Juice - Machine Learning with Rust

A brief introduction and architecture overview

Bernhard Schuster

November 30, 2017

Scope of Juice: Deep Learning, Artificial Neural Networks

Scope of Juice: Deep Learning

Does:

- Deep Learning / Neural Networks

Not:

- Classic Machine Learning
- Bayes
- k-Means
- Decission Trees
- Support Vector Machine

What I am not going to explain/show

- network architecture design
- speed comparisons
- fancy graphics

History

- Leaf
- Autumn AI
- Goal: Re-pricing prediction for online merchants

Spearow

- [1] Juice ← Leaf
- [2] Greenglas ← Cuticula
- Coaster ← Collenchyma
- [3] Org: Spearow

---

[1] make

[2] me

[3] pretty

Motivation

Competition: Tensorflow, Torch, Caffe, Theano

Spearow Goals

- Easy to run
- Easy to integrate
- Easy to deploy on 🐧 🍎 ⊞
- Ready for Embedded [4]

---

[4] anything 32bits and up with enough memory

rusty-machine, vikos, rustlearn, prophet, nn, ...

Crates

<your application>

juice

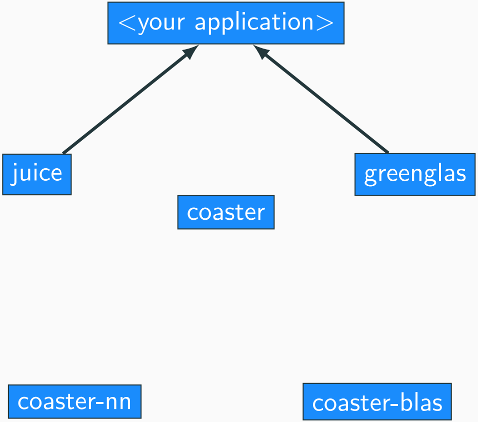
coaster

greenglas

coaster-nn

coaster-blas

Crates



<your application>

juice

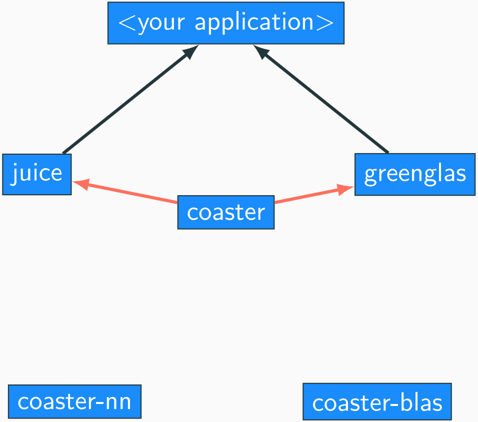
greenglas

coaster

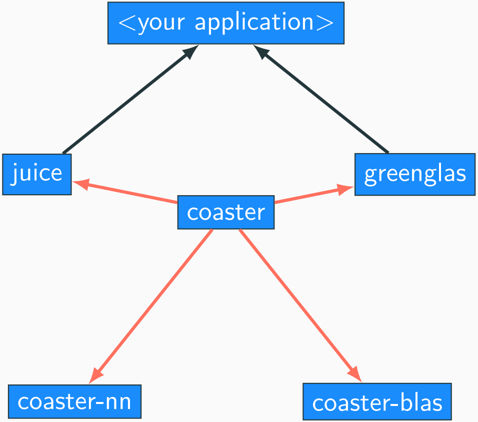
coaster-nn

coaster-blas

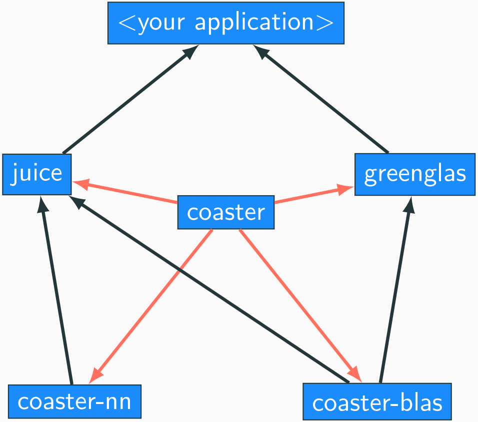
Crates



Crates



Crates



<your application>

juice

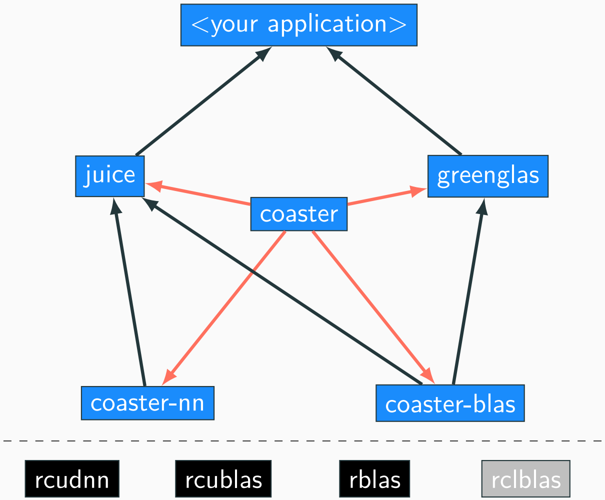
coaster

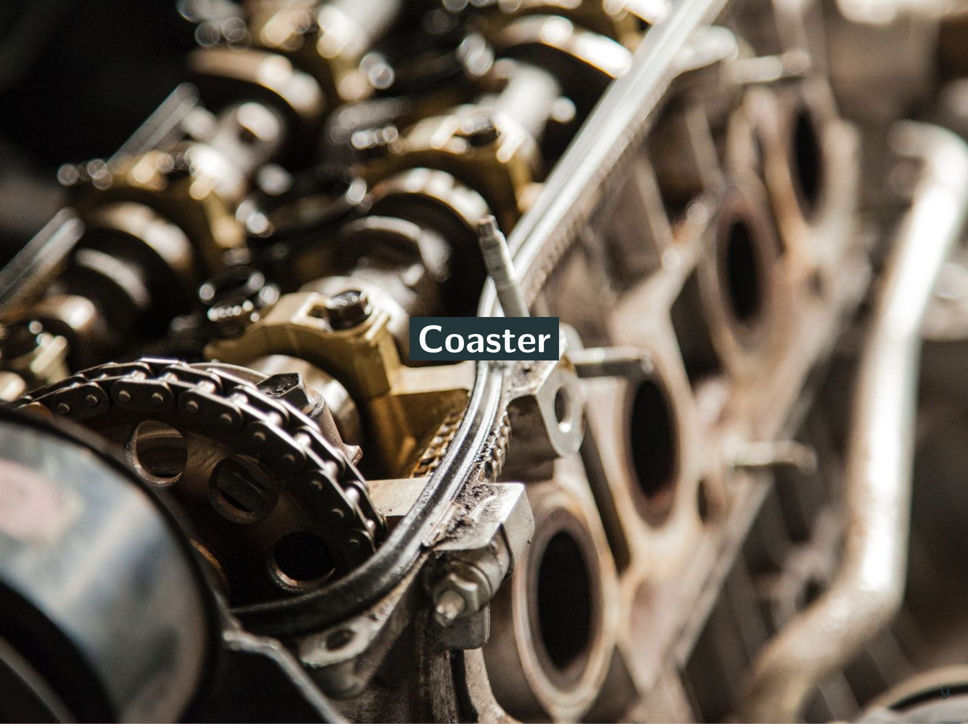
greenglas

coaster-nn

coaster-blas

Crates


<your application>
juice
greenglas
coaster
coaster-nn
coaster-blas
rcudnn
rcublas
rblas
rclblas



