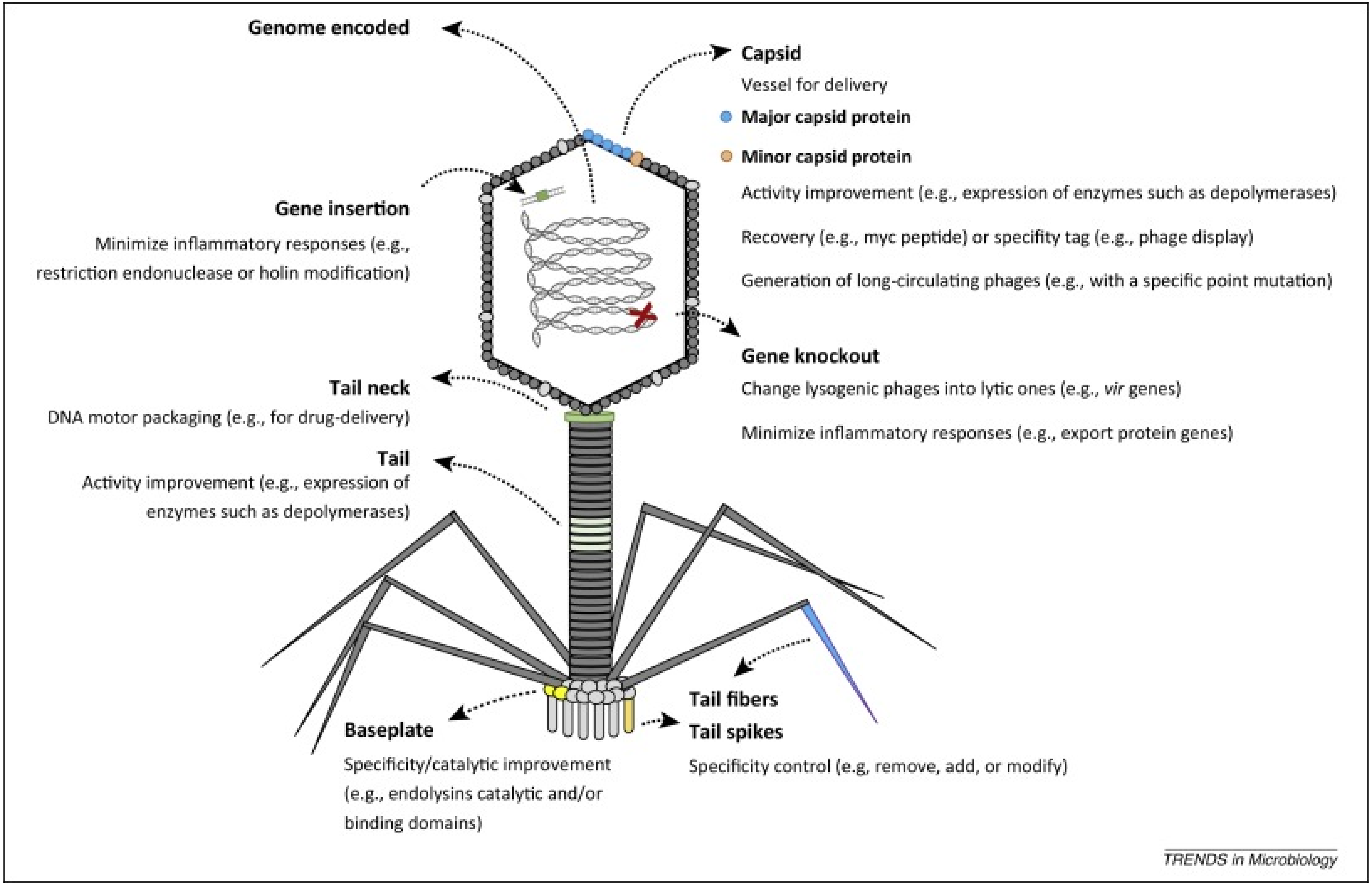
# Hyper tuning of meta parameters in Artificial neural networks to predict phage proteins structural class using 2D subspace optimization

