torch-foresight

Jan 06, 2020

Modules

This package provides a collection of modules useful for characterizing and predicting the dynamics and performance of neural nets. These consist mostly of novel metrics, derived from fields like theoretical neuroscience and information theory, aimed at helping researchers to better understand how neural networks work. The repository is meant to advance a new "Science of AI" or "Science of Deep Learning" (see neuralnet.science). It currently includes modules for computing:

- Effective Information (paper)

With the following under development:

- Gradient Noise Scale (paper)
- Information Bottleneck (paper)

CHAPTER 1

Effective Information

1.1 Theory

Effective Information captures something about the information contained in the causal structure of a network (realized as a causal diagram). It is defined as:

$$H(\langle W^{\text{out}} \rangle) - \text{EI} = \langle H(W^{\text{out}}) \rangle$$

or equivalently:

$$\text{EI} = \frac{1}{N} \sum_i D_{\text{KL}}[W_i^{\text{out}} \,||\, \langle W_i^{\text{out}} \rangle]$$

How this quantity evolves during the training of artificial neural networks may illustrate important dynamics of the network, such as "phase changes" during training (such as a transition from generalization to overfitting):

Above, we see how the effective information of a single-layer (no hidden layers) neural network evolves during training. There appears to be a phase of rapid growth (coinciding with the fast, early drop in training and test loss), then a period of decay and leveling off (during which the test dataset loss levels off), folllowed by a phase of slow increase (during which training dataset loss decreases, but test dataset loss increases) as the network "overfits" the data.

