Gradient Descent Tips

Tip Two: Stochastic and Mini-Batch Gradient Descent

Make the Training Faster

Stochastic Gradient Descent

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2$$

Loss is the summation over all training examples.

- ***Gradient Descent*** $\theta^i = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right)$

- ***Stochastic Gradient Descent***

Faster!

Pick an example x^n

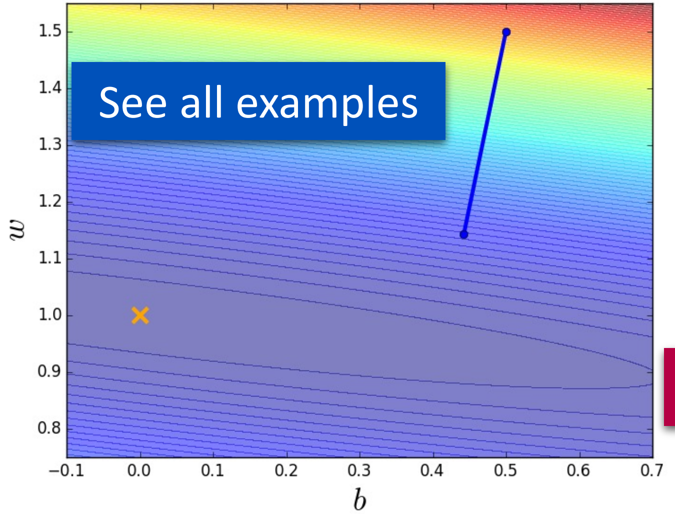
$$L^n = \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2 \qquad \theta^i = \theta^{i-1} - \eta \nabla L^n\left(\theta^{i-1}\right)$$

Loss for only one example

Stochastic Gradient Descent, Continued

Gradient Descent

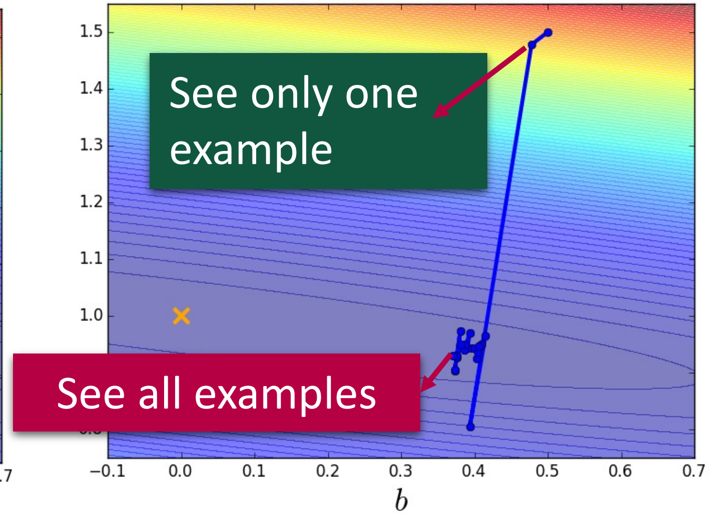
Update after seeing all examples



Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.



Mini-Batch Gradient Descent

Optimizing W, b

Batch

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{N} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Online/Stochastic

