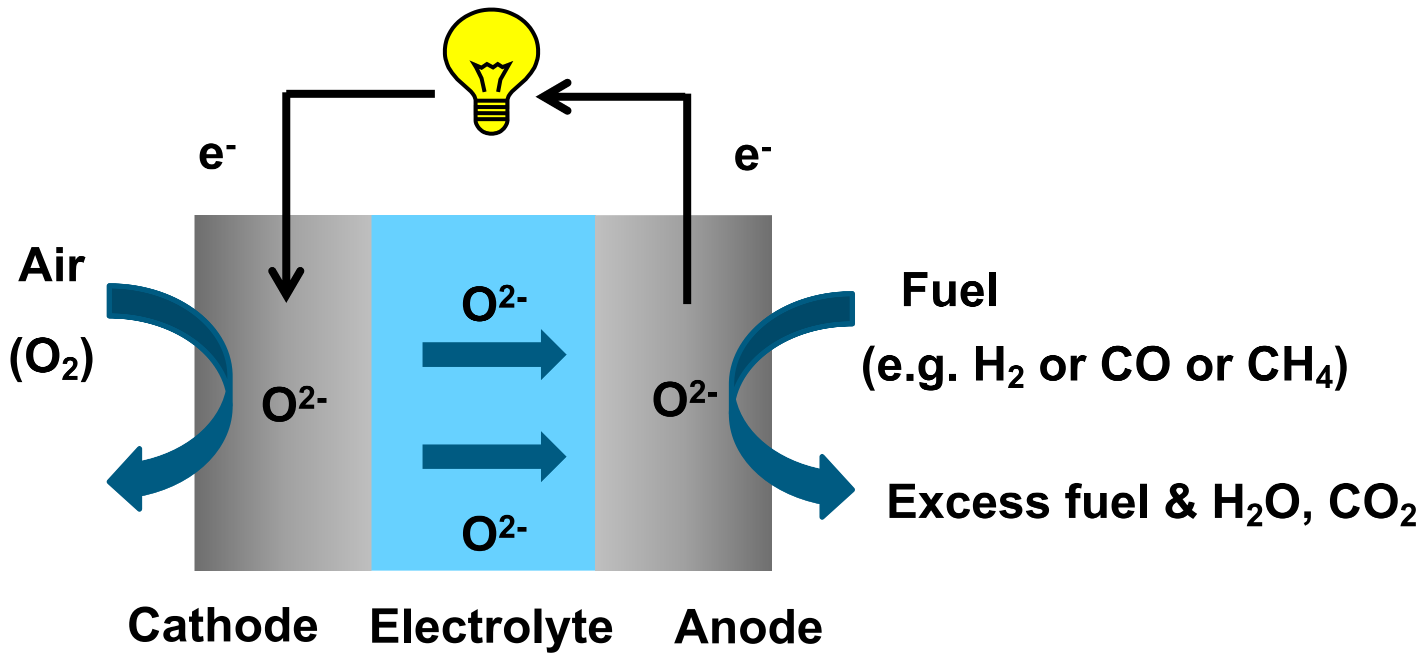
Binary classification

15.-21. Sep. 2018 |

Introduction: Solid Oxide Fuel Cell (SOFC)



Cathode: $O_2 + 2e^- = 2O^{2-}$

Anode: $O^{2-} + H_2 = H_2O + 2e^-$

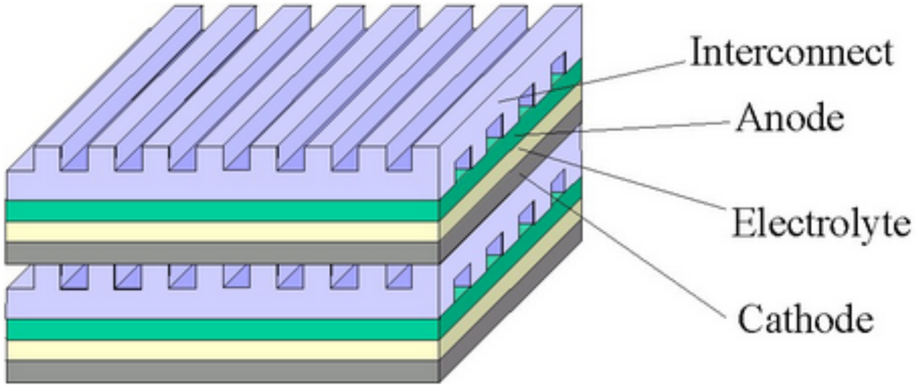
Total: $\frac{1}{2}O_2 + H_2 = H_2O$

- The theoretical cell potential of an SOFC single cell is about 1 V.
- The real potential of the cell is reduced when the current is drawn due to the different polarizations.
- In order to obtain higher voltage, an SOFC stack is required.

Introduction: Solid Oxide Fuel Cell (SOFC)