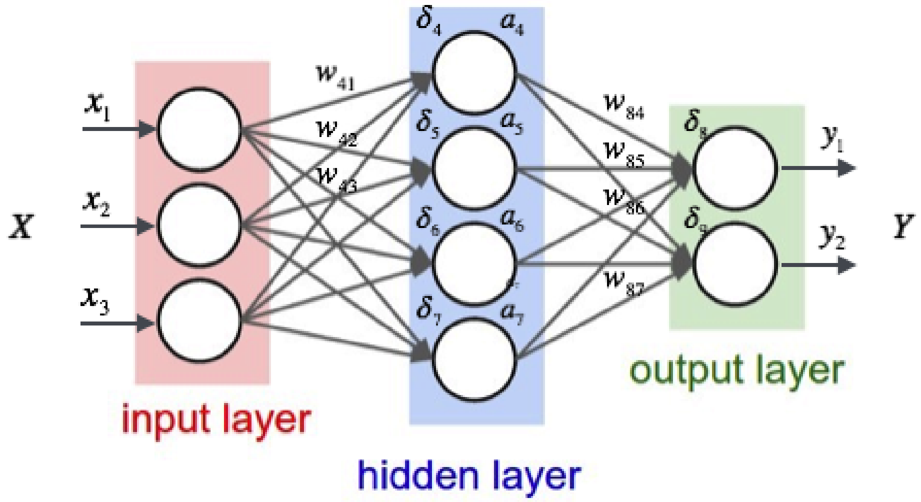
Vito Adrian Cantu
M693a - Fall 2018


Genome encoded
Capsid
Vessel for delivery
Major capsid protein
Minor capsid protein
Activity improvement (e.g., expression of enzymes such as depolymerases)
Recovery (e.g., myc peptide) or specifity tag (e.g., phage display)
Generation of long-circulating phages (e.g., with a specific point mutation)
Gene insertion
Minimize inflammatory responses (e.g., restriction endonuclease or holin modification)
Gene knockout
Change lysogenic phages into lytic ones (e.g., *vir* genes)
Minimize inflammatory responses (e.g., export protein genes)
Tail neck
DNA motor packaging (e.g., for drug-delivery)
Tail
Activity improvement (e.g., expression of enzymes such as depolymerases)
Tail fibers
Tail spikes
Specificity control (e.g, remove, add, or modify)
Baseplate
Specificity/catalytic improvement (e.g., endolysins catalytic and/or binding domains)
TRENDS in Microbiology

```
>AAA32580_1
MFGAIAGGIASALAGGAMSKLFGGGQKAASGGIQGDVLATDNNTVGMGDAGIKSAIQGSNVPNPDEAAPS
FVSGAMAKAGKGLLEGTLQAGTSAVSDKLLDLVGLGGKSAADKGKDTRDYLAAAFPELNAWERAGADASS
AGMVDAGFENQKELTKMQLDNQKEIAEMQNETQKEIAGIQSATSRQNTKDQVYAQNEMLAYQQKESTARV
ASIMENTNLSQQQQVSEIMRQMLTQAQTAGQYFTNDQIKEMTRKVSAEVDLVHQQTQNQRYGSSHIGATA
KDISNVVTDAASGVVDIFHGIDKAVADTWNNFWKDGKADGIGSNLSRK
>AAA32580_2
MFGAIAGGIASALAGGAMSKLFGGGQKAASGGIQGDVLATDNNTVGMGDAGIKSAIQGSNVPNPDEAAPS
FVSGAMAKAGKGLLEGTLQAGTSAVSDKLLDLVGLGGKSAADKGKDTRDYLAAAFPELNAWERAGADASS
AGMVDAGFENQKELTKMQLDNQKEIAEMQNETQKEIAGIQSATSRQNTKDQVYAQNEMLAYQQKESTARV
ASIMENTNLSKQQQVSEIMRQMLTQAQTAGQYFTNDQIKEMTRKVSAEVDLVHQQTQNQRYGSSHIGATA
KDISNVVTDAASGVVDIFHGIDKAVADTWNNFWKDGKADGIGSNLSRK
>AAA32580_3
MFGAIAGGIASALAGGAMSKLFGGGQKAASGGIQGDVLATDNNTVGMGDAGIKSAIQGSNVPNPDEAAPS
FVSGAMAKAGKGLLEGTLQAGTSAVSDKLLDLVGLGGKSAADKGKDTRDYLAAAFPELNAWERAGADASS
AGMVDAGFENQKELTKMQLDNQKEIAEMQNETQKEIAGIQSATSRQNTKDQVYAQNEMLAYQQKESTARV
ASIMENTNLSKQQQVSEIMRQMLTQAQTAGQYFTNDQIKEMTRKVVAEVDLVHQQTQNQRYGSSHIGATA
KDISNVVTDAASGVVDIFHGIDKAVADTWNNFWKDGKADGIGSNLSRK
>AAA32580_4
MFGAIAGGIASALAGGAMSKLFGGGQKAASGGIQGDVLATDNNTVGMGDAGIKSAIQGSNVPNPDEAAPS
FVSGAMAKAGKGLLEGTLQAGTSAVSDKLLDLVGLGGKSAADKGKDTRDYLAAAFPELNAWERAGADASS
AGMVDAGFENTKELTKMQLDNQKEIAEMQNETQKEIAGIQSATSRQNTKDQVYAQNEMLAYQQKESTARV
ASIMENTNLSKQQQVSEIMRQMLTQAQTAGQYFTNDQIKEMTRKVSAEVDLVHQQTQNQRYGSSHIGATA
KDISNVVTDAASGVVDIFHGIDKAVADTWNNFWKDGKADGIGSNLSRK
```

