Lecture 3:

CNN: Back-propagation

boris. ginzburg@intel.com

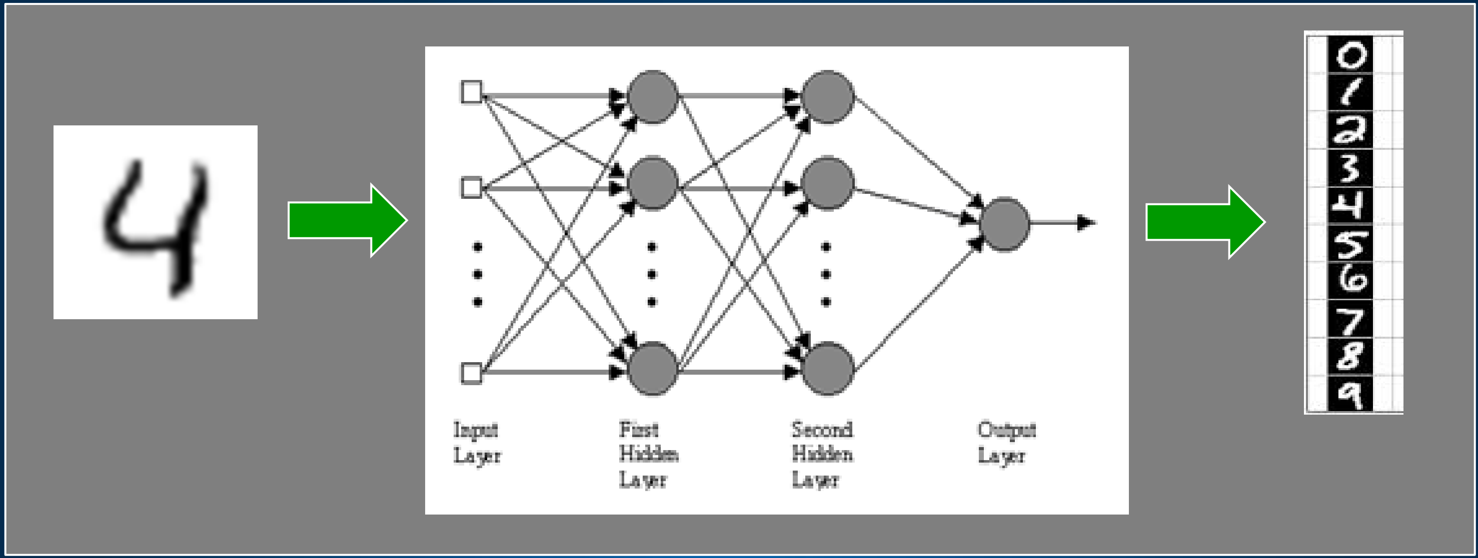
Agenda

- Introduction to gradient-based learning for Convolutional NN
- Backpropagation for basic layers
  - Softmax
  - Fully Connected layer
  - Pooling
  - ReLU
  - Convolutional layer
- Implementation of back-propagation for Convolutional layer
- CIFAR-10 training

Good Links

1. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
2. http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html#flowgraph

Gradient based training



Conv. NN is a function $y = f(x_0, w)$, where

- x_0 is image [28,28],
- w – network parameters (weights, bias)
- y – softmax output= probability that x belongs to one of 10 classes 0..9

Gradient based training

We want to find parameters *W*, to minimize an error

$E\,(f(x_0,w),y_0) = -\log\,(f(x_0,w) - y_0)$.

