S 9 3 7 0

The Steady State:

Reduce Spikiness from GPU Utilization with Apache MXNetNet (incubating)

Cyrus M Vahid
Principal Evangelist – AWS Deep Engine

cyrusmv@amazon.de

Our mission at AWS

Put machine learning in the hands of every developer

What you’ll learn about today

- Using timple tricks to maximize GPU utilization
    - Environment
    - I/O optimization
    - Imerative → Symbolic
    - Batch Size
    - Mixed Precision
- Optimization for large batch size

Environment

Environment

- GPU – Cuda optimized (mxnet-cuxx)
- CPU – Intel optimized (mxnet-mkl)
- CPU&GPU: Cuda and Intel optimized (mxnet-cuxxmkl)
- GPU: TensorRT optimzied (mxnet-tensorrt-cuxx)

Intel CPU - Speed

- Performance on Intel CPU with Intel MKL-DNN backend in release 1.3
- The c5.18xlarge instance offers a 2-socket Intel Xeon Platinum processor with 72 vCPUs.

Category	Model	Latency batchsize=1 (ms, small is better)			Throughput batchsize=128 (fps, big is better)		
		w/o MKL-DNN	w/ MKL-DNN	speedup	w/o MKL-DNN	w/ MKL-DNN	speedup
CNN/classification	ResNet-50 v1	97.19	13.04	**7.45**	10.29	163.52	**15.90**
	ResNet-50 v2	98.69	13.02	**7.58**	9.94	154.17	**15.51**
	Inception v3	175.17	16.77	**10.44**	5.74	135.33	**23.57**
	Inception v4	330.93	31.40	**10.54**	3.04	69.60	**22.87**
	DenseNet	111.66	18.90	**5.91**	8.52	149.88	**17.60**
	MobileNet	38.56	4.42	**8.73**	24.87	512.25	**20.60**
	VGG16	406.50	20.07	**20.25**	2.91	70.84	**24.31**
	AlexNet	64.60	3.80	**17.00**	26.58	965.20	**36.32**
	inception-resnet v2	181.10	49.40	**3.67**	5.48	82.97	**15.14**
CNN/object detection	Faster R-CNN	1175.74	118.62	**9.91**	0.85	8.57	**10.08**
	SSD-VGG16	721.03	47.62	**15.14**	1.43（batchsize=224)	28.90(batchsize=224)	**19.13**
	SSD-MobileNet	239.40	28.33	**8.45**	4.07(batchsize=256)	69.97(batchsize=256)	**14.18**
RNN	GNMT	683.43	94.00	**7.27**	1.46(batchsize=64)	10.63(batchsize=64)	**6.83**
GAN	DCGAN	8.94	0.24	**37.85**	109.13	4249.36	**38.94**

Intel CPU - Accuracy

- The c5.18xlarge instance offers a 2-socket Intel Xeon Platinum processor with 72 vCPUs.
- The model is from gluon model zoo by pre-trained parameters. The top1 and top5 accuracy are verified by MKL-DNN backend.

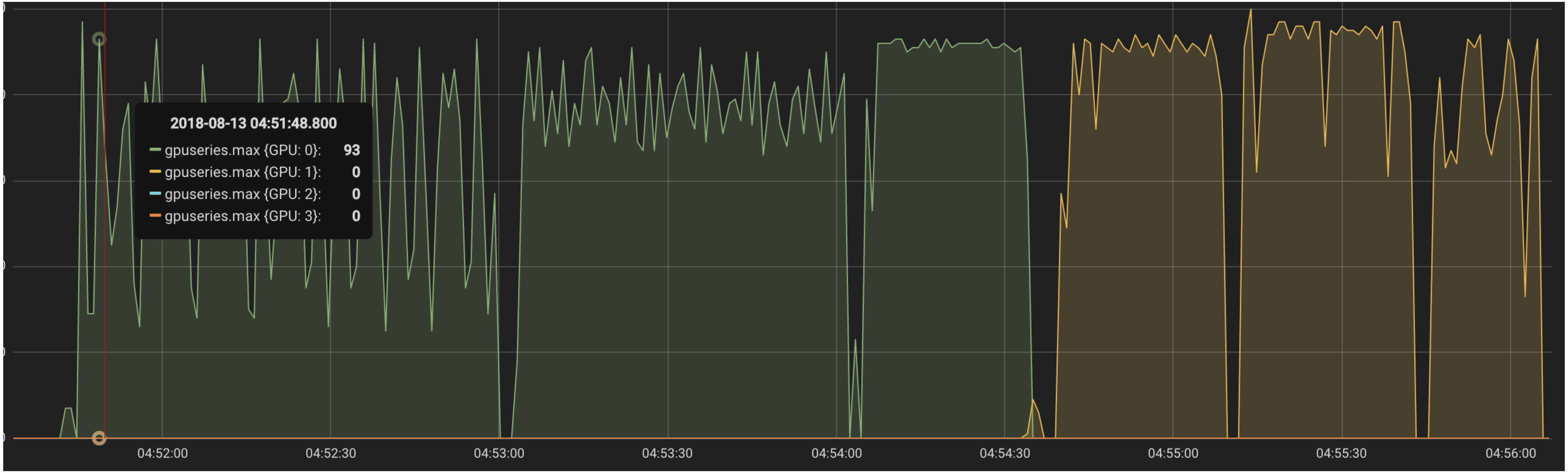
Inference Accuracy Comparison							
Alias	Network	CPU (without MKL-DNN)		CPU (with MKL-DNN) Backend		Delta	
		top1	top5	top1	top5	top1	top5
alexnet	AlexNet	0.56312500	0.78992188	0.56312500	0.78992188	0.00000000	0.00000000
densenet121	DenseNet-121	0.74203125	0.91929688	0.74203125	0.91929688	0.00000000	0.00000000
densenet161	DenseNet-161	0.77195313	0.93390625	0.77195313	0.93390625	0.00000000	0.00000000
densenet169	DenseNet-169	0.75710938	0.92828125	0.75710938	0.92828125	0.00000000	0.00000000
densenet201	DenseNet-201	0.76906250	0.93093750	0.76906250	0.93093750	0.00000000	0.00000000
inceptionv3	Inception V3 299x299	0.77609375	0.93664063	0.77609375	0.93664063	0.00000000	0.00000000
mobilenet0.25	MobileNet 0.25	0.51039063	0.75687500	0.51039063	0.75687500	0.00000000	0.00000000
mobilenet0.5	MobileNet 0.5	0.61851563	0.83789063	0.61851563	0.83789063	0.00000000	0.00000000
mobilenet0.75	MobileNet 0.75	0.66546875	0.87070313	0.66546875	0.87070313	0.00000000	0.00000000
mobilenet1.0	MobileNet 1.0	0.70093750	0.89109375	0.70093750	0.89109375	0.00000000	0.00000000
mobilenetv2_1.0	MobileNetV2 1.0	0.69976563	0.89281250	0.69976563	0.89281250	0.00000000	0.00000000
mobilenetv2_0.75	MobileNetV2 0.75	0.68210938	0.88007813	0.68210938	0.88007813	0.00000000	0.00000000
mobilenetv2_0.5	MobileNetV2 0.5	0.64453125	0.84929688	0.64453125	0.84929688	0.00000000	0.00000000
mobilenetv2_0.25	MobileNetV2 0.25	0.50890625	0.74546875	0.50890625	0.74546875	0.00000000	0.00000000
resnet18_v1	ResNet-18 V1	0.70812500	0.89453125	0.70812500	0.89453125	0.00000000	0.00000000
resnet34_v1	ResNet-34 V1	0.73960938	0.91609375	0.73960938	0.91609375	0.00000000	0.00000000
resnet50_v1	ResNet-50 V1	0.76062500	0.93046875	0.76062500	0.93046875	0.00000000	0.00000000

TensorRT

- NVIDIA TensorRT™ is a platform for high-performance deep learning inference.
- It includes a deep learning inference optimizer and runtime that delivers low latency and high-throughput for deep learning inference applications.
- TensorRT-based applications perform up to 40x faster than CPU-only platforms during inference.

Model Name	Relative TensorRT Speedup	Hardware
Alexnet	1.4x	Titan V
cifar_resnet20_v2	1.21x	Titan V
cifar_resnext29_16x64d	1.26x	Titan V
Resnet 18	1.8x	Titan V
Resnet 18	1.54x	Jetson TX1
Resnet 50	1.76x	Titan V
Resnet 101	1.99x	Titan V

GPU Utilization



I/O GPU Starvation

Naive

```
Sat Mar 16 20:44:51 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   53C    P0   123W / 300W |  11994MiB / 16130MiB |     35%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   45C    P0    41W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   48C    P0    45W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   47C    P0    42W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 1, NUM_WORKER: 1,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 306.63**
- **epoch time: 40.97**

```
Sat Mar 16 22:11:26 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   50C    P0    54W / 300W |  11604MiB / 16130MiB |     14%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   47C    P0    58W / 300W |   1494MiB / 16130MiB |     10%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   50C    P0    62W / 300W |   1484MiB / 16130MiB |     13%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   49C    P0    57W / 300W |   1504MiB / 16130MiB |     12%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 4, NUM_WORKER: 1,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 392.00**
- **epoch time: 40.22**

Data Loading

- It is almost always the case that I/O lags behind computer while using GPUS.
- There are several techniques to address improve IO.

Multi-Worker DataLoader

- CPU is used to load minibatches.
- Then the minibatches are passed to the GPU to process.
- After processing a minibatch, GPU will have to wait for the next minibatch load to be completed.
- By default Gluon dataloader uses 3 cores. We can change this value to reduce spikiness.
- The recommended value is cpu_count() - 3

Multiple Workers

```
Sat Mar 16 22:23:08 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   55C    P0   169W / 300W |  13060MiB / 16130MiB |     65%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   47C    P0    41W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   50C    P0    45W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   48C    P0    43W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 1, NUM_WORKER: 29,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 663.50**
- **epoch time: 18.59**

```
Sat Mar 16 22:18:04 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   52C    P0    59W / 300W |  11566MiB / 16130MiB |     27%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   51C    P0    94W / 300W |   1418MiB / 16130MiB |     27%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   54C    P0    93W / 300W |   1446MiB / 16130MiB |     30%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   52C    P0    85W / 300W |   1428MiB / 16130MiB |     27%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 4, NUM_WORKER: 29,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 950.43**
- **epoch time: 16.50**

Code

```
Import multiprocessing as mp

import multiprocessing as mp

NUM_WORKERS = mp.cpu_count() - 3


train_data_iter = gluon.data.DataLoader(dataset_train,

                                        shuffle=True,

                                        batch_size=batch_size,

                                        num_workers=NUM_WORKERS)
```

Off-line Pre-Processing

- For large data files, preprocessing can be done before training.
- It is also possible to incldue pre-processing in custom Dataset.
- For large data files, such as images, it is crucial to use a binary format such as RecordIO.

