In [ ]:

```
```

MATH7502 Project

Topic 4. Deep Learning

Group Member

- Haomingxuan Chen 45585209
- Yaowen Chang 45768262
- Meng Li 45282393

In [ ]:

```
```

Header Files

In [1]:

```
using LinearAlgebra
using Random
```

In [ ]:

```
```

SGD Algorithm

$y = 3x_1 + 4x_2$

Iteration formula

set

α be the $learning\ rate$

θ be the parameter waiting to optimize

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

we have

$\theta_j = \theta_j - \alpha(h_\theta(x^i) - y^i)x^i$

In [2]:

```
function sgd()

    # training set
    x = [1 1; 1 2; 1 3; 2 1; 2 2; 2 3; 2 4; 3 5]
    c = [3; 4] # c is the coefficient
    y = x * c # generate y

    # initialization
    m, d = size(x) # m is the number of the data turple, d is the number of x
    theta = zeros(d) # parameter
    alpha = 0.01 # learning rate
    limit = 0.0001 # threshold of error for stopping the iteration
    error = 0 # original error is zero
    g = 0 # the gradient

    n = 10000 # the number of iteration
    point = 0 # pointer for the stopping location
    for i in 1:n
        j = i % m
        if j == 0
            j = m
        end

        error = 1 / (2 * m) * (((x * theta) - y)' * ((x * theta) - y))
        # stop the iteration
        if abs(error) <= limit
            point = i
            break
        end

        # maintain the theta
        g = x[j, :] * ((x[j, :]' * theta) - y[j])
        theta -= alpha * g
    end

    println("the number of iterations: ", point + 1)
    println("theta: ",theta)
    println("trained formula: ", "y = ", round(theta[1], digits = 2), "x_1 + ", round(theta[2],
digits = 2), "x_2")
    println("error: ",error)
end

sgd()
```

```
the number of iterations: 838
theta: [2.97945, 4.01252]
trained formula: y = 2.98x_1 + 4.01x_2
error: 9.645302679730802e-5
```

In [ ]:

```

```

Adam Algorithm

$$y = 3x_1 + 4x_2$$

Iteration formula

$$g = (h_\theta(x^i) - y^i)x^i$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \times g$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \times g^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_i = \theta_i - \hat{m}_t \times \frac{\alpha}{\sqrt{\hat{v}_t}}$$

In [3]:

```
function Adam()

    # training set
    x = [1 1; 1 2; 1 3; 2 1; 2 2; 2 3; 2 4; 3 5]
    c = [3; 4] # c is the coefficient
    y = x * c # generate y

    # initialization
    m, d = size(x) # m is the number of the data turple, d is the number of x
    theta = zeros(d) # parameter

    alpha = 0.01 # learning rate
    momentum = 0.1 # the momentum
    limit = 0.0001 # threshold of error for stopping the iteration
    error = 0 # original error is zero
    g = 0 # the gradient

    b1 = 0.9 # default parameter
    b2 = 0.999 # default parameter
    eps = 0.00000001 # default parameter
    mt = zeros(d)
    vt = zeros(d)

    n = 10000 # the number of iteration
    point = 0 # pointer for the stopping location
    for i in 1:n
        j = i % m
        if j == 0
            j = m
        end

        error = 1 / (2 * m) * (((x * theta) - y)' * ((x * theta) - y))
        if abs(error) <= limit
            point = i
            break
        end

        g = x[j, :] * ((x[j, :]' * theta) - y[j])

        mt = b1 * mt + (1 - b1) * g
        vt = b2 * vt + (1 - b2) * (g .^ 2)

        mtt = mt / (1 - (b1^(i + 1)))
        vtt = vt / (1 - (b2^(i + 1)))

        # maintain the theta
        theta -= alpha * mtt ./ ([sqrt(vtt[1]); sqrt(vtt[2])] .+ eps)
    end

    println("the number of iterations: ", point + 1)
    println("theta: ",theta)
    println("trained formula: ", "y = ", round(theta[1], digits = 2), "x_1 + ", round(theta[2],
digits = 2), "x_2")
    println("error: ",error)
end
Adam()
```

```
the number of iterations: 5169
theta: [3.02072, 3.98696]
trained formula: y = 3.02x_1 + 3.99x_2
error: 9.984814874907856e-5
```

In [ ]:

BackPropagate Algorithm

Input x

set the corresponding activation a^1 for the input layer

Activation Function $Sigmoid$

$f(x) = \frac{1}{1+e^{-x}}$

Feedforward

for each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$

Output error δ^L

compute the vector $\delta^L = \nabla_a C \times \sigma'(z^l)$

BackPropagate the error

for each $l = L-1, L-2, \ldots, 2$ compute \delta^L = ((w^{l+1}^T\delta^{l+1}) \times \sigma'(z^l)

Output

the gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

In [4]:

```
# single hidden layer neural networks using backpropagate algorithm to train the model

# the average of abs(a)
function mean_abs(a)
    temp_sum = 0
    temp_num = 0
    for i in a
        temp_sum += abs(i)
        temp_num += 1
    end
    return temp_sum / temp_num
end

# use sigmoid as activation function
function sigmoid(x, deriv)
    if deriv == true
        x = (1 .- x) .* x
    else
        e = 2.718281828459
        x = 1 ./ (1 .+ e .^(-x))
        return x
    end
end

# train a model
function train(x, y)

    # using random as initial weights
    m, d = size(x)
    w0 = 2 * rand(d, m) - ones(d, m)
    w1 = 2 * rand(m, 1) - ones(m, 1)

    n = 10000 # number of iteration
    for i in 1:n

        # feedforward
        l0 = x
        l1 = sigmoid(l0 * w0, false)
        l2 = sigmoid(l1 * w1, false)
        l2_error = -(y - l2)

        if (i % 1000 == 0)
            println("error ", mean_abs(l2_error))
        end

        # backpropagate
        l2_delta = l2_error .* sigmoid(l2, true) # error of output_layer
        pd_h2o = l1' * l2_delta # partial derivative from hidden_layer to output_layer

        l1_error = l2_delta * w1'
        l1_delta = l1_error .* sigmoid(l1, true) # error of hidden_layer
        pd_i2h = l0' * l1_delta # partial derivative from input_layer to hidden_layer

        w1 -= pd_h2o
        w0 -= pd_i2h
    end
    return (w0, w1)
end
```

```
# use trained model to test the result
function predict(x, w0, w1)
    l1 = sigmoid(x * w0, false)
    l2 = sigmoid(l1 * w1, false)
    if l2[1] >= 0.5
        println("the prediction of ", x, " is 1")
    else
        println("the prediction of ", x, " is 0")
    end
end


# training set
x = [0 0 1; 0 1 1; 1 0 1; 1 1 1; 0 0 1]
y = [0; 1; 1; 0; 0]

# train the model
w0, w1 = train(x, y)

println("---------------------------")

# test the model
predict([0 1 1], w0, w1)
```

```
error 0.04349112077113367
error 0.025942003856079927
error 0.019702164630193773
error 0.01630461635488493
error 0.014106366357671821
error 0.012542928023103181
error 0.01136224873105968
error 0.010432802357584628
error 0.009678421305883379
error 0.009051576819171896
---------------------------
the prediction of [0 1 1] is 1
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```