Convolutional Neural Networks

David Nilsson

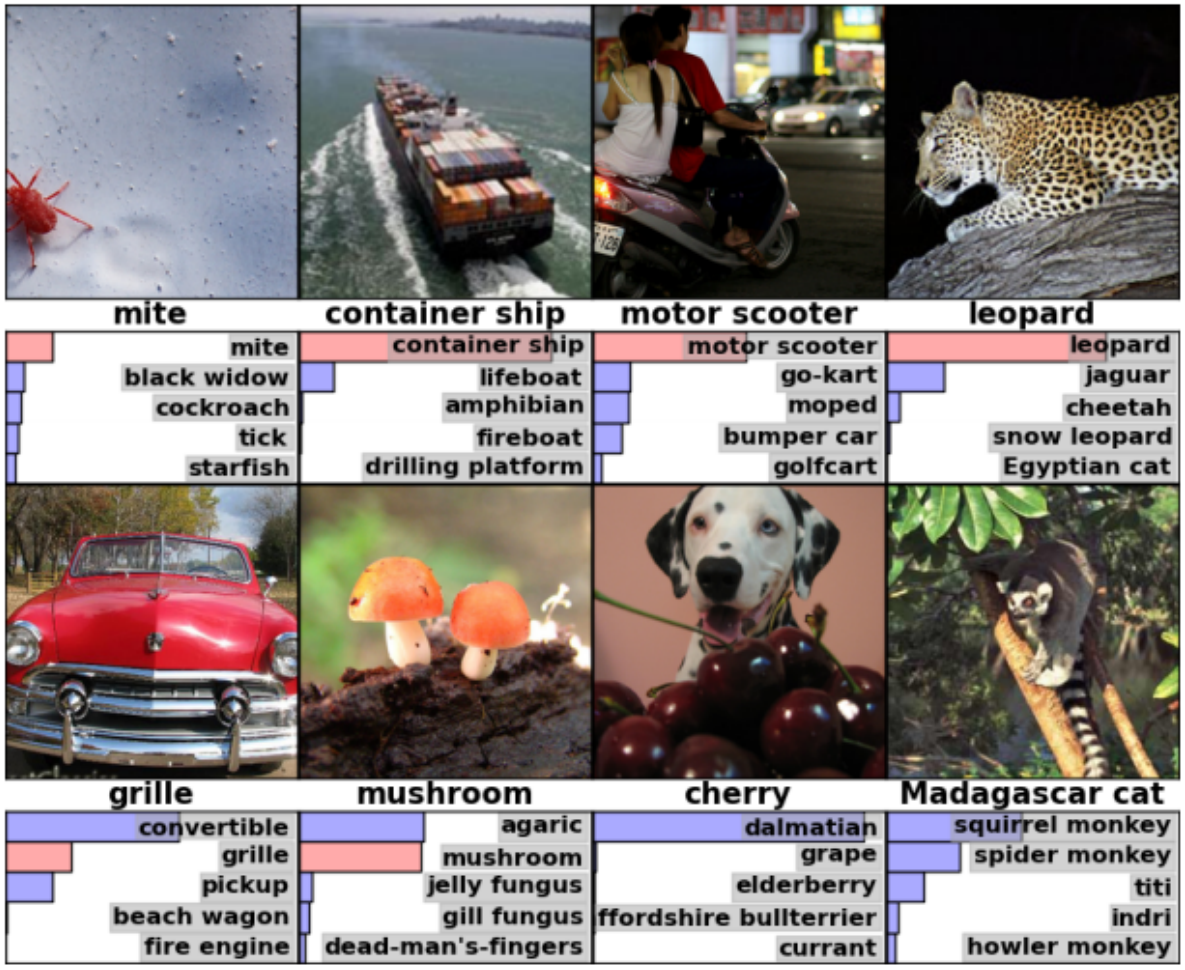
Lund University

November 17, 2016

Contents

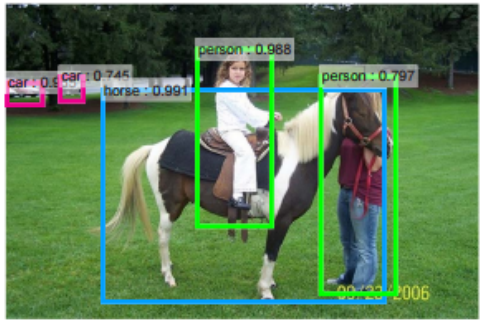
- Applications
- Historical neural net and the brain analogy
- Common layers types
  - Convolution
  - Maxpooling
  - Non-linearity - relu
  - Fully Connected
- Visualize what a neural network learns
- How to train a neural net
- Backpropagation

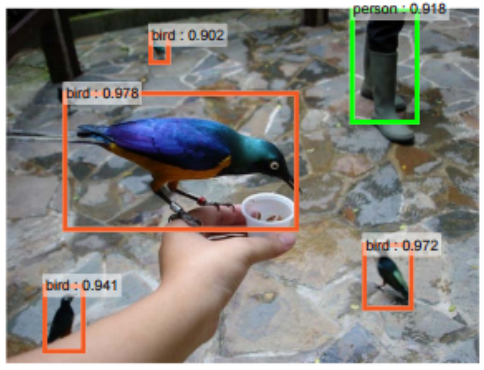
Image Classification

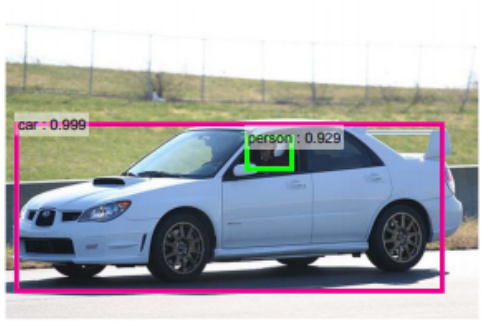


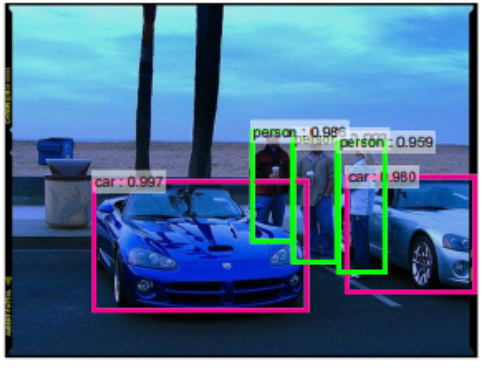
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

Object Detection



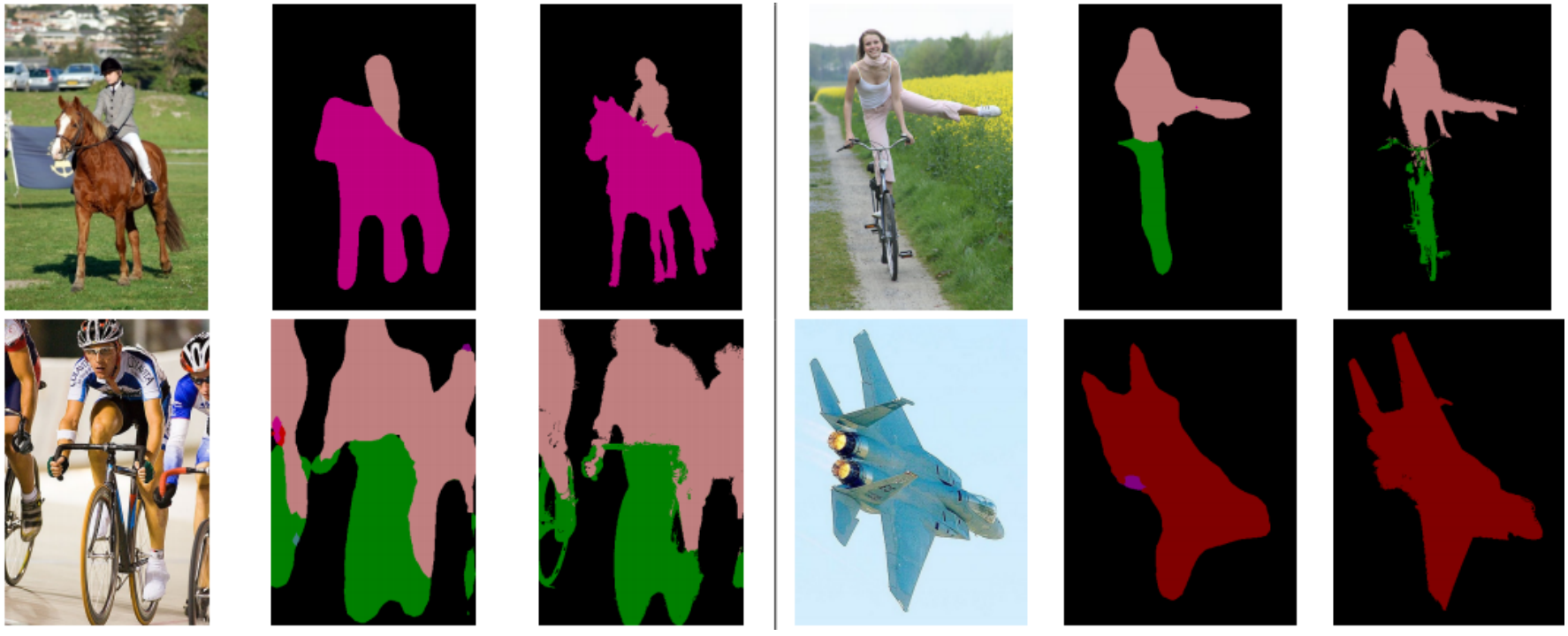






Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

Semantic Segmentation



Chen, Liang-Chieh, et al. "Semantic image segmentation with deep convolutional nets and fully connected crfs." arXiv preprint arXiv:1412.7062 (2014).

