QBUS6850

Lecture 5

Neural Network and Deep Learning- III

BUSINESS SCHOOL

QBUS6850 Team

THE UNIVERSITY OF SYDNEY

❑ Topics covered

- Train a Neural Network
- Neural Network regularization
- Big Data & High Performance Computing (HPC)
- Batch Gradient Descent (BGD)
- Stochastic Gradient Descent (SGD)
- Mini-batch Gradient Descent

❑ References

- Alpaydin (2014), Chapter 11
- Bishop (2006), Chapter 5
- Chu, et al. (2007): Map-Reduce for Machine Learning on Multicore
- Dean and Ghemawat (2008): MapReduce: Simplified Data Processing on Large Cluster

Learning Objectives

- Understand how to train a Neural Network
- Be able to build the forward propagation and backpropagation process in Python step by step and train a Neural Network with gradient descent
- Learn Convolution Neural Networks for general understanding

Train a NN

The following is pseudocode of training our three-layer network (only one hidden layer):

- Randomly initialize weights $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$
- Iterate the following procedure until stopping criterion satisfied:
  1. Implement the **forward propagation**
  2. Calculate the loss
  3. Implement the **backpropagation** and compute **partial derivatives** $\frac{\partial L(W)}{\partial W^{(2)}}$, $\frac{\partial L(W)}{\partial W^{(1)}}$
  4. Use below **gradient descent** to update weights $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$
  5. Incorporate the updated weights into the step 1

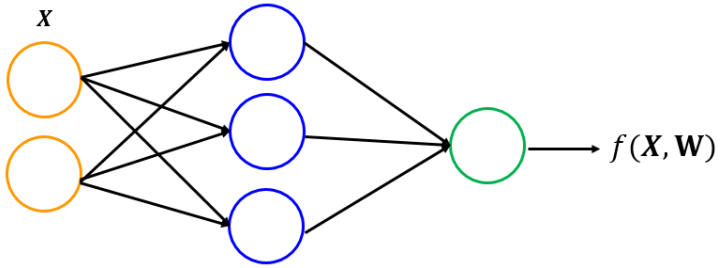
$$\mathbf{W}^{(1)} := \mathbf{W}^{(1)} - \alpha \frac{\partial \mathrm{L}(\mathbf{W})}{\partial \mathbf{W}^{(1)}}$$

$$\mathbf{W}^{(2)} := \mathbf{W}^{(2)} - \alpha \frac{\partial \mathrm{L}(\mathbf{W})}{\partial \mathbf{W}^{(2)}}$$

Python example

```
# initialize the weights
input_layer_size= 2
hidden_layer_size= 3
output_layer_size= 1

# weight parameters
# define W(1): layer 1 to layer 2
np.random.seed(0)
W1= np.random.randn(input_layer_size, hidden_layer_size)
# define W(2): layer 2 to layer 3
W2= np.random.randn(hidden_layer_size, output_layer_size)
```



Randomly initialized weights

```
In [43]: W1
Out[43]:
array([[ 0.4105985 ,  0.14404357,  1.45427351],
       [ 0.76103773,  0.12167502,  0.44386323]])

In [44]: W2
Out[44]:
array([[ 0.33367433],
       [ 1.49407907],
       [-0.20515826]])
```