Interconnect: connects each single cell in series

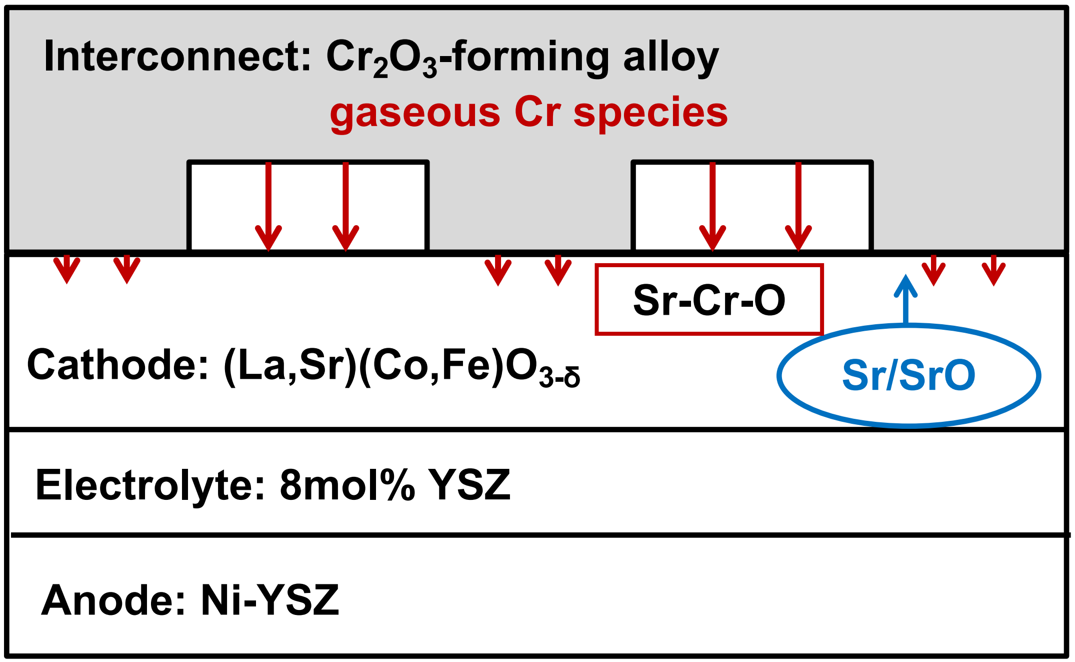
SOFC Stacks



Source: from the Internet

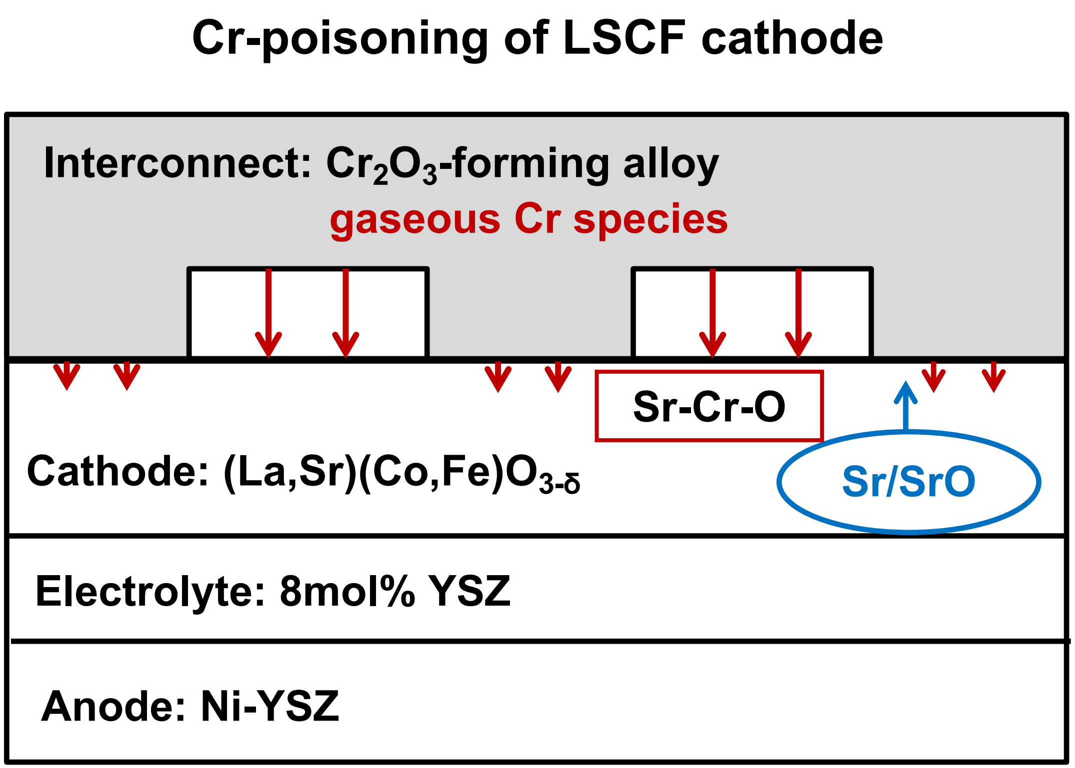
Introduction: Solid Oxide Fuel Cell (SOFC)

Cr-poisoning of LSCF cathode



- In SOFC stacks, Cr-containing steels, e.g. Crofer® 22 APU, are representative candidates of interconnect materials. During operation, a **Cr_2O_3 containing scale forms** on the metallic interconnect and leads to the **evaporation of gaseous Cr species** (e.g. CrO_3 or $CrO_2(OH)_2$) from this scale.

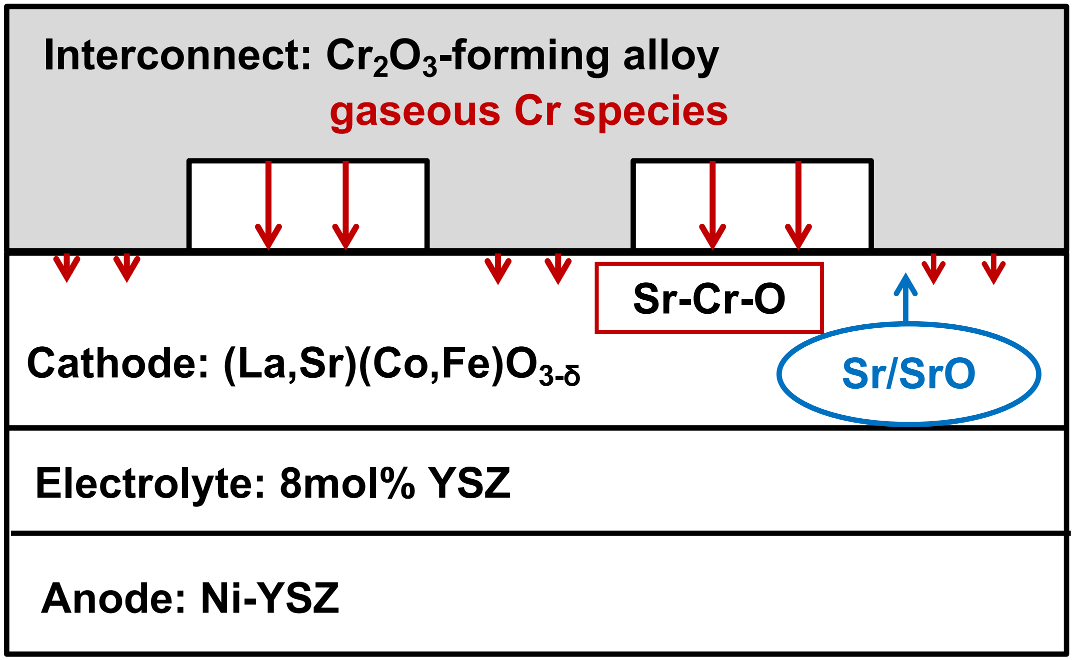
Introduction: Solid Oxide Fuel Cell (SOFC)



- $(La,Sr)(Co,Fe)O_{3-\delta}$ (LSCF) is one of the potential cathode materials in SOFC applications. However, **Sr** in this type of cathode material is a very reactive element. It tends to **segregate out from the LSCF cathode in the form of SrO** and becomes a reaction partner of volatile Cr species.

