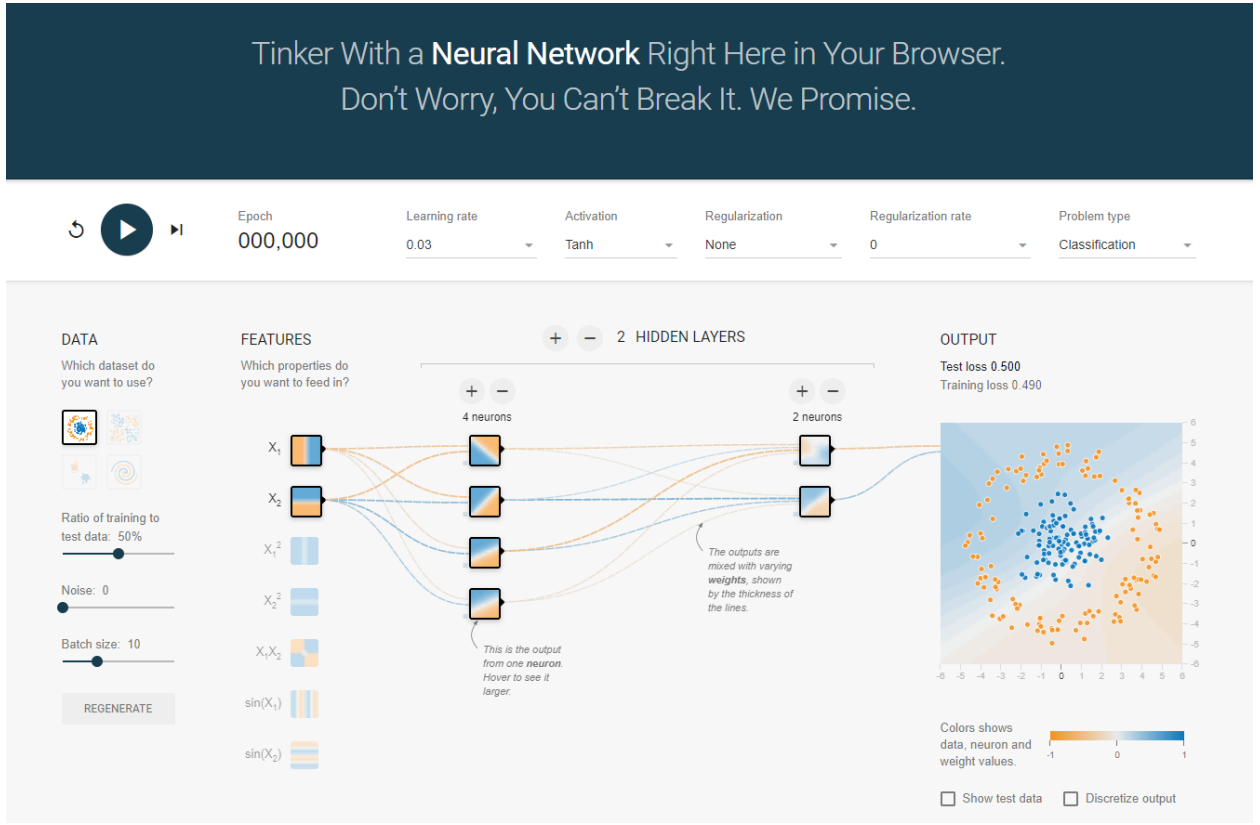
Neural Network II

Li Zeng

Data Mining JBI030

Neural Networks Playground

In this exercise, we will train a series of neural networks through this TensorFlow playground page:



TensorFlow Playground

- **Task 1:** The model as given combines our two input features into a single neuron. Will this model learn any nonlinearities? Run it to confirm your guess.

- The Activation is set to Linear, so this model cannot learn any nonlinearities. The loss is very high, and we say the model **underfits** the data.

TensorFlow Playground

- **Task 2:** Try increasing the number of neurons in the hidden layer from 1 to 2, and also try changing from a Linear activation to a nonlinear activation like ReLU. Can you create a model that can learn nonlinearities? Can it model the data effectively?

- The nonlinear activation function can learn nonlinear models. However, a single hidden layer with 2 neurons cannot reflect all the nonlinearities in this data set, and will have high loss even without noise: it still underfits the data.

TensorFlow Playground

- **Task 3:** Try increasing the number of neurons in the hidden layer from 2 to 3, using a nonlinear activation like ReLU. Can it model the data effectively? How does model quality vary from run to run?

