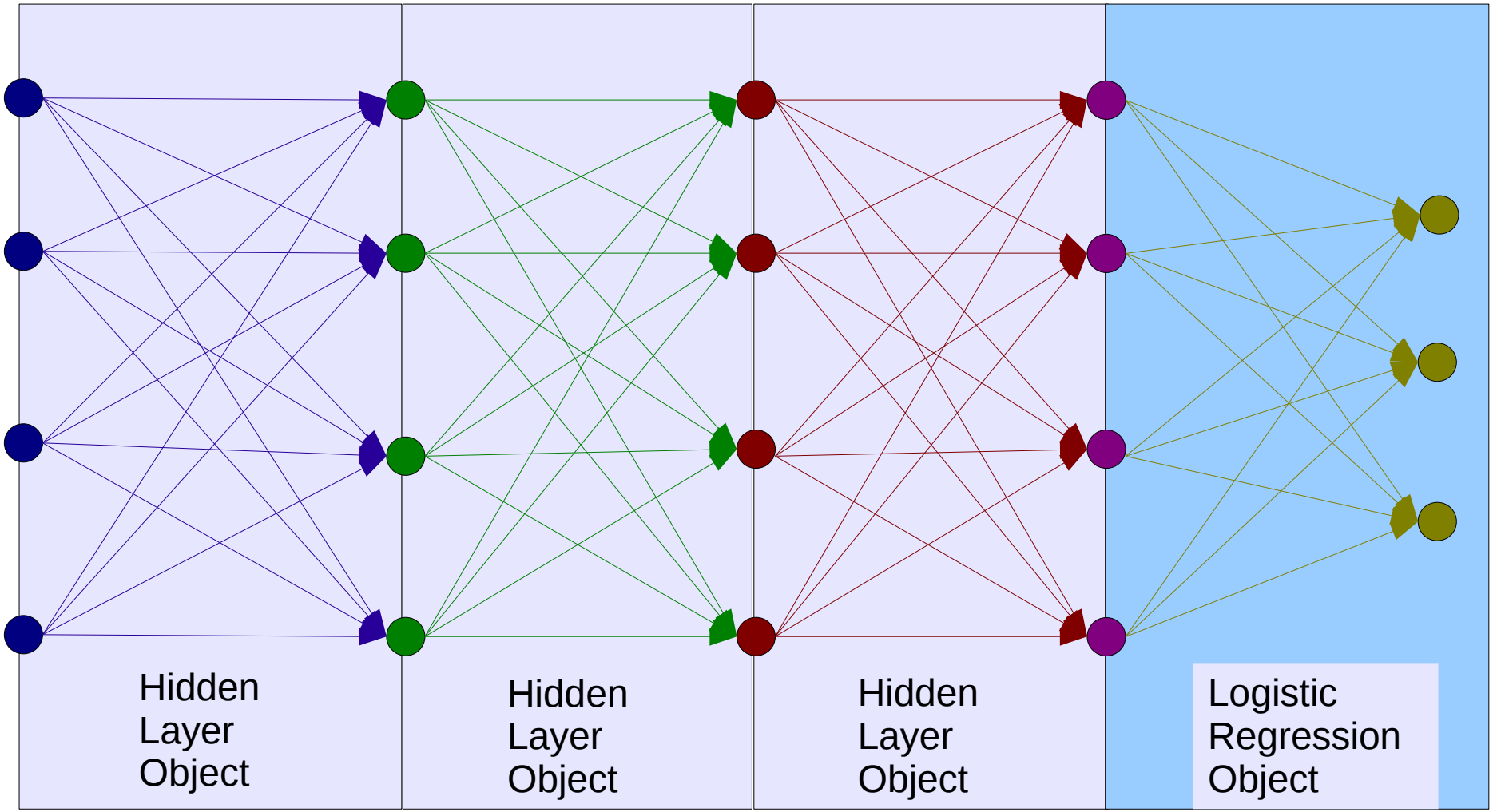
Implementation of Neural Networks with Theano

http://deeplearning.net/tutorial/

Feed Forward Neural Network (MLP)



