Classifying galaxies

PHYS 246 class 7

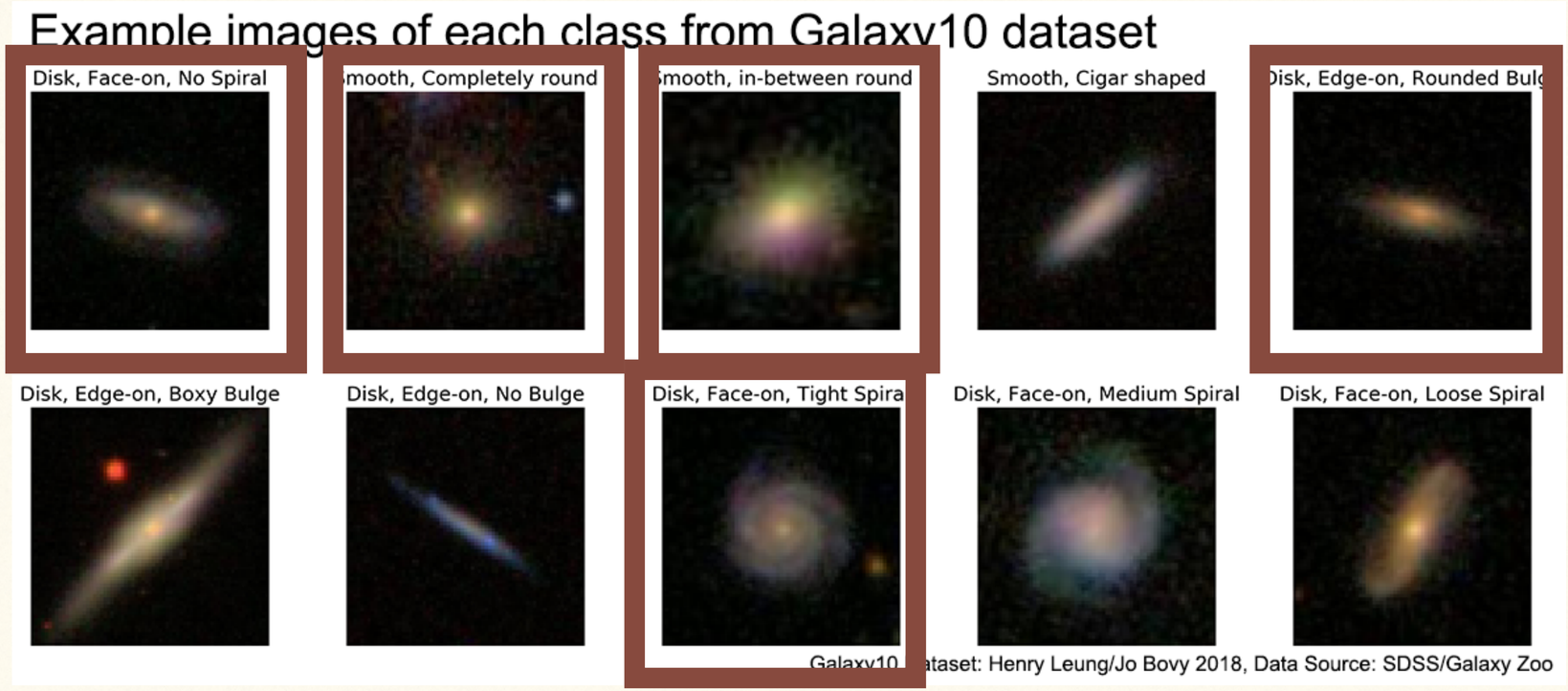
https://lkwagner.github.io/IntroductionToComputationalPhysics/intro.html

Announcements/notes

```
from google.colab import drive
drive.mount('/content/drive')
!cp /content/drive/MyDrive/Colab\ Notebooks/Dynamics.ipynb ./
!jupyter nbconvert --to HTML "Dynamics.ipynb"
```