For this we will do iterative gradient descent:

$w(t) = w(t-1) - \lambda * \frac{-\partial E}{\partial w}(t)$

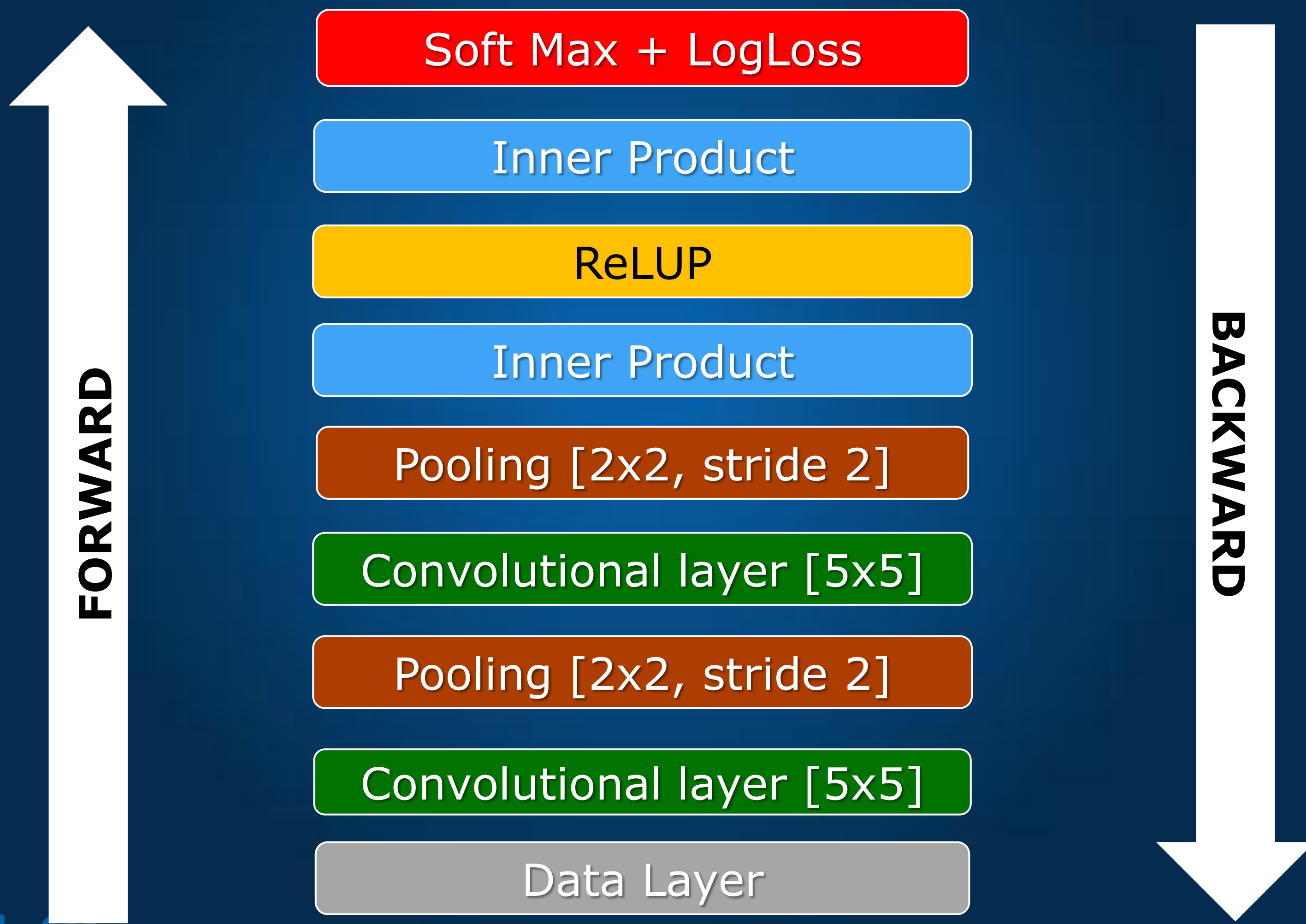
How do we compute gradient of *E* wrt weights?