Inputlayer

Outputlayer

Example for an Output Layer: `LogisticRegression` (`SoftmaxCostLayer`)

- Logistic Regression implemented as Class

  `class LogisticRegression(object)`

- Softmax-Output (Classification)
  - output also for prediction
- Computation of the Cost
  - for training

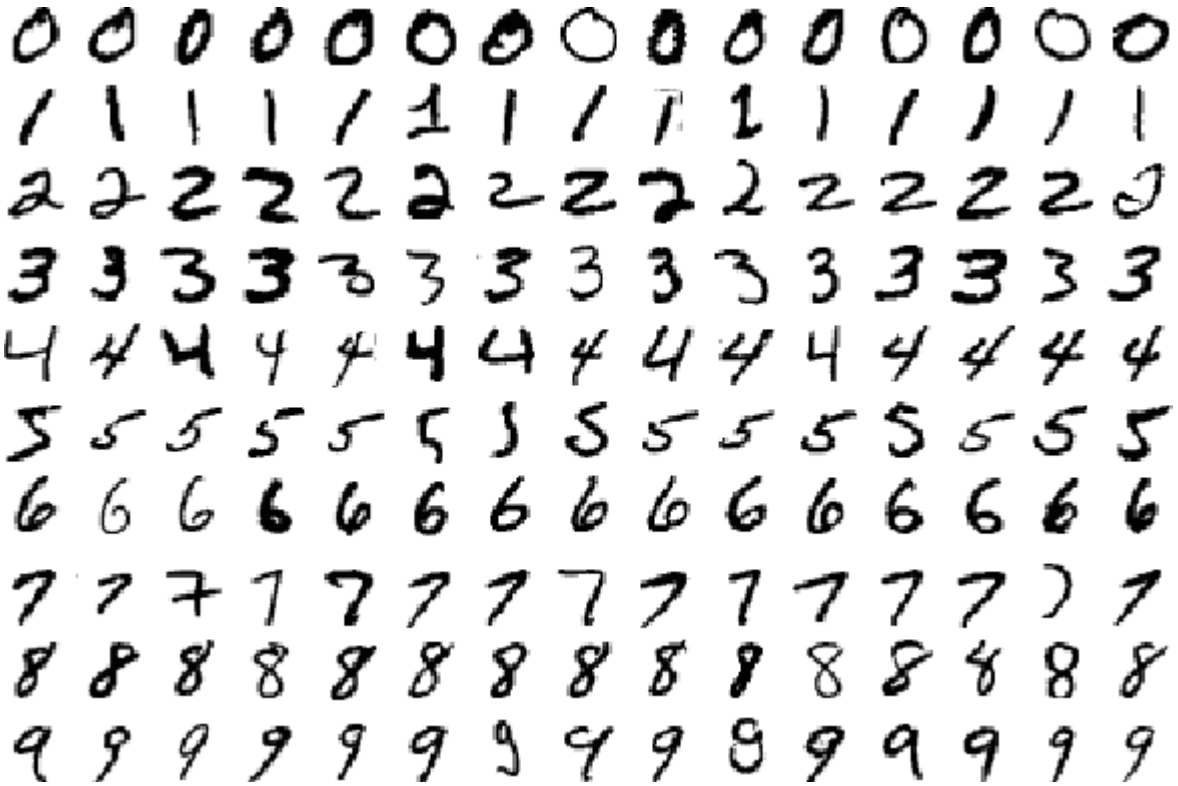
Minibatches

- `input` Data as a Matrix
  - each row corresponds to a training example
  - each column corresponds to a feature
  - input.shape == (mini_batch_size, nb_inputs)

