veni

Release 2022

Dario Coscia, Alessandro Pierro, Francesco Tomba

Aug 17, 2022

CONTENTS

A Python package for deep learning using forward automatic differentiation based on JAX.

CHAPTER

ONE

TABLE OF CONTENTS

CHAPTER

TWO

DESCRIPTION

veni is a Python package, built on JAX, providing an easy interface to deal with Neural Network using forward automatic differention. Inspired by the very recent (2021) papers of Atılım Günes Baydin et al. and David Silver et al., we have decided to implement a package able to reproduce the results, and give freedom to further investigate this new emerging area of AI.

CHAPTER THREE

DEPENDENCIES AND INSTALLATION

veni requires requires `jax`, `jaxlib`, `torch`, `numpy`, `sphinx` (for the documentation). The code is tested for Python 3, while compatibility of Python 2 is not guaranteed anymore. It can be installed directly from the source code.

3.1 Installing from source

The official distribution is on GitHub, and you can clone the repository using

```
> git clone https://github.com/DSSC-projects/veni
```

You can also install it using pip via

```
> python -m pip install git+https://github.com/DSSC-projects/veni
```

CHAPTER

FOUR

DOCUMENTATION

veni uses Sphinx for code documentation. You can view the documentation online here. To build the html version of the docs locally simply:

```
cd docs
make html
```

The generated html can be found in `docs/build/html`. Open up the `index.html` you find there to browse.

CHAPTER

FIVE

EXAMPLES AND TUTORIALS

The directory examples contains some examples showing how to use **veni**. In particular we show how to create simple deep learning architectures, how to train via forward automatic differentiation an architecture, and finally how to sample differently candidate directions.

CHAPTER

SIX

BENCHMARKS

The directory benchmarks contains some important benchmarks showing how to reproduce Atılım Günes Baydin et al. results by using the simple **veni** interface. We further provide logs for efficient analysis of the data. Further benchmark involving directions and optimizers are also available for testing.

6.1 References

To implement the package we follow these works:

- A. G. Baydin, B. A. Pearlmutter, D. Syme, F. Wood, and P. Torr. _Gradients without back- propagation, 2022
- D. Silver, A. Goyal, I. Danihelka, M. Hessel, and H. van Hasselt. _Learning by directional gradient descent. In International Conference on Learning Representations, 2022
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs (0.3.13) [Computer software]. http://github.com/google/jax
- Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alch e-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

CHAPTER

SEVEN

AUTHORS AND CONTRIBUTORS

veni is currently developed and mantained by Data Science and Scientific Computing master students:

- Francesco Tomba
- Dario Coscia
- Alessandro Pierro

Contact us by email for further information or questions about **veni**, or suggest pull requests. Contributions improving either the code or the documentation are welcome!

CHAPTER

EIGHT

HOW TO CONTRIBUTE

We'd love to accept your patches and contributions to this project. There are just a few small guidelines you need to follow.

8.1 Submitting a patch

1. It's generally best to start by opening a new issue describing the bug or feature you're intending to fix. Even if you think it's relatively minor, it's helpful to know what people are working on. Mention in the initial issue that you are planning to work on that bug or feature so that it can be assigned to you.
2. Follow the normal process of forking the project, and setup a new branch to work in. It's important that each group of changes be done in separate branches in order to ensure that a pull request only includes the commits related to that bug or feature.
3. To ensure properly formatted code, please make sure to use 4 spaces to indent the code. The easy way is to run on your bash the provided script: ./code_formatter.sh. You should also run pylint over your code. It's not strictly necessary that your code be completely "lint-free", but this will help you find common style issues.
4. Do your best to have well-formed commit messages for each change. This provides consistency throughout the project, and ensures that commit messages are able to be formatted properly by various git tools.
5. Finally, push the commits to your fork and submit a pull request. Please, remember to rebase properly in order to maintain a clean, linear git history.

CHAPTER

NINE

CITATIONS

If you are considering using **veni** on your reaserch please cite us:

APA:

```
Tomba, F., Coscia, D., & Pierro, A. (2022). veni (Version 0.0.1) [Computer software].
↪https://github.com/DSSC-projects/veni
```

BibTex:

```
@software{Tomba_veni_2022,
author = {Tomba, Francesco and Coscia, Dario and Pierro, Alessandro},
month = {6},
title = {{veni}},
url = {https://github.com/DSSC-projects/veni},
version = {0.0.1},
year = {2022}
}
```

CHAPTER

TEN

LICENSE

See the LICENSE file for license rights and limitations (MIT).

CHAPTER

ELEVEN

SEE MORE...