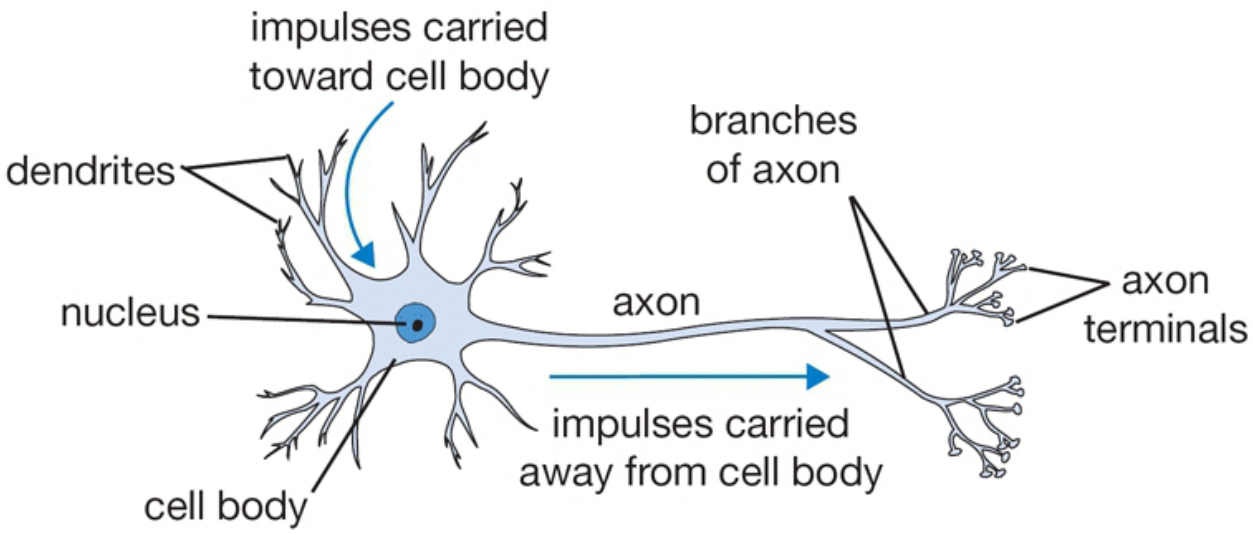
Image Captioning

Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

Playing atari

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

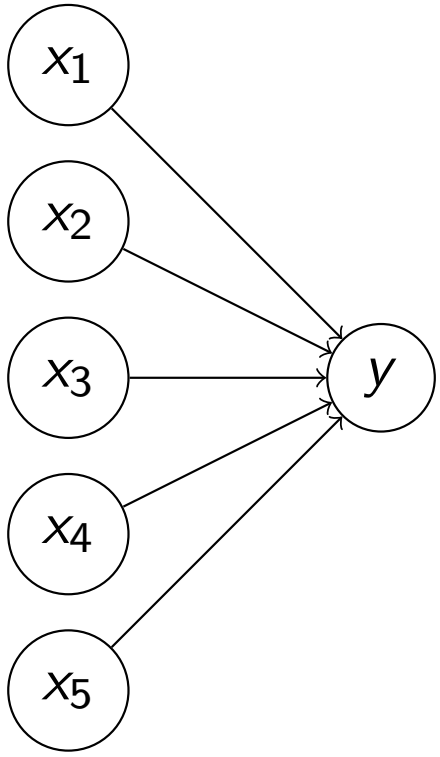
Neural Networks



- A neuron sends a signal that is branched out to other neurons.

http://cs231n.github.io/neural-networks-1/

Neurons

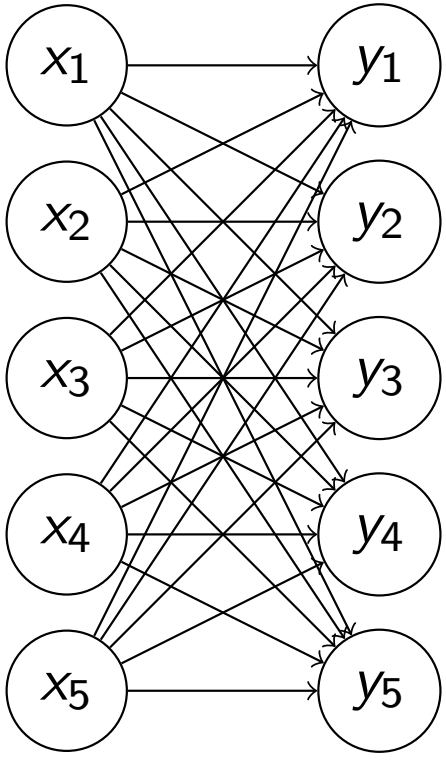


$$y = \sigma\left(\sum_{j=1} w_j x_j + b\right)$$

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

- Each neuron sends an activation x_i that is scaled with a weight w_i.
- The sum is passed through a sigmoid unit.
- The sigmoid function is a smooth approximation of the step function.

Neurons

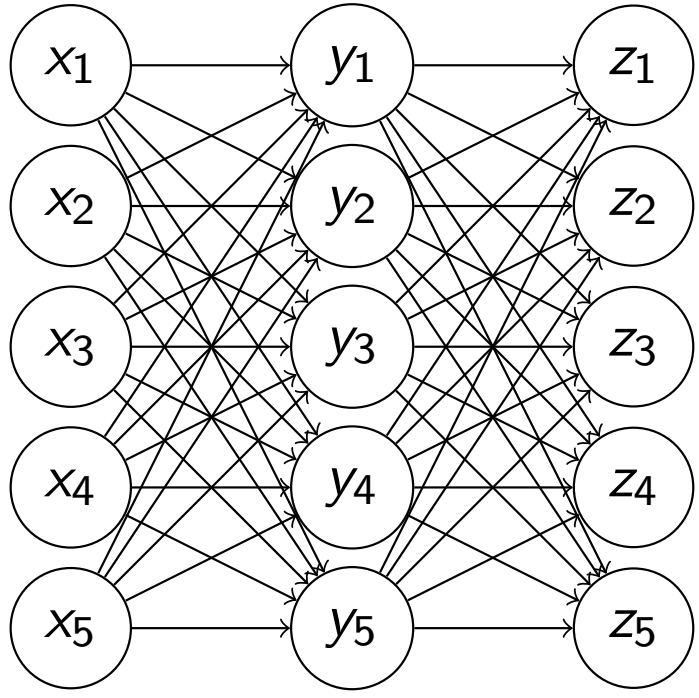


- Two layer neural network

$$y_i = \sigma\left(\sum_{j=1} w_{ij}x_j + b_i\right)$$

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

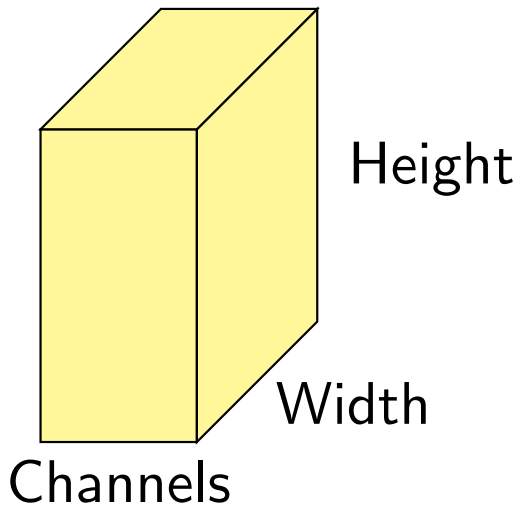
Neural Network



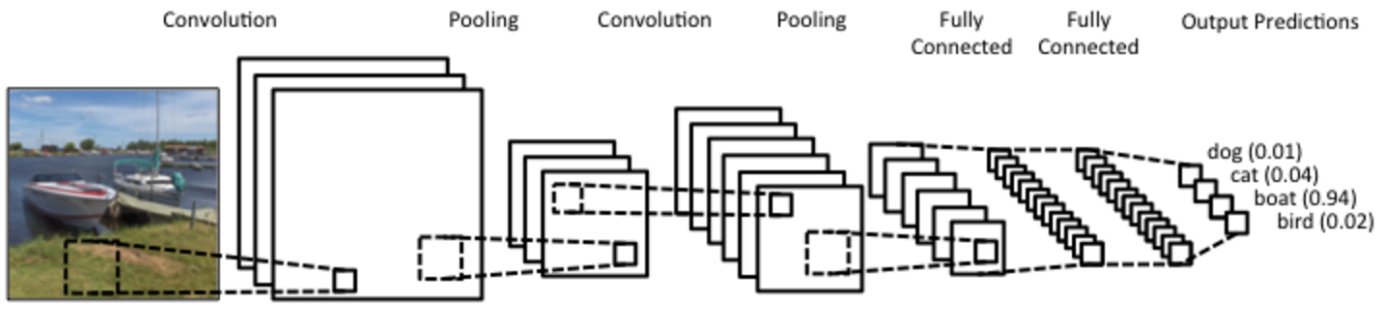
- Stack many neuron after one another in layers.

Convolutional Neural Network (CNN)

- Work with 3-dimensional volumes and mappings between volumes.
- Perfect for image data.

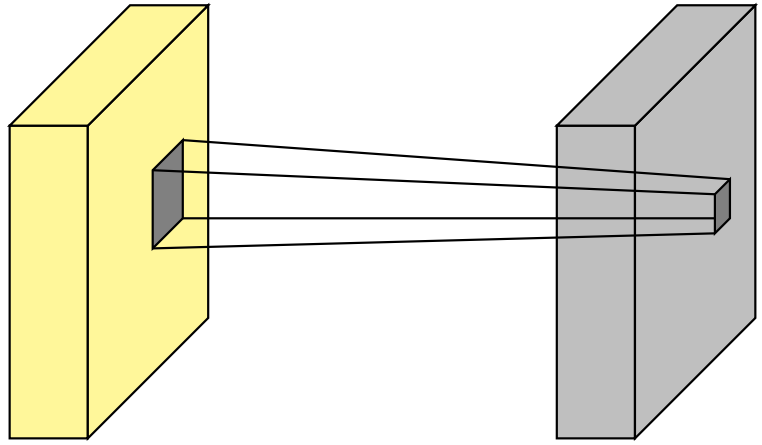


Typical CNN



- We stack many layers after one another.

