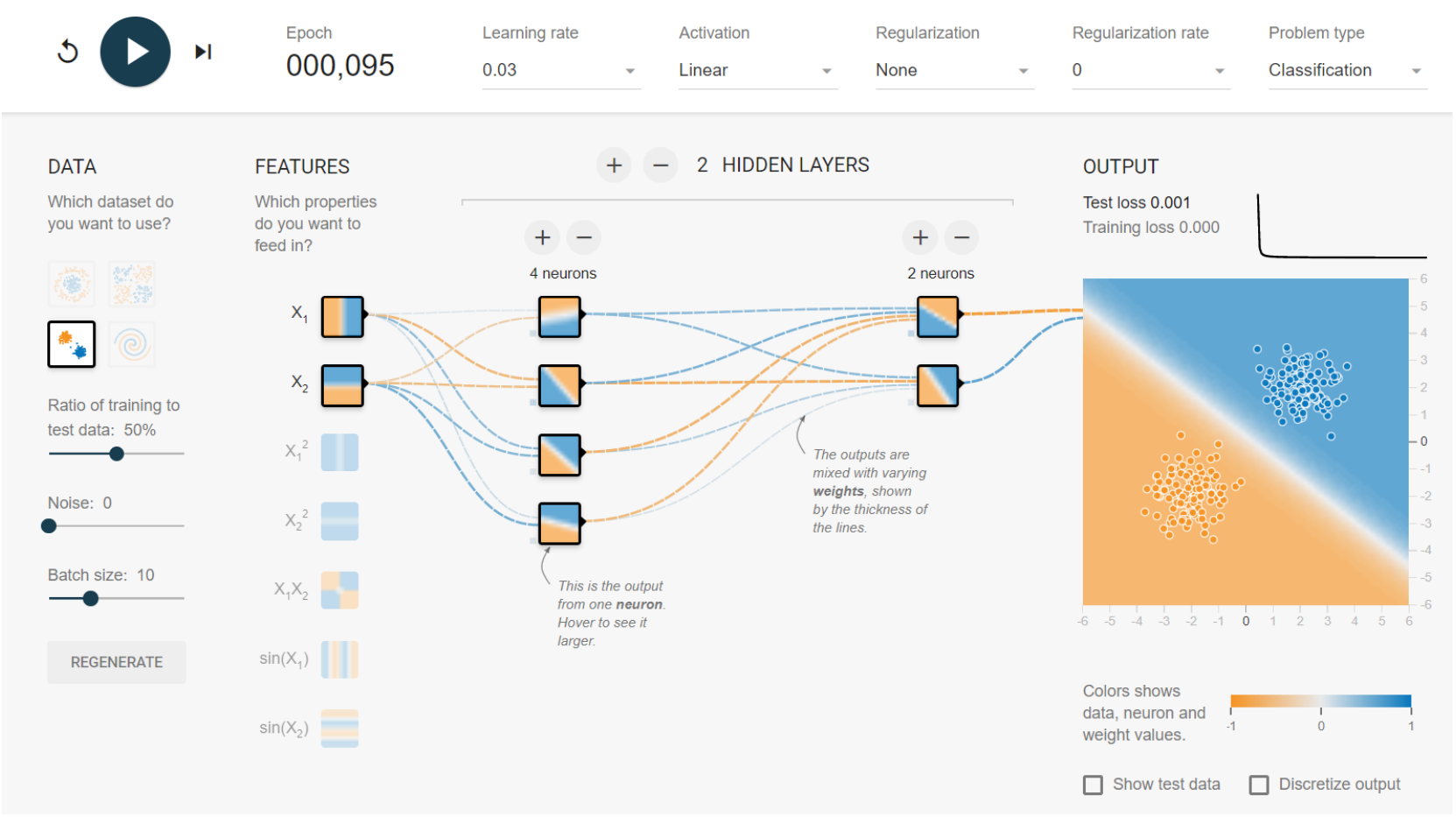
Machine Learning Using Tensorflow

Week2:
Basis of Deep Learning & Regression

Shu-Ting Pi, PhD
UC Davis