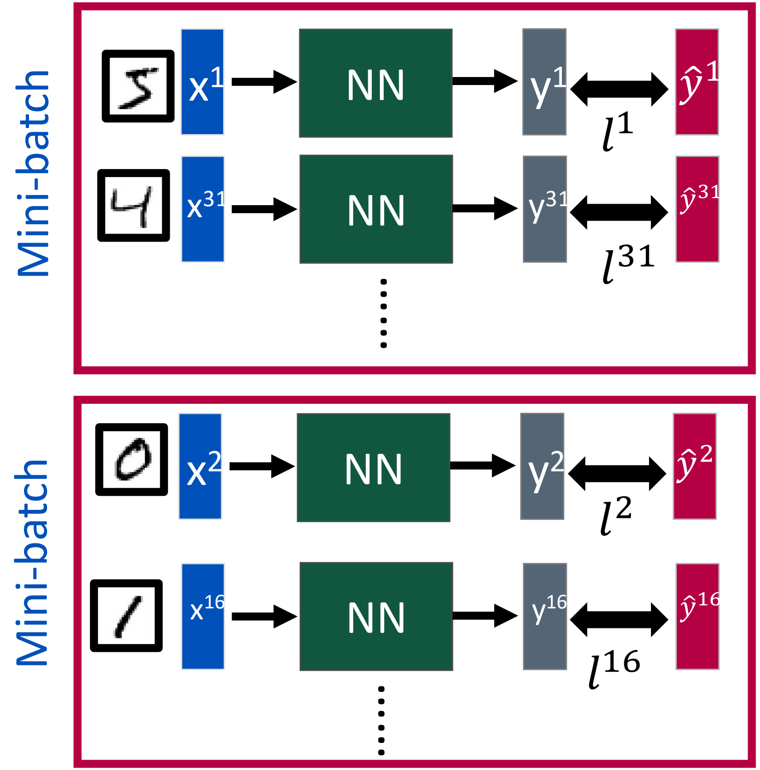
$$W_i \leftarrow W_i - \eta \frac{\partial l(x_j, y_j)}{\partial W_i}$$

Minibatch

$$W_i \leftarrow W_i - \eta \sum_{j=k}^{k+m} \frac{\partial l(x_j, y_j)}{\partial W_i}$$

We do not really minimize total loss!

Mini-Batch



- Randomly initialize network parameters

- Pick the 1[st] batch

  $L' = l^1 + l^{31} + \cdots$

  Update parameters once

- Pick the 2[nd] batch

  $L'' = l^2 + l^{16} + \cdots$

  Update parameters once

  ⋮

- Until all mini-batches have been picked

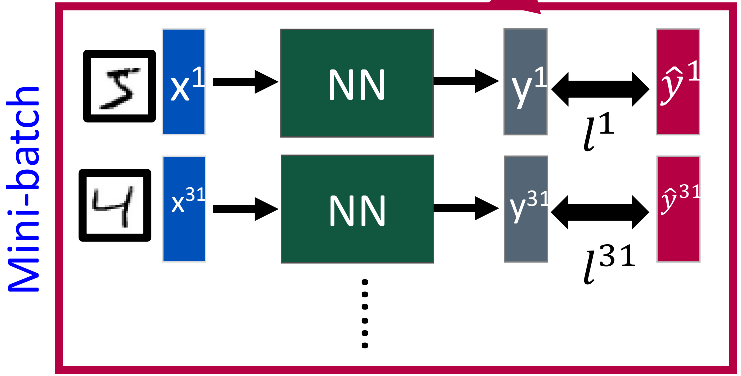
one epoch

Repeat the above process

Mini-Batch, Continued

Batch size influences both ***speed*** and ***performance***. You have to tune it.