| Function | Dereplicated by FastGroup | # of Seqs | Encoding Functions to 10 Label Neurons | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| major capsid | √ | 3,793 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| minor capsid | | 1,544 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| baseplate | √ | 4,227 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| major tail | √ | 1,851 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 |
| minor tail | √ | 1,536 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 |
| portal | √ | 3,110 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| tail fiber, major | √ | 3,213 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 |
| tail shaft,sheath | √ | 1,818 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 |
| collar | √ | 1,546 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 |
| head-tail joining | | 3,037 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |

# Artificial Neural Networks




ANN have been shown to be universal approximators of **continuous** functions on compact subsets of $R^n$

# My function...


-Learning rate
-3x dropout rate

-Inaccuracy
-Gain matrix norm

# The ANN

```
In [4]: def evaluate_f(lr=0.001, dd1=0.2, dd2=0.2, dd3=0.2, vv=0):
            opt=Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
            #opt=SGD(lr=lr, momentum=0.0, decay=0.0, nesterov=False)
            model = Sequential()
            model.add(Dense(402, input_dim=402, kernel_initializer='random_uniform',activation='relu'))
            model.add(Dropout(dd1))
            model.add(Dense(200,activation='relu'))
            model.add(Dropout(dd2))
            model.add(Dense(200,activation='relu'))
            model.add(Dropout(dd3))
            model.add(Dense(10,activation='softmax'))
            model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
            hist=model.fit(train_X, train_Y,verbose=vv, epochs=50, batch_size=200)
            scores = model.evaluate(test_X, test_Y, verbose=vv)
            if vv:
                print("Accuracy: %.2f%%" % (scores[1]*100))
            return 1-scores[1]
            #return hist;

        def evaluate_f_v(VV):
            val=evaluate_f(lr=VV[0], dd1=VV[1], dd2=VV[2], dd3=VV[3], vv=0)
            return val
```