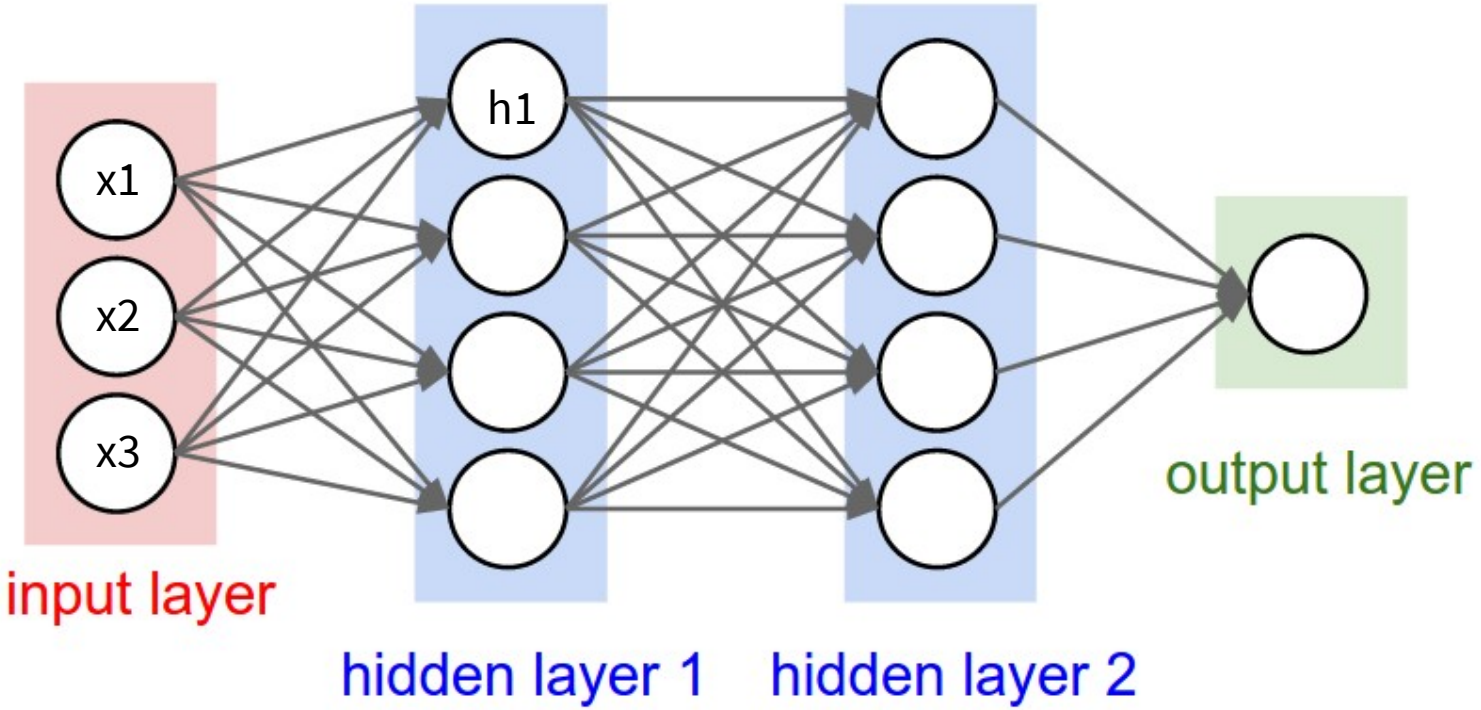
TensorFlow ™

Tensorflow Playground



TF playground is particular useful to give you some hints to fine tune the parameters of your network.