11.1 Installation

veni is currently available on GitHub and can be installed directly from source using:

```
git clone https://github.com/DSSC-projects/veni
```

or by:

```
python -m pip install git+https://github.com/DSSC-projects/veni
```

11.2 Reference manual

11.2.1 Modules

veni.function module

class veni.function.**LeakyReLu**

Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**LogSigmoid**

Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**LogSoftmax**

Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**ReLU**

Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**Sigmoid**
Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**Softmax**
Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**Softplus**
Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

class veni.function.**Tanh**
Bases: *veni.module.Activation*

forward(*x*, *params=None*)

generate_parameters()

veni.functiontools module

veni.functiontools.**CrossEntropy**(*y*, *y_hat*)
CrossEntropy loss EXPECTS: tensor of the shape (N, k1, k2, . . . , kn) where N is the number of examples in the batch

Parameters

- **y** (*jnp.array*) – Ground truth tensor
- **y_hat** (*jnp.array*) – Model predictions

Returns Loss for each batch

Return type float

veni.functiontools.**LazyCrossEntropy**(*y*, *y_hat*)
CrossEntropy loss This cross entropy implementationmay suffer numerical instabilities depending on the specific problem, consider using 'CrossEntropy'.

EXPECTS: tensor of the shape (N, k1, k2, . . . , kn) where N is the number of examples in the batch

Parameters

- **y** (*jnp.array*) – Ground truth tensor
- **y_hat** (*jnp.array*) – Model predictions

Returns Loss for each batch

Return type float

veni.functiontools.**MSE**(*y*, *y_hat*)
Mean square error loss, reduction mean

Parameters

- **y** (*jnp.array*) – Ground truth tensor

- **y_hat** (*jnp.array*) – Model predictions

Returns Loss for each batch

Return type float

`veni.functiontools.`**`leaky_relu`**(*x*)

Applies the leaky rectified linear unit function element-wise

Parameters

- **x** (*jax.array*) – input
- **negative_slope** (*float, optional*) – negative slope, defaults to 0.01

Returns leaky rectified linear unit on x

Return type jax.array

`veni.functiontools.`**`log_sigmoid`**(*x*)

Applies the logarithmic sigmoid function element-wise

Parameters **x** (*jax.array*) – input

Returns logarithmic sigmoid on x

Return type jax.array

`veni.functiontools.`**`log_softmax`**(*x*)

Applies the logarithmic softmax function element-wise.

Parameters **x** (*jax.array*) – input

Returns logarithmic softmax on x

Return type jax.array

`veni.functiontools.`**`relu`**(*x*)

Applies the rectified linear unit function element-wise

Parameters **x** (*jax.array*) – input

Returns rectified linear unit on x

Return type jax.array

`veni.functiontools.`**`sigmoid`**(*x*)

Applies the sigmoid function element-wise

Parameters **x** (*jax.array*) – input

Returns sigmoid on x

Return type jax.array

`veni.functiontools.`**`softmax`**(*x*)

Applies the softmax function element-wise.

Parameters **x** (*jax.array*) – input

Returns softmax on x

Return type jax.array

`veni.functiontools.`**`softplus`**(*x*)

Applies the softplus function element-wise. For numerical stability the implementation reverts to the linear function when input*beta > threshold.

Parameters

- **x** (*jax.array*) – input
- **beta** (*int, optional*) – paramter, defaults to 1
- **threshold** (*int, optional*) – threshold, defaults to 20

Raises **ValueError** – beta value must be greater than zero

Returns softplus on x

Return type jax.array

`veni.functiontools.tanh(x)`

Applies the tanh function element-wise

Parameters **x** (*jax.array*) – input

Returns tanh on x

Return type jax.array

veni.module module

`class veni.module.Activation(f)`

Bases: `abc.ABC`

`abstract forward(x, params=None)`

`abstract generate_parameters()`

`class veni.module.Module`

Bases: `abc.ABC`

`abstract forward(x, params=None)`

`class veni.module.Optimizer`

Bases: `abc.ABC`

`abstract update(params, grad)`

`class veni.module.Sampler`

Bases: `abc.ABC`

veni.net module

`class veni.net.AvgPool2D(kernel_size, stride=None, padding=None)`

Bases: `veni.module.Module`

`forward(x, params=None)`

Public forward method for Conv layer

Parameters

- **params** (*jnp.array*) – Parameters of the layer
- **x** (*jnp.array*) – Input

Returns Activation

Return type jnp.array

`generate_parameters()`

Generate parameters for current layer

Returns weight and bias tensors N(0,1) initialized

Return type jnp.array

property input

property key

property output

class veni.net.**Conv2D**(*inChannels*, *outChannels*, *kernelSize*, *stride*, *padding*, *key*)

Bases: *veni.module.Module*

forward(*x*, *params*)

Public forward method for Conv layer

EXPECTS: x: tensor of the form NCHW (images)x(channels)x)(height)x(width) params[0]: tensor of the form OIHW (outputCh)x(inputCh)x(kernelHeight)x(kernelWidth) params[1]: bias

Parameters

- **params** (*jnp.array*) – Parameters of the layer
- **x** (*jnp.array*) – Input

Returns Activation

Return type jnp.array

generate_parameters()

Generate parameters for current layer

Returns weight and bias tensors N(0,1) initialized

Return type jnp.array

property input

property key

property output

class veni.net.**Flatten**

Bases: *veni.module.Module*

forward(*x*, *params=None*)

returns flattened tensor

Returns _description_

Return type _type_

TODO: optimize that

generate_parameters()