# The computer

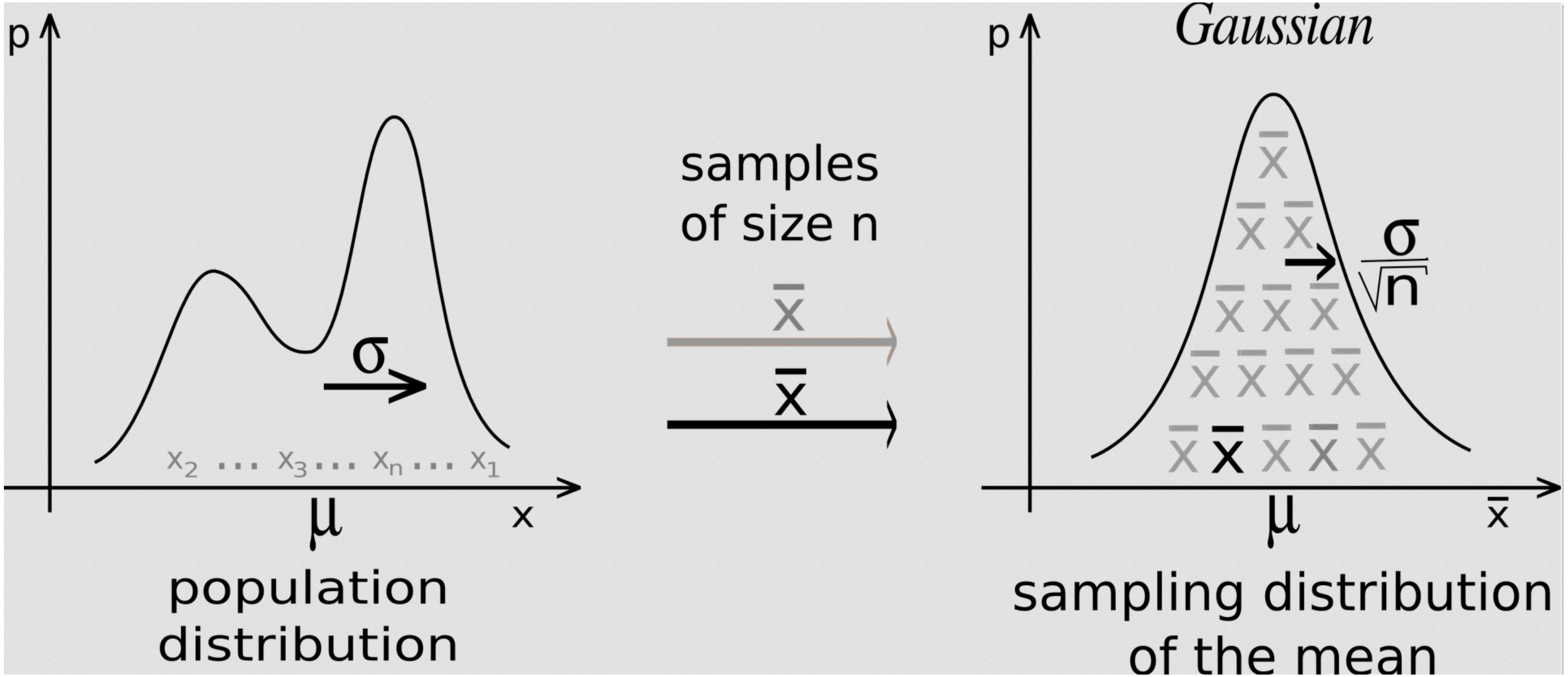
In [3]:
```
#this list the devices, just making sure there is a GPU present, you might be fine with no GPU
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 9356401667059120052
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 15595618304
locality {
  bus_id: 1
  links {
  }
}
incarnation: 7516925211051587892
physical_device_desc: "device: 0, name: Tesla V100-SXM2-16GB, pci bus id: 0000:00:1e.0, compute capability: 7.0"
]
```

| **Easier** | **Harder** |
| --- | --- |
| *Unconstrained* | *Constrained* |
| *Continuous* | *Discrete* |
| *Local Optimization* | *Global Optimization* |
| *Deterministic* | *Stochastic* |
| *Convex* | *Non-Convex* |

**Table:** *Summary of some factors impacting the difficulty of the optimization problem.*

# Evaluating F(x)



# Getting derivatives

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)_{i,j} = \frac{\left(\frac{\partial u}{\partial y}\right)_{i+1,j} - \left(\frac{\partial u}{\partial y}\right)_{i-1,j}}{2\Delta x} + \mathcal{O}(\Delta x)^2$$

$$\left(\frac{\partial u}{\partial y}\right)_{i+1,j} = \frac{u_{i+1,j+1} - u_{i+1,j-1}}{2\Delta y} + \mathcal{O}(\Delta y)^2$$

$$\left(\frac{\partial u}{\partial y}\right)_{i-1,j} = \frac{u_{i-1,j+1} - u_{i-1,j-1}}{2\Delta y} + \mathcal{O}(\Delta y)^2$$

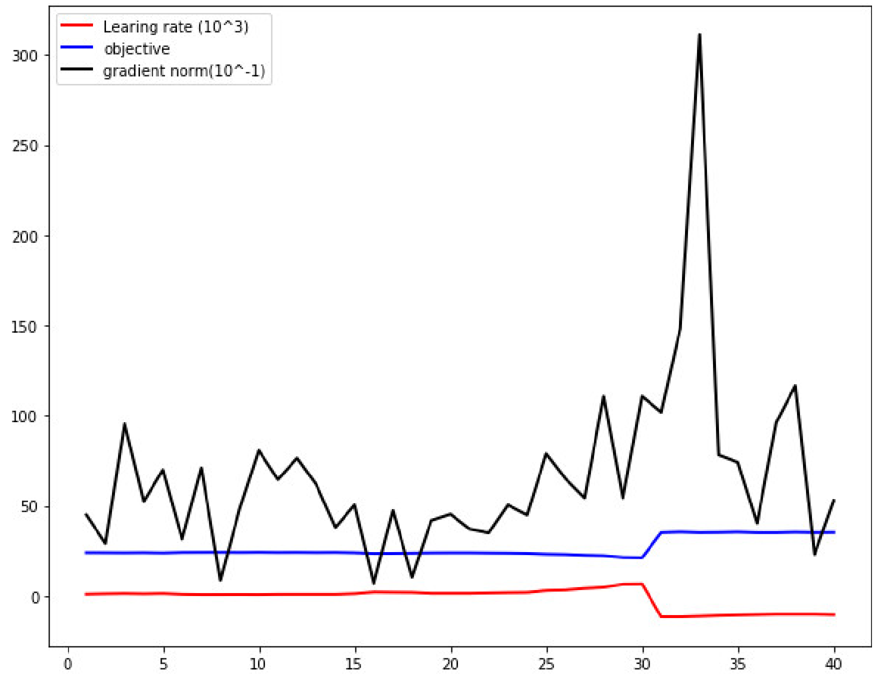
$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)_{i,j} = \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{4\Delta x \Delta y} + \mathcal{O}[(\Delta x)^2, (\Delta y)^2]$$

