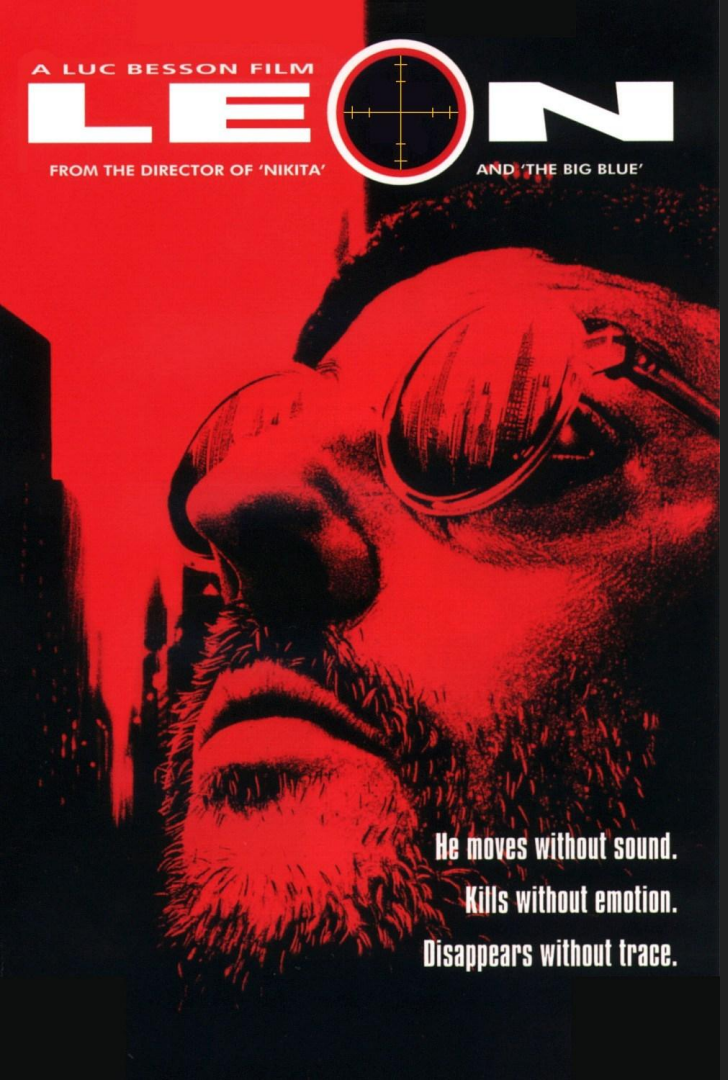
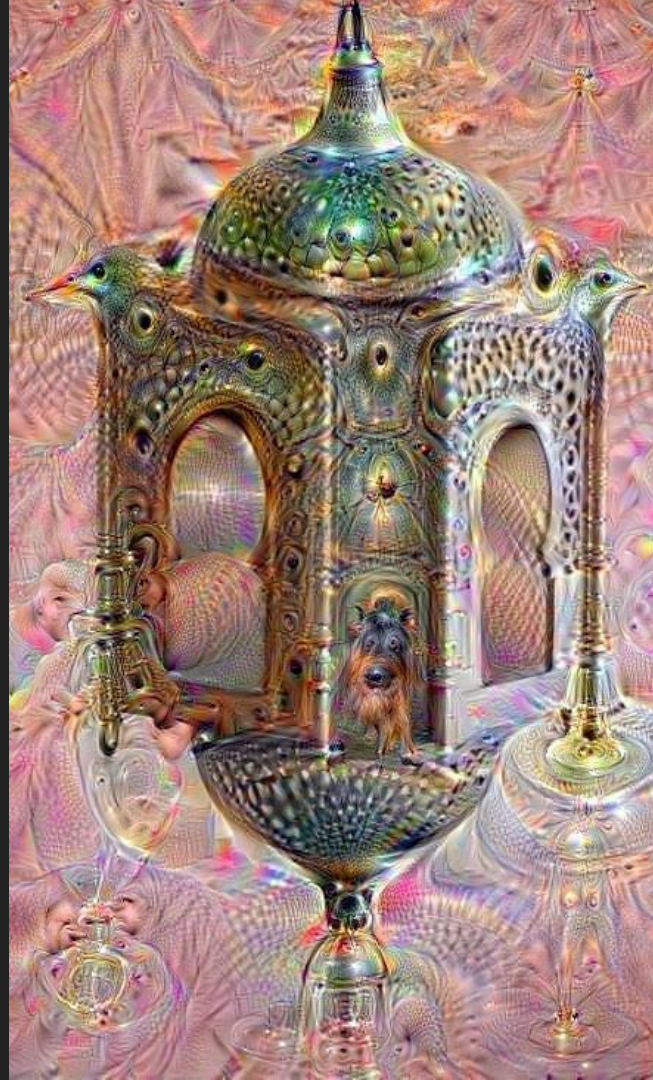