Introduction: Solid Oxide Fuel Cell (SOFC)

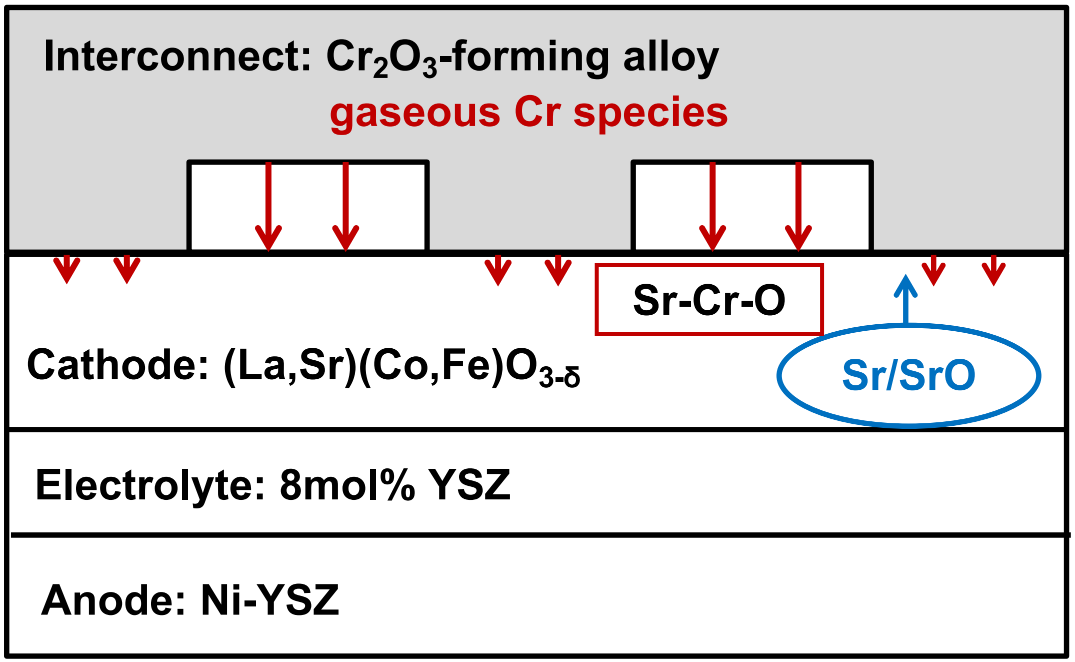
Cr-poisoning of LSCF cathode



- The **segregated SrO** reacts with the evaporated Cr species, **forming Sr-Cr-O** secondary phases that can block the area for oxygen reduction, and subsequently leads to pronounced performance degradation of the SOFC.

Introduction: Solid Oxide Fuel Cell (SOFC)

Cr-poisoning of LSCF cathode



- If dry air is used as cathode gas, CrO_3 will be the dominant gaseous Cr species.
- The reaction between segregated SrO and CrO_3 depends on pCrO3, local pO_2 and temperature.

If the LSCF cathode will be poisoned by Cr or not?

- Task: make a prediction whether there is a Cr poisoning of LSCF cathode or not by given pO_2 and $pCrO_3$ (T=700°C).
- Supervised learning problem:

  Input: pO_2 and $pCrO_3$ Output: poisoning or not

- We can solve this binary classification problem by:

Artificial Neural Network (ANN) (by Tensorflow)
or
Support Vector Machine (SVM) (by Sk-learn)

Prediction by Artificial Neural Network (ANN)

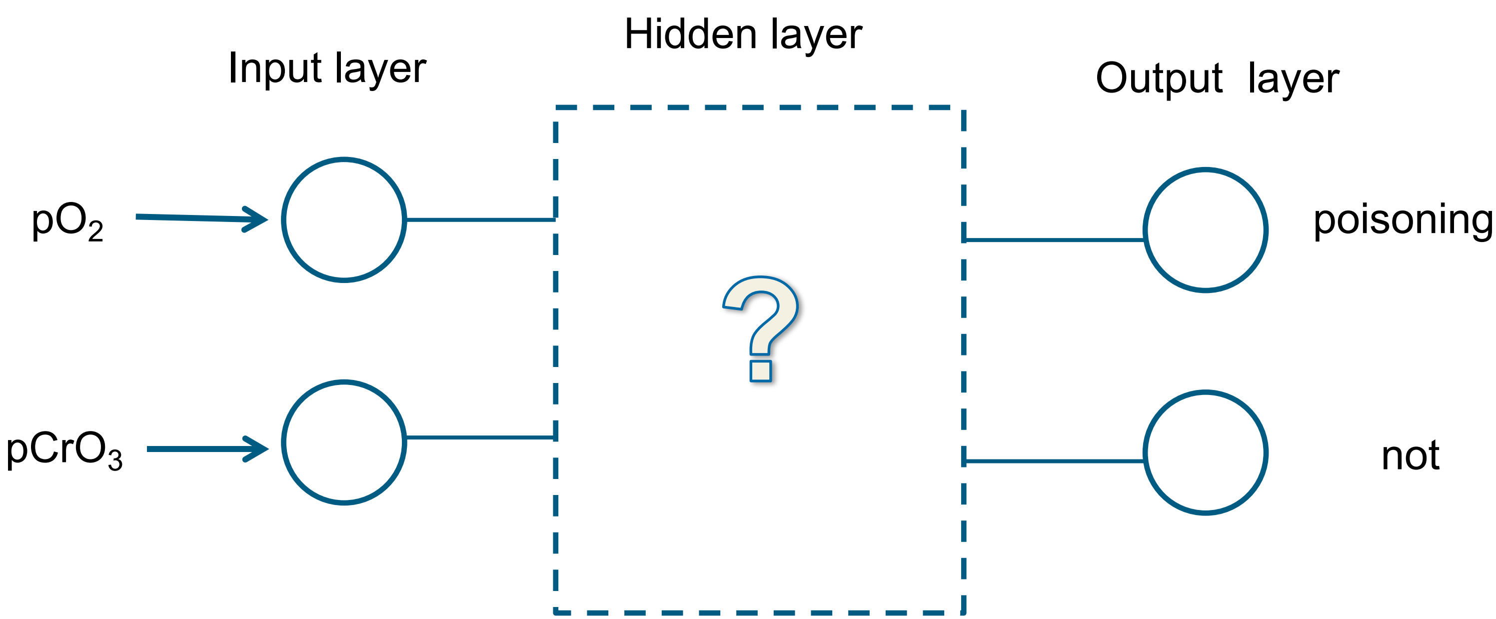
- The architecture of ANN
- Activation functions
- Weight initialization
- Cost function and cost function minimizer
- How to estimate the accuracy
- Training process
- Testing
- Make a prediction

