deep learning and image retrieval

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, September 2017

image retrieval challenges

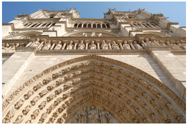









image retrieval challenges

- scale
- viewpoint
- occlusion
- clutter
- lighting

- distinctiveness
- distractors

image classification challenges









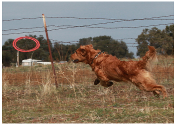



image classification challenges

- scale
- viewpoint
- occlusion
- clutter
- lighting
- number of instances
- texture/color
- pose
- deformability
- intra-class variability

data-driven approach

data-driven approach

↓

feature extraction

↓ representation

data-driven approach

feature extraction

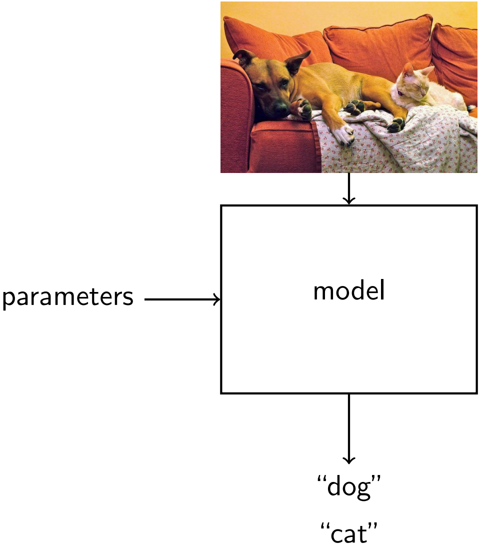
representation

parameters → classifier

"dog"

"cat"

data-driven approach



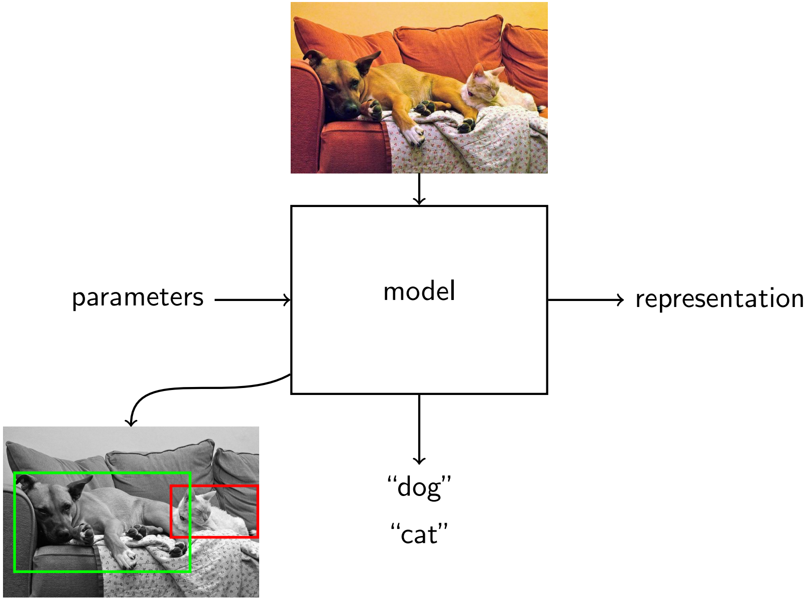
data-driven approach

parameters → model → representation

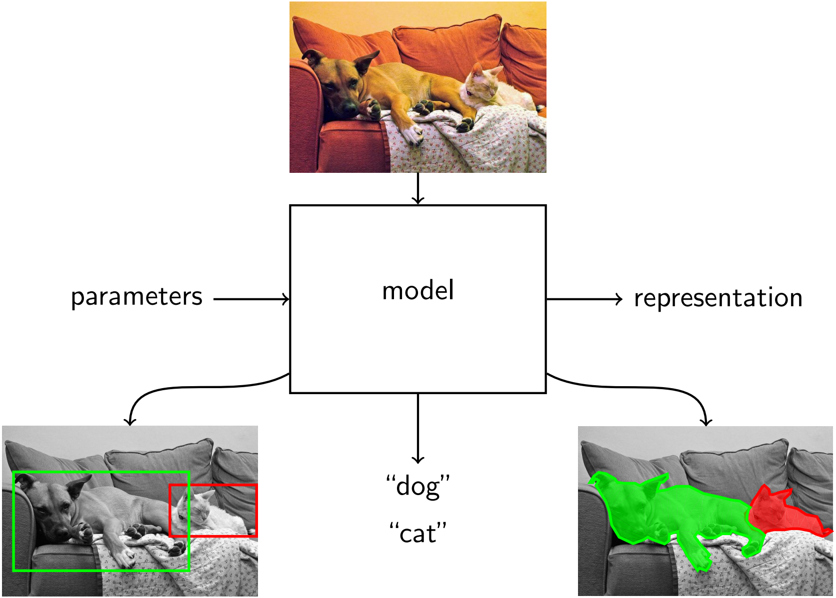
"dog"

"cat"

data-driven approach



data-driven approach



overview

- neural networks
- convolution
- image retrieval
- graph-based methods

neural networks

logistic regression

- class activations

$$a_k = \mathbf{w}_k^\top \mathbf{x} + b_k$$

- posterior class probabilities: softmax

$$y_k(\mathbf{x}) = \mathrm{softmax}_k(\mathbf{a}) := \frac{e^{a_k}}{\sum_j e^{a_j}}$$

