EECS 189 Introduction to Machine Learning
Fall 2020 DIS8

This discussion was released **Friday, October 23**.

1 Comparing batch, stochastic, and mini-batch gradient descents

In this discussion, we will observe closely how a neural network learns a function. We will first generate some training data. We will train a network using batch gradient descent, SGD, and mini-batch and will contrast these approaches. We will generate animations that show how well the neural network learns the original data as the network is being trained. Open this notebook in datahub and discuss the questions it contains.

Solution:

- **How many weights do we have to train in a neural network with 6 inputs and 3 layers with $d^{(1)} = 3$, $d^{(2)} = 4$, and $d^{(3)} = 2$?**

  We have to train $6 \cdot 3 + 3 \cdot 4 + 4 \cdot 2 = 38$ weights

- **What is the computational cost of evaluating a function implemented as a neural network with L layers and $d^{(\ell)}$ nodes in layer ℓ?**

  To carry out forward propagation at layer ℓ, we compute $s^{(\ell)} = w^{(\ell)} x^{(\ell-1)}$ and $x^{(\ell)} = g\left(s^{(\ell)}\right)$

  Assuming that addition, multiplication, and the evaluation of the activation have the same cost, the cost per layer is $d^{(\ell)}\left(2d^{(\ell-1)} - 1\right) + d^{(\ell)} = 2d^{(\ell)}d^{(\ell-1)}$. Thus, the total computational cost is $2\sum_{\ell=1}^{L} d^{(\ell)}d^{(\ell-1)}$.

- The code needed to compute the relu activation function is

```
### start relu ###
self.innerLayerActivation = lambda a: np.maximum(0,a)
self.innerLayerActivationDer = lambda a : np.heaviside(a, 0.5)
### end relu ###
```

- **What advantages and disadvantages do you see to the three methods above?**

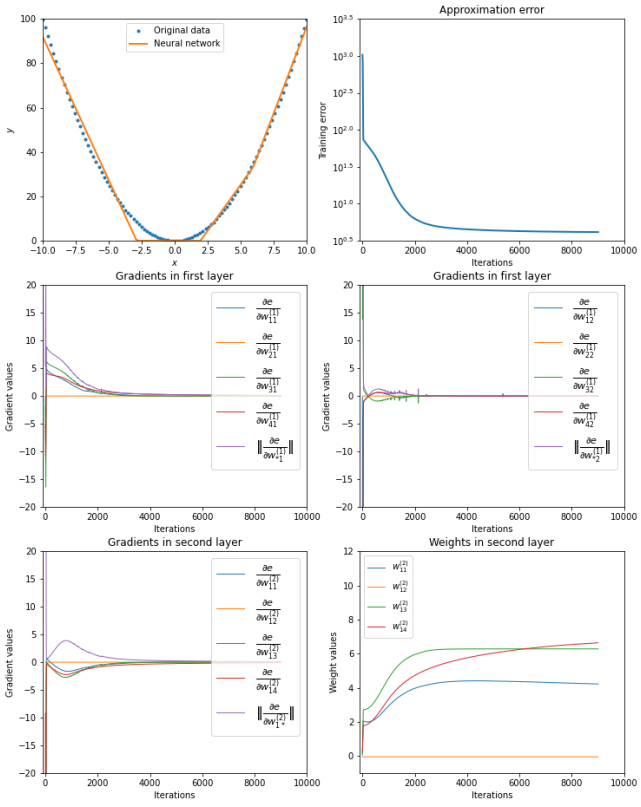
  One can see immediately that batch gradient descent needs to run N samples through the neural network before making an update. SGD just makes one evaluation of the neural network before making an update. Mini-batch is somewhere in the middle. An advantage of batch gradient descent is that every step reduces the total error, while SGD may increase the error at individual steps. On the other hand, this behavior may help SGD from getting stuck at local minima.

- **Looking at the code above, how do you think we can implement batch gradient descent vs. stochastic gradient descent vs. mini batch gradient descent?**

To carry out batch gradient descent, call adjustWeights with all training data. For SGD, call this function using a single sample, and for mini-batch, call it using a subset of the training data.