MARIANA

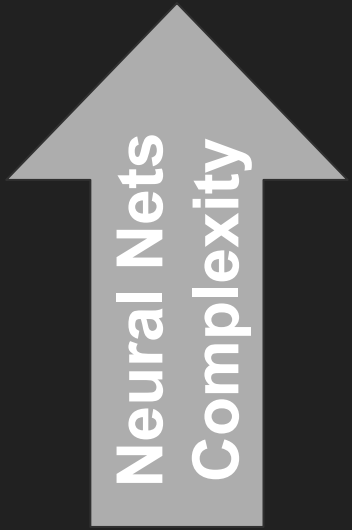
The **Cutest** Deep Learning Framework

Deep Learning is Hot

- Deep Learning is getting out of the labs
- Big companies are interested
- There’s is more ML being done
- More people want to learn it
- Big news: Deep dream, AlphaGO, ...

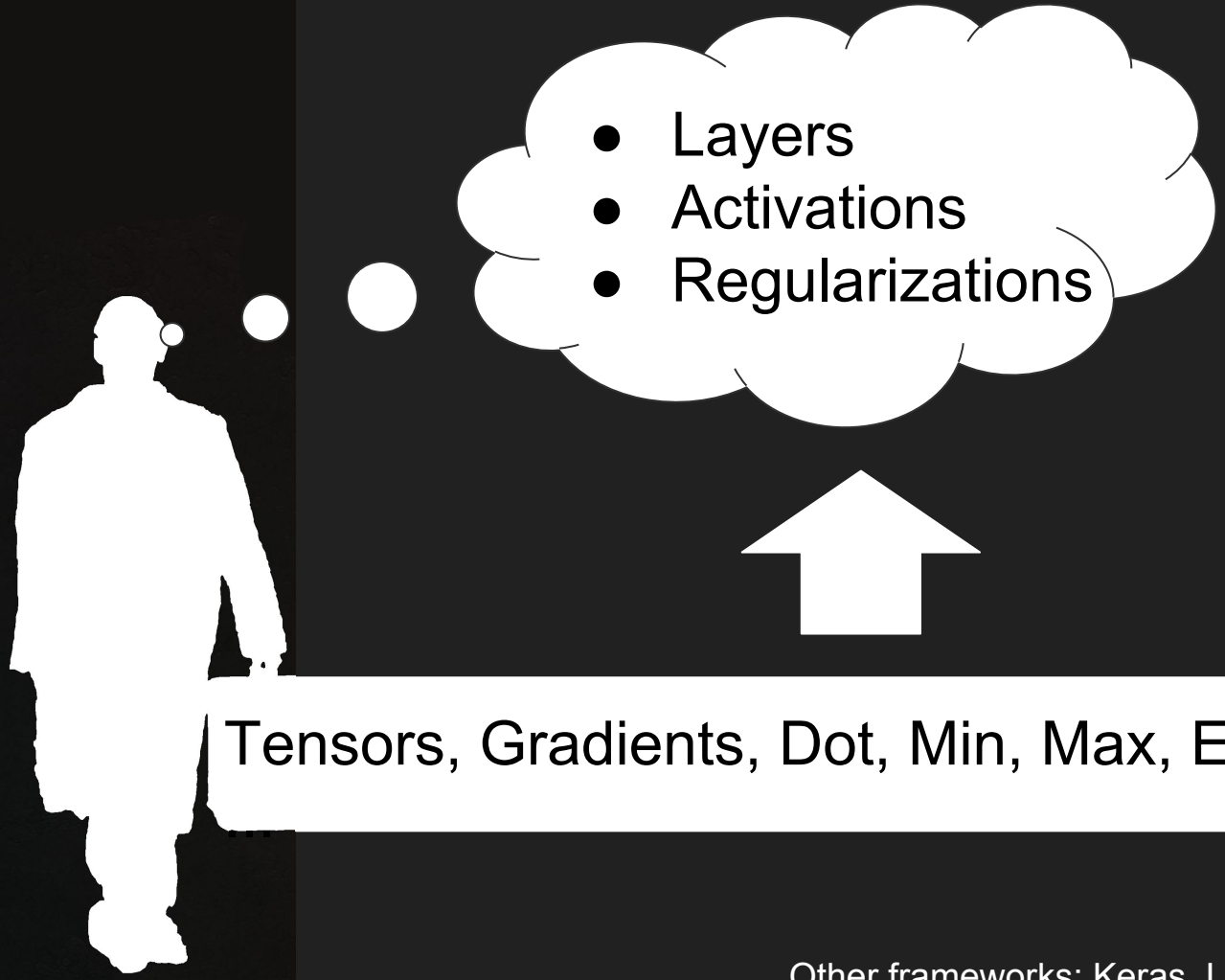


- Repetitive
- Time consuming
- Hard to debug
- Hard to teach



- Very Hard to learn

Leon minimalist poster by landlcreations



Tensors, Gradients, Dot, Min, Max, Exponential, ...

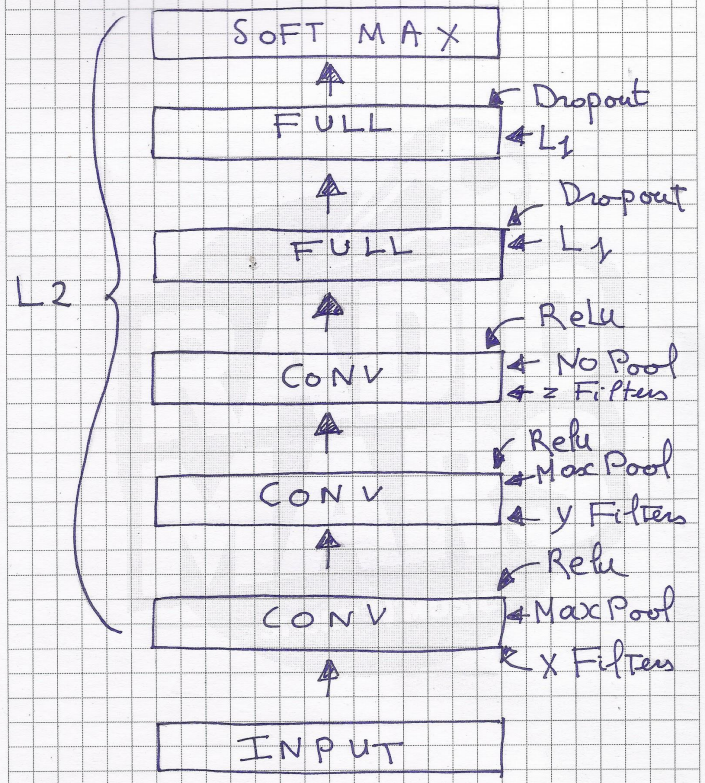
Other frameworks: Keras, Lasagne, Blocks, ...

Networks

Language

Layers, Activations, Regularizations...