Prediction by Artificial Neural Network (ANN)

- **the architecture of ANN**

Supervised learning problem:

Input: pO_2 and $pCrO_3$ Output: poisoning or not

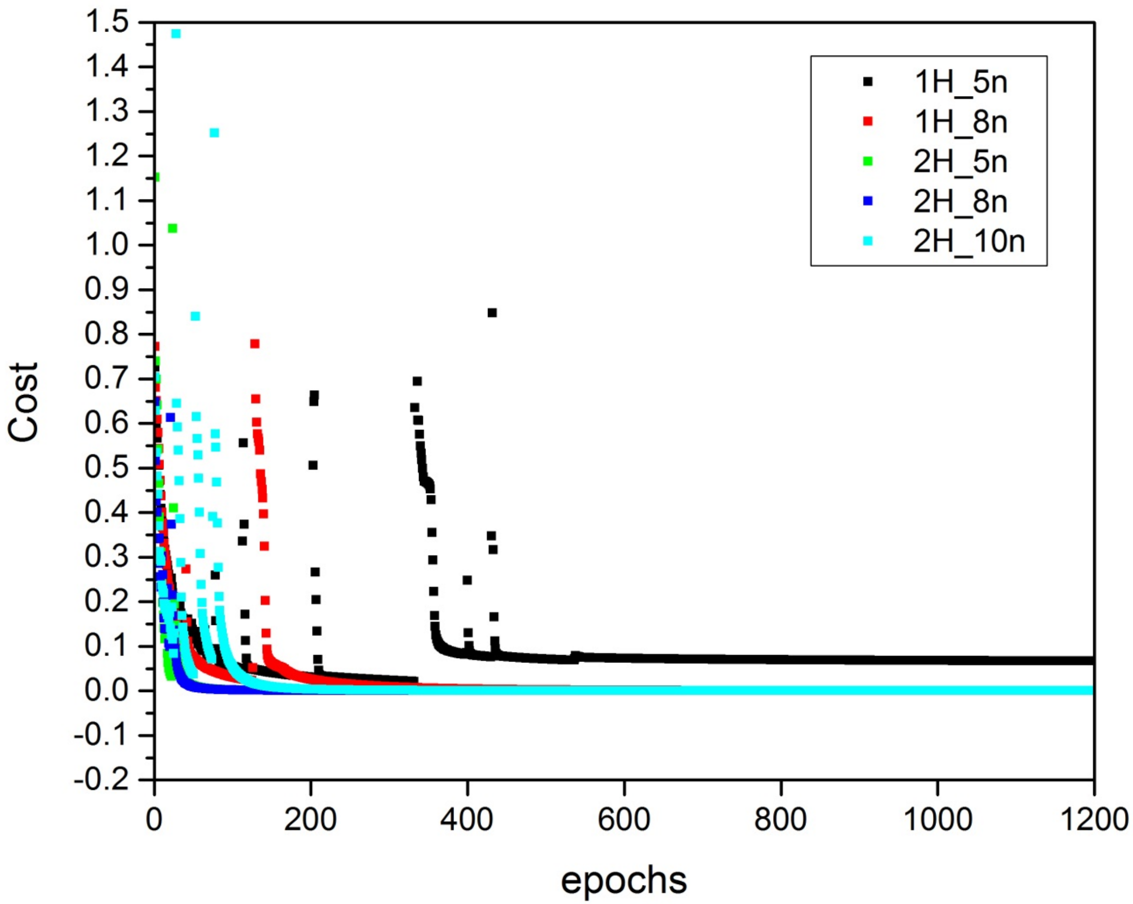


If there is poisoning, the output: 1 poisoning, 0 not means the probability of poisoning is 100%.

Prediction by Artificial Neural Network (ANN)

- **the architecture of ANN**

How does the number of hidden layer and number of neurons in each hidden layer influence the cost?



