KeraSH: A NN toolkit written in shell

Jean-Adrien & Audran

LSE IA
ducast_j@lse.epita.fr
audran.doublet@lse.epita.fr

April 9, 2019

Overview

Features Overview

- Matrix and Tensors BLAS
- Modular Neural Network Architecture (fully connected and CNN)
- More than 20 activation functions
- AutoML

Storing data in shell for "efficient" calculations

Problem Constraints:

- Lot of organized float data
- High number of operation on them
- All you can store is strings in files

Solution: a temporary file system in RAM:

```
sudo mount -t tmpfs -o size=512m tmpfs "./kerash_mountpoint"
```

KeraSH architecture

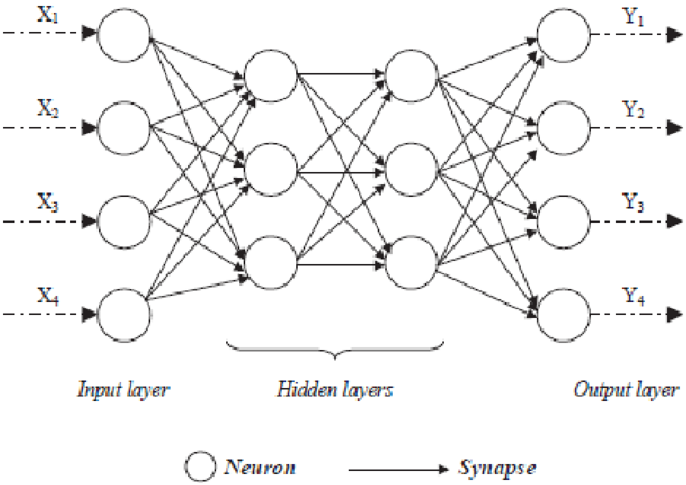
```
source ./kera.sh
store_model "my_model" ./test_model ./test_data ./test_label
create_genome "default" "${MODEL}/my_model.model"
```

```
kerash_mountpoint
└── model
    ├── my_model.model
    └── genomes
        └── genome_default
            ├── meta.dat
            ├── layer_0
            │   ├── meta.dat
            │   └── weights.dat
            └── layer_1
                ├── meta.dat
                └── weights.dat
```

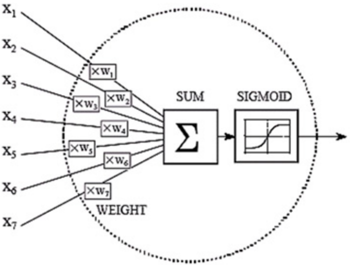
test_model file:

```
2 1 1
input
dense 10 sigmoid
dense 1 sigmoid
```

Graph approach



Activation



$$S(x) = \frac{1}{1 + e^{-x}}$$

$$S'(x) = S(x) \times (1 - S(x))$$

Activation implementation

```
function activ_sigmoid() { echo $(( 1. / (1. + exp(-+$1)) )) }

function activ_d_sigmoid() { echo $(( $(activ_sigmoid "$1") * ...)) }

function activ_relu() { echo $(( $1 < 0 ? 0.0 : $1 )) }

function activ_d_relu() { echo $(( $1 < 0 ? 0.0 : 1.0 )) }

f="sigmoid"
v=$(activ_${f} 0.5)
```

KeraSH offers more than 20 activation functions!

Forward-propagation using matrices

$$z_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} w11 & w12 & w13 \\ w21 & w22 & w23 \end{bmatrix}$$

$$z_1 = \begin{bmatrix} 0 * w11 + 0 * w21 & 0 * w12 + 0 * w22 & 0 * w13 + 0 * w23 \\ 0 * w11 + 1 * w21 & 0 * w12 + 1 * w22 & 0 * w13 + 1 * w23 \\ 1 * w11 + 0 * w21 & 1 * w12 + 0 * w22 & 1 * w13 + 0 * w23 \\ 1 * w11 + 1 * w21 & 1 * w12 + 1 * w22 & 1 * w13 + 1 * w23 \end{bmatrix}$$

$$a_1 = S(z_1)$$

Theorem (Generalized ForwardProp Formulas)

$$z_{n+1} = a_n \cdot w_{n+1} \; ; \; a_n = S(z_n)$$

Forward implementation

```
function predict_dense( )
{
    local dir="$1"
    local activation="$2"
    local layerid="$3"

    matrix_mul  3< "${input_file}" \
                4< "${dir}/weights.dat" \
                 > "$(predict_name $layerid activity)"

    matrix_apply activ_$activation < "$(predict_name $layerid activity)" \
                                   > "$(predict_name $layerid activation)"

    input_file="$(predict_name $layerid activation)"
}
```