Efficient DataLoaders - Challenge

- Tiny datasets can be loaded entirely GPU memory.
- For large datasets we can only hold examples of data in memory.
- Data loading can become a major bottleneck.

Efficient DataLoaders - Solution

- Small file size.
- Parallel (distributed) packing of data.
- Fast data loading and online augmentation.
- Quick reads from arbitrary parts of the dataset in the distributed setting.

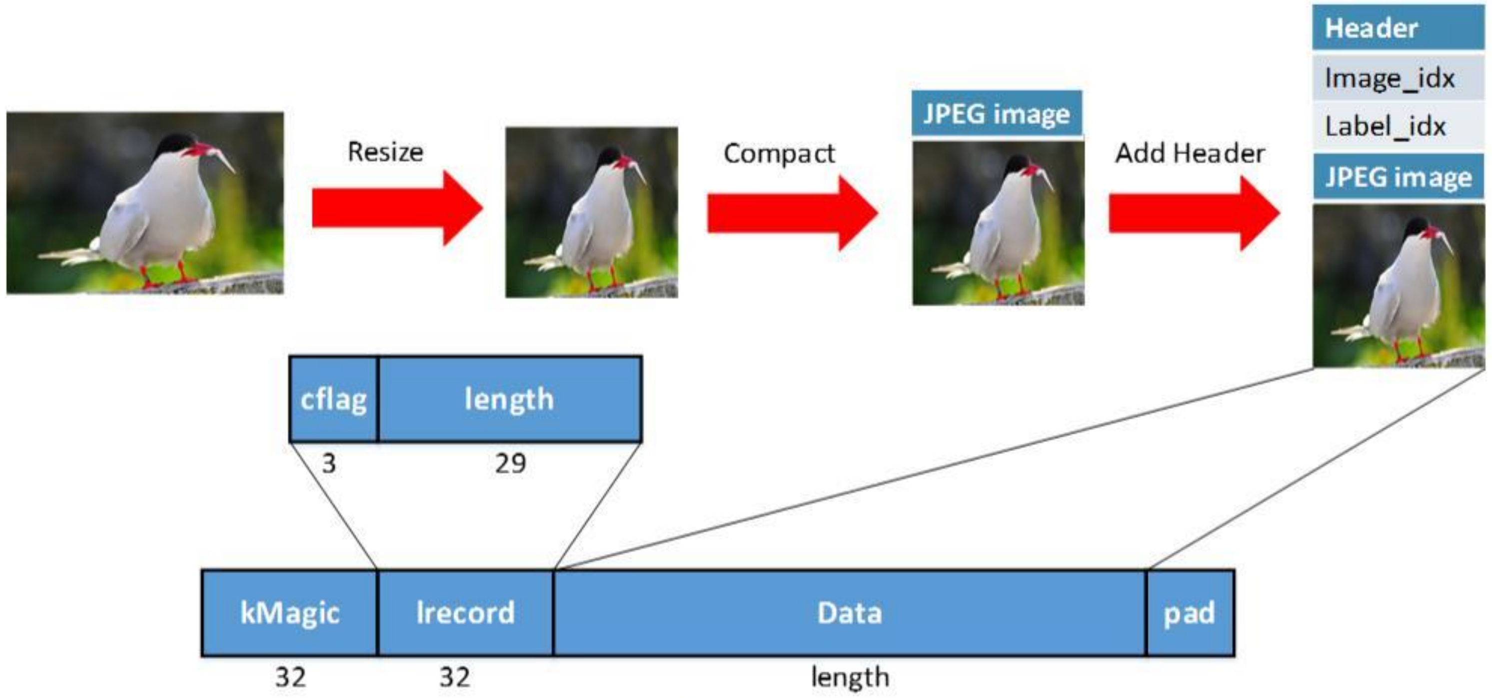


Image binary recordIO

Imperative → Symbolic

Symbolic vs. Imperative

- Imperative-style programs perform computation as you run them.
- In Symbolic programming, we define an abstract function in terms of placeholder values; then we compile the function after lazy data binding.
- Imperative programming is easy and debuggable, while symbolic is efficient.
- Symbolic computational graphs are most effetive for small networks and small batch size.

Imperative vs. Symbolic

<table>
<tr><th>Imperative</th><th>Symbolic</th></tr>
<tr><td>

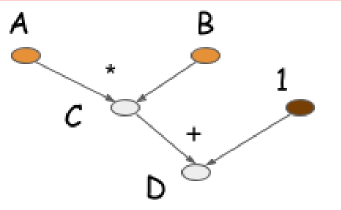
Execution Flow is the same as flow of the code:

```
import numpy as np
a = 2
b = a + 1
print d
for i in range(len(d)):
    d += np.ones(10)
```

</td><td>

Abstract functions are defined and compiled first, data binding happens next.

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
# compiles the function
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```



</td></tr>
<tr><td>

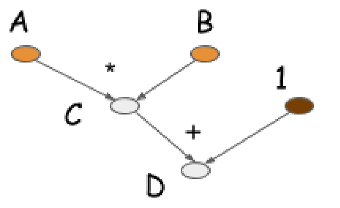
Flexible but inefficient:

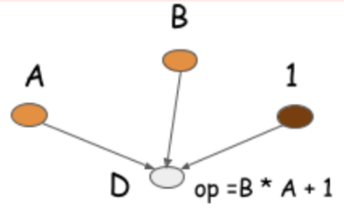
```
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

- Memory: 4 * 10 * 8 = 320 bytes
- Interim values are available
- No Operation Folding.
- Familiar coding paradigm.

</td><td>

Efficient





- Memory: 2 * 10 * 8 = 160 bytes
- Interim values are not available

- Operation Folding: Folding multiple operations into one. We run one op. instead of many on GPU. This is possible because we have access to whole comp. graph

</td></tr>
</table>

Imperative vs. Symbolic

Symbolic is “define, compile, run”

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

Imperative is “define-by-run”

```
net = nn.Sequential()
with net.name_scope():
    net.add(
        nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Flatten(),
        nn.Dense(120, activation="relu"),
        nn.Dense(84, activation="relu"),
        nn.Dense(10)
    )
net.initialize(init=init.Xavier())
```

```
for epoch in range(10):
    for data, label in train_data:
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()
        trainer.step(batch_size)
```

Imperative vs. Symbolic

Symbolic is “define, compile, run”

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

Imperative is “define-by-run”

```
net = nn.Sequential()
with net.name_scope():
    net.add(
        nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Flatten(),
        nn.Dense(120, activation="relu"),
        nn.Dense(84, activation="relu"),
        nn.Dense(10)
    )
net.initialize(init=init.Xavier())
```

```
for epoch in range(10):
    for data, label in train_data:
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()
        trainer.step(batch_size)
```

Imperative vs. Symbolic

Symbolic is “define, compile, run”

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
```

Imperative is “define-by-run”

```
net = nn.Sequential()
with net.name_scope():
    net.add(
        nn.Conv2D(channels=6, kernel_size=5, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Conv2D(channels=16, kernel_size=3, activation='relu'),
        nn.MaxPool2D(pool_size=2, strides=2),
        nn.Flatten(),
        nn.Dense(120, activation="relu"),
        nn.Dense(84, activation="relu"),
        nn.Dense(10)
    )
net.initialize(init=init.Xavier())
```

```
for epoch in range(10):
    for data, label in train_data:
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()
        trainer.step(batch_size)
```

In Gluon, you can code imperatively, and then switch mode to symbolic

Code

```
hnet = gluon.nn.HybridSequential()

with hnet.name_scope():

    hnet.add(gluon.nn.Dense(units=64, activation='relu'))

    hnet.add(gluon.nn.Dense(units=148, activation='relu'))

    hnet.add(gluon.nn.Dense(units=10))

hnet.hybridize()
```

```
snet = gluon.nn.Sequential()

with snet.name_scope():

    snet.add(gluon.nn.Dense(units=64, activation='relu'))

    snet.add(gluon.nn.Dense(units=148, activation='relu'))

    snet.add(gluon.nn.Dense(units=10))
```

Slide Type

```
snet.hybridize()
```

```
/home/ubuntu/anaconda3/envs/mxnet_p36/lib/python3.6/site-packages/ipykernel_launcher.py:1: UserWarning: All children
of this Sequential layer 'sequential3_' are HybridBlocks. Consider using HybridSequential for the best performance.
  """Entry point for launching an IPython kernel.
```

Symbolic

```
Sat Mar 16 22:28:13 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   50C    P0    59W / 300W |  16114MiB / 16130MiB |     21%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   46C    P0    41W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   49C    P0    45W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   47C    P0    43W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 1, NUM_WORKER: 29,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 941.20**
- **epoch time: 13.25**

```
Sat Mar 16 22:30:59 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   51C    P0    80W / 300W |  11468MiB / 16130MiB |     37%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   49C    P0   106W / 300W |   1318MiB / 16130MiB |     36%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   53C    P0   165W / 300W |   1346MiB / 16130MiB |     39%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   51C    P0   143W / 300W |   1330MiB / 16130MiB |     37%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 4, NUM_WORKER: 29,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 1266.16**
- **epoch time: 12.77**

Mixed Precision Training

Mixed Precision Training

- Instead of float32, we can use float16 for training a network. This reduces data size significantly and results in faster training time.