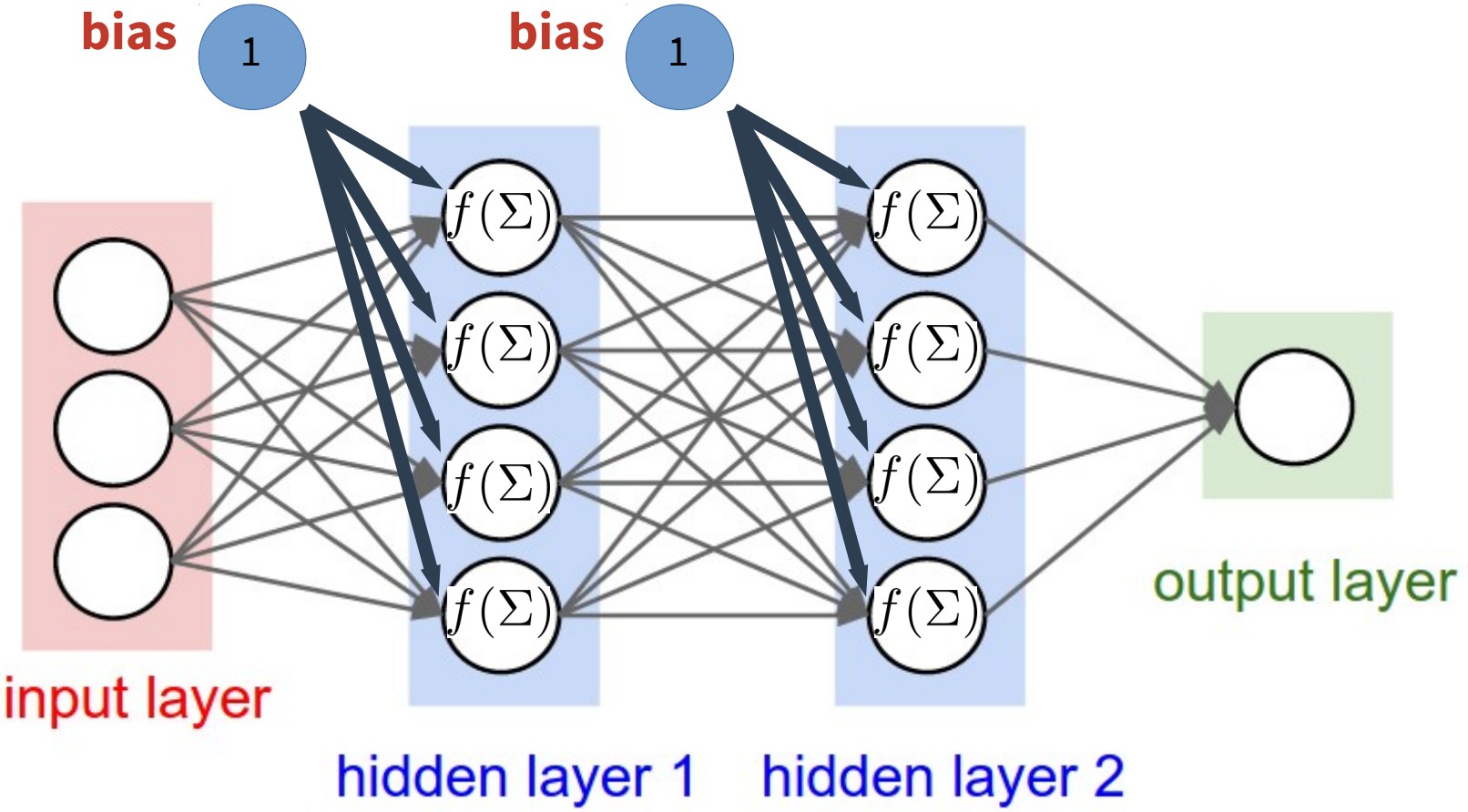
linear neural network



$$h1 = x1 \times w1 + x2 \times w2 + x3 \times w3 \ (h = a * w)$$

- x and h are connected by matrix prduct
- Only works for linear separable data
- multilayer structure is meaningless !

