Landscape Net

An Application of the Landscape Theory in Machine Learning for Eigenvalue Counting

Li Chen
Joint work with Wei Wang

MIT

May 6, 2020

Introduction

Code available at
`https://github.com/nehcili/Wave-Localization`

Mathematical Setting

- Space: $L^2(\mathbb{Z} \cap [0, 1000])$.
- Hamiltonian: $-\Delta_{\mathbb{Z}} + V$ with periodic boundary condition.
- Properties of V:
  1. $V(x)$ are iid distributions for $x \in \mathbb{Z} \cap [0, 1000]$,
  2. $V \geq 0$

Objective

Accurately and efficiently approximate the eigenvalue counting

$$N_V(E) = \# \text{ of eigenvalues of } -\Delta_{\mathbb{Z}} + V \text{ less or equal to } E$$

Introduction

Method

We develop a neural net architecture which attempts to generalize the landscape box counting (c.f. David, G., Filoche, M., Mayboroda, S.)

Machine Learning

- Suppose that a set of features $x \in \mathbb{R}^n$ determines certain quantity $t \in \mathbb{R}$ via $t = F(x)$ for some F. Assume also that x is generated through some random process.
- We would like to approximate F. We need 3 things:
  1. A loss function to minimize. We choose a (distance) function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ which quantify the error and set
  $$L_0(f) := \mathbb{E}_x[l(f(x), t)]$$
  Think $l(x, y) = (x - y)^2$.
  2. We choose a function space parameterized by $\mathbb{R}^m$
  $$\{f_\theta : \theta \in \mathbb{R}^m\}$$
  on which we minimize L_0. I.e. we select
  $$f_{\theta_0} = \mathrm{argmin}_\theta L_0(f_\theta)$$
  3. Minimization scheme.

Empirical loss

- In practice, given observed training data $(x_1, t_1), \ldots, (x_N, t_N)$, we minimize the empirical error

$$L(f) := \frac{1}{N} \sum_{i=1}^{N} l(f(x_i), t_i))$$

- Common choices of $l(x, y)$:
  1. L^1 norm: mean absolute error
  2. Square of L^2 norm: mean squared error
  3. $(x - y)/y \times 100\%$: mean absolute percent error
  4. $[\log(x/y)]^2$, ...
- When N is large, minimizing L is very difficult numerically. So we will need a suitable minimization scheme.

Architecture: choice of function space

- For a deep neural net, its function spaces consists of functions of the form

$$f_1 \circ f_2 \circ \cdots \circ f_n$$

where each f_i is a simpler function. Each f_i is called a layer

- Two common and simplest layers are
  1. Dense layer
  2. Convolution layer

Dense layer

- input: $x \in \mathbb{R}^n$, output $y \in \mathbb{R}^m$.
- Parameters:
  1. W is a $m \times n$ matrix. The weight matrix
  2. $b \in \mathbb{R}^m$ is the basis vector
  3. An activation function σ (e.g. Heaviside, $\max(0, \cdot)$)
- Action

$$y = \sigma(Ax + b)$$

Convolution layer

- A convolution layer is a dense layer. But sparse.
- input: $x \in \mathbb{R}^n$, output $y \in \mathbb{R}^m$.
- Parameters:
  1. $k, s \in \mathbb{Z} > 0$ are the kernel size and stride, respectively.
  2. A is a $m \times n$ matrix. In the i-th row, all entries but $A_{i,si}, \ldots, A_{i,si+k-1}$ are zero. Moreover, every row of A is equal upto a shift (by integer multiples of s).
  3. $b \in \mathbb{R}^m$ is the basis vector
  4. An activation function σ (e.g. Heaviside, $\max(0, \cdot)$)
- Action

$$y = \sigma(Ax + b)$$

- cartoon: https://github.com/vdumoulin/conv_arithmetic

Training: minimization scheme

- To minimize
$$L(f) := \frac{1}{N}\sum_{i=1}^{N} l(f(x_i), t_i)),$$
we perform gradient descend on θ (from f_θ). In practice, computing the gradient with all training data is costly and a stochastic version is used:
    1. Iterate over training data until convergence (each iteration is an epoch)
    2. For each epoch, divide the training data into batches, where each batch contains a small amount of training data. We iterate over the batches.
    3. For each batch, perform gradient descent.