- A single hidden layer with 3 neurons is enough to model the data set (absent noise), but not all runs will converge to a good model. Some runs will do no better than a model with 2 neurons, and you can see duplicate neurons in these cases.

TensorFlow Playground

- **Task 4:** Try 1 neuron in the first hidden layer and increase the depth of the network model

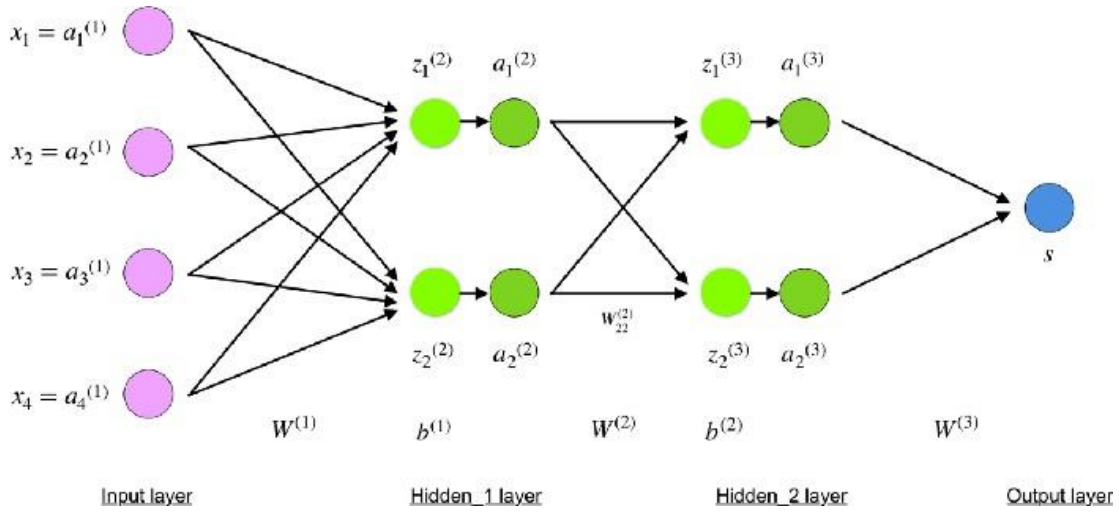
- A model with 1 neuron in the first hidden layer cannot learn a good model no matter how deep it is. This is because the output of the first layer only varies along one dimension (usually a diagonal line), which isn't enough to model this data set well. Later layers can't compensate for this, no matter how complex; information in the input data has been irrecoverably lost.

TensorFlow Playground

- **Task 5**: Continue experimenting by adding or removing hidden layers and neurons per layer. Also feel free to change learning rates, regularization, and other learning settings.

- Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model? For example, try the following architecture:
  - First hidden layer with 3 neurons.
  - Second hidden layer with 3 neurons.
  - Third hidden layer with 2 neurons.

Forward Propagation



$x = a^{(1)}$ *Input layer*

$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at Hidden$_1$ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at Hidden$_1$ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at Hidden$_2$ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at Hidden$_2$ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*

Backpropagation

According to the paper from 1989, backpropagation:

- *repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector.*

and

- *the ability to create useful new features distinguishes back-propagation from earlier, simpler methods…*

Question 1

Which problems we may face when we apply Batch Gradient Descent?

- Slow converge (especially with large training datasets).
- Cost function might easily get stuck in local minimum.
- More oscillations and noisy steps are taken towards the minimum.