- Regarding batch gradient descent, considering the animations, we can identify three regions in the plot of the approximation error. **What is the significance of those regions?**

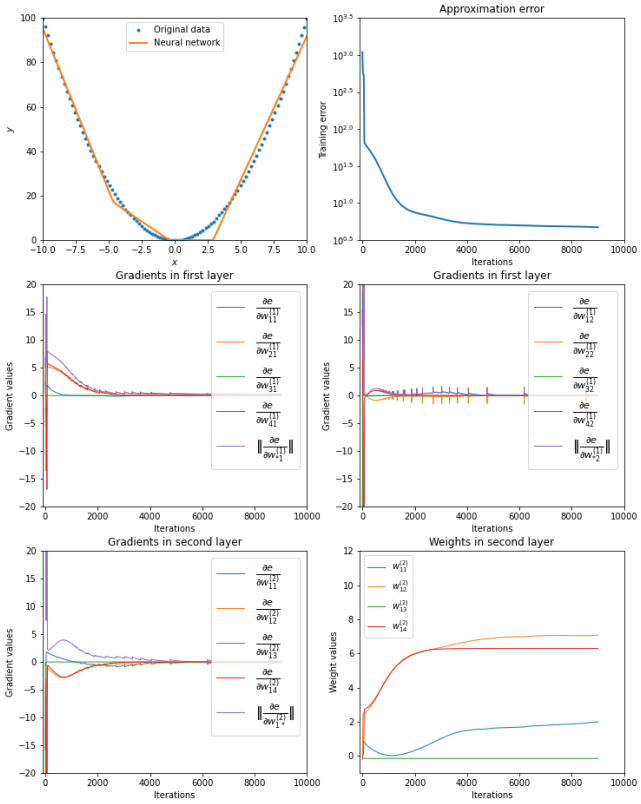
  Below is the last plot of the animation. We can observe that there is a very fast decrease in the error, followed by a moderate decrease, and finally there is a slow decrease. The fast decrease seems to cease when the neural network approximates x^2 using a function resembling $c|x|$, for some constant c. The second region seems to end when the neural network finds an optimal spot for two vertices on the x axis. The last region seems to correspond to the optimization of the position of a third vertex of the graph.



- The only randomness of the training algorithm is in the initialization of the weights of the neural network. **Change the random seed, execute the code again, and discuss what happens**

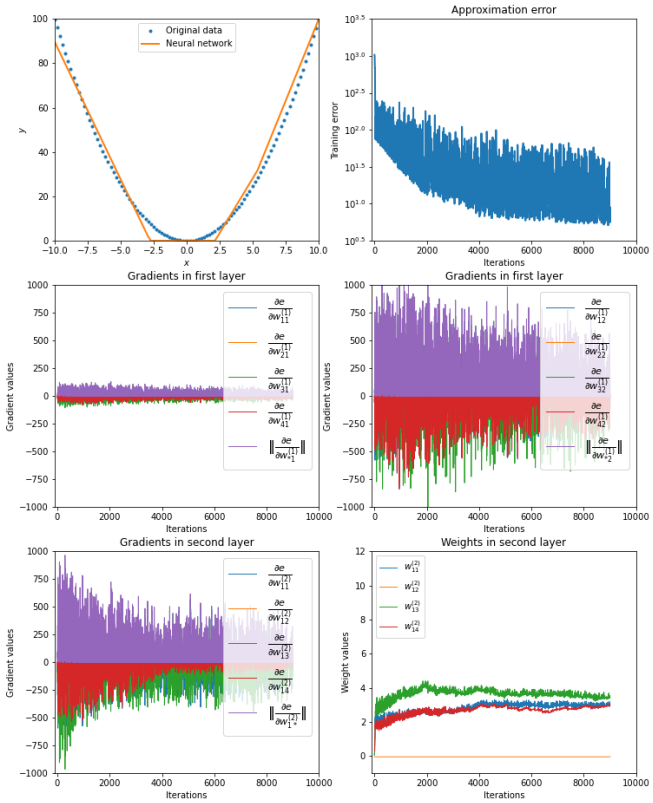
  By changing the seed to 11, we obtain the graph below. We observe the training error behaves