Code

```
optimizer = mx.optimizer.SGD(momentum=0.9, learning_rate=.001, multi_precision=True)
…

net.initialize(mx.init.Xavier(magnitude=2.3), ctx=ctx, force_reinit=True)

net.cast('float16')

…

for e in range(epoch):

          for i, (data, label) in enumerate(dataloader_train):

                    data = data.as_in_context(ctx).astype('float16')

                    label = label.as_in_context(ctx).astype('float16')

                    …
```

Mixed Precision

```
Sun Mar 17 00:23:52 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   61C    P0   142W / 300W |  14254MiB / 16130MiB |     66%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   54C    P0    44W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   59C    P0    48W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   54C    P0    45W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

```
Sun Mar 17 00:22:43 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   58C    P0   100W / 300W |  12012MiB / 16130MiB |     36%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   59C    P0    90W / 300W |   1908MiB / 16130MiB |     32%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   65C    P0    90W / 300W |   1896MiB / 16130MiB |     35%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   60C    P0   112W / 300W |   1906MiB / 16130MiB |     33%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 1, NUM_WORKER: 29,
- **BATCH_SIZE_PER_GPU: 64,**
- TYPECAST: <class 'numpy.float16'>
- **Samples/Sec: 3100**
- **epoch time: 12.00**
- **BATCH_SIZE=640 ➔ sam/sec:14000 & et:4.2**

- NUM_GPU: 4, NUM_WORKER: 29,
- BATCH_SIZE_PER_GPU: 64.0,
- TYPECAST: <class 'numpy.float16'>
- **Samples/Sec: 7400**
- **epoch time: 7.5**

Batch Size

Large Batch Size

- Larger batch size improves training efficiency but can adversly affect accuracy.
- It recommended multiply batch size by the number of gpus you are training your model on.
- For large batch sized (in ranges of hundreds and thousands), using learning rate scheduler becmoes important.
- For batch sized in ranges of thousdands optimization algorithms such as LBSGD can be used to stablize training.

Large Batch Size

Batch Size/GPU	
64	INFO:root:[Epoch 0] train=0.433659 val=0.518400 loss=1.532322 time: 28.524742 INFO:root:[Epoch 1] train=0.647567 val=0.656700 loss=0.997764 time: 27.795407 INFO:root:[Epoch 2] train=0.720331 val=0.713100 loss=0.800816 time: 27.461927
128	INFO:root:[Epoch 0] train=0.478706 val=0.562100 loss=1.439123 time: 14.813820 INFO:root:[Epoch 1] train=0.658754 val=0.697500 loss=0.959008 time: 14.829524 INFO:root:[Epoch 2] train=0.730148 val=0.713900 loss=0.770881 time: 14.250317
256	INFO:root:[Epoch 0] train=0.455829 val=0.566400 loss=1.487115 time: 8.577288 INFO:root:[Epoch 1] train=0.637139 val=0.606700 loss=1.012649 time: 8.638407 INFO:root:[Epoch 2] train=0.712139 val=0.688200 loss=0.809953 time: 7.719393
512	INFO:root:[Epoch 0] train=0.406955 val=0.461200 loss=1.600047 time: 6.671616 INFO:root:[Epoch 1] train=0.609476 val=0.599100 loss=1.087454 time: 7.109749 INFO:root:[Epoch 2] train=0.695031 val=0.658800 loss=0.859839 time: 5.541541
1280	INFO:root:[Epoch 0] train=0.338381 val=0.356700 loss=1.797568 time: 6.654431 INFO:root:[Epoch 1] train=0.518189 val=0.482300 loss=1.313464 time: 5.898391 INFO:root:[Epoch 2] train=0.611759 val=0.608200 loss=1.077867 time: 6.807596

Batch Size

```
Sat Mar 16 22:55:29 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   50C    P0   207W / 300W |  12362MiB / 16130MiB |     44%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   45C    P0    41W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   49C    P0    45W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   47C    P0    42W / 300W |     11MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 1, NUM_WORKER: 29,
- **BATCH_SIZE_PER_GPU: 640,**
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 17000**
- **epoch time: 4.0**

```
Sat Mar 16 23:08:43 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1B.0 Off |                    0 |
| N/A   49C    P0    53W / 300W |  11930MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000000:00:1C.0 Off |                    0 |
| N/A   50C    P0    68W / 300W |   3618MiB / 16130MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000000:00:1D.0 Off |                    0 |
| N/A   54C    P0    73W / 300W |   3262MiB / 16130MiB |     22%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   52C    P0    69W / 300W |   3228MiB / 16130MiB |     17%      Default |
+-------------------------------+----------------------+----------------------+
```

- NUM_GPU: 4, NUM_WORKER: 29,
- **BATCH_SIZE_PER_GPU: 640,**
- TYPECAST: <class 'numpy.float32'>
- **Samples/Sec: 50000**
- **epoch time: 4.0**
- **NUM_WORKERS=1 ➔ Samples/Sec: 1500 & Epoch Time: 60+**

Choose Number of GPUs wisely

Number of Devices

```
-------------------------
 Volatile Uncorr. ECC
 GPU-Util  Compute M.
-========================
                      0
      28%       Default
-------------------------
                      0
       0%       Default
-------------------------
                      0
       0%       Default
-------------------------
                      0
       0%       Default
-------------------------
```

- **Samples/Sec: 3200**
- **epoch time: 16**

```
+------------------------
| Volatile Uncorr. ECC
| GPU-Util  Compute M.
+========================
|                     0
|      0%       Default
+------------------------
|                     0
|      1%       Default
+------------------------
|                     0
|      0%       Default
+------------------------
|                     0
|      0%       Default
-------------------------
```

- **Samples/Sec: 3400**
- **epoch time: 15**

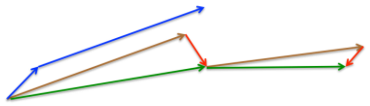
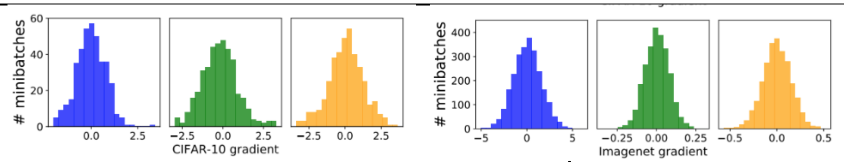
```
-------------------------
 Volatile Uncorr. ECC
 GPU-Util  Compute M.
=========================
                      0
       0%       Default
-------------------------
                      0
       0%       Default
-------------------------
                      0
       0%       Default
-------------------------
                      0
       4%       Default
-------------------------
```

- **Samples/Sec: 3700**
- **epoch time: 13.6**

Choosing the Right Optimizer

Optimizer

- Optimizers affect time to accuracy. Some major optimizers are:

Function	Description	
SGD	Stochastic Gradient Descent with momentum and weight decay	
NAG	Nesterov Accelerated Gradient	
DCASGD [3]	Delay Compensated Asynchronous Stochastic Gradient Descent	Useful for very large and very deep models
Signum[4]	Compressed Optimisation for Non-Convex Problems	 SGD vs. Signum vs. adam
FTML[5]	Follow the Moving Leader	provides improved stability over RMSProp and better performance over adam, especially in changing environments
LBSGD [6][7]	Large Batch SGD with momentum and weight decay	

Learning Rate Scheduler

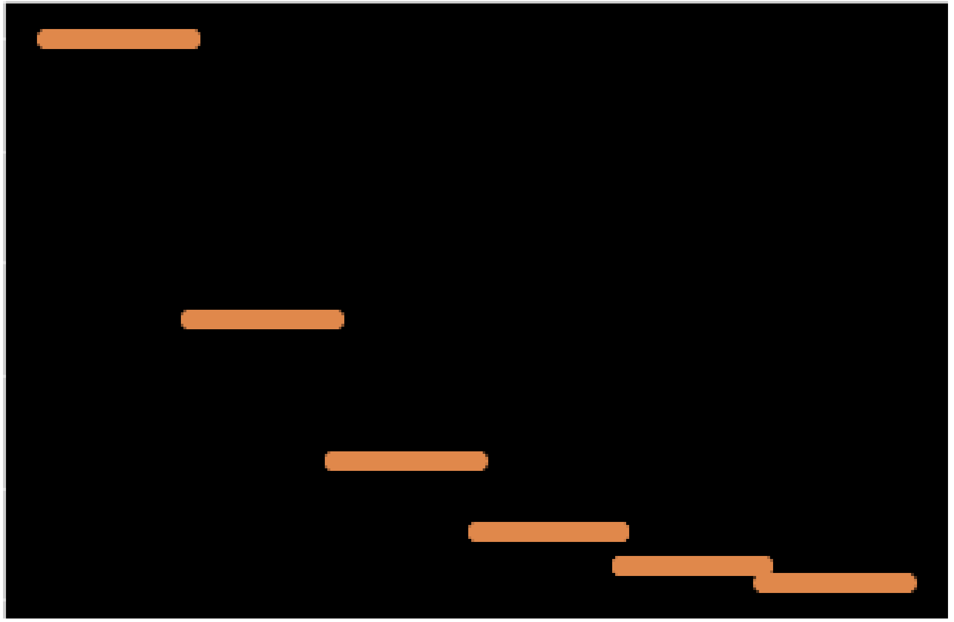
Learning Rate Scheduler

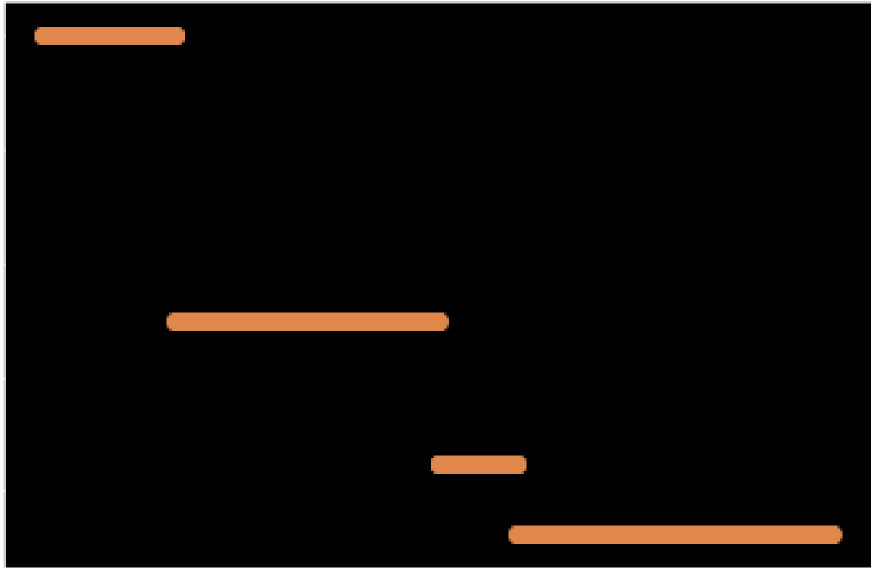
- Schedules define how the learning rate changes over time and are typically specified for each epoch or iteration (i.e. batch) of training. Schedules differ from adaptive methods (such as AdaDelta and Adam) because they:
  - change the global learning rate for the optimizer, rather than parameter-wise learning rates
  - don’t take feedback from the training process and are specified beforehand

Fixed Factor