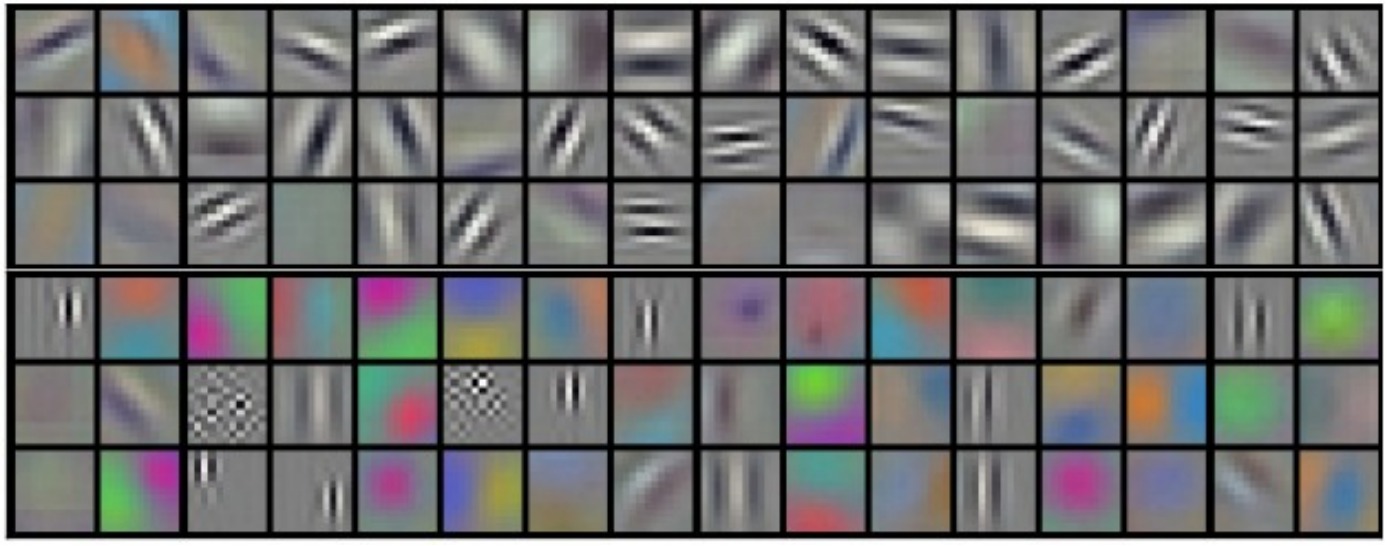
Convolutional Layer



$$y_{ij} = \sum_{i',j',c} w_{i'j'c} x_{i+i',j+j',c} + b$$

- Slide a filter across the input volume and compute dot products.
- The weights of the filter and the bias term are parameters that the neural net has to learn.

Convolutional Layer



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton.
"Imagenet classification with deep convolutional neural networks."
Advances in neural information processing systems. 2012.

Convolutional Layer

$x(:,:,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(:,:,1)$

1	-2
3	1

$y(:,:,1)$

$x(:,:,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

$w(:,:,2)$

1	-2
3	1

- Select a patch in the input and compute the dot product of the patch and the filter.
- Note that we only move the filter spatially. The filter is defined over all channels.

Convolutional Layer

$x(:,:,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(:,:,1)$

1	-2
3	1

$y(:,:,1)$

25		

$x(:,:,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

$w(:,:,2)$

1	-2
3	1

- Select a patch in the input and compute the dot product of the patch and the filter.
- Note that we only move the filter spatially. The filter is defined over all channels.

Convolutional Layer

$x(:,:,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(:,:,1)$

1	-2
3	1

$x(:,:,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

$w(:,:,2)$

1	-2
3	1

$\longrightarrow$

$y(:,:,1)$

25	27	

- Select a patch in the input and compute the dot product of the patch and the filter.
- Note that we only move the filter spatially. The filter is defined over all channels.

Convolutional Layer

$x(:,:,1)$

1	-2	0	5
7	0	1	-3
2	-2	2	-2
-5	-4	3	1

$w(:,:,1)$

1	-2
3	1

$\longrightarrow$

$y(:,:,1)$

25	27	

$x(:,:,2)$

0	-1	2	0
-4	9	6	5
3	2	1	0
0	1	3	-3

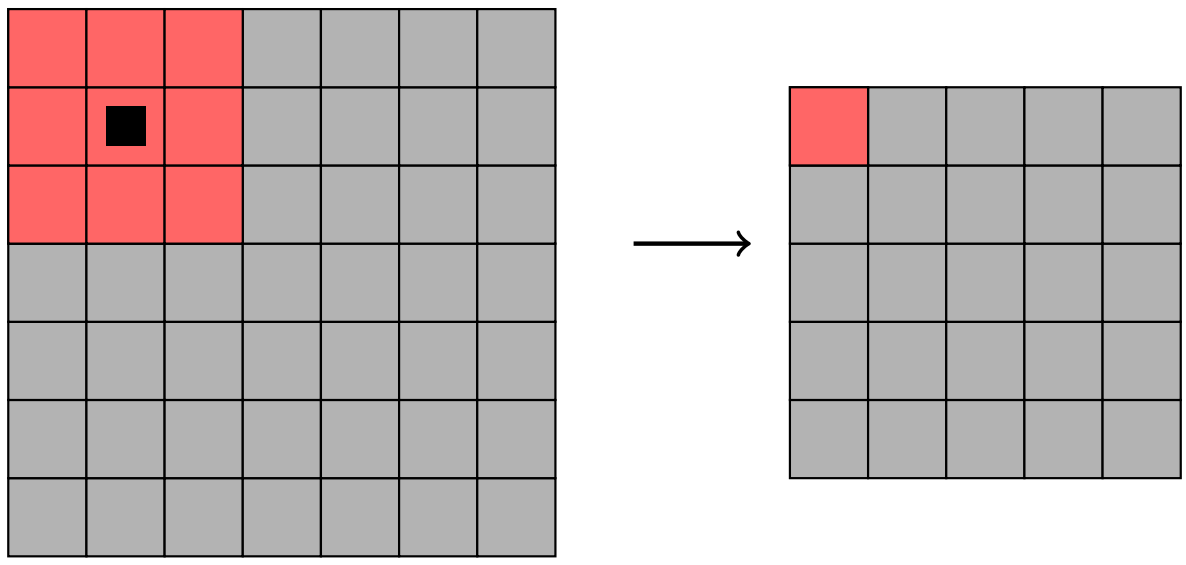
$w(:,:,2)$

1	-2
3	1

- The filter must have the same number of channels as the input (2 in this example).
- Each filter produce one output channel.
- $size(w) = [f_h, f_w, c, n]$ where n is the number of filters or output channels. We also have n bias elements.

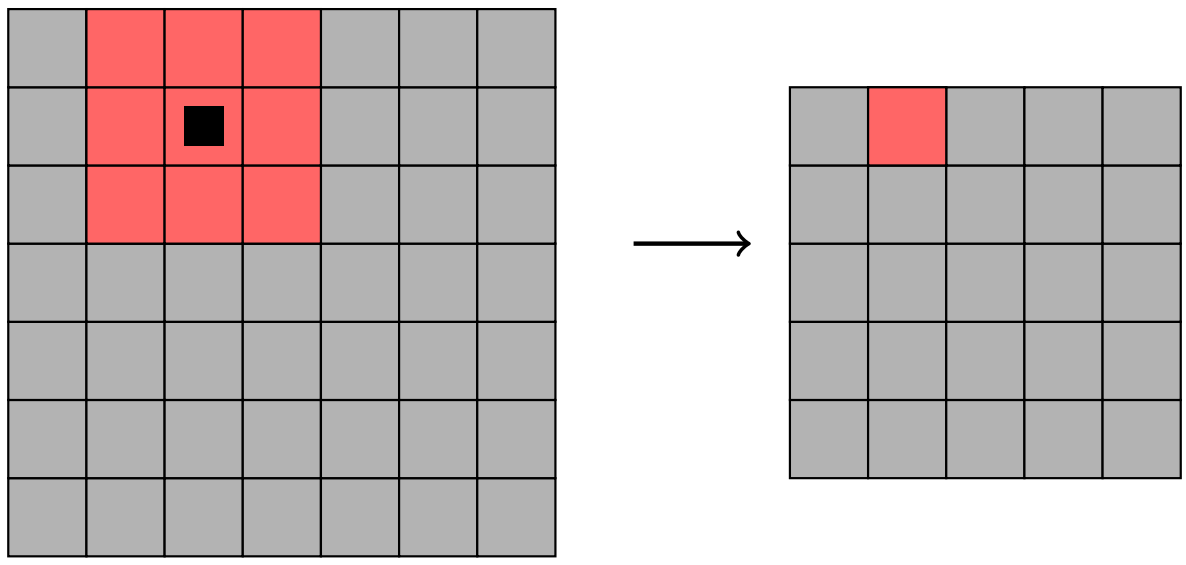
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



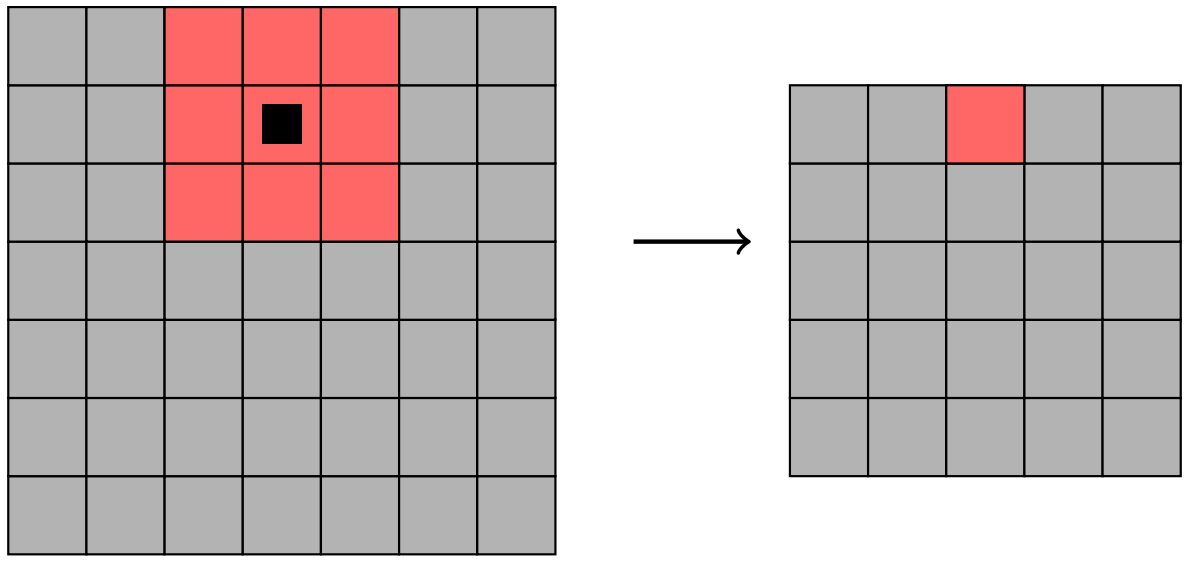
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



