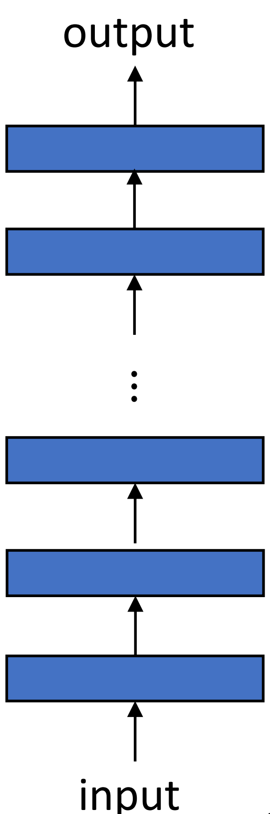
Intro to ML

December 6th, 2021

CHAPTER 12:

Deep Learning

Deep Neural Networks



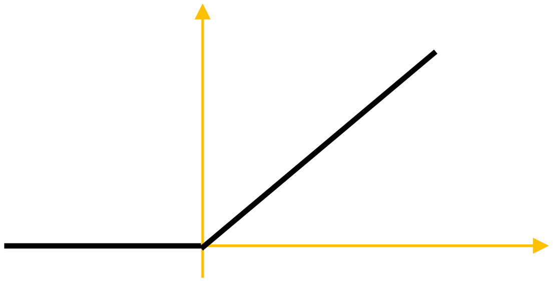
- Many hidden layers
  - Problematic in training: chain rule multiplication of derivatives (vanish of gradients or explosion)
- End-to-end training
- Learn increasingly abstract representations with minimal human contribution
  - Layers of abstraction in intuitive
  - Vision, speech, language, and so on

Representation learning is really the KEY behind Deep learning

Activation function: Rectified Linear Unit (ReLU)

Left derivative:
Relu’(a) = 1 if a>0, 0 otherwise

$$\mathrm{ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$



- Does not saturate for large *a*
- Leads to a sparse representation
    - Some of the nodes will results in 0
- No learning for *a*<0, be careful with initialization
    - Initialization should make sure all weights are positive
- Leaky ReLU: {a if a>0, αa otherwise, α is set 0.01)