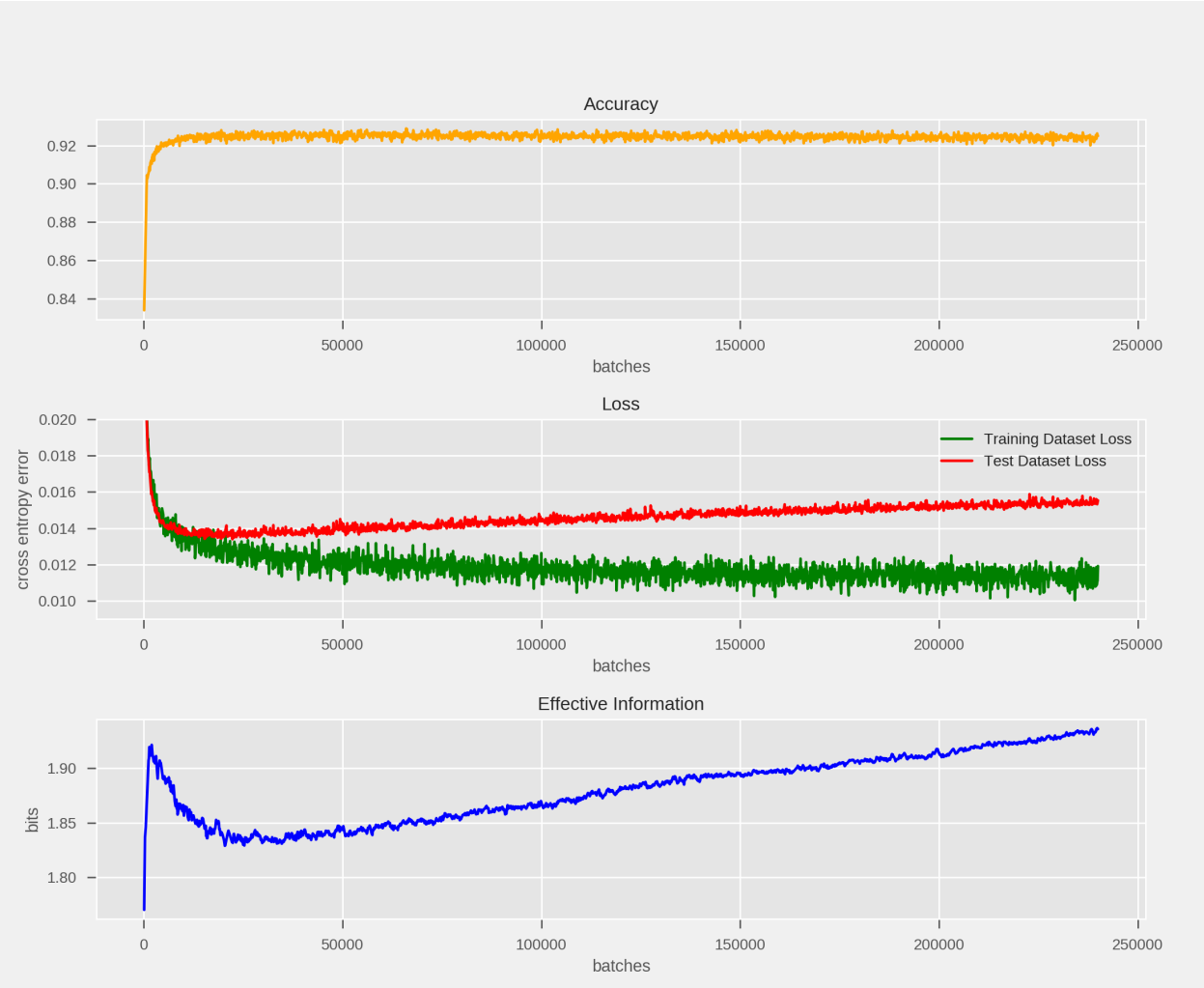
1.2 Code Documentation

`foresight.ei.H`(*x*, *dim=0*)

Compute the Shannon information entropy of x.

Given a tensor x, compute the shannon entropy along one of its axes. If x.shape == (N,) then returns a scalar (0-d tensor). If x.shape == (N, N) then information can be computed along vertical or horizontal axes by passing arguments dim=0 and dim=1, respectively.

Note that the function does not check that the axis along which information will be computed represents a valid probability distribution.


Accuracy
0.92
0.90
0.88
0.86
0.84
0
50000
100000
150000
200000
250000
batches
Loss
cross entropy error
0.020
0.018
0.016
0.014
0.012
0.010
Training Dataset Loss
Test Dataset Loss
0
50000
100000
150000
200000
250000
batches
Effective Information
bits
1.90
1.85
1.80
0
50000
100000
150000
200000
250000
batches

Parameters

- **x** (*torch.tensor*) –
- **dim** (*int*) –

Returns (torch.tensor) of a lower order than input x

foresight.ei.**degeneracy**(*model*, *input=None*, *shapes=None*, *norm=lin_norm*, *device='cpu'*)

Compute the degeneracy of neural network *model*.

Degeneracy is the entropy of the cumulative, normalized in-weights for each neuron in the graph:

$$\text{degeneracy} = H(\langle W^{\text{out}} \rangle)$$

If a *shapes* argument is provided, then *input* will not be used and need not be provided. If no *shapes* argument is provided, then an *input* argument must be provided to build its computation graph.

Parameters

- **model** (*nn.Module*) – neural network defined with PyTorch
- **input** (*torch.tensor*) – an input for the model (needed to build computation graph)
- **shapes** (*dict*) – dictionary containing mappings from child modules to their input and output shape (created by get_shapes() function)
- **norm** (*func*) – function to normalize the out weights of each neuron.
- **device** – (str): must be 'cpu' or 'cuda'

Returns degeneracy (float)

foresight.ei.**determinism**(*model*, *input=None*, *shapes=None*, *norm=lin_norm*, *device='cpu'*)

Compute the determinism of neural network *model*.

Determinism is the average entropy of the outweights of each node (neuron) in the graph:

$$\text{determinism} = \langle H(W^{\text{out}}) \rangle$$

If a *shapes* argument is provided, then *input* will not be used and need not be provided. If no *shapes* argument is provided, then an *input* argument must be provided to build its computation graph.

