BDAPro
@DIMA

Benchmarking MXNet and TensorFlow

Presented by:

- **Matteo Paltenghi**
- **Kaoutar Chennaf**

Mentor: **Behrouz Derakhshan**

Agenda

BDAPro
@DIMA

1. Intro To Deep Learning (DL)
2. Mxnet And Tensorflow: History And Major Features
3. Api Architecture
4. Benchmark - Fundamental DL Operations
5. Benchmark - Complete Neural Network: Lenet
6. Summary And Future Work

1. Introduction to Deep Learning

Intro to Neural Network

BDAPro
@DIMA

Perceptron Model

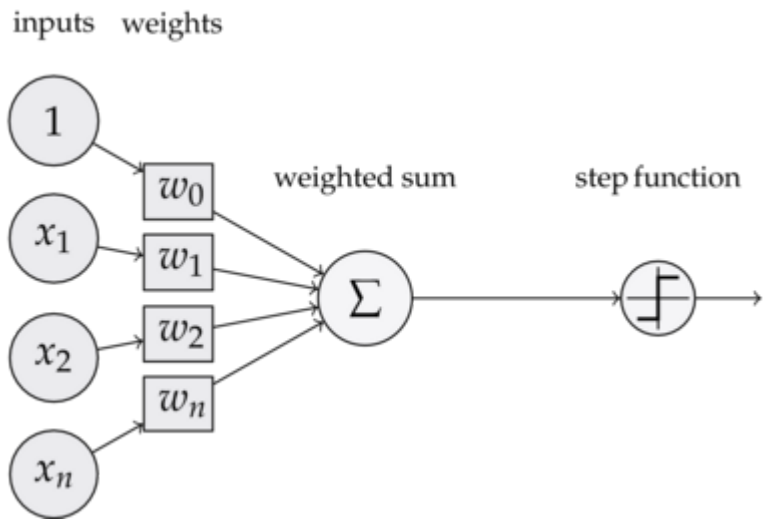


Image source: https://blog.dbrgn.ch/2013/3/26/perceptrons-in-python/

Multilayer Perceptron

(Feed Forward Neural Network = Signal flow only in one direction)
(Deep Learning = when we have more than one hidden layer)

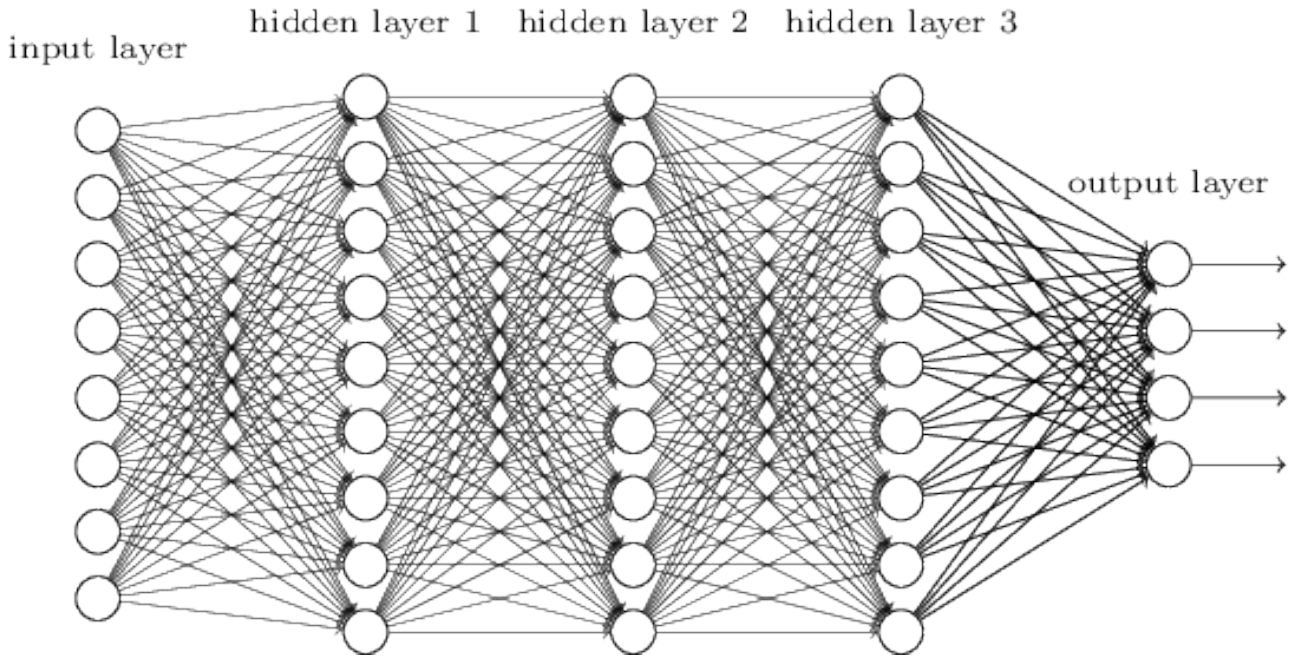
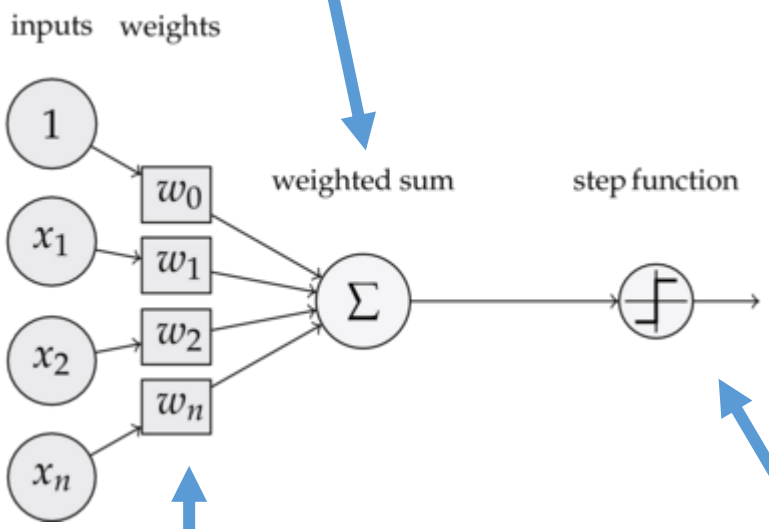


Image source: http://neuralnetworksanddeeplearning.com/chap6.html

Fundamental Operations

BDAPro
@DIMA

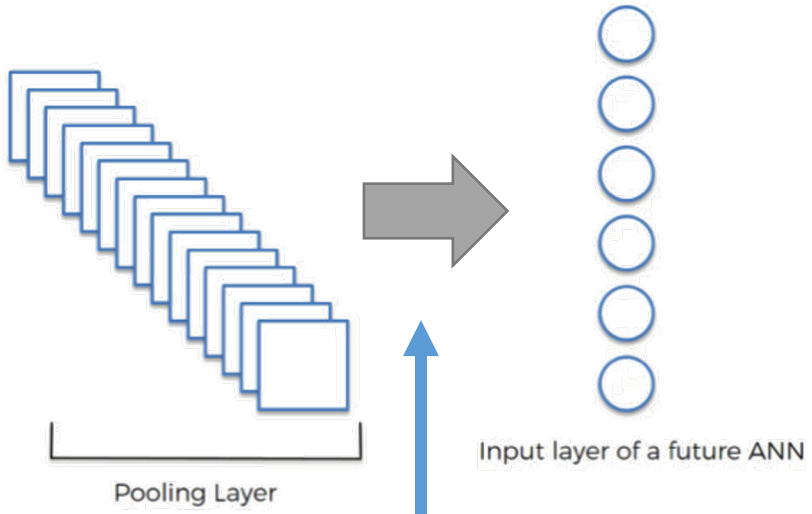
2. Dot product | 3. Matrix multiplication



1. Vector creation

4. Activation function

6. Vector Normalization



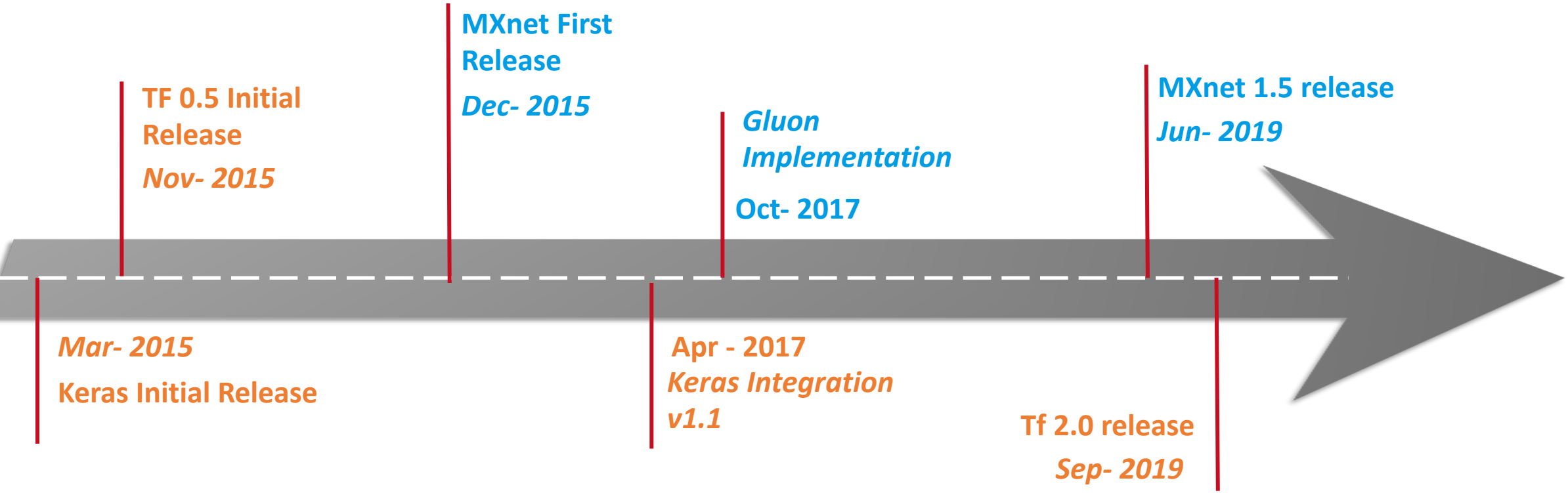
5. Flattening

Image source: https://blog.dbrgn.ch/2013/3/26/perceptrons-in-python/

Image source: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

2. History and Major Differences

System	Core Lang	Bindings Langs	Devices (beyond CPU)	Distributed	Imperative	Declarative
Tensorflow	C++	Python	GPU/**TPU**/Mobile	✓	✓ (with TF 2.0)	✓
MXNet	C++	Python/R/Julia/Go	GPU/Mobile	✓	✓	✓



Who is using

Image source: https://www.hqew.net/news/news-36655

Who is using mxnet

BDAPro
@DIMA

ABEJA, Inc.

Red Hat BIDS

BOSCH

smartbooks

Avito

Hewlett Packard Enterprise

中国移动通信
CHINA MOBILE

Continental

aws

Source: https://enlyft.com/tech/products/mxnet

Major Pros

BDAPro
@DIMA

Tensorflow

- It has a lot of documentation and guidelines;
- It offers monitoring for training processes of the models and visualization (**Tensorboard**);
- It's backed by a **large community** of devs and tech companies;
- It supports **TPU** when running on Google Cloud ServiceTensorflow Lite enables on-device inference with low latency for mobile devices;
- **Tensorflow JS** - enables deploying models in JavaScript environments, both frontend and Node.js backend.
    - TensorFlow.js also supports defining models in JavaScript and training them directly in the browser using a Keras-like API.

Mxnet

- It's quite **fast, flexible, and efficient** in terms of running DL algorithms;
- It features **advanced GPU support**, including multiple GPU mode;
- It has a **high-performance imperative API**;
- It's highly scalable;
- It provides rich support for **many programming languages**, such as Python, R, Scala, Javascript, and C++, among others;

Major Cons

BDAPro
@DIMA

TensorFlow

- It struggles with **poor results for speed** in benchmark tests compared with, for example, CNTK and MXNet,
- There is also one significant limitation: the **only fully supported language is Python**.

MXnet

- It has a much smaller community behind it compared with Tensorflow;
- It's **not so popular** among the research community.

Changes in TensorFlow

BDAPro
@DIMA

It brings us a bunch of exciting features, such as:

- It is possible to use **Keras inside TensorFlow**. It ensures that new Machine Learning models can be built with ease.
- Supports debugging your graphs and networks - TensorFlow 2.0 runs with eager execution by default for ease of use and smooth debugging. (**Imperative style supported**)
- **Robust model deployment** in production on any platform.
- Powerful experimentation for research.
- **Simplifying the API** by cleaning up deprecated APIs and reducing duplication.

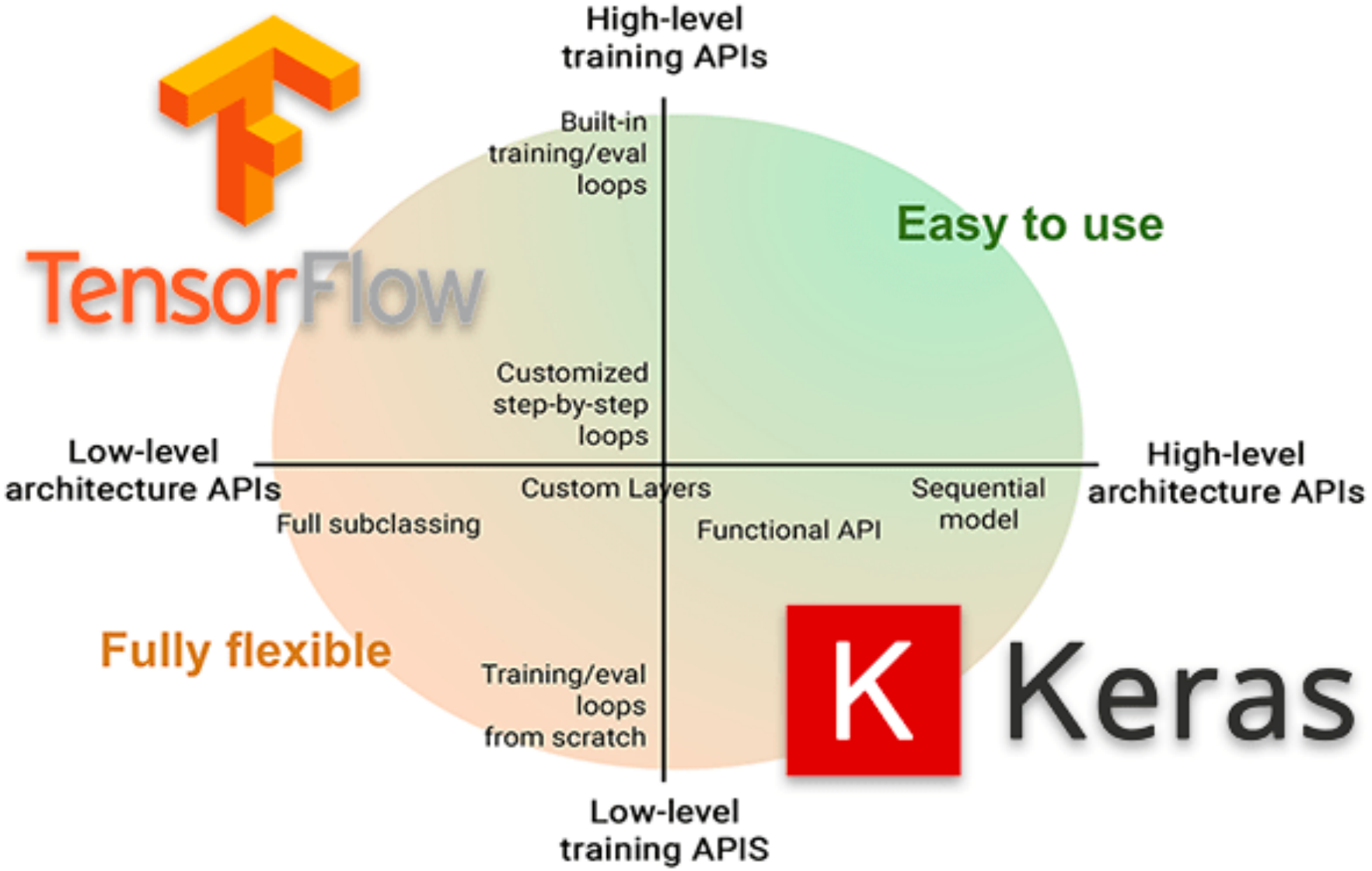


Image source: https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/

3. API Architectures

Imperative

```
# Imperative
import numpy as np

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

(+) Tend to be More Flexible

(+) Easy to try New Ideas (for Researches)

(+) Easy to specify Arbitrary Control Flow

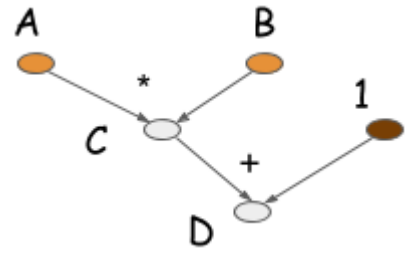
(-) Debugging happens during execution

(-) It can be More Difficult to Reuse

(-) Difficult to inspect, copy, or clone

Symbolic