Some derivative left

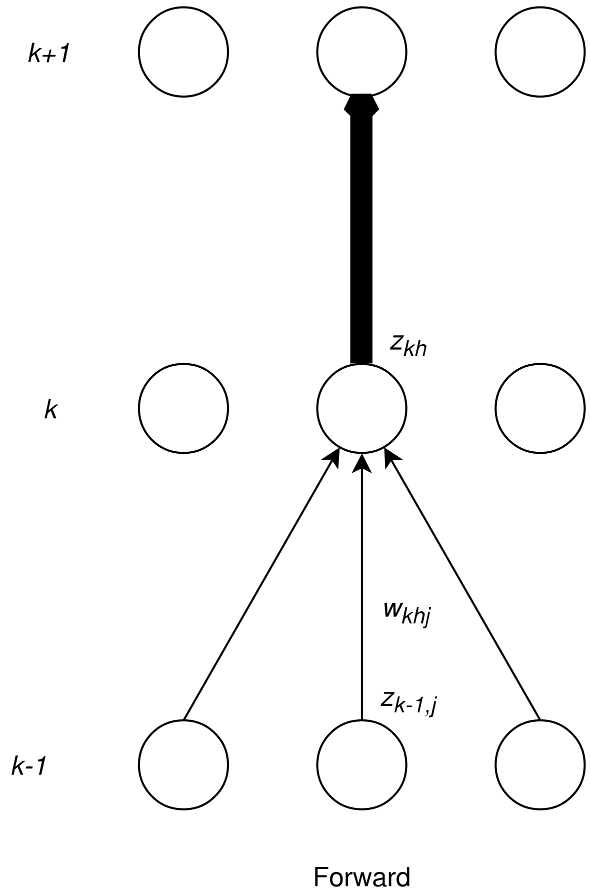
Generalizing Backpropagation

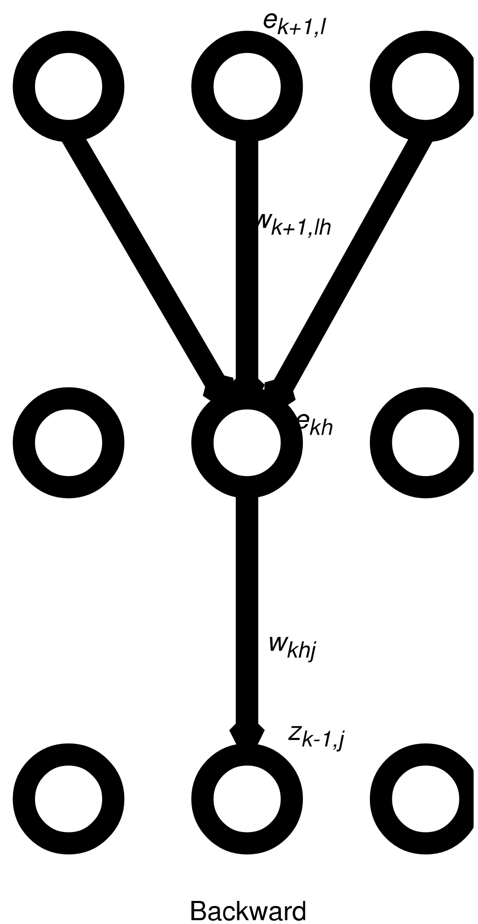
$$z_{kh}^t = f\left(\sum_j w_{khj} z_{k-1,j}^t\right)$$

Forward pass

$$e_{kh}^t = \sum_l e_{k+1,l}^t w_{k+1,lh}$$

Error





$$\Delta w_{khj} = \eta \sum_t e_{kh}^t f'(z_{kh}^t) z_{k-1,j}^t$$

- If k is the output layer, there is no layer k+1
  - e_{kh}^t is the derivative of the error function with respect to the output z_{kh}.
- If not, follow the previous backward rules
- f' is the derivative of the activation function
- Then follow simple stochastic gradient descent for every weight to update the model parameter

Improving Training Convergence

- Gradient decent. Simply, local, changes uses only the values of the presynaptic and postsynaptic and the error (suitably backpropagated) – can converge slowly

- Momentum. At each update, also add a fraction of the average of past gradients (t denotes iteration):

$$\begin{aligned} s_i^t &= \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \\ \Delta w_i^t &= -\eta s_i^t \end{aligned}$$

Running average of past gradient

Update using such an average

Adaptive Learning Factor

- RMSprop. Make update inversely proportional to the sum of past gradients, so, update more when gradient is small and less where it is large.

Magnitude of the gradient

$$\Delta w_i^t = -\frac{\eta}{\sqrt{r_i^t}} \frac{\partial E^t}{\partial w_i}$$

where r_i is the accumulated past gradient,

$$r_i^t = \rho r_i^{t-1} + (1-\rho) \left| \frac{\partial E^t}{\partial w_i} \right|^2$$

p is generally set at 0.999

ADAM: Adaptive Learning Factor w/ Momentum

First moment – just the gradient (momentum)

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \qquad \Delta w_i^t = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}$$

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left|\frac{\partial E^t}{\partial w_i}\right|^2$$

Second moment –square of gradient (adaptive learning factor)

Initially both *s* and *r* terms are 0, so we bias-correct them (both α and ρ are <1, so they get smaller as *t* gets large):

$$\tilde{s}_i^t = \frac{s_i^t}{1-\alpha^t} \text{ and } \tilde{r}_i^t = \frac{r_i^t}{1-\rho^t}$$

