enzo

Release 0.0.1.dev6

Apr 09, 2021

Quick Start

CHAPTER 1

Enzo (in progress)

A neural network designed from scratch in Python (no tensorflow, pytorch, etc.).

1.1 Why from scratch?

Neural networks have grown to become very popular and useful. Because they are so complex, it is tough to have a core understanding of *how* the networks actually "learn." Coding one from scratch turned out to be extremely helpful.

1.2 Why in Python?

- Python is easy and quick to use, meaning that the time I spend coding will mostly be devoted to actually coding the network.
- More importantly, Python is easy to understand, meaning the code can serve as a learning resource for those who would like a better understanding of how neural networks work.

CHAPTER 2

Examples

CHAPTER 3

Models

Neural network models.

class enzo.models.**Model**(*layers*)

Simple densely connected neural network model.

Parameters **layers** (list of *enzo.layers.Layer*) – The list of layers in this model. The first layer in this list must have an explicit input length.

forward(*samples*)

Return and store in *self.outputs* the activation matrix of this layer after forward propagation.

outputs

The activations of the final layer of this *Model* for the most recent samples

Type list or ndarray

CHAPTER 4

Layers

Layers for sequential models.

class enzo.layers.**DenseLayer**(*n_units*, *activation=None*, *input_length=None*)

A densely connected layer for neural networks.

Parameters

- **n_units** (*int*) – The number of neurons in the layer.
- **activation** (*function, optional*) – The activation function for this layer. Default `enzo.activations.relu()`
- **input_length** (*int, optional*) – The length of the vector of inputs this layer will receive. For hidden layers, this should the number of units in the previous layer. `enzo.models.Model` automatically defines *input_length* for all layers excluding the first.

Notes

The weights matrix (*self.weights*) has each column corresponding to one unit's weights. This allows forward propagation with a matrix where each row is one sample to be *__matmul__*-ed with *self.weights* to generate activations.

build(*input_length=None*)

Initialize the weights of this `DenseLayer`.

Parameters **input_length** (*int*) – The shape of inputs to this layer (samples).

forward(*samples*)

Return and store in *self.outputs* the activation matrix of this layer after forward propagation.

class enzo.layers.**Layer**

Parent class for all custom layers.

Subclasses must implement `build()`.

build(*input_length*)

Initiate weights and other attributes that depend on *input_length*

Parameters **input_length** (*int*) – The shape of inputs to this layer (samples).

class `enzo.layers.`**`SoftmaxLayer`**(*n_units*, *input_length=None*)

Simple softmax-activated layer to follow a `DenseLayer`.

forward(*samples*)

Return and store in *self.outputs* the activation matrix of this layer after forward propagation.

CHAPTER 5

Activation Functions

Activation functions.

`enzo.activations.`**`noactivation`**(*rows*)

Do nothing, return *rows*.

See also:

`enzo.derivatives.d_noactivation()`

`enzo.activations.`**`relu`**(*rows*)

Apply max(0, n) to each n in *rows*.

Parameters **rows** (*array_like*)

See also:

`enzo.derivatives.d_relu()`

`enzo.activations.`**`sigmoid`**(*rows*)

Apply 1 / (1 + e ^ -n) to each n in *rows*.

Parameters **rows** (*array_like*)

See also:

`enzo.derivatives.d_sigmoid()`

`enzo.activations.`**`softmax`**(*rows*)

Perform softmax scaling for each row in *rows*.

See also:

`enzo.derivatives.d_softmax()`

CHAPTER 6

Derivatives

Derivative functions for functions.

`enzo.derivatives.`**`d_crossentropy`**(*y_true*, *y_pred*, *epsilon=1e-12*)

The derivative of the crossentropy loss function.

Parameters

- **y_true** (*array_like*)
- **y_pred** (*array_like*)
- **epsilon** (*float*) – The value at which *y_pred* is lower-bounded, by default 1e-12

Returns

Return type array_like

Notes

The point of an *epsilon* (ϵ) is to allow the computation of $\frac{y}{\hat{y}}$ which is undefined at $\hat{y} = 0$ by computing $\frac{y}{\min(\hat{y}, \epsilon)}$. (Note: $\hat{y}$ is any value in *y_pred* and y is any value in *y_true*).

See also:

`enzo.losses.crossentropy()`

`enzo.derivatives.`**`d_noactivation`**(*rows*)

The derivative of a f(x)=x activation (*noactivation*).

Parameters **rows** (*array_like*)

Returns The derivative evaluated at each row in *rows*.

Return type array_like

See also:

`enzo.activations.noactivation()`

`enzo.derivatives.d_relu(rows)`

The derivative of the rectified linear unit (*relu*).

Parameters **rows** (*array_like*)

Returns The derivative evaluated at each row in *rows*.

Return type array_like

Notes

`d_relu()` evaluated at 0 is 0 despite the fact that the true derivative of ReLU evaluated at 0 is undefined. This allows for a continuous derivative function, letting weights set to 0 to have a derivative.

$$\left.\frac{dr}{dx}\right|_0 = 0$$

See also:

`enzo.activations.relu()`

`enzo.derivatives.d_sigmoid(rows)`

The derivative of the sigmoid activation function.

Parameters **rows** (*array_like*)

Returns The derivative evaluated at each row in *rows*.

Return type array_like

See also:

`enzo.activations.sigmoid()`

`enzo.derivatives.d_softmax(rows)`

The derivative of the softmax activation function.

Parameters **rows** (*array_like*)

Returns The derivative evaluated at each row in *rows*.

Return type array_like

See also:

`enzo.activations.softmax()`

`enzo.derivatives.with_derivative(derivative)`

Decorator for functions with derivatives

Parameters **derivative** (*callable*) – The derivative of the decorated function.

CHAPTER 7

Losses

Loss functions.

`enzo.losses.`**`crossentropy`**(*y_true*, *y_pred*, *epsilon=1e-12*)

Calculate the crossentropy loss of *y_true* with respect to *y_pred*.

Parameters

- **y_true** (*array_like*) – One-hot encoded ture labels.
- **y_pred** (*array_like*) – Model predictions.
- **epsilon** (*float, optional*) – The value at which *y_pred* is lower-bounded, by default 1e-12

Notes

The point of an *epsilon* (ϵ) is to allow the computation of $\log(\hat{y})$ which is undefined at $\hat{y} = 0$ by computing $\log(\min(\hat{y}, \epsilon))$. (Note: $\hat{y}$ is any value in *y_pred*).

See also:

`enzo.derivatives.d_crossentropy()`

CHAPTER 8

Exceptions

Errors and exceptions.

exception enzo.exceptions.**BackBeforeForwardException**

Raise when back-propagation is run before forward-propagation.

exception enzo.exceptions.**LayerBuildingError**

Raise when *enzo.layers.DenseLayer.build()* fails.

Python Module Index

e

Index