```
# Symbolic
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)
```



(+) Tend to be More Efficient

(+) Easy to inspect

(+) Run Extensive checks before Execution

(+) Easier to copy or clone

(+) Fit Most use cases

(-) Few Special use cases do not fit

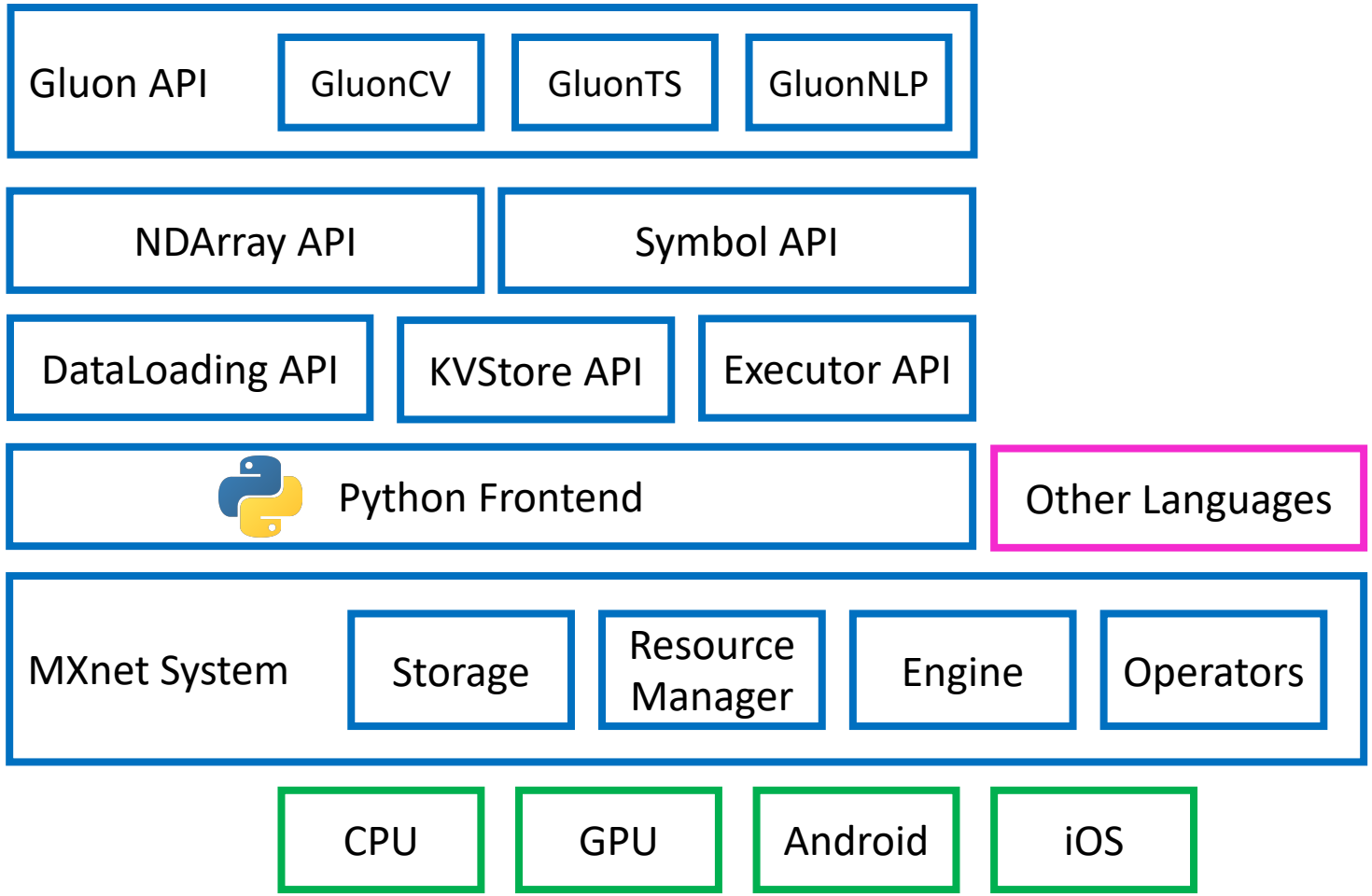
There Is No Strict Boundary

MXNet Documentation : “it's possible to make an imperative program more like a traditional symbolic.” program or vice versa.

TensorFlow Blog: “The two styles are fully interoperable as well, so you can mix and match (for example, you can nest one model type in another).”

Inventor of Keras (now part of TensorFlow 2.0)

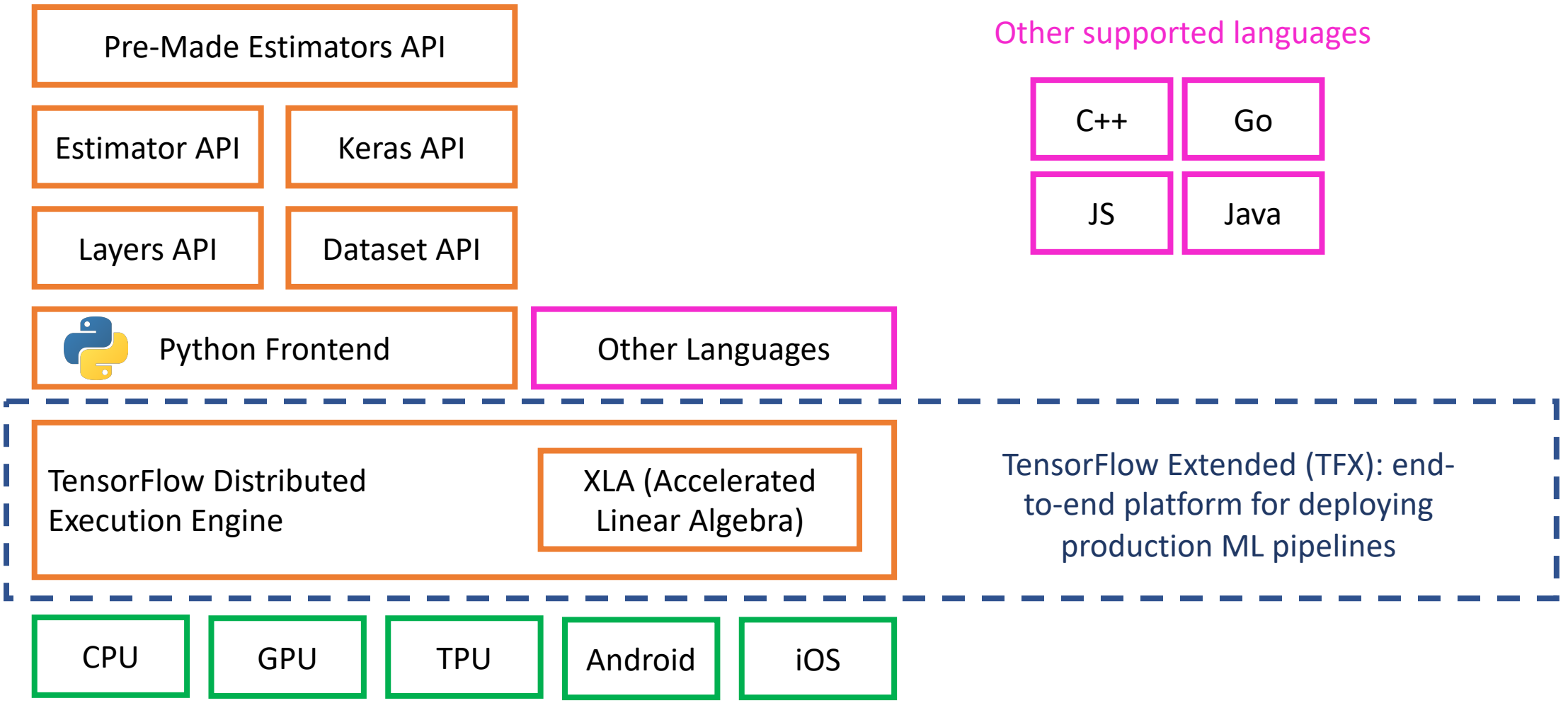
MXnet API Architecture



Other supported languages

C++ | R
Scala | Java
Perl | Julia
Clojure

TensorFlow API Architecture



Fundamental Operations

BDAPro
@DIMA

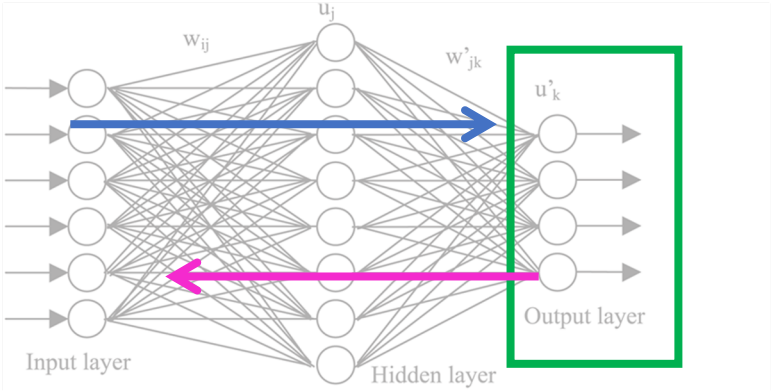
Operation	MXNet	TensorFlow
1. Vector creation	mx.ndarray.from_numpy(np_vector)	tf.convert_to_tensor(np_vector)
2. Dot Product	mx.ndarray.dot(first, second)	tf.tensordot(first, tf.transpose(second), axes=1)
3. Matrix Multiplication	mx.nd.linalg.gemm2(first, second)	tf.tensordot(first, second, axes = [[1], [0]])
4. Activation Function	mx.ndarray.Activation(input_vector, act_type = ”sigmoid”)	tf.keras.activations.sigmoid(input_vector)
5. Flattening	matrix.reshape(shape=(-1,))	tf.reshape(matrix, [-1])
6. Normalization	matrix/mx.nd.norm(matrix, ord=2, axis=None)	tf.linalg.normalize(matrix, ord=2, axis=None)[0]

MLP Construction

```keras
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

# Get your data
x_train = np.random.random((1000, 20))
y_train = np.random.randint(2, size=(1000, 1))

# Build the model
model = Sequential()
model.add(Dense(64, input_dim=20, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Configures the model for training
model.compile(loss='binary_crossentropy',
              optimizer = optimizers.SGD(lr=0.01),
              metrics=['accuracy'])

# Start the training loop
model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
```

```gluon
import mxnet as mx
from mxnet import gluon
from mxnet.gluon.contrib.estimator import estimator

# Get your data
x_train = np.random.random((1000, 20))
y_train = np.random.randint(2, size=(1000, 1))
dataset = mx.gluon.data.dataset.ArrayDataset(x_train, y_train)
train_data_loader = gluon.data.DataLoader(dataset, batch_size=128)

# Build the model
model = gluon.nn.Sequential()
with model.name_scope():
    model.add(gluon.nn.Dense(64, activation="relu"))
    model.add(gluon.nn.Dense(64, activation="relu"))
    model.add(gluon.nn.Dense(1, activation="sigmoid"))

# Configures the model for training
loss_fn = gluon.loss.SigmoidBinaryCrossEntropyLoss
trainer = gluon.Trainer(model.collect_params(), 'sgd', {'learning_rate': 0.01})
train_acc = mx.metric.Accuracy() # Metric to monitor

# Define the estimator, by passing to it
# the model, loss function, metrics, trainer object and context
est = estimator.Estimator(net = model,
                          loss = loss_fn,
                          metrics = train_acc,
                          trainer = trainer)

# Start the training loop
est.fit(train_data = train_data_loader, epochs=20)
```

Inside the fit() method



BDAPro
@DIMA

Training Loop

Autodifferentiation -> Reach the objective

Take one sample → Compute the output **(Forward pass)** → Compare output label with the real label → Compute the error **(Loss/Error function)** → Compute the gradient of the error → Update the weights **(Backward pass)** → Next Sample

Error Functions -> Encode the objective

Error Functions

y(weights) = functions to be derivated

BDAPro
@DIMA

Mean Square Error (regression):

$$= \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Binary Cross Entropy (classification):

$$= -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

TensorFlow

binary_crossentropy(...)
categorical_crossentropy(...)
categorical_hinge(...)
cosine_similarity(...)
hinge(...)
kullback_leibler_divergence(...)
logcosh(...) mean_absolute_error(...)
mean_absolute_percentage_error(...)
mean_squared_error(...)
mean_squared_logarithmic_error(...)
poisson(...)
sparse_categorical_crossentropy(...)
squared_hinge(...)

mxnet

L2Loss()
L1Loss()
SigmoidBinaryCrossEntropyLoss()
SigmoidBCELoss
SoftmaxCrossEntropyLoss()
SoftmaxCELoss
KLDivLoss()
CTCLoss()
HuberLoss()
HingeLoss()
SquaredHingeLoss()
LogisticLoss()
TripletLoss()
PoissonNLLLoss()
CosineEmbeddingLoss()

Automatic Differentiation implementation

BDAPro
@DIMA

mxnet.**autograd** API

What does it do? "It expedites the gradient calculation by automatically calculating derivatives."

```
from mxnet import nd
from mxnet import autograd

x = nd.array([[1, 2], [3, 4]])
# tell what to monitor
x.attach_grad()
# record operation
with autograd.record():
    # do the operation you want
    y = 2 * x * x
# compute backward pass
y.backward()
# derivative is stored in:
x.grad
```

tf.**GradientTape** API

What does it do? "It computes the gradient of a computation with respect to its input variables."