Other frameworks: Keras, Lasagne, Blocks, ...



Train: SGD + Momentum

MARIANA

- Feedforward Graphs by default
  - Fully connected
  - Convolution
  - Embeddings
- No restriction in connectivity
  - Multi-Inputs, Multi-Outputs, Forks
- Layer independent hyper-parameters
- Save / Reload models
- Export to HTML for visualization
- Fully Documented (tariqdaouda.com)
- Github (tariqdaouda/Mariana)
- No recurrent nets yet but it is planned

Abstraction: Layers

- Containers
  - Other abstractions
  - Parameters
- Independent
  - Each layer can have its own set of abstractions
- Types
  - Inputs
  - Hiddens (Weight, bias)
  - Output (Weights, bias + Theano functions: train, test, ...)

Abstraction: Layers

```
class SoftmaxClassifier(Classifier_ABC) :
    """A softmax (probabilistic) Classifier"""
    def __init__(self, nbOutputs, learningScenario, costObject, temperature = 1, name = None, **kwargs) :
        Classifier_ABC.__init__(self,
            nbOutputs,
            activation = MA.Softmax(temperature = temperature),
            learningScenario = learningScenario,
            costObject = costObject,
            name = name, **kwargs)
        self.targets = tt.ivector(name = "targets_" + self.name)

    def setCustomTheanoFunctions(self) :
        """defined theano_classify, that returns the argmax of the output"""
        self.classify = MWRAP.TheanoFunction("classify", self, [ tt.argmax(self.outputs) ])
```

Abstraction: Activations

- Activation functions: ReLU, Tanh, ...

```
class Tanh(Activation_ABC):
    def function(self, x):
        import theano.tensor as tt
        return tt.tanh(x)
```

Abstraction: Regularizations

- Added to the cost during training: L1, L2, ...

```
class L1(SingleLayerRegularizer_ABC) :
    def __init__(self, factor) :
        SingleLayerRegularizer_ABC.__init__(self)
        self.factor = factor
        self.hyperparameters = ["factor"]

    def getFormula(self, layer) :
        return self.factor * ( abs(layer.W).sum() )
```

Abstraction: Learning Scenarios

- How to optimize: SGD, Momentum, …
- Can be inherited from Output layers

```
class GradientDescent(LearningScenario_ABC):
    def __init__(self, lr):
        LearningScenario_ABC.__init__(self)
        self.lr = lr
        self.hyperParameters = ["lr"]

    def getUpdates(self, layer, cost) :
        updates = []
        for param in layer.getParams() :
            gparam = tt.grad(cost, param)
            updates.append((param, param - self.lr * gparam))

        return updates
```

Abstraction: Costs

- The objectives to optimize: MSE, Negative log likelihood, ...
- Can only be attached to output layers

```
class MeanSquaredError(Cost_ABC) :
    """The all time classic"""
    def costFct(self, targets, outputs) :
        cost = tt.mean((outputs - targets) ** 2)
        return cost
```