```
schedule = mx.lr_scheduler.FactorScheduler(step=250, factor=0.5)
schedule.base_lr = 1
plot_schedule(schedule)
```

Miltifactor





```
schedule = mx.lr_scheduler.MultiFactorScheduler(step=[250, 750, 900],
factor=0.5) schedule.base_lr = 1
plot_schedule(schedule)
```

Factor

https://mxnet.incubator.apache.org/tutorials/gluon/learning_rate_schedules.html

Recap

Recap

- I/O is your main bottleneck. Using multiple workers can easily double your speed and reduce jittery GPU utilization
- Binary input format can improve performance by orders of magnitude.
- Hybridizing your network can significantly reduce training time. It is most effective for shallow networks with small batch-size (inference).

Recap

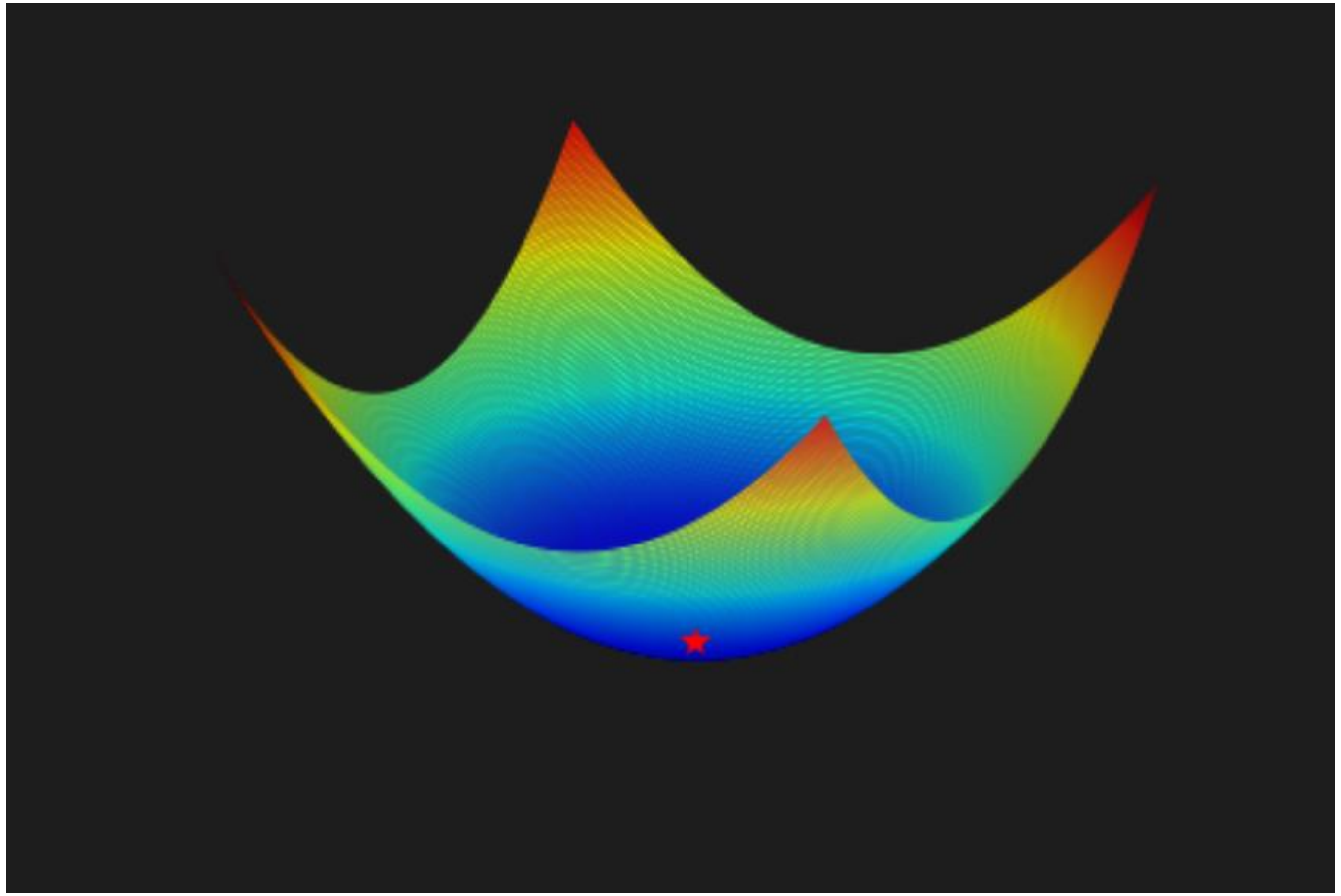
- Mixed precision training can halve the training time. Using `multi_precision` option for your optimizer helps increase accuracy.
- Large batch sizes can reduce training time in orders of magnitude but can reduce accuracy. Using correct optimizer and learning rate scheduling helps with regaining accuracy.
- Mode GPU is not always a good idea. Choose the right number of GPUs based on your data and your networks.

Thank You

More on Optimization

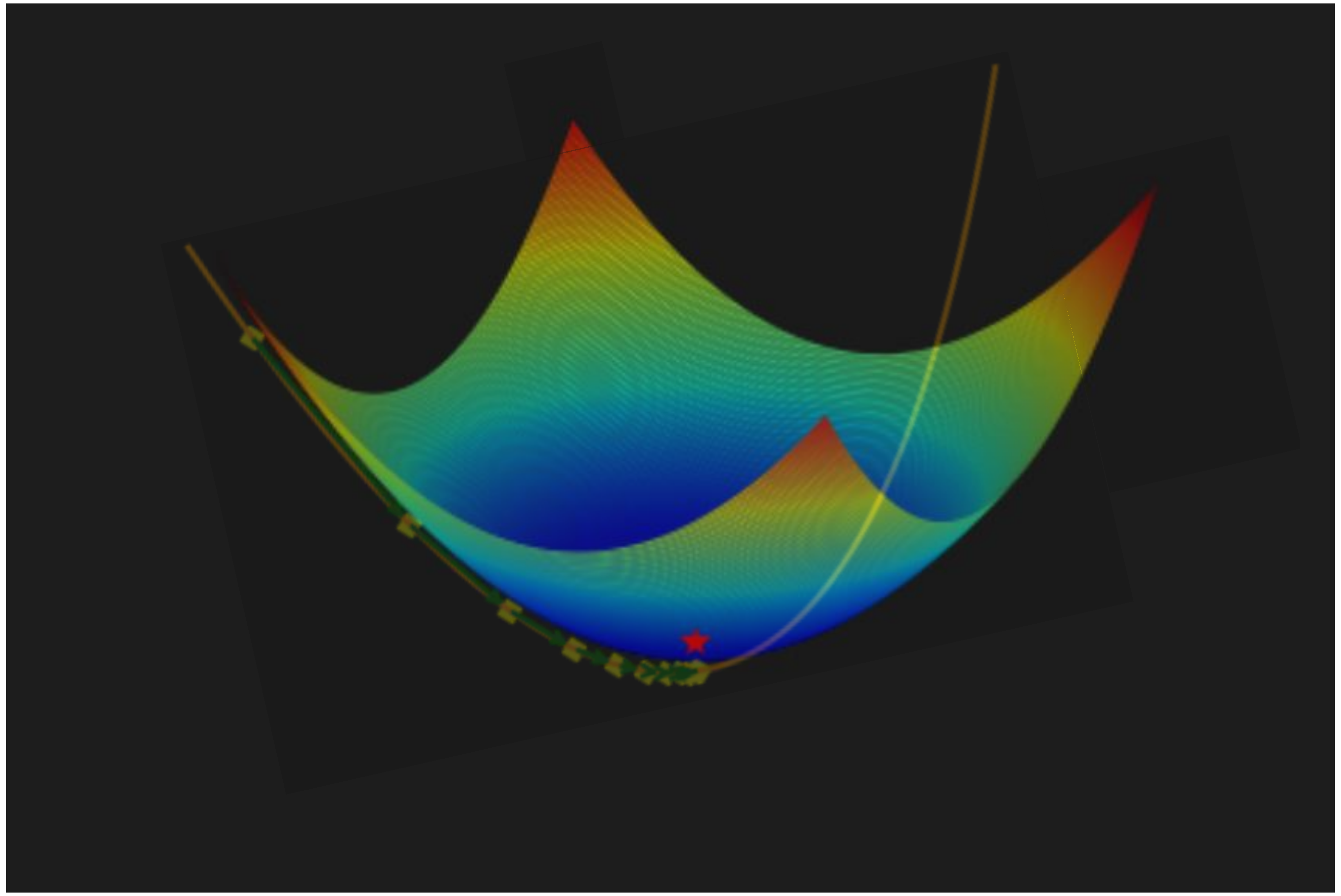
Gradient Descent

- After training over data we sill have an error surface.



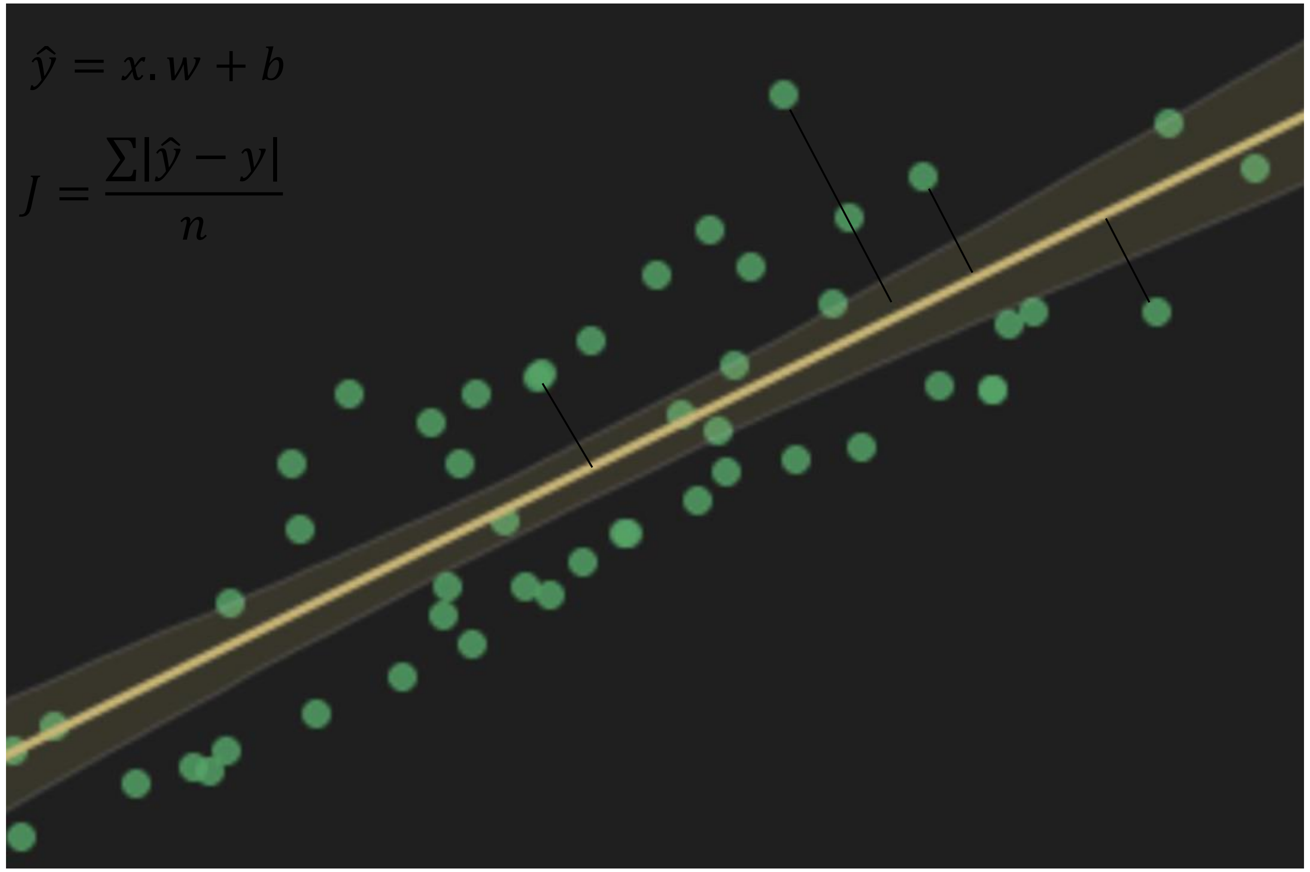
Gradient Descent

- After training over data we sill have an error surface.
- The goal of optimization is to reach the minima of the surface, and thus reducing error



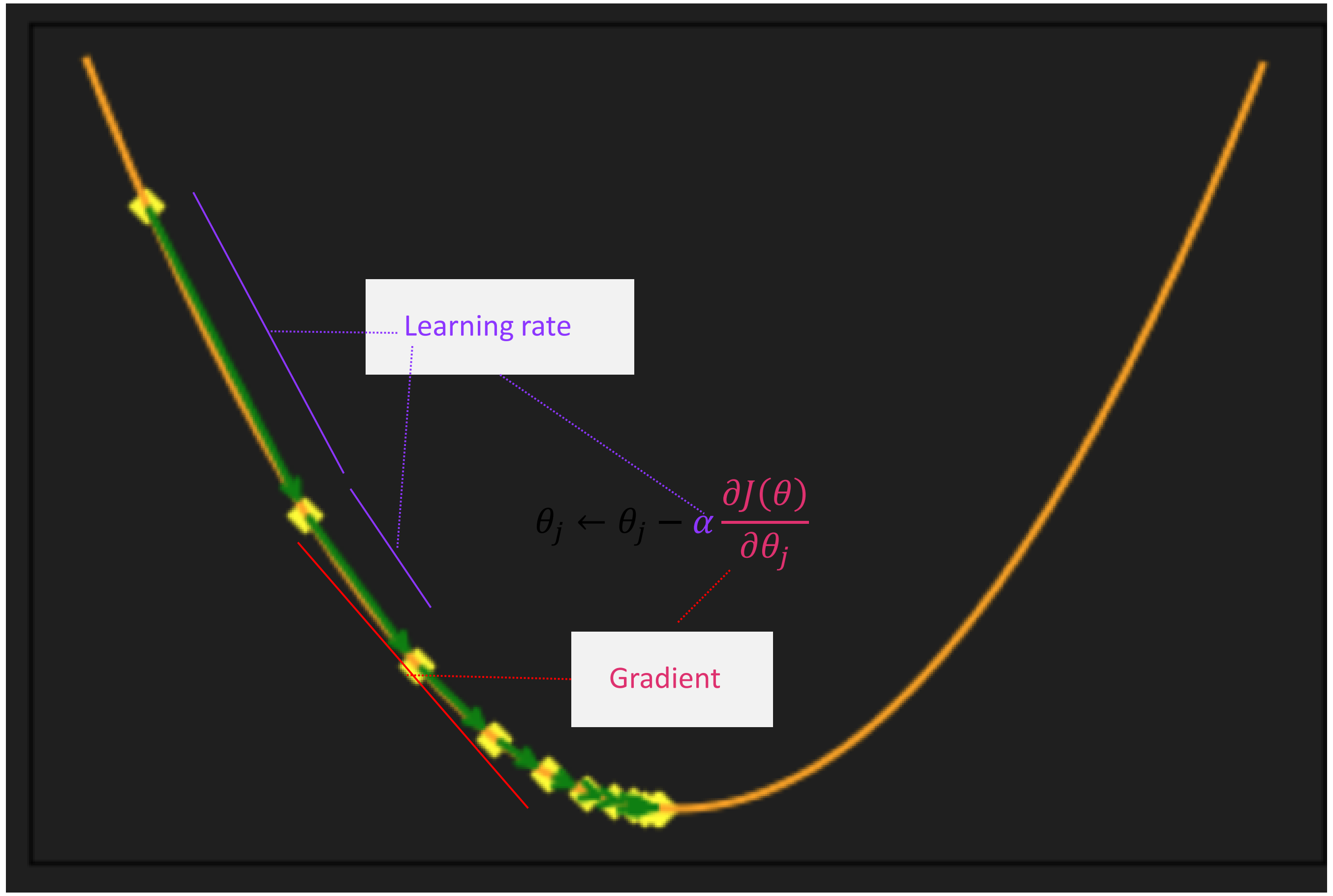
Loss Function

- Loss Function, J, is a measure of how well an algorithm models a dataset.
- There are several loss functions and one can combine them. Some of the more popular loss functions are RMST, Hinge, L1, L2, …