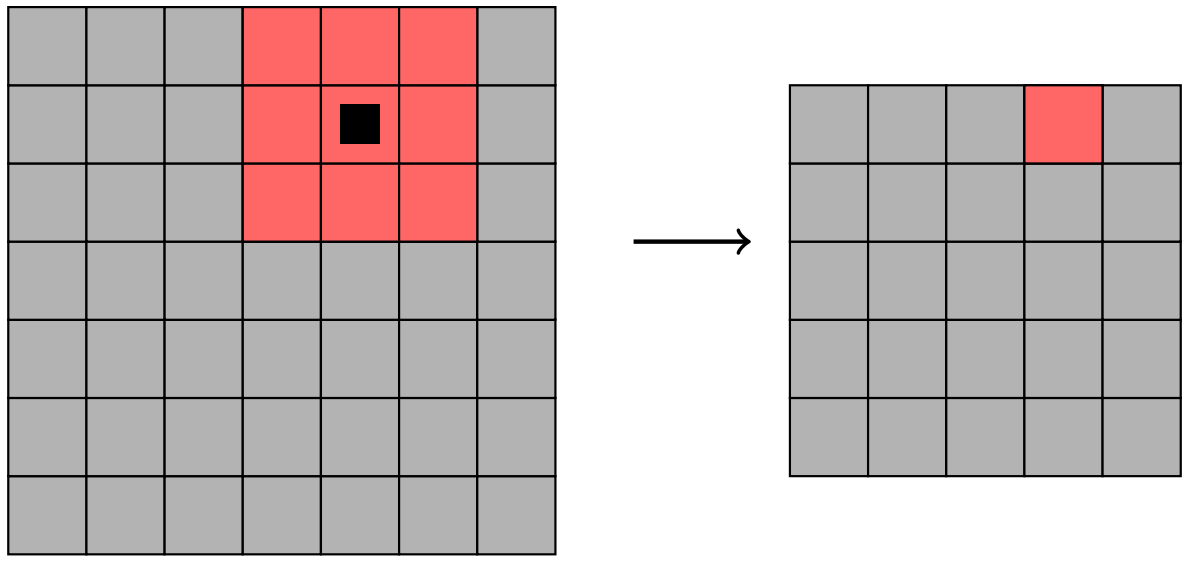
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



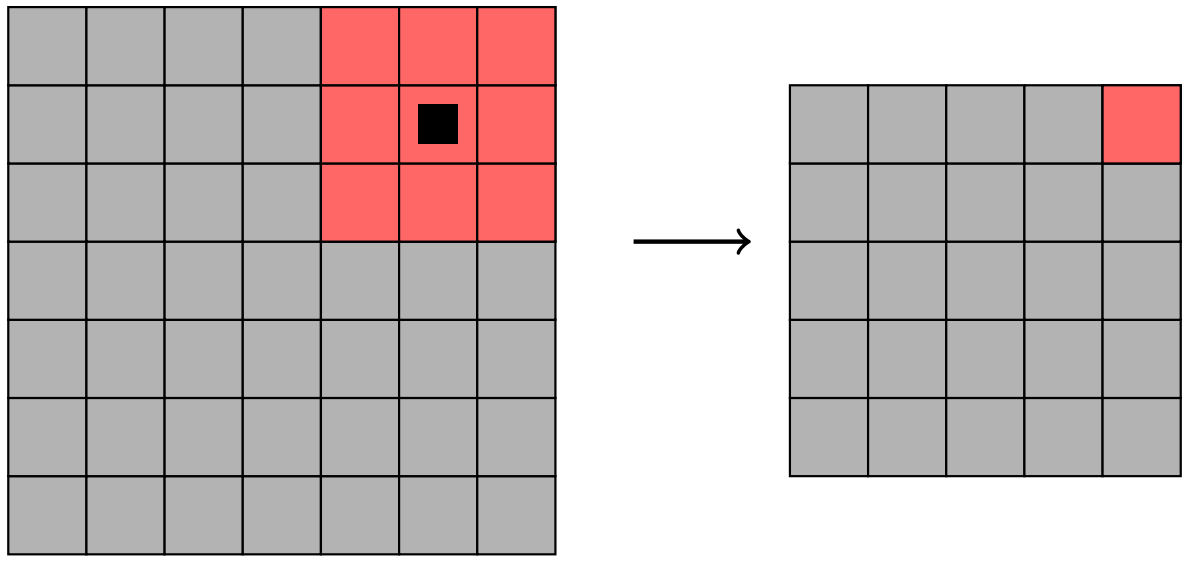
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



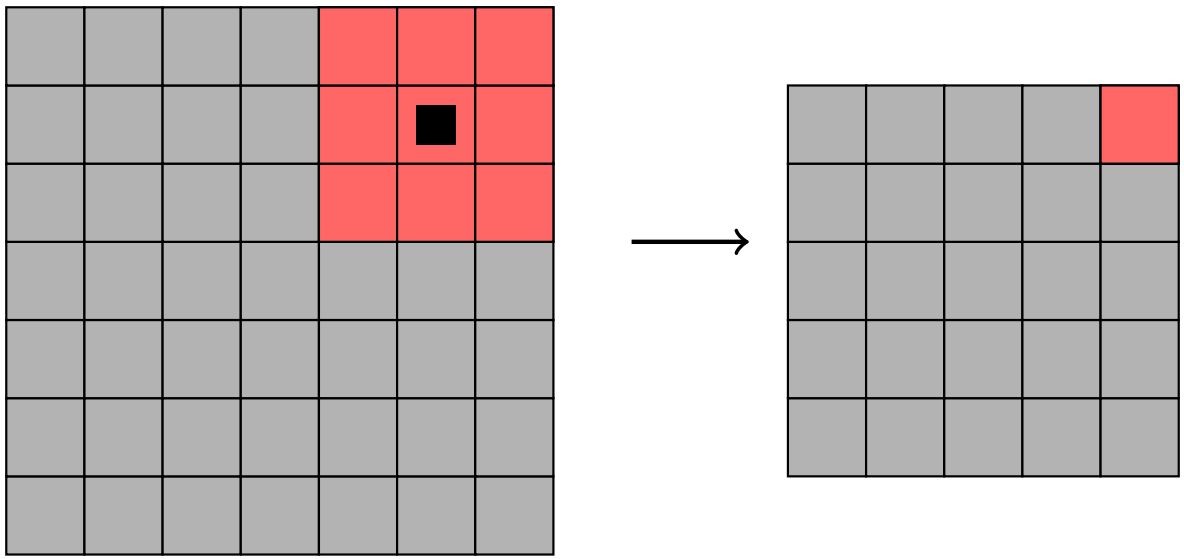
Convolutional Layer

- We convolve a 7x7 image with a 3x3 filter. What is the output size?



Convolutional Layer

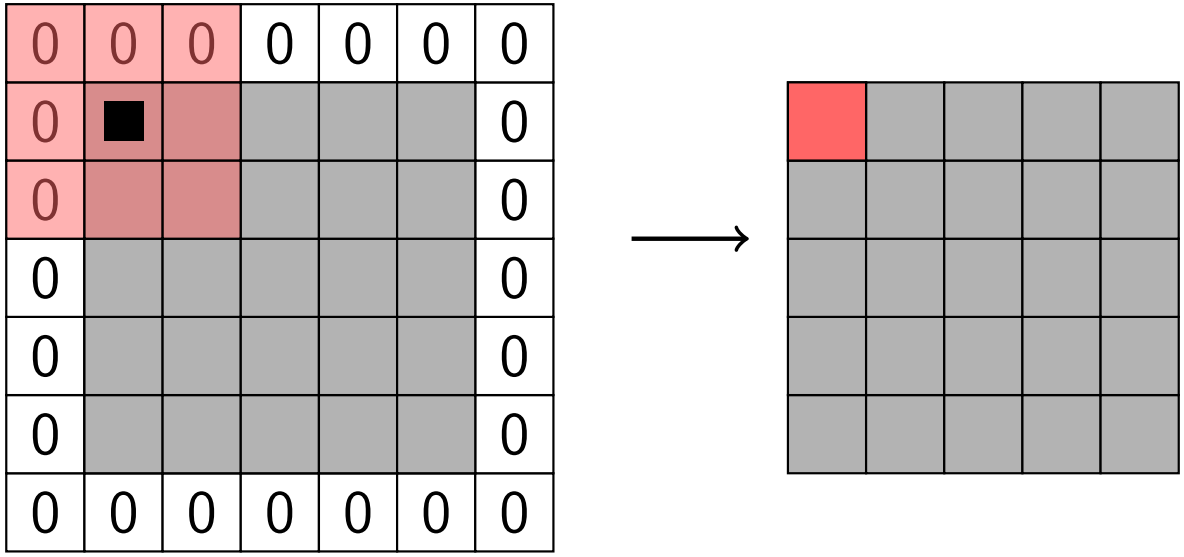
- We convolve a 7x7 image with a 3x3 filter. What is the output size?



In general: Convolving a $H \times W$ map with a $f_H \times f_W$ filter will result in size $(H - f_H + 1) \times (W - f_W + 1)$.

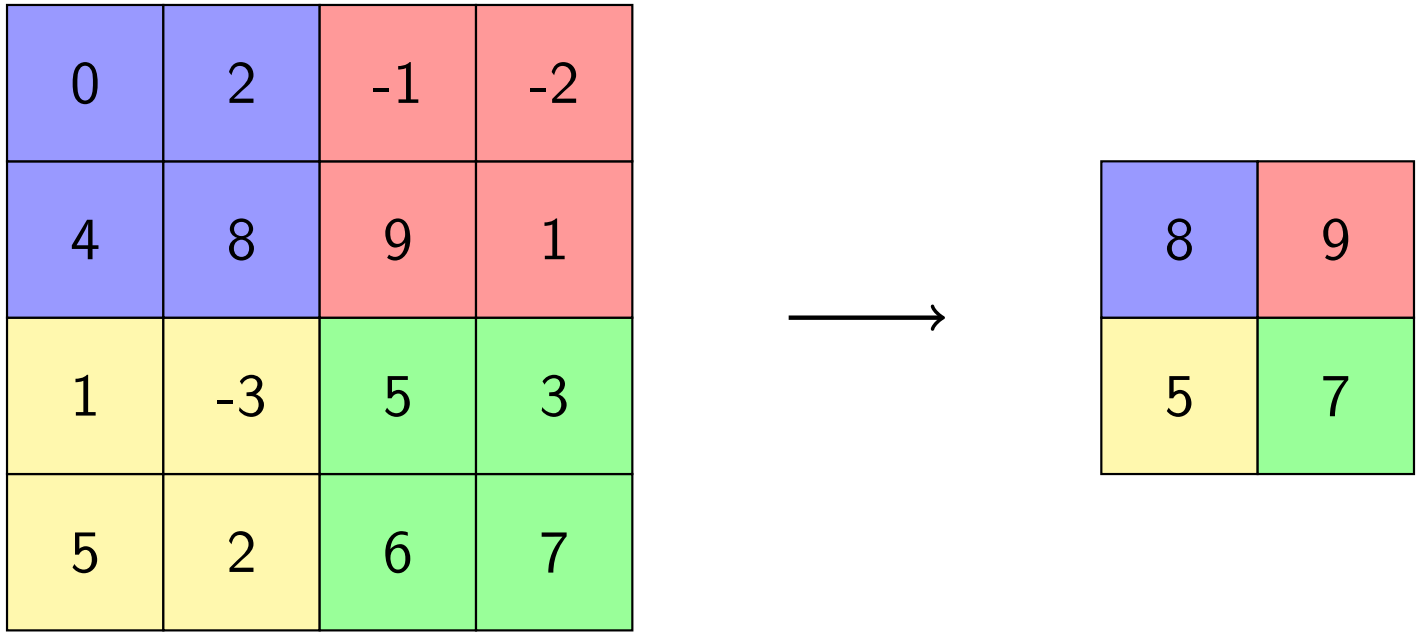
Convolutional layer

- What if we want the same output and input sizes?
- Pad with zeros!