Correction for different iterations

Batch Normalization

- Z-normalize hidden unit values (m and s are mean and stdev over the current minibatch):

$$\tilde{a}_j = \frac{a_j - m_j}{s_j}$$

- Can then rescale (γ and β are also learned):

$$\hat{a}_j = \gamma_j \tilde{a}_j + \beta_j$$

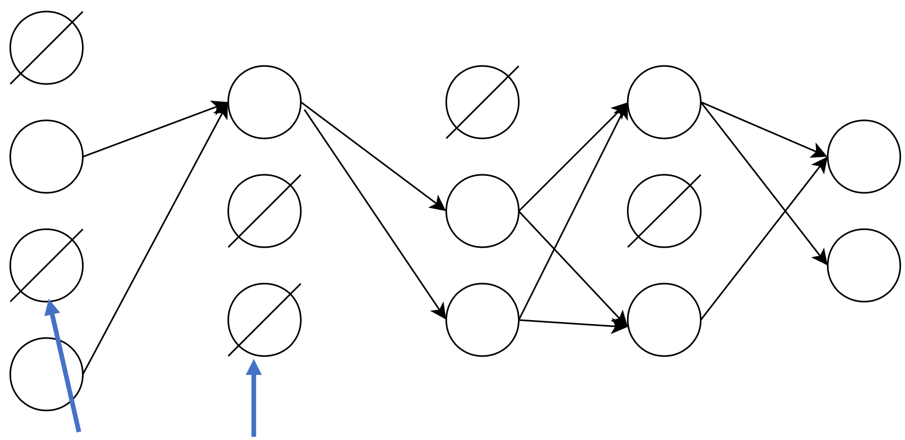
Regularization: Weight Decay

- If a weight is 0, we have a simpler model.
- Initially all weights are close to 0 and they are moved away as learning proceeds.
- Early stopping: Stop before too many weights are updated.
- Weight decay: Add a term that penalizes non-zero weights:

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2 \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda' w_i$$

Regularization: Dropout

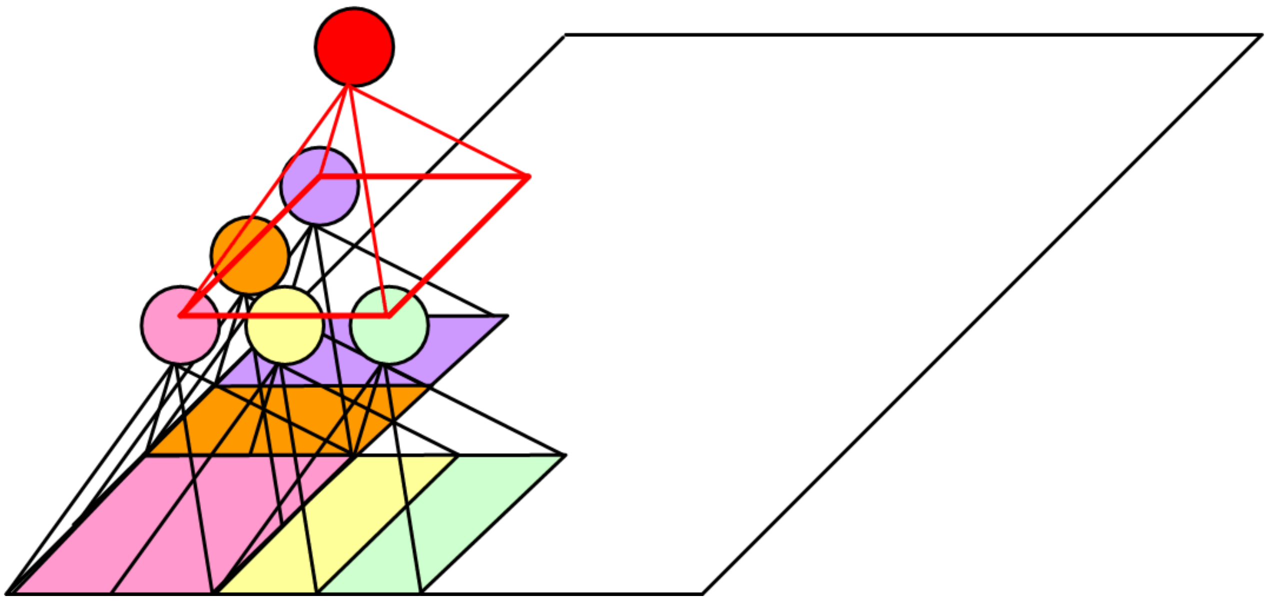
- What if we drop them certain nodes completely?
- Reduce the number of parameters in a ‘random’ fashion to ensure less overfitting and more robust results