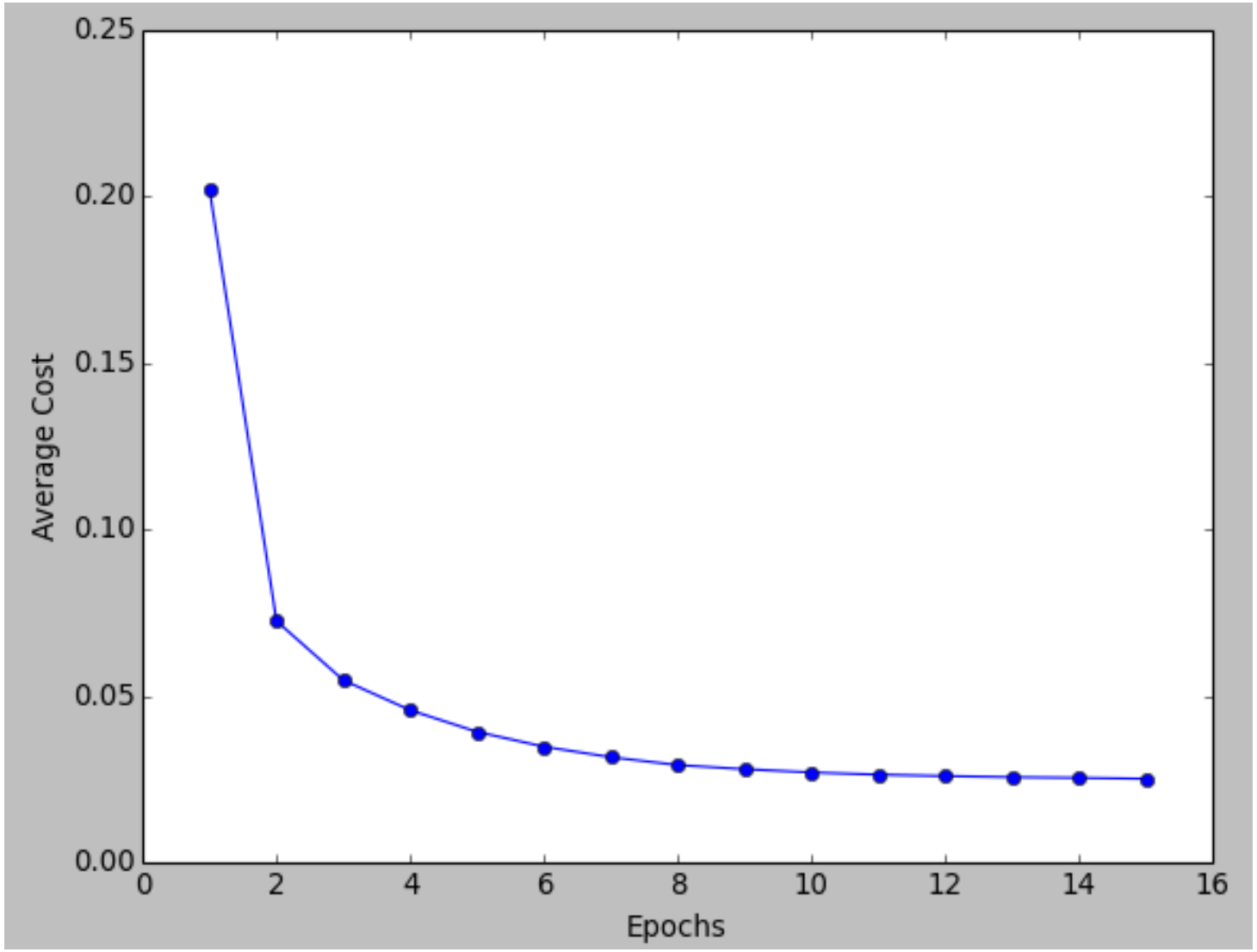
Question 1

Which problems we may face when we apply Batch Gradient Descent?

- Slow converge (especially with large training datasets).
- Cost function might easily get stuck in local minimum.
- More oscillations and noisy steps are taken towards the minimum.

Batch Gradient Descent

- In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.
- Batch Gradient Descent is great for convex or relatively smooth error manifolds.



Question 2

What are the correct statements for Stochastic Gradient Descent?

- It is computationally fast as only one sample is processed at a time.
- It is easier to fit in the memory due to a single training example being processed by the network.
- Due to noisy steps, it may take longer to achieve convergence to the minima of the loss function.