- For more information please check: https://tinyurl.com/y7c6ub5k



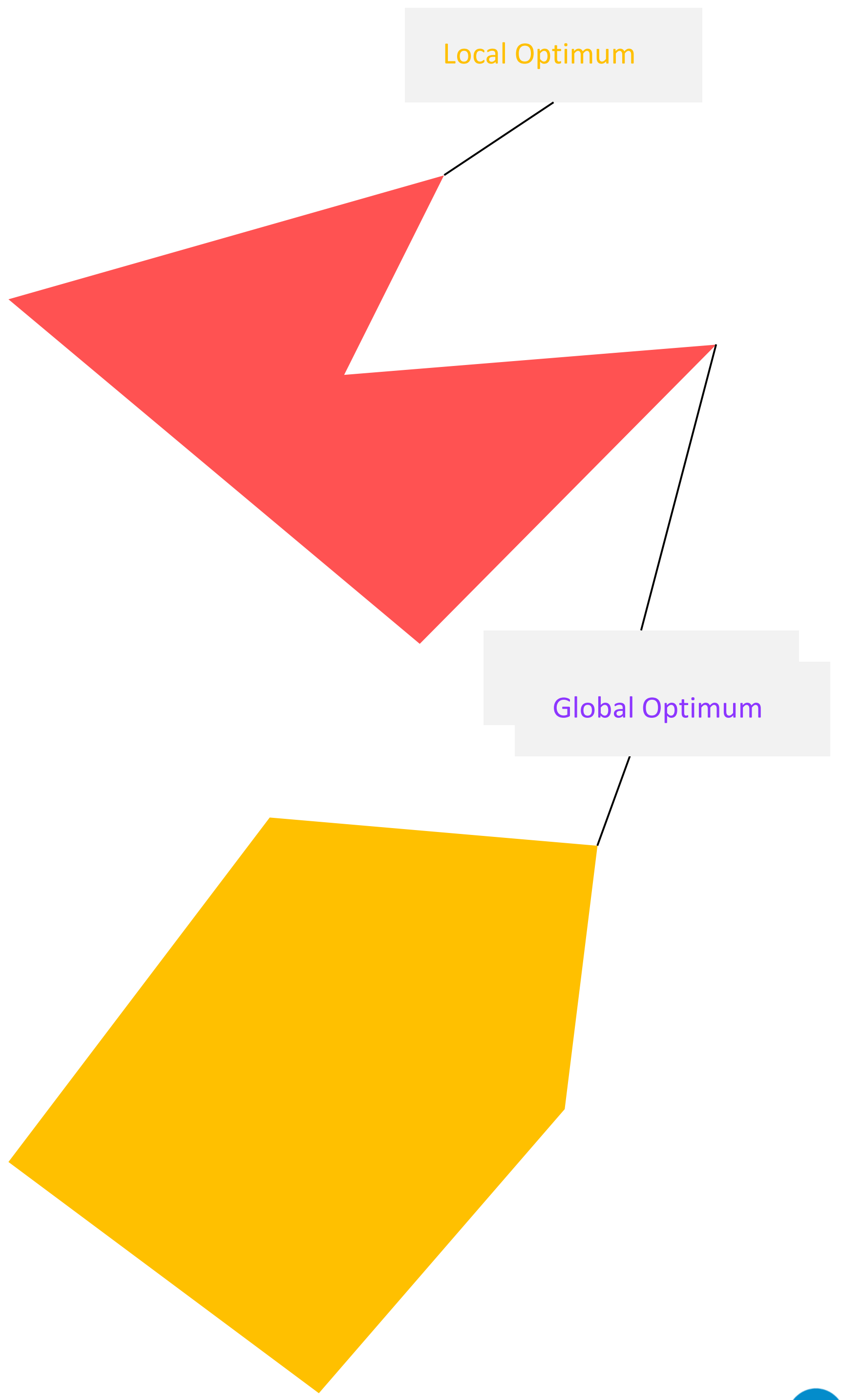
Gradient Descent

- Loss Function, J, is a measure of how well an algorithm models a dataset.
- Weights are adjusted in opposite direction of calculated gradients.



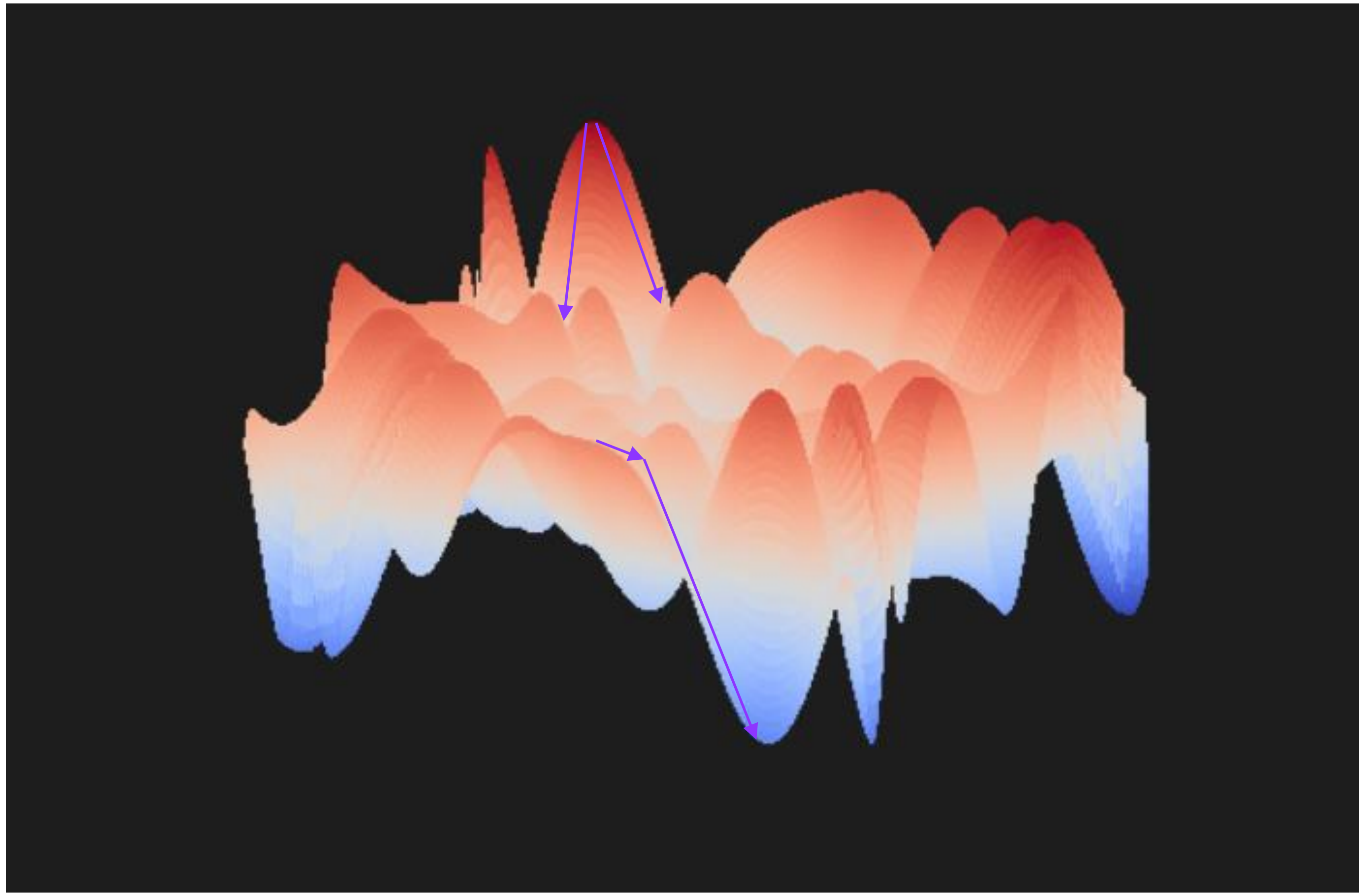
Non-Convex Error Surface

- $f: \mathbb{R}^n \rightarrow \mathbb{R}\ is\ convext\ if\ and\ if$ $\forall\ x_1, x_2 \epsilon \mathbb{R}^n\ and\ \forall \lambda \epsilon [0,1]$:
  - $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$
  - With a convex objective and a convex feasible region, there can be only one optimal solution. (globally optimal)
- Non-Convex optimization problem may have multiple feasible regions and multiple locally optimal points within each region.
  - It can take time exponential to determine there is no solution, an optimal solution exists or objective function is unbounded.

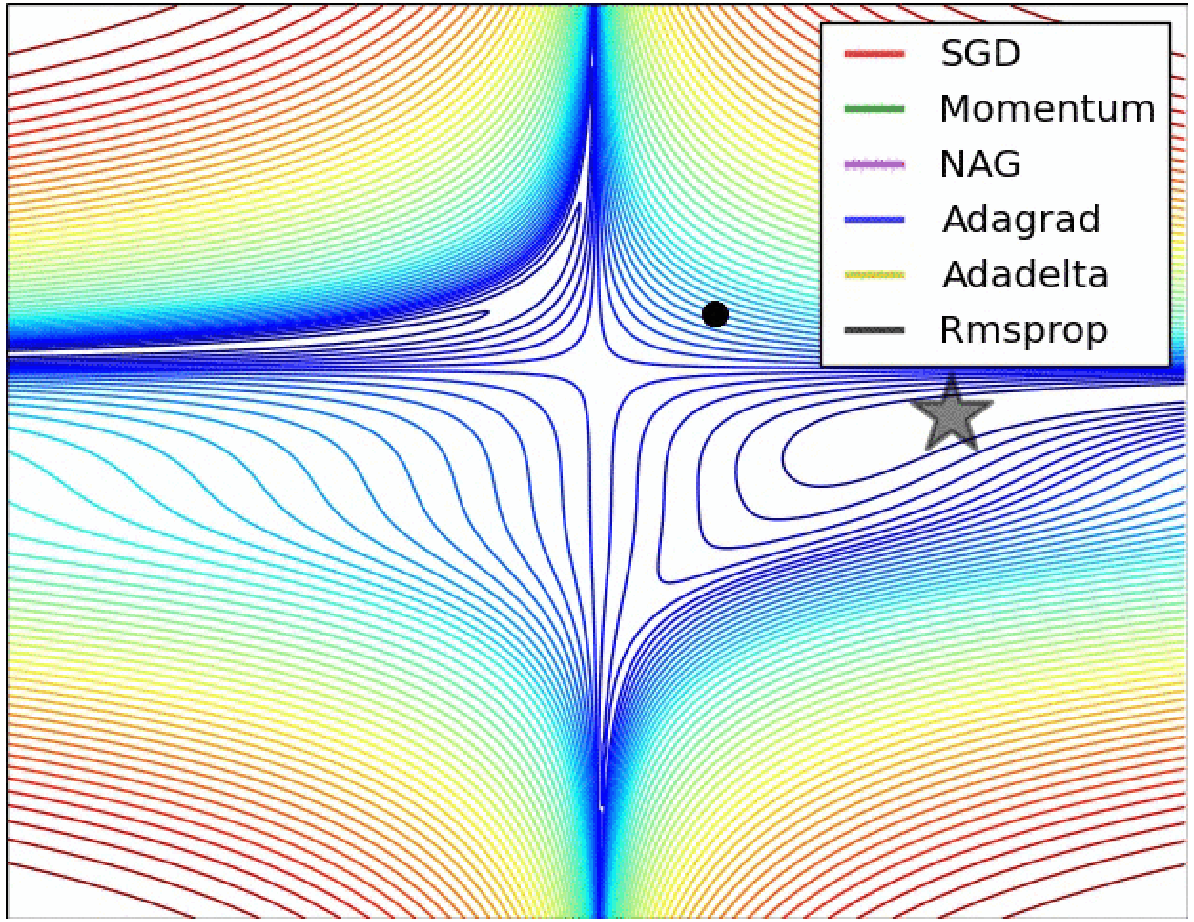


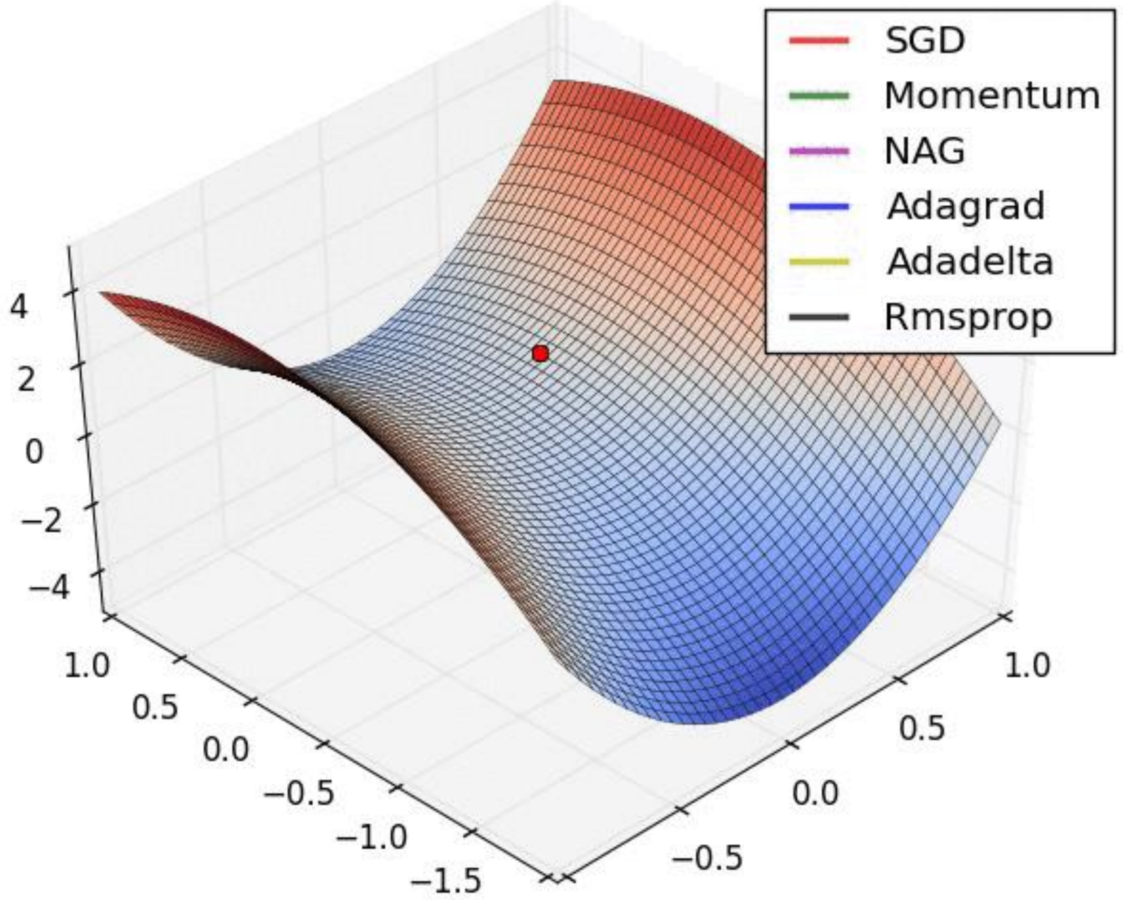
Non-Convex Error Surface

- In deep learning we almost exclusively need to solve a complex non-convex optimization problem in an n-dimensional vector space.



Optimizers

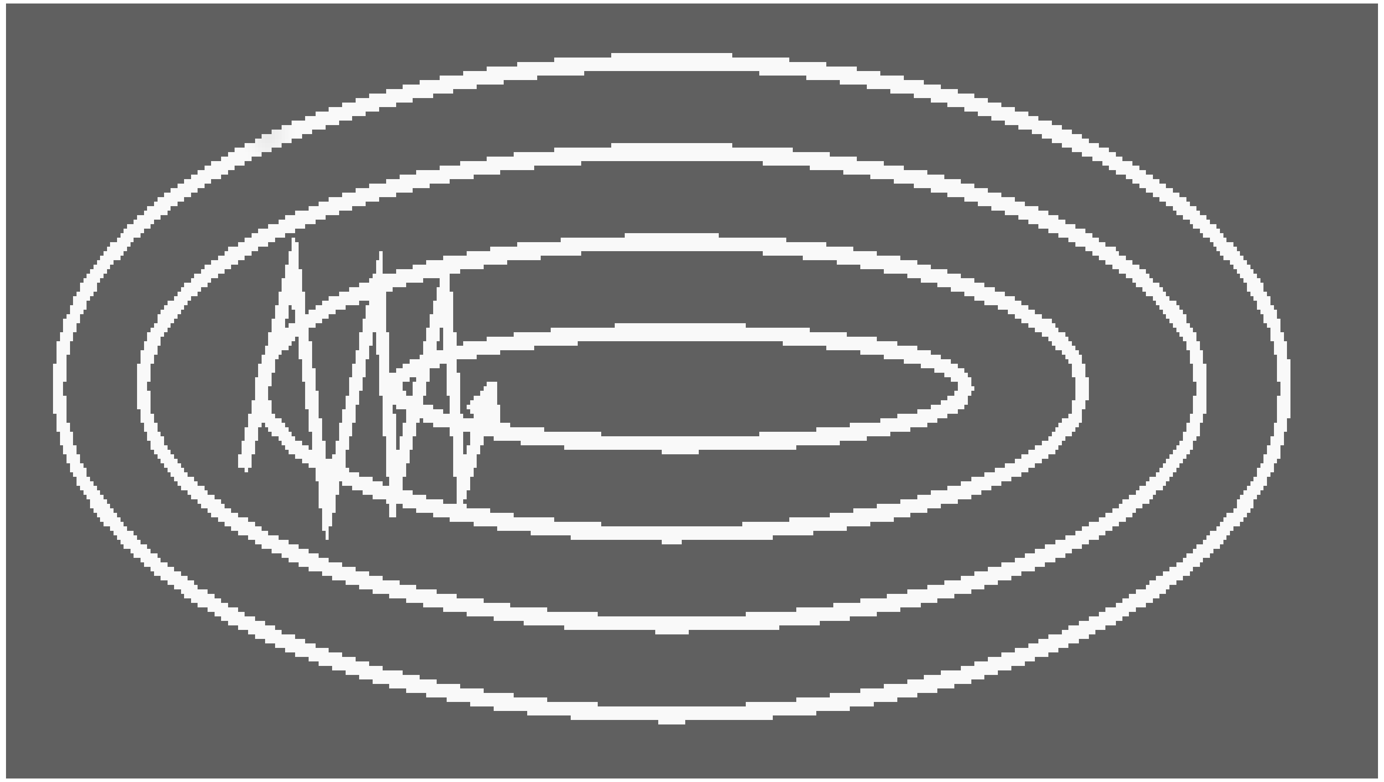




http://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants

SGD + Momentum and WD

Applies SGD to each minibatch. A good rule of thump for setting the parameters is learning rate of .001 with momentum of .9. SGD has trouble navigating areas that one one dimention is significantly more steep than others. SGD can get stuck in local minimas. adding a momentum term we can address this problem to some extent



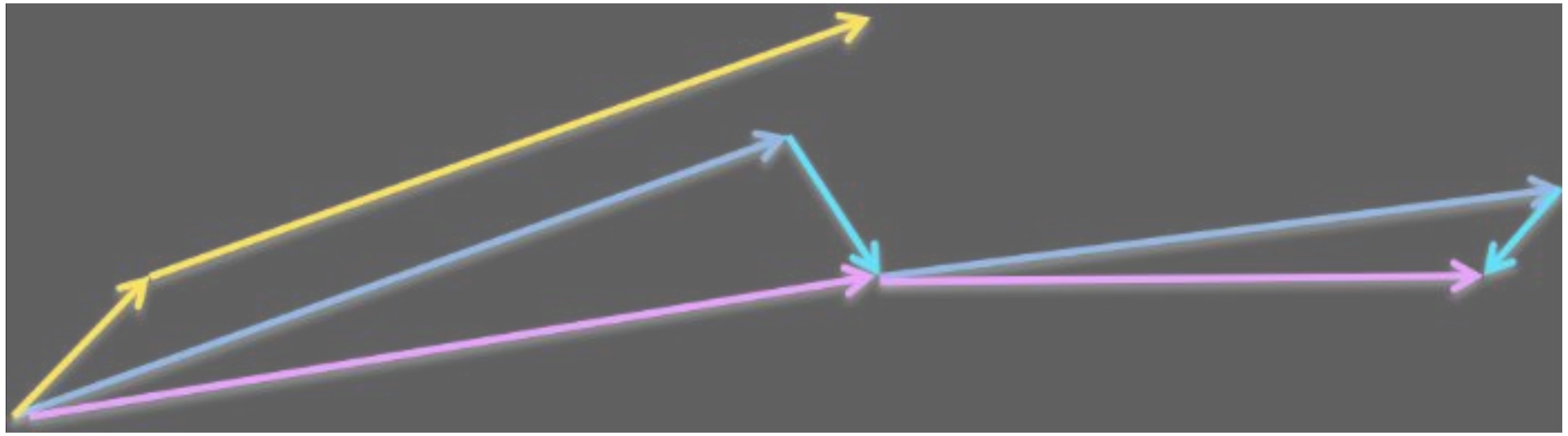
https://github.com/cyrusmvahid/GluonBootcamp/blob/master/labs/regression.ipynb

SGD