Nonlinear neural network

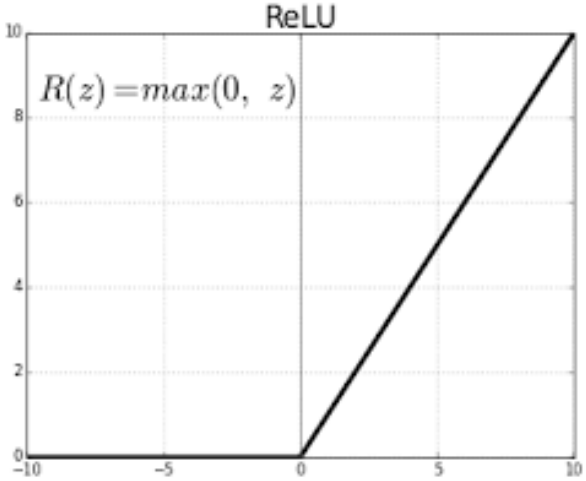


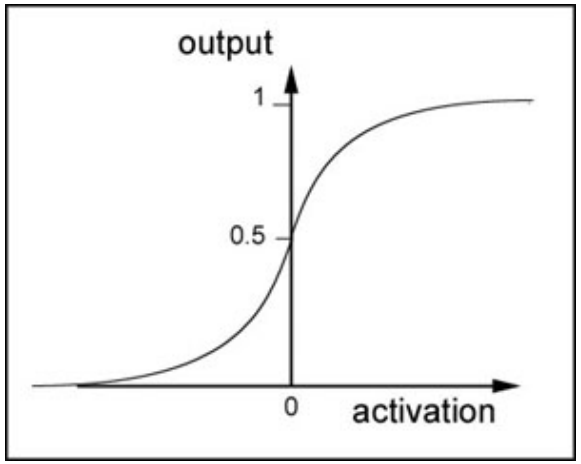
Biases: sensitivity of the node

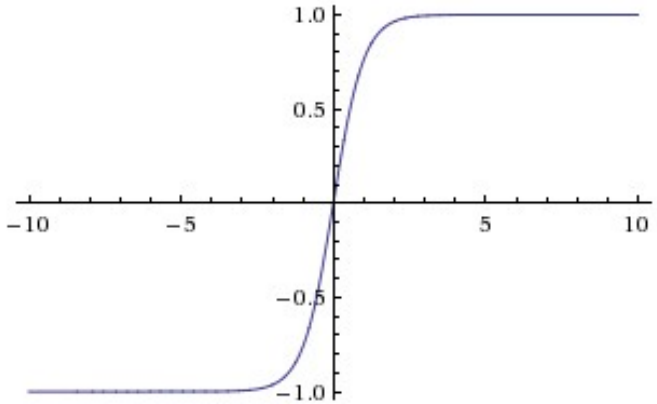
Activation function: nonlinearlize the inputs

Activation functions

How to nonlinearize? Think about human neural network!