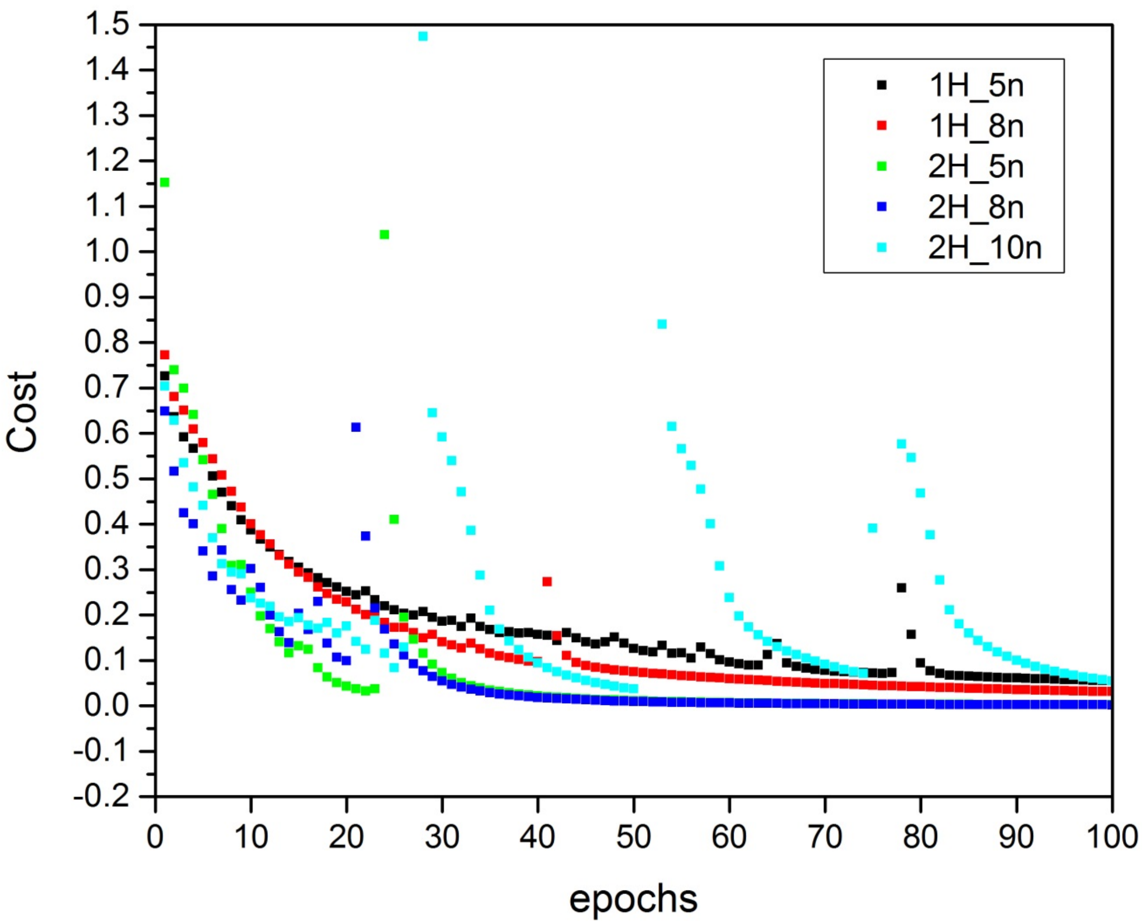
- Learning rate: 0.03, 1200 epochs, batch size: 3, total batch: 9.
- 1H_5n: 1 hidden layer and 5 neurons in each hidden layer

We can choose:
2 hidden layers
5 neurons in each hidden layer

Prediction by Artificial Neural Network (ANN)

- **the architecture of ANN**

How does the number of hidden layer and number of neurons in each hidden layer influence the cost?



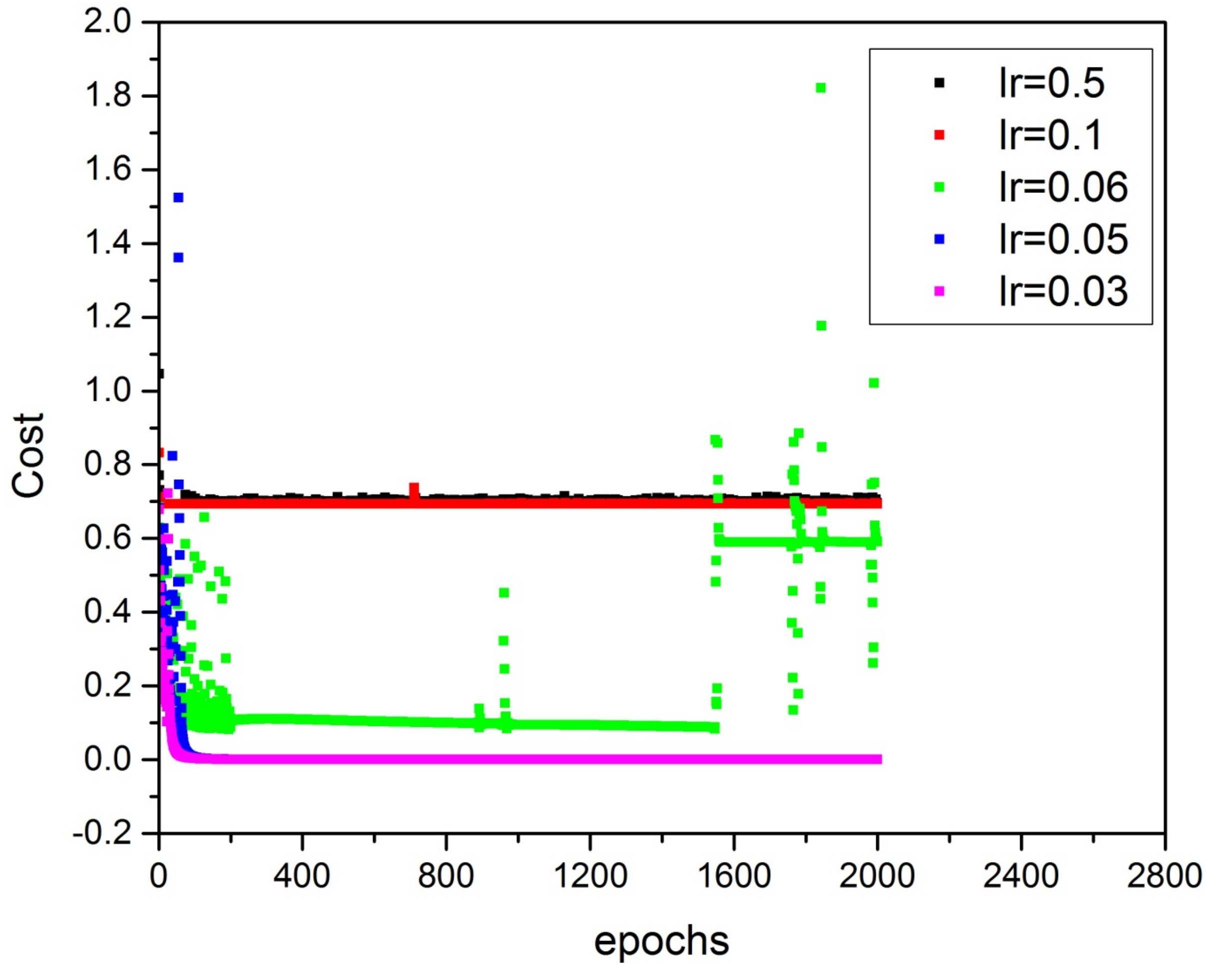
- Learning rate: 0.03, 1200 epochs, batch size: 3, total batch: 9.
- 1H_5n: 1 hidden layer and 5 neurons in each hidden layer