Input or hidden unit dropped out (output set to 0) with probability *p* in each minibatch

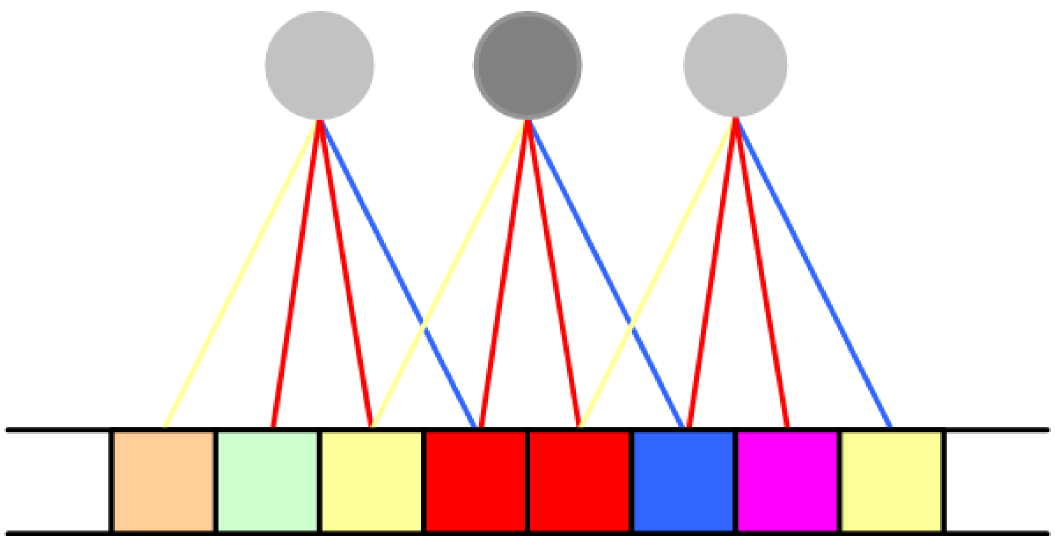
Regularization: Convolutions

- Each unit is connected to a small set of units in the preceding layer. In images, this corresponds to a local patch (LeCun et al 1989).



