AWS MxNet Gluon Notes

GluonCV is based on Apache MXNet

- hybridize() converts Imperative to symbolic

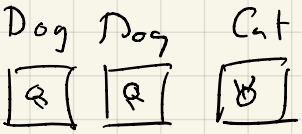
David Hubel & Torsten Wiesel
Receptive fields of single Neurons in the cat's striate cortex Johns Hopkins 1959

See edges before complex patterns

CNN
- Kunihiko Fukushima - Neocognitron
- Yann LeCun - LeNets

Image Classification — Labels for Image
Dog vs Cat

Object Detection —



Semantic Segmentation —

Instance Segmentation —

AWS Sagemaker

ML >> Sage Maker

Name: Coursera-demo

type: ml.t2.medium
t3

Life cycle configuration - not used - but for template create

Volume size - adjusted to 10 GB

Create a new Role

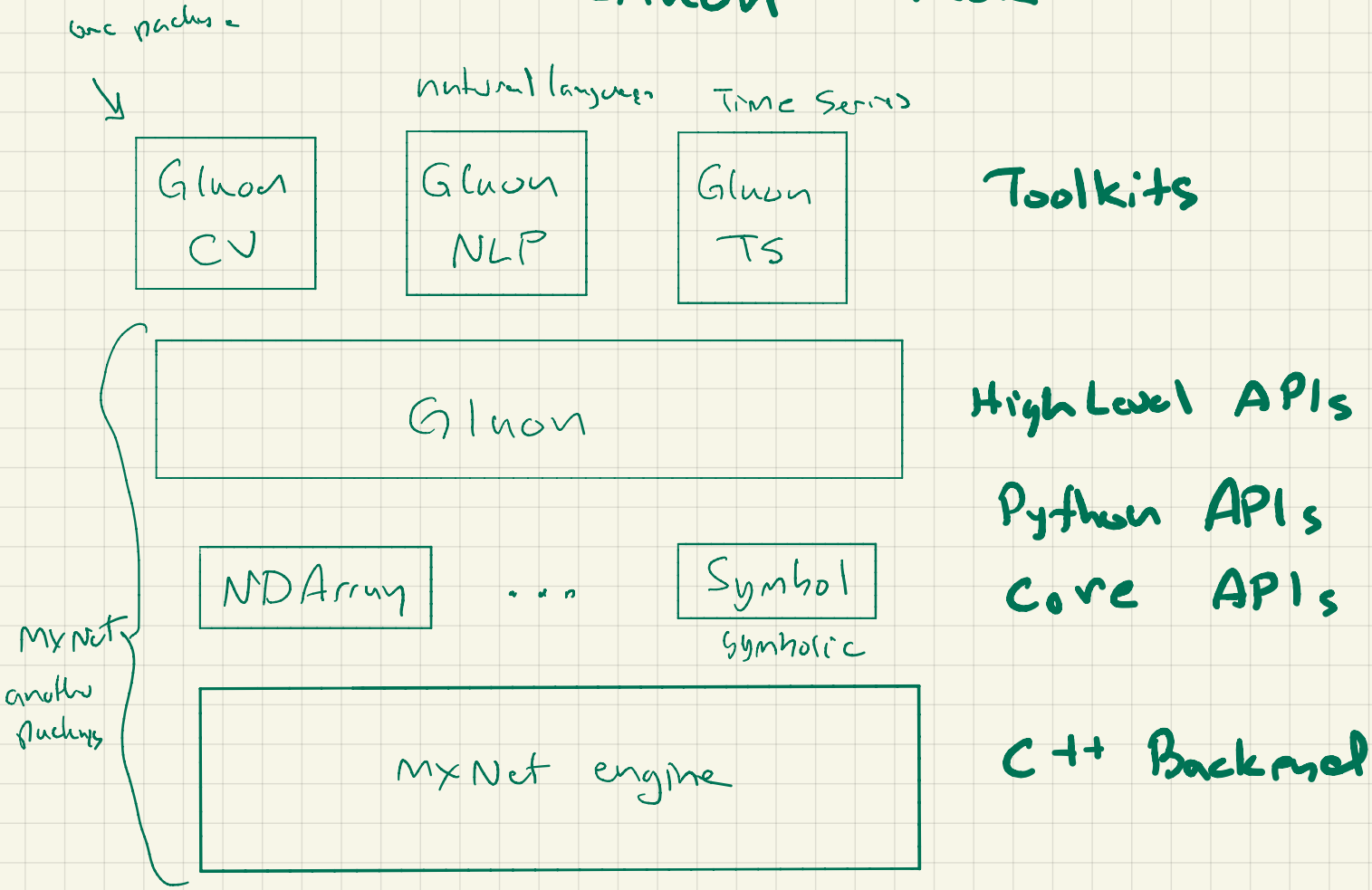
- Set S3 to none

IAM Role - Amazon Sage Maker - Execution Role

New Conda_Mxnet_p36

```
import mxnet as mx
a = mx.nd.array([1,2,3])
a.square()
```

Gluon Stack



if using an ec2 instance

```
source activate mxnet_p36
python
import mxnet as mx
```

python

MxNet Installation

Basic

pip install mxnet

CPU optimized

intel math kernel

pip install mxnet-mkl

GPU optimized

Cuda

pip install mxnet-cu101

Cuda 10.1

activate environment before installation

GluonCV Installation

Stable

pip install gluoncv

nightly

pip install gluoncv --pre

Sanity Check

```
python
import mxnet
import gluoncv
```

Image Classification

Datasets

CIFAR-10

10 basic classes

60,000 images

32 pixels x 32 pixels

ImageNet

2009

14 million Images

22K classes

ImageNet 22K or ImageNet 1k

22K classes

14M images

1K classes

1M images

flatten hierarchy

↑

starting point for Transfer learning

Classification model Architectures

ResNet

MobileNet - good choice for mobile devices

ResNet 18 & 50

VGG - Superseeded by ResNet

ResNet & Logits

Understanding Softmax

$\text{logits}_i \in [-\infty, +\infty]$

$\text{softmax}(\text{logits}_i) \in [0,1]$

and

$\sum_i \text{softmax}(\text{logit})_i = 1$

$$\text{softmax}(\text{logits})_i = \frac{e^{\text{logits}_i}}{\sum_j e^{\text{logits}_j}}$$

logits are exponentiated before normalizing.

A higher class logit will lead to a higher class probability but its size will depend upon other logits.

Some examples of this. →

```
mx.nd.softmax(mx.nd.array((-0.5, 0.5)))
```

[0.2689, 0.731]

Positive logit gives higher probability than the negative logit.

```
mx.nd.softmax(mx.nd.array((-0.5, 1.0)))
```

still need everything to sum to 1

[0.1824, 0.818]

```
mx.nd.softmax(mx.nd.array((-1.0, 1.0)))
```

[0.119, 0.881]

same values in set

```
mx.nd.softmax(mx.nd.array((1.0, 1.0)))
```

equal probability

[0.5, 0.5]

Coursera week 3 lesson 3

OBJECT DETECTION

Data Sets

Pascal VOC

2007 10 k images
24,500 objects

2012 11.5 k images

Coco Common objects in Context

2017 123 k
886 k obj
80 classes

Sports foods
beds refrigerators

Model Architectures

Faster-RCNN ← best mean average precision ← slowest w/ ResNet

SSD w/ VGG or ResNet or MobileNet

YOLO w/ DarkNet or MobileNet

↑ yolo3 ↖ faster?

Image Segmentation

makes low level predictions → Image Segmentation

Semantic

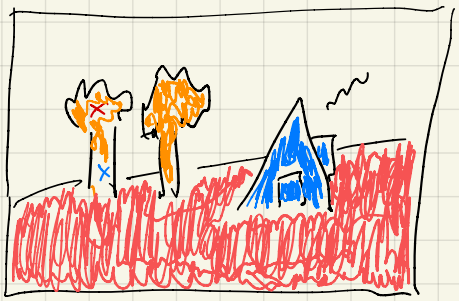
Instance

Example of Semantic Segmentation

classify every pixel

× a pixel

×



# -1	Background			default is black
# 0	House	5%	15%	assign blue
# 1	Tree	90%	70%	assign yellow
# 2	Earth	5%	15%	assign pink

One dataset has 20k images & 140 classes

Datasets semantic segmentation

Pascal VOC

2012 2900 images
6 900 classes

Image Segmentation

COCO

2017 123,000 images
886,000 objects

only use 20 classes from 80. Same 20 as Pascal VOC

ADE 20K

20,000 images
~ 400,000 objects
1404 classes

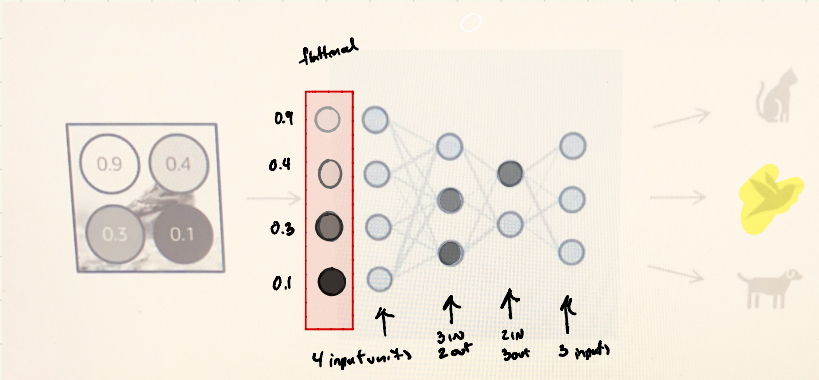
Models - Semantic segmentation

FCN with ResNet

PSP with "

Deeplab "

Neural Network Essentials

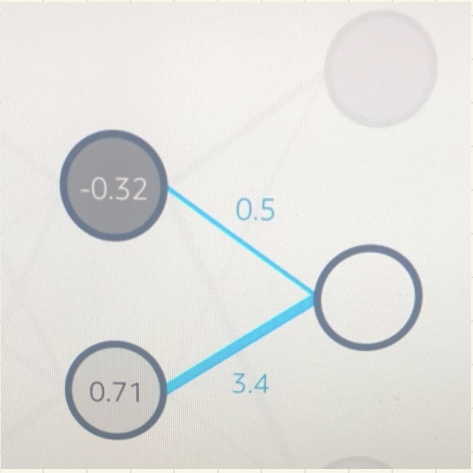


all output units connected to all input units. Connections are weighted.

Neural Network Training Procedure

0. Initialize - above w/ high level forward pass.
1. Forward Pass
2. Backward Pass
3. Update Step

Forward Pass



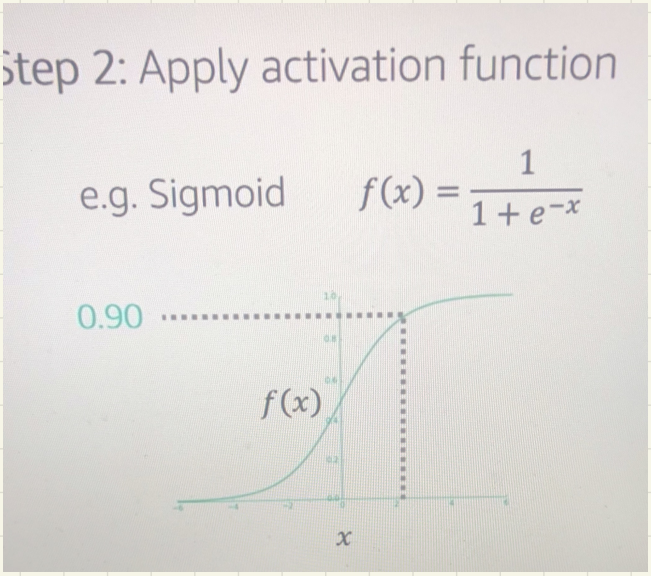
Step 1: Sum up contributions

$$(-0.32 * 0.5) + (0.71 * 3.4) = 2.254$$

Step 2: Apply activation function - a non linear function

e.g. sigmoid $f(x) = \dfrac{1}{1+e^{-x}}$

Called activation because it determines how active a neuron output is.



Backward Pass

loss function

start with gradient of loss for last layer.

Back propagation

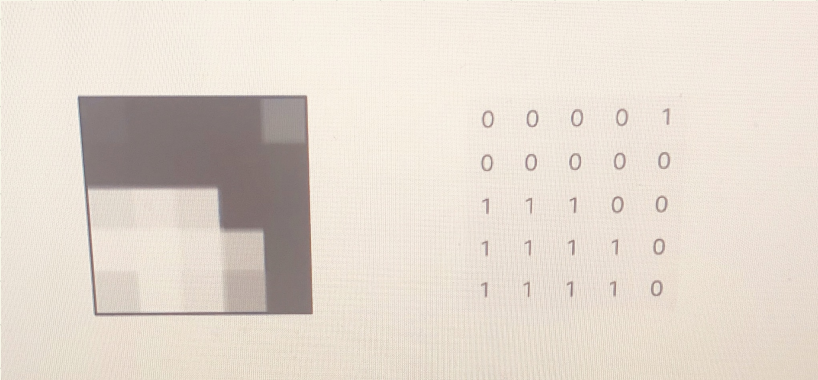
Update step. with all new weights updated.

Convolutional Networks

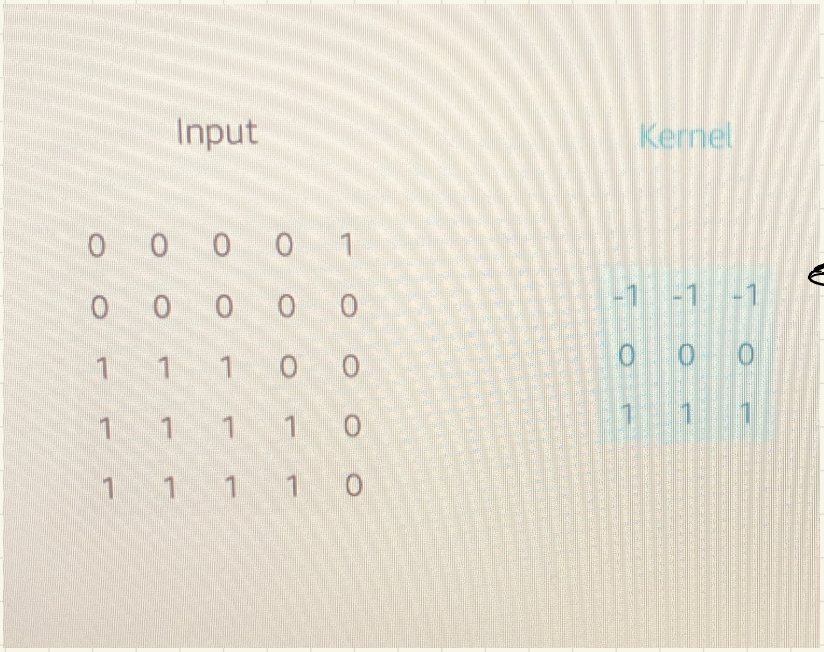
Learn local patterns

Take small region of Image.