Updated weights after running the algorithm by 5000 iterations

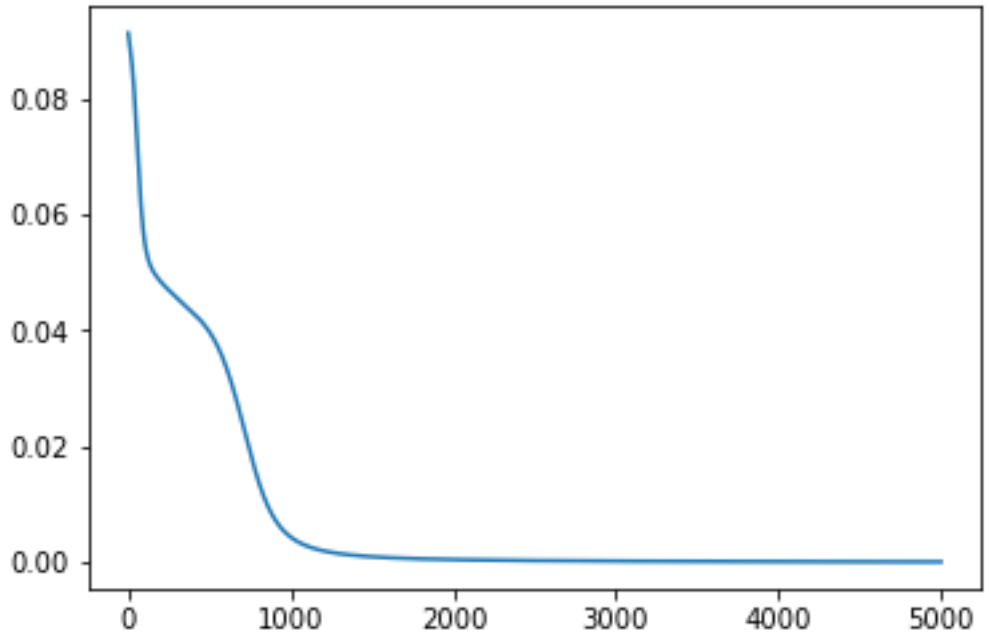
```
In [50]: W1
Out[50]:
array([[ 5.16731003, -3.53264626,  6.55122938],
       [ 3.68593837,  6.68709015, -3.87191772]])

In [51]: W2
Out[51]:
array([[-7.28588035],
       [ 6.26743831],
       [ 6.03562133]])
```

Standard Keras and PyTorch Examples

See: `Lecture05_Example01.py` and `Lecture05_Example02.py`

Loss function plot using gradient descent. numIterations= 5000.



Predicted y and actual y

```
In [722]: y_pred
Out[722]:
array([[ 0.99978688],
       [ 0.4000043 ],
       [ 0.60000046]])

In [723]: y
Out[723]:
array([[ 1. ],
       [ 0.4],
       [ 0.6]])
```

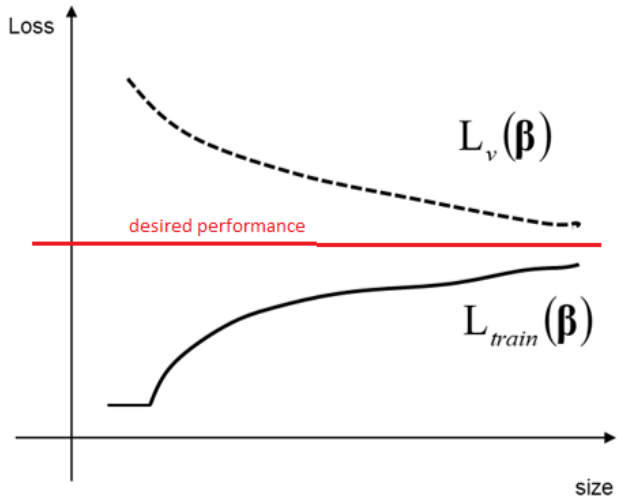
NN Architecture Design

- How to select the number of hidden layers?
- How to select the number of units in each hidden layer?
- How to perform feature selection?
- ...

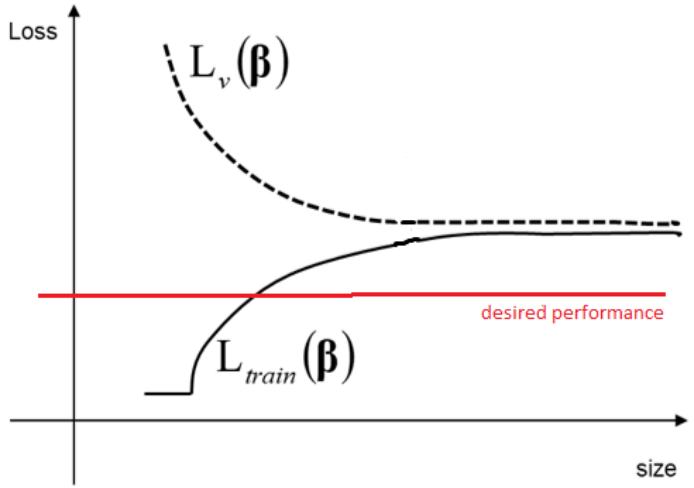
We can still incorporate the model selection techniques to choose the best model, including training & validation & test sets, CV, etc.