Abstraction: Decorators

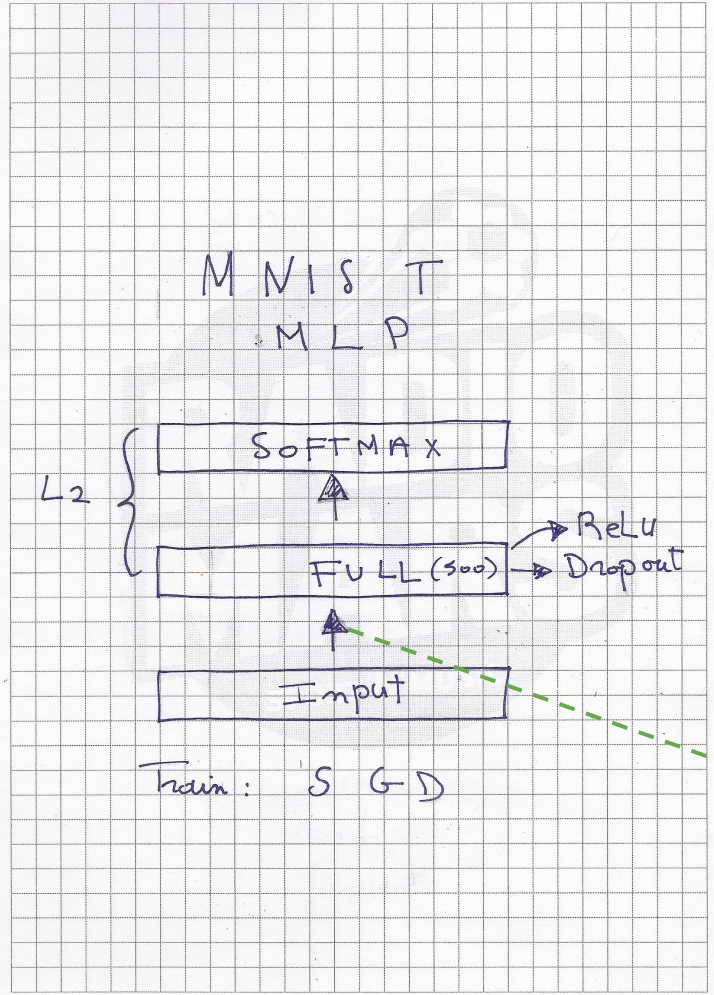
- Modify layer's behaviour: Dropout, ...

```
class GlorotTanhInit(Decorator_ABC) :
    """Set up the layer to apply the tanh initialisation introduced by Glorot et al. 2010"""
    def decorate(self, layer) :
        rng = numpy.random.RandomState(MSET.RANDOM_SEED)
        layer.W = rng.uniform(
                    low = -numpy.sqrt(6. / (layer.nbInputs + layer.nbOutputs)),
                    high = numpy.sqrt(6. / (layer.nbInputs + layer.nbOutputs)),
                    size = (layer.nbInputs, layer.nbOutputs)
                )
```