Coaster

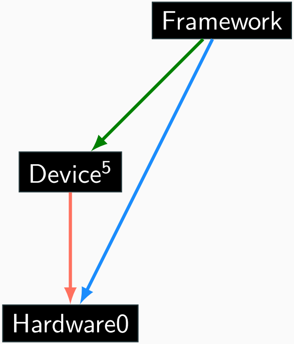
Abstraction

- device memory
- internal structure
- backend specific

Tensor

```
pub struct SharedTensor<T> {
    desc: TensorDesc,
    locations: RefCell<Vec<TensorLocation>>,
    up_to_date: Cell<BitMap>,
}
```

Backend Abstraction[6]

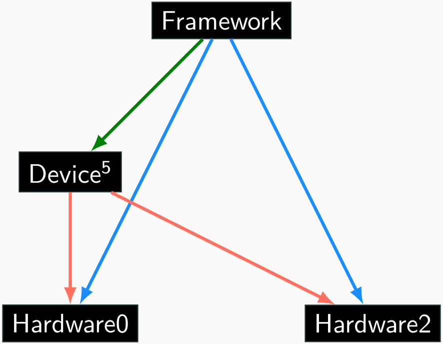


Framework

Device[5]

Hardware0

Backend Abstraction[6]

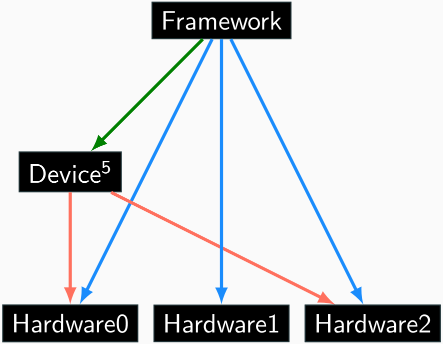

Framework

Device[5]

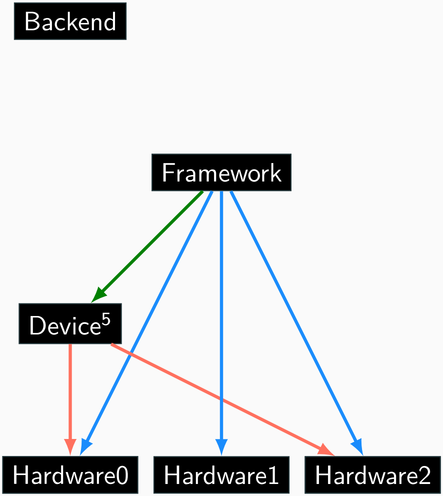
Hardware0

Hardware2

Backend Abstraction[6]



Backend Abstraction[6]



Backend

Framework

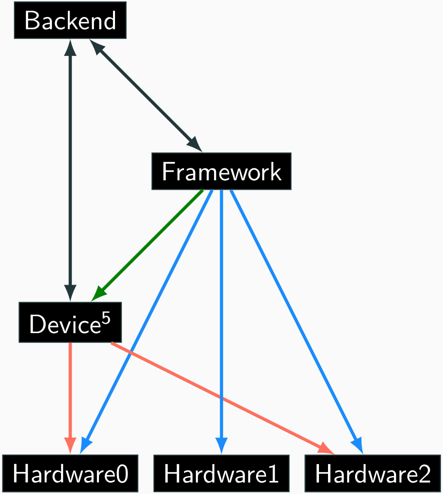
Device[5]

Hardware0

Hardware1

Hardware2

Backend Abstraction[6]


Backend

Framework

Device[5]

Hardware0

Hardware1

Hardware2

---

[5] better HardwareGroup/DeviceGroup

[6] the members, associated types differ!