When to get more data?

High Variance Problem.



High bias Problem.



- Do a sanity check based on a small subset ($N = 10{,}000$), and check how the learning curves looks like
  - If left (high variance problem), then run the algorithm with more data
  - If right (high bias problem), more data will probably not really help much=> we should take actions on fixing the high bias problem, e.g. use more neurons or hidden layers

Regularization

NN Regression with Regularization

Can we keep having more and more neurons and hidden layers in the NN to improve the fitting on the training data?

- L: total number of layers of the NN
- S_l: number of units (not including bias unit) in the NN layer l
- One output unit

$$L(\mathbf{W}) = \frac{1}{2N}\sum_{n=1}^{N}(f(\mathbf{x}_n, \mathbf{W}) - t_n)^2 + \frac{\lambda}{2N}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(w_{ij}^{(l)}\right)^2$$

(Binary) Classification with Regularization

- L: total number of layers of the NN
- S_l: number of units (not including bias unit) in the NN layer l
- One output unit (binary classification)

$$L(\mathbf{W}) = -\frac{1}{N}\left[\sum_{n=1}^{N}\left(t_n \log\left(f(\mathbf{x}_n, \mathbf{W})\right) + (1 - t_n)\log\left(1 - f(\mathbf{x}_n, \mathbf{W})\right)\right)\right] + \frac{\lambda}{2N}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(w_{ij}^{(l)}\right)^2$$

Regularization with PyTorch

$$L(\mathbf{W}) = \frac{1}{2N}\sum_{n=1}^{N}(f(\mathbf{x}_n, \mathbf{W}) - t_n)^2 + \frac{\lambda}{2N}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(w_{ij}^{(l)}\right)^2$$

$$L(\mathbf{W}) = -\frac{1}{N}\left[\sum_{n=1}^{N}\left(t_n \log\left(f(\mathbf{x}_n, \mathbf{W})\right) + (1 - t_n)\log\left(1 - f(\mathbf{x}_n, \mathbf{W})\right)\right)\right] + \frac{\lambda}{2N}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(w_{ij}^{(l)}\right)^2$$

`Lecture05_Example02.py` for setting λ

THE UNIVERSITY OF SYDNEY

Deep Learning

Big Data

- In 2012, Gartner updated the definition of big data as follows: "Big Data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."
- Big Data has three main characteristics: Volume (amount of data), Velocity (speed of data in and out), Variety (range of data types and sources).
  - **Volume** - Volume describes the amount of data generated by organizations or individuals. Big data doesn't sample: it just observes and tracks what happens.
  - **Velocity** - Velocity describes the frequency at which data is generated, captured and shared. Big data is often available in real-time
  - **Variety** - Big data means much more than rows and columns. It means unstructured text, video, audio that can have important impacts on company decisions – if it's analyzed properly in time.

https://en.wikipedia.org/wiki/Big_data

Big Data Examples

- Twitter CEO Dick Costolo revealed that the company has gone from 90 million tweets per day in September of 2010 to 100 million at the beginning of 2011 to 1/4 billion tweets per day as of Oct 17 2011
- eBay uses two data warehouses at 7.5 petabytes and 40PB as well as a 40PB Hadoop cluster for search, consumer recommendations, and merchandising. (May 10 2013)
- Walmart handles more than 1 million customer transactions every hour, which are imported into databases estimated to contain more than 2.5 petabytes (2560 terabytes) of data – the equivalent of 167 times the information contained in all the books in the US Library of Congress.