```
INFO:root:[Epoch 0] train=0.320232 val=0.311300 loss=1.828540 time: 6.539909
INFO:root:[Epoch 1] train=0.499419 val=0.504000 loss=1.360491 time: 6.521951
INFO:root:[Epoch 2] train=0.589263 val=0.563700 loss=1.134461 time: 6.828416
INFO:root:[Epoch 3] train=0.651242 val=0.636300 loss=0.973226 time: 6.240821
INFO:root:[Epoch 4] train=0.696394 val=0.624500 loss=0.856011 time: 5.716242
INFO:root:[Epoch 5] train=0.726643 val=0.680200 loss=0.777433 time: 5.796741
INFO:root:[Epoch 6] train=0.752424 val=0.703200 loss=0.703050 time: 6.936821
INFO:root:[Epoch 7] train=0.769631 val=0.685400 loss=0.654198 time: 6.403769
INFO:root:[Epoch 8] train=0.788081 val=0.762600 loss=0.605454 time: 5.668295
INFO:root:[Epoch 9] train=0.798658 val=0.770500 loss=0.576482 time: 6.360962
INFO:root:[Epoch 10] train=0.805749 val=0.759700 loss=0.553458 time: 6.030818
INFO:root:[Epoch 11] train=0.814223 val=0.700600 loss=0.532443 time: 6.420320
INFO:root:[Epoch 12] train=0.822676 val=0.742500 loss=0.506356 time: 5.543132
INFO:root:[Epoch 13] train=0.832812 val=0.777800 loss=0.482470 time: 5.963372
INFO:root:[Epoch 14] train=0.842548 val=0.793400 loss=0.453167 time: 5.632769
INFO:root:[Epoch 15] train=0.847296 val=0.772600 loss=0.440297 time: 5.402728
INFO:root:[Epoch 16] train=0.851402 val=0.774200 loss=0.428900 time: 5.821625
INFO:root:[Epoch 17] train=0.855749 val=0.790000 loss=0.415191 time: 5.847228
INFO:root:[Epoch 18] train=0.860357 val=0.798300 loss=0.400935 time: 5.816122
INFO:root:[Epoch 19] train=0.866607 val=0.815800 loss=0.384234 time: 5.358579
```

NAG

- Momentum does continue down the slope with speed. Knowing where the gradient is headed helps slowing down the descent towards the local minimas.
- NAG looks ahead to approximate the future position of parameters and adaptively adjust the momentum.



- NAG first makes a big jump in the direction of the previous accumulated gradient
- measures the gradient and then makes a correction , which results in the complete NAG update.
- NAG can help with RNN training performance.

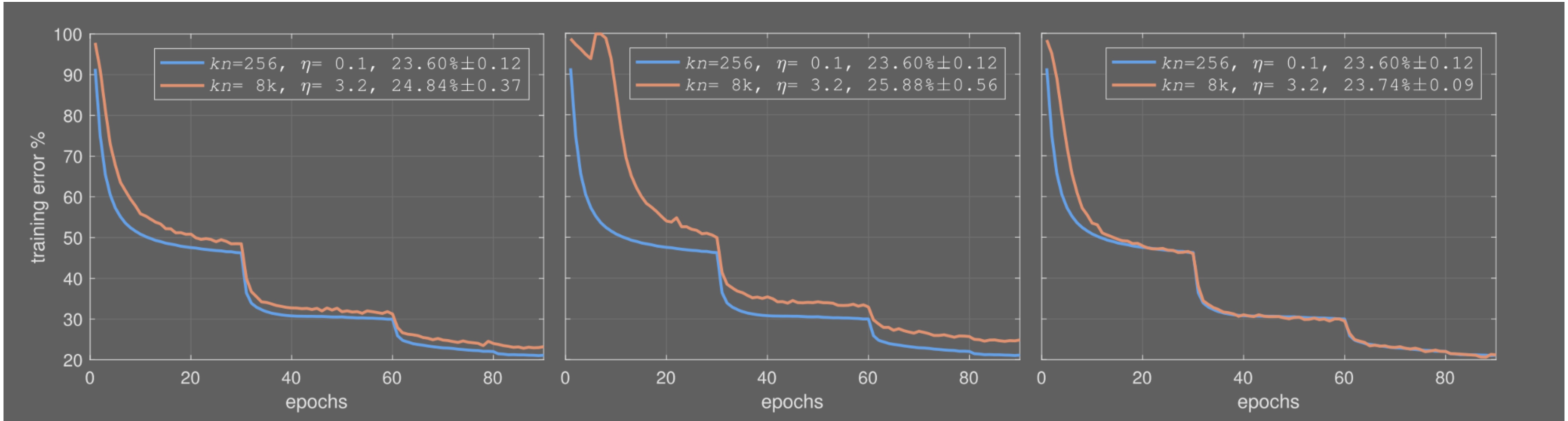
https://github.com/cyrusmvahid/GluonBootcamp/blob/master/labs/regression.ipynb

NAG