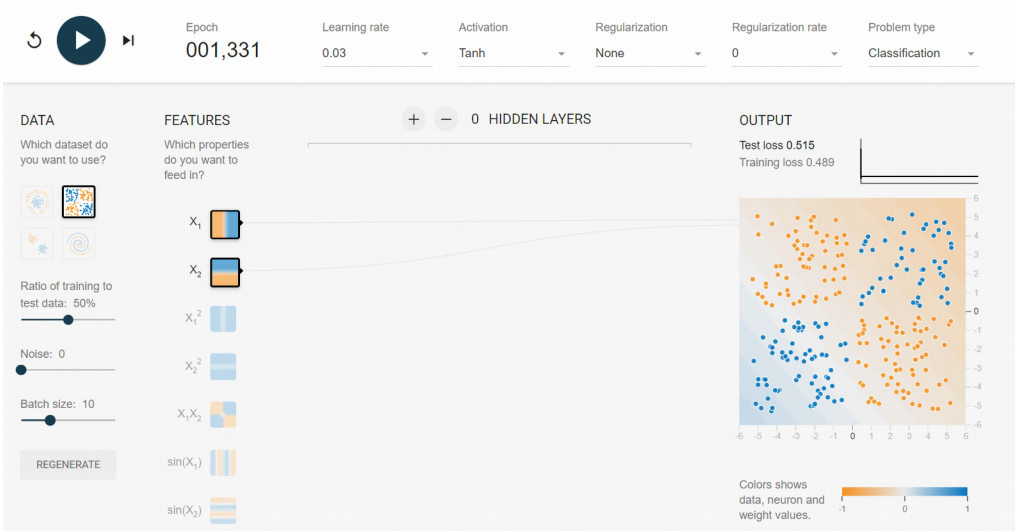


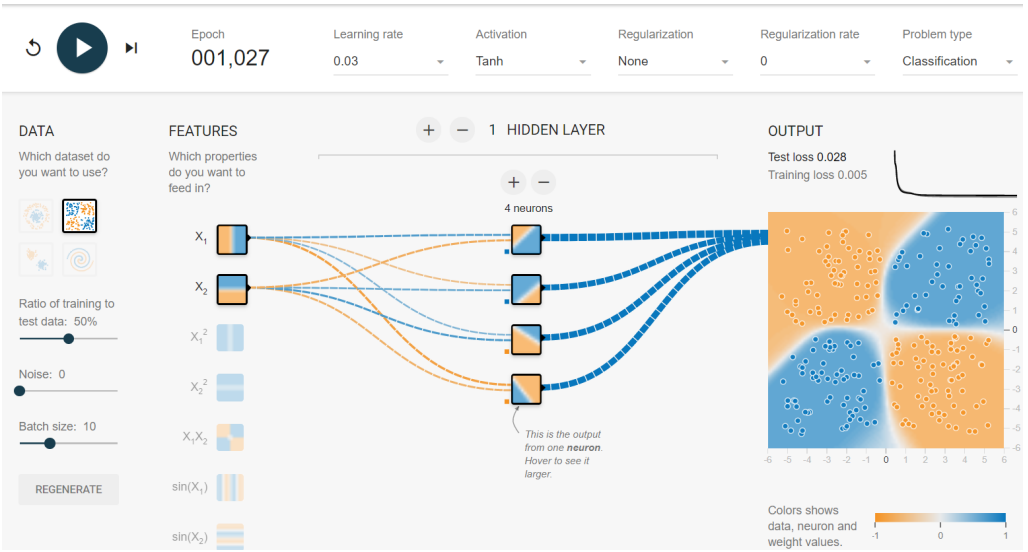
Relu **Sigmoid** **tanh**

Neural nodes become "active" after the input if higher than a threshold and become "numb" if input is too high (sigmoid & tanh).

Why multilayer?

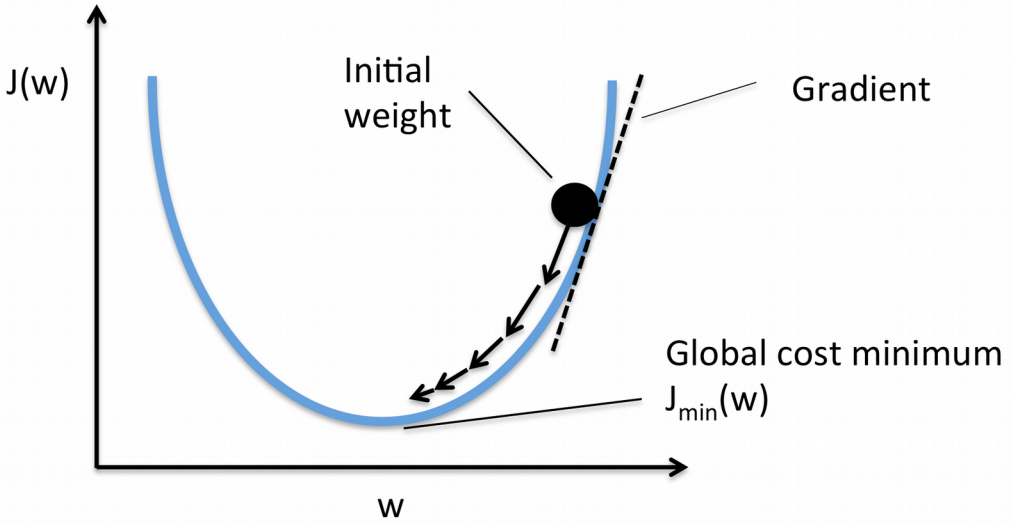
It is proven that single layer NN can not solve “XOR” problem!