Framework

```
/// Defines a Framework.
pub trait IFramework {
    /// The Hardware representation for this Framework.
    type H: IHardware;
    /// The Device representation for this Framework.
    type D: IDevice + Clone;
    /// The Binary representation for this Framework.
    type B: IBinary + Clone;
```

---

[7] or better: CPU

Framework

```
/// Defines a Framework.
pub trait IFramework {
    /// The Hardware representation for this Framework.
    type H: IHardware;
    /// The Device representation for this Framework.
    type D: IDevice + Clone;
    /// The Binary representation for this Framework.
    type B: IBinary + Clone;
```

- cuda
- OpenCL
- Native[7]

---

[7] or better: CPU

Backend

```
pub struct Backend<F: IFramework> {
    framework: Box<F>,
    device: F::D,
}
```

Backend

```
pub struct Backend<F: IFramework> {
    framework: Box<F>,
    device: F::D,
}
```

Backend is tied to a Framework

Backend has a Device

Device has Hardwares

Juice

Memory objects with arbitrary (but defined!) structure and dimensions

=

Blobs

=

SharedTensor

parameters

=

weights (+ bias)

Architecture

trait ILayer **common interface that is expected to be impl'd by all layers**

---

[8] mostly a helper to make LayerConfig impl easier

[9] or any other for that matter, just i.e.

Architecture

trait ILayer	**common interface that is expected to be impl'd by all layers**
LayerConfig	**handles input and output tracking, name based**

---

[8] mostly a helper to make LayerConfig impl easier

[9] or any other for that matter, just i.e.

Architecture

trait ILayer	**common interface that is expected to be impl'd by all layers**
LayerConfig	**handles input and output tracking, name based**
LayerType	**holds configuration data for the particular layer**[8]

---

[8] mostly a helper to make LayerConfig impl easier

[9] or any other for that matter, just i.e.

Architecture

trait ILayer	**common interface that is expected to be impl'd by all layers**
LayerConfig	**handles input and output tracking, name based**
LayerType	**holds configuration data for the particular layer**[8]
PoolingConfig[9]	**holds specific information**

---

[8] mostly a helper to make LayerConfig impl easier

[9] or any other for that matter, just i.e.

Layers implemented

activation	ReLU
activation	Sigmoid
activation	TanH
common	Dropout[10]
common	Softmax
common	LogSoftmax
common	Convolution [11]
common	Linear
common	Pooling
util	Reshape
util	Flatten

---

[10] unmerged, but it is there pr#13

[11] the native impl has only forward impl

Containers

- **most common: Sequential / SequentialConfig**
- **stackable, container in a container in a container** [12]
- **are Layers/LayerConfigs too**