```
import tensorflow as tf

x = tf.constant([[1, 2], [3, 4]])
# record the tape
with tf.GradientTape() as tape:
    # tell what to monitor
    tape.watch(x)
    # do the operation you want
    y = 2 * x * x
# read the gradient from the tape
dy_dx = tape.gradient(y, x)
```

Variables are automatically watched by GradientTape

4. Microbenchmark results

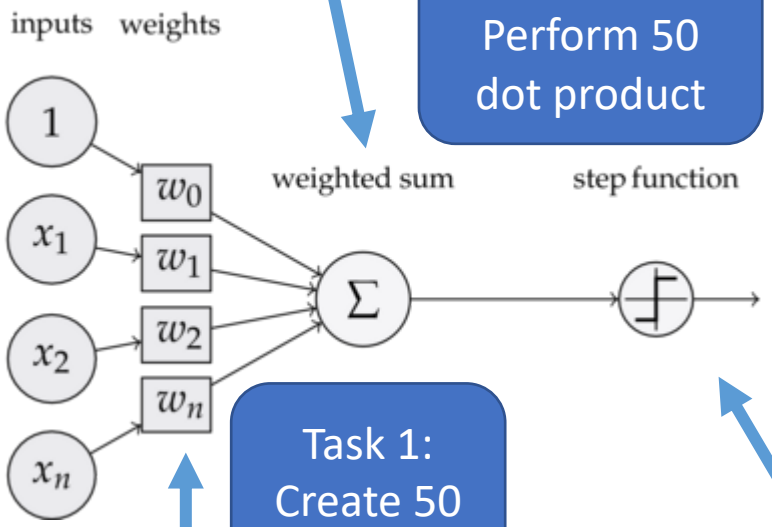
Fundamental Operations

BDAPro @DIMA

2. Dot product | 3. Matrix multiplication

Task 2: Perform 50 dot product

Task 3: Perform 50 matrix multiplication



Task 1: Create 50 vectors

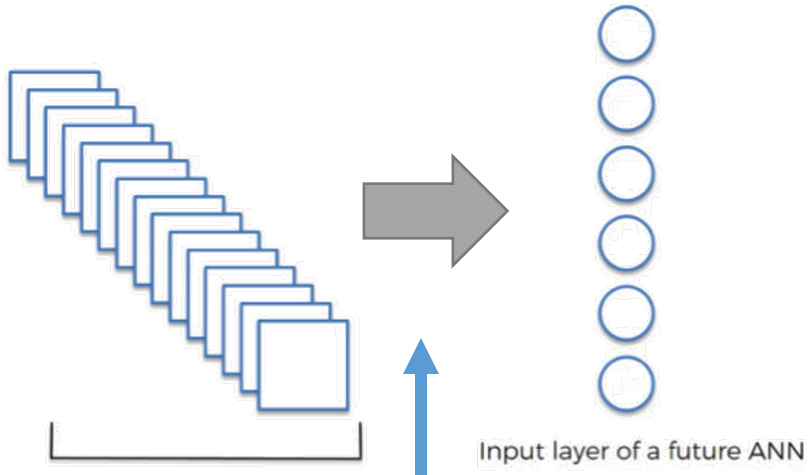
Task 4: Execute the activation function on 50 vectors

1. Vector creation

4. Activation function

Task 6: Normalize 50 vectors

6. Vector Normalization



Task 5: Flatten 50 matrices to vectors

5. Flattening

Image source: https://blog.dbrgn.ch/2013/3/26/perceptrons-in-python/

Image source: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

Tasks 1 – Vector Creation

BDAPro
@DIMA

Task 1:
Create 50
vectors

Task 1.1:
from Numpy

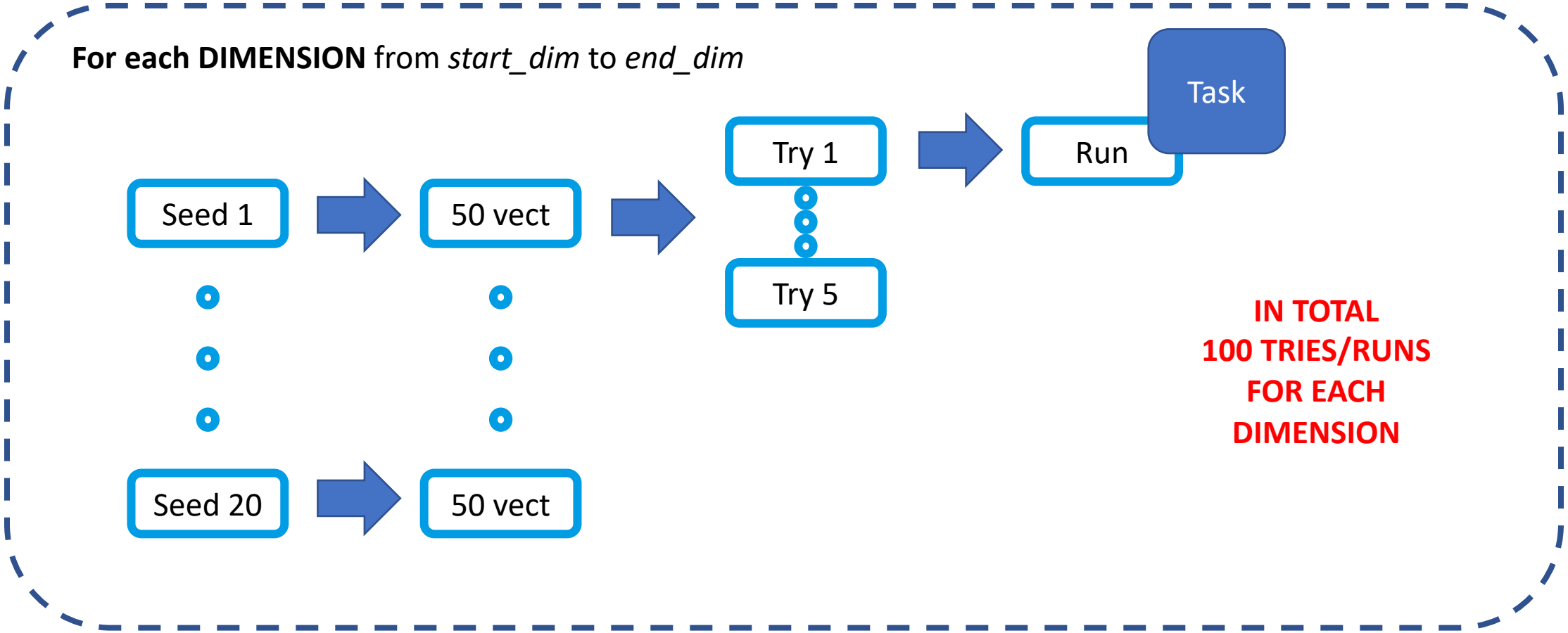
NumPy

Task 4.2:
From List

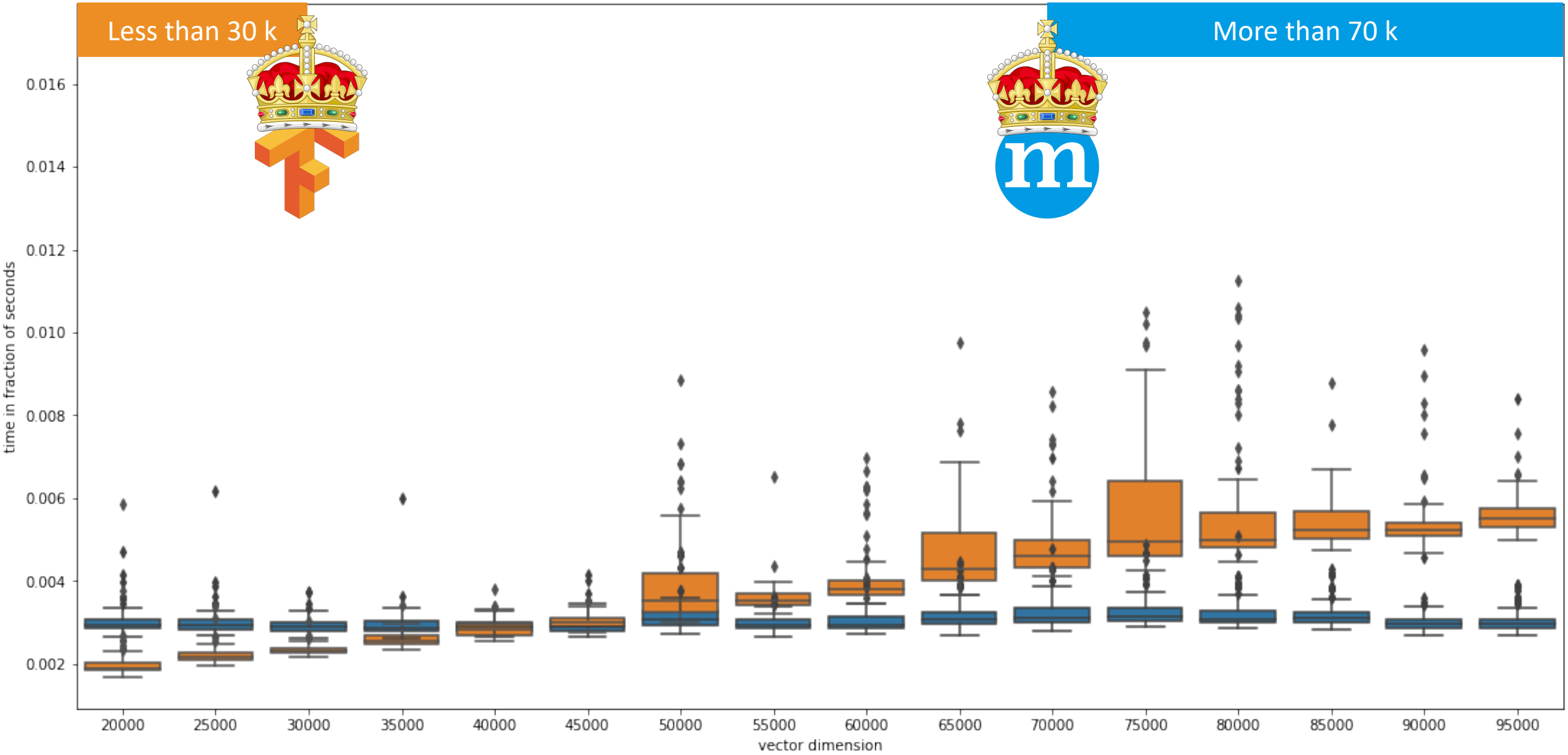
python™ list

Experiment Description

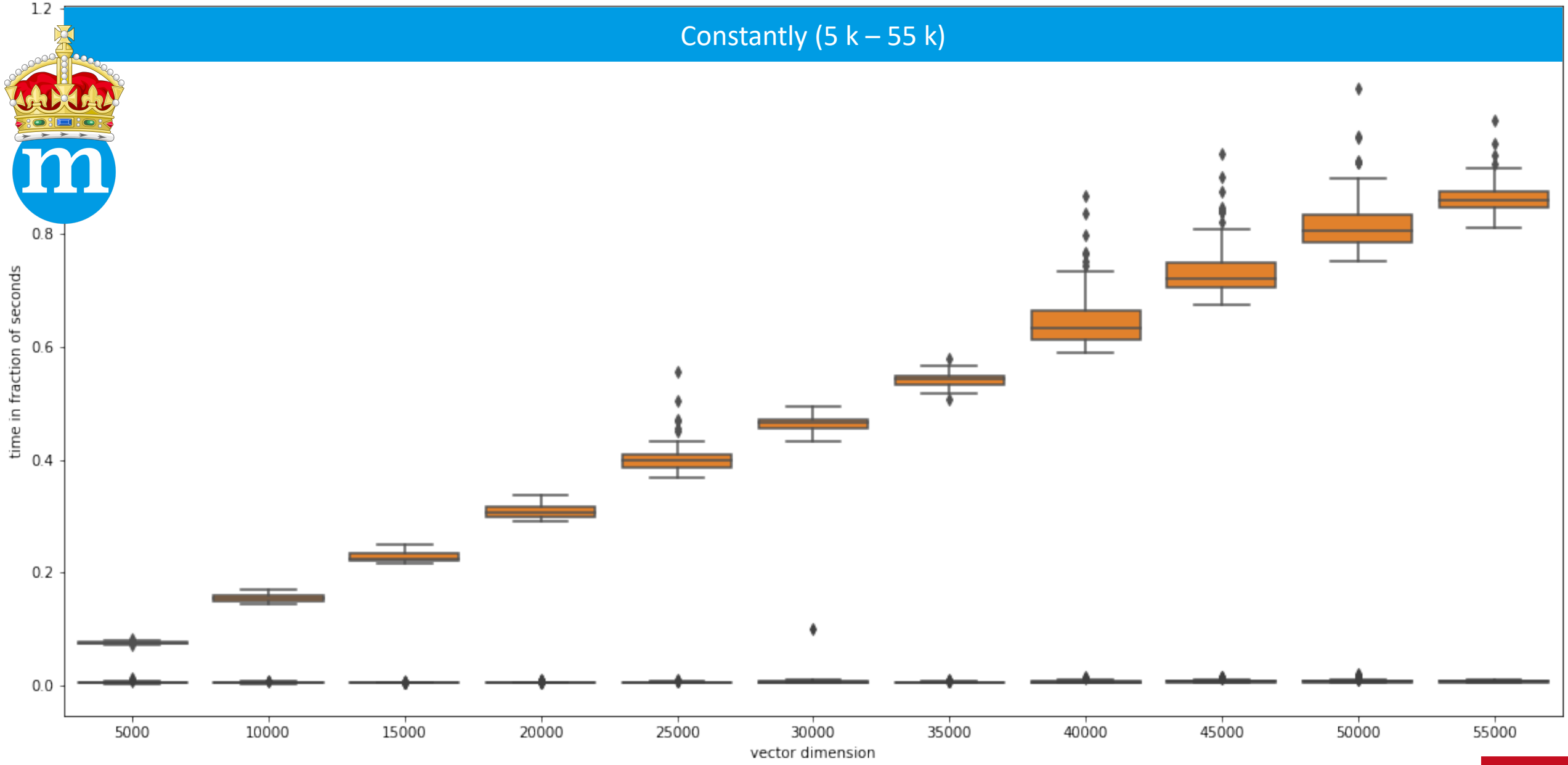
BDAPro
@DIMA



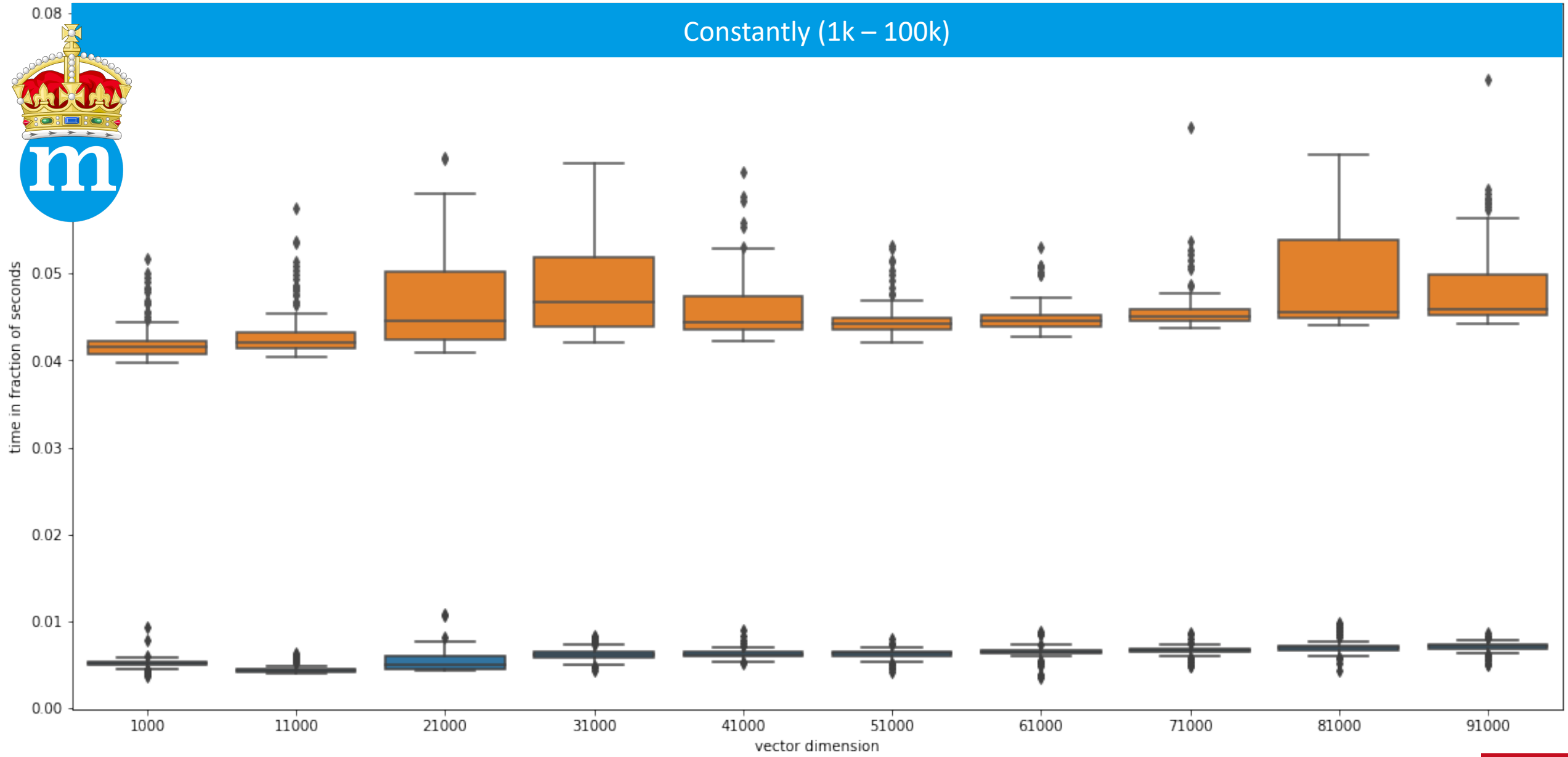
Task 1.1 - Initialization of 50 vectors - from Numpy input