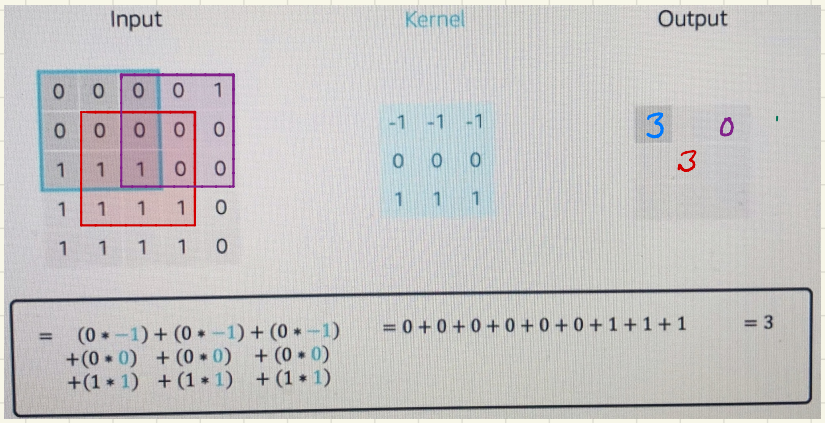
Threshold the pixels.

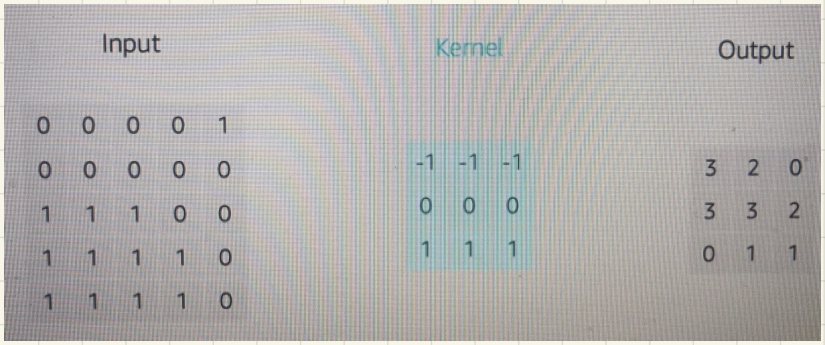


Start w/ 3x3 kernel. Usually its random values and optimized during training. Here we start with values shown.



← this will find vertical edges



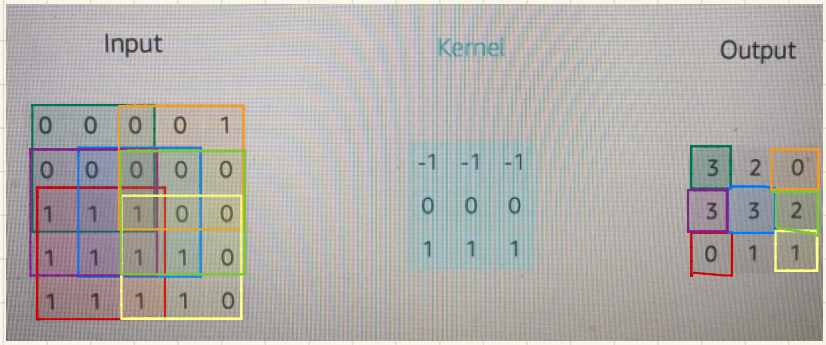


Each window can be done in parallel since each output cell calculation is independent of other outputs.

We find higher values in output corresponding to windows with a vertical edge. ← He says vertical but he means horizontal.

Horizontal edge

No edge



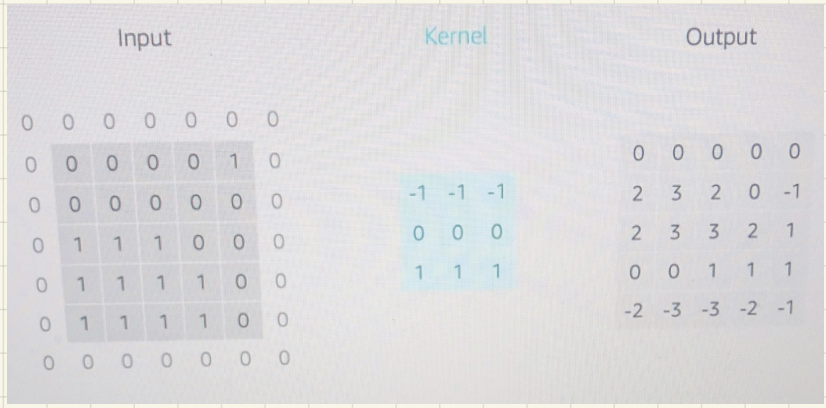
0 0 0
0 0 0 ⇒ 3 ⇐ 0 0 0
1 1 1 ⇑ 1 1 1
1 1 1

0 0 0
1 1 0
1 1 1

0 0 0
1 0 0 ⇒ 2 ⇐ 0 0 0
1 1 0 1 1 0

1 0 0
1 1 0 ⇒ 1 ⇐ 1 1 0
1 1 0 1 1 1
1 1 1

Since you lose resolution, a padding row of all zero's are added.



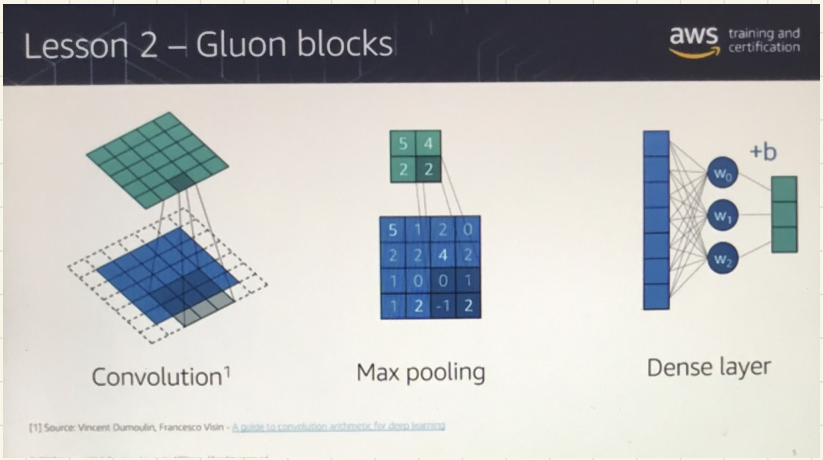
MXNet operators and blocks will auto pad for us.

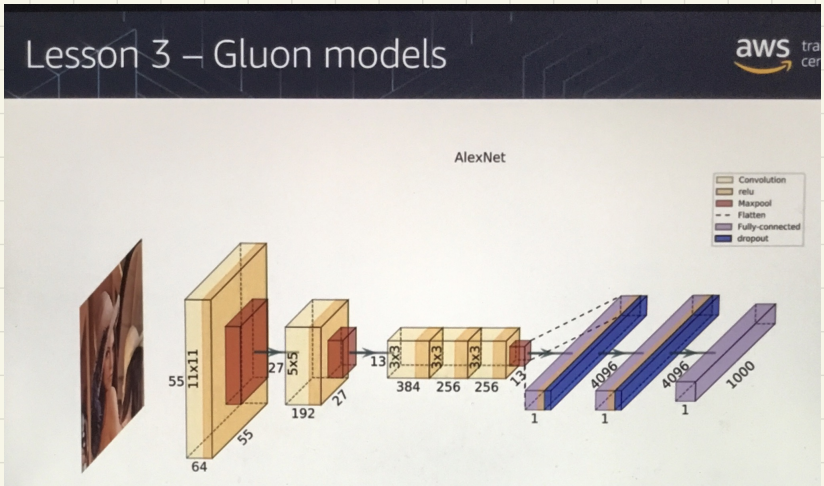
Overall we have used nine parameters to transform a 5x5 input to a 5x5 spatial output of vertical edge features.

Week 4

MxNet ND Arrays Overview

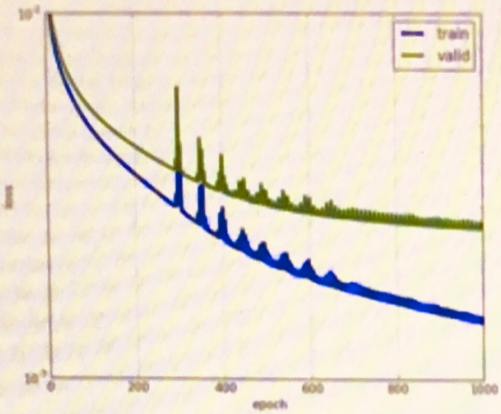
512 x 512 x 3 ← Channel RGB



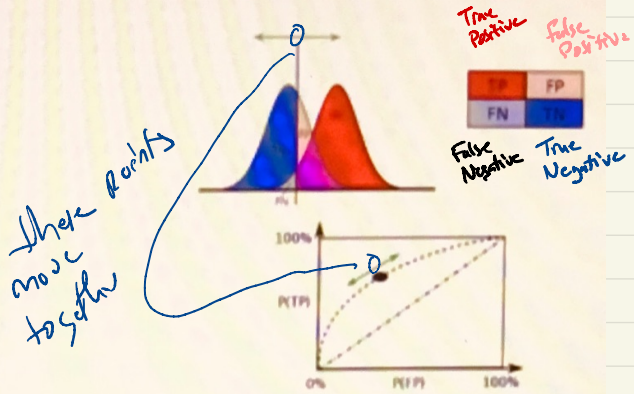


Lesson 4 – Model evaluation

aws



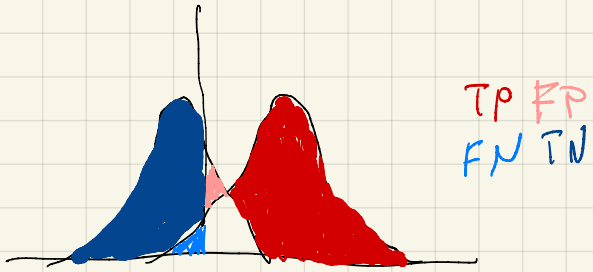
Model loss[1]



Model metrics[2]

[1] Source: How to interpret "loss" and "accuracy" for a machine learning model
[2] Source: Receiver operating characteristic

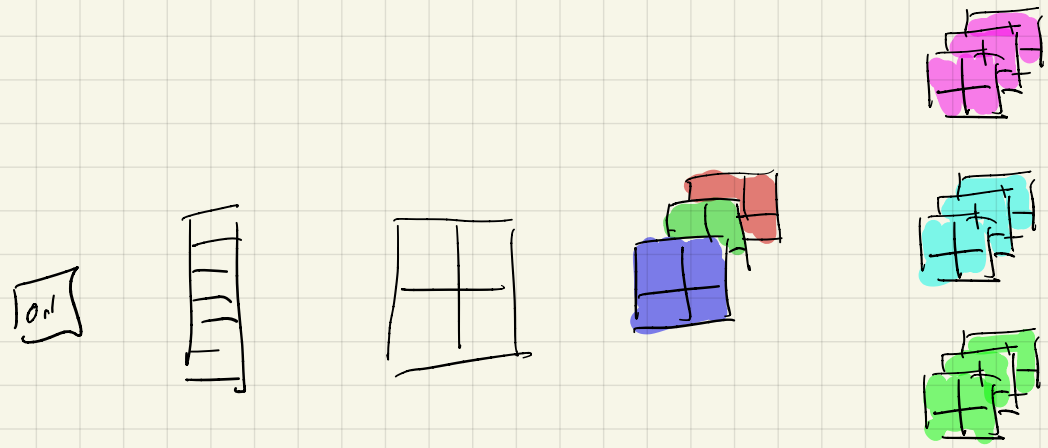
© 2019 Amazon Web Services, Inc. or its Affiliates. All rights reserved.



MxNet ND Arrays Detail

ND Arrays <=> Tensors Same thing

Examples



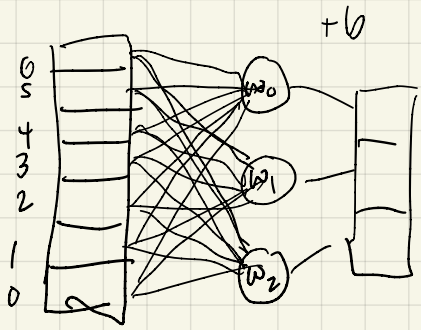
0D	1D	2D	3D	4D
Scalar	Vector	MATRIX		
(1,)	(6,)	(2,2)	(2,2,3)	(3,2,2,3)

No 0x0
So consider
(1,)

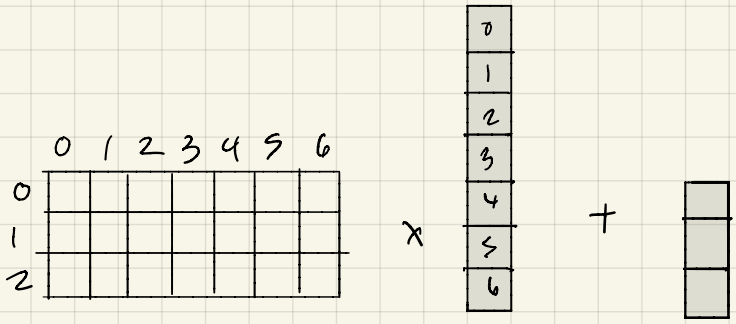
Batch
images
processing
in parallel

image is 3 matrix each value 0-255
for RGB channels.

w x input + bias



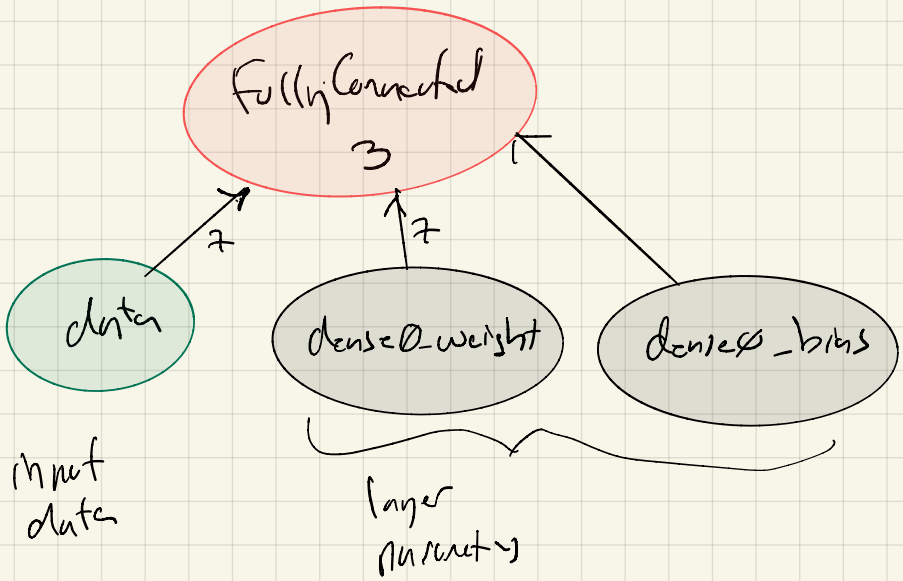
Dense layer



weights (3,7) | x (7,1) | Bias (3,)

output = **W** · X + b

Computation Graph



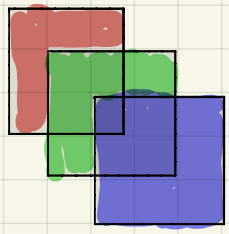


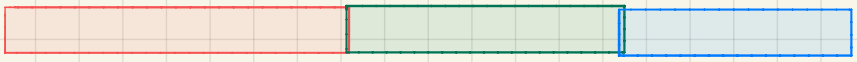
image
(3,3,3)