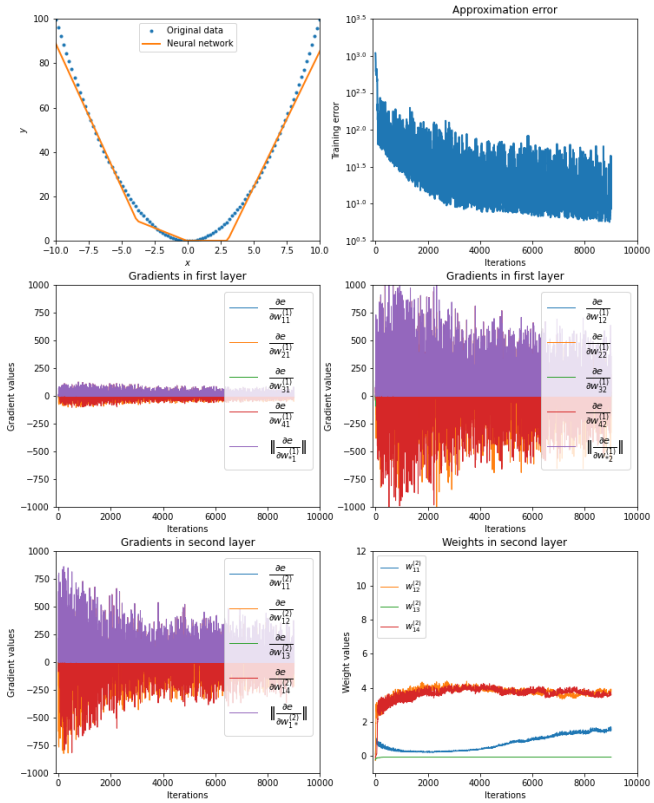
similarly as in the previous case, but the learned function is different from the first. We conclude that the initial weights play a role in the function that is learned.



- Regarding SGD, **generate the visualizations, and discuss what happens**

  The two figures below are the last plots of the animations run with seeds 10 and 11, respectively. We observe that the error has a decreasing trend as a function of the iteration, but the plots are quite noisy.


Original data
Neural network
x
y
Approximation error
Training error
Iterations
Gradients in first layer
Gradient values
Iterations
Gradients in first layer
Gradient values
Iterations
Gradients in second layer
Gradient values
Iterations
Weights in second layer
Weight values
Iterations



- **Compare the gradients computed in batch gradient descent and SGD. Do the SGD gradients converge to zero? Why?**

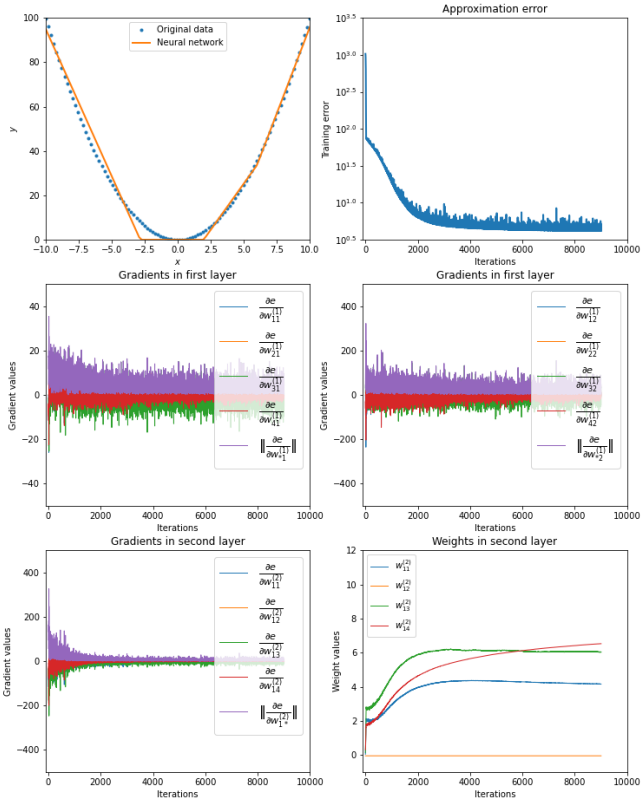
  The gradients converge to zero when using batch gradient descent updates, but they don't when using SGD. This makes sense because at each SGD step we optimize the error seen by a different sample.