Loss function E is chain of functions. Let' s go layer by layer, from last layer back, and use the chain rule for gradient of complex functions:

$$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial y_{l-1}}$$

$$\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial w_l}$$

LeNet topology



Layer:: Backward( )

```
class Layer {
    Setup (bottom, top);          // initialize layer
    Forward (bottom, top);        //compute : y_l = f(w_l, y_{l-1})
    Backward( top, bottom);       //compute gradient
}
```

Backward: we start from gradient $\frac{\partial E}{\partial y_l}$ from last layer and

1) propagate gradient back : $\frac{\partial E}{\partial y_l} \rightarrow \frac{\partial E}{\partial y_{l-1}}$

2) compute the gradient of E wrt weights w_l: $\frac{\partial E}{\partial w_l}$

Softmax with LogLoss Layer

Consider the last layer, softmax with log-loss (MNIST example):

$$E = -\log(p_{y0}) = -\log(\frac{e^{y0}}{\sum_0^9 e^{yk}}) = -y0 + \log(\sum_0^9 e^{yk})$$

For all k=0..9 , except k_0 (right answer) we want to decrease $p_{k:}$

$$\frac{\partial E}{\partial y_k} = \frac{e^{y_k}}{\sum_0^9 e^{y_k}} = p_k ,$$

for k=k_0 (right answer) we want to increase $p_{k:}$

$$\frac{\partial E}{\partial y_{k0}} = -(1 - p_{k0})$$

See http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression

Inner product (Fully Connected) Layer

Fully connected layer is just Matrix – Vector multiplication:

$$y_l = W_l * y_{l-1}$$

So $$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} * {W_l}^T$$

and $$\frac{\partial E}{\partial W_l} = \frac{\partial E}{\partial y_l} * y_{l-1}$$

Note: we need y_{l-1}, so we should keep them on forward pass.

ReLU Layer

Rectified Linear Unit :

$$y_l = \max\ (0, y_{l-1})$$

so

$$\frac{\partial\ L}{\partial y_{l-1}} = \begin{cases} = 0, \quad if\ (y_l < 0) \\ = \dfrac{\partial\ L}{\partial y_l}, otherwise \end{cases}$$

ICRI-CI
Intel Collaborative Research Institute
Computational Intelligence

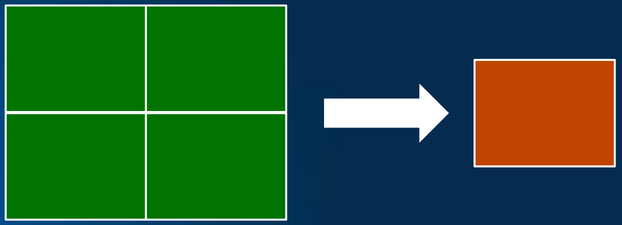
Max-Pooling Layer

Forward :

for (p = 0; p< k; p++)

for (q = 0; q< k; q++)

y_n (x, y) = max(y_n (x, y), y_{n-1}(x + p, y + q));



Backward:

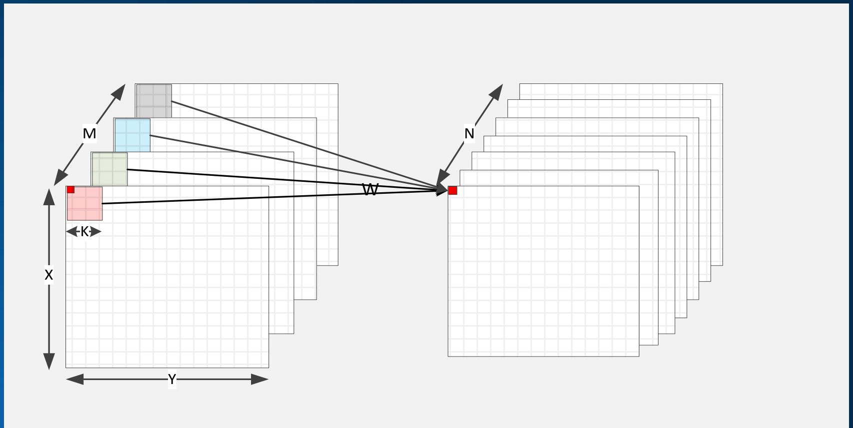
$$\frac{\partial L}{\partial y_{n-1}}(x+p, y+q) = \begin{cases} = 0, if\ (\ y_n\,(x,y)! = y_{n-1}\,(x+p, y+q)) \\ = \dfrac{\partial L}{\partial\, y_n}(x,y), \qquad otherwise \end{cases}$$

Quiz:

1. What will be gradient for Sum-pooling?
2. What will be gradient if pooling areas overlap? (e.g. stride =1)?

Convolutional Layer :: Backward

```
for (n = 0;  n < N;  n ++)
  for (m = 0;  m < M;  m ++)
    for(y = 0; y<Y; y++)
      for(x = 0; x<X; x++)
        for (p = 0; p< K; p++)
          for (q = 0;  q< K; q++)
            y_L (n; x, y) +=  y_L-1(m, x+p, y+q) * w (n ,m;  p, q);
```



Let' s use the chain rule for convolutional layer

$$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial y_{l-1}};$$

$$\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial w_{l-1}}$$

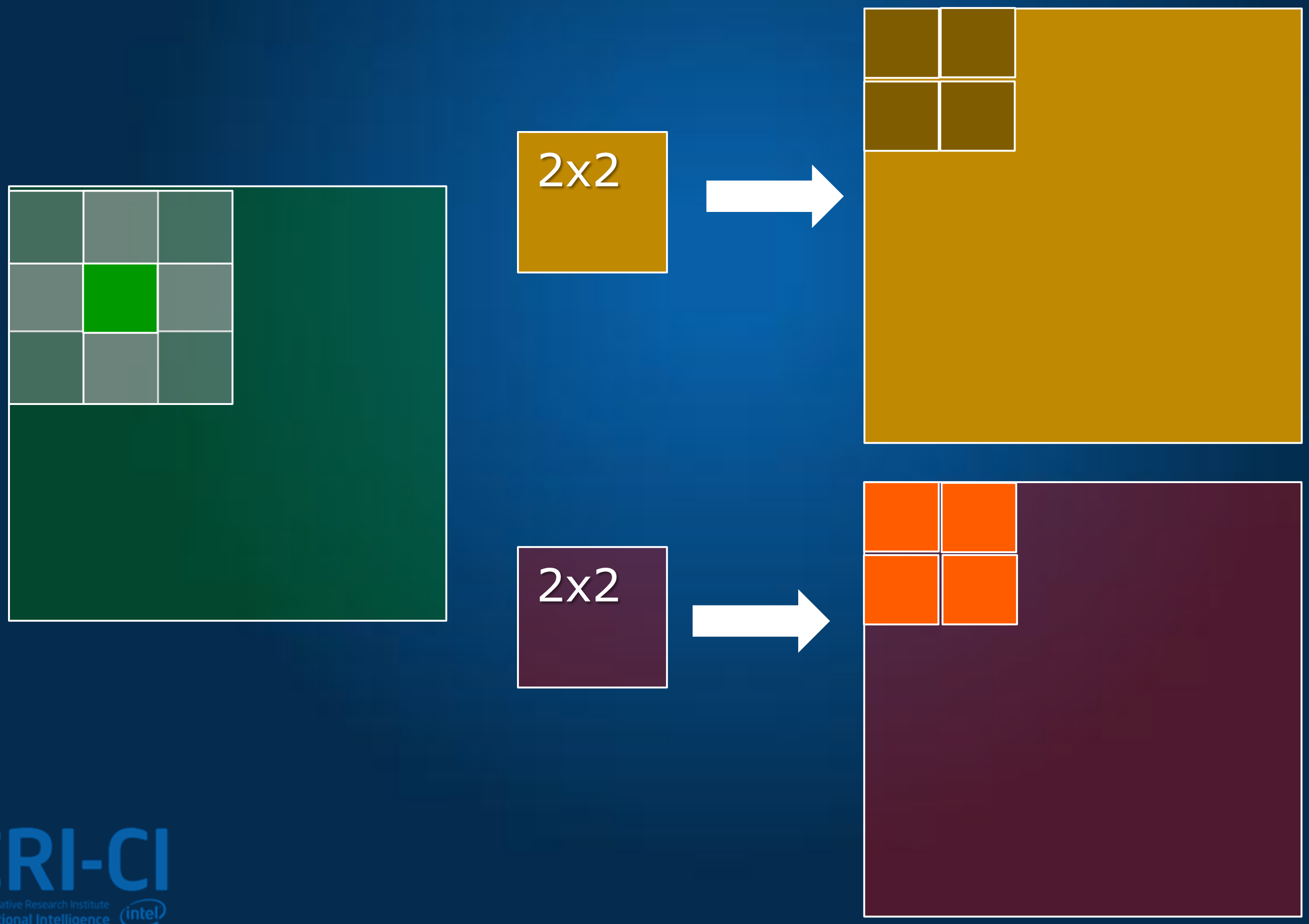
ICRI-CI Intel Collaborative Research Institute Computational Intelligence intel

intel

Convolutional Layer :: Backward

Example: M=1, N=2, K=2.