Row
(27)

- MXNet is a channel first framework
- Images should be input as

  (Batch, channel, height, width) = (N, C, H, W)

- OpenCV format (H, W, C)



- MXNet format (C, H, W)

MxNet vs NumPy

```
from mxnet import nd
import mxnet as mx
import numpy as np
```

NDArray from Tuples

```
nd.array(((1,2,3),(5,6,7)))
```

Output

```
[[1. 2. 3.]
 [5. 6. 7.]]
<NDArray 2x3 @cpu(0)>
```

1s

```
x = nd.ones(shape=(2,3))
x
```

Output

```
[[1. 1. 1.]
 [1. 1. 1.]]
<NDArray 2x3 @cpu(0)>
```

Randomly Sampled

```
y = nd.random.uniform(low=-1, high=1, shape=(2,3))
y
```

Output

```
[[0.897  0.186  0.403
 0.688  0.205  0.716]]
<NDArray 2x3 @cpu(0)>
```

NDArray Metadata

(x.shape, x.size, x.dtype, x.context)

Output

Product of dimensions

((2,3), 6, numpy.float32, cpu(0))

Type

nd.ones((2,3), dtype=np.int8) or .astype

out

```
[[1 1 1
 [1 1 1]]
<NDArray 2x3 @cpu(0)>
```

y.astype(np.float16)

out

```
[[0.09766  0.1857  0.4304 ]
 0.6885    0.2056  0.716 ]]
<NDArray 2x3 @cpu(0)>
```

Context

nd.ones((2,3), ctx = mx.cpu())

out

```
[[1 1 1
 [1 1 1]]
<NDArray 2x3 @cpu(0)>
```

where the NDArray is stored in memory

CPU or GPU

nd.ones((2,3), ctx = mx.gpu(1)) also added

out 1

out

```
[[1 1 1
 [1 1 1]]
<NDArray 2x3 @gpu(0)>
```

Lets see why it matters

```
In [11]: x_np, y_np = np.ones((10000,5000)), np.ones((5000,10000))
         x_nd_cpu, y_nd_cpu = nd.ones((10000,5000), ctx=mx.cpu()), nd.ones((500
         x_nd_gpu, y_nd_gpu = nd.ones((10000,5000), ctx=mx.gpu()), nd.ones((500
```

two arrays of 10k x 5k each in python numpy, cpu & gpu.

```
tic = time.time()
np.dot(x_np, y_np)
print("Numpy time {:.4f}s".format(time.time()-tic))
```

Numpy time 2.8512s

```
tic = time.time()
np.dot(x_nd_cpu, y_nd_cpu).wait_to_read()
print("Numpy time {:.4f}s".format(time.time()-tic))
```

Numpy time 2.8512s

```
tic = time.time()
np.dot(x_nd_gpu, y_nd_gpu).wait_to_read()
print("Numpy time {:.4f}s".format(time.time()-tic))
```

Numpy time 0.0855 s

Converting between MXNet and NumPy

```
a = x.asnumpy()
(type(a), a)
```

out

```
(numpy.ndarray, array([[1., 1., 1.],
                       [1., 1., 1.]],
                      dtype=float32))
```

output is blocked. Synchronous until conversion is complete

Results come back in numpy but operation still running?

wait in python until result is converted from mxnet.

Numpy gets a copy of the array.

```
nd.array(a)
```

out

```
[[1. 1. 1.]
 [1. 1. 1.]]
<NDArray 2x3 @cpu(0)>
```

Common ND Array Operations

```
from mxnet import nd
```
← ND package for operations

```
import mxnet as mx
import numpy as np
```

Define Two matrices of Same size

```
x = mx.nd.random.uniform(shape=(2,3))
y = mx.nd.full(shape=(2,3), val=2)
x, y
```

```
([[0.5488135   0.5928446   0.71518934]
  [0.84426576  0.60276335  0.8579456 ]]
<NDArray 2x3 @cpu(0)>,

[[2.  2.  2.]
 [2.  2.  2.]]
<NDArray 2x3 @cpu(0)>)
```

```
X*Y
```

```
[[1.097627   1.1856892  1.430787
  1.6885315  1.2055267  1.7158912]]
<NDArray 2x3 @cpu(0)>
```

Exponentiation

```
y.exp()
```

```
[[7.389056  7.389056  7.389056]
 [7.389056  7.389056  7.389056]]
```

Plot

```
%matplotlib
plt.plot(np.arange(0,3,0.1),
         nd.arange(0,3,0.1).exp().asnumpy())
```

convert back to numpy

Matrix Multiplication

```
x.shape , y.shape
((2,3), (2,3))
```

Need to Transpose

```
nd.dot(x, y.T)
```

```
[[3.7136948    3.7869448
  4.6099496    4.6099496]]
```

Indexing

```
x
```

```
      0                1                2
0  [[0.5488135    0.5928446    0.71518934]
1   0.8442657     0.60276335   0.8579456 ]]
```

```
x[:, 1:3]
```

R C

```
[[0.5928446    0.71518934
  0.60276335   0.8579456 ]]
```

exclusive

Row 1:2 → 1

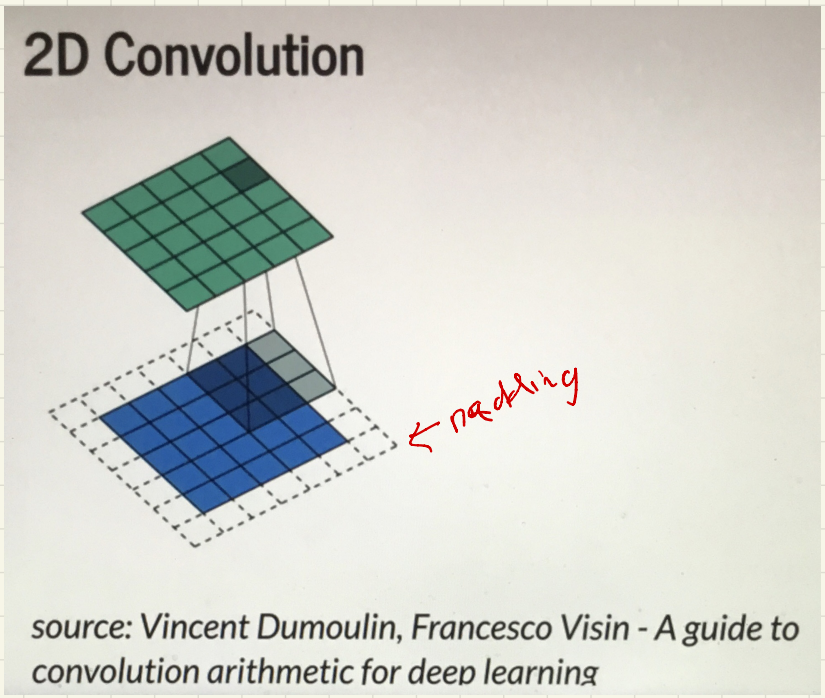
Cols 0:2 → 0, 1

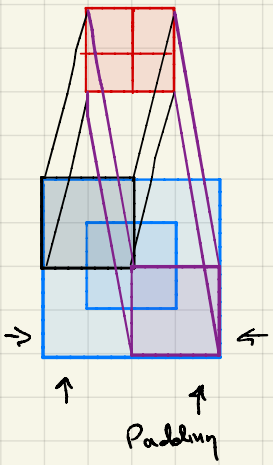
```
y[1:2, 0:2] = 4
y
```

```
      0   1   2
1  [[ 2   2   2 ]
2     4   4   2 ]]
```

Gluon Blocks

```
import mxnet as mx
from mxnet import nd
from mxnet.gluon import nn
```





← See pdf in repo

```
layer = nn.Conv2D(
    channels = 1,
    kernel_size = (3,3),
    in_channels = 1,
    strides = (1,1),
    padding = (0,0),
    activation = 'relu',
    prefix = 'conv_')
```

Mandatory

channels: how many output channels "how many different convolutional masks applied in parallel to same input"

kernel_size: size of convolutional kernel. Common size is 3x3

strides: How many cells should your kernel move between patches. If specify 2x2, then convolution operation dimishes spatial resolution by 2.

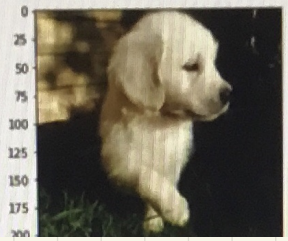
activation: Rectified Linear Units Activation

layer

Conv2D(1→1, kernel_size = (3,3), stride = (1,1), Activation(relu))

Lets try to apply convolution to this cute dog picture.

```
In [4]: o=plt.imshow(plt.imread('../support/dog3.jpg'))
```



```
layer.initialize(mx.init.Constant([
    [-1, 0, 1],  # Sobel Operator
    [-2, 0, 2],  # Detect Vertical lines
    [-1, 0, 1]]
))
```

Presentation

typos?

$$\text{Sobel}_{\text{vertical}} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

90° If1 different

$$\text{Sobel}_{\text{horizontal}} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Most Common

wikipedia

$$\begin{matrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ -1 & 0 & -1 \end{matrix}$$

90° "Just a Transpose"

$$\begin{matrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

blog

$$\begin{matrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{matrix} \quad G_x$$

x-direction

$$\begin{matrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \quad G_y$$

y-direction

Sobel VERTICAL

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

$\xrightarrow{-90°}$

Sobel Horizontal

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Load Image in gray scale

load image in grayscale

```
image = mx.image.imread('../support/dog3.jpg', 0).
                                        astype('float32')

image_channel_first = image.transpose((2,0,1))
image_batchified = image_channel_first.expand_dims(axis=0)
output = layer(image_batchified)
o = plt.imshow(output.squeeze().asnumpy(), cmap='gray')
```

← So its channel first instead of last.

← Convert 3D array to 4D array with leading batch axis - NCHW

← perform convolution

in `layer.collect_params()`

out `Conv_(`

Weight `Parameter conv_weight`

output input

`(shape=(1,1,3,3)`

`dtype=<class 'numpy.float32'>)`

1 output channel
1 input channel
3x3 kernel

out `Parameter conv_bias`

Bias `(shape=(1,)`

`dtype=<class 'numpy.float32'>)`

Accessing the weights

in `layer.collect_params()['conv_weight']`

out `Parameter conv_weight`

`(shape=(1,1,3,3)`

`dtype=<class 'numpy.float32'>)`

in `layer.weight`

out `Parameter conv_weight`

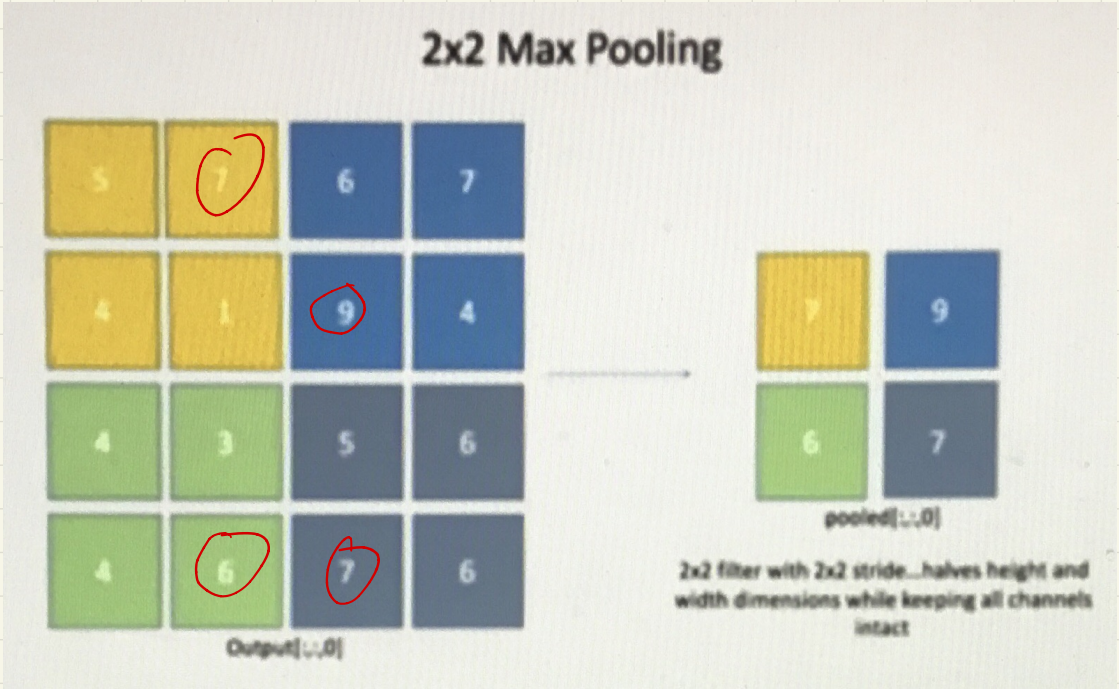
`(shape=(1,1,3,3)`

`dtype=<class 'numpy.float32'>)`

```
in      layer.weight.data()

out     [[[ -1   0   1
            -2   0   2
            -1   0   1 ]]]
        <NDArray 1x1x3 @cpu(0)>

        layer.bias.data()
         [0 ]
        <NDArray 1 @cpu(0)>
```

Max Pooling

Max



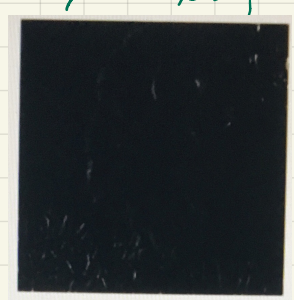
Max pooling is an operation that reduces spatial resolution of a feature map while conserving the highest activations,

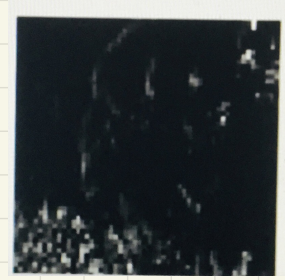
reduces feature map factor by of 4

```
layer = nn.MaxPool2D(pool_size=(4,4))
layer
```

MaxPool2D(size=(4,4), stride=(4,4), padding=(0,0), ceil_mode=False)

```
pooled = layer(output)

plt.imshow(output.squeeze().asnumpy(), cmap='gray')
plt.axis('off')
plt.show()
plt.imshow(pooled.squeeze().asnumpy(), cmap='gray')
0 = plt.axis('off')
```



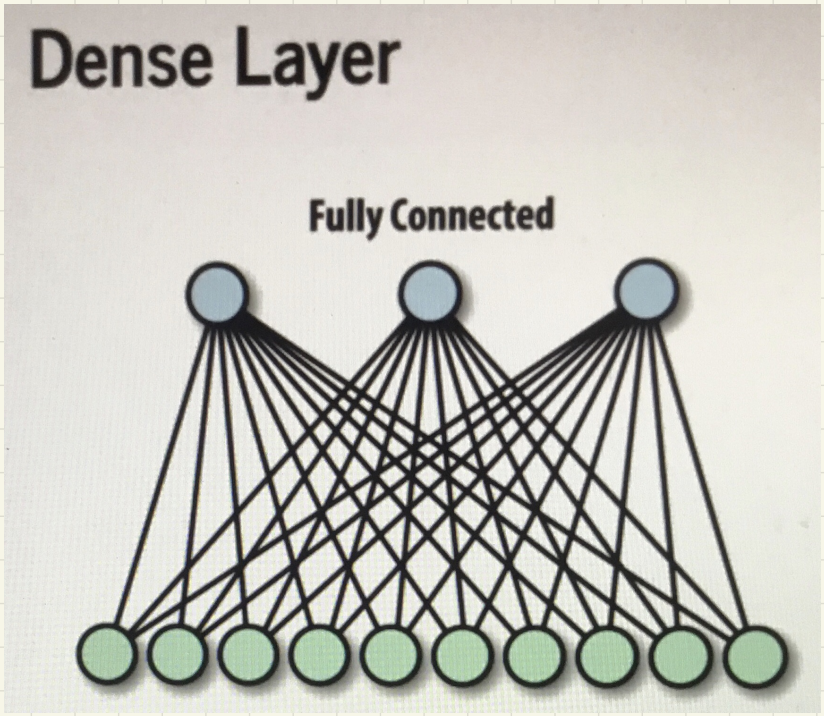


IN: layer.collect_params()

OUT: pool0_(

)

← No parameters for a pooling layer.