---

[12] no docker involved

ILayer

ILayer

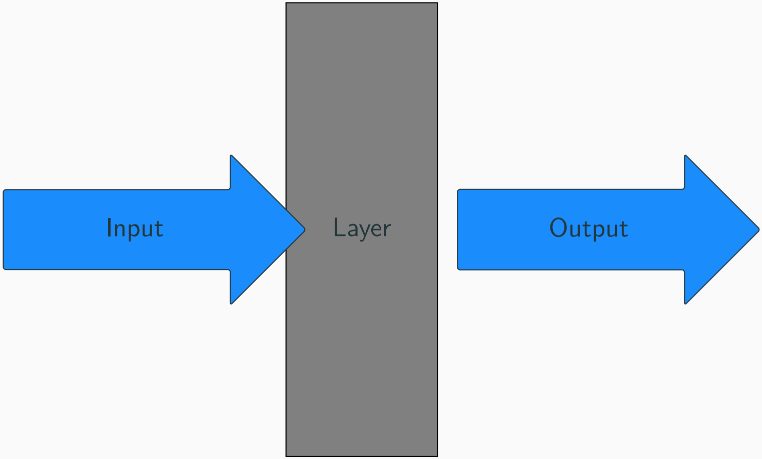
=

ComputeOutput

\+ ComputeInputGradient

\+ ComputeParametersGradient

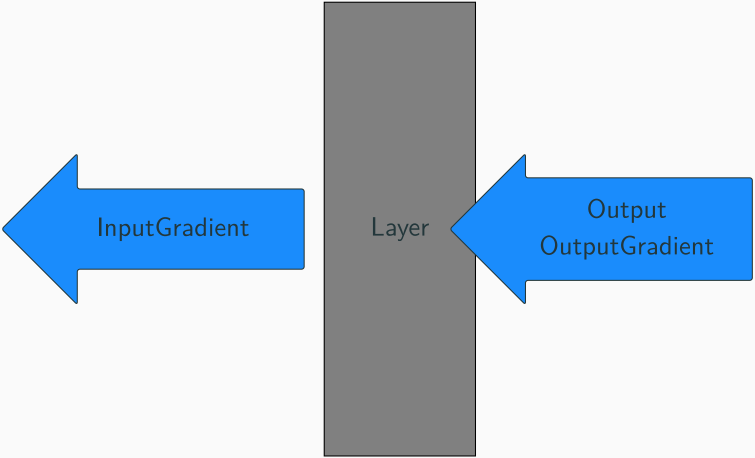
DataFlow: ComputeOutput



Layer: ComputeOutput

```
/// A Layer that can compute the output for a given input.
pub trait ComputeOutput<T, B: IBackend> {
    /// Compute output for given input
    /// and write them into `output_data`.
    fn compute_output(&self,
              backend: &B,
              weights_data: &[&SharedTensor<T>],
              input_data: &[&SharedTensor<T>],
              output_data: &mut [&mut SharedTensor<T>]);
}
```

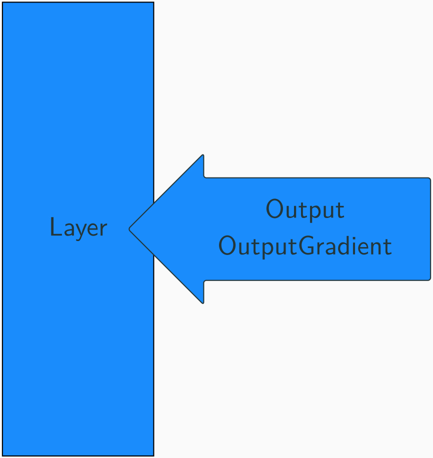
DataFlow: ComputeInputGradient



Layer: ComputeInputGradient

```
/// A Layer that can compute the gradient with respect to its input.
pub trait ComputeInputGradient<T, B: IBackend> {
    /// Compute gradients with respect to the inputs
    /// and write them into `input_gradients`.
    fn compute_input_gradient(&self,
                              backend: &B,
                              weights_data: &[&SharedTensor<T>],
                              output_data: &[&SharedTensor<T>],
                              output_gradients: &[&SharedTensor<T>],
                              input_data: &[&SharedTensor<T>],
                              input_gradients: &mut [&mut SharedTensor<T>]);
}
```

DataFlow: ComputeParametersGradient

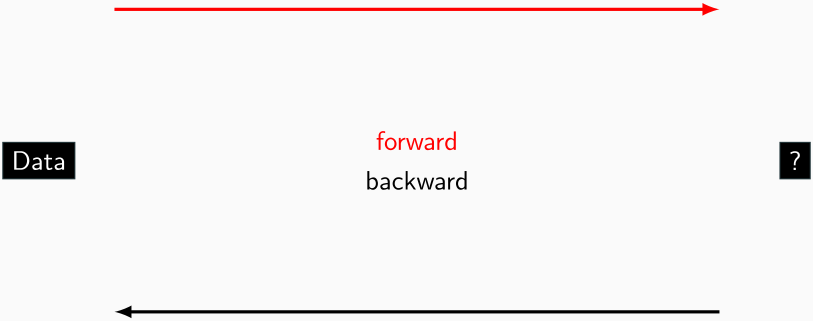


Layer: ComputeParametersGradient

```
/// A Layer that can compute the gradient with respect to its parameter
pub trait ComputeParametersGradient<T, B: IBackend> {
    /// Compute gradients with respect to the parameters
    /// and write them into `parameters_gradients`.
    fn compute_parameters_gradient(&self,
                                   backend: &B,
                                   output_data: &[&SharedTensor<T>],
                                   output_gradients: &[&SharedTensor<T>],
                                   input_data: &[&SharedTensor<T>],
                                   parameters_gradients: &mut [&mut SharedTenso
        // not required for all layers,
        // only those with internal state to be optimized
    }
}
```

Something is still missing.