Back-Propagation

$$J = \frac{1}{batch\_size} \times \sum (Y_{expected} - Y_{output})^2$$

Theorem (Output Layer Gradient Matrix)

$$\delta = -(Y_{expected} - Y_{output}) \odot S'(z); \frac{\partial J}{\partial W} = a_{n-1}^T * \delta$$

Theorem (Hidden Layers Gradient Matrix)

$$\delta_n = (\delta_{n+1} * W_{n+1}^T) \odot S'(z_n); \frac{\partial J}{\partial W_n} = a_{n-1}^T * \delta_n$$

Back-Propagation Implementation

```
# S'(Zn)
matrix_apply "activ_d_$activation" < $(predict_name $layerid activity) \
                                   > "$(tmp_name 2)"

# DELTAn = DELTA(n+1) o S'(Zn)
matrix_mul_p2p 3< "$(predict_name $nextlayer delta)"  4< "$(tmp_name 2)" > "$(tmp_name 4)"

# A_t(n-1)
matrix_transpose < "$(predict_name $prevlayer activation)" > "$(tmp_name 3)"

# Gradient = A_t(n-1)*DELTAn
matrix_mul 4< "$(tmp_name 4)" 3< "$(tmp_name 3)" > "$(tmp_name 2)"

# Bias Gradient = DELTAn
matrix_mul_scalar 1.0 < "$(tmp_name 4)" > "$(predict_name $layerid bias_gradients)"

# Gradient sum
matrix_add_inplace $gradients $(tmp_name 2) $gradients

# DELTA(n) = DELTA(n) * W_Tn
matrix_mul 3< "$(tmp_name 4)" 4< "$dir/weights_t.dat" \
                 > "$(predict_name $layerid delta)"
```

Multi-Threading in ZSH

```
zsh ./training/_batch_part.zsh $gen_id $batch_size $vec &
pid=$!
wait $pid
```

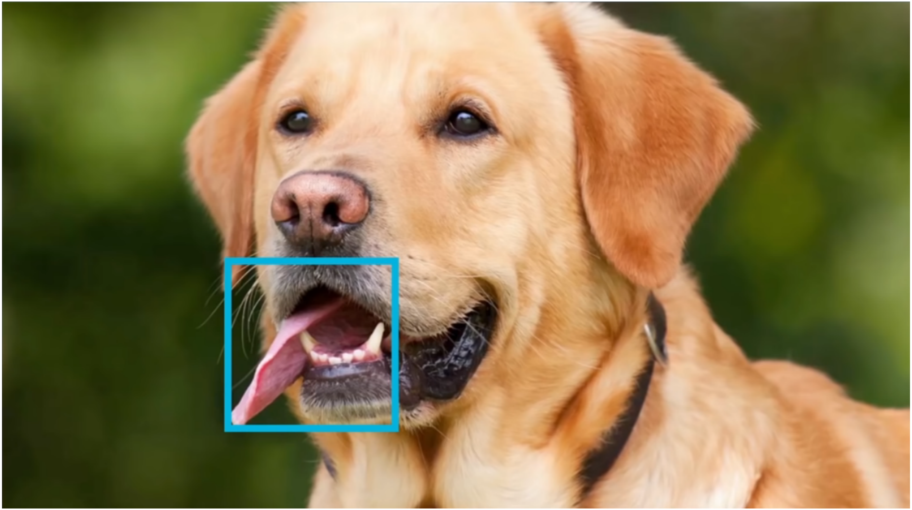
Issues:

- zsh must restart the whole project at each fork
- difficult to effectively synchronize thread

Implementation

- a process has is own temporary folder in the tmpfs (./mat/$$/)
- a process must compute a part of a batch
- a process compute a partial sum of gradients, cost and accuracy
- the main process calculates gradient sums using partial sums, then changes the weights of the layers

Why another topology?