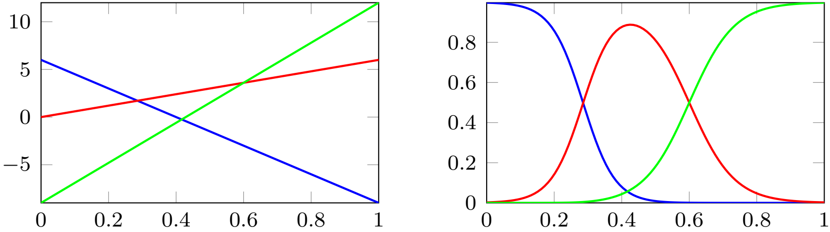
logistic regression

- class activations

$$a_k = \mathbf{w}_k^\top \mathbf{x} + b_k$$

- posterior class probabilities: softmax

$$y_k(\mathbf{x}) = \mathrm{softmax}_k(\mathbf{a}) := \frac{e^{a_k}}{\sum_j e^{a_j}}$$



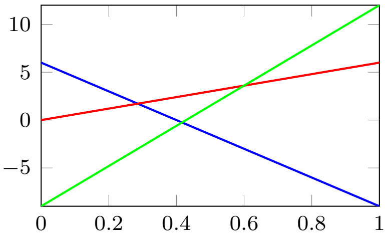
logistic regression

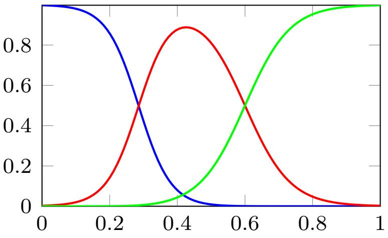
- class activations

$$a_k = \mathbf{w}_k^\top \mathbf{x} + b_k = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

- posterior class probabilities: softmax

$$y_k(\mathbf{x}) = \mathrm{softmax}_k(\mathbf{a}) := \frac{e^{a_k}}{\sum_j e^{a_j}} = p(\mathcal{C}_k|\mathbf{x})$$





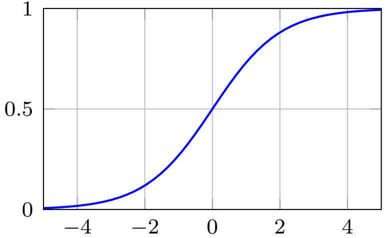
binary logistic regression

- activation

$$a = \mathbf{w}^\top \mathbf{x} + b$$

- posterior probability of class $\mathcal{C}_1$: sigmoid

$$y(\mathbf{x}) = \sigma(a) := \frac{1}{1 + e^{-a}}$$



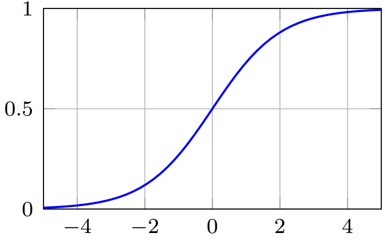
binary logistic regression

- activation

$$a = \mathbf{w}^\top \mathbf{x} + b = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

- posterior probability of class $\mathcal{C}_1$: sigmoid

$$y(\mathbf{x}) = \sigma(a) := \frac{1}{1+e^{-a}} = p(\mathcal{C}_1|\mathbf{x})$$



cross-entropy loss function

- input samples $\mathbf{X} = (x_{nd})$, activations $\mathbf{A} = (a_{nk})$
- output class probabilities $\mathbf{Y} = (y_{nk})$, $y_{nk} = \operatorname{softmax}_k(\mathbf{a}_n)$
- target variables $\mathbf{T} = (t_{nk})$, $t_{nk} = \mathbb{1}[\mathbf{x}_n \in \mathcal{C}_k]$
- average cross-entropy

$$L = -\ln p(\mathbf{T}) = -\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}$$

- gradient

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{1}{N}(\mathbf{Y} - \mathbf{T})$$

by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

cross-entropy loss function

- input samples $\mathbf{X} = (x_{nd})$, activations $\mathbf{A} = (a_{nk})$
- output class probabilities $\mathbf{Y} = (y_{nk})$, $y_{nk} = \mathrm{softmax}_k(\mathbf{a}_n)$
- target variables $\mathbf{T} = (t_{nk})$, $t_{nk} = \mathbb{1}[\mathbf{x}_n \in \mathcal{C}_k]$
- average cross-entropy

$$L = -\ln p(\mathbf{T}) = -\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}$$

- gradient

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{1}{N}(\mathbf{Y} - \mathbf{T})$$

by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

cross-entropy loss function

- input samples $\mathbf{X} = (x_{nd})$, activations $\mathbf{A} = (a_{nk})$
- output class probabilities $\mathbf{Y} = (y_{nk})$, $y_{nk} = \mathrm{softmax}_k(\mathbf{a}_n)$
- target variables $\mathbf{T} = (t_{nk})$, $t_{nk} = \mathbb{1}[\mathbf{x}_n \in \mathcal{C}_k]$
- average cross-entropy