IN layer = nn.Dense(units=10, in_units=3, activation="relu")
layer

OUT Dense(3 -> 10, Activation(relu))

IN layer.initialize(mx.init.One())

inputs = mx.nd.ones((1,3))
inputs, layer(inputs)

initialized to

bias is ∅

Out (
[[1. 1. 1.]]
<NDArray 1x3 @cpu(0)>,
[[3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]]
<<NDArray 1x10 @cpu(0)>)

IN layer.weight.data()

OUT

```
[[1. 1. 1.]  1
 [1. 1. 1.]  2
 [1. 1. 1.]  3
 [1. 1. 1.]  4
 [1. 1. 1.]  5
 [1. 1. 1.]  6
 [1. 1. 1.]  7
 [1. 1. 1.]  8
 [1. 1. 1.]  9
 [1. 1. 1.]] 10
```

<NDArray 10x3 @cpu(0)>

layer.bias.data()

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

<NDArray 10 @cpu(0)>

Initialization of Gluon Blocks

```
import mxnet as mx

from mxnet import nd, init
from mxnet.gluon import nn
```

Different types of initialization

Weights take random values from uniform distribution (-0.07 & 0.07)

Bias are all set to zero.

Other ways to init weights

In `init.Uniform` ← random uniform weights

`init.Normal`

`init.Xavier`

`init.Zero`

`init.One`

`init.Constant`

Out `mxnet.initializer.Constant`

Xavier Initialization

Designed to keep the scale of the gradients roughly the same in all layers

The initializer fills the weights with random numbers in the range of [-c, c] where

$$c = \sqrt{\frac{\text{magnitude} = 3 \text{ by default}}{0.5 \times (\underset{9}{n_{in}} + \underset{9}{n_{out}})}} = -0.57 \ \& \ +0.57$$

IN

```
layer = nn.Conv2D(channels=1, in_channels=1
                  kernel_size=(3,3))

layer.initialize(init.Xavier)

layer.weight.data()
```

OUT

```
[[[  0.306   0.513  -0.530
    -0.193  -0.332  -0.056
     0.114  -0.571   0.395 ]]]

<NDArray 1x1x3x3 @cpu(0)>
```

Xavier Initialization large number of output

-

```
layer = nn.Conv2D(channels=512, in_channels=512
                  kernel_size=(3,3))
layer.initialize(init.Xavier)
layer.weight.data()
```

output large 3x512 array values are much smaller

-0.01 to 0.01

Deferred Initialisation

You can define layers purely by output size and it will determine input size by input?

IN
```
layer = nn.Conv2D(channels=1, kernel_size=(3,3))
layer.weight
```

don't know of depth

OUT Parameter conv2D_weight (Shape=(1, 0, 3,3), dtype < ... float32)

```
layer.initialize(init.Xavier)
layer(mx.nd.ones((1, 8, 224, 224)))
layer.weight.data()
```

OUT Parameter conv2D_weight (Shape=(1, 8, 3,3), dtype < ... float32)

Initialization Context

```
layer = nn.Conv2D(channels=1, kernel_size=(3,3))
layer.initialize(init.Xavier(), ctx=mx.gpu())

layer(nd.uniform(shape=(1,3,9,9), ctx=mx.gpu()))
```

output is also on gpu.

Set data

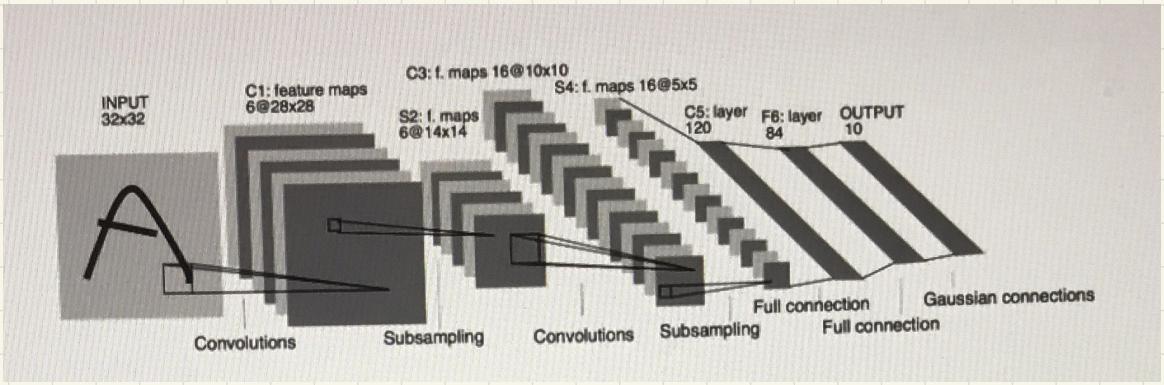
```
layer.weight.set_data(nd.ones((1,3,3,3), ctx=mx.gpu()))
layer.weight.data()
```

```
[[[ 1.  1.  1.]
  [ 1.  1.  1]
  [ 1.  1.  1]]

 [[ 1.  1.  1.]
  [ 1.  1.  1]
  [ 1.  1.  1]]

 [[ 1.  1.  1.]
  [ 1.  1.  1]
  [ 1.  1.  1]]]
<NDArray 1x3x3x3 @gpu(0)>
```

Sequential Gluon Blocks

These are all subclasses of the Gluon.block abstract class

```
import mxnet as mx
from mxnet import nd
from mxnet.gluon import nn

net = nn.Sequential()
```



For example lets try to reproduce a simple LeNet architecture
Yann Le Cun

convolution w/ 6 channels of 5x5 kernel
max pooling layer 2x2
convolutional layer of 16 channels 5x5 kernel
Maximum layer 2x2
3 fully connected
one w/ 120 tanh activation
84 hidden units
10 output units

```
net.add(
    nn.Conv2D(6, (5,5), activation='tanh')
    nn.MaxPool2D((2,2)),
    nn.Conv2D(16, (5,5)), activation='tanh')
    nn.MaxPool2D((2,2)),
    nn.Dense(120, activation='tanh')
    nn.Dense(84),
    nn.Dense(10)
)
```

We haven't specified the number of inputs

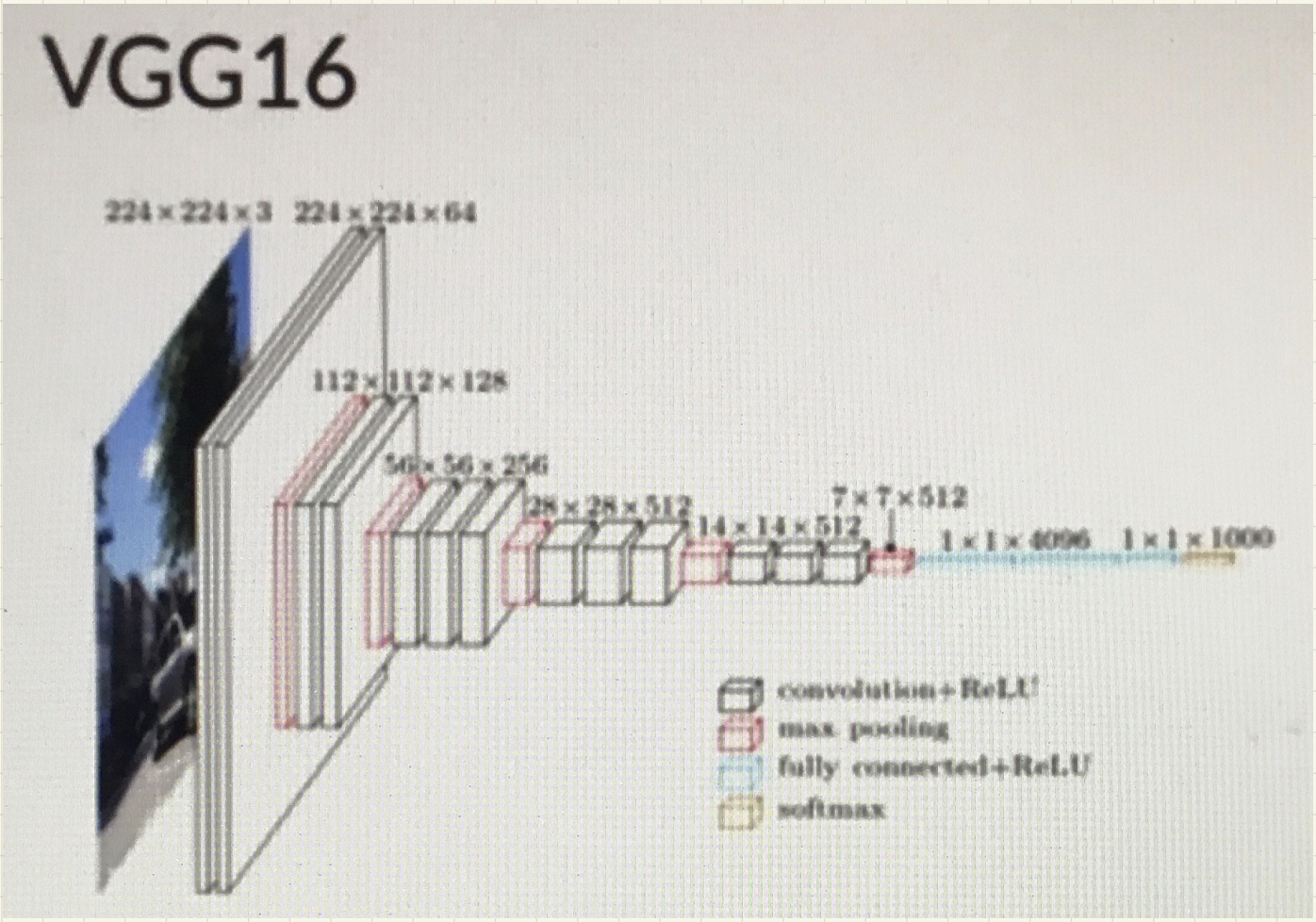
```
net.initialize()
```

grayscale image 28x28

```
net(nd.ones((1,1,28,28)))
```

equivalent to

```
net.forward(nd.ones((1,1,28,28)))
```



Another famous architecture that uses a simple sequential sequential pattern is VGG. It is popular for Image classification tasks. Its large number of parameters make it impractical compared to smaller or more efficient new architectures like ResNet or Inception.

However the new architectures cannot be encapsulated simply by chaining together components in a sequential fashion.

Custom Gluon Blocks

```
import mxnet as mx
from mxnet import nd
from mxnet.gluon import nn
```

multi layer perceptron

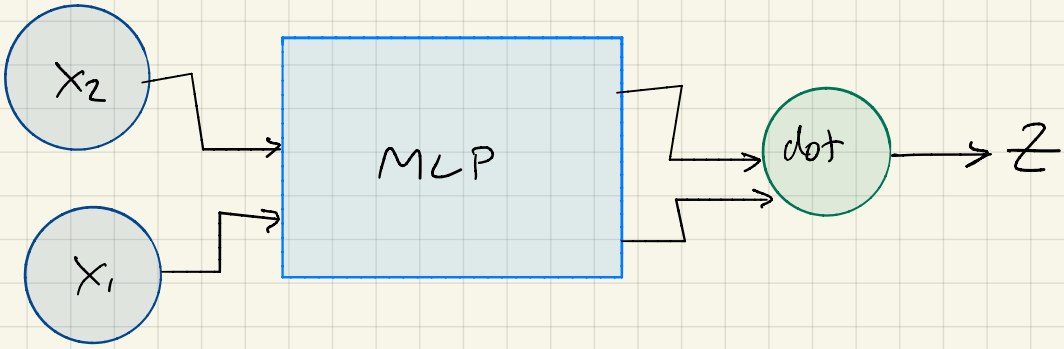
model constructor

```
class MLP(nn.Block):
    def __init__(self, hidden_units=256, **kwargs):
        # Call the construct of the MLP parent class
        # Block to perform ....
        # necessary initialization.
        # In this way other function parameters...
        # also be specified when constructing an
        # instance, such as the parameter, params,
        # described in the following sections
        super(MLP, self).__init__(**kwargs)
        self.hidden = nn.Dense(hidden_units,
                               activation='relu')  # HIDDEN LAYER
        self.output = nn.Dense(10)  # OUTPUT LAYER
    def forward(self, x):
        y = self.hidden(x)
        return self.output(y)
```

another name for fully connected neural network

Describes how data flows through the network

```
net = MLP(hidden_units=512)
net.initialize()
net(nd.random.uniform(shape=(1,64)))
```



use MLP to build a siamese network

```
class SiameseNetwork(nn.Block):

    def __init__(self, hidden_units = 256, **kwargs):
        super(SiameseNetwork, self).__init__(**kwargs)
        self.mlp = nn.Sequential()
        with self.mlp.name_scope():
            self.mlp.add(
                nn.Dense(128, activation='relu'),
                nn.Dense(128, activation='relu'),
                nn.Dense(128, activation='relu')
            )

    def forward(self, x1, x2):
        y1 = self.mlp(x1)
        y2 = self.mlp(x2)
        y1 = y1.expand_dims(axis=1)      # we add dummy
        y2 = y2.expand_dims(axis=2)      # dimensions to have
        return nd.batch_dot(y1, y2)      # Y1: (N,1,C)
                                         # Y2: (N,C,1)
```

previous match

```
net = SiameseNetwork()
net.initialize()
x1 = nd.random.uniform(shape = (1, 64))
x2 = nd.random.uniform(shape = (1, 64))
net(x1,x2)
```

Out

```
[[[0.10969]]]
```

NDArray 1x1x1

Visualization of Gluon Blocks

```
import mxnet as mx
from mxnet import nd, viz
from mxnet.gluon import nn, model_zoo
```

for visualize

pretrained model

visual geometric group Oxford

```
vgg11 = model_zoo.vision.vgg11(pretrained=True)
```

first way to visualize

```
print(vgg11)
```

It has two main blocks.