# Number of calls to F(x)

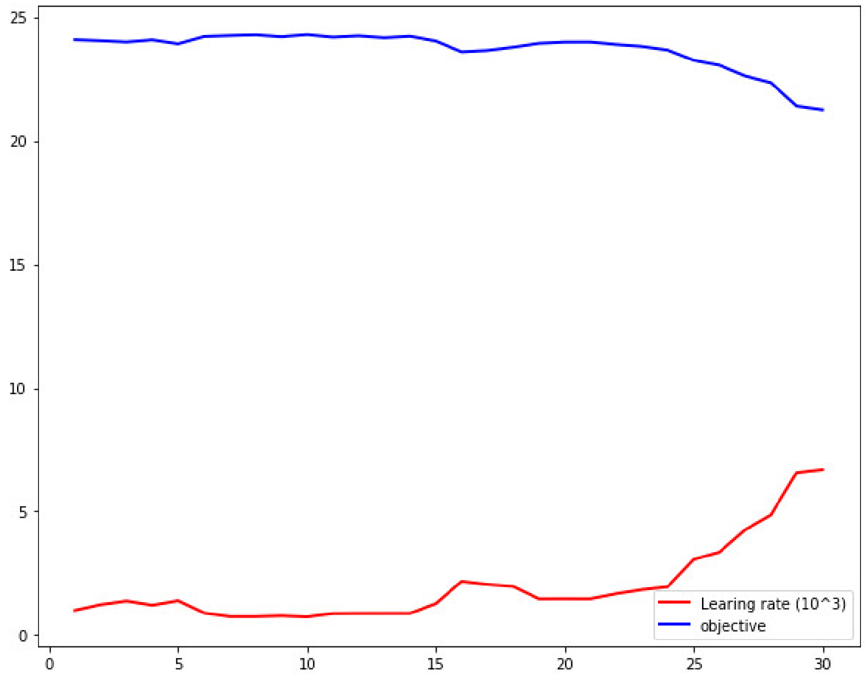
- F=1*5
- G=8*5
- H=32*5
- Total =~ 42*5

```
In [15]: XX2=numpy.array([0.001,0.2,0.2,0.2])
         #GG_i=GG

In [16]: %%time
         FF=avg_evaluate_f_v(XX2)

CPU times: user 1min 48s, sys: 6.07 s, total: 1min 55s
Wall time: 58.5 s

In [17]: %%time
         GG_i=evaluate_G_v(XX2)

CPU times: user 15min 41s, sys: 35.8 s, total: 16min 16s
Wall time: 8min 43s

In [19]: %%time
         HH=evaluate_H_v(FF,XX2)

CPU times: user 1h 20min 48s, sys: 1min 54s, total: 1h 22min 43s
Wall time: 52min 35s
```

# Did it work?



Learing rate (10^3)
objective
gradient norm(10^-1)
300
250
200
150
100
50
0
0
5
10
15
20
25
30
35
40

Learing rate (10^3)
objective
25
20
15
10
5
0
0
5
10
15
20
25
30

# Wort it?

- On a 1*1*1*1 grid using h=0.001 there are 10^12 points
- If we can converge in less than 1000 steps it will be well worth it

```
XX=pickle.load(open( "2_path.p", "rb" ) )
```

```
print(XX[0,...],' -> ',avg_evaluate_f_acc_v(XX[0,...]))
print(XX[29,...],' -> ',avg_evaluate_f_acc_v(XX[29,...]))
```

```
[0.001  0.2    0.2    0.2   ]  ->  0.5819259260053988
[0.00669244 0.19995767 0.20278899 0.1964307 ]  ->  0.8011851851851851
```

# Things you learn

- Multiplying many numbers give you a big number…
- Don’t leak memory (or beer) into your GPU
- Optimize the right function