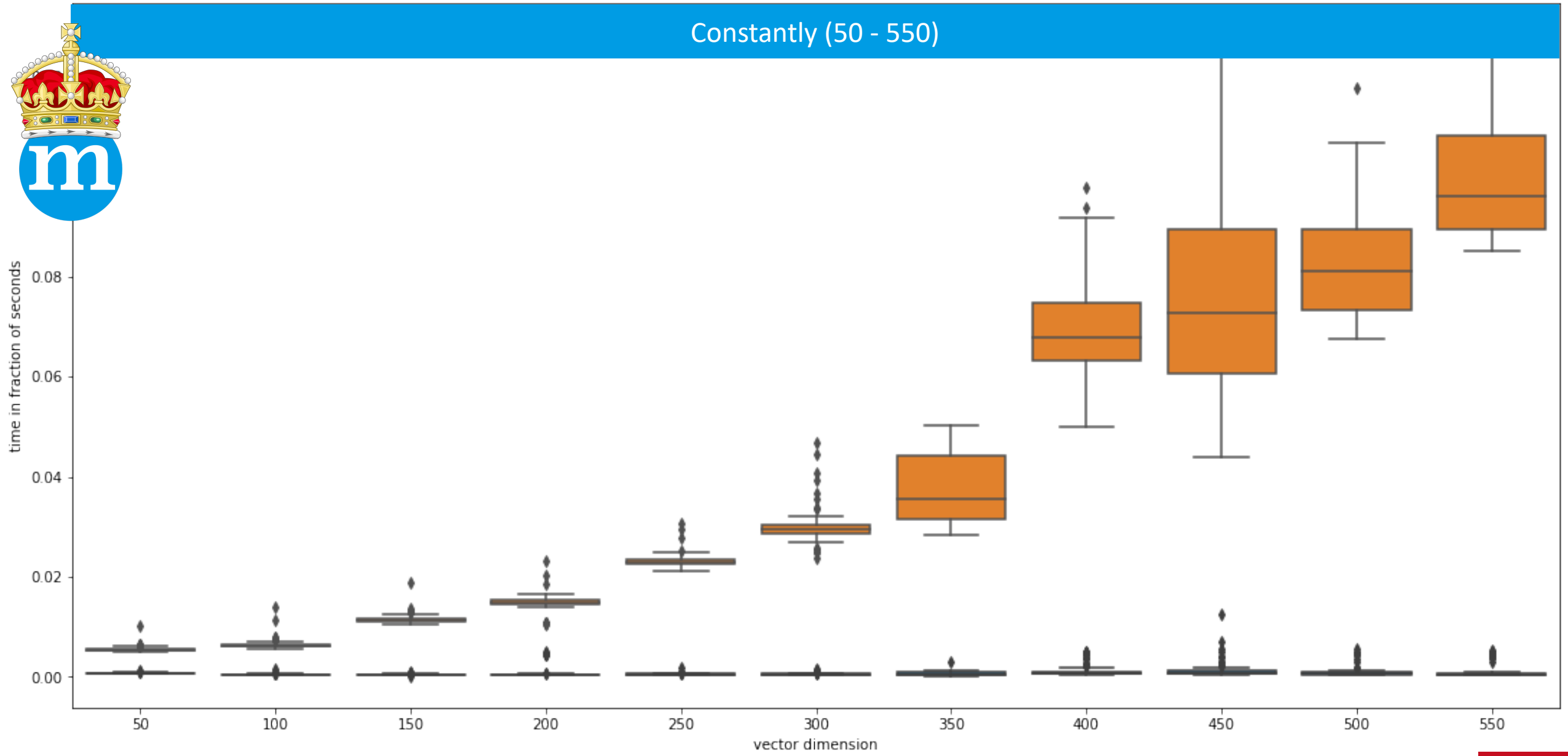
Task 1.2 - Initialization of 50 vectors - from list input



Task 2 - Dot Product of 50 vectors



Task 3 – 50 Matrix multiplication –between square matrices



Tasks 4 – Activation Functions

BDAPro
@DIMA

Task 4: Execute the activation function on 50 vectors

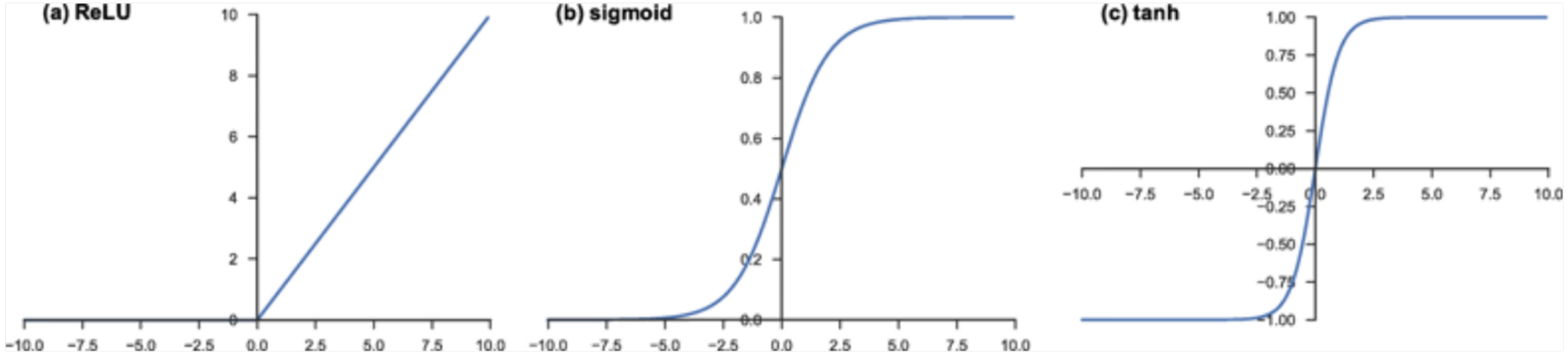
Task 4.1: Sigmoid

Task 4.2: ReLu

Task 4.3: tanh

Task 4.4: softReLu

Task 4.5: softsign



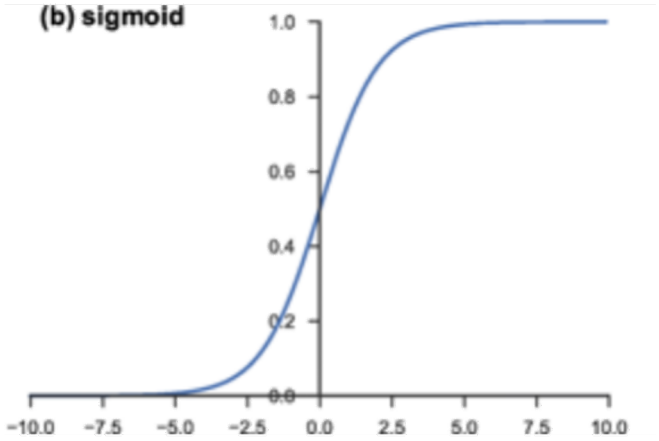
Activation functions commonly applied to neural networks: **a** rectified linear unit (ReLU), **b** sigmoid, and **c** hyperbolic tangent (tanh)

Task 4.1 - Activation on 50 vectors - Sigmoid

Less than 500

More than 36 k

Task 4.1: Sigmoid



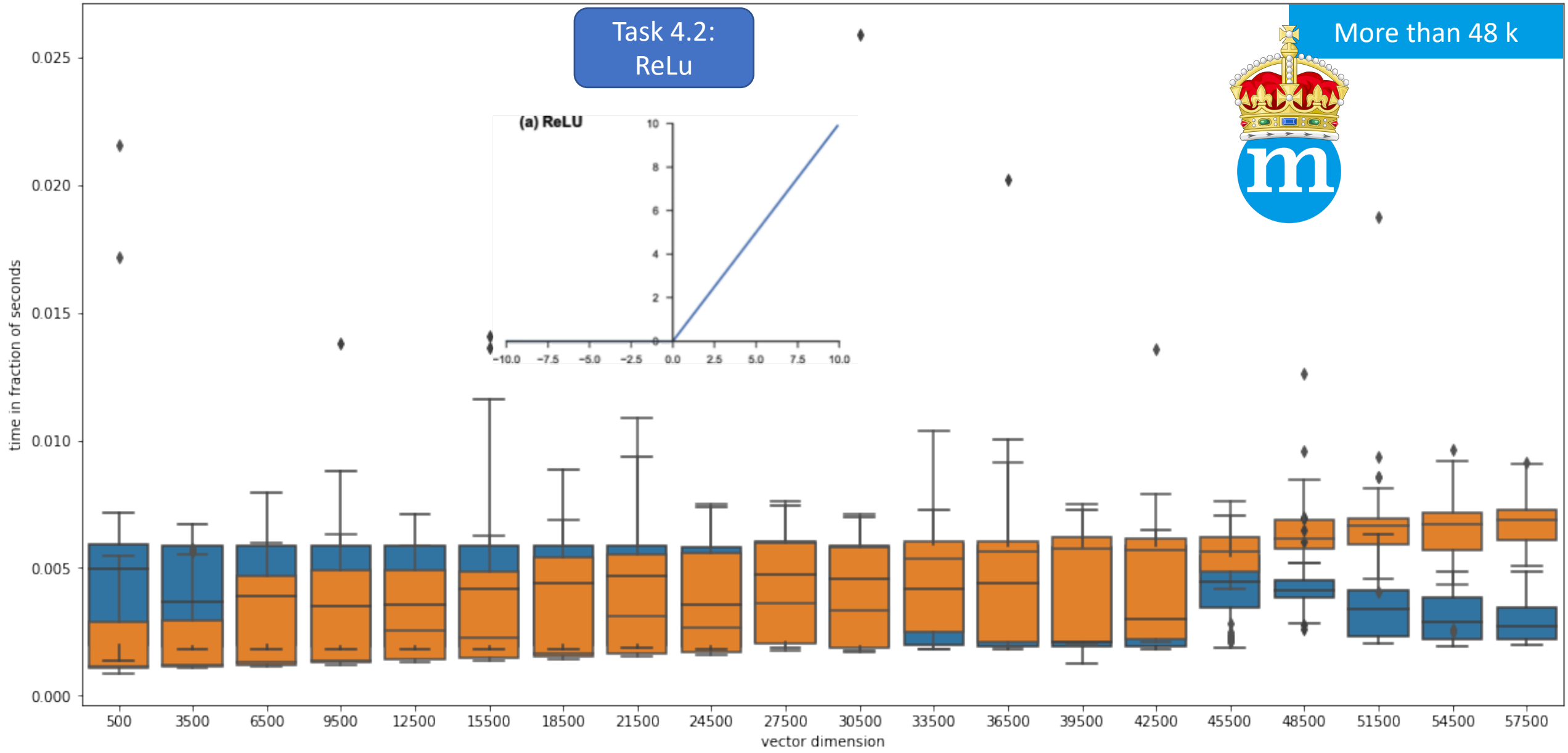
time in fraction of seconds

0.06
0.05
0.04
0.03
0.02
0.01
0.00

500 3500 6500 9500 12500 15500 18500 21500 24500 27500 30500 33500 36500 39500 42500 45500 48500 51500 54500 57500

vector dimension

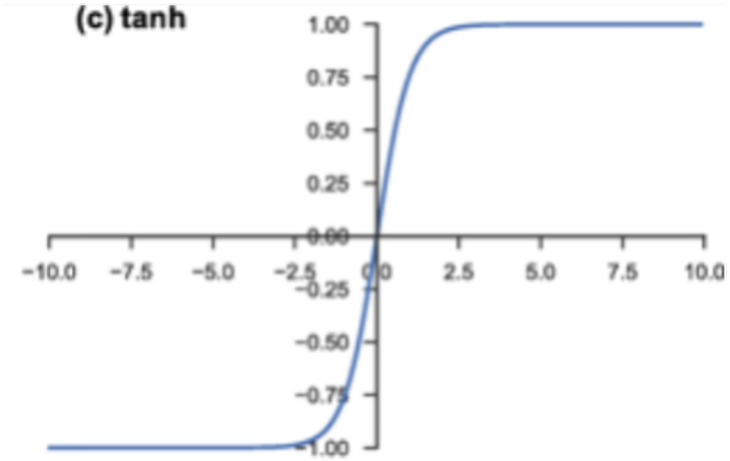
Task 4.2 - Activation on 50 vectors - Relu



Task 4.3 - Activation on 50 vectors - TanH

Task 4.3: tanh

More than 36 k



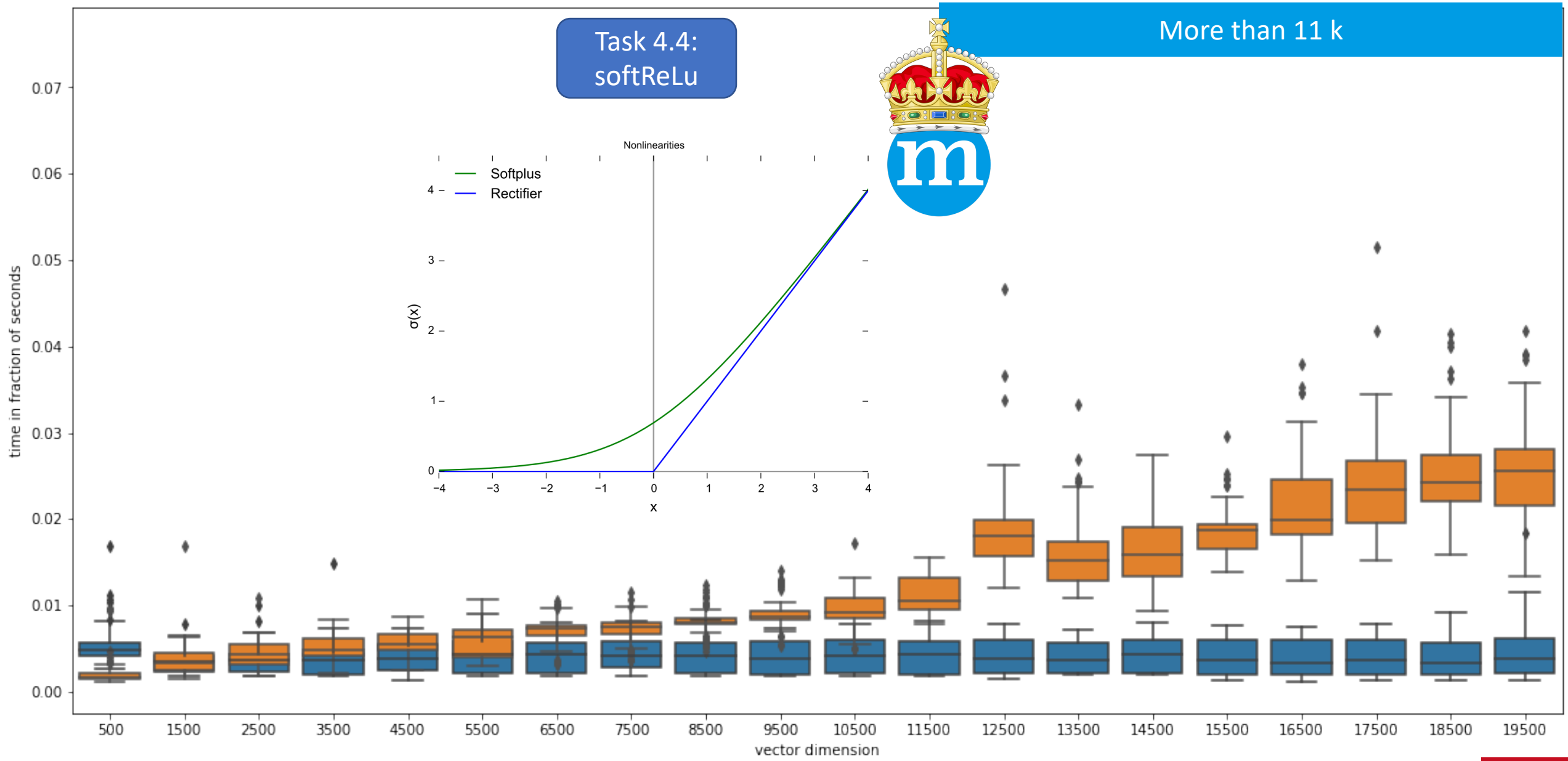
time in fraction of seconds

0.00
0.01
0.02
0.03
0.04
0.05

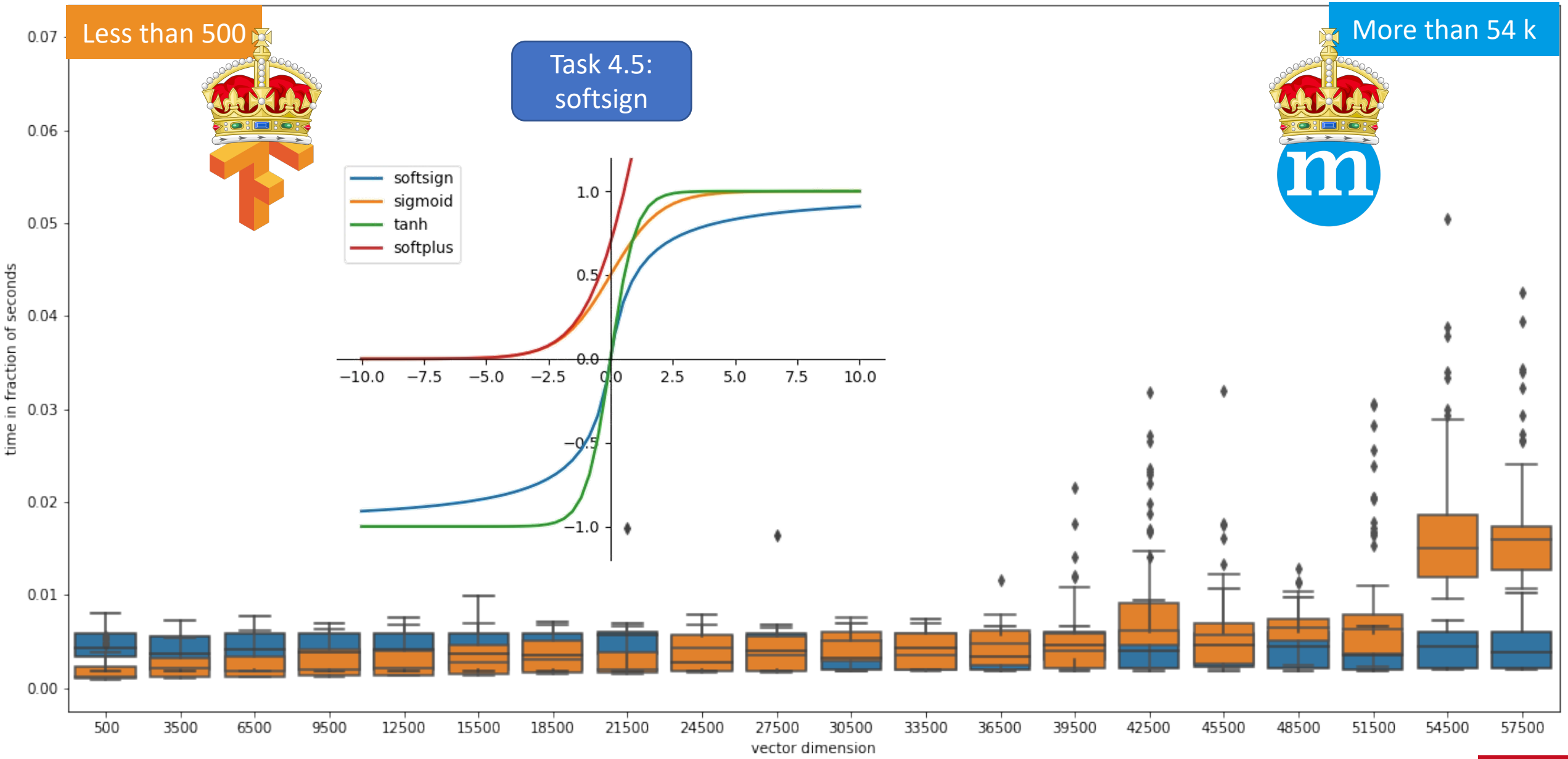
500 3500 6500 9500 12500 15500 18500 21500 24500 27500 30500 33500 36500 39500 42500 45500 48500 51500 54500 57500