- features – has many layers
  - alternates
    - ConV2D
    - MaxPool2D
- Output
  - one layer

```
vgg11.summary(nd.ones((1, 3, 224, 224)))
```

It can print shapes of each layer when given dummy input of proper shape.

```
vgg11.summary(nd.ones((1,3,224,224)))
```

weights learned during training

Layer (type)	Output Shape	Param #
Input	(1, 3, 224, 224)	0
Conv2D-1	(1, 64, 224, 224)	1792
Activation-2	(1, 64, 224, 224)	0
MaxPool2D-3	(1, 64, 112, 112)	0
Conv2D-4	(1, 128, 112, 112)	73856
Activation-5	(1, 128, 112, 112)	0
MaxPool2D-6	(1, 128, 56, 56)	0
Conv2D-7	(1, 256, 56, 56)	295168
Activation-8	(1, 256, 56, 56)	0
Conv2D-9	(1, 256, 56, 56)	590080
Activation-10	(1, 256, 56, 56)	0
MaxPool2D-11	(1, 256, 28, 28)	0
Conv2D-12	(1, 512, 28, 28)	1180160
Activation-13	(1, 512, 28, 28)	0
Conv2D-14	(1, 512, 28, 28)	2359808
Activation-15	(1, 512, 28, 28)	0
MaxPool2D-16	(1, 512, 14, 14)	0
Conv2D-17	(1, 512, 14, 14)	2359808
Activation-18	(1, 512, 14, 14)	0
Conv2D-19	(1, 512, 14, 14)	2359808
Activation-20	(1, 512, 14, 14)	0
MaxPool2D-21	(1, 512, 7, 7)	0
Activation-22	(1, 4096)	0
Dense-23	(1, 4096)	102764544
Dropout-24	(1, 4096)	0
Activation-25	(1, 4096)	0
Dense-26	(1, 4096)	16781312
Dropout-27	(1, 4096)	0
Dense-28	(1, 1000)	4097000
VGG-29	(1, 1000)	0

Convolution increase

max pool reduce

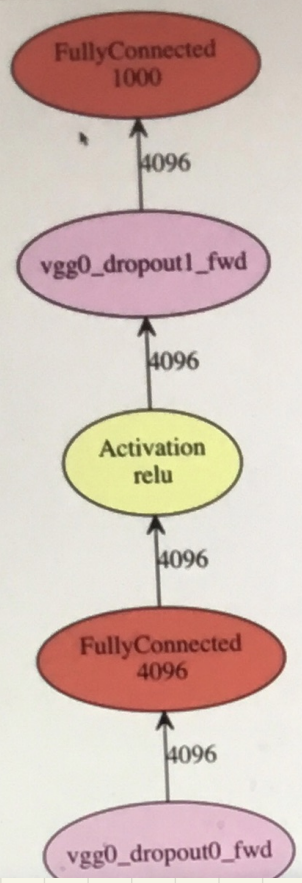
Final Dense layer is huge

```
Parameters in forward computation graph, duplicate included
   Total params: 132863336
   Trainable params: 132863336
   Non-trainable params: 0
```

File Edit View Insert Cell Kernel Widgets Help

In [6]:
```
viz.plot_network(vgg11(mx.sym.var('data')), shape={'data':(1,3,224,224)},node_attrs={"shape":"oval","fixedsize":"false"
```

Out[6]:



output

Input is way down here

```
In [7]: mobile_net = model_zoo.vision.mobilenet_v2_1_0(pretrained=True)

In [8]: mobile_net
```

← a lot more blocks than VGG

```
Out[8]: MobileNetV2(
          (features): HybridSequential(
            (0): Conv2D(3 -> 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
            (1): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=32)
            (2): RELU6(

            )
            (3): LinearBottleneck(
              (out): HybridSequential(
                (0): Conv2D(32 -> 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=32)
                (2): RELU6(

                )
                (3): Conv2D(1 -> 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
                (4): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=32)
                (5): RELU6(

                )
                (6): Conv2D(32 -> 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (7): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=16)
              )
            )
            (4): LinearBottleneck(
              (out): HybridSequential(
                (0): Conv2D(16 -> 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): BatchNorm(axis=1, eps=1e-05, momentum=0.9, fix_gamma=False, use_global_stats=False, in_channels=96)
                (2): RELU6(
```

when we print the summary, it does not have as many parameters so it will train faster.

```
mobile_net.hybridize()
mobile_net.(nd.ones((1,3,224,224)))
mobile_net.export('mobile_net')
```

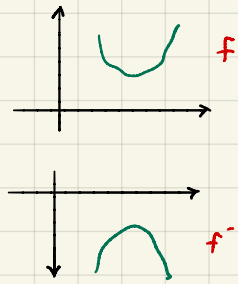
Let's visualize the computational graph w/

https://lutzroeder.github.io/netron

Metrics vs Losses

By convention we usually define objective functions so that low values are better than high values.

You can take any function which higher is better and map to new function.



$$f' = f$$

Lower is better ↑ higher is better

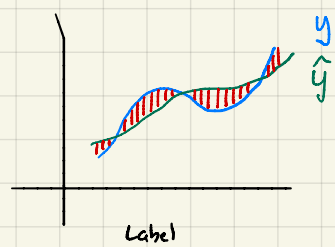
Because lower is better, this is called loss function or cursed function.

When trying to predict continuous numerical values such as regression, there is an objective function which is sum of squared errors.

Regression

Objective: Sum of square errors

$$f = \sum (y - \hat{y})^2$$



Classification

Objective: error rate

$$f = \frac{|y \neq \hat{y}|}{|y|}$$

Pred \ Label	1	0
1	TP	FP
0	FN	TN

MINIMIZE

Loss vs. Metric

Loss

objective minimized during training

Metric

Human readable assessment

Metrics for binary classification

Pred \ Label	1	0
1	TP	FP
0	FN	TN

$$\text{Metric} = \frac{\text{Numerator}}{\text{Denominator}}$$

MINIMIZE

use if mistakes with false positives are costly

$$\text{Precission} = \frac{\text{Correctly predicted TRUE}}{\text{All TRUE predictions}} = \frac{TP}{TP + FP}$$

Pred \ Label	1	0
1	TP	FP
0	FN	TN

use if missing positive example are costly

$$\text{Recall} = \frac{\text{Correctly predicted TRUE}}{\text{All TRUE examples}} = \frac{TP}{TP + FN}$$

Pred \ Label	1	0
1	TP	FP
0	FN	TN

$$\text{Accuracy} = \frac{TP + TN}{\text{Total numer of examples}}$$

Pred \ Label	1	0
1	TP	FP
0	FN	TN

"F1 Score" combines precission and recall.

It uses the "harmonic mean of precission & recall"

$$F1 = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Metrics for Regression

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{(\text{Predicted}_i - \text{Actual}_i)^2}{N}}$$

ROOT MEAN SQUARE ERRO

$$MAE = \sum_{i=1}^{N} \frac{|\text{Predicted}_i - \text{Actual}_i|}{N}$$

MAX ABSOLUTE ERROR

Metrics for Multi Class

```
import mxnet as mx
from mxnet as metric

accuracy = metric.Accuracy()
```

Correct Predictions

```
labels = nd.array([1,2,3])
predictions = nd.array([1,2,3])
accuracy.update(labels, predictions)
accuracy.get()
```

('accuracy', 1.0)

Erroneous Predictions

```
accuracy.reset()
predictions = nd.array([1,1,1])
accuracy.update(labels, predictions)
accuracy.get()
```

('accuracy', 0.333)

```
top_2_accuracy = metric.TopKAccuracy(top_k=2)
predictions = mx.nd.array([[.4, .5, .1],
                           [.4, .5, .1],
                           [0.4, .5, .1]])
labels = mx.nd.array([0,0,0])
accuracy.reset()
accuracy.update(labels, predictions)
accuracy.get()
```

Using top 2 classes highest probability is first class in all cases.

highest probability is for second class

```
('accuracy', 0.0)
top_2_accuracy.update(labels, predictions)
accuracy.get()
('top_k_accuracy_2', 1.0)
```

Metrics specific to certain tasks and domain

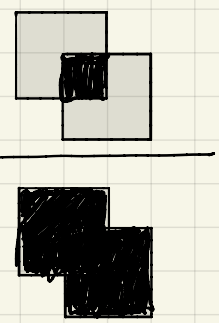
object detection

metric is precision at Intersection of union threshold

when we say "precision at 0.5 for an object task"

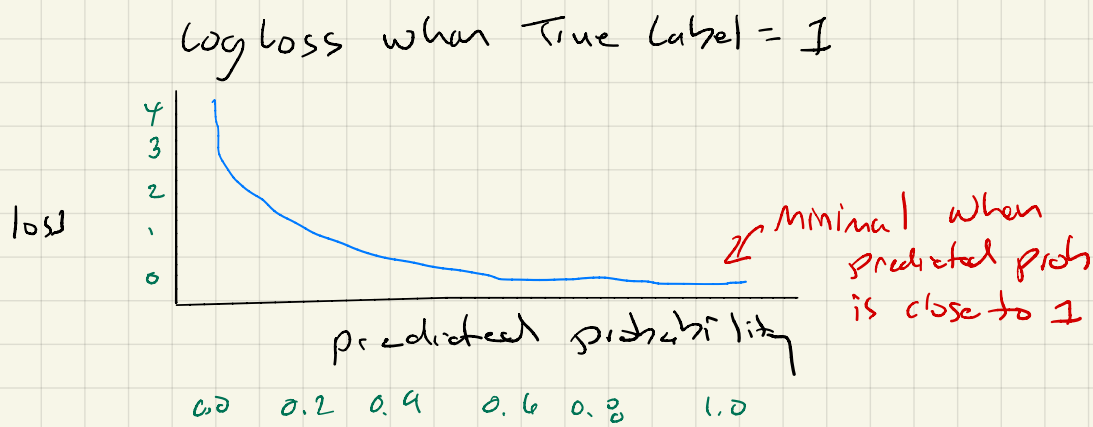
we mean that any correctly labeled class with more than 0.5 IOU intersection of overlap with a ground truth bounding box is considered a correct prediction.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} =$$



Losses

```
import mxnet as mx
from mxnet.gluon import loss
import numpy as np

x = np.arange(0.01, 1.00, 0.001)
plt.plot(x, -np.log(x))
plt.title("Log loss when True Label = 1")
plt.xlabel("predicted probability")
o = plt.ylabel("loss")
```



$$\text{softmax}(x_i) = \frac{e_i^x}{\sum_{j=1}^{N} e_j^x} \tag{6-1}$$

```
nd.array([[1,2,3,4,5]]).softmax().sum()
```

1.0

It's important to note that this loss function first normalizes the raw output of your network using the softmax activation.

Cross Entropy Loss Function for N Classes

$$loss = -\sum_{i=0}^{N} p_i \cdot \log(pred_i)$$

0 1 2

predictions = nd.array([[10, 20, 40, 20, 5]])

Highest probability

label = nd.array([1])

Predicted class is 1

loss_fn(predictions, label)

[20.]

label = nd.array([2])

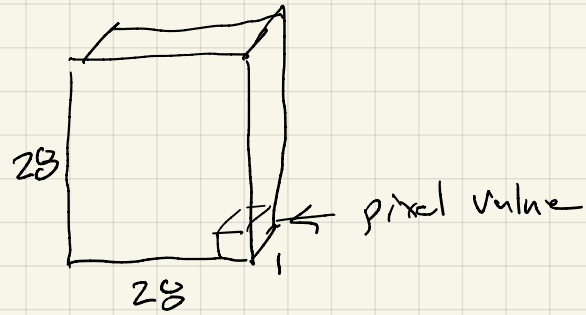
Predicted class is 2

loss_fn(predictions, label)

[0.]