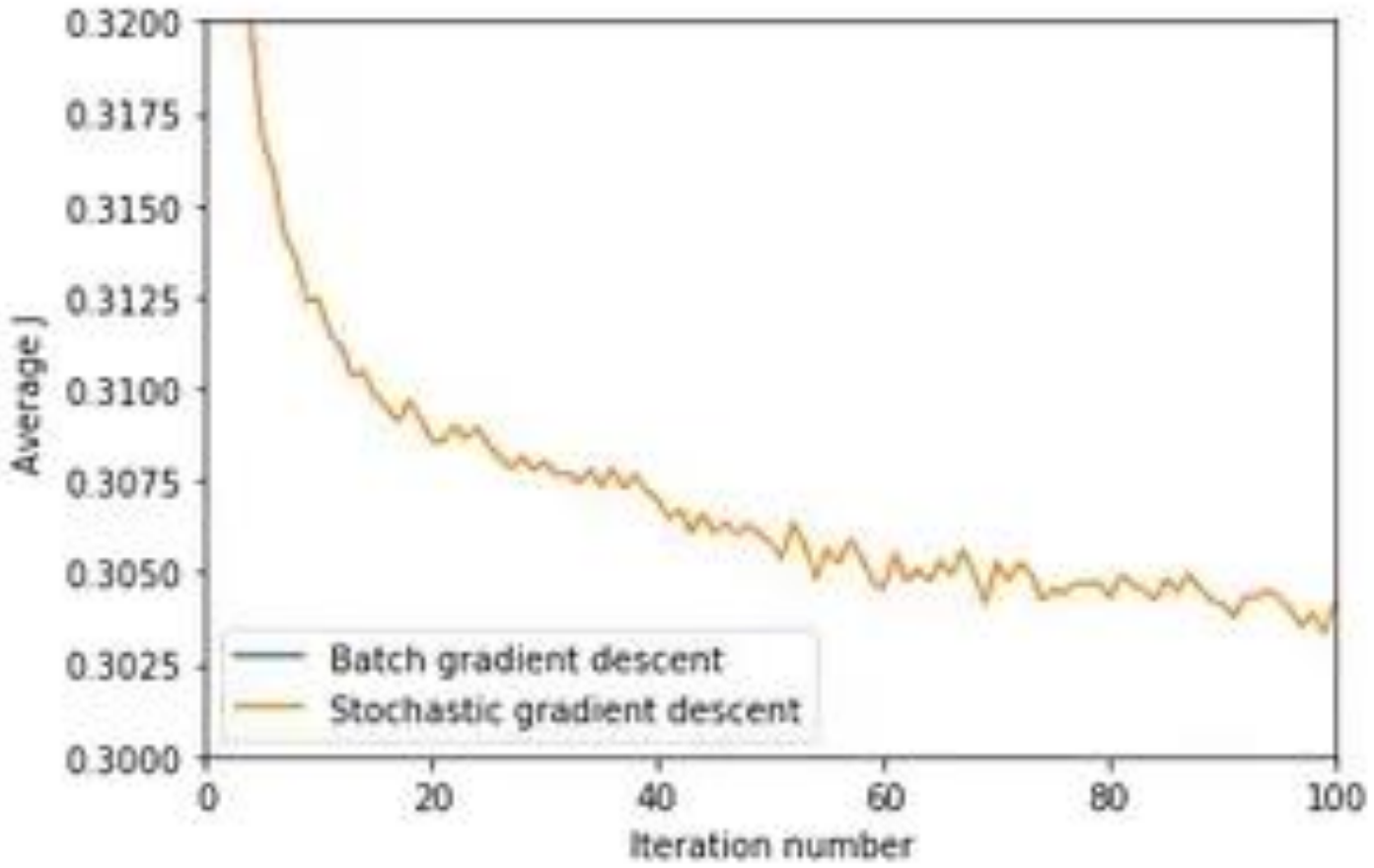
Question 2

What are the correct statements for Stochastic Gradient Descent?

- It is computationally fast as only one sample is processed at a time.
- It is easier to fit in the memory due to a single training example being processed by the network.
- Due to noisy steps, it may take longer to achieve convergence to the minima of the loss function.

Stochastic Gradient Descent

- Since we are considering just one example at a time the cost will fluctuate over the training examples and it will **not** necessarily decrease. But in the long run, you will see the cost decreasing with fluctuations.



Stochastic Gradient Descent

- Because the cost is so fluctuating, it will never reach the minima but it will keep dancing around it.
- SGD can be used for larger datasets. It converges faster when the dataset is large as it causes updates to the parameters more frequently.

Question 3

What can happen if the learning rate is too big in a Gradient Descent problem?

- Training will progress very slowly as you are making very tiny updates to the weights in your network.
- A higher rate could result in a model that might not be able to predict anything accurately.
- Oscillate and never reach the minimum (no converge).

Question 3

What can happen if the learning rate is too big in a Gradient Descent problem?

- Training will progress very slowly as you are making very tiny updates to the weights in your network.
- A higher rate could result in a model that might not be able to predict anything accurately.
- Oscillate and never reach the minimum (no converge).

Question 4

Which statements about Epochs are correct when we train a Neural Network?

- Too many epochs may lead to poor generalization performance of the model.
- The number of epochs for Gradient Descent methods is not crucial for learning.
- Too small number of epochs is not enough to make the model learn well, and it can cause under-fitting
- Epochs should not be considered a hyper parameter and not be optimized during the training of a model.

Question 4

Which statements about Epochs are correct when we train a Neural Network?

- Too many epochs may lead to poor generalization performance of the model.
- The number of epochs for Gradient Descent methods is not crucial for learning.
- Too small number of epochs is not enough to make the model learn well, and it can cause under-fitting
- Epochs should not be considered a hyper parameter and not be optimized during the training of a model.