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



100 examples in a mini-batch

Batch size = 1 ➡ Stochastic gradient descent

- Pick the 1st batch

  $L' = l^1 + l^{31} + \cdots$

  Update parameters once

- Pick the 2nd batch

  $L'' = l^2 + l^{16} + \cdots$

  Update parameters once

  ⋮

- Until all mini-batches have been picked

Repeat 20 times — one epoch

Speed

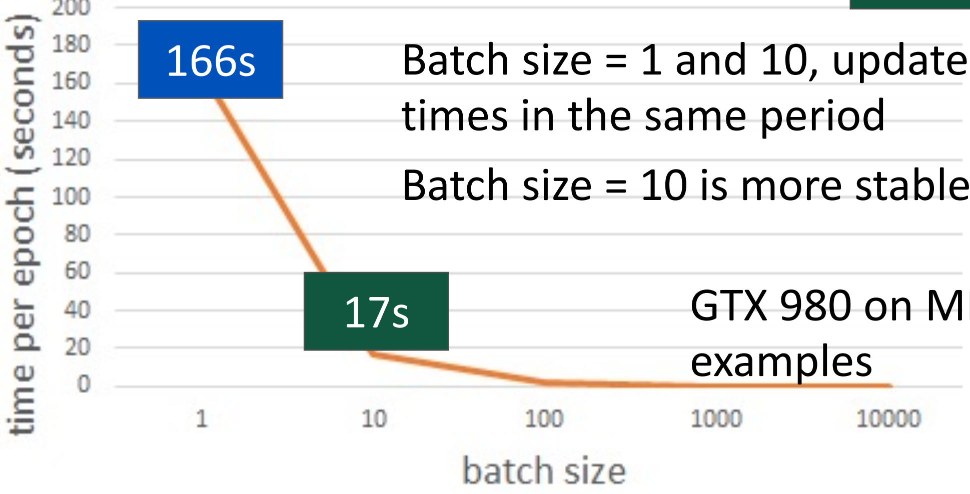
Very large batch size can yield worse performance.

Smaller batch size means more updates in one epoch

- E.g. 50000 examples
- batch size = 1, 50000 updates in one epoch
- batch size = 10, 5000 updates in one epoch

166s 1 epoch

17s 10 epoch



Batch size = 1 and 10, update the same amount of times in the same period

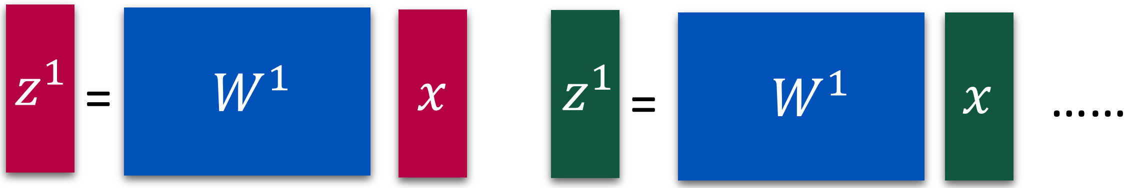
Batch size = 10 is more stable, converge faster

GTX 980 on MNIST with 50000 training examples

Speed - Matrix Operation

Why mini-batch is faster than stochastic gradient descent?

Stochastic Gradient Descent

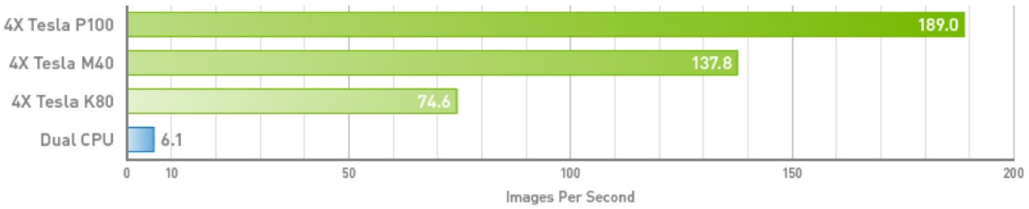


Mini-batch

Practically, which one is faster?