Lab Assignment Notes

Image



Batch

1024 ×

N	C	H	W
(1024,	1,	28,	28)
0	1	2	3

Average Pixel

num_images = batch.shape[0] = 1024

sum = batch.sum(axis=0) = (1, 28, 28)

avg. pixel = sum / num_images

normalize calculate mean & SD

for each $\frac{\text{image} - \text{mean}}{\text{SD}}$

this z-score but they just
were image - mean

most values will be -1 & +1

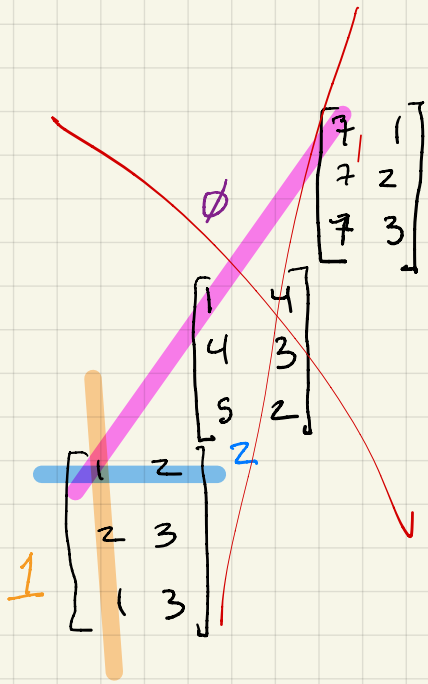
WRONG

testy = np.ndarray([[[1, 2], [2, 3], [1, 3]],
[1, 4], [4, 3], [5, 2]]
[7, 1], [7, 2], [7, 3]]])

testy.sum(axis=0)

[[9 7]
[13 8]
[13 8]]

3x2



testy.sum(axis=1)

[[4 8]
[10 9]
[21 6]]

testy.sum(axis=2)

[[3 5 4] 8? 9?
[5 7 7]
[8 9 10]]

4? 7 10 which is right

testy.sum(axis=[1,2])

if sums are done above this is valid

[12 19 27]

3	5	8
5	7	9
4	7	10
12	19	27

Correct

testy = mx.nd.array ([[[1, 2], [2, 3], [1, 3]],
[1, 4], [4, 3], [5, 2]]
[7, 1], [7, 2], [7, 3]]])

3x3x2

0 1 2

testy.sum(axis=0)

[[9 7]
[13 8]
[13 8]]

3x2

$$\begin{bmatrix} 1 & 3 \\ 5 & 2 \\ 7 & 3 \end{bmatrix}$$

3) Reduce axis 1

$$\begin{bmatrix} 2 & 3 \\ 4 & 3 \\ 7 & 2 \end{bmatrix}$$

1

OR

$$\begin{bmatrix} 4 & 8 \\ 10 & 9 \\ 21 & 6 \end{bmatrix}$$

Reduce axis 2

$$\begin{bmatrix} 12 \\ 19 \\ 27 \end{bmatrix}$$

Transpose

[12 19 27]

0

$$\begin{bmatrix} 1 & 2 \\ 1 & 4 \\ 7 & 1 \end{bmatrix}$$

2

testy.sum(axis=1)

[[4 8]
[10 9]
[21 6]]

testy.sum(axis=2)

[[3 5 4]
[5 7 7]
[8 9 10]]

2) this, transpose and sum

$$\begin{bmatrix} 3 & 5 & 4 \\ 5 & 7 & 7 \\ 8 & 9 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 & 8 \\ 5 & 7 & 9 \\ 4 & 7 & 10 \end{bmatrix}$$

↓ sum

[12 19 27]

testy.sum(axis=[1,2])

[12 19 27]

this can be done three ways

1) take axis 1 result

$$\begin{bmatrix} 4 & 8 \\ 10 & 9 \\ 21 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 10 & 21 \\ 8 & 9 & 6 \end{bmatrix}$$

transpose & sum

↓

[12 19 27]

Numpy Arrays

Goal is to mock a [1024, 3, 28, 28]

```
toty = mx.nd.array([[1,2],[3,4]])
testy.shape   (2,2)
```

2x2
(2,2)

```
testy   [[ 1,2 ]
         [ 3,4 ]]
```

```
testy = mx.nd.array(  [
                        [[1,2],[3,4]],
                        [[5,6],[7,8]],
                        [[9,10],[11,12]]]
                      ])
```

3x2x2
(3,2,2)

```
testy.shape   (3,2,2)
testy
```

$$
\begin{array}{l}
[[\boxed{\begin{matrix}[1 & 2] \\ [3 & 4]\end{matrix}}] \quad \} \; 2\times2 \\
[\boxed{\begin{matrix}[5 & 6] \\ [7 & 8]\end{matrix}}] \quad \} \; 2\times2 \\
[\boxed{\begin{matrix}[9 & 10] \\ [11 & 12]\end{matrix}}]] \quad \} \; 2\times2
\end{array}
\quad \Big\} \; 3\times1
$$

testy = mx.nd.array([

4x1

2x3

2x2

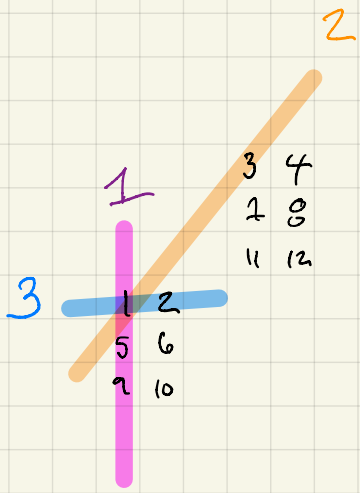
```
[
  [ [ [1, 2], [3, 4] ],
  [ [5, 6], [7, 8] ],
  [ [9, 10], [11, 12] ]
],
[
  [ [13, 14], [15, 16] ],
  [ [17, 18], [19, 20] ],
  [ [21, 22], [23, 24] ]
],
[
  [ [25, 26], [27, 28] ],
  [ [29, 30], [31, 32] ],
  [ [33, 34], [35, 36] ]
],
[
  [ [37, 38], [39, 40] ],
  [ [41, 42], [43, 44] ],
  [ [45, 46], [47, 48] ]
]
])
```

```
4 [[[[[ 1   2 ]
      [ 3   4 ]]  2

      [ 5   6 ]
      [ 7   8 ]]  2

      [ 9   10 ]
      [ 11  12 ]]]  3

      [ 13  14 ]
      [ 15  16 ]]  2

      [ 17  18 ]
      [ 19  20 ]]  2

      [ 21  22 ]
      [ 23  24 ]]]  3

3 [[[[ 25  26 ]
      [ 27  28 ]]  2

      [ 29  30 ]
      [ 31  32 ]]  2

      [ 33  34 ]
      [ 35  36 ]]]  3

3 [[[[ 37  38 ]
      [ 39  40 ]]  2

      [ 41  42 ]
      [ 43  44 ]]  2

      [ 45  46 ]
      [ 47  48 ]]]]  4
```

(4, 3, 2, 2)

axis ∅

```
4 [[[[[ 1   2 ],
      [ 3   4 ]], 2

      [ 5   6 ],
      [ 7   8 ]], 2

      [ 9  10 ],
      [ 11 12 ]]], 3

      [ 13  14 ]
      [ 15  16 ]]  2

      [ 17  18 ]
      [ 19  20 ]]  2

      [ 21  22 ]
      [ 23  24 ]]] 3
```




```
  1
 25
 13
 37
----
 16
  6
----
 76
```

```
3 [[[[ 25  26 ]
      [ 27  28 ]]  2

      [ 29  30 ]
      [ 31  32 ]]  2

      [ 33  34 ]
      [ 35  36 ]]]  3

3 [[[[ 37  38 ]
      [ 39  40 ]]  2

      [ 41  42 ]
      [ 43  44 ]]  2

      [ 45  46 ]
      [ 47  48 ]]]]  4
```

Spot check - Top Left

Sum(axis = 0) → (3, 2, 2)

```
[[[[ 76  80 ]   2x2
   [ 84  88 ]]]

  [[[ 92  96 ]
    [100 104 ]]]   3x1

  [[[108 112 ]
    [116 120 ]]]]
```

1 + 25 + 13 + 37 =

```
  1
 25
 13
 37
----
 16
  6
----
 76
```

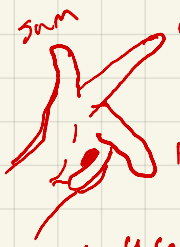
Spot check Top Right

2 + 14 + 26 + 38

```
  2
 14
 26
 38
----
 20
  6
----
 80
```

axis ∅ (4 elements in this axis)

```
4 [[[[[ 1   2 ]
      [ 3   4 ]]  2

      [ 5   6 ]
      [ 7   8 ]]  2

      [ 9  10 ]
      [ 11 12 ]]]  3
```

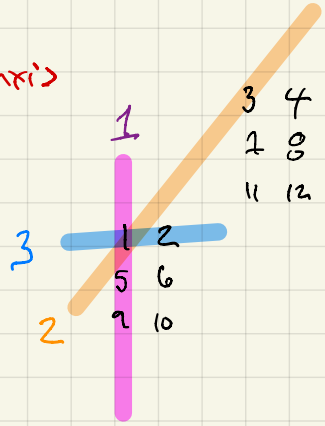


- use left hand
- put thumb along sum axis
- middle along other axis
- Rotate hand so thumb points to face and middle finger is closer

moves in these directions match result

```
[ 13  14]
[ 15  16]]  2

[ 17  18]
[ 19  20]]  2

[ 21  22]
[ 23  24]]]  3
```

01 just realize as you move min axis you move closer in Result.




```
3 [[[[ 25  26]
      [ 27  28]]  2

      [ 29  30]
      [ 31  32]]  2

      [ 33  34]
      [ 35  36]]]  3
```

```
3 [[[[ 37  38]
      [ 39  40]]  2

      [ 41  42]
      [ 43  44]]  2

      [ 45  46]
      [ 47  48]]]]  4
```

Sum(axis = 1) → (4, 2, 2)

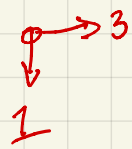
```
1   2
5   6
9   10
---
15  18

3   4
7   8
11  12
---
21  24
```

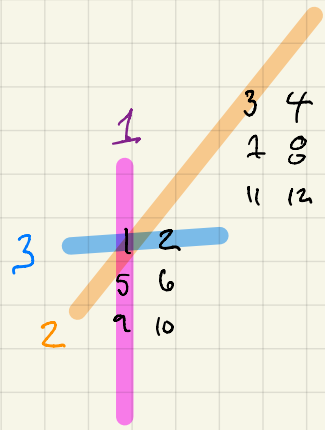
```
[[[[15  18]
   [21  24]]

[[[51  54]
   [57  60]]

[[[87  90]
   [93  96]]]

[[[123  126]
   [129  132]]]]
```

axis ∅ (4 elements in this axis)

```
4 [[[[ 1   2 ]
     [ 3   4 ]]  2

     [ 5   6 ]
     [ 7   8 ]]  2

     [ 9  10 ]
     [ 11 12 ]]]  3
```

```
[ 13  14 ]
[ 15  16 ]]  2

[ 17  18 ]
[ 19  20 ]]  2

[ 21  22 ]
[ 23  24 ]]]  3
```





sum along axis 2

→ Move along axis 3 Move Right in result

Move axis 1 moves down in result

```
3 [[[[ 25  26 ]
     [ 27  28 ]]  2

     [ 29  30 ]
     [ 31  32 ]]  2

     [ 33  34 ]
     [ 35  36 ]]]  3
```

```
3 [[[[ 37  38 ]
     [ 39  40 ]]  2

     [ 41  42 ]
     [ 43  44 ]]  2

     [ 45  46 ]
     [ 47  48 ]]]]  4
```

Sum(axis = 2) → (4, 3, 2)

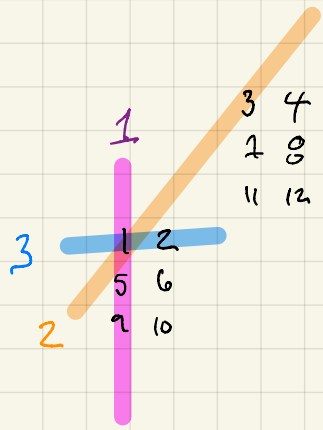
1	2	13	14	25	26	37	38
3	4	15	16	27	28	39	40
4	6	28	30	52	54	76	78
5	6	17	18	27	30	41	42
7	8	19	20	31	32	43	44
12	14	36	38	58	62	84	86
9	10	21	22	33	34	45	46
11	12	23	24	35	36	47	48
20	22	44	46	68	70	92	94
Set ∅		Set 1		Set 2		Set 3	

```
[[[ 4   6 ]
  [ 12  14 ]
  [ 20  22 ]]]   Set 0

[[[ 28  30 ]
  [ 36  38 ]
  [ 44  46 ]]]   Set 1

[[[ 52  54 ]
  [ 58  62 ]
  [ 68  70 ]]]   Set 2

[[[ 76  78 ]
  [ 84  86 ]
  [ 92  94 ]]]]   Set 3
```

axis ∅ (4 elements in this axis)

```
4 [[[[[ 1   2 ]
      [ 3   4 ]]  2

      [ 5   6 ]
      [ 7   8 ]]  2

      [ 9  10 ]
      [ 11 12 ]]]  3
```

```
[ 13  14 ]
[ 15  16 ]]  2

[ 17  18 ]
[ 19  20 ]]  2

[ 21  22 ]
[ 23  24 ]]]  3
```



→ move along axis 2
Moves in Result similar

→ axis = 3 is summed

↓ move along axis 1
Moves in Result similar

```
3 [[[[ 25  26 ]
      [ 27  28 ]]  2

      [ 29  30 ]
      [ 31  32 ]]  2

      [ 33  34 ]
      [ 35  36 ]]]  3
```

```
3 [[[[ 37  38 ]
      [ 39  40 ]]  2

      [ 41  42 ]
      [ 43  44 ]]  2

      [ 45  46 ]
      [ 47  48 ]]]]  4
```

Sum(axis = 3) → (4, 3, 2)

1	3	13	15	25	27	37	39	$\begin{bmatrix} 3 & 7 \\ 11 & 15 \\ 19 & 23 \end{bmatrix}$
2	4	14	16	26	28	38	40	
3	7	27	31	51	55	75	79	
5	7	17	19	29	31	41	43	$\begin{bmatrix} 27 & 31 \\ 35 & 39 \\ 44 & 47 \end{bmatrix}$
6	8	18	20	30	32	42	44	
11	15	35	39	59	63	83	87	
9	11	21	23	33	35	45	47	$\begin{bmatrix} 75 & 79 \\ 83 & 87 \\ 91 & 95 \end{bmatrix}$
10	12	23	24	34	36	46	48	
19	23	44	47	67	71	91	95	

Notes on How to build.

```
[[[[1, 2], [2, 3], [1, 3]],
  [1, 4], [4, 3], [5, 2]]
  [7, 1], [7, 2], [7, 3]]]
```

Commas move along existing axis to make new columns

BACK TO BACK closing braces move to next row

Three back to back closing move to next set

col col

Row

Set

$$\begin{bmatrix} 1 & 3 \\ 5 & 2 \\ 7 & 3 \end{bmatrix}$$

1

$$\begin{bmatrix} 2 & 3 \\ 4 & 3 \\ 7 & 2 \end{bmatrix}$$

ϕ

2

$$\begin{bmatrix} 1 & 2 \\ 1 & 4 \\ 7 & 1 \end{bmatrix}$$

```
4 [[[[[ 1   2 ]
      [ 3   4 ]] 2

      [ 5   6 ]
      [ 7   8 ]] 2

      [ 9   10 ]
      [ 11  12 ]]] 3
```

3	4
7	8
11	12

1	2
5	6
9	10

Transforms.Compose

from mxnet.gluon.data.vision

sequentially compose multiple transforms in list

```
channel_std = 0.31
transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(channel_average, channel_std)
    ])
```

channel_average = get_channel_average_from_batch(batch)

lots

Gluon Fundamentals Continued

what we will learn

Splitting Iterative Training process into three steps

- forward
- backward
- optimize

Lesson 1 Automatic Differentiation
back propagation

Lesson 2 Optimizers

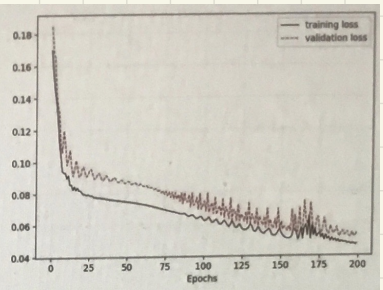
$$W_{i+1} = W_i + lr \cdot -\nabla_{W_i} \text{loss}$$

Stochastic Gradient Descent SGD

Lesson 3 Data

Lesson 4 Training

```
for data, label in train_data:
    with autograd.record():
        output = model(data)
        loss = loss_fn(output, label)
    loss.backward()
    trainer.step(batch_size)
```