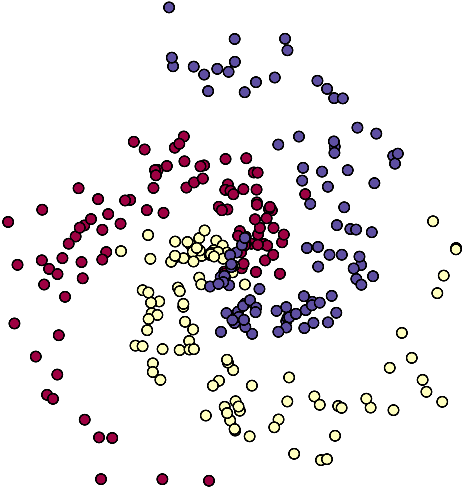
$$L = -\ln p(\mathbf{T}) = -\frac{1}{N}\sum_n \sum_k t_{nk} \ln y_{nk}$$

- gradient

$$\frac{\partial L}{\partial \mathbf{A}} = \frac{1}{N}(\mathbf{Y} - \mathbf{T})$$

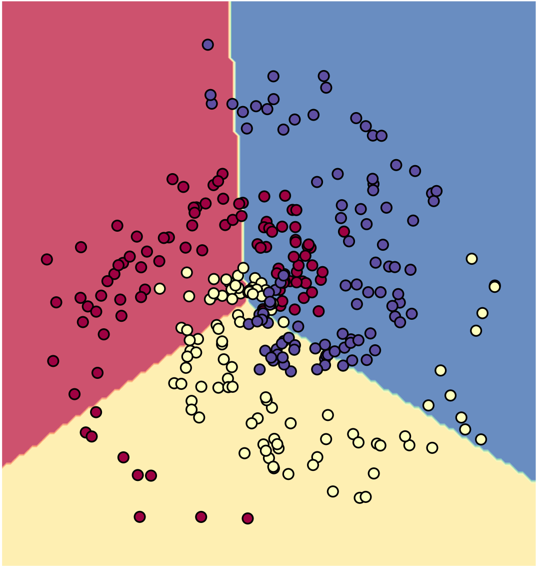
  by increasing a class activation, the loss decreases if the class is correct, and increases otherwise

toy example



credit: Andrej Karpathy

toy example



credit: Andrej Karpathy

two-layer network

- describe each sample with a feature vector obtained by a nonlinear function
- model this function after a (binary) logistic regression unit
- layer 1 activations → "features"

$$\mathbf{z} = h(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1)$$

- layer 2 activations → class probabilities

$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}_2^\top \mathbf{z} + \mathbf{b}_2)$$

two-layer network

- describe each sample with a feature vector obtained by a nonlinear function
- model this function after a (binary) logistic regression unit
- layer 1 activations → "features"

$$\mathbf{z} = h(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1)$$

- layer 2 activations → class probabilities

$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}_2^\top \mathbf{z} + \mathbf{b}_2)$$

two-layer network

- describe each sample with a feature vector obtained by a nonlinear function
- model this function after a (binary) logistic regression unit
- layer 1 activations → "features"

$$\mathbf{z} = h(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1)$$

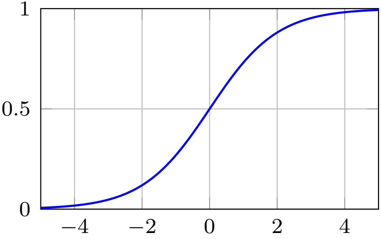
- layer 2 activations → class probabilities

$$\mathbf{y} = \mathrm{softmax}(\mathbf{W}_2^\top \mathbf{z} + \mathbf{b}_2)$$

activation function h

sigmoid (element-wise)

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



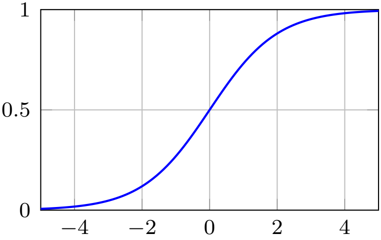
rectified linear unit (ReLU)

$$\mathrm{relu}(x) = [x]_+ = \max(0, x)$$

activation function h

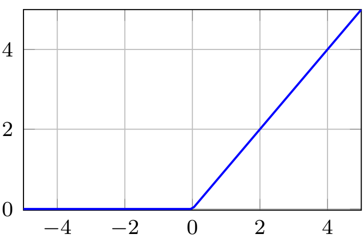
sigmoid (element-wise)

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



rectified linear unit (ReLU)

$$\mathrm{relu}(x) = [x]_+ = \max(0, x)$$



optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = -\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk} + \frac{\lambda}{2}(\|W_1\|_F^2 + \|W_2\|_F^2)$$

- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = -\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk} + \frac{\lambda}{2} (\|W_1\|_F^2 + \|W_2\|_F^2)$$

- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}} (\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = \boxed{-\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}} + \frac{\lambda}{2}(\|W_1\|_F^2 + \|W_2\|_F^2)$$

data term

- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = \underbrace{\boxed{-\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}}}_{\text{data term}} + \underbrace{\boxed{\frac{\lambda}{2} (\|W_1\|_F^2 + \|W_2\|_F^2)}}_{\text{regularization term}}$$

- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}} (\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = \underbrace{\boxed{-\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}}}_{\text{data term}} + \underbrace{\boxed{\frac{\lambda}{2} (\|W_1\|_F^2 + \|W_2\|_F^2)}}_{\text{regularization term}}$$

- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}} (\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

optimization

- input samples $\mathbf{X} = (x_{nd})$, output class probabilities $\mathbf{Y} = (y_{nk})$
- target variables $\mathbf{T} = (t_{nk})$
- network parameters $\boldsymbol{\theta} = ((\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2))$
- loss function

$$L = f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}) = \underbrace{\boxed{-\frac{1}{N} \sum_n \sum_k t_{nk} \ln y_{nk}}}_{\text{data term}} + \underbrace{\boxed{\frac{\lambda}{2}(\|W_1\|_F^2 + \|W_2\|_F^2)}}_{\text{regularization term}}$$

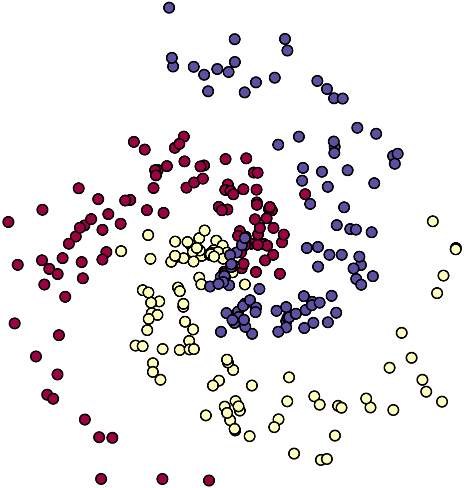
- optimization

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} f(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta})$$

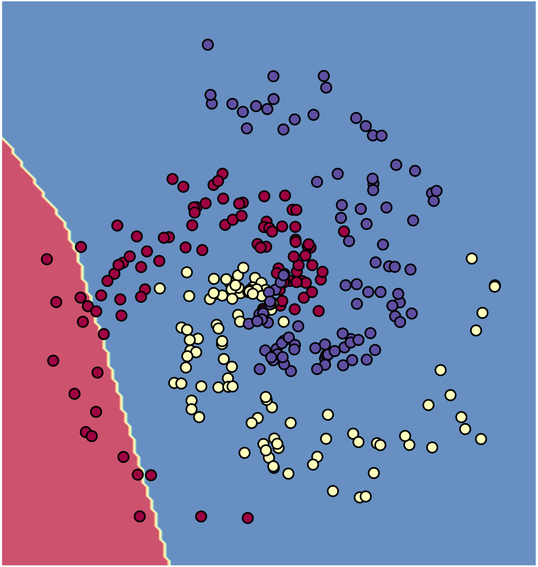
- gradient descent

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \epsilon \frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{X}, \mathbf{T}; \boldsymbol{\theta}^t)$$