Regularization: Weight Sharing

The same weights are used in different locations



1d example with 1x4 convolutions and weight sharing

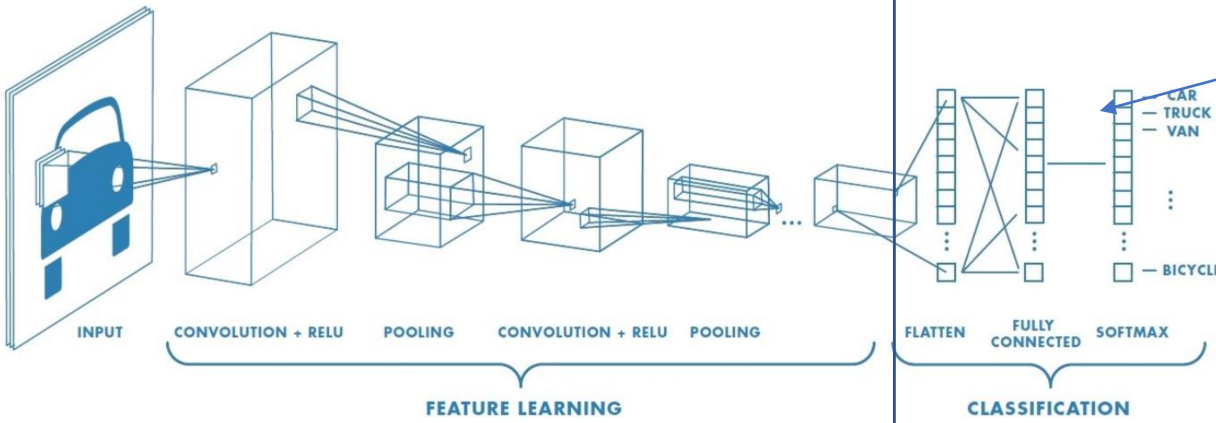
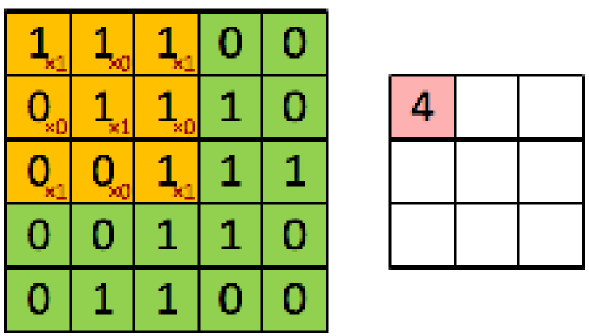


Figure 2 : Neural network with many convolutional layers

Just multilayer perceptron



Image

Convolved Feature

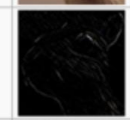
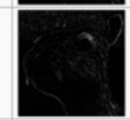
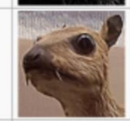
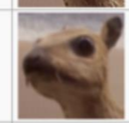
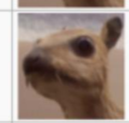
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