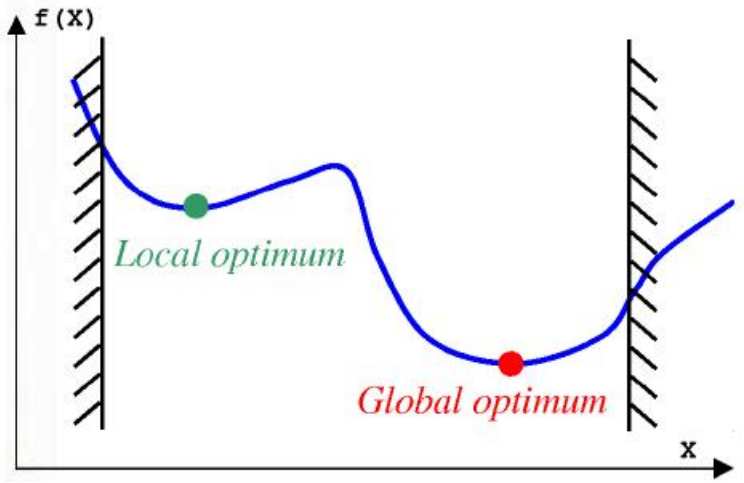




Don't dig into the math, let's prove it using tensorflow playground!

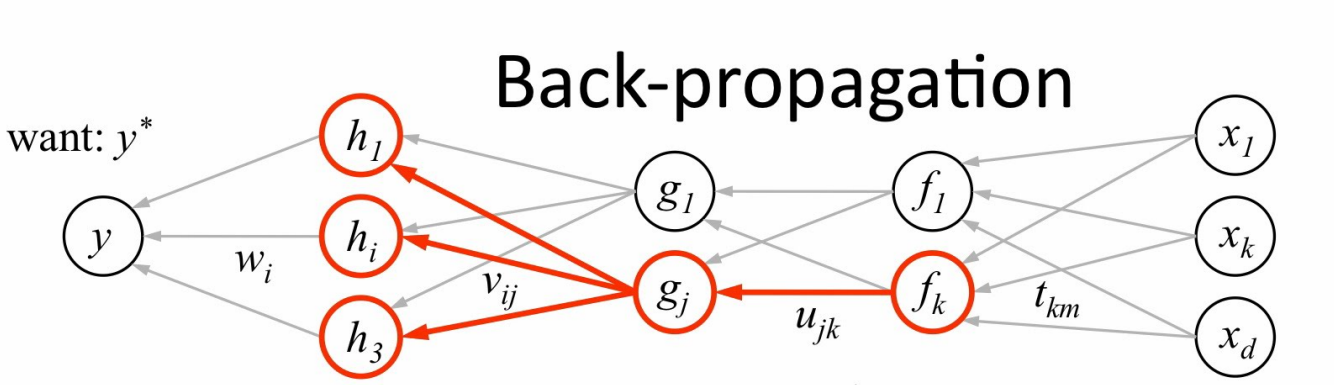
Optimizer





- `tf.train.Optimizer`
- ★ `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- ★ `tf.train.MomentumOptimizer`
- ★ `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- **Alpha go!** `tf.train.RMSPropOptimizer`

Back propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y\text{-}y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