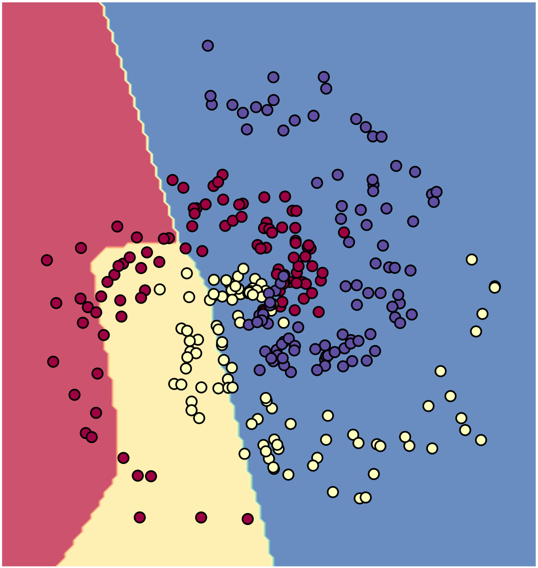
toy example



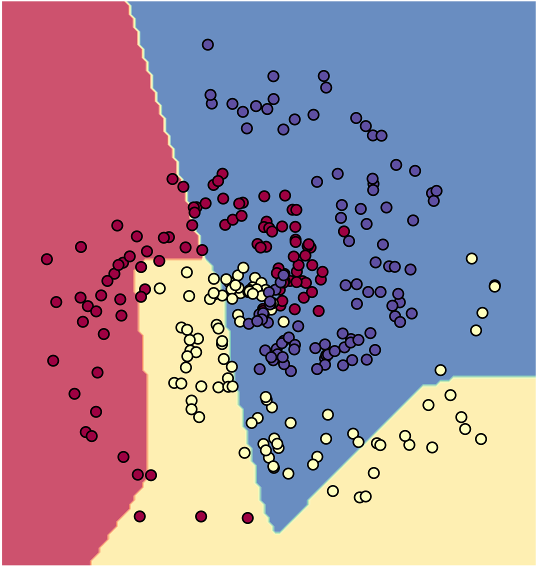
toy example



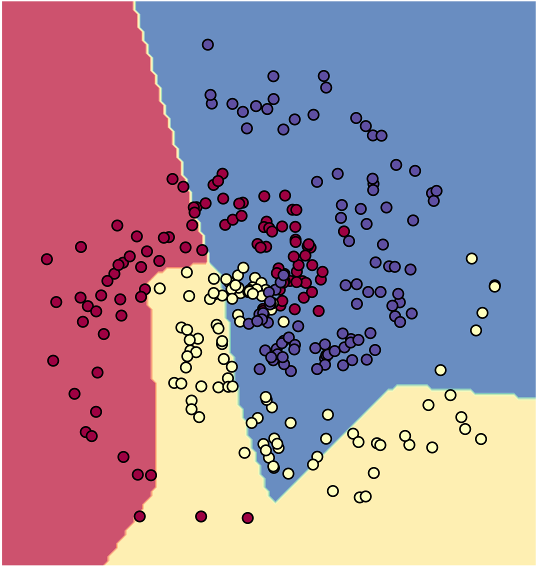
toy example



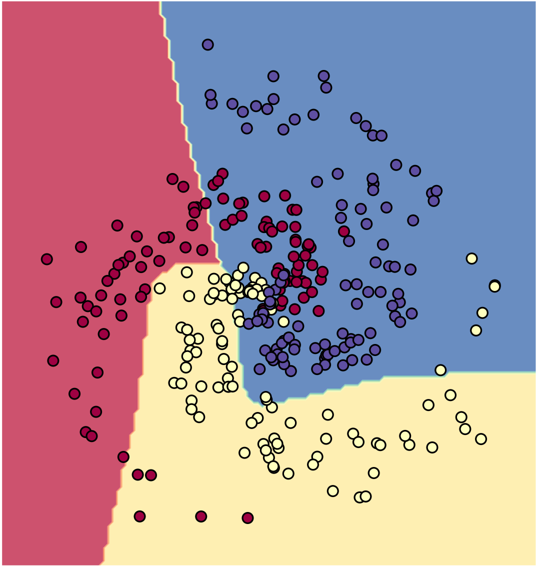
toy example



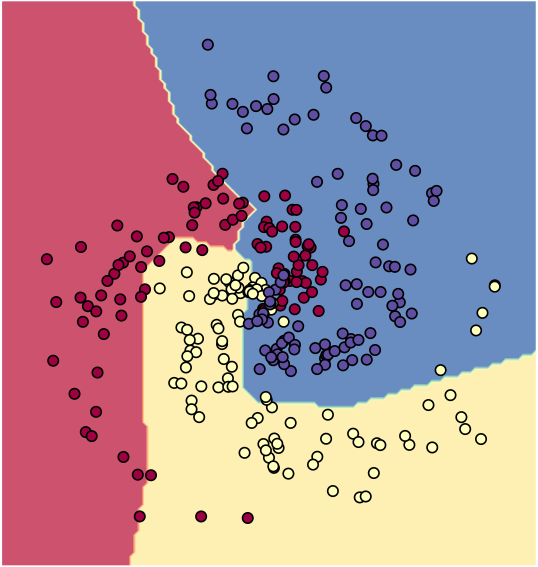
toy example



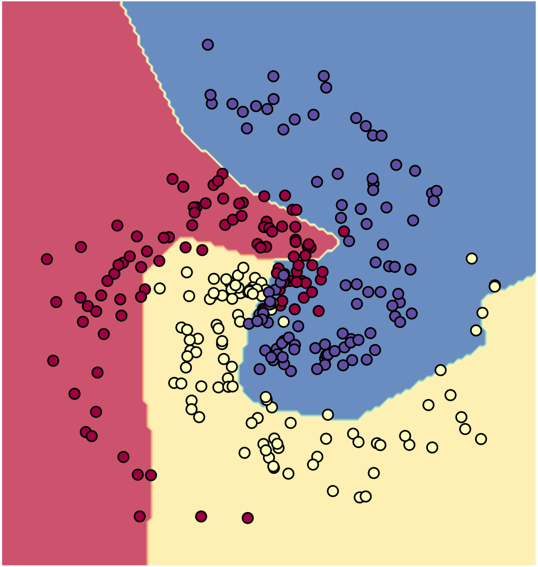
toy example



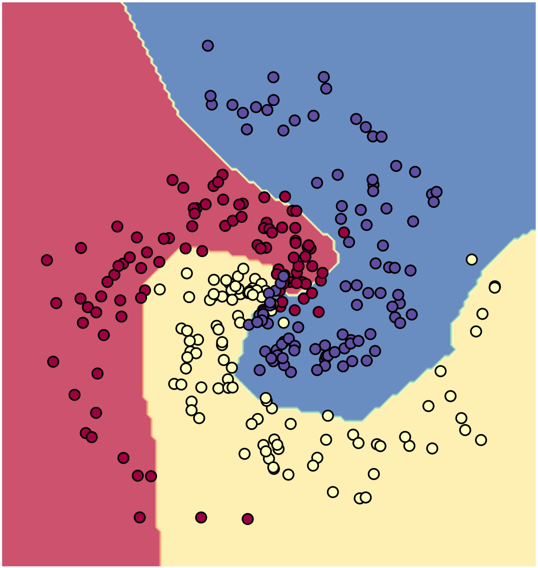
toy example



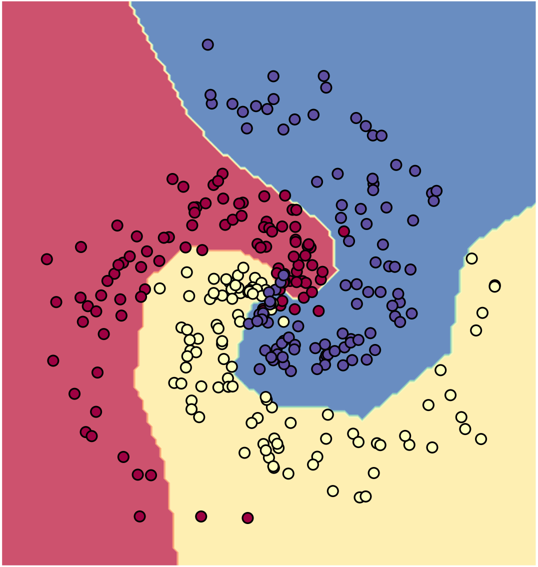
toy example



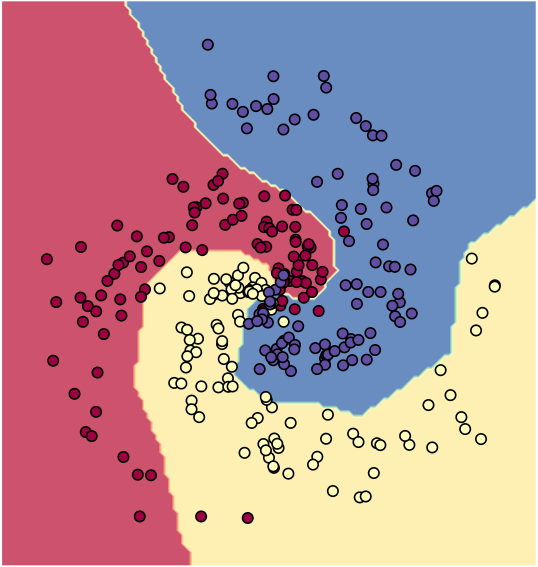
toy example



toy example



toy example



computing the gradient

- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$

$$\mathbf{x} \to \mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3 \to \mathbf{x}_4 \to \mathbf{x}_5 \to L$$

computing the gradient

- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$

$$\mathbf{x} \to \mathbf{x}_1 \to \mathbf{x}_2 \to \mathbf{x}_3 \to \mathbf{x}_4 \to \mathbf{x}_5 \to L$$

computing the gradient

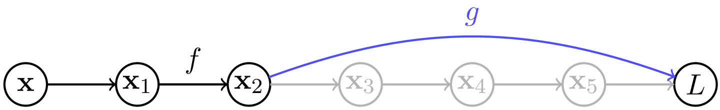
- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



computing the gradient

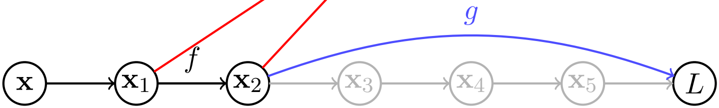
- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot \boxed{Df(\mathbf{x})}$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \boxed{\frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



computing the gradient

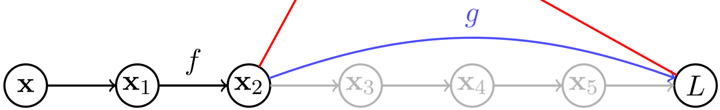
- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = \boxed{Dg(\mathbf{y})} \cdot Df(\mathbf{x})$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \boxed{\frac{\partial L}{\partial \mathbf{x}_2}} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