Note: the numbers are from online resources and can be out of dated and inaccurate

Linear Regression with Big Data

- Below is the gradient descent update rule (**one iteration**) that we used for the linear regression with one feature.

$$\beta_0 := \beta_0 - \alpha \frac{\partial L(\beta_0, \beta_1)}{\partial \beta_0} = \beta_0 - \alpha \frac{1}{N} \sum_{n=1}^{N} (\beta_0 + \beta_1 x_{n1} - t_n)$$

$$\beta_1 := \beta_1 - \alpha \frac{\partial L(\beta_0, \beta_1)}{\partial \beta_1} = \beta_1 - \alpha \frac{1}{N} \sum_{n=1}^{N} (\beta_0 + \beta_1 x_{n1} - t_n) x_{n1}$$

Update simultaneously

- This seems to be an easy task based on our previous examples
- Now suppose we have some census data of $N = 1{,}000{,}000{,}000$ examples (this is very possible and realistic)
- Then in order to calculate the above gradient descent update, we need to **sum up 1,000,000,000 elements**
- This is a very **expense** calculation
- Also called **batch gradient descent** ("batch" means we use all of the training examples at a time.)

The way the **batch gradient descent** algorithm works with big data:

- We need to stream all of these records through computer, because probably we cannot store all the 1 billion records in computer memory
- We need to read into computer memory all 1 billion records in order to compute the derivative term
- So we need to read through each record, and slowly accumulate the sum in order to compute the derivative
- Complete all the above work, we can take **one iteration** of **batch gradient descent**
- Suppose the batch gradient descent needs 10,000 iterations to converge, this whole process can take really really long time

Stochastic Gradient Descent

- Stochastic gradient descent (SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions
- With SGD, the gradient of the loss is estimated with **one sample** at a time and the model is updated along the way with the learning rate
- Stochastic gradient descent can be applied to linear regression, logistic regression, NN, etc. (Think about their loss functions.)
- We use regression to illustrate the idea of stochastic gradient descent.

Batch Gradient Descent

Batch gradient descent update rule (**one iteration**) for regression:

$$\beta_j := \beta_j - \alpha \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_j} = \beta_j - \alpha \frac{1}{N} \sum_{n=1}^{N} (f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n) x_{nj}$$

for $j = 0,1,2, \ldots, d$

where

$$f(\mathbf{x}_n, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \cdots + \beta_d x_{nd}$$

With large dataset the main pain point is to sum over all the N examples

- How to solve this pain point?
- Try to not do this summation ($\sum_{n=1}^{N}$) over all the N examples

Stochastic Gradient Descent

Regression loss function for linear model

$$L(\boldsymbol{\beta}) = \frac{1}{2N}\sum_{n=1}^{N}(f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)^2$$

Split this $L(\boldsymbol{\beta})$ into two parts:

$$\text{Loss}(\boldsymbol{\beta}, \mathbf{x}_n, t_n) = \frac{1}{2}(f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)^2$$

No summation for this loss term

$$L(\boldsymbol{\beta}) = \frac{1}{N}\sum_{n=1}^{N}\text{Loss}(\boldsymbol{\beta}, \mathbf{x}_n, t_n)$$

Stochastic Gradient Descent

$$\text{Loss}(\boldsymbol{\beta}, \mathbf{x}_n, t_n) = \frac{1}{2}(f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)^2$$

$$\frac{\partial \text{Loss}(\boldsymbol{\beta}, \mathbf{x}_n, t_n)}{\partial \beta_j} = (f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)x_{nj} \quad (\text{for } j = 0,1,2,\dots,d)$$

- Now the partial derivative has no summation term: we can calculate the "small" gradient with respect to **each training examples** $(\mathbf{x}_n, t_n)$
- Or in other words, we update the parameter based on these "small" gradient/learning, instead of calculating all 1 billion "small" gradients and then sum them

SGD- Algorithm

- Choose an initial vector of parameters and learning rate α.
- Repeat the following procedure until an approximate minimum is obtained:
  - **Randomize** the data set.
  - for $\mathrm{n} = 1,2,3,\ldots,N$, do

Note there is no summation here

$$\beta_j := \beta_j - \alpha \frac{\partial \mathrm{Loss}(\boldsymbol{\beta}, \mathbf{x}_n, t_n)}{\partial \beta_j} = \beta_j - \alpha(f(\mathbf{x}_n, \boldsymbol{\beta}) - t_n)x_{nj}$$

$$(\text{for } j = 0,1,2,\ldots,d)$$

We used regression as an example for the above SGD algorithm, while other machine learning algorithms like neural network can employ SGD as well.