Parameters

- **model** (*nn.Module*) – neural network defined with PyTorch
- **input** (*torch.tensor*) – an input for the model (needed to build computation graph)
- **shapes** (*dict*) – dictionary containing mappings from child modules to their input and output shape (created by get_shapes() function)
- **norm** (*func*) – function to normalize the out weights of each neuron.
- **device** – (str): must be 'cpu' or 'cuda'

Returns determinism (float)

foresight.ei.**ei**(*model*, *input=None*, *shapes=None*, *norm=lin_norm*, *device='cpu'*)

Compute the effective information of neural network *model*.

Effective information is a useful measure of the information contained in the weighted connectivity structure of a network. It is used in theoretical neuroscience to study emergent structure in networks. It is defined by:

$$\text{EI} = \text{determinism} - \text{degeneracy}$$

explicitly:

$$\text{EI} = \langle H(W^{\text{out}}) \rangle - H(\langle W^{\text{out}} \rangle)$$

Which is equal to the average KL-divergence between the normalized out-weights of the neurons and the distribution of in-weights across the neurons in the network.

If a *shapes* argument is provided, then *input* will not be used and need not be provided. If no *shapes* argument is provided, then an *input* argument must be provided to build its computation graph.

Parameters

- **model** (*nn.Module*) – neural network defined with PyTorch
- **input** (*torch.tensor*) – an input for the model (needed to build computation graph)
- **shapes** (*dict*) – dictionary containing mappings from child modules to their input and output shape (created by get_shapes() function)
- **norm** (*func*) – function to normalize the out weights of each neuron.
- **device** – (str): must be 'cpu' or 'cuda'

Returns ei (float)

foresight.ei.**get_shapes**(*model*, *input*)

Get a dictionary {module: (in_shape, out_shape), ... } for modules in *model*.

Because PyTorch uses a dynamic computation graph, the number of activations that a given module will return is not intrinsic to the definition of the module, but can depend on the shape of its input. We therefore need to pass data through the network to determine its connectivity.

This function passes *input* into *model* and gets the shapes of the tensor inputs and outputs of each child module in model, provided that they are instances of VALID_MODULES.

Parameters

- **model** (*nn.Module*) – feedforward neural network
- **input** (*torch.tensor*) – a valid input to the network

Returns tuple(in_shape, out_shape)}

Return type Dictionary {*nn.Module*

foresight.ei.**lin_norm**(*W*)

Turns 2x2 matrix W into a transition probability matrix.

Applies a relu across the rows (to get rid of negative values), and normalize the rows based on their arithmetic mean.

Parameters **W** (*torch.tensor) of shape (2, 2*) –

Returns (torch.tensor) of shape (2, 2)

foresight.ei.**soft_norm**(*W*)

Turns 2x2 matrix W into a transition probability matrix.

The weight/adjacency matrix of an ANN does not on its own allow for EI to be computed. This is because the out weights of a given neuron are not a probability distribution (they do not necessarily sum to 1). We therefore must normalize them.

Applies a softmax function to each row of matrix W to ensure that the out-weights are normalized.

Parameters **W** (*torch.tensor) of shape (2, 2*) –

Returns (torch.tensor) of shape (2, 2)

CHAPTER 2

Gradient Noise Scale

2.1 Theory

The Gradient Noise Scale is a statistical measure which roughly predicts the optimal batch size for a given training task. During stochastic gradient descent, weight updates are performed using a gradient which is not computed from the whole dataset, but from a fraction of it (by randomly sampling a finite number of samples and averaging the gradient that each of these individually produce). The variance in the gradient between batches indicates how complex the dataset is. If this variance is high, then one should try to improve the accuracy of the stochastic gradient by increasing the batch size. A simplified version of this metric can be defined:

$$_{\text{simple}} = \frac{\text{tr}(\Sigma)}{G^2}$$

2.2 Code Documentation

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index