Project 2 - FYSSTK4155

Jacob Alexander Hay

2018-11-13

The Code can be found on my GitHub: `https://github.com/curious-discordian/FYSSTK4155`

Abstract:

I've learned that the abstract is the key elements and findings.

This project and the code linked to above takes (as usual) a slightly different turn. The neural network being lightweight, and the Phase-estimate being by FFT.

Introduction :

A quick motivation, and explanation of the Ising lattice/model; In it's simplest regard, the Ising model is simply a way of modelling how many different atoms in a structure (like a lattice) effect eachother.

This can be anything from charged particles, spin, magnetic properties, etc. to something as simple as a model of how your political inclinations are affected by those closest to you.

The model simply assumes that there is "stuff", and that there is a pattern to how one "stuff" affects other "stuff". In this respect it can be applied to any kind of problem that a physicist may be interested in. (Physics as a field here boils down to the simple assumption that there are patterns in the world around us, and that these patterns can be mathematically described)

With that in mind, our task at hand becomes a bit simpler to parse. We will be focusing on two components; Which things affect which things, and in what way.

To estimate this we introduce an abstract value of Energy, another word we physicists like to use to say that there is a measurable thing that seems to be constant; if something pushes, another thing is pushed, and if you sum it all up then the net effect is nil.

The second thing we deal with is the correlation matrix, where we can look up how one thing affects another, and also neatly read off how much they affect eachother.
That's all.

The Whats:

a: Make the data; an Ising lattice-creation function.

b: Estimates of coupling constants. (using lin-reg)

c: Estimate of phase based on the configurations (order vs disorder)

d: Regression analysis using neural network/backpropogation

e: Phase estimatees using neural network.

f: Final consideration and comparison of the alternatives.

The Hows:

a: We assign an amount of random variables to -1 or 1, and sum these variables up in accord with a default correlation matrix, as in the Project description PDF. Then we do this about 999999 more times to generate data.

b: Using linear regression we estimate the coupling constant in this linear system (it's a daisy-chain of effect, so we can do that).

c: The second goal is to "train" a model to discern whether the temperatures are above a critical point or below, based on the amount of order or disorder in the system. i.e. if there seems to be a lot of correlation then the model is "ordered", and if not then it is disordered. To do this we need a function to read the data as well.

d: We should then write a (central) neural network algorithm that is comparable to the linear regression, i.e. it should discern the correlation coefficient of a 1D ising model.

e: Then the secondary effect should be to extend the correlations to identify whether larger areas of the correlations are coupled, so for this we will want to hook up a fitting cross-entropy classification to the cost of the neural network.

f: Finally we will want to do some critical evaluation of the above portions, and some more nagging from me on how we could discern this in other ways.

Some Results:

Phase-estimates:

Firstly; let's graph something out. The Fourier transform of the two temperatures around the critical point look something like this:

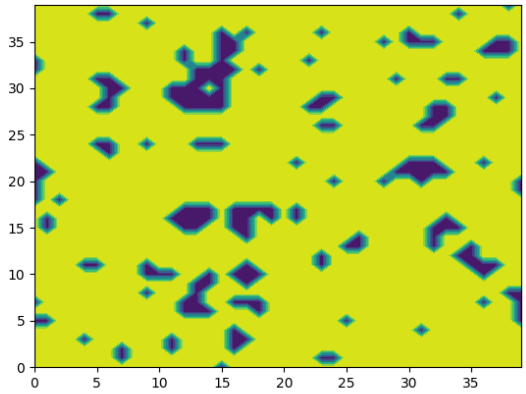

Figure 1: sub-critical $T = 2.25$

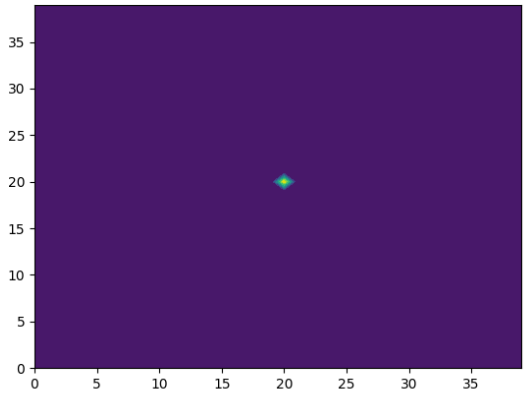

Figure 2: sub-critical FFT $T = 2.25$

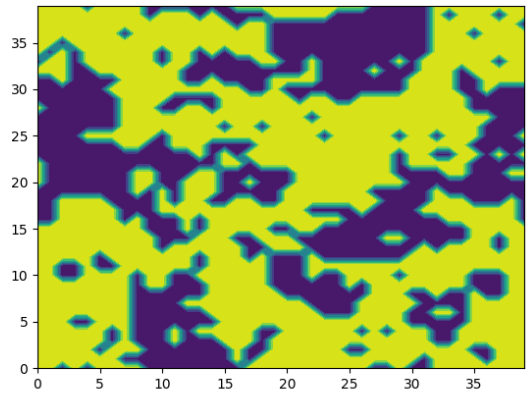

Figure 3: sub-critical $T = 2.5$

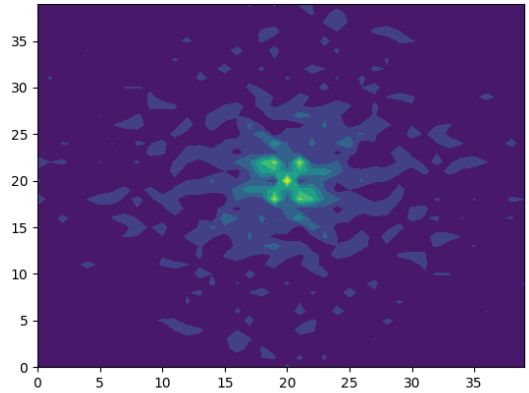

Figure 4: sub-critical FFT $T = 2.5$

And to expand upon this; What you are seeing is not waves, but the coefficients of waves. That is; every step away from the center represents a 2D Fourier-coefficient.

There is also a neat symmetry to these images, which makes them beautiful to look at. The reason for this symmetry is simple; $\cos(t) = \frac{1}{2} \cdot \left(e^{it} + e^{-it}\right)$

Similarly to the normal Fourier transform, the response has a dual component, mirrored in the top and bottom of the spectrum. Essentially it is like it folds around, so any iteration in a positive direction is simultaneously mirrored by one in the negative direction (see the above decomposition of the cosine-function.)

Simply stated: The first possible dot in the center represents the data that is completely uniform. Any response away from that center represents a contribution of higher-order terms, and as such indicates

disorder in the original data.

Network estimates

The target, and the key to, neural networks is that we can train a fairly lightweight neural network to approximate the same levels as seen in the Fourier approximation.

So as mentioned before, the concept is simply to train the network, and then read off the primal key component of the weights at last. The first

Connection to Project 1:

In project 1 I covered some off-topic directions trying to cover and come up with a "clever" way of utilizing wavelets to do regression, instead of projecting into a polynomial space, essentially.

Though the project quickly scaled beyond the scope of the exercise, and therefore ended up as an incomplete submission, the theory covered will be useful here.

Neural networks are in a sense a stochastic wavelet-analysis, combined with backpropogation and linear programming concepts (or dynamic programming). So with this in mind, I'll refer back where appropriate in this project.

In this we've scaled back, and make use of the special case of Fourier Analysis. Especially in the case of estimating the order vs disorder, i.e. phase-transition. The key element is that the Fourier is mappable to polynomials, so we could essentially be doing a polynomial regression in stead.

The deciding factor in using FFT here was simply that I one of the passing images in Project 1 was very similar to the "visualization of the translation invariant probability measure..." image on the wikipedia article for the Ising-model.