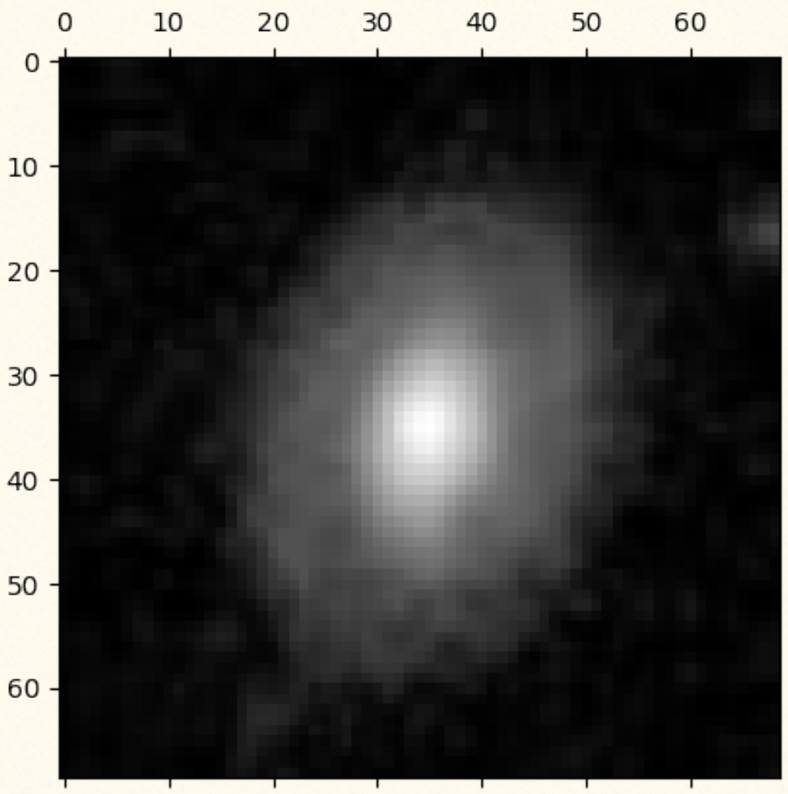
- 'Particle physics' is due tonight.
- I updated the 'Classifying galaxies' notebook. **you must turn in the new version with spring 2025 on the top**. (it's better!)
-

The problem: classifying galaxies



We have down-selected the database to only contain these 5. Representation: [1,0,0,0,0], [0,1,0,0,0]

Our task



69x69 values with value [0,1]

$$f : \mathbb{R}^{4761} \rightarrow \mathbb{R}^{5}$$

[0.70281476,
0.62053204,
0.7327105,
0.3870779,
0.46099216]

5 numbers that represent confidence in each category.

Discussion: How would you parameterize a linear (technically affine) function from $\mathbb{R}^{4761} \to \mathbb{R}^5$? (This will be a building block!)

Answer:

$y = Mx + b$

where

$y, b \in \mathbb{R}^5$

$x \in \mathbb{R}^{4761}$

M is a 5×4761 matrix

A simple neural network: our $f : \mathbb{R}^{4761} \to \mathbb{R}^5$

$v_{in} : 4761$

$W_1 : 100 \times 4761$
$b_1 : 100$

Hidden layer

$y_1 : 100$

$y_1 = \tanh(W_1 v_{in} + b_1)$

$W_1 : 5 \times 100$
$b_2 : 5$

Output

$y_2 : 5$

$y_2 = \dfrac{1}{W_2 y_1 + b_2}$

Normalize output to [0,1]

$a : 5$

$a = \dfrac{1}{1 - \exp(-y_2)}$

Accuracy

Prediction: $a = [0.70281476, 0.62053204, \mathbf{0.7327105}, 0.3870779, 0.46099216]$
Truth: $y = [1. 0. 0. 0. 0.]$

Highest probability is i=2, which is wrong! Return false.

We want to train our model to have the highest accuracy.

Loss function

Prediction: $a = [0.70281476, 0.62053204, 0.7327105, 0.3870779, 0.46099216]$
Truth: $y = [1. 0. 0. 0. 0.]$

Want to have a small loss function when the first number is large:
$-\log(a_0)$
And a large loss function when the other numbers are large:
$\sum_{i>0} \log(1 - a_i)$

General formula:

$\sum_i - y_i \log(a_i) - (1 - y_i)\log(1 - a_i)$: "cross-entropy"

Discussion:
Why not just the number of correct predictions?

Optimizing the loss function

476705 total parameters.