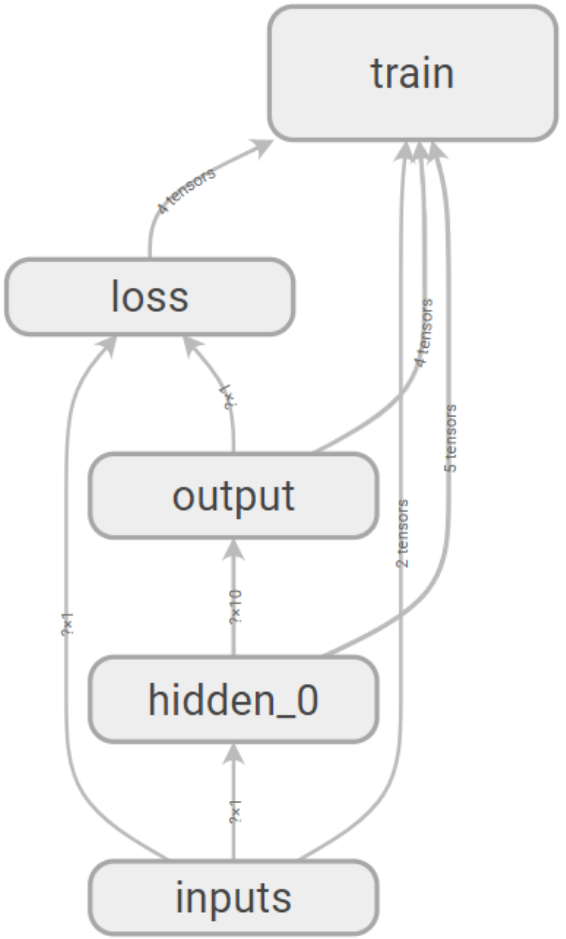
(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

How does tensorflow work?

Construct computation graph

- tf.Variable: variables, default: trainable
- tf.constant: set up constant array
- tf.placeholder: feed until session
- tf.layers.dense: create nn layers
- tf.train.GradientDescentOptimizer: train scheme



Tensorflow session

Make tensors flow

- tf.global_variables_initializer()