Generate parameters for current layer

Returns weight and bias tensors N(0,1) initialized

Return type jnp.array

property input

property key

property output

class veni.net.**Linear**(*input*, *output*, *key*, *bias=True*)

Bases: *veni.module.Module*

forward(*x*, *params*)
Public forward method for Linear layer

Parameters

- **params** (*jnp.array*) – Parameters of the layer
- **x** (*jnp.array*) – Input

Returns Activation

Return type jnp.array

generate_parameters()

property input

property key

property output

class veni.net.**MLP**(*layers*, *func*, *key*)
Bases: *veni.module.Module*

forward(*x*, *params*)

generate_parameters()

property key

property layers

single_forward(*x*, *params*)

class veni.net.**MaxPool2D**(*kernel_size*, *stride=None*, *padding=None*)
Bases: *veni.module.Module*

forward(*x*, *params=None*)
Public forward method for Conv layer

Parameters

- **params** (*jnp.array*) – Parameters of the layer
- **x** (*jnp.array*) – Input

Returns Activation

Return type jnp.array

generate_parameters()
Generate parameters for current layer

Returns weight and bias tensors N(0,1) initialized

Return type jnp.array

property input

property key

property output

class veni.net.**Sequential**(*list*)
Bases: *veni.module.Module*

forward(*x*, *params*)
Forward method for sequential object

Parameters

- **params** (*jnp.array*) – _description_
- **x** (*jnp.array*) – _description_

Returns activation

Return type jnp.array

generate_parameters()

Generate parameters for layers in sequential

Returns _description_

Return type jnp.array

veni.optim module

class veni.optim.**Adam**(*params*, *beta1=0.9*, *beta2=0.999*, *eta=0.001*)

Bases: *veni.module.Optimizer*

update(*params*, *grads*)

Update method for Adam

Parameters

- **params** (*jax.array*) – paramters to optimize
- **grad** (*jax.array*) – loss gradient

Returns optimized parameters

Return type jax.array

class veni.optim.**NormalLikeSampler**

Bases: *veni.module.Sampler*

class veni.optim.**RademacherLikeSampler**

Bases: *veni.module.Sampler*

class veni.optim.**SGD**(*params*, *momentum=0*, *dampening=0*, *eta=0.001*)

Bases: *veni.module.Optimizer*

update(*params*, *grad*)

Update method for SGD

Parameters

- **params** (*jax.array*) – paramters to optimize
- **grad** (*jax.array*) – loss gradient

Returns optimized parameters

Return type jax.array

class veni.optim.**TruncatedNormalLikeSampler**(*lower=- 1*, *upper=1*)

Bases: *veni.module.Sampler*

class veni.optim.**UniformLikeSampler**

Bases: *veni.module.Sampler*

veni.optim.**grad_fwd**(*params*, *x*, *y*, *loss*, *dirs=1*, *sampler=<veni.optim.NormalLikeSampler object>*)

Function to calculate the gradient in forward mode using 1 or more directions

Parameters

- **params** (*List*) – Parameters of the model
- **x** (*jnp.array*) – Input of the model
- **y** (*jnp.array*) – labels
- **loss** (*Callable*) – loss function
- **dirs** (*int, optional*) – Number of directions used to calculate the gradient, defaults to 1
- **sampler** (*Class, optional*) – Sampler used to sample gradient direction for each layer, defaults to NormalLikeSampler()

Returns Gradient as list of all components for each layer

Return type List

`veni.optim.`**`plist_reduce`**(*vs*, *js*)

Multiply the jacobian vector product with the tangent directions

Parameters

- **vs** (*list(tuple(jnp.array, jnp.array))*) – tangent directions
- **js** (*float*) – jacobian vector product

Returns multiply the jacobian vector product with the tangent directions

Return type list(tuple(jnp.array, jnp.array))

veni.utils module

class `veni.utils.`**`FlattenAndCast`**

Bases: `object`

class `veni.utils.`**`NumpyLoader`**(*dataset*, *batch_size=1*, *shuffle=False*, *sampler=None*, *batch_sampler=None*, *num_workers=0*, *pin_memory=False*, *drop_last=False*, *timeout=0*, *worker_init_fn=None*)

Bases: `Generic[torch.utils.data.dataloader.T_co]`

batch_size: Optional[int]

dataset: torch.utils.data.dataset.Dataset[torch.utils.data.dataloader.T_co]

drop_last: bool

num_workers: int

pin_memory: bool

pin_memory_device: str

prefetch_factor: int

sampler: Union[torch.utils.data.sampler.Sampler, Iterable]

timeout: float

`veni.utils.`**`numpy_collate`**(*batch*)

`veni.utils.`**`one_hot`**(*x*, *k*, *dtype=<class 'jax.numpy.float32'>*)

Create a one-hot encoding of x of size k.

PYTHON MODULE INDEX

V

INDEX