Result

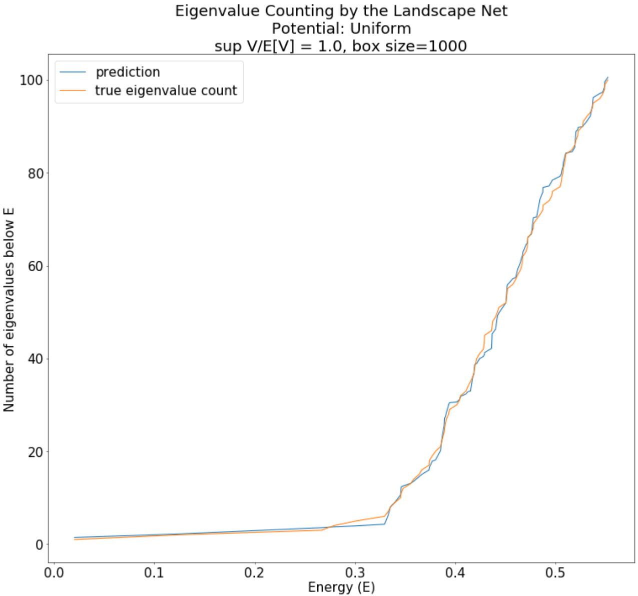
Summary

Training time	Training error	Test error	Trainable Params
1 hour	$\approx$ 1 to 2	$\approx$ 2 to 3	$\approx$ 77,000

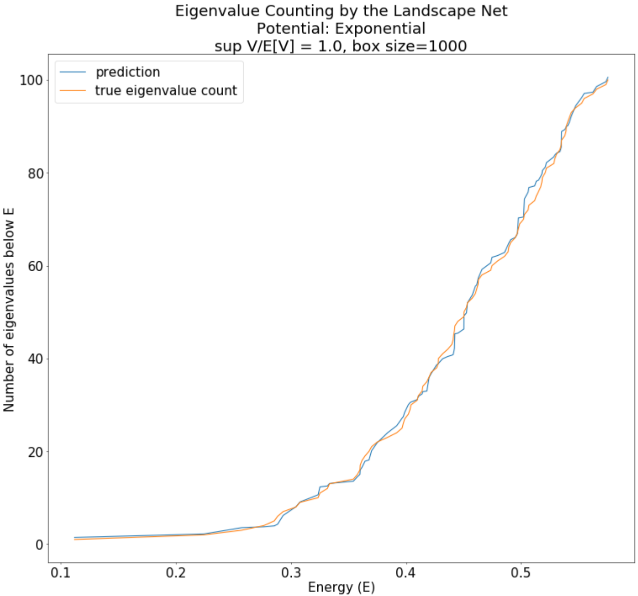
Training/Testing

- We trained 300 sets of data where each set consists of 100 tuples: (features, targets). $N = 300 \times 100$ for error computation.
- A feature is some combinations of the original potential V, the landscape potential W (and its derivative), the energy E. A target is the true eigenvalue counting.
- We tested on 30 sets of independent data. $N = 30 \times 100$ for error computation.

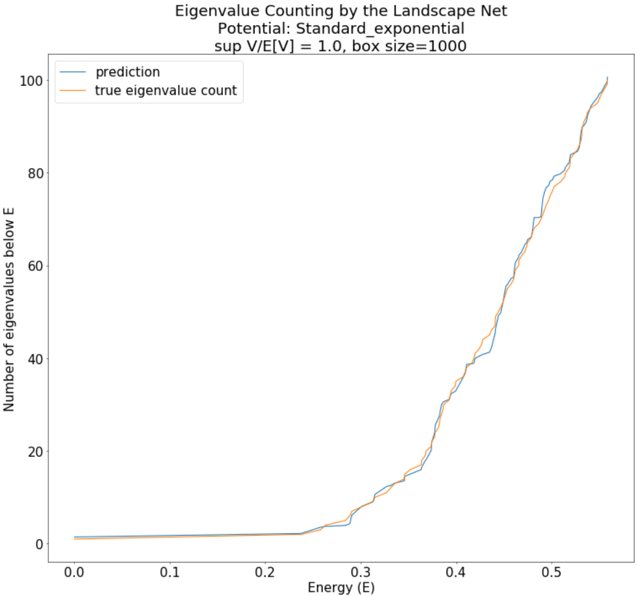
Results


Eigenvalue Counting by the Landscape Net
Potential: Uniform
sup V/E[V] = 1.0, box size=1000
prediction
true eigenvalue count
Number of eigenvalues below E
Energy (E)
0
20
40
60
80
100
0.0
0.1
0.2
0.3
0.4
0.5