That tartan-looking patchwork was enough for me to want to find a "clever" way of applying some similar functionality as earlier in order to generate some nice-looking visuals, and the Fourier transform of a disordered crystal/liquid canvas is something I know will suit the bill.

Note that I don't really consider that to be the scientific standard, but the visual intuition and representation of the data is important in the connection, and will be a whole lot easier for us to recognize quickly. Intuitively; if the data has a pattern, then there is probably another way of representing the data where that pattern is easier seen.

More mathematics.

Before introducing the neural networks, there is an element we need to address; Why do they work ?

Also in sticking with the previous work here, we need to understand a few key elements, so that we can take it apart and reassemble it before comparing to the earlier models we are asked to compare with.

So we now will look to the concept of Linear Programs, in combination with the theory of Perceptrons.

In few words; a linear program, solvable with stuff such as the Simplex alorithm, is a set of sums that are subject to a cost function that is either maximized or minimized.

I suggest reading up on the Simplex Algorithm here; `https://en.wikipedia.org/wiki/Simplex_algorithm`.

One interesting aspect is the inclusion of "slack" variables, which are comparable to introducing a little bit of noise. (In essence they are only there to ensure that we don't end up stagnating, but are otherwise assumed to be "very small")

From linear programming there is another concept that is also important, which is the Theory/Theories of Duality (i.e. Strong or Weak duality). The Duality of a problem is evaluated as being the inverse problem, with a weak duality being one where the convergence of one is not necessarilly the same as the convergence of the other. (i.e. a min/max pair, where maximizing the inverse of a minimization problem does not "meet" at a fixed point.)

The reason for bringing this up is that the Perceptrons are, in essence, convoluted versions of similar problems. With the Neural Networks being series of such, where the constraints are dynamically set. (I could call this Dynamic Programming, but the Linear Problem examples are easier to follow, at least to start with.)

Programs and scripts

```
### ------------------ Phase analysis : ---------------------------------
## By basic Fourier, we should be able to discern the probability that
## We are dealing with a stable/ordered phase.
##
## Please note that the probabilities are not calibrated here, but
## we can think of this as a pseudo-probability.
##
## It seems good enough for the measure of whether something is stable
## or not.

def FFT_analysis(t,show_demo=False):
    """
    Pass a temperature in the valid data-pool.
    optionally show_demo [boolean] to show an example.
    Returns the percentage of samples at that point where the order
    is over 80%.
    (Seemed like a fair place to start)
    """
    def FFT_phase(data):
        # Assuming the data is singular here.
        analysis = np.abs(np.fft.fftshift(np.fft.fft2(data.reshape(L,L))))
        # Note the shift-function, which transposes q1 with q3 and q2 with q4.
        # This is a pretty standard way of dealing with FFT2, as the
        # positive frequencies are on the first half, and the negative on the second.
        analysis = normalize(analysis)
        return analysis

    data_read = read_t(t)
    FFT = np.array([FFT_phase(x.reshape(L,L)) for x in data_read])
    # Now for the hypothesis; it seems like the data is ordered
    # when the max value is over about .8,
    # and unordered if it is not.
    # So let's take the max of each of these, and then
    maxes = np.array([np.max(x) for x in FFT])
    percentage = np.sum(np.where(maxes>.80,1,0))/float(len(maxes))
    if show_demo:
        # Should ensure that the pick is representative of the
        # genereral cases (i.e. pick the mean max-value)
        pick = np.argsort(maxes)[len(maxes)//2] #index of median value
        data = data_read[pick]
        plt.figure()
        plt.subplot(211)
        plt.contourf(data.reshape(L,L))
        xx,yy = np.meshgrid(np.linspace(-L/2,L/2,L),np.linspace(-L/2,L/2,L))
        plt.subplot(212)
        plt.contourf(xx,yy,FFT[pick])

        plt.show()

    return percentage
```