- The result is also 5×5.
- Pad with $(f-1)/2$ if you want to retain the size.

Maxpooling Layer

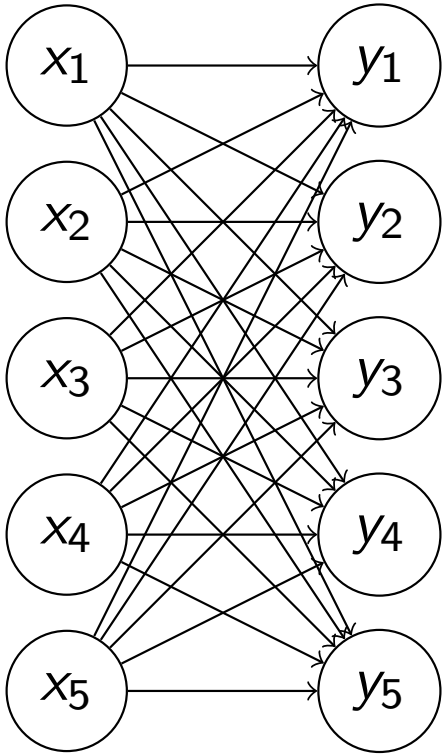


- 2x2 maxpooling

$$y_{ijc} = \max\{x_{2i,2j,c}, x_{2i,2j+1,c}, x_{2i+1,2j,c}, x_{2i+1,2j+1,c}\}$$

- Subsample and retain the strongest activations.
- Computed for each channel independently.

Fully Connected Layer



$$y_i = \sum_{j=1} a_{ij} x_j + b_i$$

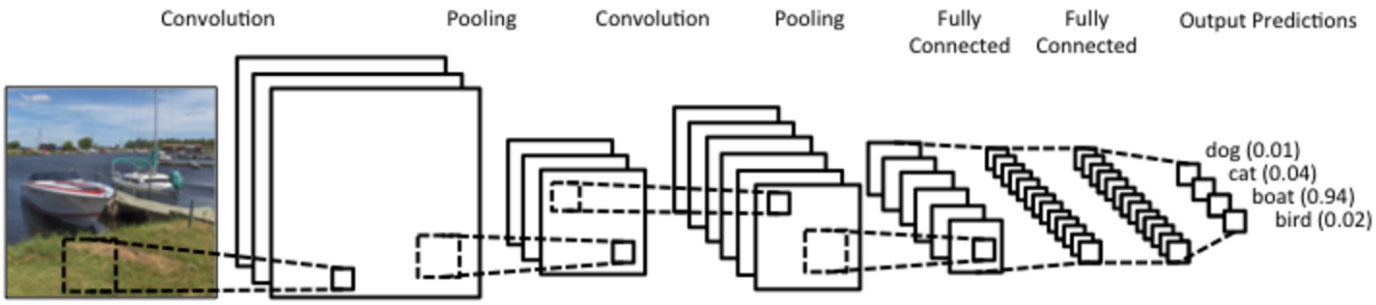
$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

- All input nodes are connected to all output nodes.
- Parameters **A** and **b** should be learned.

Non-linearities

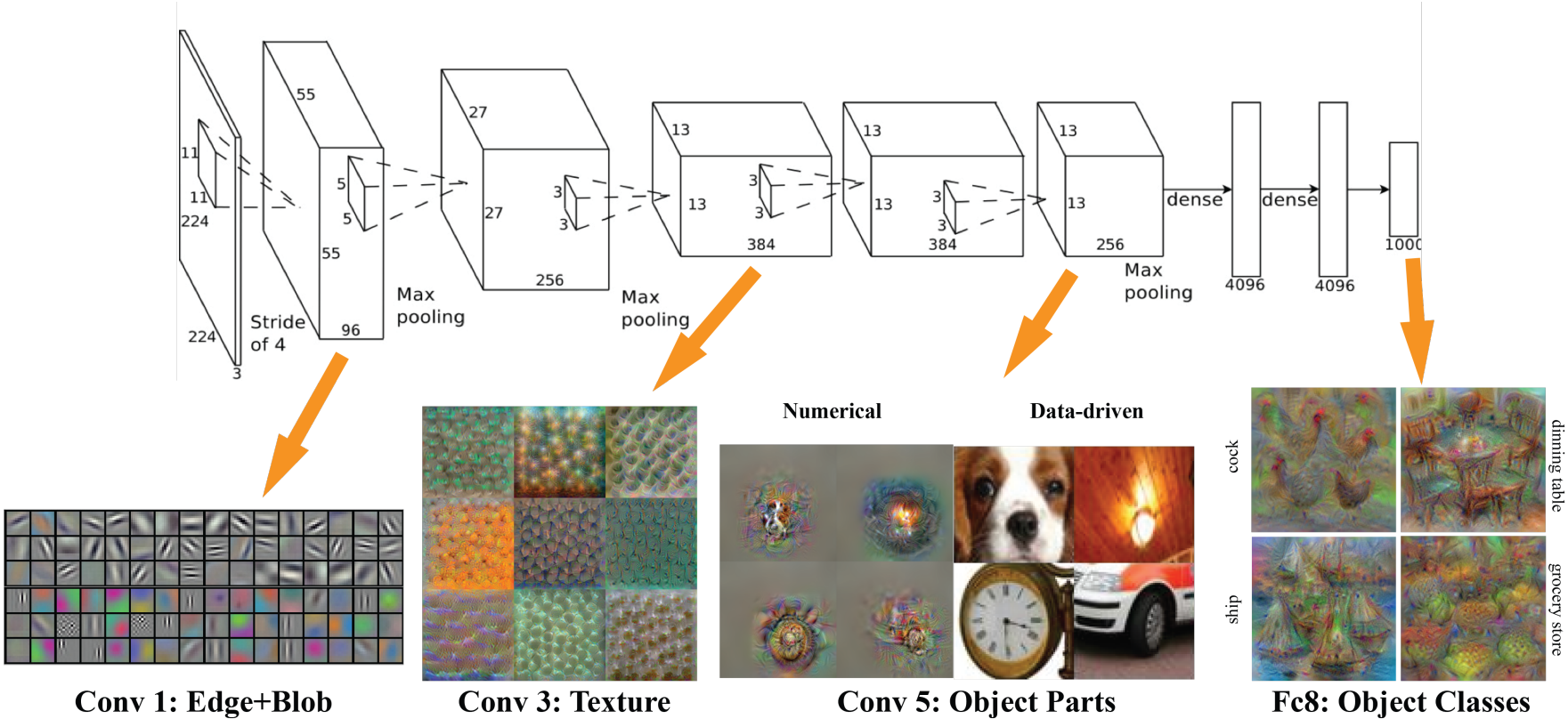
- The network should be highly non-linear as a function from input to output.
- Use layers that apply some non-linear function element-wise.
- Relu: $y = max(0, x)$
- Sigmoid: $y = \sigma(x) = \frac{1}{1+e^{-x}}$, range $[0, 1]$
- Tanh: $y = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, range $[-1, 1]$
- Use relu in the project!
- For deep nets, use relu. For recurrent nets, use a clever combination of sigmoids and tanhs. Popular recurrent architectures are LSTMs and GRUs. They use sigmoids to get adaptive gating of features.

Typical CNN

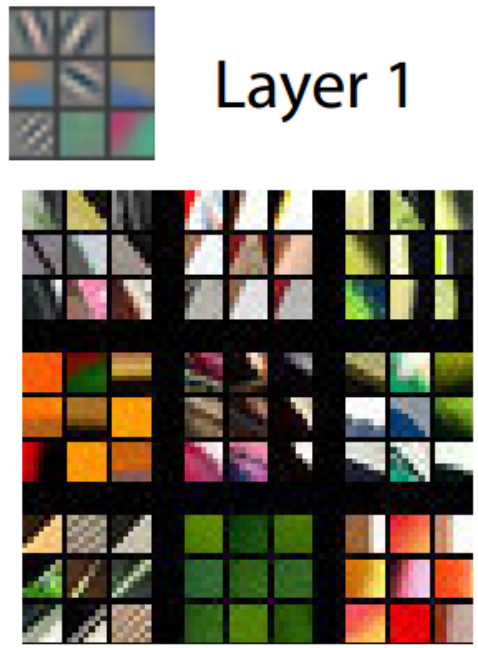


- Use the layers as building blocks in a large network.
- Convolution - relu - max-pooling
- Often a few conv-relu pairs between the max-pooling layers.
- One or more fully connected layers at the very end.