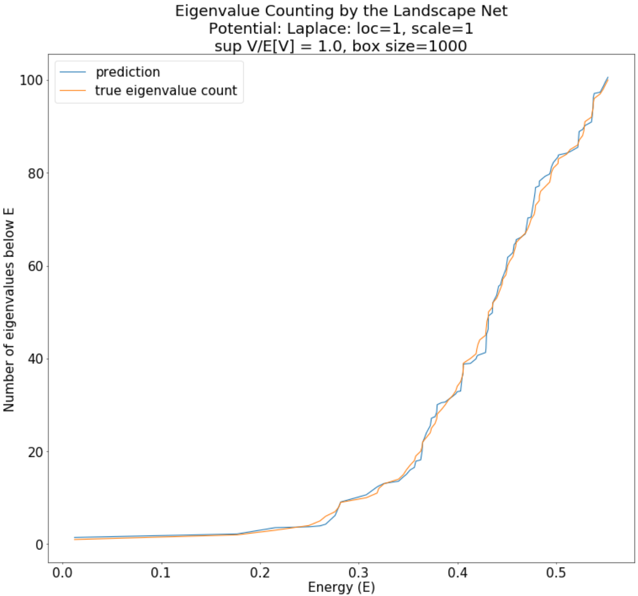
Results


Eigenvalue Counting by the Landscape Net
Potential: Exponential
sup V/E[V] = 1.0, box size=1000
prediction
true eigenvalue count
Number of eigenvalues below E
Energy (E)
0
20
40
60
80
100
0.1
0.2
0.3
0.4
0.5

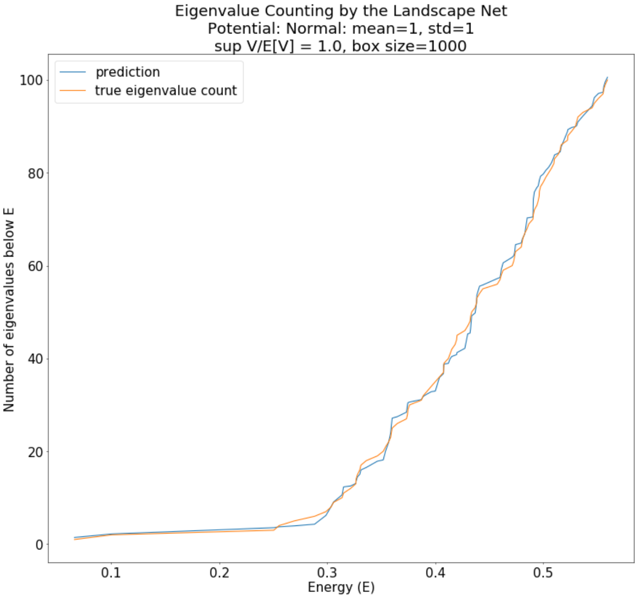
Results


Eigenvalue Counting by the Landscape Net
Potential: Standard_exponential
sup V/E[V] = 1.0, box size=1000
prediction
true eigenvalue count
Number of eigenvalues below E
Energy (E)
0
20
40
60
80
100
0.0
0.1
0.2
0.3
0.4
0.5

Results


Eigenvalue Counting by the Landscape Net
Potential: Laplace: loc=1, scale=1
sup V/E[V] = 1.0, box size=1000
prediction
true eigenvalue count
Number of eigenvalues below E
Energy (E)
0
20
40
60
80
100
0.0
0.1
0.2
0.3
0.4
0.5

Results


Eigenvalue Counting by the Landscape Net
Potential: Normal: mean=1, std=1
sup V/E[V] = 1.0, box size=1000
prediction
true eigenvalue count
Number of eigenvalues below E
Energy (E)
0
20
40
60
80
100
0.1
0.2
0.3
0.4
0.5

Sensitivity of the landscape potential

- This project first started as an attempt to build a neural net to compute the first N eigenvalues given an input potential.

$$\text{potential} \rightarrow N \text{ eigenvalues}$$

- In the same setting, we trained a simple model to predict the first $N = 20$ eigenvalues. The model has the following architecture:

$$\text{input} \mapsto 4 \text{ convolution layers} \mapsto 3 \text{ dense layers} \mapsto \text{output}$$

Sensitivity of the landscape potential

- We trained 2 copies of the simple model with two feature and target pairs: using V vs using W as different feature input while keeping the same target (first 20 true eigenvalues) for each case.
- The result is very dramatic. ($N = 20$ below)

Input	Training time	Training error*	Test error*
V	15 min	$\approx 9\%$	$\approx 10\%$
$W = 1/u$	15 min	$\approx 2\%$	$\approx 2\%$