- **How does the behavior of stochastic gradient descent compare with that of gradient descent?**

  We observe that the SGD-learned functions are quite similar to the functions learned for the corresponding seeds using batch gradient descent. An important difference is that the error sometimes increases when we make SGD updates to the weights. This happens because the rule for updating weights does not optimize the entire training error, like in batch gradient descent. Moreover, the gradients computed through SGD are much larger than those computed using batch gradient descent.

- Now you have seen the three descent strategies. **Would you be surprised if someone told you that mini-batch gradient descent is the most popular technique? Why?**

The following plot shows the same function learned with mini-batch gradient descent updates. Intuitively, one could say that mini-batch is a middle road between SDG and gradient-descent. The gradients of mini-batch are smaller than those of SGD, and the amount of computation needed to make an update to the weights can be considerably smaller in mini-batch than in batch gradient descent.

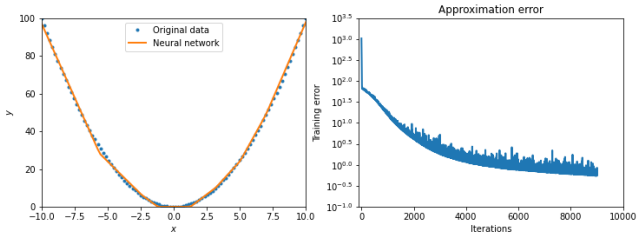


- **Add more nodes to the hidden layer of the neural network using the code below and see what happens.**

  The following code initializes a neural network with 10 middle layers:

```
### start moreNodes ###
nNwk = neuralNetwork([2,10,1])
### end moreNodes ###
```

  The plot below show the result of training this neural network. Qualitatively, we can appreciate that this neural network yields a tighter approximation to the training data, an observation confirmed by the error plot.



The following problems are 5.b and 5.c from homework 8. They are not dependent on 5.a.

5 SGD in the overparametrized regime

This is a problem that helps you understand why we cared so much about the properties of minimum-norm solutions in the context of machine learning. The standard way of training neural networks in practice is stochastic gradient descent (and variants thereof). We need to understand how it behaves. Here, we just try to minimize squared error. (A similar story holds for logistic loss.)

Consider the standard least-squares problem:

$$\min_{\mathbf{w}} L(\mathbf{w}; \mathbf{X}, \mathbf{y}), \text{ where } L(\mathbf{w}; \mathbf{X}, y) := \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2.$$

Here $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ is a $n \times d$ matrix of features and $\mathbf{y}$ is an n-dimensional vector of labels. Say $d > n$. For a single training sample $(\mathbf{x}_i, y_i)$ let $f_i(\mathbf{w}) := \frac{1}{2}(\mathbf{x}_i^\top \mathbf{w} - y_i)^2$. A *stochastic gradient* is $\nabla_{\mathbf{w}} f_i(\mathbf{w}) = (\frac{d}{d\mathbf{w}} f_i(\mathbf{w}))^\top$ where i is uniformly sampled from $\{1, \ldots, n\}$. By the properties of vector calculus, recall that the derivative of a scalar function of a vector is represented by a row vector, and the gradient is the transpose of that row vector so that we have a regular column vector.

(b) **Prove that any linear combination of stochastic gradients must lie in the row span of X.** That is, it must be a linear combination of the $\{\mathbf{x}_i\}$.

(HINT: Here, you need to actually take the derivative and see what it is.)

Solution: Will be released with HW8 solutions.

(c) Suppose that rows of $\mathbf{X}$ are linearly independent, and that stochastic gradient descent with constant step size $\eta > 0$ initialized at $\mathbf{w}_0 = \mathbf{0}$ converges to something when we view the sequence of vectors $\mathbf{w}_t$ as an infinite sequence. **Show that it converges to the minimum norm interpolating solution $\mathbf{w} = \mathbf{X}^\dagger \mathbf{y}$.** Here $\mathbf{X}^\dagger$ is the Moore-Penrose Pseudoinverse. You can feel free to assume that each training point is used infinitely often by SGD.

(Hint: Remember, what does it mean for a sequence to converge? What does that imply about the individual gradients? What does that mean about the quantity $\|\mathbf{X}\mathbf{w}_t - \mathbf{y}\|^2$? Finally, this part comes after the previous part for a reason.)

Solution: Will be released with HW8 solutions.

Contributors:

- Alexander Tsigler
- Anant Sahai
- Inigo Incer