```
INFO:root:[Epoch 0] train=0.308433 val=0.319800 loss=1.886015 time: 6.550777
INFO:root:[Epoch 1] train=0.471334 val=0.500900 loss=1.434551 time: 5.405174
INFO:root:[Epoch 2] train=0.571675 val=0.560500 loss=1.186487 time: 5.425138
INFO:root:[Epoch 3] train=0.641867 val=0.642200 loss=1.002207 time: 5.514042
INFO:root:[Epoch 4] train=0.693770 val=0.654600 loss=0.866025 time: 5.631780
INFO:root:[Epoch 5] train=0.731831 val=0.712600 loss=0.763841 time: 6.143763
INFO:root:[Epoch 6] train=0.757812 val=0.713200 loss=0.693119 time: 5.937472
INFO:root:[Epoch 7] train=0.779327 val=0.737500 loss=0.634709 time: 6.136395
INFO:root:[Epoch 8] train=0.795513 val=0.760800 loss=0.589994 time: 5.702718
INFO:root:[Epoch 9] train=0.808313 val=0.746400 loss=0.552479 time: 5.358437
INFO:root:[Epoch 10] train=0.818129 val=0.772100 loss=0.523184 time: 5.997128
INFO:root:[Epoch 11] train=0.827544 val=0.799500 loss=0.500787 time: 5.829901
INFO:root:[Epoch 12] train=0.834896 val=0.772900 loss=0.471773 time: 5.716117
INFO:root:[Epoch 13] train=0.844692 val=0.782900 loss=0.452683 time: 6.698682
INFO:root:[Epoch 14] train=0.850160 val=0.791600 loss=0.431896 time: 6.603370
INFO:root:[Epoch 15] train=0.855409 val=0.806500 loss=0.416784 time: 6.226276
INFO:root:[Epoch 16] train=0.861098 val=0.792300 loss=0.398030 time: 5.621616
INFO:root:[Epoch 17] train=0.866847 val=0.823800 loss=0.385180 time: 5.526948
INFO:root:[Epoch 18] train=0.870994 val=0.805400 loss=0.367664 time: 5.894297
INFO:root:[Epoch 19] train=0.875441 val=0.770500 loss=0.359724 time: 5.594626
```

LBSGD with Momentum and WD

Large batches help increasing training speed, but come at teh cost of loss of stability. There are techniques that allow us to train models on large batches (1000 scale) while performing as well as training the same model on small batch size. The techniques hinge on the idea of adaptive batch size and include warmupss, Linear Scaling, and Batch Normalization



(a) no warmup (b) constant warmup (c) gradual warmup

LBSGD