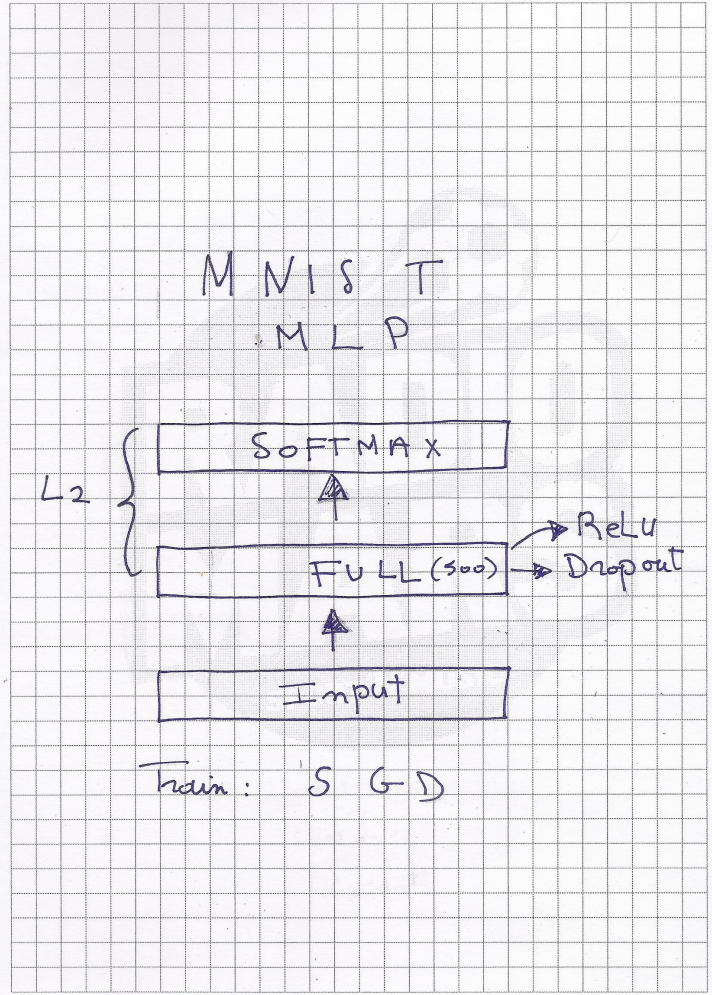
Abstraction: Poolers

- Only for conv layers: Max-pooling, ...

```
class NoPooling(ConvPooler_ABC) :
    """No pooling. The convolution is kept as is"""
    def pool(self, convLayer) :
        hOutputs = convLayer.inputHeight - convLayer.filterHeight + 1
        wOutputs = convLayer.inputWidth - convLayer.filterWidth + 1

        return convLayer.convolution, hOutputs, wOutputs
```




```python
ls = MS.GradientDescent(lr = 0.01)
cost = MC.NegativeLogLikelihood()

i = ML.Input(28*28, name = 'InputLayer')

h = ML.Hidden(500,
    activation = MA.ReLU(),
    decorators = [MD.BinomialDropout(0.1)],
    regularizations = [ MR.L2(0.0001) ],
    name = "Hidden" )

o = ML.SoftmaxClassifier(10,
    learningScenario = ls,
    costObject = cost,
    regularizations = [ MR.L2(0.0001) ],
    name = "OutputLayer")

mlp = i > h > o
```




```python
for epoch in xrange(100) :