$W_1 : 100 \times 4761$
$b_1 : 100$

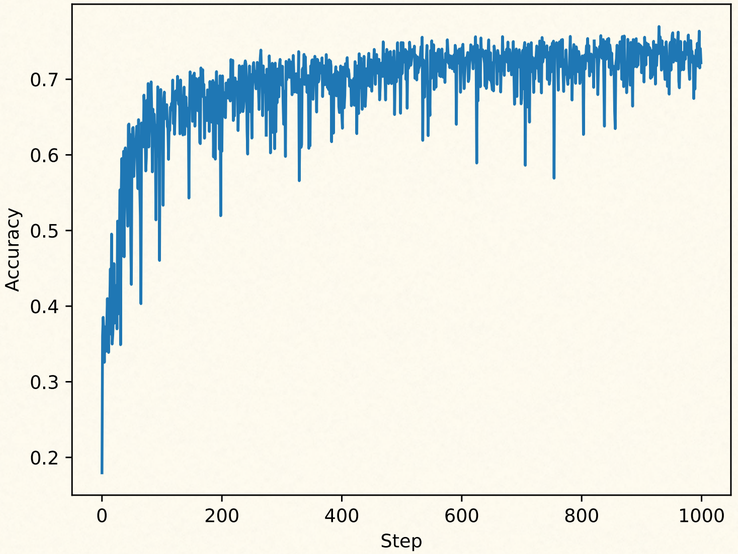
$W_1 : 5 \times 100$
$b_2 : 5$

Second derivative matrix would be 1.7 terabytes of memory.

Stochastic gradient descent:

1. Evaluate the gradient of the loss for ONE random image in the training set
2. Make a tiny change in parameters in the direction of the gradient.
3. Repeat many many times

The reason this works is pretty subtle. It's important that we do not optimize on all images in the set at once; otherwise we will overfit.



Test versus training set: how do you know if we overfit?

Why are test questions different from homework questions?

The test set is reserved and not looked at during training. Your model's performance on the test set will tell you whether

Using jax

Very similar to numpy, but you use jnp instead of np.

You get:
1) just-in-time compilation (speed)
2) automatic execution on GPU
3) automatic differentiation
4) vmap

Requires a functional style
(no global variables, no side effects, can't modify arguments)

```
function = jit(function)

cpu=True
if cpu:
    jax.config.update('jax_platform_name', 'cpu')
#    jax.config.update("jax_enable_x64", True)
else:
    pass

gradient = jax.grad(function)
```

Automatic differentiation (forward mode)

```cpp
#include <iostream>
struct Dual {
   float realPart, infinitesimalPart;
   Dual(float realPart, float infinitesimalPart=0): realPart(realPart),
infinitesimalPart(infinitesimalPart) {}
   Dual operator+(Dual other) {
      return Dual(
         realPart + other.realPart,
         infinitesimalPart + other.infinitesimalPart
      );
   }
   Dual operator*(Dual other) {
      return Dual(
         realPart * other.realPart,
         other.realPart * infinitesimalPart + realPart * other.infinitesimalPart
      );
   }
};
// Example: Finding the partials of z = x * (x + y) + y * y at (x, y) = (2, 3)
Dual f(Dual x, Dual y) { return x * (x + y) + y * y; }
int main () {
   Dual x = Dual(2);
   Dual y = Dual(3);
   Dual epsilon = Dual(0, 1);
   Dual a = f(x + epsilon, y);
   Dual b = f(x, y + epsilon);
   std::cout << "∂z/∂x = " << a.infinitesimalPart << ", "
             << "∂z/∂y = " << b.infinitesimalPart << std::endl;
   // Output: ∂z/∂x = 7, ∂z/∂y = 8
   return 0;
}
```

Suppose

$$g(x, y) = x + y$$
$$f(x, y) = x \cdot y$$

Then

$$h(x, y) = x \cdot (x + y) + y \cdot y = f(x, g(x, y)) + f(x, y)$$

vmap

Allows you to add dimensions to your arrays.

say:
def f(a, **x**):
    return a@**x**

takes in a (matrix) and **x** (vector) and returns a vector

fvmap = jax.vmap(f, in_axes=(None,**0**), out_axes=(0))
takes in a (matrix) and x (N vectors) and returns N vectors