We can choose:
2 hidden layers
5 neurons in each hidden layer

Prediction by Artificial Neural Network (ANN)

- **the architecture of ANN**

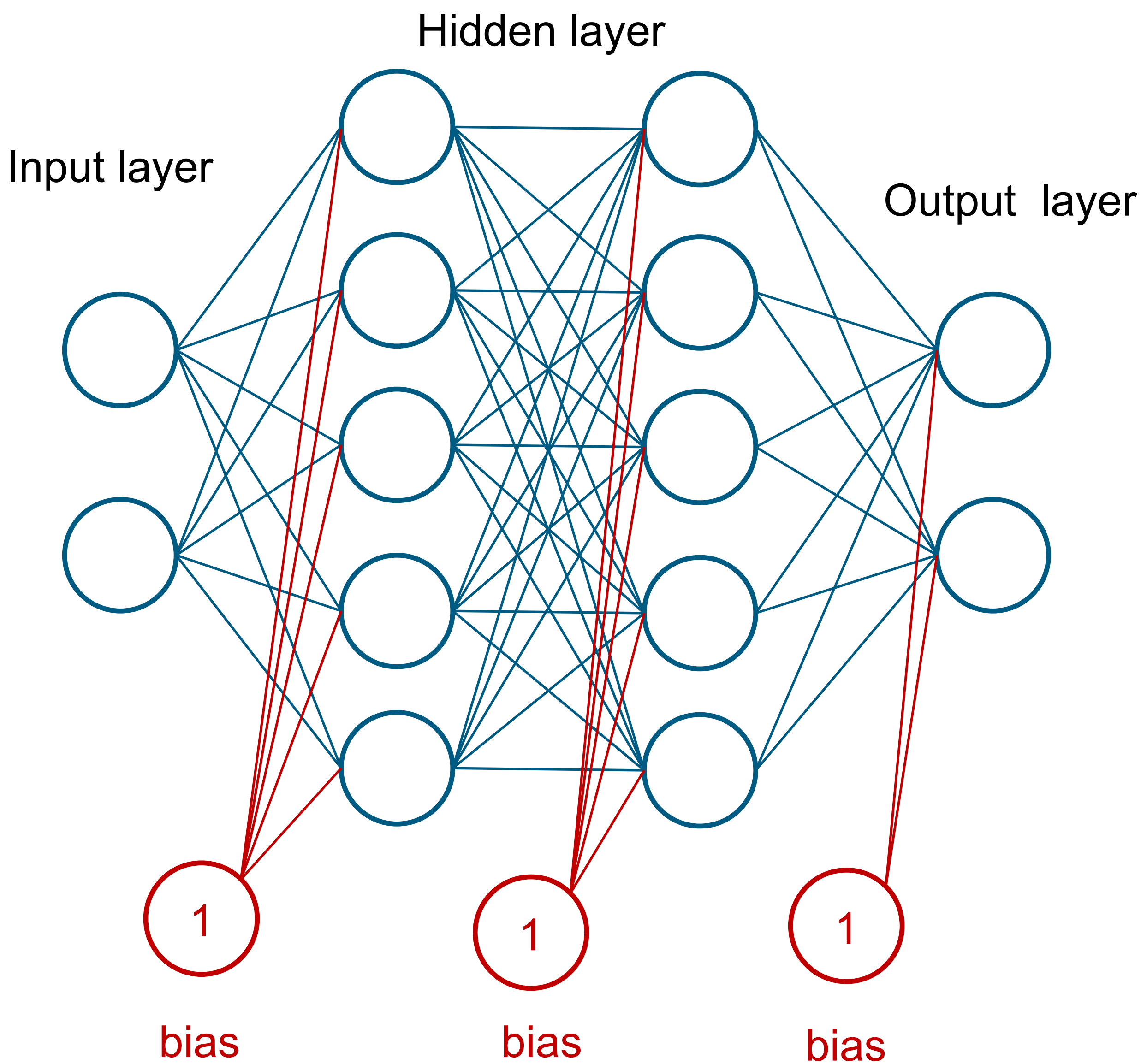
If we take 2 hidden layers and 5 neuron in each hidden layer, how does learning rate (lr) influence the cost?



Mitglied der Helmholtz-Gemeinschaft

Prediction by Artificial Neural Network (ANN)

- **the architecture of ANN**

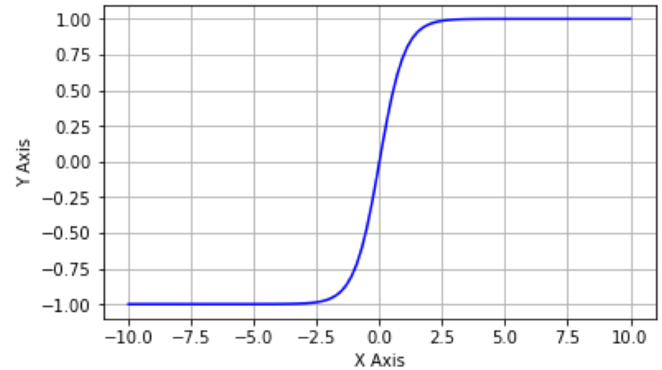


Prediction by Artificial Neural Network (ANN)

JÜLICH FORSCHUNGSZENTRUM

- **Activation functions**

2 hidden layers: $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 1 - \frac{2}{1+e^{2x}}$



Output layer: softmax function $F(z_j) = \frac{e^{z_j}}{\sum_j e^{z_j}}$

It gives the probability.