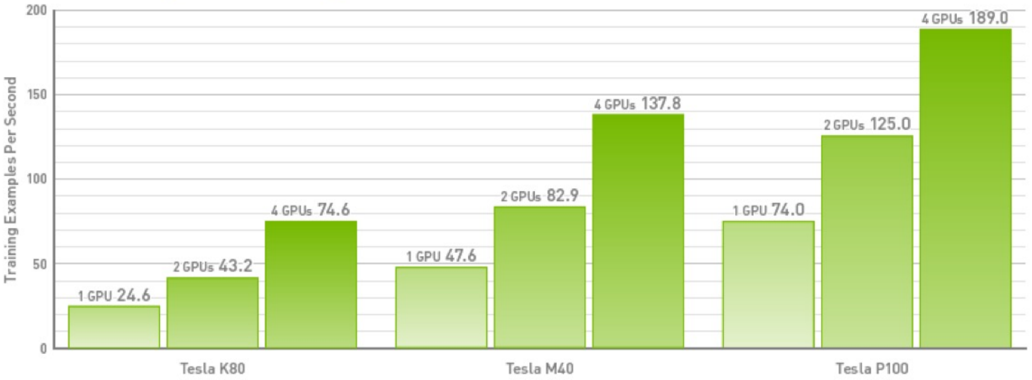
GPU Support

TensorFlow Image Classification Training Performance



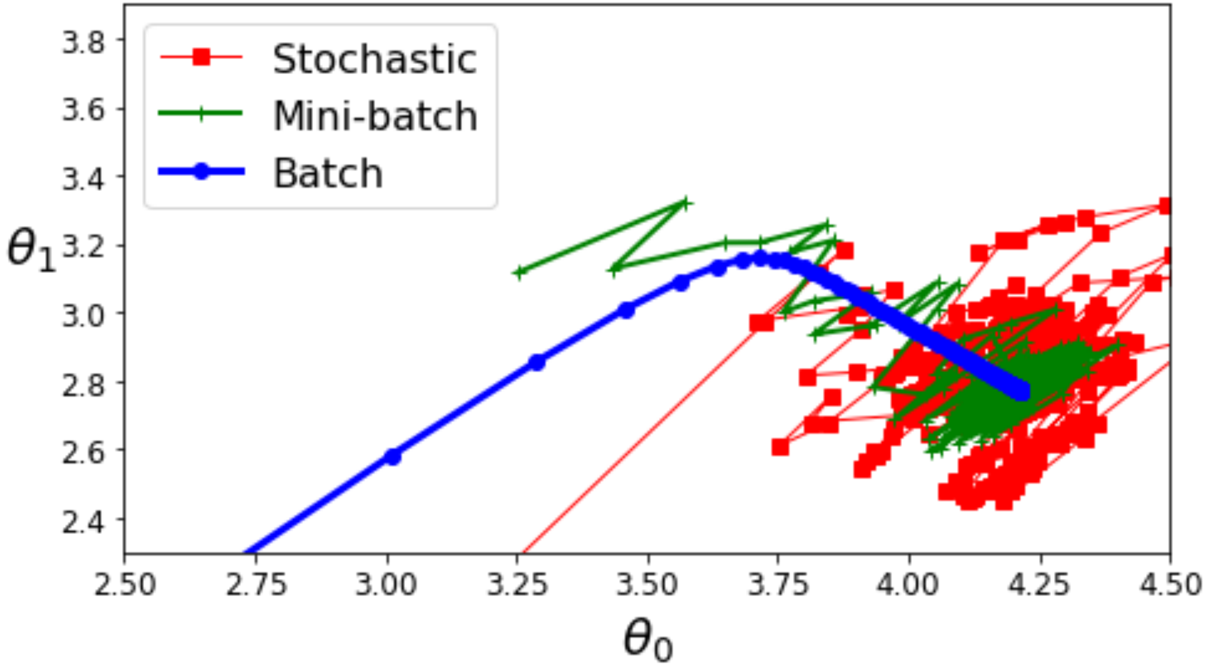
Dual CPU System: Dual Intel E5-2699 v4 @ 3.6 GHz | GPU-Accelerated System: Single Intel E5-2699 v4 @ 3.6 GHz, NVIDIA® Tesla® K80/M40/P100 (PCIe) | Google's Inception v3 image classification network, 500 steps; 64 Batch Size; cuDNN v5.1

TensorFlow Inception v3 Training Scalable Performance on Multi-GPU Node



GPU-Accelerated System: Single Intel E5-2699 v4 @ 3.6 GHz, NVIDIA® Tesla® K80/M40/P100 (PCIe) | Google's Inception v3 image classification network, 500 steps; 64 Batch Size; cuDNN v5.1

Comparison



Stochastic
Mini-batch
Batch
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
θ_1
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50
θ_0