Source: loss functions

aws

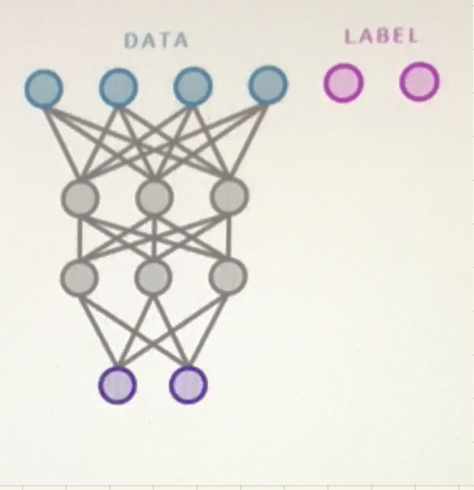
Automatic Differentiation

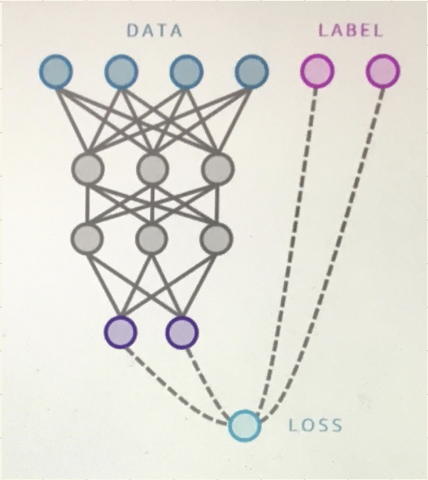
Neural Network Training

Three steps

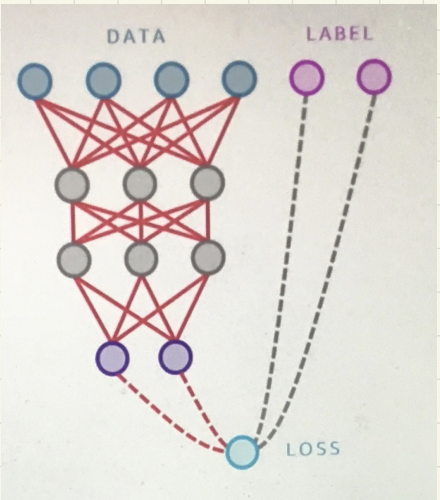
- Forward - pass data through model - model prediction
- Backward
- Optimize

Forward

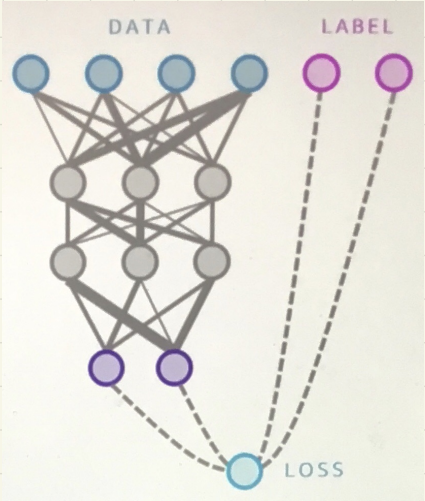




Backward



Optimize

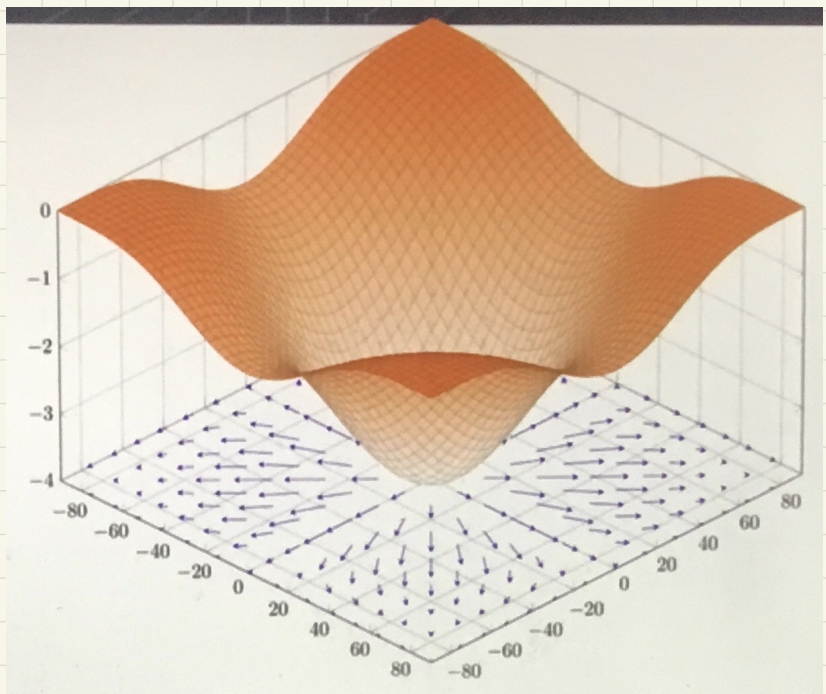


Gradients

The gradient of a function is the derivative of the function at a point

gradient is $f'(x)_{x=a}$

The gradient gives the direction of steepest increase of the function



Gradient descent follows the negative gradient to minimize a function

Automatic Differentiation

Computationally evaluate gradients of arbitrary functions

How?

- Record operations and build a computational graph
- Repeatedly apply chain rule to compute gradients

Backpropagation

Use the gradients above

Efficient algorithm to train neural networks using automatic differentiation

For a model with a single layer with parameters

- W
- b

$O =$

Backpropagation

Use the gradients above

Efficient algorithm to train neural networks using automatic differentiation

For a model with a single layer with parameters

- W
- b

Forward

outputs of nn

weight

inputs

bias

$$o = \sigma(Wx + b)$$

actual label

$$l = f(o, y)$$

output

Backward

$$\nabla_o l = f'(o, y)$$

$$\nabla_\sigma l = \nabla_o l \cdot \sigma'(Wx + b)$$

$$\nabla_W l = \nabla_\sigma l \cdot x$$

$$\nabla_b l = \nabla_\sigma l$$

Recap

Neural Network Training

- Forward
- Backward
- Optimize

Gradients and Automatic Differentiation

- Compute loss function
- Automatically compute gradients
- Update with backpropagation

Automatic Differentiation - MXNet AutoGrad

```
importing mxnet as mx

from mxnet import nd
from mxnet import autograd
```

differentiate $f(x) = 2x^2$ with respect to parameter x

```
x = nd.array([[1,2],[3,4]])        2x2 array
x.attach_grad()
```

$y = f(x)$

```
def f(x):
    return 2*x**2

with autograd.record():
    y = f(x)

x, y

(
 [[1  2]
  [3  4]]
 <NDArray 2x2 @cpu(0)>,

 [[2  8]
  [18  32]]
 <NDArray 2x2 @cpu(0)>)
```

Backward Propagation of y

```
y.backward()
```

Backward Propagation of y

y.backward()

$y = 2x^2$

$\frac{dy}{dx} = 4x$

x, x.grad

```
(
 [[ 1  2 ]
  [ 3  4 ]]
 <NDArray 2x2 @cpu(0)>,

 [[ 4  8 ]
  [ 12  16 ]]
 <NDArray 2x2 @cpu(0)>)
```

Dynamic Functions

Use Python control flows

f(x)

- Take a vector x of two random numbers in [-1, 1]
- X is multiplied by 2, until its norm (length) reach 1000
- IF x's sum is positive, return 1st element otherwise 2nd

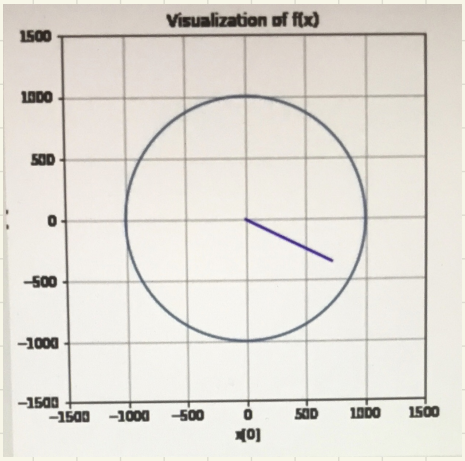
Dynamic Functions

Use Python control flows

f(x)

- Take a vector x of two random numbers in [-1, 1]
- X is multiplied by 2, until its norm (length) reach 1000
- IF X's sum is positive, return 1st element otherwise 2nd

```
def f(x):
    x = x*2
    while x.norm().asscalar() < 1000:
        x = x*2
    # if sum positive
    # pick first
    if x.sum().asscalar() >= 0:
        y = x[0]
    # else pick 2nd
    else:
        y = x[1]
    return y
```



Vector increases until radius of 1000 reached

```
x = nd.random.uniform(-1, 1, shape=2)
x
```

[0.0976 , 0.1857]

```
x.attach_grad()
with autograd.record():
    y = -f(x)
y.backward()
```

$y = k \cdot x[0]$ $\qquad \frac{dy}{dx} = \begin{vmatrix} 0 \\ k \end{vmatrix}$

or

$y = k \cdot x[1]$ $\qquad \frac{dy}{dx} = \begin{vmatrix} k \\ 0 \end{vmatrix}$

Double factor, n is number of doubles

with $k = 2^n$ where n is the number of times x was multiplied by 2

```
x
```

[0.0976 , 0.18569]

```
x.grad
```

[8192 , 0]

Quiz 1

which of the following is NOT true?

F • we compute gradients using forward mode automatic differentiation. ← we don't automatically diff.

T • Direction of gradient denotes the direction of steepest ascent for function

T • At minima or maxima of diff function the value of gradient is Ø.

T • Gradient descent is an iterative process for approximately finding the minimum of a loss function.

which code block will print the derivative of x with respect to y?

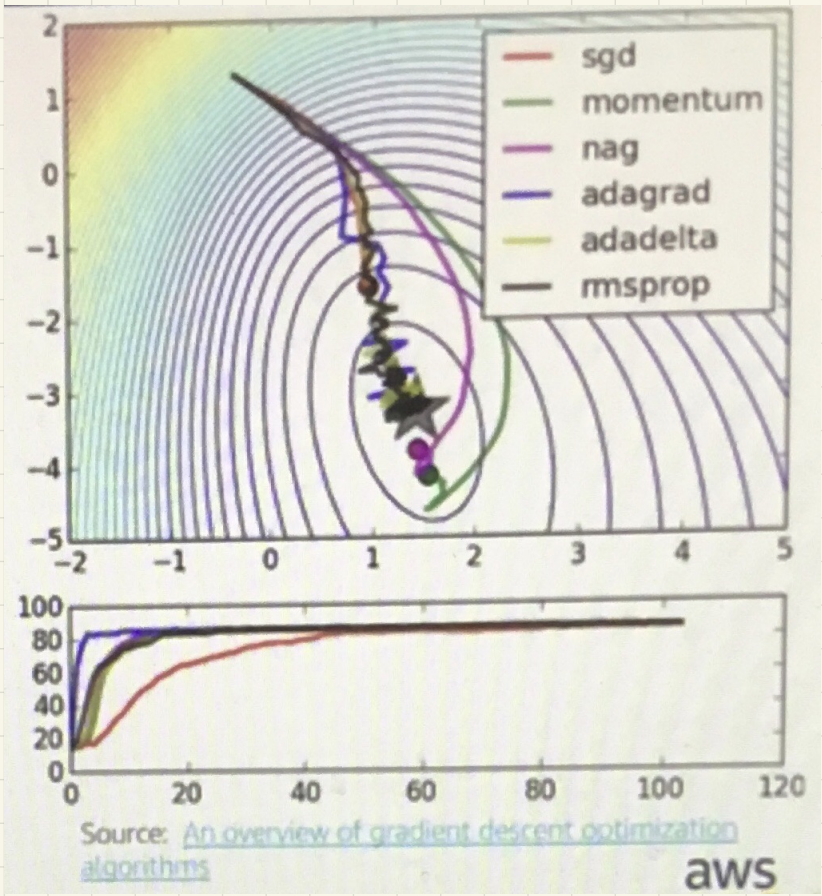
```
y. = data
y.attach_grad()
def f(y):
    return some form of y which is the derivative.
    if y=x^2, he returned 2x
with autograd.record():
    x = f(y)
x, y
```

then he does x.backward()
y.grad()

MxNet Optimizers

An optimizer updates the model parameters to minimize the loss function.

It uses the gradient of the loss wrt to parameter values.



Stochastic Gradient Descent SGD

- Use minibatch of data for forward and backward passes.
  subset of batch

- Gradient is computed on minibatch loss wrt current model parameters.

- Parameters are updated according to:

$$w_{i+1} = w_i + lr \cdot -\nabla_{w_i} \text{loss}$$

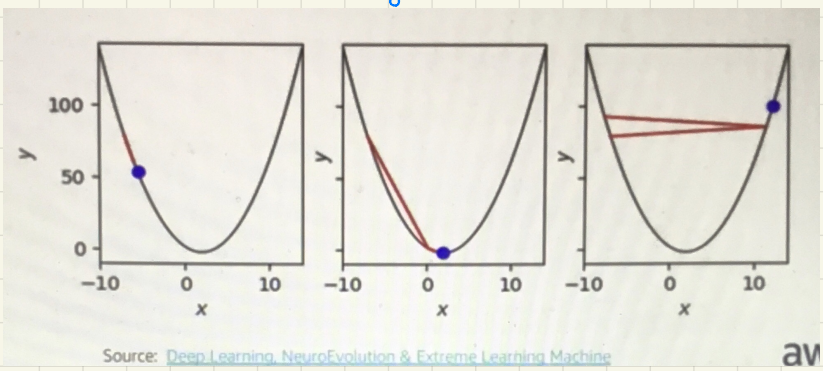
learning rate

Learning Rates

- Learning rate controls how big of a step the optimizer takes.

- Learning rate significantly impacts the speed on convergence

to slow good to fast

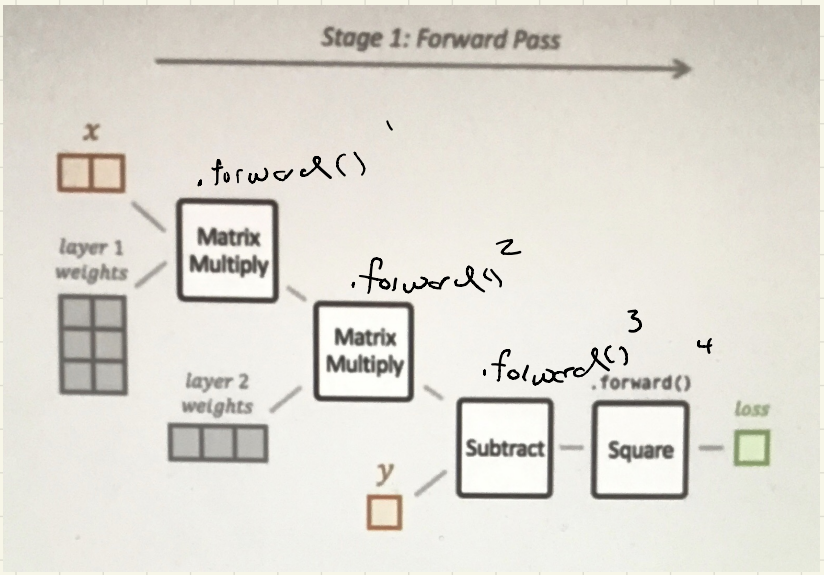


- Learning rates can be
  - Scheduled - as training steps increase, make learning rate smaller since model has learned some weights and they only need to be adjusted slightly
  - Adaptive - Each parameter gets its own learning rate.