    for i in xrange(0, len(train_set[0]), miniBatchSize) :
        score = MLP.train("OutputLayer",
            InputLayer = train_set[0][i : i +miniBatchSize],
            targets = train_set[1][i : i +miniBatchSize] )[0]

    valScore = MLP.test("OutputLayer",
        InputLayer = validation_set[0],
        targets = validation_set[1] )[0]
```

MARIANA: Training abstractions

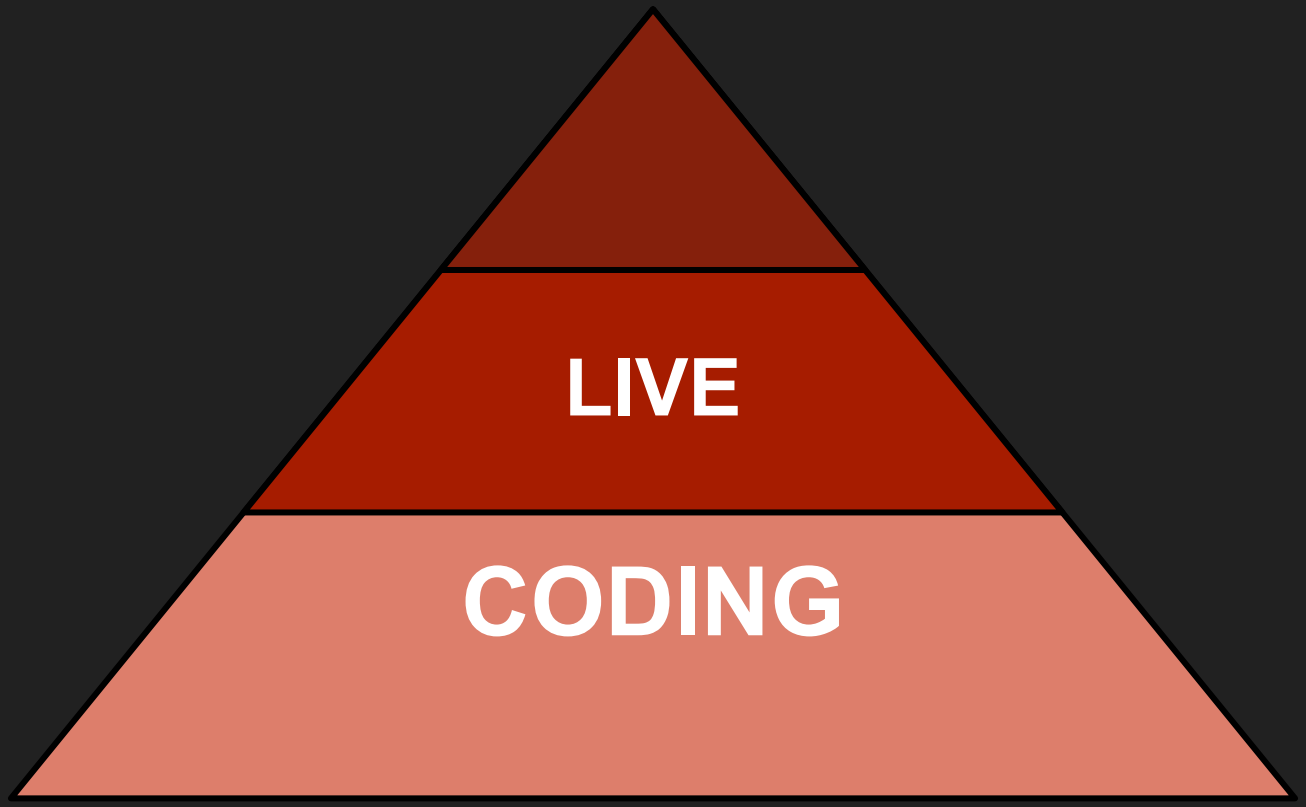
- Trainers (encapsulate all the training)
  - **Emergency savings**
  - Stop criteria (Early stopping, ...)
  - Dataset mappers (Layer => data, Oversampling, ...)
  - Recorders (record / print)

```
def __init__(self,
    trainMaps,
    testMaps,
    validationMaps,
    trainMiniBatchSize,
    stopCriteria = [],
    testMiniBatchSize = -1,
    validationMiniBatchSize = -1,
    saveIfMurdered = True) :
```

```
def start(self,
    runName,
    model,
    recorder = "default",
    trainingOrder = 0,
    moreHyperParameters={}) :
```