MNIST Dataset

- Handwritten Images (size: 28x28)
- 10 Classes
- input for visilble units: flattened image (vector)

numpy.reshape



Train Error (Classification)

- Target class encoded as "one-hot": 1-of-k
- Cross entropy loss

likelihood: $$\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)} | x^{(i)}, W, b))$$

(average) loss: $$\ell(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D})$$

- cost of a mini batch in Theano

```
return -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])
```

example index

vector with class id values

matrix with shape (batch-size, nb-of-classes)

Class Probabilities

- Probability of each class is given by Softmax

$$P(Y = i|x, W, b) = softmax_i(Wx + b)$$
$$= \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}}$$

matrix with shape (size-of-minibatch, num-classes)

- `self.p_y_given_x = T.nnet.softmax(T.dot(input, self.W) + self.b)`

  - input is a matrix with shape (size-of-minibatch, nb-of-features)
  - the input is the first element of the dot-product, so the rows of the result say the different examples

Class Prediction

$$y_{pred} = \text{argmax}_i P(Y = i | x, W, b)$$

- Symbolic description of how to compute the prediction as the class whose probability is maximal

vector of class labels

```
self.y_pred =
  T.argmax(self.p_y_given_x,
           axis=1)
```

argmax gives the index of a array

the argmax is computed per row (over the column axis = 1)

- Layer has a list of parameters

```
self.params = [self.W, self.b]
```

Model Parameters are Theano Shared Variables

```
self.W = theano.shared(
            value=numpy.zeros(
                (n_in, n_out),
                dtype=theano.config.floatX
            ),
            name='W',
            borrow=True
        )
# initialize the biases b as a vector of n_out 0s
self.b = theano.shared(
            value=numpy.zeros(
                (n_out,),
                dtype=theano.config.floatX
            ),
            name='b',
            borrow=True
        )
```

Gradient

- Learning is done by mini-batch stocastic gradient descent (MSGD)
  - use just a few examples for frequent updates
- Symbolic gradient calculation by theano

```
g_W = T.grad(cost=cost,
             wrt=classifier.W)

g_b = T.grad(cost=cost,
             wrt=classifier.b)
```

- in ML-Libraries typically encapsulated in a "Trainer-Class" (not in the Tutorial)

Updates

- Update Rule (for each parameter)

$$\theta \leftarrow \theta - \alpha \frac{\partial Cost}{\partial \theta}$$

- Store updates in a list for the `train_fn`

```
updates =
  [(classifier.W,
    classifier.W - learning_rate * g_W),
   (classifier.b,
    classifier.b - learning_rate * g_b)]
```

- generate symbolic variables for input (x and y represent a minibatch)

```
x = T.matrix('x')  # data, presented as rasterized
                   # images


y = T.ivector('y')  # labels, presented as 1D vector
                    # of [int] labels
```

- construct the logistic regression class
- Each MNIST image has size 28*28

```
classifier = LogisticRegression(input=x, n_in=28 * 28,
n_out=10)
```

Compilation of the training function by `theano.function`

`train_set_x, train_set_y` are stored in shared variables, so they could be hold in GPU memory.

With `givens` a dictionary is given for the computatiional graph

`y` is the target used in the cost function
`x` is the input used in the constructor call

```
train_model = theano.function(
        inputs=[index],
        outputs=cost,
        updates=updates,
        givens={ x: train_set_x[index * batch_size:
                                (index + 1) * batch_size],
                 y: train_set_y[index * batch_size:
                                (index + 1) * batch_size]
        }
    )
```

Classification error for Test and Validation

- For Validation and Testing

```
T.mean(T.neq(self.y_pred, y))
```

is 1 if both values are same, else 0

```
test_model = theano.function(
    inputs=[index],
    outputs=classifier.errors(y),
    givens={ x: test_set_x[index * batch_size:
        (index + 1) * batch_size],
             y: test_set_y[index * batch_size:
        (index + 1) * batch_size]
         }
     )
```

same for validation_model (with validation_set_x and ..y)