Convolution matrix

35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

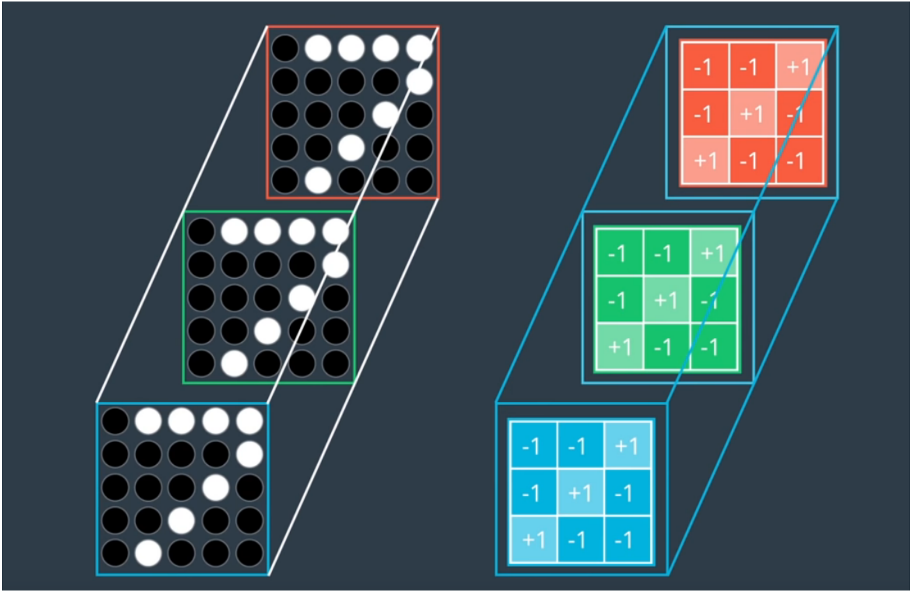
$\times$

	0	1	0	
	0	0	0	
	0	0	0	

$=$

		42		

Use convolution

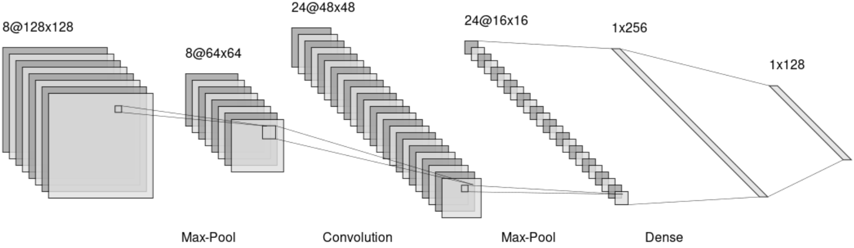


Implementation

```
for (( z = 0; z < d_k; z++ ));
do
    tensor_slice 0 0 $z $w_k $h_k 1 1 0 <&4 > $(tmp_name 1)

    for (( y = 0; y < count_y; y++ ));
    do
        for (( x = 0; x < count_x; x++ ));
        do
            tensor_slice $((x * stride)) $((y * stride)) $z \
                                $w_k $w_k 1 $pad <&3 > $(tmp_name 2)
            $f 3< $(tmp_name 1) 4< $(tmp_name 2)
        done
    done
done
```

Role of pooling layers



Create a CNN model in KeraSH

```
256 256 3
input -
convolution relu 2 1 0 3 3
max_pooling - 1 0 2 2
convolution relu 2 1 0 3 3
max_pooling - 1 0 2 2
flatten -
dense 30 softmax
```

Auto ML Concepts

Objectives:

- Start from an empty topology
- Evaluate network performance on training
- Apply random mutation
- If performance is better, save the model
- Continue to apply mutations

Mutations

- Add a new hidden layer
- Resize and hidden layer
- Change activation function for a layer

Evolution

Train a network of input matrix of size 2x1 on a population of size 1 with 3 iterations/generation:

```
source ./kera.sh
evolve_from_scratch 2 1 xor_ev ./test_data ./test_label 1 3
```

Any questions ?