SGD Properties

- The **randomization** (shuffling) of the data is done to avoid a bias in the optimization algorithm by providing the data examples in a particular order.
- Stochastic gradient descent is a popular algorithm for training a wide range of models in machine learning, including support vector machines, logistic regression and graphical models. When combined with the backpropagation algorithm, it is the de facto standard algorithm for training neural networks.
- The convergence of stochastic gradient descent has been analysed using the theories of convex minimization and of stochastic approximation.
- Stochastics gradient descent can find good **approximated estimates** compared to the ones from batch gradient descent.
- Also there are more sophisticated gradient descent variant algorithms such as Adam, Adagrad, and RMSprop etc

https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Example 1

Input data size: $N = 10000$ examples, $\mathrm{d} = 1000$ features

```
# Estiamte coefficients
t0 = time.time()
lr_obj_big.fit(x_big, y_big)
t1 = time.time()
total = t1-t0
print("BGD Running Time {0}".format(total))

clf = linear_model.SGDRegressor()

t0 = time.time()
clf.fit(x_big, y_big)
t1 = time.time()
total = t1-t0
print("SGD Running Time {0}".format(total))
```

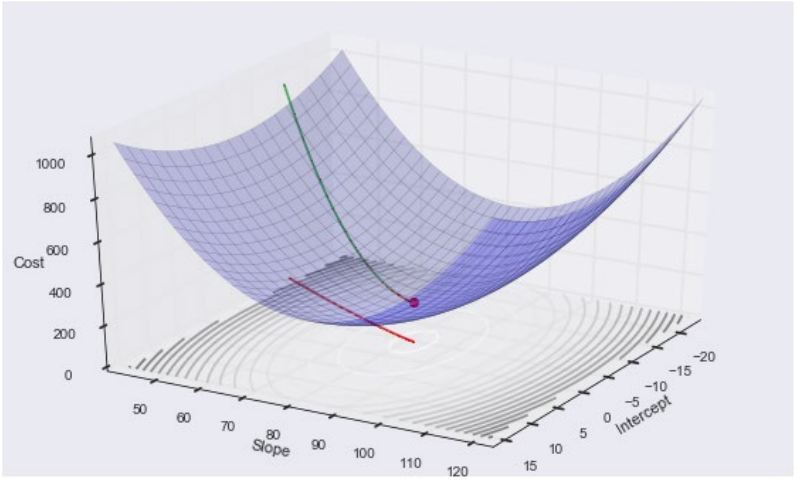
BGD Running Time 1.125 seconds
SGD Running Time 0.18799996376 seconds

- Generally, the stochastic gradient descent method will get close to the optimal parameter **much faster than** the batch gradient descent method.