- Training in Loop:
  - maximally `n_epochs`

```
while (epoch < n_epochs) and (not done_looping):
```

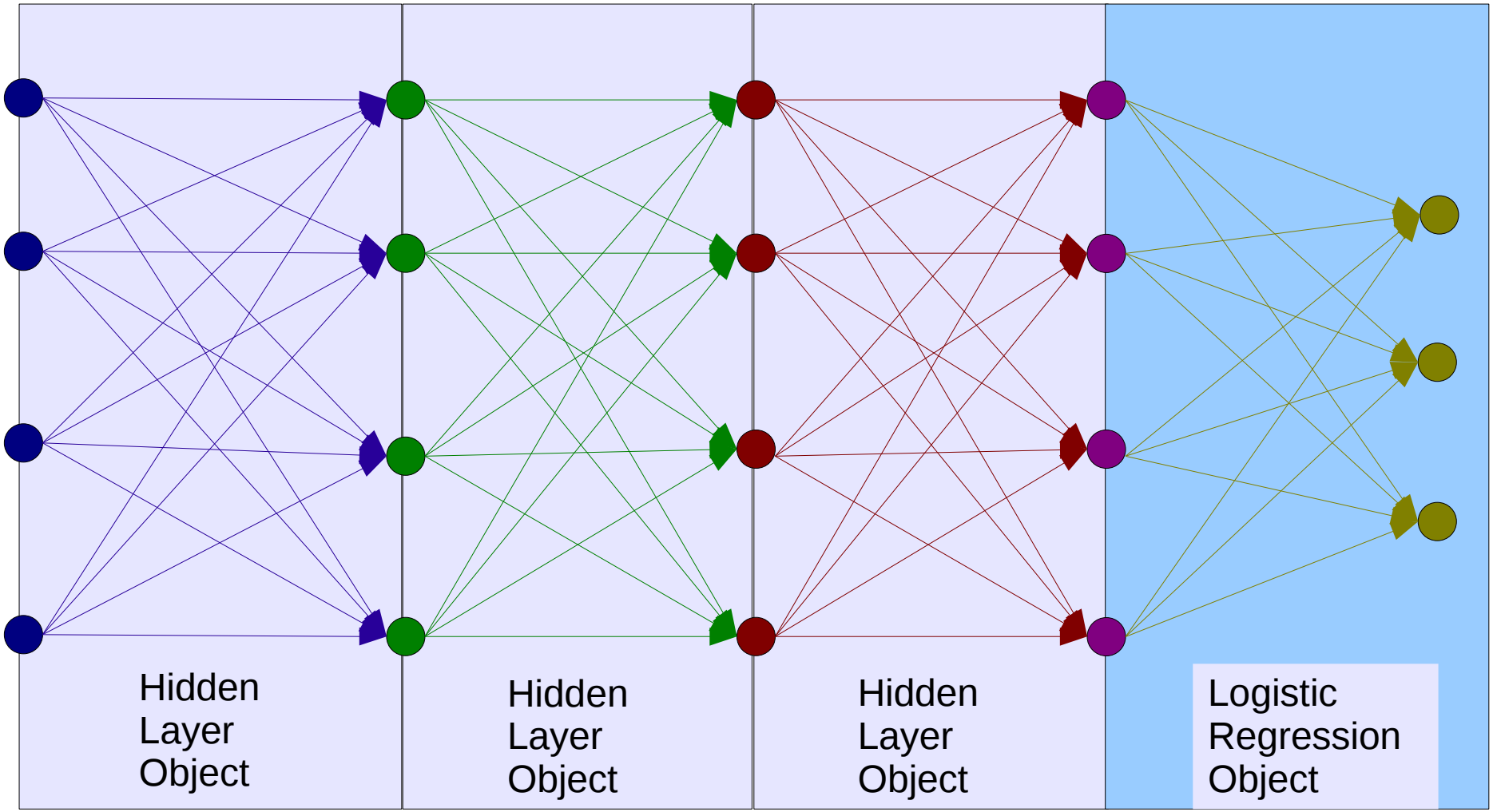
- Early Stopping
  - based Validation error + Patience hyperparam

    if condition satiesfied

    `not_done_looping = True`

http://deeplearning.net/tutorial/logreg.html

Feed Forward Neural Network (MLP)