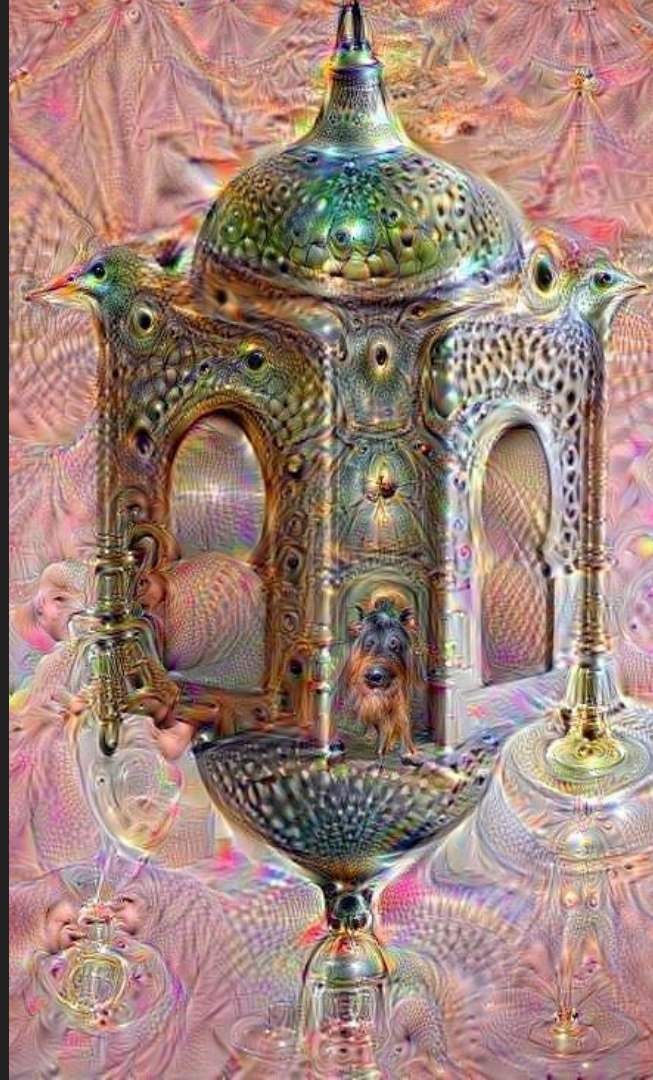
Training Abstractions: Stop Criteria

```
class EpochWall(StopCriterion_ABC) :
    """Stops training when maxEpochs is reached"""
    def __init__(self, maxEpochs) :
        StopCriterion_ABC.__init__(self)
        self.maxEpochs = maxEpochs

    def stop(self, trainer) :
        if trainer.store["runInfos"]["epoch"] >= self.maxEpochs :
            return True
        return False

    def endMessage(self) :
        return "Reached epoch wall %s" % self.maxEpochs
```