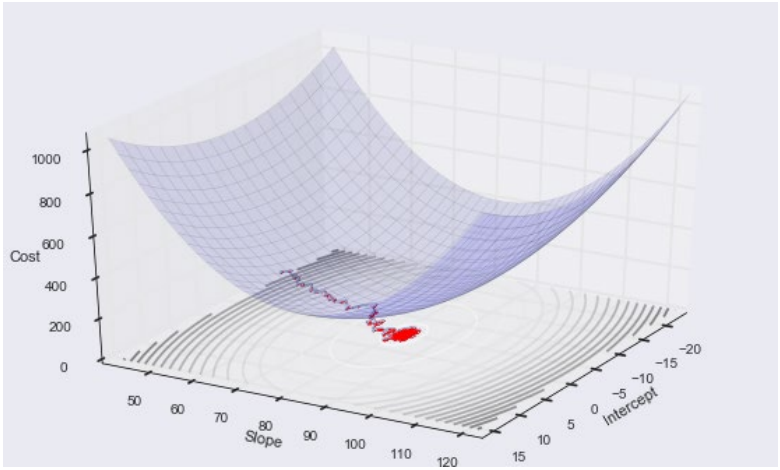
Example 2

Parameter Update Path with SGD & BGD

Batch gradient descent



Stochastic gradient descent

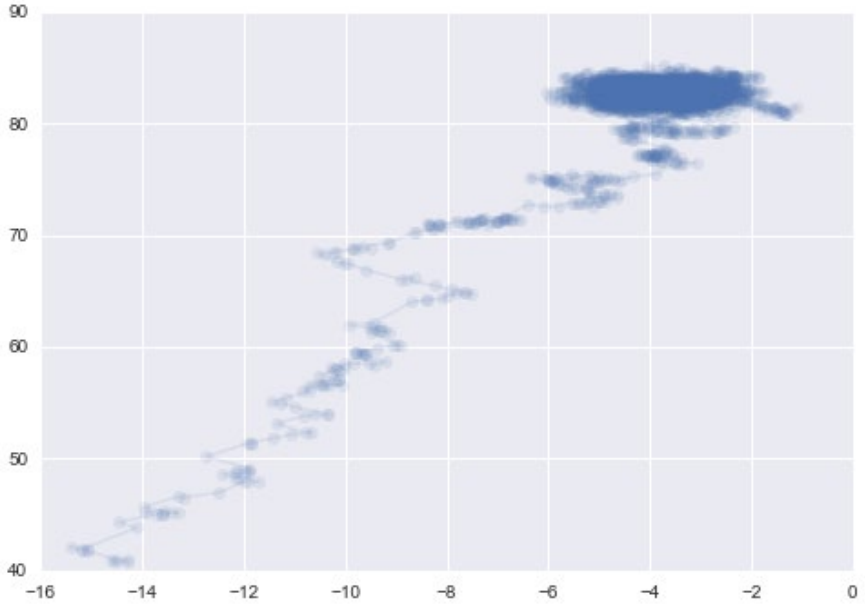


- SGD in general has lower accuracy compared to BGD

https://am207.github.io/2017/wiki/gradientdescent.html#stochastic-gradient-descent

Example 2

Parameter Update Path with SGD



Mini-batch Gradient Descent

Mini-batch Gradient Descent

- Mini-batch gradient descent is a trade-off between stochastic gradient descent and batch gradient descent.
- In mini-batch gradient descent, the gradient is summed over a small number of samples. This is between the SGD batch size of 1 sample, and the BGD batch size of all the training samples.
- Mini-batch gradient descent computation can make use of vectorization libraries rather than computing each step separately.
- It may also result in smoother convergence, as the gradient computed at each step uses more training examples than SGD.

Mini-batch Gradient Descent

- To summarize the gradient descent algorithms:
  - Batch gradient descent uses all N examples in each iteration.
  - Stochastic gradient descent uses 1 example in each iteration.
  - Mini-batch gradient descent uses b ($b < N$) example in each iteration.

Use $b = 5$ as example in mini-batch gradient descent

$$\left((\mathbf{x}_i, t_i), (\mathbf{x}_{i+1}, t_{i+1}), (\mathbf{x}_{i+2}, t_{i+2}), (\mathbf{x}_{i+3}, t_{i+3}), (\mathbf{x}_{i+4}, t_{i+4})\right)$$

$$\beta_j := \beta_j - \alpha \frac{\partial \mathrm{L}(\boldsymbol{\beta})}{\partial \beta_j} = \beta_j - \alpha \frac{1}{5} \sum_{n=i}^{i+4} (f(\mathbf{x}_{\boldsymbol{n}}, \boldsymbol{\beta}) - t_n) x_{nj}$$

Each mini-batch we have 5 examples=> make parameter update with summation over every 5 examples

Mini-batch GD- Algorithm

Suppose $N = 1000, b = 5$

- Choose an initial vector of parameters and learning rate α.
- Repeat the following procedure until an approximate minimum is obtained:
  - Randomize the data set.
  - for $\mathrm{i} = 1,6,11,16,21 \ldots, 996$ do

Summation over $b = 5$ examples

$$\beta_j := \beta_j - \alpha \frac{\partial \mathrm{L}(\boldsymbol{\beta})}{\partial \beta_j} = \beta_j - \alpha \frac{1}{5} \sum_{n=i}^{i+4} (f(\mathbf{x_n}, \boldsymbol{\beta}) - t_n) x_{nj}$$

$$(\text{for } j = 0,1,2, \ldots, d)$$

We used regression as an example for the above Mini-batch GD algorithm, while other machine learning algorithms like neural network can employ Mini-batch GD as well.