Optimizer Recap

- Optimizers update model parameters to minimize loss function
- Optimizers use the gradient of the loss wrt model parameters
- Stochastic Gradient Descent (SGD) uses minibatch loss to perform the gradient update
- Learning rate is a hyperparameter that determines the size of the update step

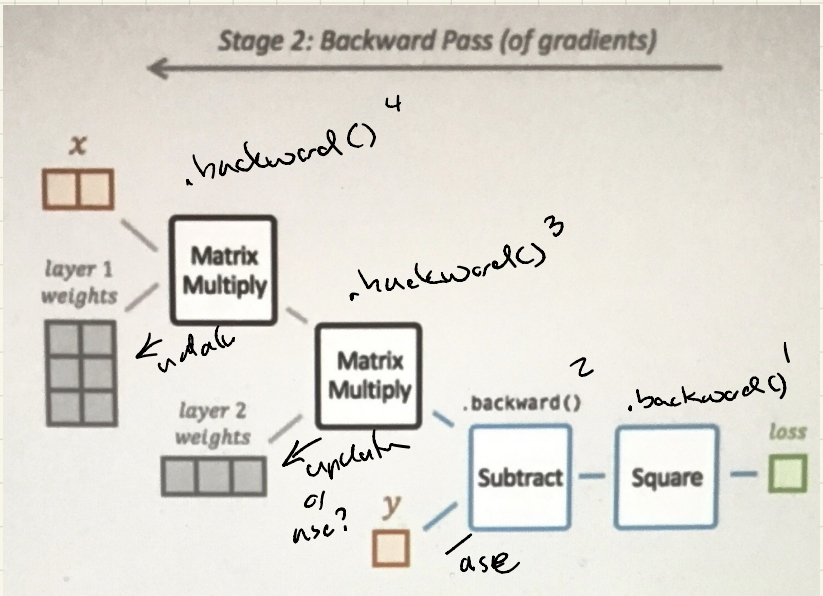
Optimus- gluon Trainers



it model

called not

you can call net.forward() or simply net



Usual imports

```
from mxnet import nd, autograd, optimizer, gluon

net = gluon.nn.Dense()
net.initialize()

batch_size = 8
x = nd.random.uniform(shape=(batch_size, 4))
y = nd.random.uniform(shape=(batch_size,))

loss = gluon.loss.L2Loss()
def forward_backward():
    with autograd.record():
        l = loss(net(x), y)
    l.backward()

forward_backward()

trainer = gluon.Trainer(net.collect_params(),
                        optimizer='sgd',
                        optimizer_params={'learning_rate':1})

curr_weight = net.weight.data().copy()
print(curr_weight)

Trainer step

trainer.step(batch_size)
print(net.weight.data())
```

forward

backward

results differ

Result of SGD Update

```
print(cur_weight - net.weight.grad() * 1 / batch_size)
```

values are identical to trainer.step results

Using Optimizer Instance

```
optim = optimizer.Adam(learning_rate=1)
trainer = gluon.Trainer(net.collect_params(), optim)

forward_backward()
trainer.step(batch_size)
net.weight.data()
```

Changing the Learning Rate

```
trainer.learning_rate
trainer.set_learning_rate(0.1)
trainer.learning_rate
```

Lesson 2 Quiz

1) which of the following statements is true?

F The learning rate does not significantly affect the speed of convergence of the stochastic gradient decent.

✓ T optimizers are implementations of general purpose gradient based algorithms for function minimization.

F An optimizers is always gauranteed to converge to the global minimum for any function.

F Using the same hyperparameters always converges to the same solution.

2) which of the following is needed to instantiate a gluon trainer?

- o Model parameters
- o Optimizer algorithm
- o optimizer hyperparameters
- ✓ → o All of the above

3) which of the following code blocks need to be adjusted so that when executed sequentially a single weight update step of stochastic gradient descent is done correctly?

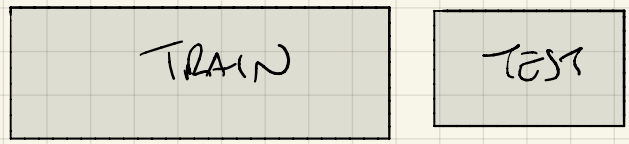
? ✗ o
```
trainer = gluon.Trainer(net.collect_params(),
                        adam,
                        optimizer_params={'lr':0.5})
```

labels = y ok? OK o
```
with autograd.record()
    loss = loss_fn(net(data), labels)
```

not sure why OK o
```
autograd.backward(loss)
```

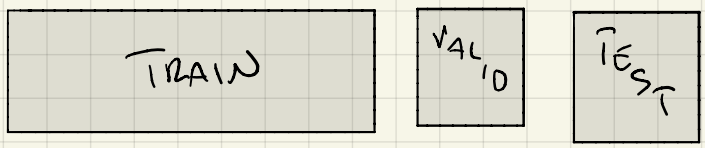
OK o
```
trainer.step(batch_size)
```

Lesson 3 - Datasets & Data Loaders
Data in Machine Learning



HOLD OUT

DATA PARTITIONING



Train, Validate, Test

Candidate models

Final model test

DATASET VS DATA LOADER

The Gluon data-set class

is the abstraction for the generic dataset

The Gluon data-loader class

is the abstraction responsible collecting data samples generated by the datasets into mini-batches and feeding them to the network during training.

The data loader allows us to train our models on subsets of the data. The datasets are to huge to do otherwise. The data loader provides an efficient asynchronous batch iterator over our dataset.

Gluon Datasets

```
import mxnet as mx

from mxnet import gluon, nd
from matplotlib.pyplot import imshow
%matplotlib inline


mx.random.seed(42)
X = mx.random.uniform(shape=(10,3))
y = mx.random.uniform(shape=(10,1))
```

Once we have our data stored as ndarrays, we can simply use gluon.data.dataset.ArrayDataset to create the gluon dataset

```
dataset = gluon.data.dataset.ArrayDataset(X,y)
```

We can access data at a particular index in the Dataset and it returns the data sample.

```
sample = dataset[4]
sample

(
  [0.74707  0.3746  0.4636],
  [0.3544])
```

We get a tuple of a data sample and its corresponding label

Preloaded Datasets

Some datasets like mnist & cifar10 are quite common benchmark datasets in the machine learning community.

These are already preloaded in gluon and can be created by calling

```
train_dataset = gluon.data.vision.datasets.MNIST(train=True)
valid_dataset = gluon.data.vision.datasets.MNIST(train=False)
```

We can visualize the data from the mnist data set

```
sample_image = train_dataset[19][0]
print(sample_image.shape)
imshow(sample_image[:,:,0].asnumpy(), cmap='gray')
```

(28, 28, 1)

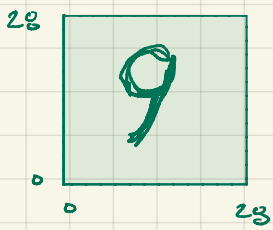


Image Folder Datasets

A dataset for loading image files stored in a folder structure

```
# image.dataset =
    gluon.data.vision.datasets.ImageFolderDataset('path_to_folder')
```

Custom Datasets

Extend the gluon datasets & dataloader class to implement custom handling for your data

```
class MyCustomDataset(gluon.data.Dataset):
    def __init__(self, dict_data):
        self.dict_data = dict_data
        self.dict_keys = list(dict_data.keys())

    def __getitem__(self, idx):
        return self.dict_data[self.dict_keys[idx]]

    def __len__(self):
        return len(self.dict_data)
```

ex

```
dataset = MyCustomDataset({'a': mx.nd.array(1),
                           'b': mx.nd.array(2),
                           'c': mx.nd.array(3,75)})

dataset[2]
```

3.6

Gluon Transformations

```
import mxnet as mx
from mxnet import gluon, nd

def transform_fn(data, label):
    data = data.astype('float32')/255
    return data, label

train_dataset = gluon.data.vision.datasets.MNIST(train=True).transform(transform_fn)
valid_dataset = gluon.data.vision.datasets.MNIST(train=False).transform(transform_fn)

sample_image = train_dataset[19][0]
nd.max(sample_image)
```

1

Vision Transforms

gluon.data.vision.transforms has implemented transformation functions that are common in computer vision

```
from mxnet.gluon.data.vision import transforms
```

ToTensor

Converts an image NDArray of shape (H×W×C) in the range [0, 255] to a float32 tensor NDArray of shape (C×H×W) in the range [0, 1)

```
train_dataset = gluon.data.vision.datasets.MNIST(train=True)
train_dataset[19][0].shape
```

(28, 28, 1)

```
to_tensor = transform.ToTensor()
train_dataset = train_dataset.transform_first(to_tensor)
train_dataset[19][0].shape
```

not to label

(1, 28, 28)

Normalize

Normalize a tensor of shape (C×H×W) with mean & SD

Each sample subtract

DIVIDE

```
mean, std = (0.1307,), (0.3081,)
normalize = transform.Normalize(mean, std)
train_dataset = train_dataset.transform_first(normalize)
```

Compose

sequentially compose multiple transforms

```
transform_fn = transforms.Compose([transpose.ToTensor(),
                                   transforms.Normalize(mean, std)])

train_dataset = gluon.data.vision.MNIST(train=True).transform(
                                   transform_fn)
```

Data Augmentation via Transforms

Some other transforms provided by Gluon include:

- transforms.Resize
- transforms.CenterCrop
- transforms.RandomResizedCrop
- transforms.RandomFlipLeftRight
- transforms.RandomBrightness

and many more

Gluon Data loaders

The dataloader is essentially an iterator that goes through a dataset, minibatch by minibatch until all datasamples have been used during training, signaling the end of one epoch.

Batch Usage

```
import mxnet as mx
from mxnet import gluon, nd
from matplotlib.pyplot import imshow
%matplotlib inline
```

Dataset

Using the toy dataset filled with draws from a random uniform distribution

```
mx.random.seed(42)
X = mx.random.uniform(shape=(10,3))
y = mx.random.uniform(shape=(10,1))
dataset = gluon.data.dataset.ArrayDataset(X,y)
```

Data Loader

A dataloader is used to create mini-batches of samples from a Dataset & provides a convenient iterator interface for looping through these batches.

```
data_loader = gluon.data.DataLoader(dataset, batch_size=5, last_batch='keep')
```

also 'discard'

```
for X_batch, y_batch in data_loader:
    print("X_batch has shape {}".format(X_batch.shape))
    print("y_batch has shape {}".format(y_batch.shape))
```

Parallel Data Loading

Dataloader allows loading data in parallel for maximum performance.

```
from multiprocessing import cpu_count
CPU_COUNT = cpu_count()

data_loader = gluon.data.DataLoader(dataset, batch_size=5, num_workers=CPU_COUNT):
for X_batch, y_batch in data_loader:
    print("X_batch has shape {}".format(X_batch.shape))
    print("y_batch has shape {}".format(y_batch.shape))
```

Lazy Transformations

We can lazily apply transformations on the dataset when creating the dataloader and set the dataset to be shuffled,

```
from mxnet.gluon.data.vision import transforms
train_dataset = gluon.data.vision.datasets.MNIST(train=True)
transform = transforms.Compose(
            [transforms.RandomFlipTopBottom(),
             transforms.RandomFlipLeftRight()])

train_dataloader = gluon.data.DataLoader(
            train_dataset.transform_first(transform),
            batch_size = 5,
            shuffle = True)

for X_batch, y_batch in train_dataloader:
        imshow(X_batch[3,:,:,0].asnumpy(),
               cmap='gray')
        break
```

Neural Network Training & Evaluation

Neural Network Training

```
from mxnet import nd, gluon, init, autograd, metric
from mxnet.gluon import nn
from mxnet.gluon.data.vision import datasets, transforms
import matplotlib.pyplot as plt
from time import time
```

Get Data

```
mnist_train = datasets.FashionMNIST(train=True)
X, y = mnist_train[0]
print('X shape: %s dtype: %s ' % (X.shape, X.dtype))
print('Number of images: %d' % len(mnist_train))

X, y = mnist_train[0:6]
plot_images(X, y)
```

Transform dataset

- channel first, float32
- normalize

```
transformer = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(0.13, 0.31)])

mnist_train = mnist_train.transform_first(transformer)
```

Data Loading

```
batch_size = 256
train_data = gluon.data.DataLoader(mnist_train,
                                   batch_size=batch_size,
                                   shuffle=True
                                   num_workers=4)


for data, label in train_data:
    print(data.shape, label.shape)
    break
```

Define the Model

```
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())
```

Loss

softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()

Metric: Accuracy

train_acc = metric.Accuracy()

Optimization

trainer = gluon.Trainer(net.collect_params(),
optimizer="sgd",
optimizer_params={'learning_rate': 0.1})

Training Loop

```
for epoch in range(10):
    train_loss = 0.
    tic = time()
    for data, label in train_data:
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()

        trainer.step(batch_size)

        train_loss += loss.mean().asscalar()
        train_acc.update(label, output)


    print("Epoch[%d] Loss:%.3f Acc:%.3f Perf: %.1f img/sec"%(
        epoch, train_loss/len(train_data),
        train_acc.get()[1],
        len(mnist_train)/(time()-tic)))

net.save_parameters("trained_net.params")
```

Neural Network Evaluation

evaluate a pre-trained model

```
from mxnet import nd, gluon, metric

from mxnet.gluon import nn
from mxnet.gluon.data.vision import datasets, transforms

import matplotlib.pyplot as plt
```

Build the model

```
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)
    )
```

Load the parameters

net.load_parameters('trained_net.params')

Predict

Validation Dataset

mnist-valid = datasets.FashionMNIST(train=False)

Data transformation

```
transform_fn = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(0.13, 0.31)])
```

Predict first 6 images

```
In [6]: preds = []
        for idx in range(6):
            image, label = mnist_valid[idx]
            image = transform_fn(image).expand_dims(axis=0)
            pred = net(image).argmax(axis=1)
            preds.append(pred.astype('int32').asscalar())
```

plot_images(images, labels, preds)

Validation DataLoader

Validation DataLoader

```
In [9]: batch_size = 256
        valid_data = gluon.data.DataLoader(
            mnist_valid.transform_first(transform_fn),
            batch_size=batch_size, num_workers=4)
```

Accuracy

valid_acc = metric.Accuracy()

Validation loop

```
for data, label in valid_data:
    output = net(data)
    valid_acc.update(label, output)

print("Validation Acc: %.3f "%(valid_acc.get()[1]))
```

```
Validation Acc: 0.899
```