch.relu(Z2)
Z3 = torch.matmul(A2,W[2]) + B[2]
A3 = torch.relu(Z3)
Y = torch.matmul(A3,W[3]) + B[3]
err = ((Y-T)**2).mean()
# ---------------------------------------
```

Now that the error has been computed, we can apply automatic differentiation to get the parameters. Like for the first exercise, we print the gradient of the first weight parameter of the first layer.

[8]:
```
err.backward()

print(numpy.linalg.norm(W[0].grad[0,0]))
```

```
1.5422822
```

Here, we can verify that the value of the gradient obtained by manual and automatic differentiation are the same.