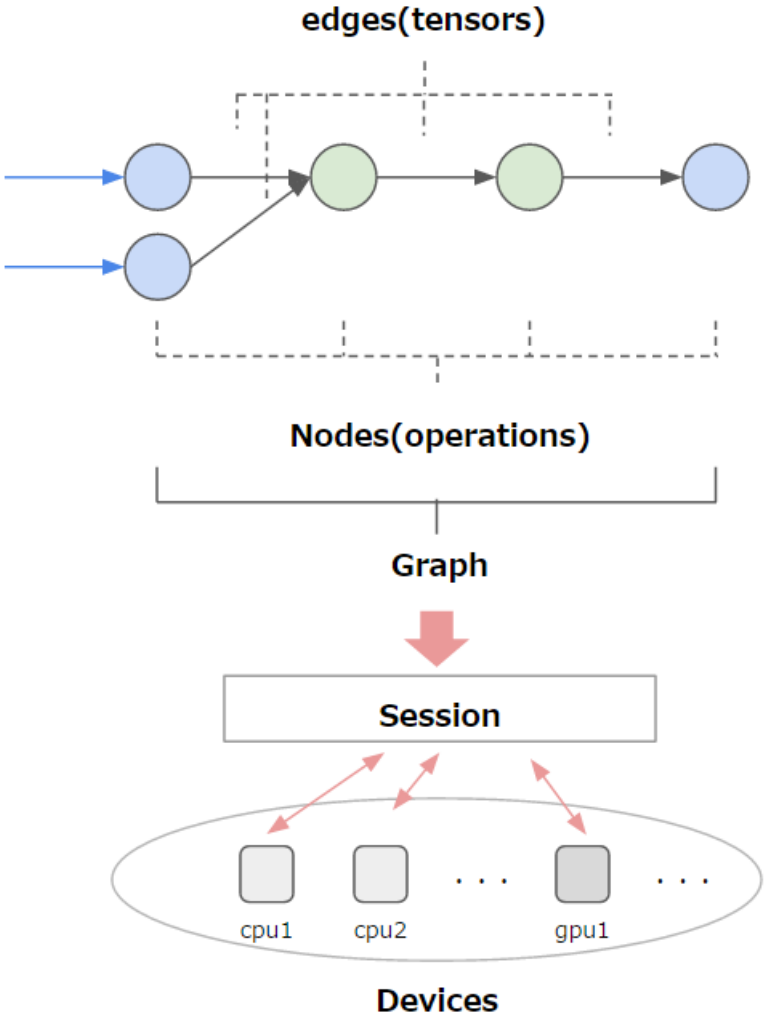
  Initial values assignment operator

- tf.Session()

  Tensorflow session create

- tf.Session.run(, feed_dict={})

  Evaluate results on particular object



Input data

```
14  import tensorflow as tf
15  import numpy as np
16  import matplotlib.pyplot as plt
17
18  # Parameters ========================
19  # input data (y=x**x_power+x_shift+noise)
20  dataset_size=500
21  x_power=5
22  x_shift=0.5
23  noise_std=0.05      # noise standard deviation
24  # layer
25  layer_node=10       # nodes of the hidden layer
26  act_func=tf.nn.relu  # activation_function
27  # train
28  steps=1000          # training steps
29  step_show=100       # number of steps to show results
30  learning_rate=0.1
31
32  # generate data =====================
33  x_data = np.linspace(-1, 1, dataset_size)[:, np.newaxis] # dataset_size x 1
34  noise = np.random.normal(0, noise_std, x_data.shape)
35  y_data = x_data**x_power - x_shift + noise
```

Compution graph

```python
# computation graph =================

# typical layer structure
def add_layer(inputs, dim_in, dim_out, activation_function=None):
    # set weight, initial = random numbers
    Weights = tf.Variable(tf.random_normal([dim_in, dim_out]))
    # set biases, initial = 0.1
    biases = tf.Variable(tf.zeros([1, dim_out]) + 0.1)
    Wx_plus_b = tf.matmul(inputs, Weights) + biases
    # set activation_function
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs

# define placeholder for inputs to network
x_tf = tf.placeholder(tf.float32, [None, 1])
y_tf = tf.placeholder(tf.float32, [None, 1])

# add hidden layer
l1 = add_layer(x_tf, 1, layer_node, activation_function=act_func)

# add output layer
prediction = add_layer(l1, layer_node, 1, activation_function=None)

# the error between prediction and real data
loss = tf.reduce_mean(tf.reduce_sum(tf.square(y_tf-prediction), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
```

Tensorflow session

```
67  # tensorflow session ==================
68  # important step
69  sess = tf.Session()
70
71  # initialize variables
72  init = tf.global_variables_initializer()
73  sess.run(init)
74
75  # plot the real data
76  fig = plt.figure()
77  ax = fig.add_subplot(1,1,1)
78
79  for i in range(steps+1):
80      # training
81      sess.run(train_step, feed_dict={x_tf: x_data, y_tf: y_data})
82      if i % step_show == 0:
83          # evaulate values
84          prediction_value = sess.run(prediction, feed_dict={x_tf: x_data})
85          loss_value=sess.run(loss,feed_dict={x_tf: x_data, y_tf: y_data})
86          # plot the prediction
87          plt.cla()
88          lines = ax.plot(x_data, prediction_value, 'r-', lw=5)
89          ax.scatter(x_data, y_data)
90          plt.title('step={0}, Loss={1:.4f}'.format(i,loss_value), fontdict={'size': 12, 'color': 'green'})
91          plt.pause(0.5)
92
93  plt.ioff()
94  plt.show()
```

Extra Exercise

Improve the code:

code02_2: using batch data set to train NN

code02_3: using tf.dense.layer to generate NN

Questions:

Try change activation functions, number of nodes

Will you always get better results by increasing number of nodes?