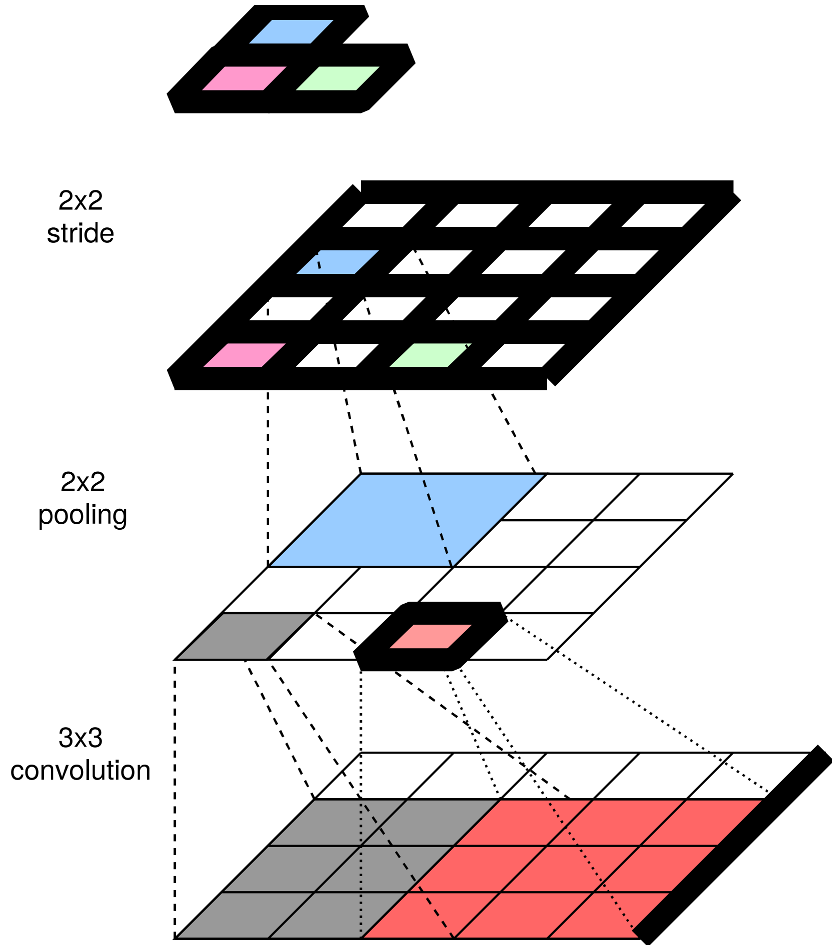
Figure 7 : Some common filters

https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

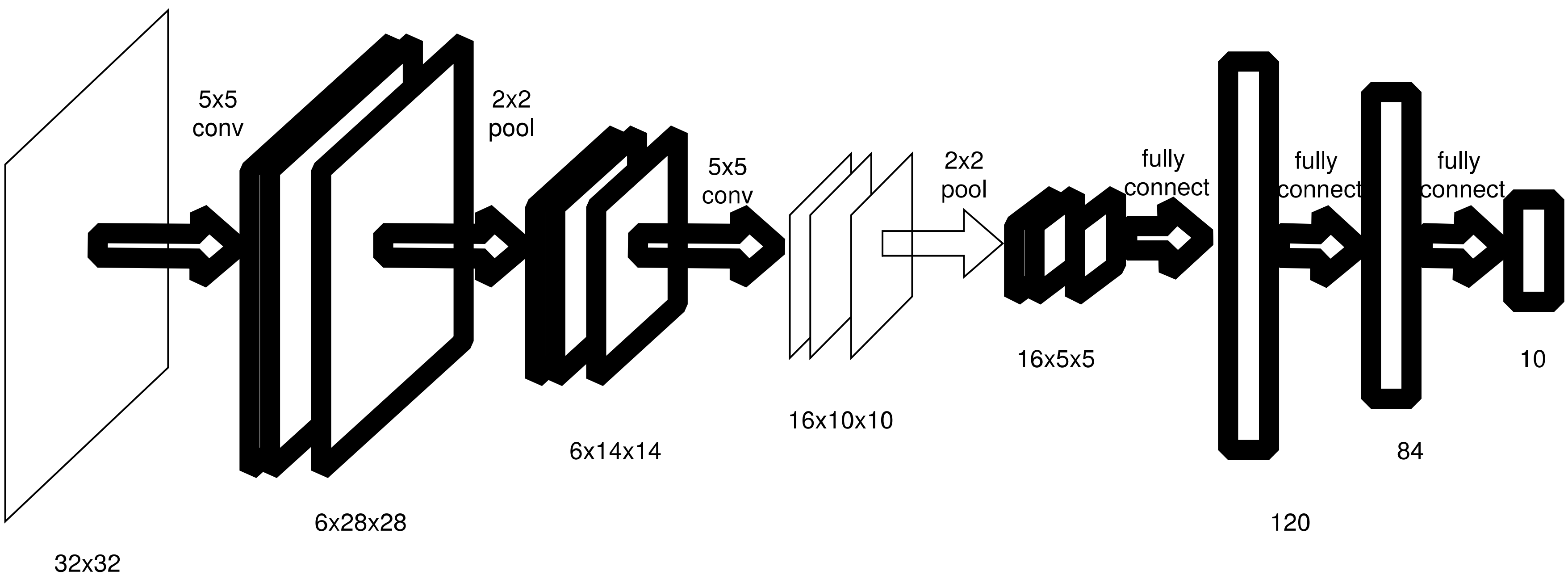
Multiple Convolutional Layers

Move the filter

Max, average, so on



LeNet-5 (LeCun et al 1998)



Has approximately 60K weights and trained on 60K examples (MNIST data set)