vector dimension

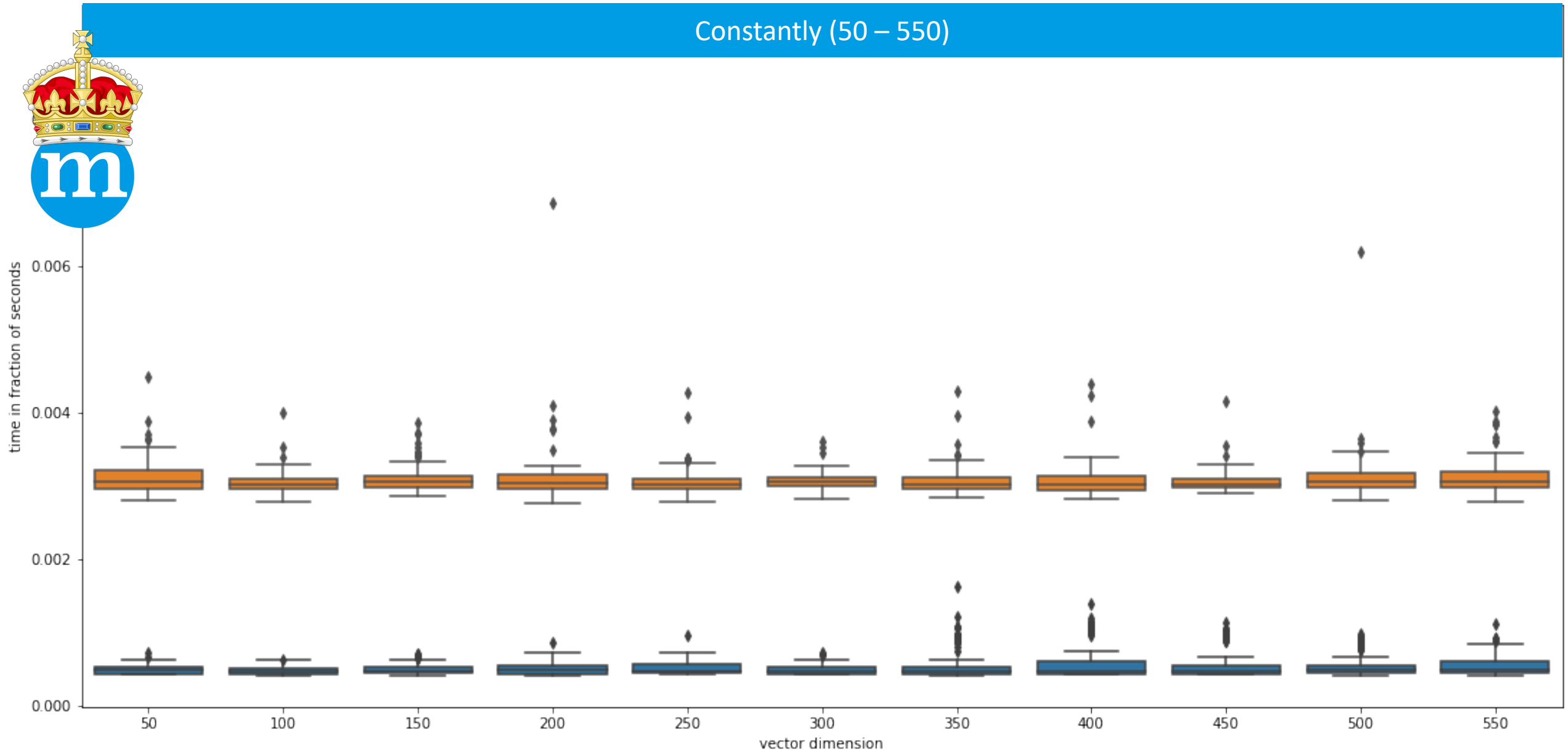
Task 4.4 - Activation on 50 vectors - SoftRelu



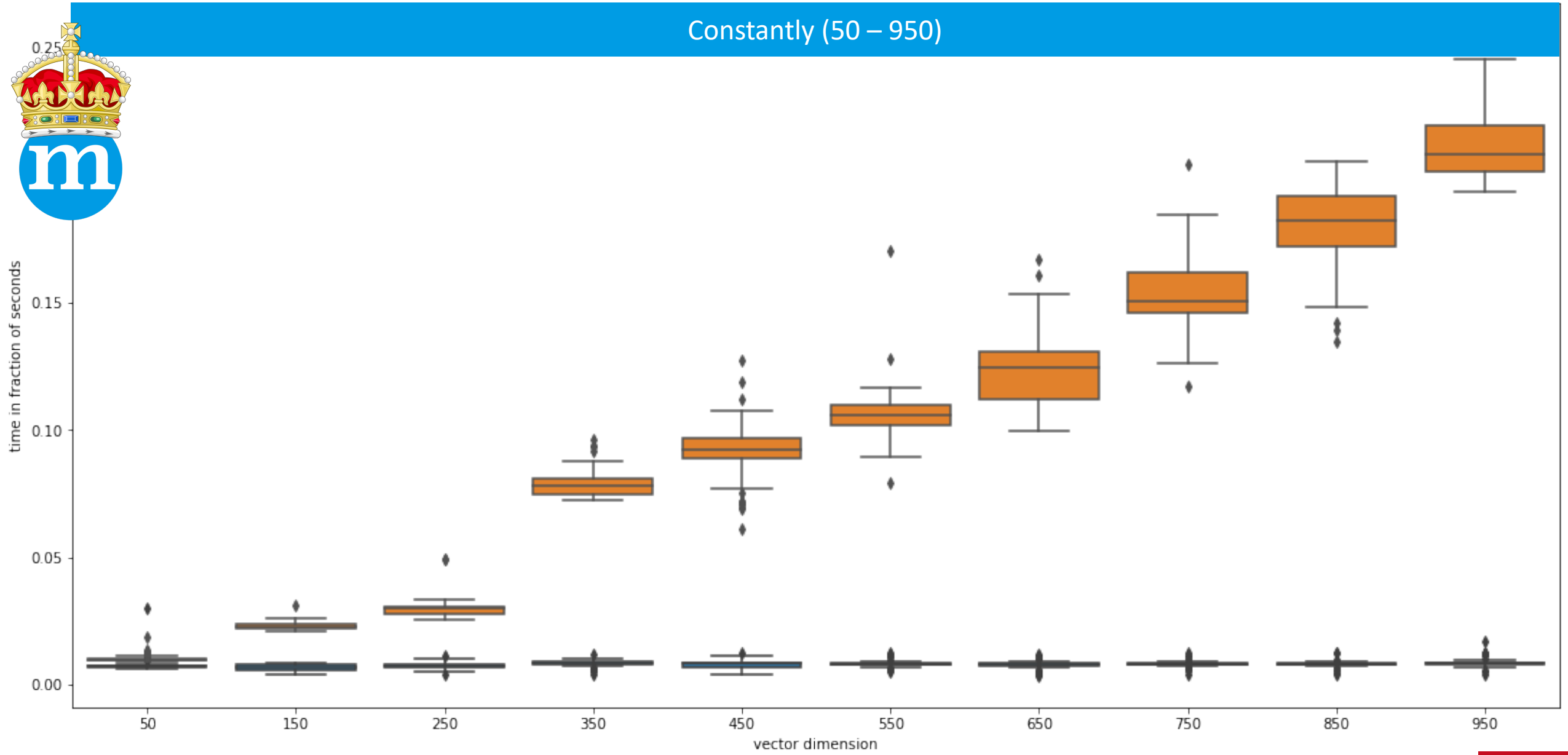
Task 4.5 - Activation on 50 vectors - SoftSign



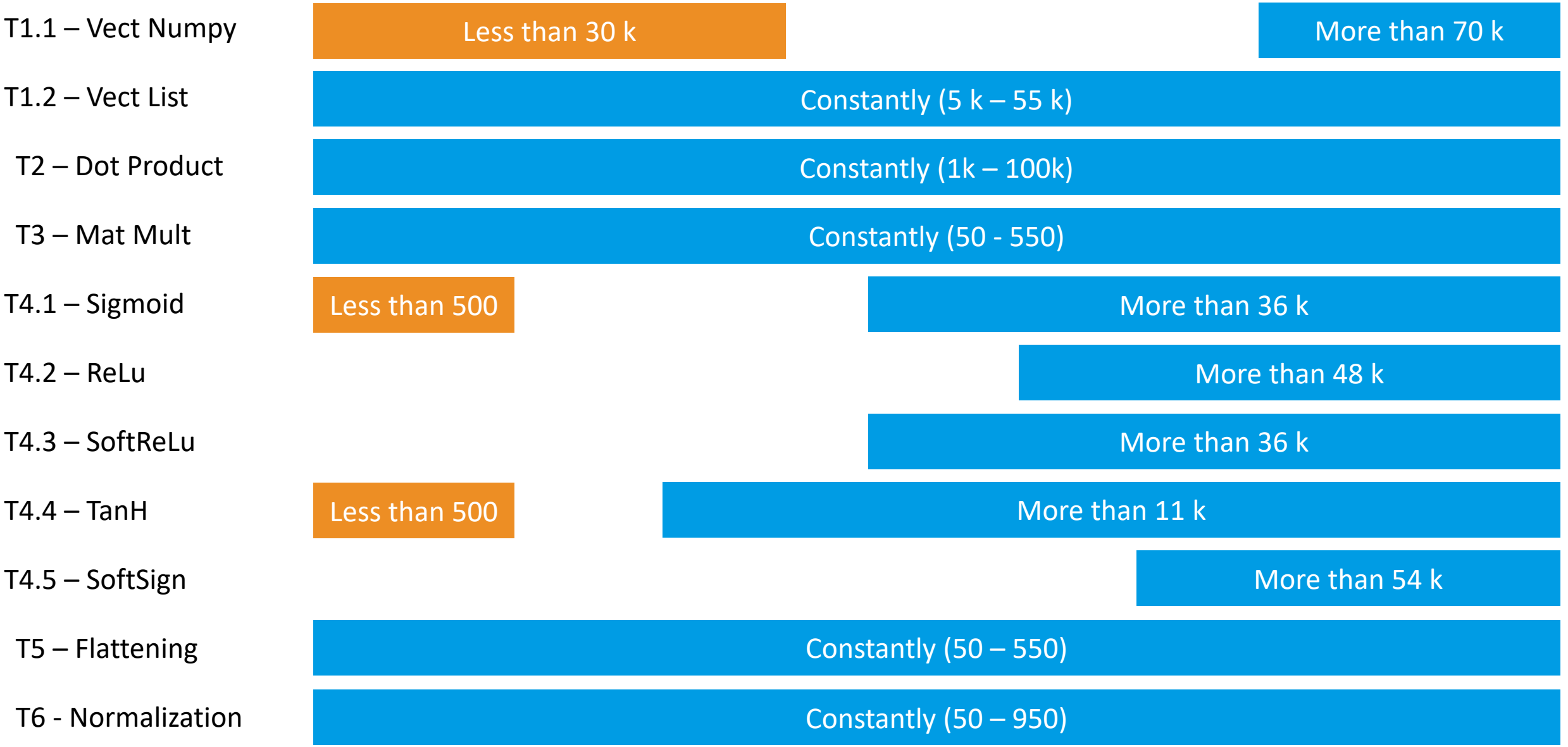
Task 5 – Flattening of 50 square matrices



Task 6 – Normalize 50 matrices



Benchmark Summary – Fundamental Operations



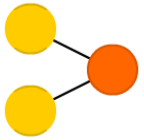
5. LeNet

A mostly complete chart of

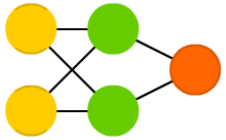
Neural Networks

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

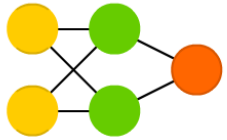
Perceptron (P)



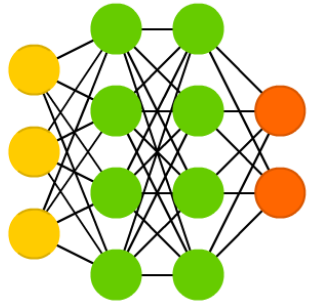
Feed Forward (FF)



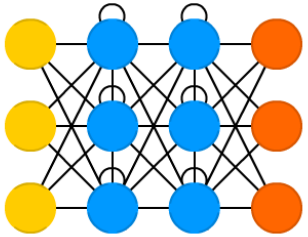
Radial Basis Network (RBF)



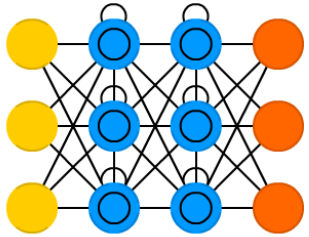
Deep Feed Forward (DFF)



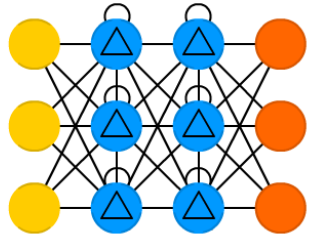
Recurrent Neural Network (RNN)



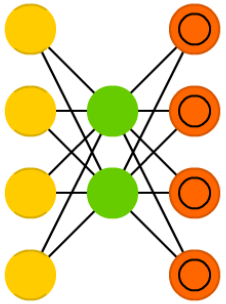
Long / Short Term Memory (LSTM)



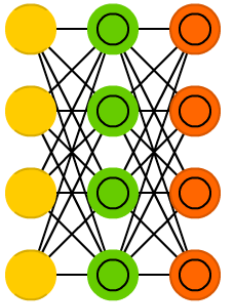
Gated Recurrent Unit (GRU)



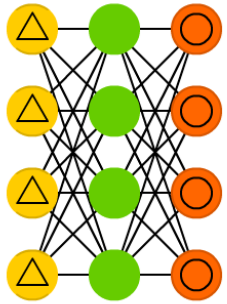
Auto Encoder (AE)



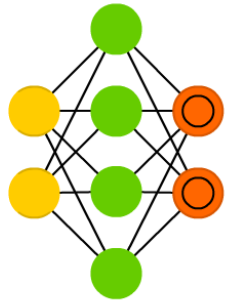
Variational AE (VAE)



Denoising AE (DAE)

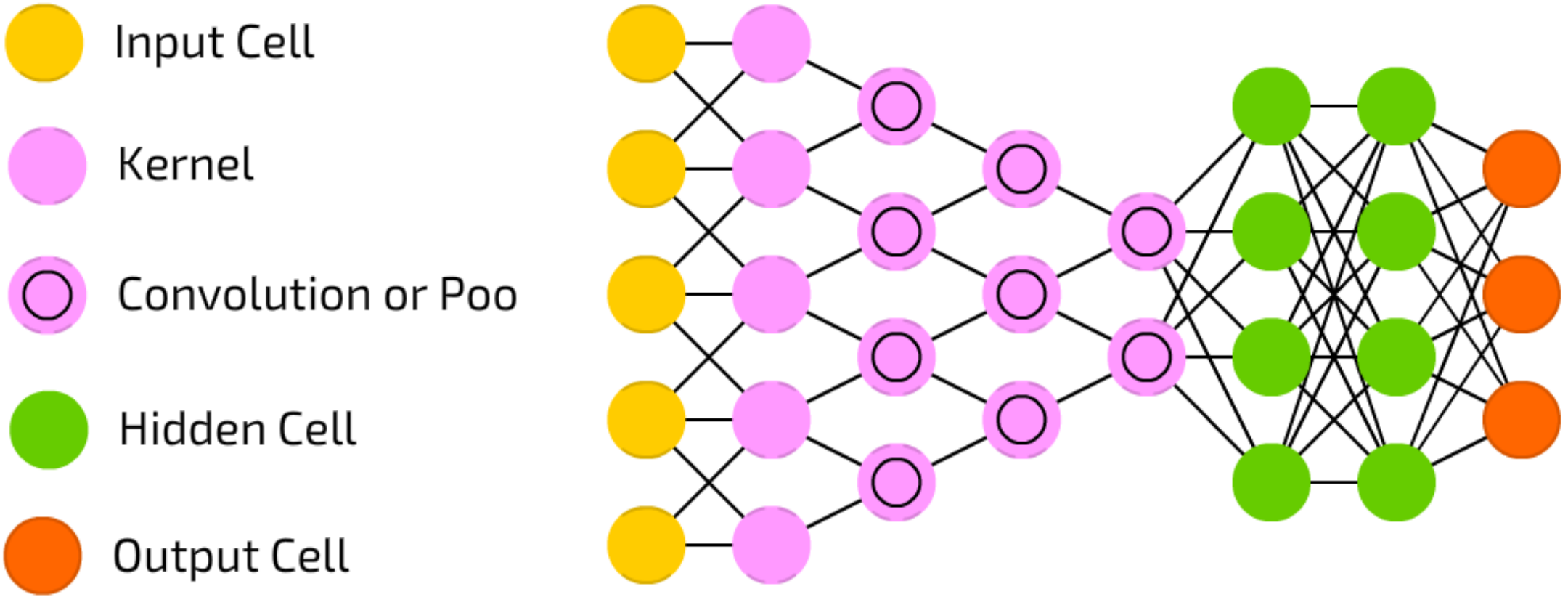


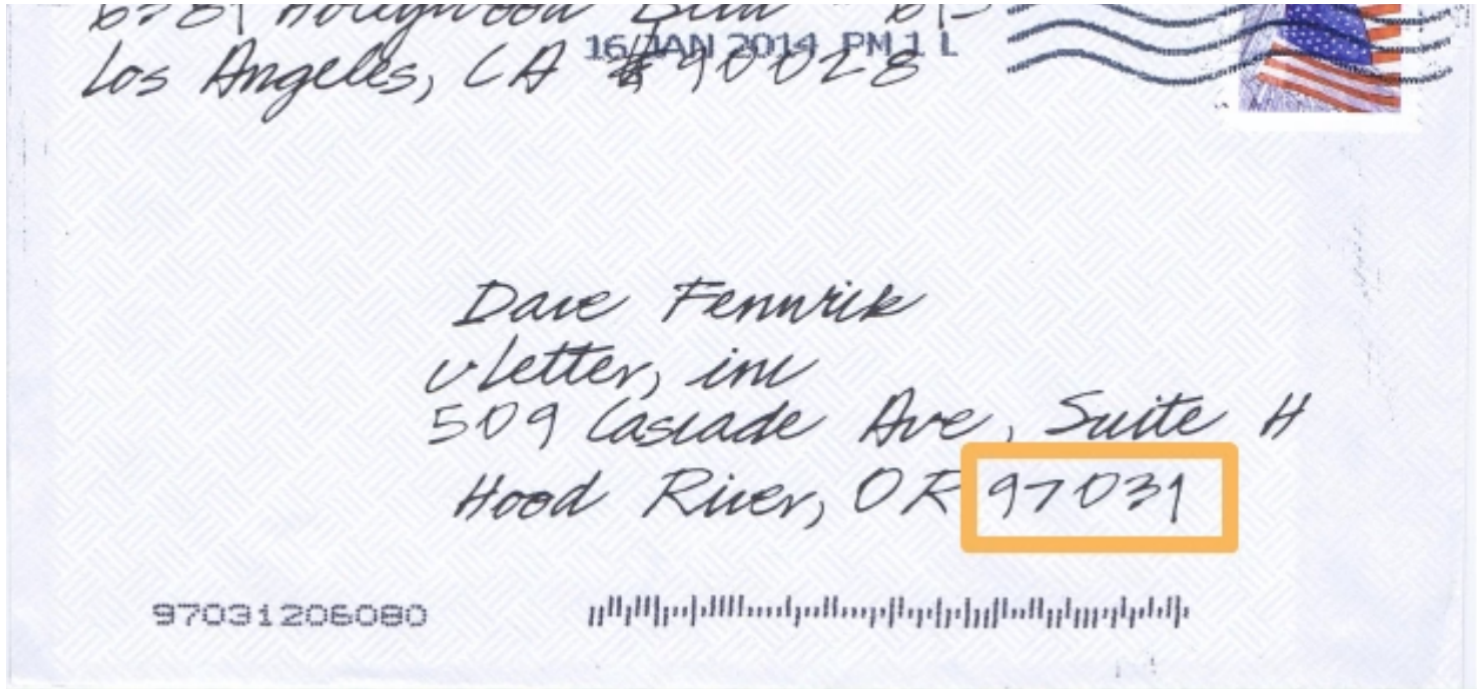
Sparse AE (SAE)



BDAPro
@DIMA

Deep Convolutional Network (DCN)