Hierarchical Features

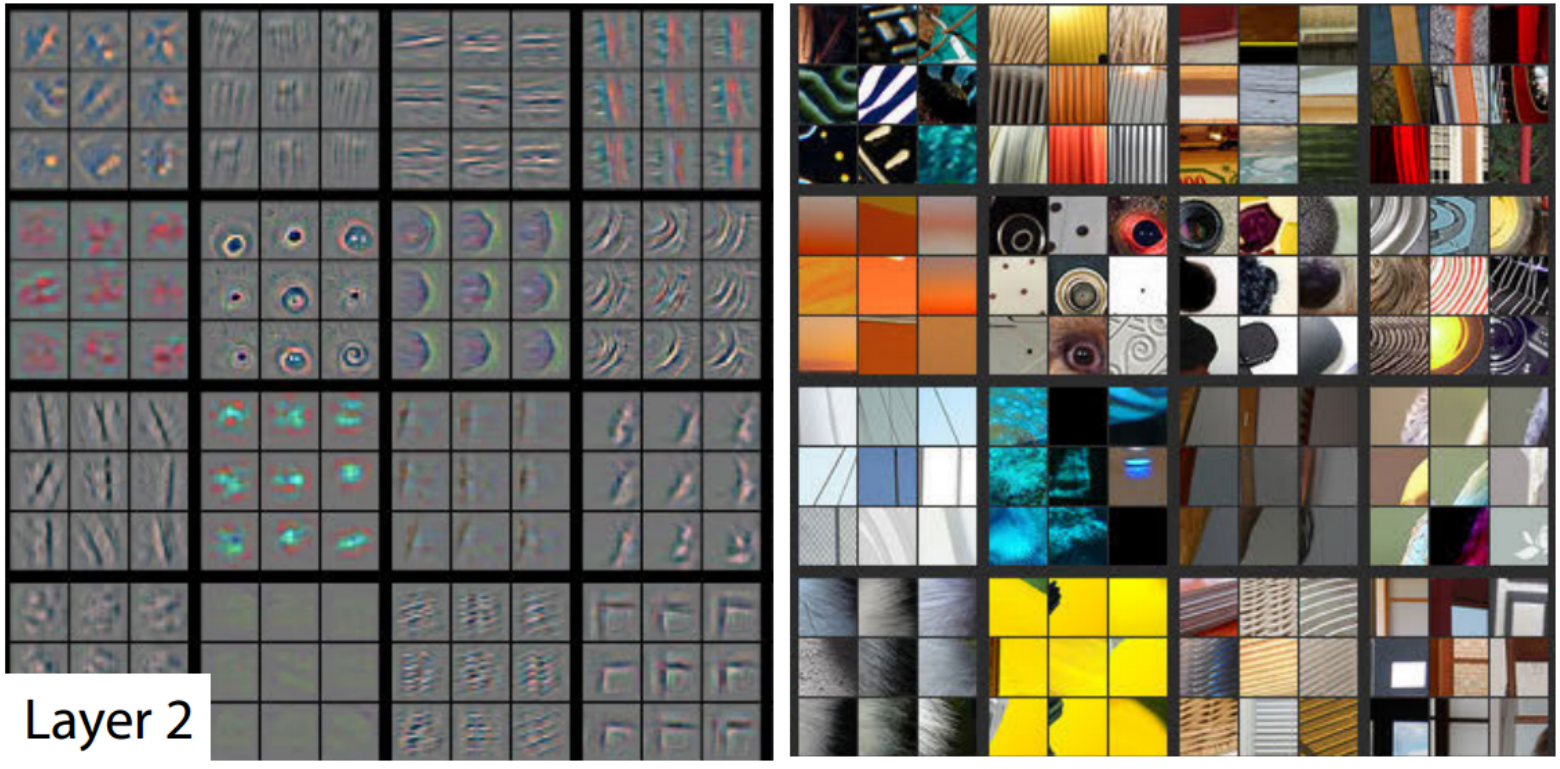


Hierarchical Features



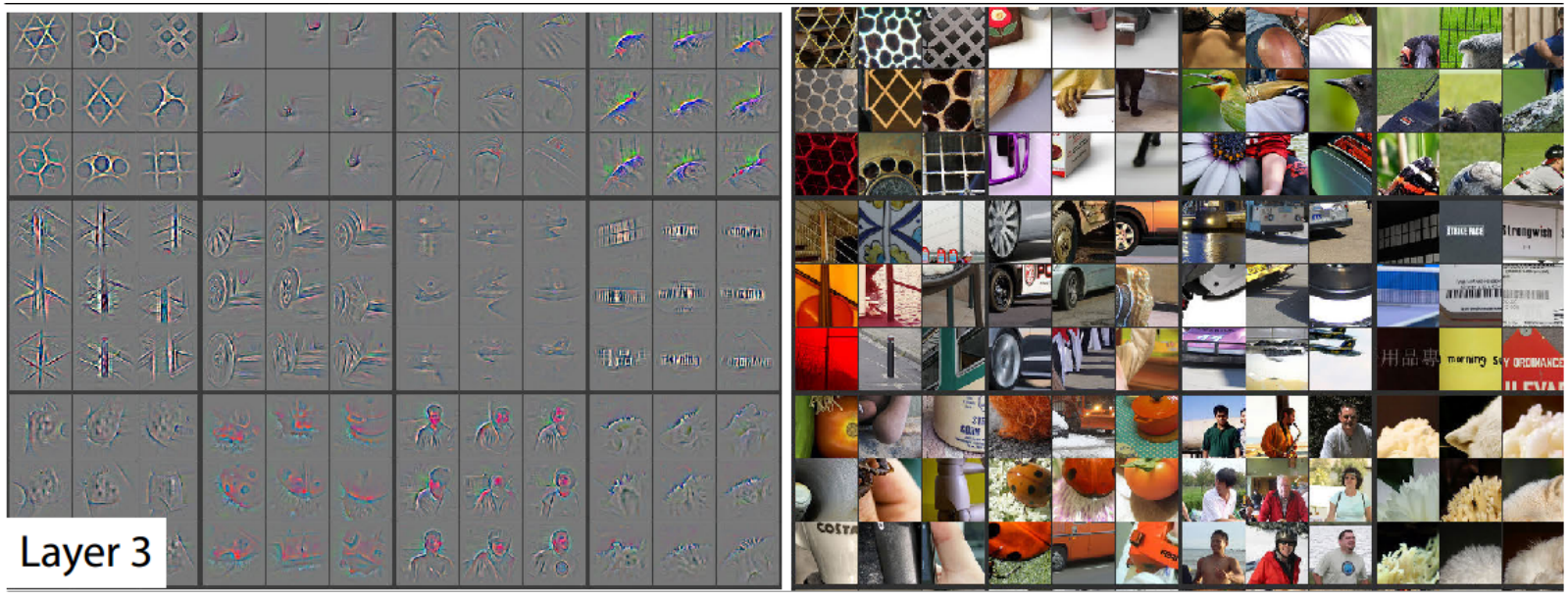
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



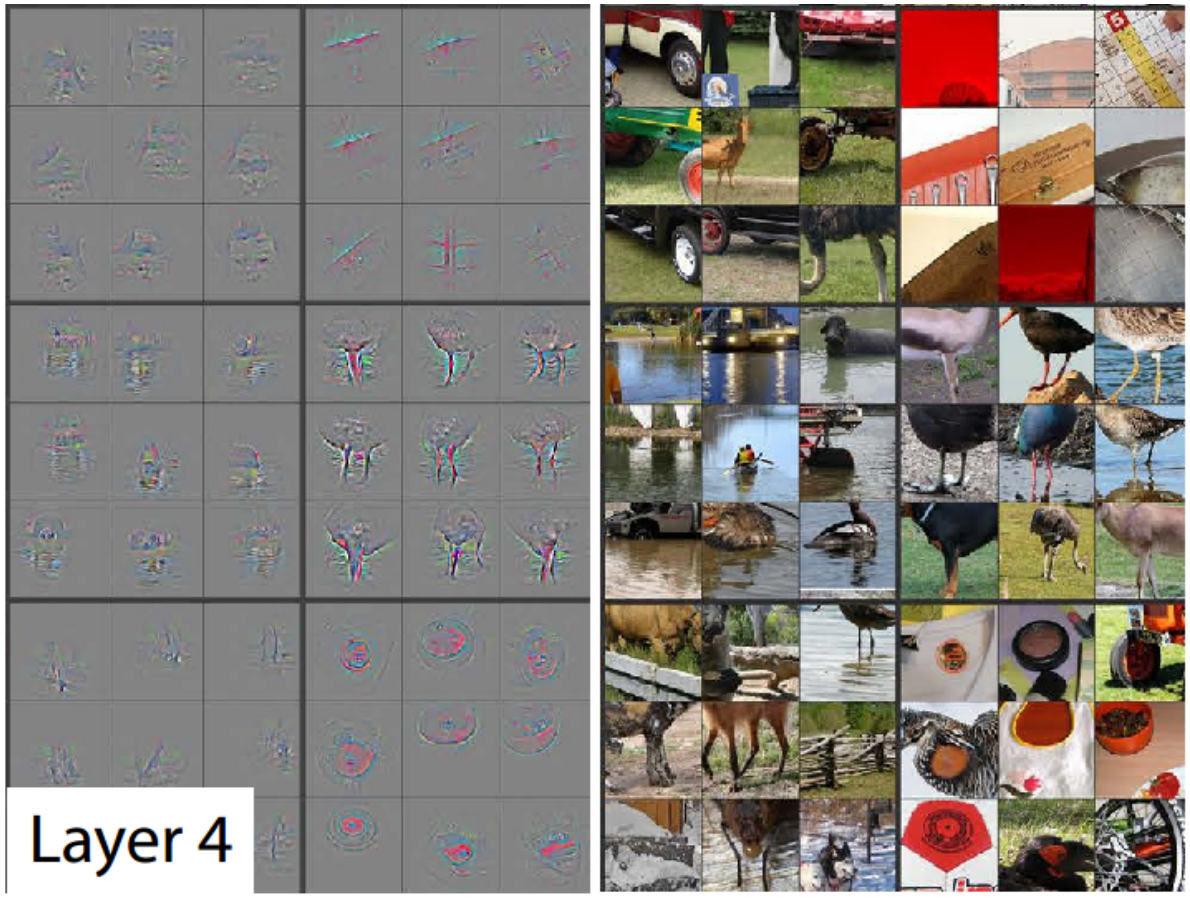
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