Property

- Mini-batch GD can perform significantly better than SGD, because the code can make use of vectorization libraries rather than computing each step separately, to realize parallel computing
- Mini-batch GD reduces the variance of the parameter updates, which can lead to **more stable and smoother convergence** compared with SGD, as the gradient computed at each step uses more training examples than SGD
- An potential issue: suppose we have 1050 training samples and we choose $b = 100$ in Mini-batch GD. Algorithm takes first 100 examples (from 1st to 100th) from the training dataset and update. Next it takes second 100 examples (from 101st to 200th) and update again. The problem usually happens with the last set of samples. In our example we've used 1050 which is not divisible by 100 without remainder. **The simplest solution is just to use the rest 50 examples and update.**

http://ruder.io/optimizing-gradient-descent/

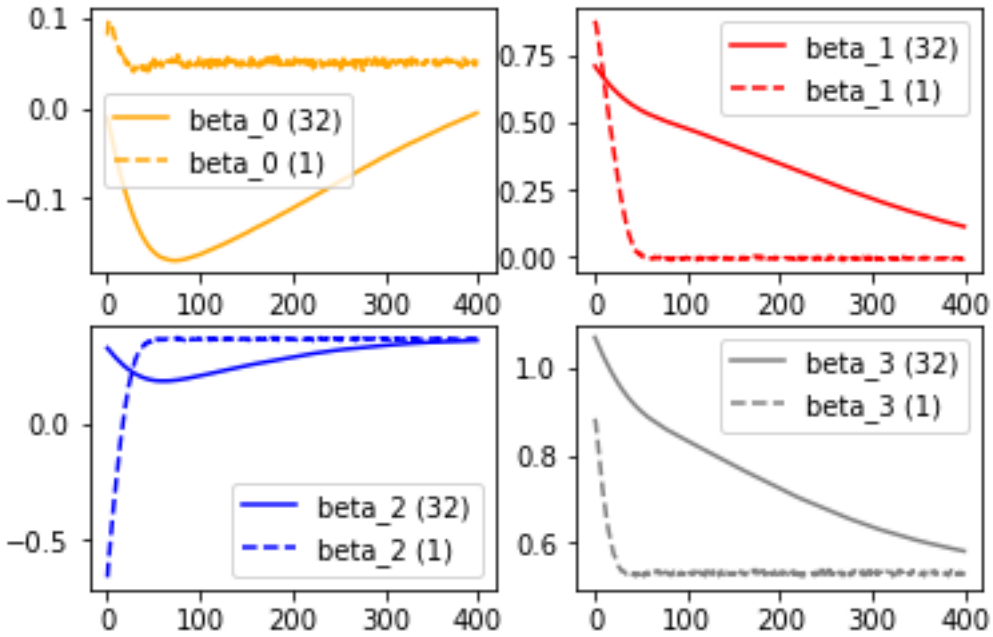
Configuration

- Mini-batch sizes b, commonly called “batch size” for brevity, are often tuned to an aspect of the computational architecture on which the implementation is being executed, such as **a power of 2** that fits the memory requirements of the GPU or CPU hardware like 32, 64, 128, 256, etc.
- Batch size b is a slider on the learning process.
  - Small values give a learning process that converges quickly at the cost of noise in the training process.
  - Large values give a learning process that converges slowly with accurate estimates of the error gradient.
- A good default for batch size b might be 32.
- It is a good idea to review learning curves of model validation loss against training time with different batch sizes when tuning the batch size.
- Tune batch size and learning rate after tuning all other hyperparameters.

https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Python Example

$N = 150$ training examples. 2 layer NN for regression.
"(32)" represents mini-batch GD method with $b = 32$, and "(1)" represents SGD method.



Mini-batch GD results in smoother convergence, while SGD gets close to the optimal parameter much faster.

BGD
SGD
Mini-batch GD
Comparison

- This stochastic gradient approach allows us to start making progress on the minimization problem right away. It is computationally cheaper, but it results in a larger variance of the parameter update in comparison with batch gradient descent.

- Generally, the stochastic gradient descent method will get close to the optimal parameter much faster than the batch method. Thus the stochastic gradient descent method is useful when we want a quick approximation for the solution to our optimization problem.

- Mini-batch GD has the advantage that the variance of parameter update is reduced compared with SGD, while the computational burden is still reasonable compared with BGD as we do not use all examples for each update.