Inputlayer

Outputlayer

Hidden Layers

- implemented as Class

  `class HiddenLayer(object)`

- Each layer computes a non-linear transformation:

$$\vec{a}^{(l+1)} = activation\left(W^{(l)} \cdot \vec{a}^{(l)} + \vec{b}^{(l)}\right)$$

$$input = \vec{a}^{(0)}$$

Activation Functions

- tanh
- sigmoid (logistic function)
- rectifiers
  - linear rectified units
  - maxout

Weights Initialization

- e.g. tanh weight initialization

$$\left[-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}\right]$$

  - X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, International conference on artificial intelligence and statistics 2010

- Pretraining (see e.g. autoencoders)
- Orthogonal Initialization

Andrew M. Saxe, James L. McClelland, Surya Ganguli: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, ICLR 2014

see also
http://christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/weightInitialization.html

Connecting the layers
(in Tutorial `class MLP(object)`)

```
self.hiddenLayer = HiddenLayer(
            rng=rng,
            input=input,
            n_in=n_in,
            n_out=n_hidden,
            activation=T.tanh
        )


self.logRegressionLayer = LogisticRegression(
            input=self.hiddenLayer.output,
            n_in=n_hidden,
            n_out=n_out
)
```

`self` corresponding to `MLP`

L1 and L2 Regularization

```
self.L1 = (
          abs(self.hiddenLayer.W).sum()
          + abs(self.logRegressionLayer.W).sum()
      )


self.L2_sqr = (
          (self.hiddenLayer.W ** 2).sum()
          + (self.logRegressionLayer.W ** 2).sum()
      )
```

- `self` corresponds to `MLP`

```
self.L1 = (
            abs(self.hiddenLayer.W).sum()
            + abs(self.logRegressionLayer.W).sum()
        )

self.L2_sqr = (
            (self.hiddenLayer.W ** 2).sum()
            + (self.logRegressionLayer.W ** 2).sum()
        )

self.negative_log_likelihood = (
     self.logRegressionLayer.negative_log_likelihood

     )

self.params = self.hiddenLayer.params +
                self.logRegressionLayer.params
```

- `self` corresponds to `MLP`

Gradient Calculation/Update Rules

```
cost = (
        classifier.negative_log_likelihood(y)
        + L1_reg * classifier.L1
        + L2_reg * classifier.L2_sqr
    )

gparams = [T.grad(cost, param) for param in
                              classifier.params]

updates =
   [(param, param - learning_rate * gparam)
      for param, gparam in zip(classifier.params,
                               gparams)
   ]
```

- Training in Loop:
  - maximally `n_epochs`

```
while (epoch < n_epochs) and (not done_looping):
```

- Early Stopping
  - based Validation error + Patience hyperparam

    if condition satiesfied

    `not_done_looping = True`

http://deeplearning.net/tutorial/mlp.html

Hyperparameters

- Learning rate
- How many layers, how many units per layer
- Regularization
- Activation function of the units
- Momentum
- etc.

- Yoshua Bengio, Practical recommendations for gradient-based training of deep architectures,Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science Volume 7700, 2012 http://arxiv.org/abs/1206.5533
- YA LeCun, L Bottou, GB Orr, KR Müller: Efficient BackProp, Neural Networks: Tricks of the Trade 1998

Libraries/Frameworks

- Pylearn2 (based on Theano)
- Torch
- Caffe (mainly focused on vision)
- for a complete list see
  - http://deeplearning.net/software_links/