computing the gradient

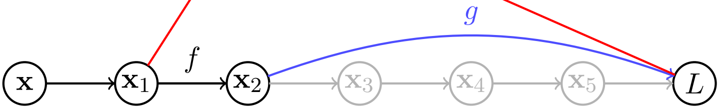
- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$\boxed{D(g \circ f)(\mathbf{x})} = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

- how to use it:

$$\boxed{\frac{\partial L}{\partial \mathbf{x}_1}} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



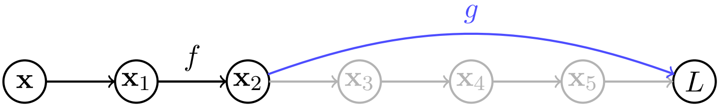
computing the gradient

- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$
$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



computing the gradient

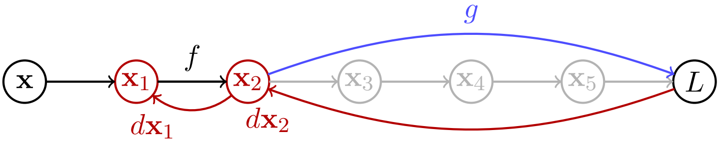
- chain rule: if f is differentiable at $\mathbf{x}$ and g is differentiable at $\mathbf{y} = f(\mathbf{x})$, then $g \circ f$ is differentiable at $\mathbf{x}$ and

$$D(g \circ f)(\mathbf{x}) = Dg(\mathbf{y}) \cdot Df(\mathbf{x})$$

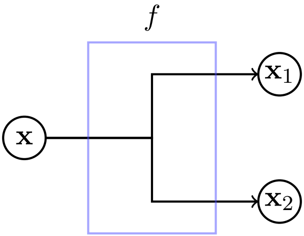
- how to use it:

$$\frac{\partial L}{\partial \mathbf{x}_1} = \frac{\partial L}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1}$$

$$d\mathbf{x}_1 = d\mathbf{x}_2 \cdot Df(\mathbf{x}_1)$$



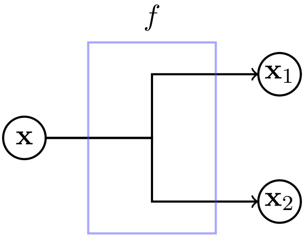
variable sharing



$$Df(\mathbf{x}) = \frac{\partial(\mathbf{x}_1, \mathbf{x}_2)}{\partial \mathbf{x}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\frac{\partial}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial}{\partial \mathbf{x}_1} & \frac{\partial}{\partial \mathbf{x}_2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{\partial}{\partial \mathbf{x}_1} + \frac{\partial}{\partial \mathbf{x}_2}$$