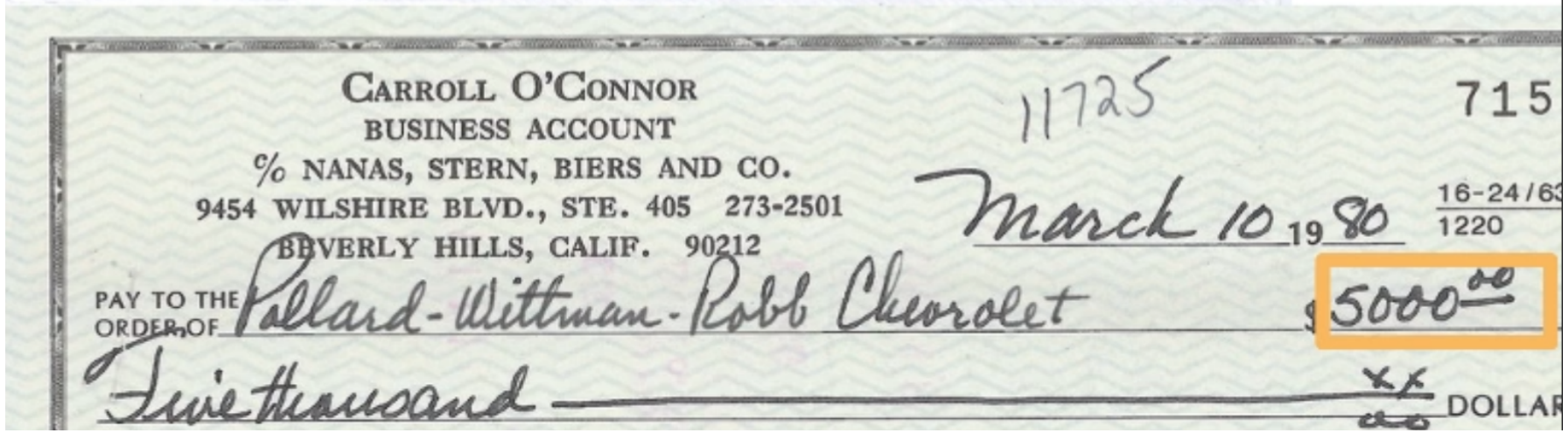


Image source: https://www.youtube.com/watch?v=m3BrTjo2zUA

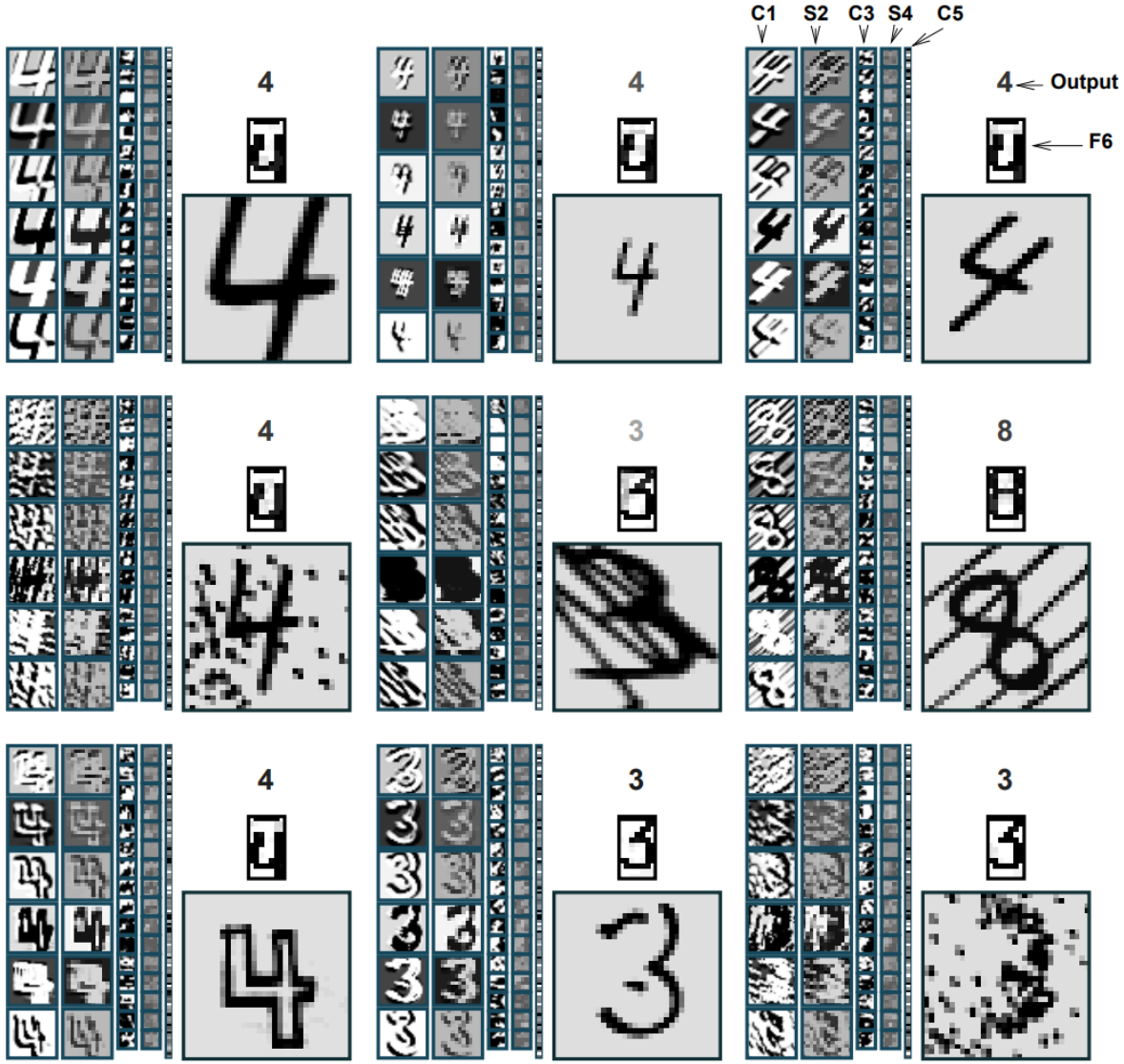


Fig. 13. Examples of unusual, distorted, and noisy characters correctly recognized by LeNet-5. The grey-level of the output label represents the penalty (lighter for higher penalties).

Image source: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

LeNet Architecture

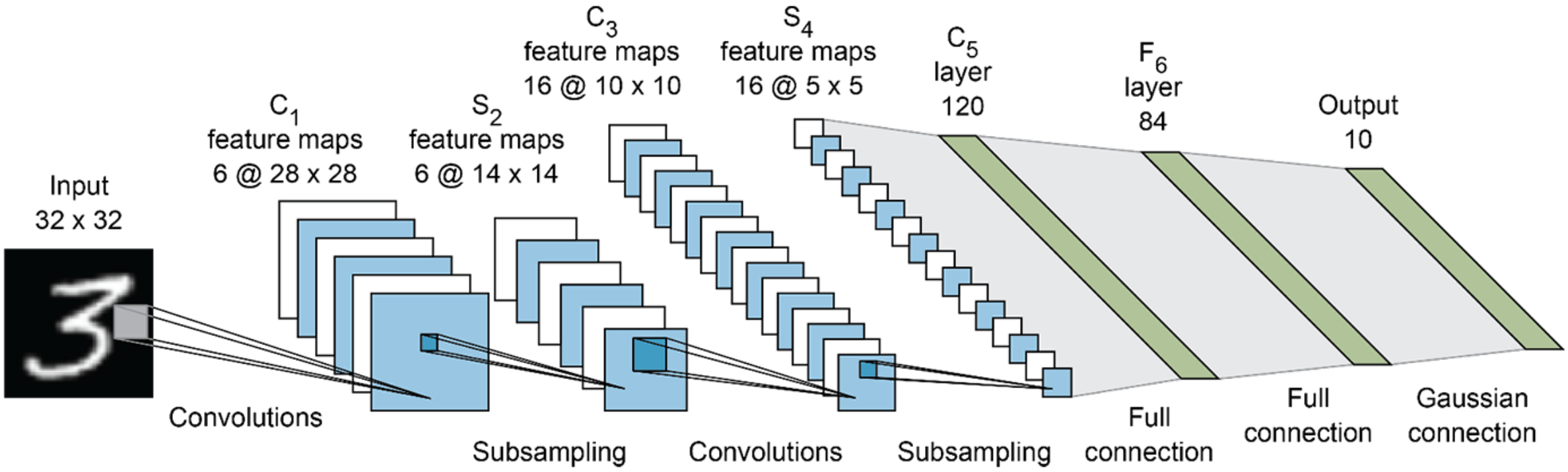
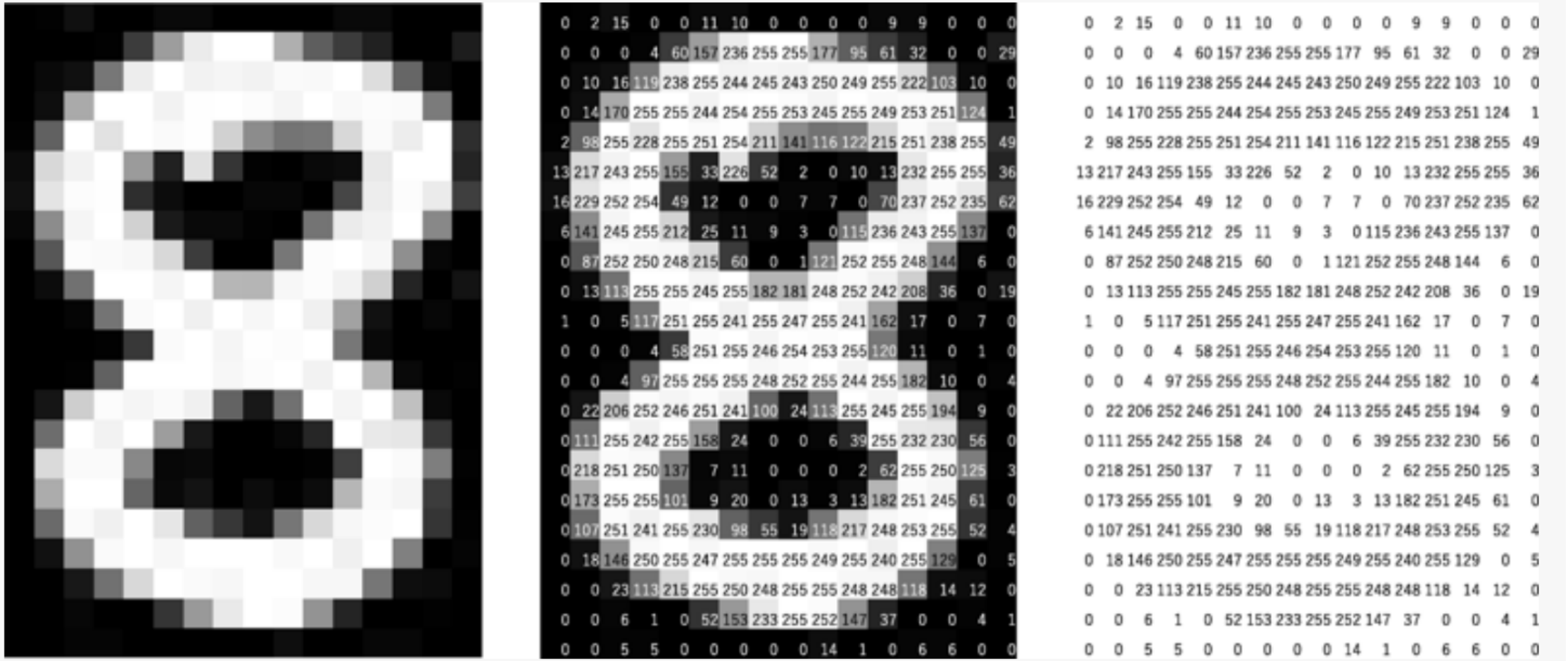


Image source: https://link.springer.com/article/10.1007/s13244-018-0639-9

A list of parameters and hyperparameters in a convolutional neural network (CNN)

	Parameters	Hyperparameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding, activation function
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function
Others		Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regularization, weight initialization, dataset splitting

Figure source: https://link.springer.com/article/10.1007/s13244-018-0639-9

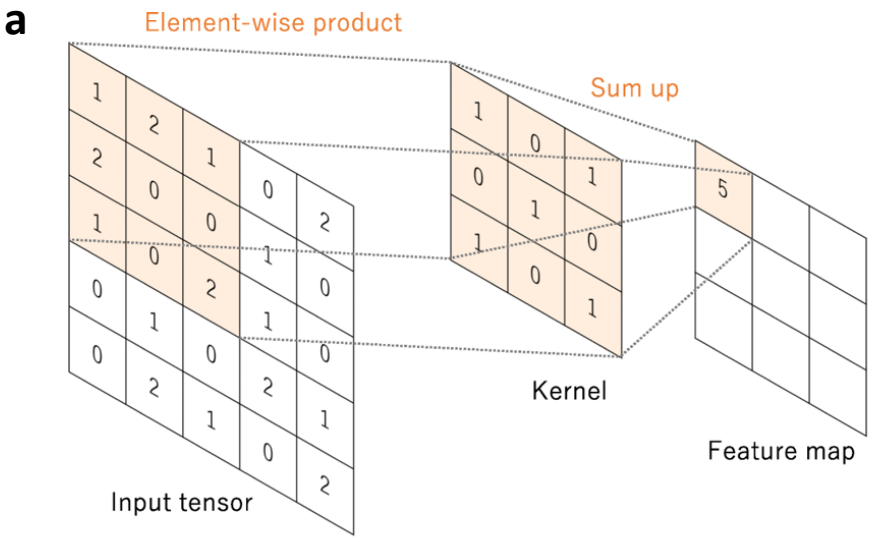


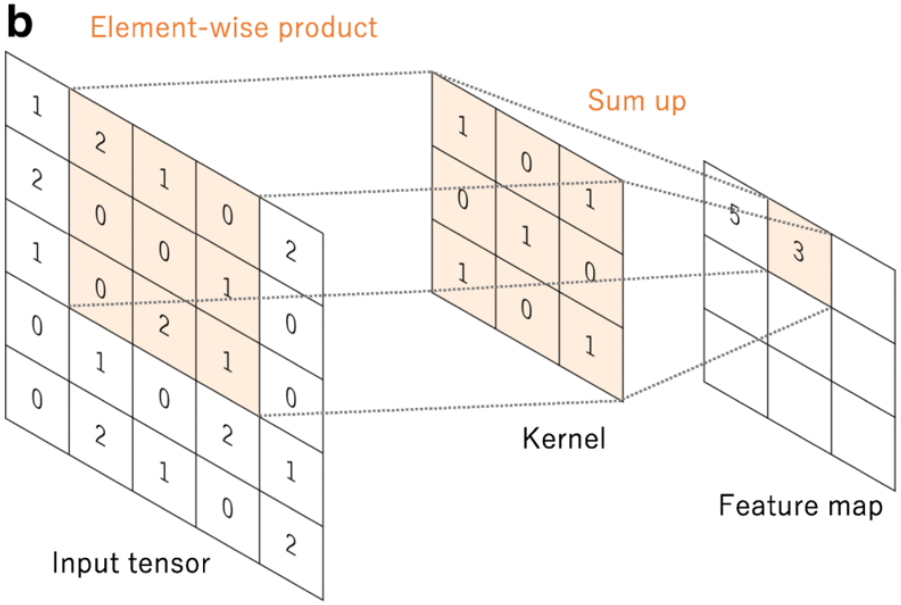
0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Image source: https://link.springer.com/article/10.1007/s13244-018-0639-9

Convolution





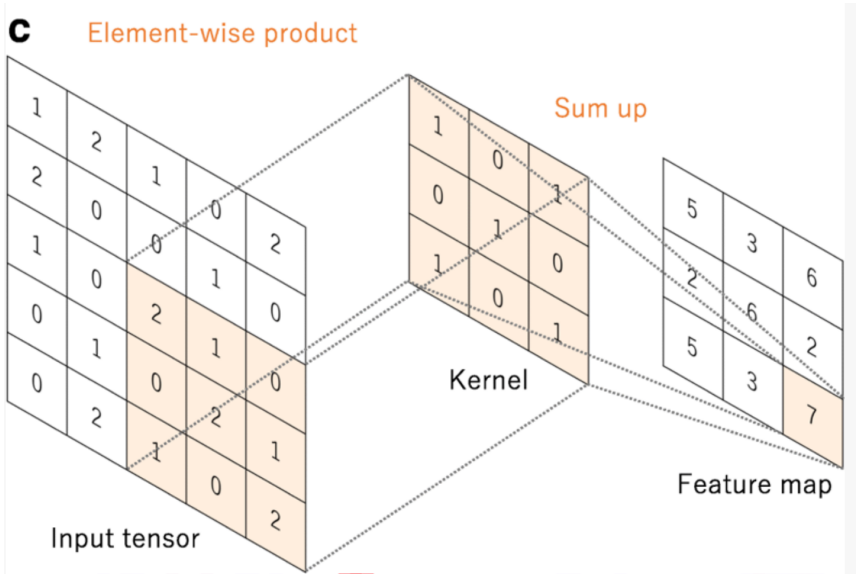
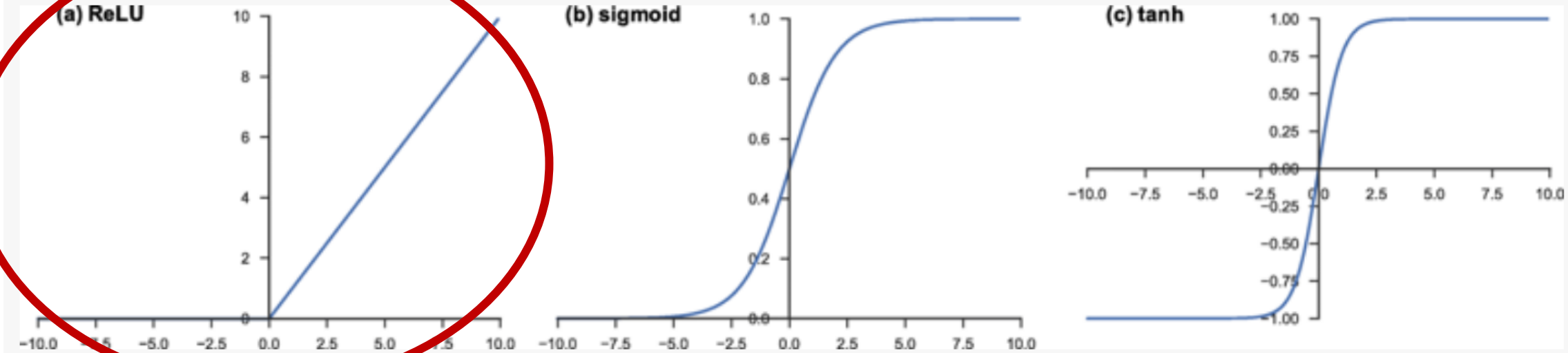


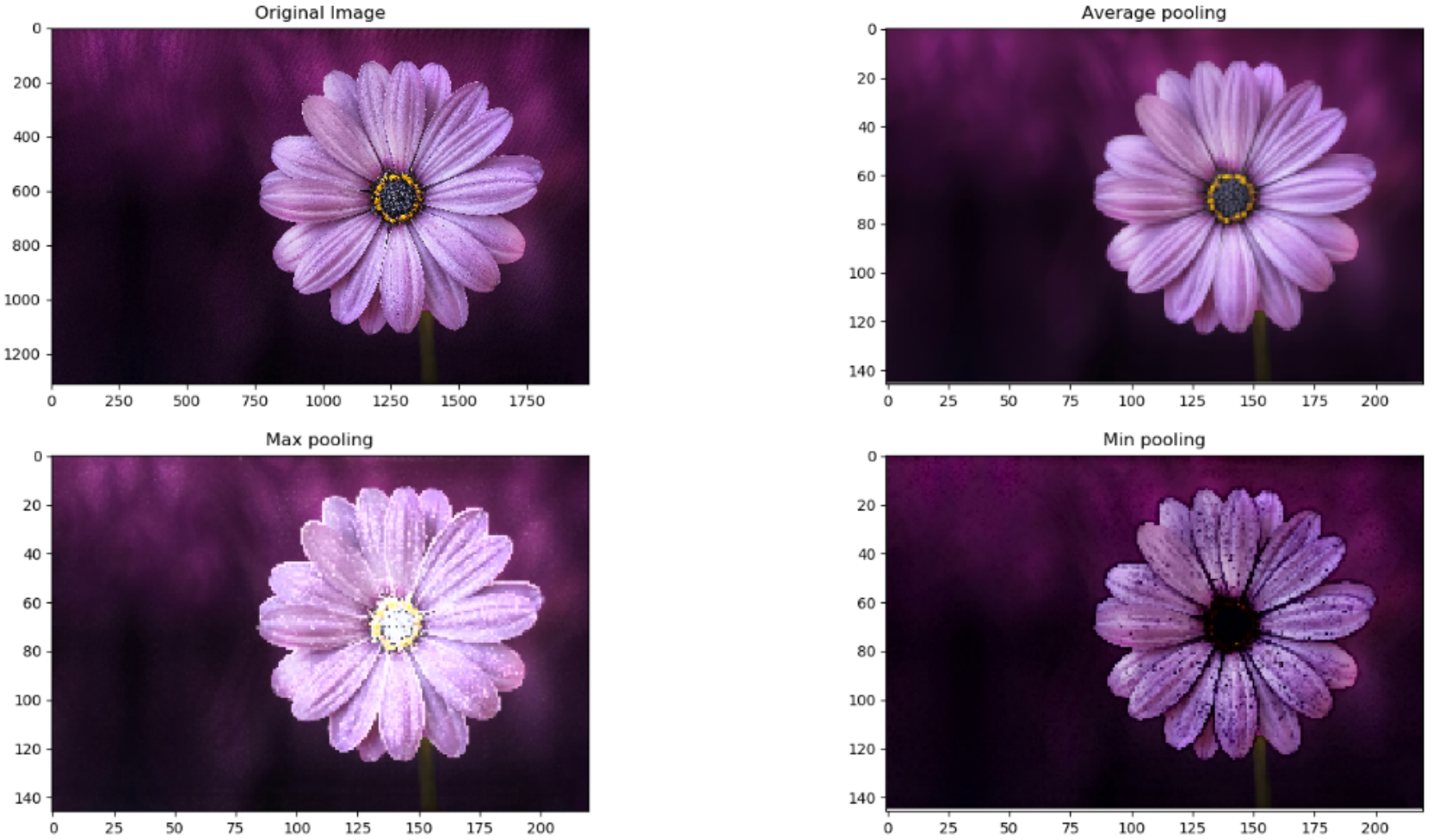