LIVE
CODING

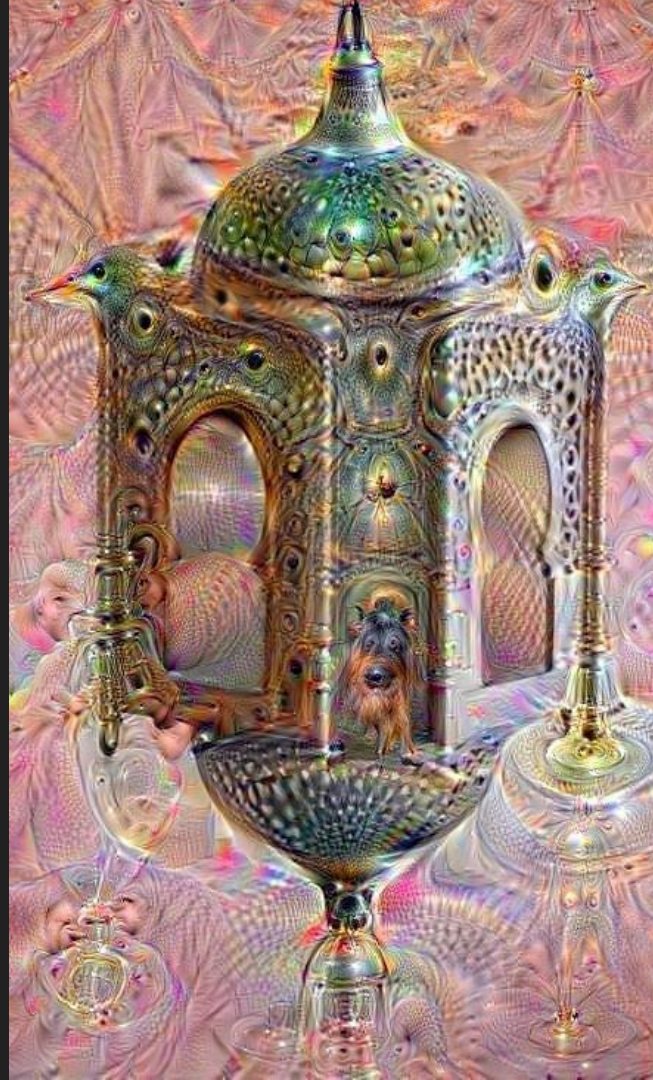
Upcoming Version

- Embeddings for Conv Nets
- New abstraction: Initializations
- Improved model saving and loading (JSON, HDF5)
- Model generation log



Special Thanks

- Testers:
  - Jonathan Séguin (IRIC, UdeM)
  - Assya Trofimov (IRIC, UdeM)
- Theano devs:
  - Frédéric Bastien (MILA, UdeM)
  - Pascal Lamblin (MILA, UdeM)
- Supervisors:
  - Claude Perreault (IRIC, UdeM)
  - Sébastien Lemieux (IRIC, UdeM)



Thank You!

- **Full Documentation**:
  - tariqdaouda.com

- **Code + Examples**
  - **Github**: tariqdaouda/Mariana