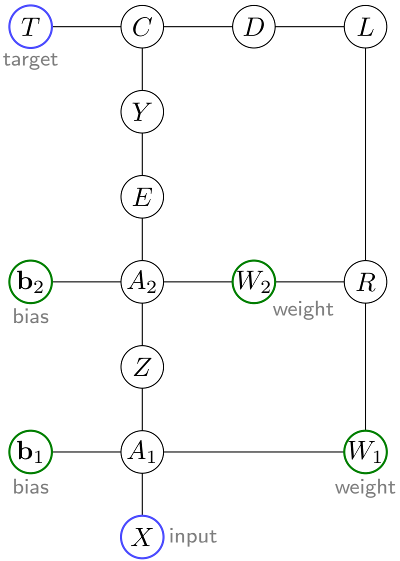
variable sharing



$$Df(\mathbf{x}) = \frac{\partial(\mathbf{x}_1, \mathbf{x}_2)}{\partial \mathbf{x}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

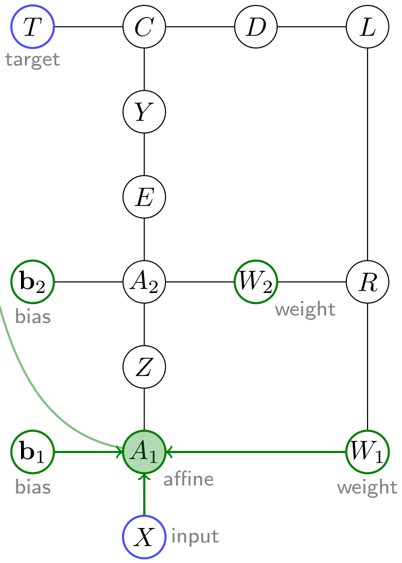
$$\frac{\partial}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial}{\partial \mathbf{x}_1} & \frac{\partial}{\partial \mathbf{x}_2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{\partial}{\partial \mathbf{x}_1} + \frac{\partial}{\partial \mathbf{x}_2}$$

backpropagation



backpropagation

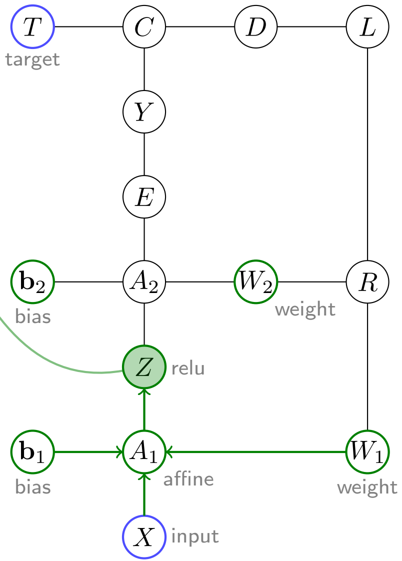
$$\boxed{A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1}$$



backpropagation

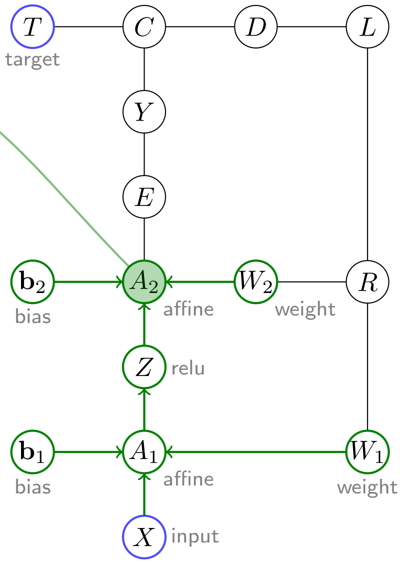
$A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1$

$\boxed{Z = \max(0, A_1)}$



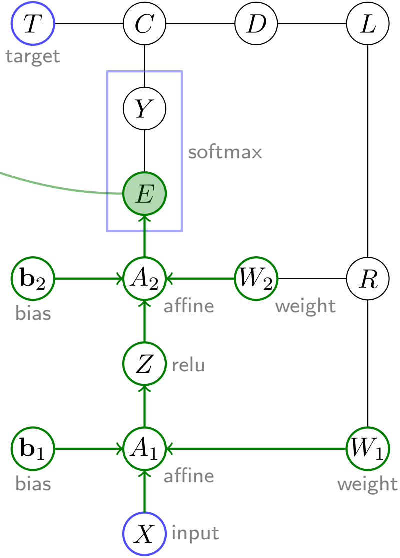
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$\boxed{A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2}$