Something is still missing (?)



Data

forward

backward

?

Solver

- connects forward and backward layer
- optimized towards objective[13]
- contains almost all tuneable hyperparameters[14]

---

[13] implicitly defined loss function

[14] Parameters which are not optimized towards

SolverConfig

```rust
pub struct SolverConfig {
    pub name: String,
    pub network: LayerConfig,
    pub objective: LayerConfig,
    pub solver: SolverKind,
    // more..
}
```

Be Fast, Stay Fast

Speed

- 95% spent in optimized libraries (i.e. cudnn)
- avoid implementing things manually (blas)
- skip operations if possible
- provide in place operations
- never re-allocate device memory if possible / hold device mem as long as possible

Alexnet example (network cfg)

```rust
extern crate env_logger;
extern crate coaster as co;
extern crate juice;

use co::prelude::*;

let mut cfg = SequentialConfig::default();
cfg.add_input("data", &[128, channels, px_dim, px_dim]);

let conv1_layer_cfg = ConvolutionConfig {
num_output: 64,
filter_shape: vec![11],
padding: vec![2],
stride: vec![4],
};

cfg.add_layer(LayerConfig::new("conv1", conv1_layer_cfg));
cfg.add_layer(LayerConfig::new("conv1/relu", LayerType::ReLU));
```

Alexnet example (network cfg)

```
cfg.add_layer(LayerConfig::new("conv2",
                   ConvolutionConfig {
                       num_output: 192,
                       filter_shape: vec![5],
                       padding: vec![2],
                       stride: vec![1],
                   }));
cfg.add_layer(LayerConfig::new("conv2/relu",
                   LayerType::ReLU));
cfg.add_layer(LayerConfig::new("pool2",
                   PoolingConfig {
                       mode: PoolingMode::Max,
                       filter_shape: vec![3],
                       stride: vec![2],
                       padding: vec![0],
                   }));
```

Alexnet example (network cfg)

```
// more layers ..
cfg.add_layer(LayerConfig::new("fc1",
         LinearConfig { output_size: 4096 }));
cfg.add_layer(LayerConfig::new("fc2",
         LinearConfig { output_size: 4096 }));
cfg.add_layer(LayerConfig::new("fc3",
         LinearConfig { output_size: 1000 }));

// create pseudo probabilities + log
cfg.add_layer(LayerConfig::new("log_softmax", LayerType::LogSoftmax));

// set up backends
let backend = Rc::new(Backend::<Cuda>::default().unwrap());

let mut network = Layer::from_config(backend.clone(),
                 &LayerConfig::new("alexnet",
                            LayerType::Sequential(cfg)));
```

Alexnet example (classifer cfg)

```
let mut classifier_cfg = SequentialConfig::default();
classifier_cfg.add_input("network_out", &[batch_size, 10]);
classifier_cfg.add_input("label", &[batch_size, 1]);
// set up nll loss
let nll_layer_cfg = NegativeLogLikelihoodConfig { num_classes: 10 };
let nll_cfg = LayerConfig::new("nll",
                    LayerType::NegativeLogLikelihood(nll_layer_cfg));
classifier_cfg.add_layer(nll_cfg);
```

Alexnet example (training/backward pass, 1)

```
// set up solver
let mut solver_cfg = SolverConfig {
    minibatch_size: batch_size,
    base_lr: learning_rate,
    momentum: momentum,
    ..SolverConfig::default()
};
solver_cfg.network = LayerConfig::new("network", cfg);
solver_cfg.objective = LayerConfig::new("classifier",
                    classifier_cfg);
let mut solver = Solver::from_config(backend.clone(),
                    &solver_cfg);

// set up confusion matrix
let mut confusion = ConfusionMatrix::new(10);
confusion.set_capacity(Some(1000));
```