Image source: https://link.springer.com/article/10.1007/s13244-018-0639-9



Activation functions commonly applied to neural networks: **a** rectified linear unit (ReLU), **b** sigmoid, and **c** hyperbolic tangent (tanh)

Image source: https://link.springer.com/article/10.1007/s13244-018-0639-9

Pooling



Average, Max and Min pooling of size 9x9 applied on an image

Image source: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9

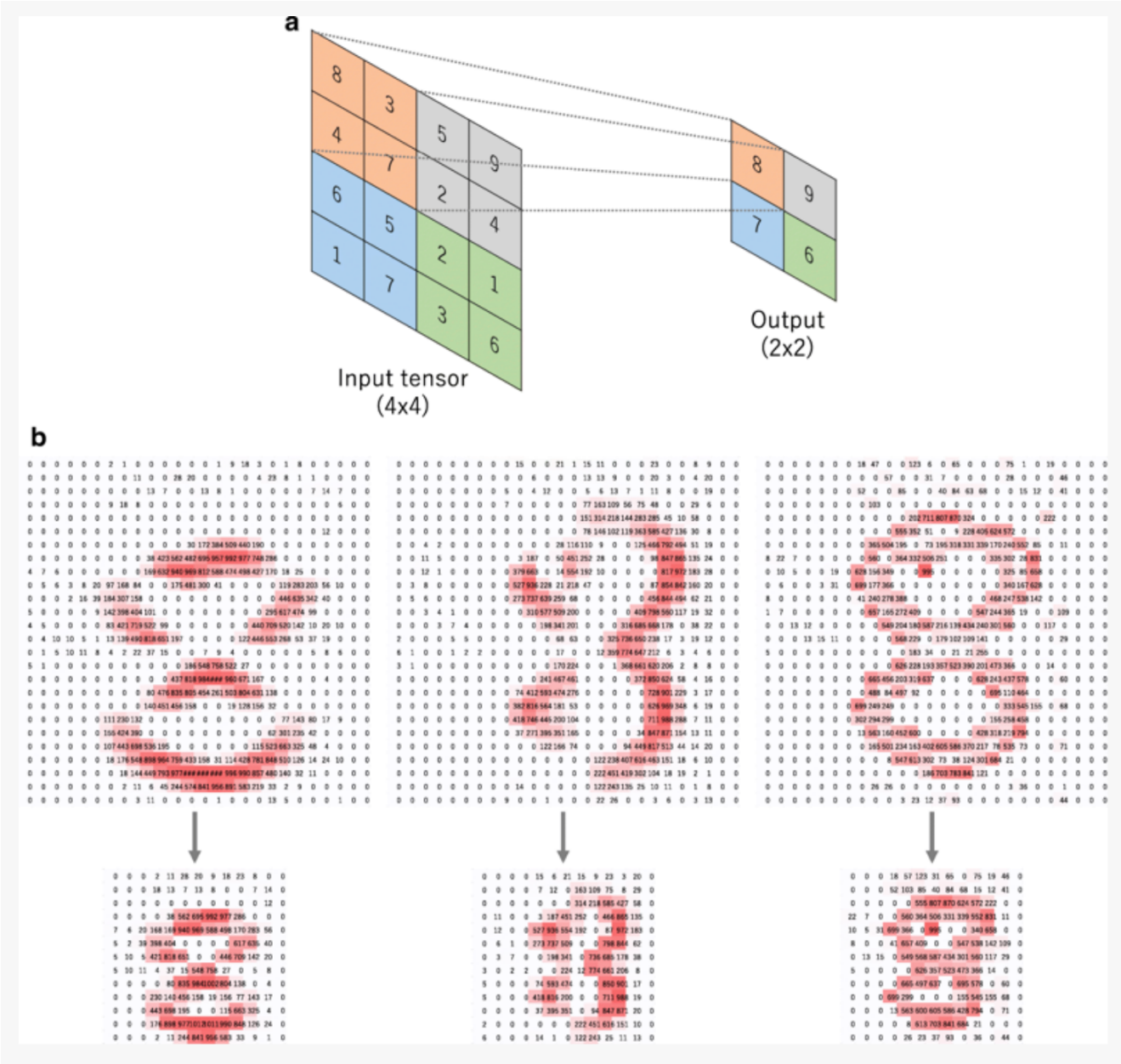


Image source: https://link.springer.com/article/10.1007/s13244-018-0639-9

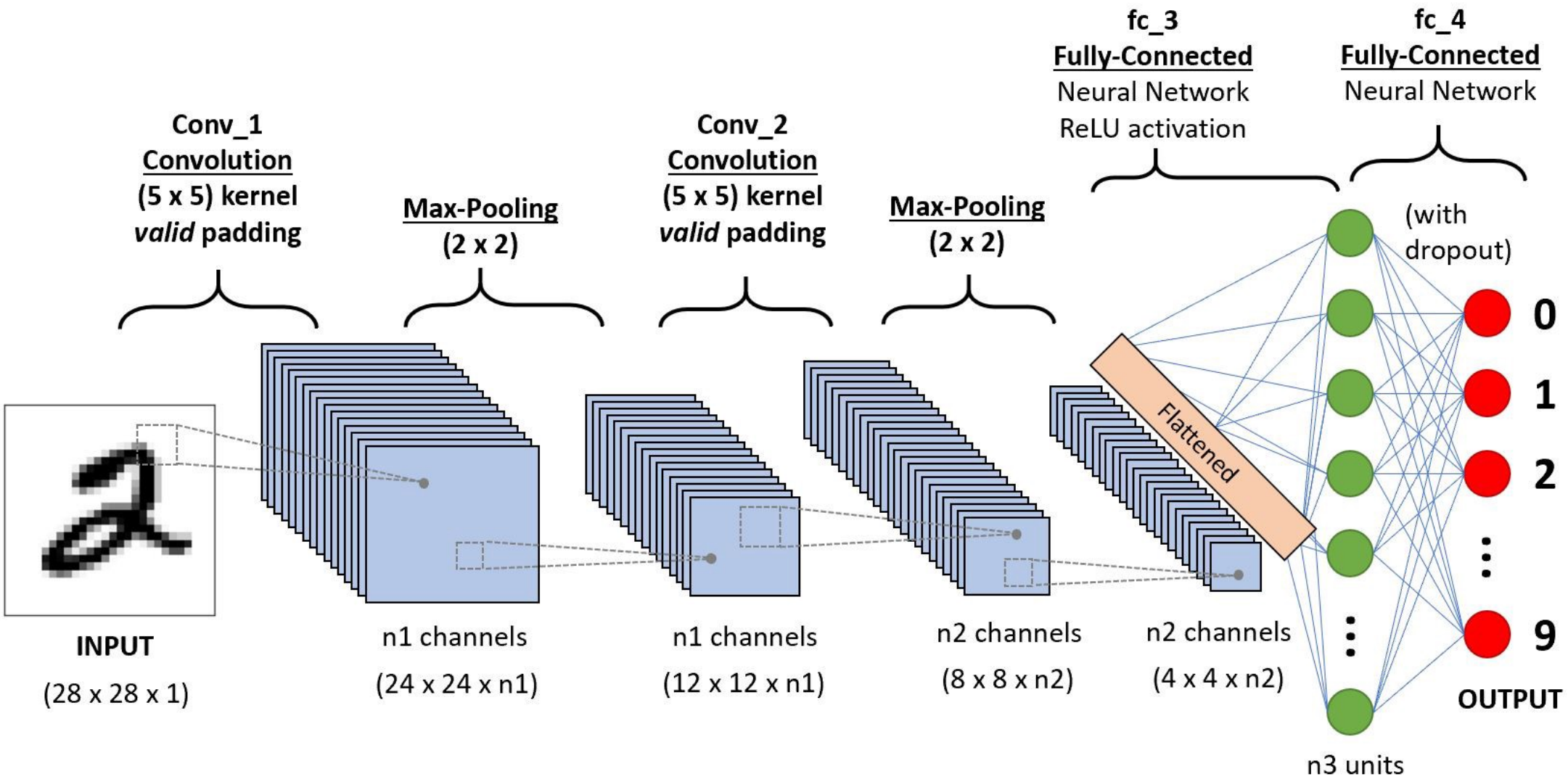


Image source: https://www.researchgate.net/figure/Toy-example-illustrating-the-drawbacks-of-max-pooling-and-average-pooling_fig2_300020038

LeNet Architecture

BDAPro
@DIMA

Layer		Feature Map	Input Size	Kernel Size	Stride	Activation
Input	Image	1	32*32	-	-	-
1	Convolution	6	30*30	5*5	1	Relu
2	Average Pooling	6	15*15	2	2	-
3	Convolution	16	13*13	3*3	1	Relu
4	Average Pooling	16	6*6	2	2	-
Flatten						
5	FC	-	120	-	-	Relu
6	FC	-	84	-	-	Relu
Output	FC	-	10	-	-	Softmax

LeNet Architecture

BDAPro
@DIMA

Hyperparameter	Value
Optimizer	Adam
Learning rate	0.2
Loss function	SOFTMAX-CROSSENTROPY
mini-batch size	200
epochs	10
# images	70000
# training images	60000
# test images	10000
# training images	48000
# validation images	12000

LeNet Construction

BDAPro
@DIMA

GLUON