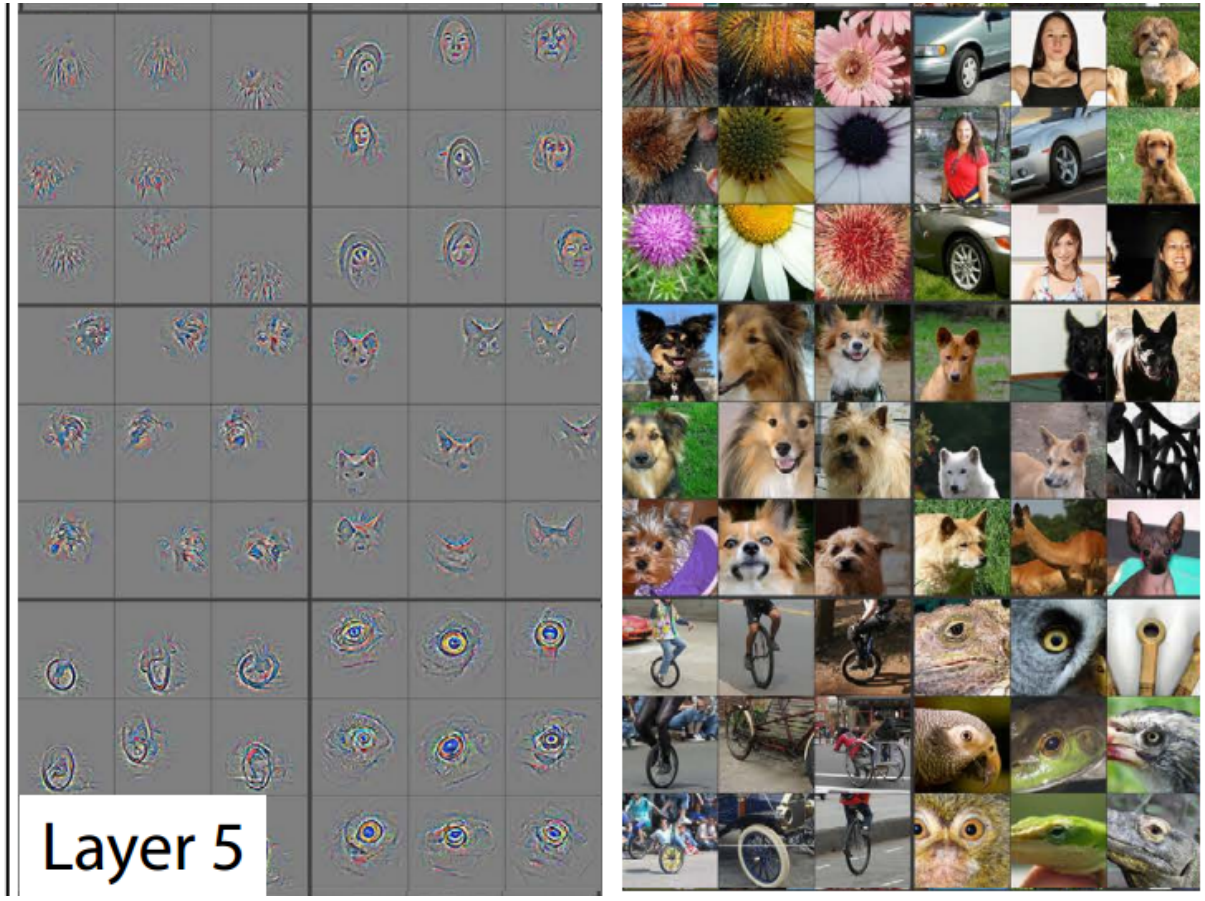
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Hierarchical Features



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European Conference on Computer Vision. Springer International Publishing, 2014.

Training a Neural Net

- We define a loss on top of the final layer.
- We wish to find good parameters $\boldsymbol{\theta}$.
- Given training data $(\mathbf{x}_i, y_i)_{i=1}^N$. We wish to minimize the loss

$$L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

- We compute the gradient $\frac{\partial L}{\partial \boldsymbol{\theta}}$ using backpropagation.
- The parameters are updated using gradient descent or some other first order optimization method.

Training a Neural Net

- In practice, N is very large and computing the gradient for all examples requires a lot of computations.
- Instead, select a random small subset with n elements of the training data and compute the gradient with respect to

$$L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})$$

- Select a new random subset, called a batch, every time.
- For the MNIST dataset in the project, $n = 4$ and $N = 50000$.
- We can update the parameters much faster.
- The loss function is highly non-linear and non-convex, so computing the exact gradient might not be desirable.

Training a Neural Net

- Gradient descent, take a step in the negative direction of the gradient. The learning rate α controls how long steps we take.

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha \frac{\partial L}{\partial \boldsymbol{\theta}}$$

- Using gradient descent with small batches is usually called stochastic gradient descent (SGD).
- If we use a small batch the gradient might be noisy. Use a moving average of the gradient estimation

$$\mathbf{m}_n = \mu \mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \boldsymbol{\theta}}$$

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha \mathbf{m}_n$$

- We call μ the momentum parameter. Typical values are 0.9 or 0.99.

Training a Neural Net

- In practice we often use a regularizer for the parameters, so the function to minimize is

$$\frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$$

- where λ is the weight decay parameter. By using that $\frac{\partial}{\partial \boldsymbol{\theta}}\left(\frac{\lambda}{2}\|\boldsymbol{\theta}\|^2\right) = \lambda\boldsymbol{\theta}$, gradient descent is changed to

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha\left(\frac{\partial L}{\partial \boldsymbol{\theta}} + \lambda\boldsymbol{\theta}_n\right)$$

- and gradient descent with momentum is changed to

$$\mathbf{m}_n = \mu\mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \boldsymbol{\theta}}$$
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha(\mathbf{m}_n + \lambda\boldsymbol{\theta}_n)$$

Losses

- The loss should model whatever we want the neural network to do.
- Image classification - the loss should be low if the networks prediction matches the correct label and high otherwise.
- Minimizing such a function will direct the network to its intended functionality.
- Suppose we have a final layer with computed values $\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$ being scores for the different classes. The correct label is c.
- Inspired by statistical methods, we will turn it into probabilities and minimize a negative log likelihood.

Losses

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- y_i models the probability of class i
- This is called a softmax transformation. If the x-values are not too close $y_i \approx 1$ for the largest x_i and $y_i \approx 0$ for the other.
- Note that $0 < y_i < 1$ and $\sum_{i=1}^{n} y_i = 1$, so it models probabilities.
- Negative log likelihood of class c

$$-\log(y_c) = -\log\left(\frac{e^{x_c}}{\sum_{j=1}^{n} e^{x_j}}\right) = -x_c + \log\left(\sum_{j=1}^{n} e^{x_j}\right)$$

- A low value corresponds to a good and confident prediction.

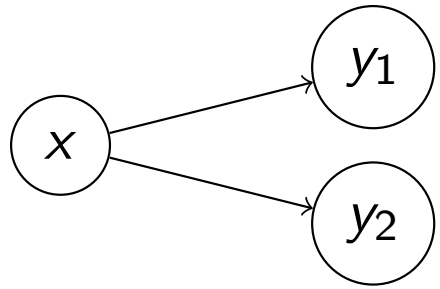
Backpropagation

- How can we compute the gradient with respect to the parameters?
- For given node x, look at all outgoing connections to nodes y_i in the next layer. Using the chain rule from calculus in several variables, we have

$$\frac{\partial L}{\partial x} = \sum_i \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial x}$$

- For each node y_i the contribution to the gradient at x is the gradient at y_i, $\frac{\partial L}{\partial y_i}$ but we scale it with the local partial derivative $\frac{\partial y_i}{\partial x}$.

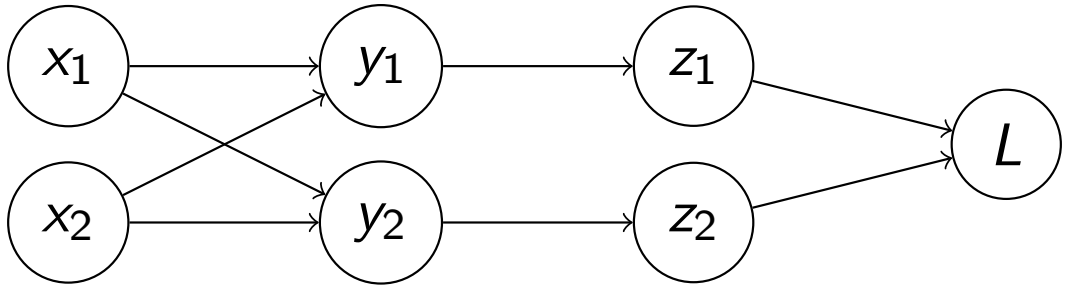
Backpropagation



- Increasing x by Δx cause y_1 to increase by $\frac{\partial y_1}{\partial x}\Delta x$ and in turn L to increase $\frac{\partial L}{\partial y_1}\Delta y_1 = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x}\Delta x$.
- Similarly, y_2 increase by $\frac{\partial y_2}{\partial x}\Delta x$ and in turn L to increase $\frac{\partial L}{\partial y_2}\Delta y_2 = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x}\Delta x$.
- In total, L has increased $\frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x}\Delta x + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x}\Delta x$, so

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x}$$

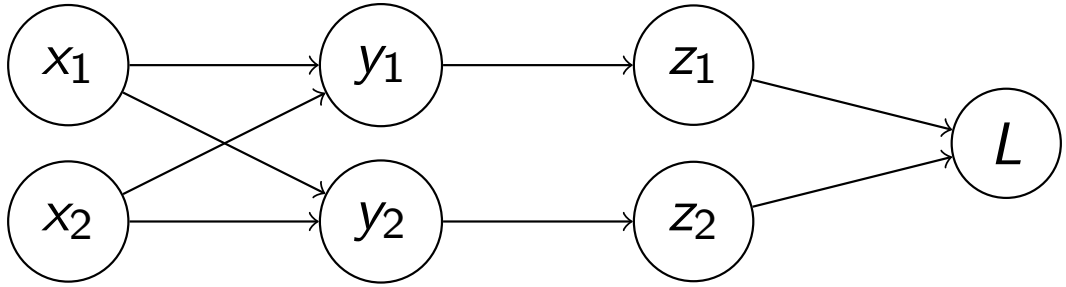
Backpropagation



$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \left\{ L = \frac{1}{2}(z_1 - z_2)^2 \right.$$

- We want to compute $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$.