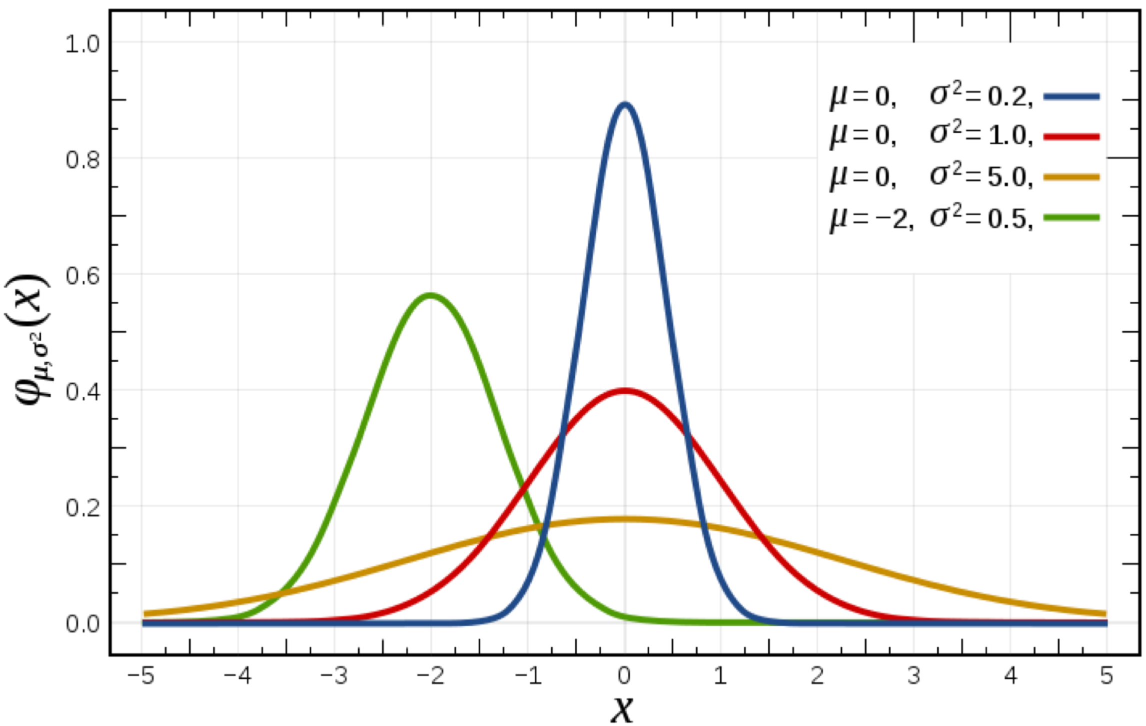
- In our ANN, there are 2 neurons (O_1 and O_2) in the output layer.
- The net input to the 2 neurons in the output layer is $O_{1,\,in}$ and $O_{2,\,in}$, respectively.
- The output of neuron O_1: $O_{1,\,out} = \frac{e^{O_{1,in}}}{e^{O_{1,in}} + e^{O_{2,in}}}$
- The output of neuron O_2: $O_{2,\,out} = \frac{e^{O_{2,in}}}{e^{O_{1,in}} + e^{O_{2,in}}}$

Prediction by Artificial Neural Network (ANN)

- **Weight initialization** e.g. by normal distribution

  - Connecting weights between neurons are initialized by a normal distribution with mean=0 and standard deviation=0.1.

  - Connecting weights between bias and neuron are initialized by a normal distribution with mean=0 and standard deviation=1.



Prediction by Artificial Neural Network (ANN)

- **Cost function and cost function minimizer**

  - **Cross entropy loss function**: commonly used to quantify the difference between the **target probability distribution** $\boldsymbol{y_i}$ and **the predicted probability distribution** $\boldsymbol{\hat{y}_i}$ (if there are N samples, and each labeled by $i = 1,2, \dots N$):

$$L = \frac{1}{N}\sum_{i=1}^{N} -y_i * log_2\hat{y}_i$$

  - For a simple problem of classification (C classes) using the Softmax classifier, most people use the cross-entropy loss function to quantify the objective.

  - The cost function will be reduced through weight updating

Prediction by Artificial Neural Network (ANN)

- **Cost function and cost function minimizer**

  - **Gradient descent minimizer**: $\hat{w}_i := \hat{w}_i - \alpha * \frac{\partial L}{\partial \hat{w}_i}$

    α: learning rate, how fast the $\hat{w}_i$ changes

    L: cost function or loss function or error function

    $\hat{w}_i$: the connecting weights in neural networks

For this problem, Gradient descent minimizer needs much more epochs to reduce cost function.

Prediction by Artificial Neural Network (ANN)

JÜLICH FORSCHUNGSZENTRUM

- **Cost function and cost function minimizer**

  - **Adam optimizer** Adam: Adaptive Moment Estimation √

    Initial condition: weights initialization; $m_0 = v_0 = 0$

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \widehat{w}_{t-1}}$$ (Update biased first moment estimate)

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial \widehat{w}_{t-1}}\right)^2$$ (Update biased second raw moment estimate)

$$\widehat{m}_t \leftarrow \frac{m_t}{1-(\beta_1\ )^t}$$ (Compute bias-corrected first moment estimate)

$$\hat{v}_t \leftarrow \frac{v_t}{1-(\beta_2\ )^t}$$ (Compute bias-corrected second raw moment estimate)

$$\boldsymbol{w_t \leftarrow w_{t-1} - \alpha \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}}$$ (Update weights, α is learning rate; ϵ is a small scalar to prevent division by zero)

Default setting: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$

Mitglied der Helmholtz-Gemeinschaft

Prediction by Artificial Neural Network (ANN)

- **Accuracy estimation**

  - e.g. With a given pO_2 and $pCrO_3$ the target probability distribution is [1, 0], which means the probability of poisoning is 100% and the probability of not poisoning is 0.
  - After several iterations, the predicted probability distribution is: [0.9, 0.1].
  - As a simple estimation, let we say the predicted probability distribution: [0.9, 0.1] gives a correct prediction. Because the probability of poisoning is larger than the probability of not poisoning.

Find the index with the largest value of target probability distribution and predicted probability distribution, respectively, to see if the indexes are the same.

Prediction by Artificial Neural Network (ANN)

- **Training process**
  - Loading data and splitting data into training data and test data
  - Set batch size, learning rate and epochs
  - Start the training process: reducing cost function by updating weights

- **Testing**
  - Using the test data
  - Let the input of test data go through the ANN, it will give us predicted value
  - Compare the predicted value with the target value

- **Make a prediction**
  - The trained model can be used to predict whether the cathode is poisoned by Cr or not by giving certain pO_2 and $pCrO_3$.

How to do it in Tensorflow?

How to do it in Tensorflow?