```
handwritten_net = nn.Sequential()
handwritten_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))
```

Keras

```
model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu', input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=10, activation = 'softmax'))
```

Training Loop

BDAPro
@DIMA

GLUON

```
In [6]: # TRAIN THE NETWORK WITH ACCURACY
epoch = 15
# CHECK IF GPUS ARE PRESENT
gpus = mxnet.test_utils.list_gpus()
ctx =  [mxnet.gpu()] if gpus else [mxnet.cpu(0), mxnet.cpu(1)]
handwritten_net.initialize(mxnet.init.Xavier(), ctx=ctx, force_reinit=True)
trainer = gluon.Trainer(handwritten_net.collect_params(), 'sgd', {'learning_rate': 0.02})
# Use Accuracy as the evaluation metric.
metric = mxnet.metric.Accuracy()
softmax_cross_entropy_loss = gluon.loss.SoftmaxCrossEntropyLoss()

'''
# Ask the profiler to start recording
profiler.set_state('run')
'''

for i in range(epoch):
    # Reset the train data iterator.
    train_data.reset()
    # Loop over the train data iterator.
    for batch in train_data:
        # Splits train data into multiple slices along batch_axis
        # and copy each slice into a context.
        data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
        # Splits train labels into multiple slices along batch_axis
        # and copy each slice into a context.
        label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
        outputs = []
        # Inside training scope
        with autograd.record():
            for x, y in zip(data, label):
                z = handwritten_net(x)
                # Computes softmax cross entropy loss.
                loss = softmax_cross_entropy_loss(z, y)
                # Backpropogate the error for one iteration.
                loss.backward()
                outputs.append(z)
        # Updates internal evaluation
        metric.update(label, outputs)
        # Make one step of parameter update. Trainer needs to know the
        # batch size of data to normalize the gradient by 1/batch_size.
        trainer.step(batch.data[0].shape[0])
    # Gets the evaluation result.
    name, acc = metric.get()
    # Reset evaluation result to initial state.
    metric.reset()
    print('training acc at epoch %d: %s=%f'%(i, name, acc))
'''
```

Keras

```
In [18]: EPOCHS = 15
BATCH_SIZE = 200
```

```
In [19]: X_train, y_train = train['features'], to_categorical(train['labels'])
X_validation, y_validation = validation['features'], to_categorical(validation['labels'])

train_generator = ImageDataGenerator().flow(X_train, y_train, batch_size=BATCH_SIZE)
validation_generator = ImageDataGenerator().flow(X_validation, y_validation, batch_size=BATCH_SIZE)
```

```
In [20]: print('# of training images:', train['features'].shape[0])
print('# of validation images:', validation['features'].shape[0])

steps_per_epoch = X_train.shape[0]//BATCH_SIZE
validation_steps = X_validation.shape[0]//BATCH_SIZE

#log_dir = "logs/profile/{}".format(time())
#tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = log_dir, histogram_freq=1, profile_batch = 3)

model.fit_generator(train_generator, steps_per_epoch=steps_per_epoch, epochs=EPOCHS,
                    validation_data=validation_generator, validation_steps=validation_steps)
                   # shuffle=True, callbacks=[tensorboard_callback])
```

Accuracy Results

BDAPro
@DIMA

GLUON

```
mxnet.nd.waitall()
# Ask the profiler to stop recording
profiler.set_state('stop')
# Dump all results to log file before download
profiler.dump()
'''
```

```
training acc at epoch 0: accuracy=0.553133
training acc at epoch 1: accuracy=0.883817
training acc at epoch 2: accuracy=0.912917
training acc at epoch 3: accuracy=0.927500
training acc at epoch 4: accuracy=0.937967
training acc at epoch 5: accuracy=0.944550
training acc at epoch 6: accuracy=0.949750
training acc at epoch 7: accuracy=0.954333
training acc at epoch 8: accuracy=0.956900
training acc at epoch 9: accuracy=0.960633
training acc at epoch 10: accuracy=0.963350
training acc at epoch 11: accuracy=0.965267
training acc at epoch 12: accuracy=0.967050
training acc at epoch 13: accuracy=0.968633
training acc at epoch 14: accuracy=0.970317
```

Out[6]:
```
"\n# Make sure all operations have completed\nmxnet.nd.waitall()\n# Ask the profiler to stop recording\nprofile
r.set_state('stop')\n# Dump all results to log file before download\nprofiler.dump()  \n"
```

In [7]:
```
# TEST THE NETWORK
metric = mxnet.metric.Accuracy()
# Reset the validation data iterator.
val_data.reset()
# Loop over the validation data iterator.
for batch in val_data:
    # Splits validation data into multiple slices along batch_axis
    # and copy each slice into a context.
    data = gluon.utils.split_and_load(batch.data[0], ctx_list=ctx, batch_axis=0)
    # Splits validation label into multiple slices along batch_axis
    # and copy each slice into a context.
    label = gluon.utils.split_and_load(batch.label[0], ctx_list=ctx, batch_axis=0)
    outputs = []
    for x in data:
        outputs.append(handwritten_net(x))
    # Updates internal evaluation
    metric.update(label, outputs)
print('validation acc: %s=%f'%metric.get())
assert metric.get()[1] > 0.90
```

```
validation acc: accuracy=0.970400
```

Keras

In [18]:
```
EPOCHS = 15
BATCH_SIZE = 200
```

In [19]:
```
X_train, y_train = train['features'], to_categorical(train['labels'])
X_validation, y_validation = validation['features'], to_categorical(validation['labels'])

train_generator = ImageDataGenerator().flow(X_train, y_train, batch_size=BATCH_SIZE)
validation_generator = ImageDataGenerator().flow(X_validation, y_validation, batch_size=BATCH_SIZE)
```

In [20]:
```
print('# of training images:', train['features'].shape[0])
print('# of validation images:', validation['features'].shape[0])

steps_per_epoch = X_train.shape[0]//BATCH_SIZE
validation_steps = X_validation.shape[0]//BATCH_SIZE

#log_dir = "logs/profile/{}".format(time())
#tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = log_dir, histogram_freq=1, profile_batch = 3)

model.fit_generator(train_generator, steps_per_epoch=steps_per_epoch, epochs=EPOCHS,
                    validation_data=validation_generator, validation_steps=validation_steps)
                   # shuffle=True, callbacks=[tensorboard_callback])
```

```
# of training images: 48000
# of validation images: 12000
Epoch 1/15
240/240 [==============================] - 98s 408ms/step - loss: 0.5154 - accuracy: 0.9023 - val_loss: 0.0899
- val_accuracy: 0.9718
Epoch 2/15
240/240 [==============================] - 92s 385ms/step - loss: 0.0821 - accuracy: 0.9756 - val_loss: 0.0758
- val_accuracy: 0.9776
Epoch 3/15
240/240 [==============================] - 68s 282ms/step - loss: 0.0563 - accuracy: 0.9826 - val_loss: 0.0572
- val_accuracy: 0.9825
Epoch 4/15
240/240 [==============================] - 70s 292ms/step - loss: 0.0409 - accuracy: 0.9875 - val_loss: 0.0682
- val_accuracy: 0.9788
Epoch 5/15
240/240 [==============================] - 80s 333ms/step - loss: 0.0331 - accuracy: 0.9890 - val_loss: 0.0571
- val_accuracy: 0.9829
Epoch 6/15
240/240 [==============================] - 95s 395ms/step - loss: 0.0275 - accuracy: 0.9911 - val_loss: 0.0570
- val_accuracy: 0.9835
Epoch 7/15
240/240 [==============================] - 97s 406ms/step - loss: 0.0242 - accuracy: 0.9919 - val_loss: 0.0524
- val_accuracy: 0.9854
Epoch 8/15
```

GLUON

Keras

BDAPro
@DIMA

	5 epochs			10 epochs			15 epochs		
MXnet	#	Time	Accuracy	#	Time	Accuracy	#	Time	Accuracy
	1	19.4	0.312883	1	24.68	0.560933	1	23.91	0.553133
	2	17.16	0.816533	2	27.1	0.882233	2	20.29	0.883817
	3	34.13	0.897083	3	21.48	0.9128	3	19.82	0.912917
	4	16.96	0.91745	4	20.09	0.9284	4	19.96	0.9275
	5	1.04	0.932433	5	20.34	0.93835	5	24.16	0.937967
	Total	128.71		6	20.27	0.9458	6	23.3	0.94455
	validation		0.9336	7	20.67	0.950633	7	18.58	0.94975
				8	20.16	0.954017	8	19.61	0.954333
				9	30.3	0.957183	9	20.18	0.9569
				10	10.31	0.960133	10	20.69	0.960633
				Total	215.4		11	19.8	0.960633
				validation		0.9602	12	21.12	0.96335
							13	19.49	0.96705
							14	19.97	0.968633
							15	14.99	0.970317
							Total	305.87	
							validation		0.9704
TF	#	Time	Accuracy	#	Time	Accuracy	#	Time	Accuracy
	1	68	0.8415	1	90	0.8959	1	98	0.9023
	2	75	0.9756	2	101	0.9753	2	92	0.9756
	3	84	0.9827	3	98	0.9826	3	68	0.9826
	4	65	0.9864	4	83	0.9867	4	70	0.9875
	5	67	0.9899	5	93	0.9885	5	80	0.989
	Total	359		6	99	0.9905	6	95	0.9911
	validation		0.9847	7	101	0.9926	7	97	0.9919
				8	105	0.9935	8	101	0.9945
				9	100	0.9936	9	78	0.9944
				10	100	0.9953	10	76	0.9957
				Total	970		11	93	0.9951
				validation		0.9857	12	100	0.9955
							13	78	0.9952
							14	75	0.997
							15	77	0.9964
							Total	1278	
							validation		0.9863

GLUON

Keras

BDAPro
@DIMA

	5 epochs			10 epochs			15 epochs		
MXnet	#	Time	Accuracy	#	Time	Accuracy	#	Time	Accuracy
	1	19.4	0.312883	1	24.68	0.560933	1	23.91	0.553133
	2	17.16	0.816533	2	27.1	0.882233	2	20.29	0.883817
	3	34.13	0.897083	3	21.48	0.9128	3	19.82	0.912917
	4	16.96	0.91745	4	20.09	0.9284	4	19.96	0.9275
	5	1.04	0.932433	5	20.34	0.93835	5	24.16	0.937967
	Total	128.71		6	20.27	0.9458	6	23.3	0.94455
	validation		0.9336	7	20.67	0.950633	7	18.58	0.94975
				8	20.16	0.954017	8	19.61	0.954333
				9	30.3	0.957183	9	20.18	0.9569
				10	10.31	0.960133	10	20.69	0.960633
				Total	215.4		11	19.8	0.960633
				validation		0.9602	12	21.12	0.96335
							13	19.49	0.96705
							14	19.97	0.968633
							15	14.99	0.970317
							Total	305.87	
							validation		0.9704
TF	#	Time	Accuracy	#	Time	Accuracy	#	Time	Accuracy
	1	68	0.8415	1	90	0.8959	1	98	0.9023
	2	75	0.9756	2	101	0.9753	2	92	0.9756
	3	84	0.9827	3	98	0.9826	3	68	0.9826
	4	65	0.9864	4	83	0.9867	4	70	0.9875
	5	67	0.9899	5	93	0.9885	5	80	0.989
	Total	359		6	99	0.9905	6	95	0.9911
	validation		0.9847	7	101	0.9926	7	97	0.9919
				8	105	0.9935	8	101	0.9945
				9	100	0.9936	9	78	0.9944
				10	100	0.9953	10	76	0.9957
				Total	970		11	93	0.9951
				validation		0.9857	12	100	0.9955
							13	78	0.9952
							14	75	0.997
							15	77	0.9964
							Total	1278	
							validation		0.9863

LeNet Comparison - Summary

BDAPro
@DIMA

TensorFlow - Keras

- **More accuracy**
- Takes way more time to train
- **Faster convergence**
- More declarative
- Easier to implement
- Extensive documentation
- Wider community
- Minor reproducibility issues

MXNet - Gluon

- Slightly less accuracy
- **More Speed in training**
- **'Slower' convergence**
- Less declarative
- 'harder' to implement
- Fewer documentation
- Smaller community
- **Reprodicibility issues**

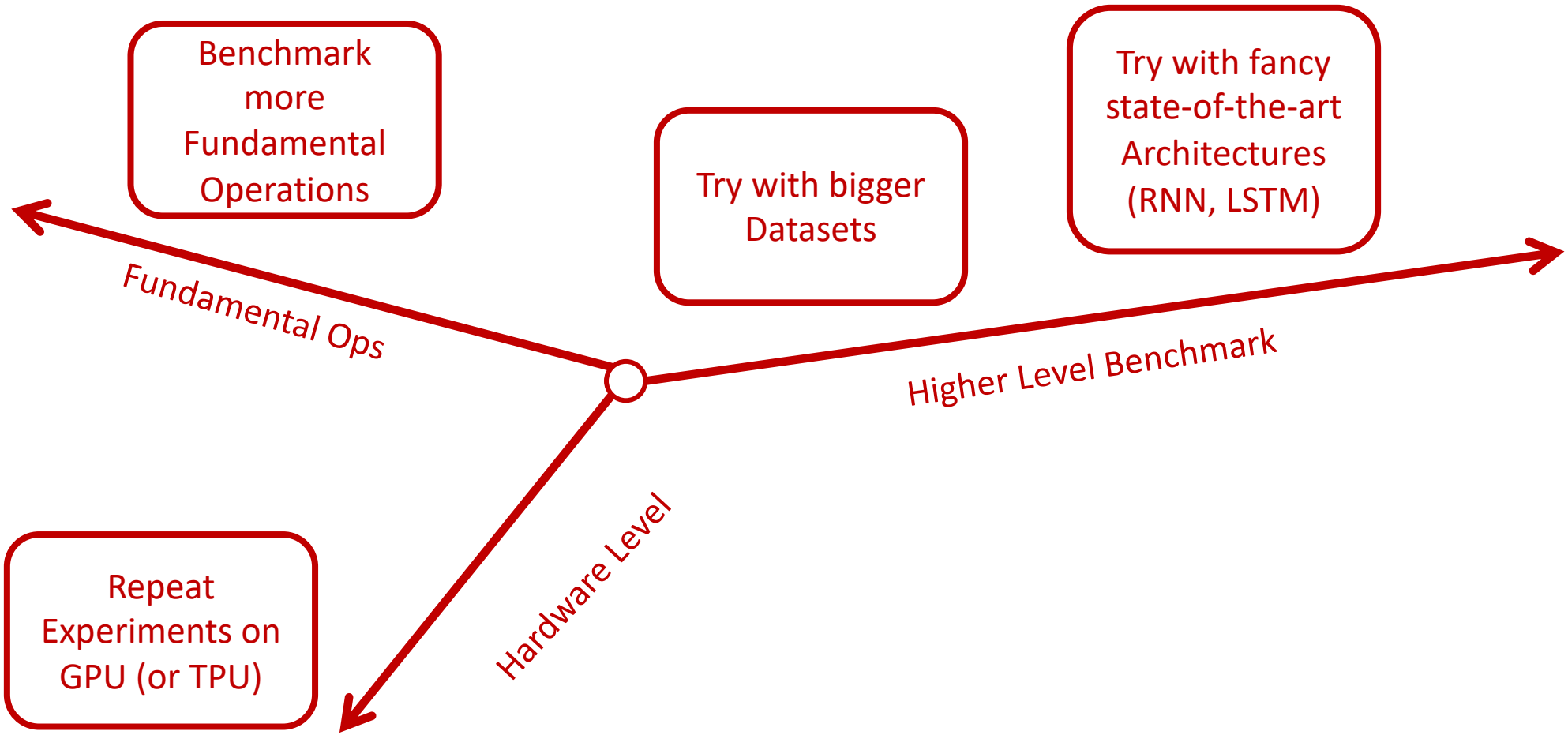
6. Summary – Future Work

Summary

BDAPro
@DIMA

- Two architectures are very similar in terms of high level features
- Same mixed paradigm approach: declarative (more efficient) and imperative (more intuitive)
- Very different in terms of performance speed (on CPU) both for fundamental operations and complete training
- Two huge documentations: TF bigger but not well organized, Mxnet smaller but in constant expansion with high quality resources.

Future Work – Possible Directions



Questions?

Suggestions?

Thank you!

x. Porting Rysia Framework to TensorFlow 2.0

Paper: End-to-End Benchmarking of Deep Learning Platforms

Rysia:
Declarative Benchmarking Framework

BDAPro
@DIMA

Blueprint

1. Define the network (representative workload)

Rysia

2. Translate into fundamental operations

TensorFlow

mxnet

PYTORCH

Operator
Matrix-Multiplication
Convolution (2D)
Bias Addition
Max Pooling (2D)

amazon web services™

3. Execute and get metrics

Porting

Major Release

1.12.2 -> **2.0.0**

TensorFlow

1.4.1 -> **1**.5.1

mxnet

1.12.2

```
W = tf.Variable(
  tf.glorot_uniform_initializer()(
    (10, 10)))
b = tf.Variable(tf.zeros(10))
c = tf.Variable(0)

x = tf.placeholder(tf.float32)
ctr = c.assign_add(1)
with tf.control_dependencies([ctr]):
  y = tf.matmul(x, W) + b
init =
  tf.global_variables_initializer()

with tf.Session() as sess:
  sess.run(init)
  print(sess.run(y,
  feed_dict={x: make_input_value()}))
  assert int(sess.run(c)) == 1
```

2.0.0

```
W = tf.Variable(
  tf.glorot_uniform_initializer()(
    (10, 10)))
b = tf.Variable(tf.zeros(10))
c = tf.Variable(0)

@tf.function
def f(x):
  c.assign_add(1)
  return tf.matmul(x, W) + b

print(f(make_input_value())
assert int(c) == 1
```