```
INFO:root:[Epoch 0] train=0.310337 val=0.261300 loss=1.858803 time: 6.850179
INFO:root:[Epoch 1] train=0.493790 val=0.492400 loss=1.374855 time: 5.764549
INFO:root:[Epoch 2] train=0.584836 val=0.547400 loss=1.150604 time: 6.900003
INFO:root:[Epoch 3] train=0.647596 val=0.569500 loss=0.984875 time: 5.679046
INFO:root:[Epoch 4] train=0.686679 val=0.655000 loss=0.878320 time: 6.349431
INFO:root:[Epoch 5] train=0.717808 val=0.645500 loss=0.800551 time: 6.575341
INFO:root:[Epoch 6] train=0.746695 val=0.650900 loss=0.719336 time: 5.927774
INFO:root:[Epoch 7] train=0.764864 val=0.646100 loss=0.671761 time: 5.562227
INFO:root:[Epoch 8] train=0.780228 val=0.684500 loss=0.629181 time: 6.023513
INFO:root:[Epoch 9] train=0.795172 val=0.722300 loss=0.587467 time: 6.372227
INFO:root:[Epoch 10] train=0.806390 val=0.778500 loss=0.552990 time: 6.265981
INFO:root:[Epoch 11] train=0.815284 val=0.776300 loss=0.530719 time: 5.755747
INFO:root:[Epoch 12] train=0.825781 val=0.736600 loss=0.503931 time: 6.308614
INFO:root:[Epoch 13] train=0.829567 val=0.781600 loss=0.489560 time: 5.724034
INFO:root:[Epoch 14] train=0.837079 val=0.745200 loss=0.465858 time: 6.084397
INFO:root:[Epoch 15] train=0.846134 val=0.794900 loss=0.443486 time: 6.147125
INFO:root:[Epoch 16] train=0.851322 val=0.802900 loss=0.430405 time: 6.474534
INFO:root:[Epoch 17] train=0.855529 val=0.795500 loss=0.414212 time: 5.807887
INFO:root:[Epoch 18] train=0.857973 val=0.803400 loss=0.404103 time: 5.543543
INFO:root:[Epoch 19] train=0.866326 val=0.793300 loss=0.384499 time: 6.101834
```

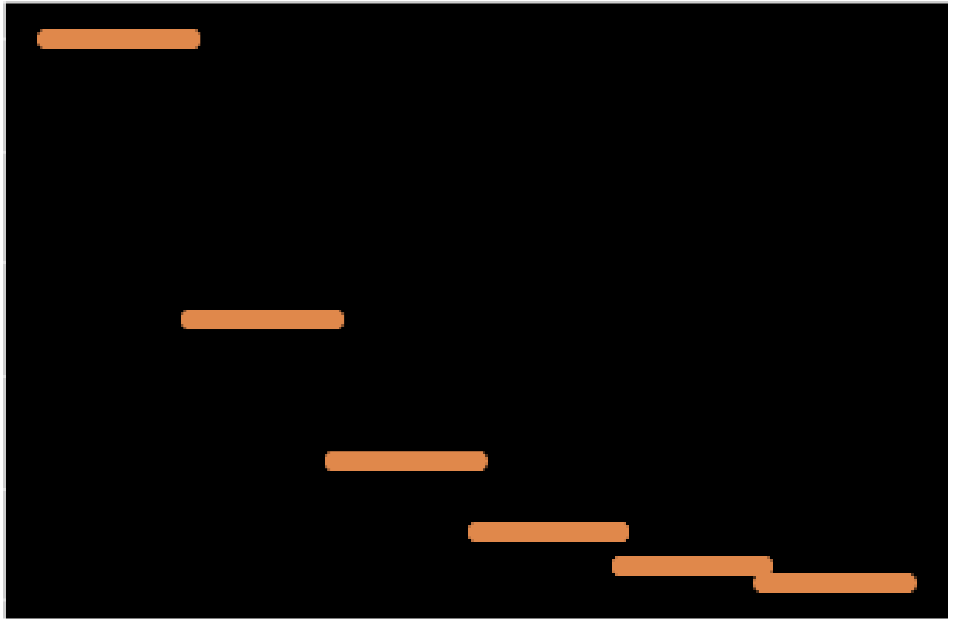
Learning Rate Scheduler

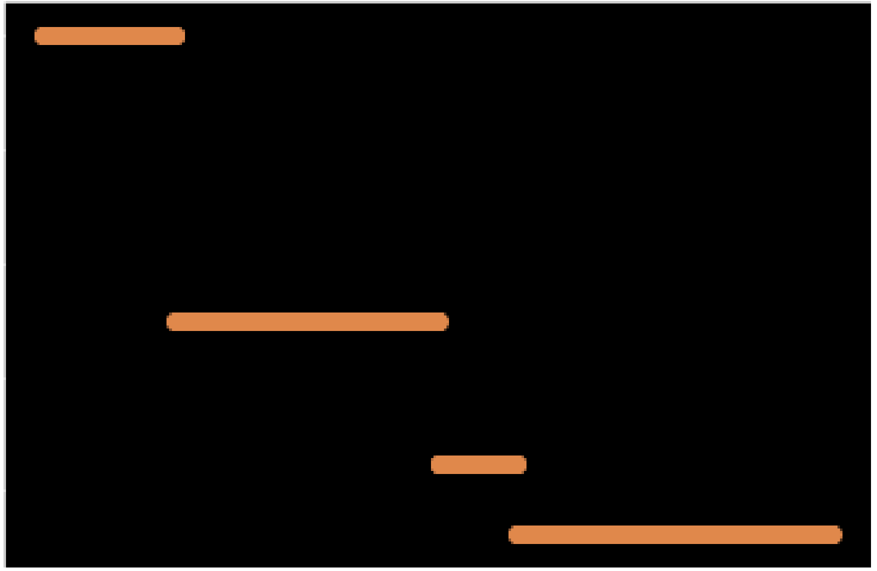
- Schedules define how the learning rate changes over time and are typically specified for each epoch or iteration (i.e. batch) of training. Schedules differ from adaptive methods (such as AdaDelta and Adam) because they:
  - change the global learning rate for the optimizer, rather than parameter-wise learning rates
  - don’t take feedback from the training process and are specified beforehand

Fixed Factor

```
schedule = mx.lr_scheduler.FactorScheduler(step=250, factor=0.5)
schedule.base_lr = 1
plot_schedule(schedule)
```

Miltifactor





```
schedule = mx.lr_scheduler.MultiFactorScheduler(step=[250, 750, 900],
factor=0.5) schedule.base_lr = 1
plot_schedule(schedule)
```

Factor

https://mxnet.incubator.apache.org/tutorials/gluon/learning_rate_schedules.html

Thank You