We have to:

1. **Build a computational graph (e.g. our ANN)**
2. **Run the computational graph (using a tf.Session())**

A **Graph** contains**:**

- a set of **operation objects**: represent units of computation
- **tensor objects**: represent the units of data that flow between operations.

```
import tensorflow as tf

# Build a dataflow graph.
c = tf.constant([[1.0, 2.0], [3.0, 4.0]])
d = tf.constant([[1.0, 1.0], [0.0, 1.0]])     tensor objects
e = tf.matmul(c, d)    # an Operation object

# Construct a `Session` to execute the graph.
sess = tf.Session()

# Execute the graph and store the value that `e` represents in `result`.
result = sess.run(e)

print result
```

Mitglied der Helmholtz-Gemeinschaft

Build ANN (make a graph)

```
# declare input data placeholder X: pO2 and pCrO3
X = tf.placeholder(tf.float32, [None, 2], name="pO2_and_pCrO3")
# declare the output data placeholder  Y - poisoning "[1, 0]" or not "[0, 1]"
Y = tf.placeholder(tf.float32, [None, 2], name='poisoning_or_not')
# declare the weights connecting the input layer to the 1st hidden layer
W1 = tf.Variable(tf.random_normal([2, 5], mean=0, stddev=0.1), name='W1')
b1 = tf.Variable(tf.random_normal([5]), name='b1')
# activation function in the 1st hidden layer, h1_out is the output of the 1st hidden layer
h1_out= tf.nn.tanh((tf.matmul(X, W1)+b1), name='activationLayer1')
# declare the weights connecting the 1st to 2nd hidden layer
W2 = tf.Variable(tf.random_normal([5, 5], mean=0, stddev=0.1), name='W2')
b2 = tf.Variable(tf.random_normal([5]), name='b2')
# activation function in the 2nd hidden layer, h2_out is the output of the 2nd hidden layer
h2_out= tf.nn.tanh((tf.matmul(h1_out, W2)+b2), name='activationLayer2')
# the weights connecting the 2nd hidden layer to the output layer
W3 = tf.Variable(tf.random_normal([5, 2], mean=0, stddev=0.1), name='W3')
b3 = tf.Variable(tf.random_normal([2]), name='b2')
#activation function (softmax) of output layer, Softmax function gives probability; y_pre is the output (prediction) of the ANN
y_pre = tf.nn.softmax((tf.matmul(h2_out, W3) + b3), name='activationOutputLayer')
```

Build ANN (make a graph)

```
# declare input data placeholder X: pO2 and pCrO3
X = tf.placeholder(tf.float32, [None, 2], name="pO2_and_pCrO3")
# declare the output data placeholder Y - poisoning "[1, 0]" or not "[0, 1]"
Y = tf.placeholder(tf.float32, [None, 2], name='poisoning_or_not')
```

Inserts a *placeholder* for a tensor that will be always fed.

tf.placeholder(dtype, shape=None, name=None)

Build ANN (make a graph)


```
# declare the weights connecting the input layer to the 1st hidden layer
W1 = tf.Variable(tf.random_normal([2, 5], mean=0, stddev=0.1), name='W1')
b1 = tf.Variable(tf.random_normal([5]), name='b1')

# activation function in the 1st hidden layer, h1_out is the output of the 1st hidden layer
h1_out= tf.nn.tanh((tf.matmul(X, W1)+b1), name='activationLayer1')
```

A *tf.Variable* represents a tensor whose value can be changed by running ops on it.

```
tf.Variable(<initial-value>, name=<optional-name>)
```

Give random values from a normal distribution:

```
tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, name=None)
```

Computes hyperbolic tangent of x element-wise:

```
tf.nn.tanh(x, name=None)
```

Similar for 2 hidden layer and output layer!

Build ANN (make a graph)

```
#activation function (softmax) of output layer,  Softmax function gives probability
y_pre = tf.nn.softmax((tf.matmul(h2_out, W3) + b3), name='activationOutputLayer')

# to constrain y_pre, make sure there is no log(0)
y_pre_clipped = tf.clip_by_value(y_pre, 1e-10, 0.9999999)

#cost function, cross_entropy
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(y_pre_clipped), 1))

# optimizer
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# Add an operation to initialize all variables
initial = tf.global_variables_initializer()

# Add an operation to save and restore all the variables
saver = tf.train.Saver()
```

Build ANN (make a graph)

```
# define an training accuracy assessment operation
# tf.argmax( ,1) gives the index with the largest value across axis=1;
# tf.equal(x,y) returns the 'true' when x==y
# tf.cast: transfer 'true' to '1' and 'false' to '0'
# tf.reduce_mean:  calculated the mean value

training_correct_prediction = tf.equal(tf.argmax(Y, 1), tf.argmax(y_pre, 1))

training_accuracy = tf.reduce_mean(tf.cast(training_correct_prediction, tf.float32))
```

Build ANN (make a graph)

Loading data

```
f = open('dataset.txt')
X_data = []
Y_data = []
for line in f.readlines():
    line = line.strip()             # remove \n at the end of the line
    columns = line.split()
    col0 = float(columns[0])   # make float out of 0 column
    col1 = float(columns[1])   # make float out of 1 column
    col2 = float(columns[2])   # make float out of 2 column
    col3 = float(columns[3])   # make float out of 3 column
    source1 = [col0, col1]
    source2 = [col2, col3]
    X_data.append(source1)
    Y_data.append(source2)
f.close()

# split data into training data and testing data
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.1)
# output the number of training sample
Num_X_train = len(X_train)
print ("number of training samples:", Num_X_train)
```

Run the computational graph (using a tf.Session())

```
# start the session
with tf.Session() as sess:

  # initialise the variables
  sess.run(initial)
  # calculate total batch
  total_batch_float = tf.divide(Num_X_train, batchsize)
  total_batch = int(total_batch_float)
  print ("total_batch:", total_batch)
  # start the training loop
  for epoch in range(epochs):
      avg_cost = 0
      for i in range(total_batch):
          op, c = sess.run([optimizer, cost], feed_dict={X: X_train, Y: Y_train})
          avg_cost += c / total_batch
     print("Epoch:", (epoch + 1), "cost =", "{:.8f}".format(avg_cost))
 print("training accuracy:", sess.run(training_accuracy, feed_dict={X: X_train, Y: Y_train}))
 save_path = saver.save(sess, './trained_model')
 print("Model saved in file: %s" % save_path)
```