* error = mean absolute percent error (i.e. $l(x, y) = (x - y)/y \times 100\%$, y is the true target)

Box counting

- Let W denote the landscape potential and $\Omega = [0, L]^d$ or $[0, L]^d \cap \mathbb{Z}$. Let

  $N^W(E) :=$ # of cubes on which $\inf W \leq E$, among all cubes that tile the domain Ω with side length $E^{-1/2}$,

- For suitable potentials, one can prove that (c.f. David, G., Filoche, M., Mayboroda, S., and Zhang, S., Wang, W.)

$$c_1 N^W(c_2 E) \leq N_V(E) \leq C_3 N^W(C_4 E).$$

for suitable constants c_1, c_2, C_3, C_4.

Box counting

- We can use box counting as a regression model with 2 parameters c_1 and c_2:

$$\text{predicted eigenvalue count} = c_1 N^W(c_2 E)$$

- Pros of box counting
  1. Computationally fast. Time complexity = O(time to compute landscape potential).
  2. Scalable: works on arbitrarily large domains. Only has 2 parameters.
  3. Already relatively accurate
- Limitation: when used as a regression model, it only has 2 parameters: under fitting.
- So we would like to add some more parameters and pay a bit more time to get a more accurate result.

Generalizing box counting

- The current box counting can be written as

$$\sum_{\text{cubes}} \Theta(f(W,E)\mid_{\text{cube}}),$$

summed over cubes of side length $E^{-1/2}$ and

1. Θ is the Heaviside function,
2. $f(W,E)\mid_{\text{cube}} = E - \inf_{\text{cube}} W$.

- We generalize this expression as

$$\sum_{\text{cubes}} w_{\text{cube}} G(f(W,E)\mid_{\text{cube}})$$

where the sum is summed over cubes that tile the domain and

1. G is an activation function (e.g. Heaviside, $\max(0,\cdot)$, etc),
2. w_{cube} are weights,
3. f is to be learned from training on data.

- This is nothing but a dense layer in a neural net!

How to train for f?

- Observation: if ρ is the true density such that $N_V(E) = \int \rho$, then by picking
  1. $f(W, E) = \int_{\text{cube}} \rho$
  2. $G(x) = \max(0, x)$
  3. $w_{\text{cube}} = 1$

  our model includes this case. Can we learn something from ρ?
- Yes! If the Hamiltonian is $-\hbar^2 \Delta + V$ (on $L^2(\mathbb{R}^d)$), then

$$
\begin{aligned}
\rho(x) &= \hbar^{-3} \int dp\, 1_{\{p^2 + W(x) - E < 0\}} \text{ (Weyl term)} \\
&+ \hbar^{-1} \sum_i F_i(V(x), W(x), \nabla W(x)) \int dp\, G_i(p^2 + W(x) - E) \\
&+ O(\hbar)
\end{aligned}
$$

  for some local functions F_i and G_i
- We simply follow this format for f.

Decider blocks

- A decider block consists of 3 convolution layers f_1, f_2, f_3 with architecture

$$(x_i, V, W, W') \mapsto f_1(x_i) + f_2(V, W, W') \cdot f_3(x_i) = x_{i+1}$$

  where $x_0 = (V - E, W - E)$.
- We will feed the output of a previous decider block, x_i, to the next decider block as (x_i, V, W, W').
- Then, We place in series several decider blocks to form the decider block layer, which tries to learn f.

Architecture of the landscape net

- Recall that our generalization to box counting iss

$$\sum_{\text{cubes}} w_{\text{cube}} G(f(W,E)\,|_{\text{cube}})$$

- Landscape net. Input $= V, W, E$

input $\rightarrow$ decider blocks layer (this is f)
$\rightarrow$ convolution layers (this is part of the box counting sum)
$\rightarrow$ dense layers (this is part of the box counting sum)
$\rightarrow$ output: prediction for $N_V(E)$

Analysis of the model

- As it stands, the current architecture probably can be scaled up. Since we mainly use convolution layers and such layers with stride s reduces the input dimension by a factor of s. Let N denote the input dimension. We will need $\log_s(N)$ layers, each with $k \geq s$ parameters.
- At each layer there is at most kN operations.
- If we have C channels (i.e. number of parallel layers), we have C^2kN operation per layer since every channel sends signal to every other channel.
- In total, we have a time complexity of $O(C^2kN\log_s(N))$ (compared to the $O(N)$ time complexity of box counting).
- For a very rough perspective. The popular image classifier AlexNet has a parameter size/input dimension ratio of 400. Landscape Net has 77.