Take one pixel in level (n-1). Which pixels in next level are influenced by it?



Convolutional Layer :: Backward

Let' s use the chain rule for convolutional layer:

Gradient $\frac{\partial E}{\partial y_{l-1}}$ is sum of convolution with gradients $\frac{\partial E}{\partial y_l}$ over all feature maps from "upper" layer:

$$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial y_{l-1}} = \sum_{n=1}^{N} back\_corr(W, \frac{\partial E}{\partial y_l})$$

Gradient of E wrt w is sum over all "pixels" (x,y) in the input map :

$$\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial l} \times \frac{\partial y_l(w, y_{l-1})}{\partial w_l} = \sum_{\substack{0 \le x \le X \\ 0 < y < Y}} \left(\frac{\partial E}{\partial y_l}(x, y) \circ y_{l-1}(x, y) \right)$$

Convolutional Layer :: Backward

How this is implemented:

```
backward ( ) { ...
// im2col  data to col_data
 im2col_cpu( bottom_data , CHANNELS_, HEIGHT_,  WIDTH_, KSIZE_, PAD_, STRIDE_,
                 col_data);
 // gradient w.r.t. weight.:
caffe_cpu_gemm (CblasNoTrans, CblasTrans,  M_, K_, N_, 1.,  top_diff,    col_data , 1.,
                 weight_diff );
// gradient w.r.t. bottom data:
caffe_cpu_gemm (CblasTrans, CblasNoTrans, K_, N_, M_, 1., weight ,  top_diff , 0.,
                 col_diff );
// col2im back to the data
col2im_cpu(col_diff, CHANNELS_, HEIGHT_, WIDTH_, KSIZE_, PAD_, STRIDE_,
                 bottom_diff );
}
```

Convolutional Layer : im2col

Implementation is based on reduction of convolution layer to matrix – matrix multiply (See Chellapilla et all , “High Performance Convolutional Neural Networks for Document Processing”)

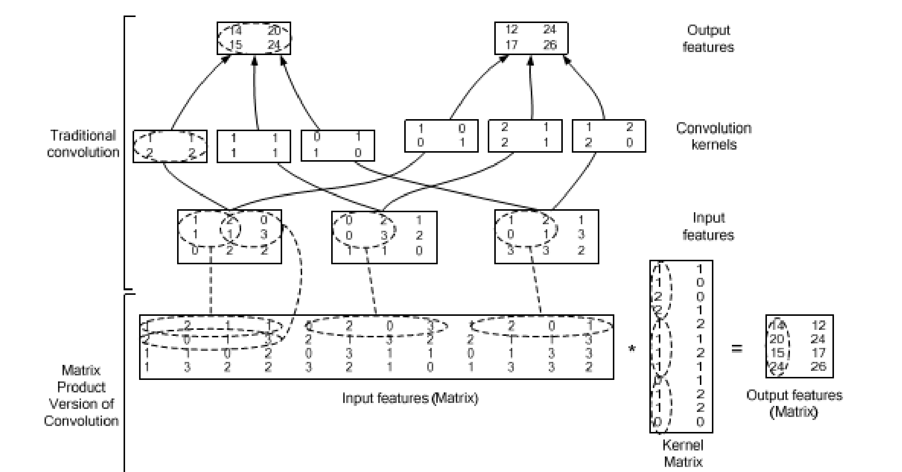


Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.