Alexnet example (training/backward pass, 2)

```
let inp = SharedTensor::<f32>::new(&[batch_size, channels, px_dim,
let label = SharedTensor::<f32>::new(&[batch_size, 1]);
let inp_lock = Arc::new(RwLock::new(inp));
let label_lock = Arc::new(RwLock::new(label));

for _ in 0..(example_count / batch_size) {
    let mut targets = Vec::new();
    for (batch_n, (label_val, input)) in decoded_images
                                           .by_ref()
                                           .take(batch_size)
                                           .enumerate() {
        let mut inp = inp_lock.write().unwrap();
        let mut label = label_lock.write().unwrap();
        write_batch_sample(&mut inp, &input, batch_n);
        write_batch_sample(&mut label, &[label_val], batch_n);
        targets.push(label_val as usize);
    }
    // ...
```

Alexnet example (training/backward pass, 2)

```
for _ in 0..(example_count / batch_size) {
    // ...
    // train the network!
    let infered_out = solver.train_minibatch(
                                    inp_lock.clone(),
                                    label_lock.clone());
    let mut infered = infered_out.write().unwrap();
    let predictions = confusion.get_predictions(&mut infered);

    confusion.add_samples(&predictions, &targets);
    println!(
        "Last sample: {} | Accuracy {}",
        confusion.samples().iter().last().unwrap(),
        confusion.accuracy()
    );
}
```

Alexnet example (prediction/inference/forward pass)

```
let inp = SharedTensor::new(&[1, channels, px_dim, px_dim]);
let inp_lock = Arc::new(RwLock::new(inp));
let predictions = network.forward(&[inp_lock.clone()]);
// do something with those predictions
```

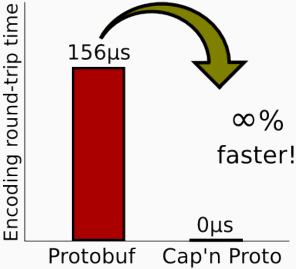
mnist, fashion-mnist

`github.com/spearow/juice-examples`

save/load/store/export

cap'n'proto

- interface to other languages
- load, store weights and the layer architecture
- it executes *really fast!*



What happened in 2017?

Summary 2017

- Learning a whole lot more about rust [15]
- Merging remaining large pull request [16]
- impl Dropout in coaster-nn for backends cuda and native
- Setting up CI ci.spearow.io / @sirmergealot 
- Auto generating config files for CI spearow/ci
- Fight a lot with CI, get GPU hardware access from containers from garden hypervisor [17]

---

[15] build.rs, bindgen, PhantomData, Any trait and more

[16] hands down @alexandermorozov

[17] Still an ugly hack

Roadmap 2018

Roadmap 2018

- Recursive Neural Networks / Long Term Short Term Memory [coaster-nn]
- OpenCL Backend [juice/coaster-nn]
- Autodiff [juice/coaster-nn]
- Accuracy enum instead of type usage [juice/greenglas/coaster]

Low hanging fruit / math challenges

- Honour biases [juice]
- Gradient calculation for ND-convolution native backend [coaster-nn]
- Regression examples [juice-examples]
- Add preprocessing filters [18] [greenglas]

---

[18] FFT, wavelet, denoise, addnoise, cutout, rescale, blur, ..

Questions?

bernhard@spearow.io

spearow.io

gitter.im/spearow/juice

Credits

Presentation by Bernhard Schuster - ahoi.io
Theme by Matze Vogelsang and contributors -
`https://bloerg.net`bloerg.net

The theme *itself* is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The images are taken from *unsplash* and are free to use for whatever you want.

Font Awesome icons SIL OFL 1.1

Icons displayed made by Bernhard Schuster.