Backpropagation

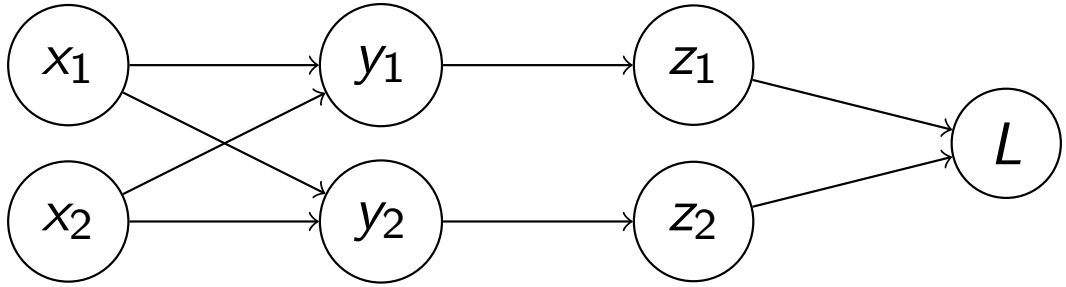


$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \begin{cases} L = \frac{1}{2}(z_1 - z_2)^2 \end{cases}$$

- We want to compute $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$.
- Naive solution: Differentiate

$$L = \frac{1}{2}(z_1 - z_2)^2 = \frac{1}{2}(y_1^2 - e^{y_2})^2 = \frac{1}{2}((2x_1 - x_2)^2 - e^{-x_1+x_2})^2$$

Backpropagation



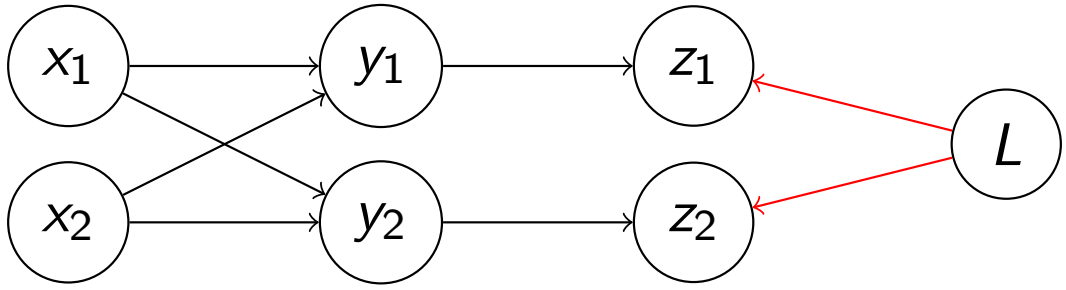
$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \left\{ L = \frac{1}{2}(z_1 - z_2)^2 \right.$$

- We want to compute $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$.
- Naive solution: Differentiate

$$L = \frac{1}{2}(z_1 - z_2)^2 = \frac{1}{2}(y_1^2 - e^{y_2})^2 = \frac{1}{2}((2x_1 - x_2)^2 - e^{-x_1+x_2})^2$$

- This will not scale if we add more layers.

Backpropagation

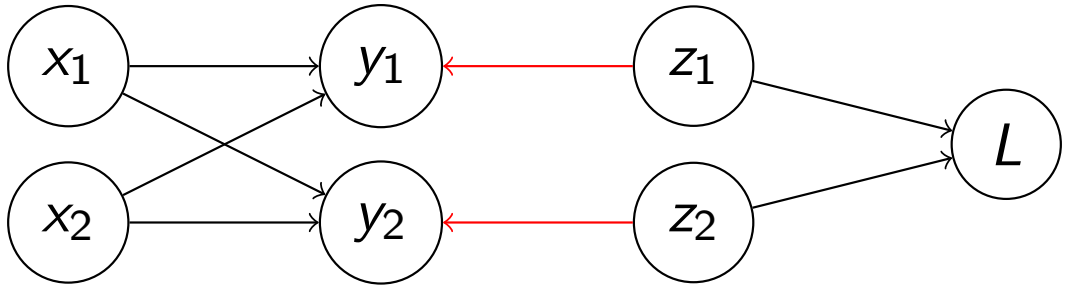


$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \left\{ L = \frac{1}{2}(z_1 - z_2)^2 \right.$$

$$\frac{\partial L}{\partial z_1} = \frac{\partial}{\partial z_1}\left(\frac{1}{2}(z_1 - z_2)^2\right) = z_1 - z_2$$

$$\frac{\partial L}{\partial z_2} = \frac{\partial}{\partial z_2}\left(\frac{1}{2}(z_1 - z_2)^2\right) = -(z_1 - z_2)$$

Backpropagation

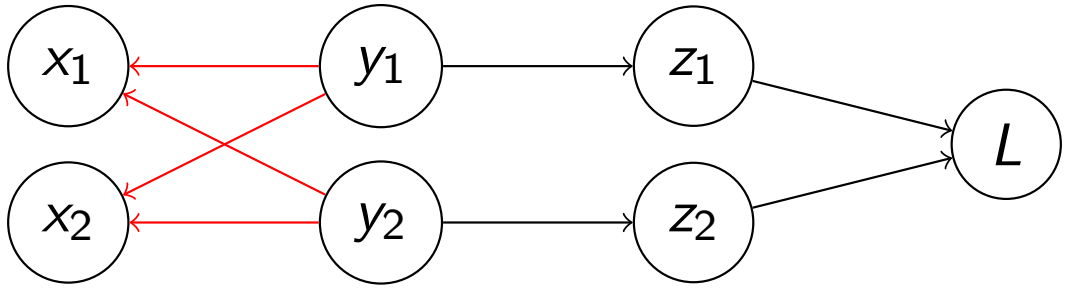


$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \begin{cases} L = \frac{1}{2}(z_1 - z_2)^2 \end{cases}$$

$$\frac{\partial L}{\partial y_1} = \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial y_1} = 2y_1\frac{\partial L}{\partial z_1}$$

$$\frac{\partial L}{\partial y_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial y_2} = e^{y_2}\frac{\partial L}{\partial z_2}$$

Backpropagation

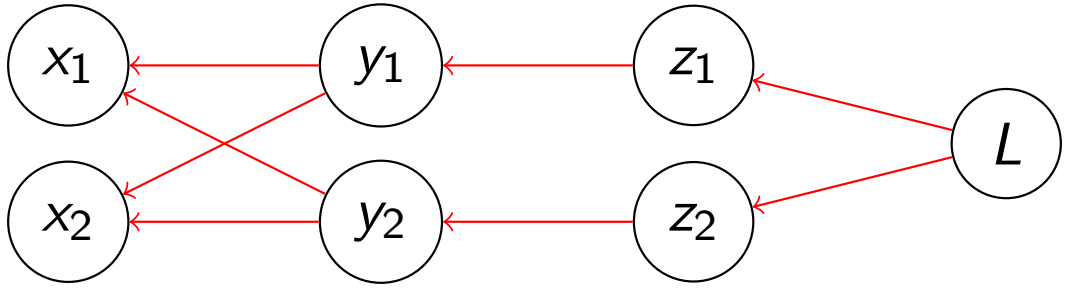


$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \left\{ L = \frac{1}{2}(z_1 - z_2)^2 \right.$$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_1} = 2\frac{\partial L}{\partial y_1} - \frac{\partial L}{\partial y_2}$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_2} = -\frac{\partial L}{\partial y_1} + \frac{\partial L}{\partial y_2}$$

Backpropagation



$$\begin{cases} y_1 = 2x_1 - x_2 \\ y_2 = -x_1 + x_2 \end{cases} \quad \begin{cases} z_1 = y_1^2 \\ z_2 = e^{y_2} \end{cases} \quad \begin{cases} L = \frac{1}{2}(z_1 - z_2)^2 \end{cases}$$

$$\begin{cases} \frac{\partial L}{\partial x_1} = 2\frac{\partial L}{\partial y_1} - \frac{\partial L}{\partial y_2} \\ \frac{\partial L}{\partial x_2} = -\frac{\partial L}{\partial y_1} + \frac{\partial L}{\partial y_2} \end{cases} \quad \begin{cases} \frac{\partial L}{\partial y_1} = 2y_1\frac{\partial L}{\partial z_1} \\ \frac{\partial L}{\partial y_2} = e^{y_2}\frac{\partial L}{\partial z_2} \end{cases} \quad \begin{cases} \frac{\partial L}{\partial z_1} = z_1 - z_2 \\ \frac{\partial L}{\partial z_2} = -(z_1 - z_2) \end{cases}$$

Backpropagation

- To compute the gradient of the network parameters we compute the gradients one layer at a time.
- For each layer type we implement the forward and backward passes.
- Computing the gradient with respect to all parameters in the network is done by running backward passes through all layers in the network starting at the loss.
- This approach highly modular and we can easily add new layers and change the architecture.

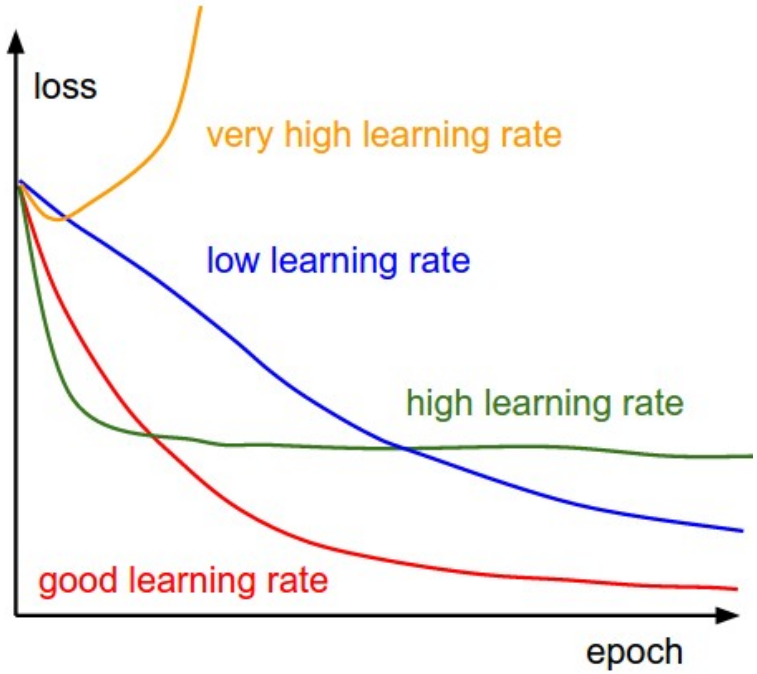
Implementing Backpropagation

- You will derive and implement it for a fully connected layer, relu layer and softmax loss.
- Important to check that it is correct.

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + \mathbf{e}_i\epsilon) - f(\mathbf{x} - \mathbf{e}_i\epsilon)}{2\epsilon}$$

- In the code there are tests to check that your implementation is correct.

Practical Training Tips - The Learning Rate



- The most important parameter in training is the learning rate. If the loss is diverging, decrease it. If the loss does not change much early in training you should probably increase it. If the loss seems to decrease, keep the value and let it train for more iterations. When the loss plateaus, decrease the learning rate.

http://cs231n.github.io/neural-networks-3/

Practical Training Tips

- Is the test accuracy much less than the training accuracy? Increase weight decay.
- Tune the learning rate by changing it by a factor 2 or $1/2$ depending on the behaviour of the loss.
- More layers increase the model capacity, but beware of overfitting.
- Do not start training from scratch every time. Reuse the model using `save` and `load` in Matlab.
- Decrease the learning rate and call `training` again with new settings if the loss and accuracy plateaus.
- A commonly used way to initialize parameters is to count how many ingoing connections n_{in} there are and then initialize the parameters by sampling random values from a Gaussian with mean 0 and standard deviation $\sqrt{\frac{2}{n_{in}}}$.

Training with a GPU