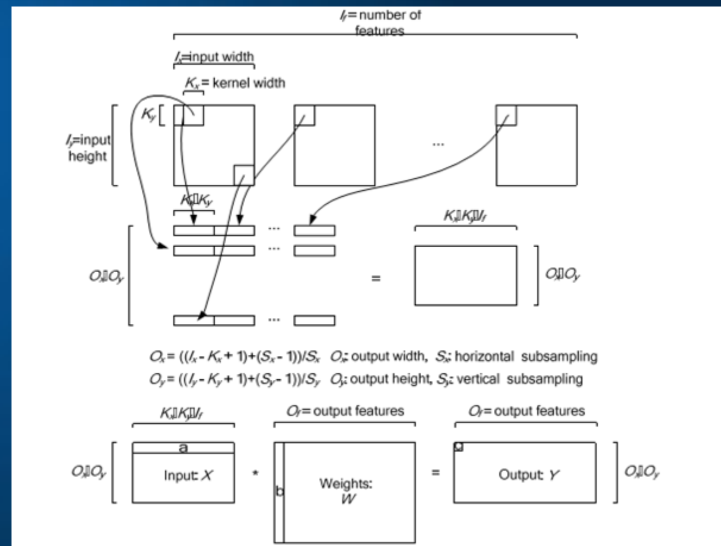


Figure 3. Unrolling the convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted), to produce a matrix-matrix product version.

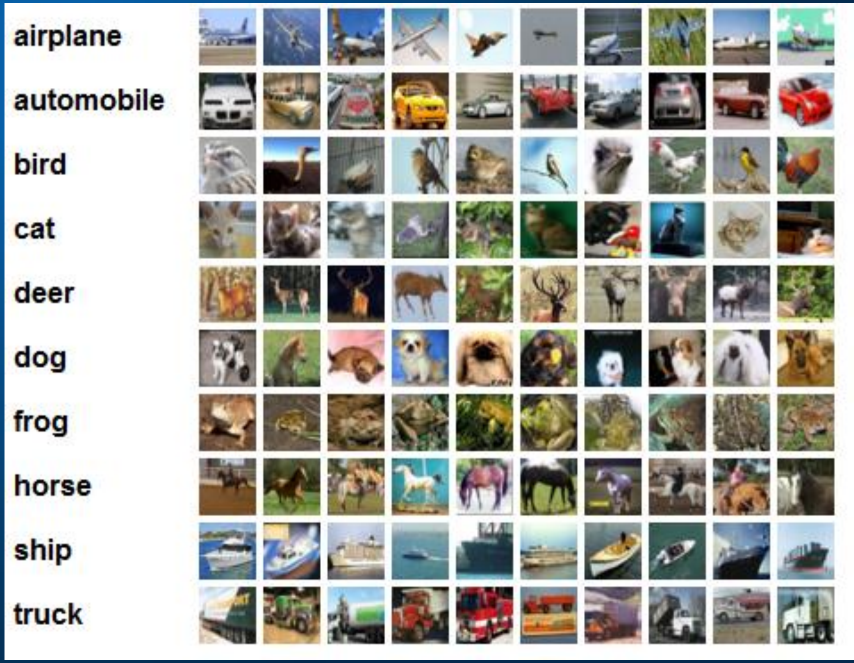
CIFAR-10 Training

http://www.cs.toronto.edu/~kriz/cifar.html

https://www.kaggle.com/c/cifar-10

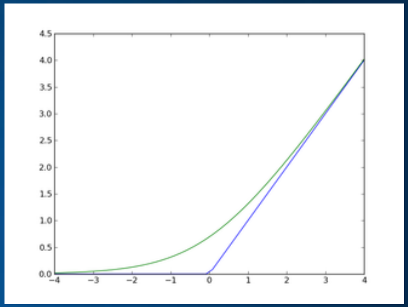
60000 32x32 colour images in 10 classes, with 6000 images per class. There are:

- 50000 training images
- 10000 test images.



Exercises

1. Look at definition of following layers (Backward)
   - sigmoid, tanh
2. Implement a new layer:
   - softplus $y_l = \log(1 + e^{y_{l-1}})$



3. Train CIFAR-10 with different topologies

Project:

1. Port CIFAR-100 to caffe