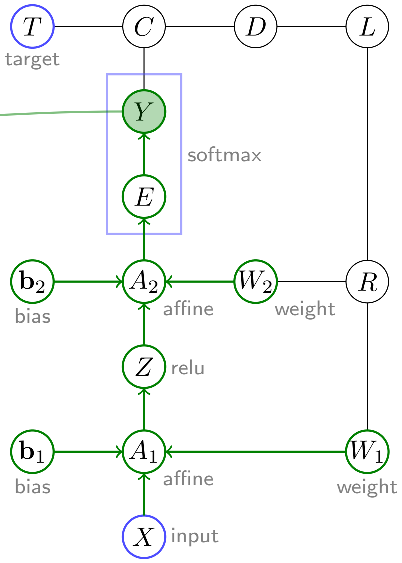
backpropagation

$$A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$$
$$\boxed{E = \exp(A_2)}$$
$$Y = E/\mathrm{sum}_1(E)$$
$$C = -\mathrm{sum}_1(T * \log(Y))$$
$$D = \mathrm{sum}_0(C)/N$$



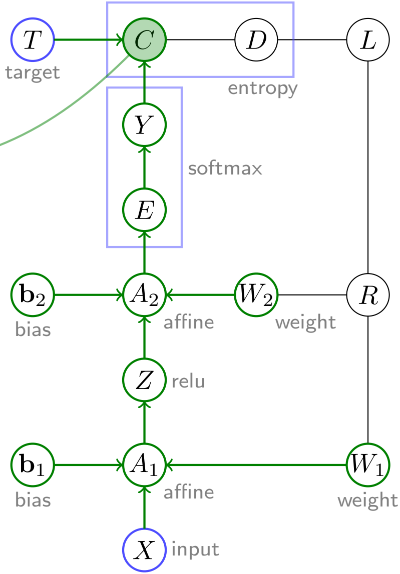
backpropagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/N$



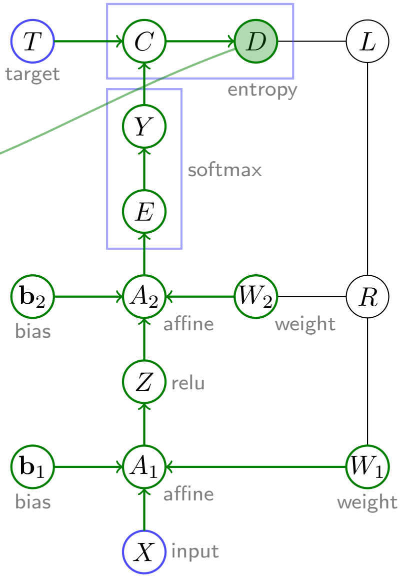
backpropagation

$$A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$$
$$E = \exp(A_2)$$
$$Y = E / \mathrm{sum}_1(E)$$
$$\boxed{C = -\mathrm{sum}_1(T * \log(Y))}$$
$$D = \mathrm{sum}_0(C)/N$$



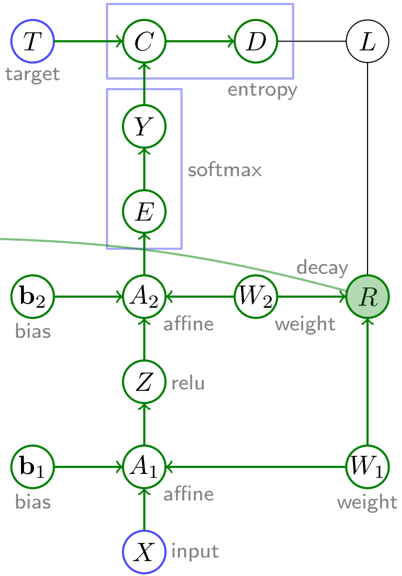
backpropagation

$$A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$$
$$E = \exp(A_2)$$
$$Y = E/\mathrm{sum}_1(E)$$
$$C = -\mathrm{sum}_1(T * \log(Y))$$
$$\boxed{D = \mathrm{sum}_0(C)/N}$$



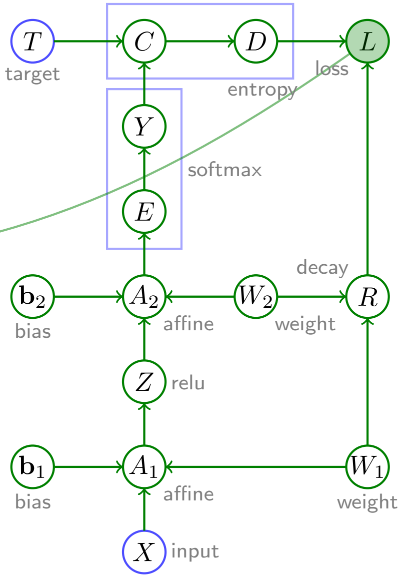
backpropagation

$$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$$
$$Z = \textcolor{blue}{\max}(0, A_1)$$
$$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$$
$$E = \textcolor{blue}{\exp}(A_2)$$
$$Y = E/\textcolor{blue}{\mathrm{sum}}_1(E)$$
$$C = -\textcolor{blue}{\mathrm{sum}}_1(T * \textcolor{blue}{\log}(Y))$$
$$D = \textcolor{blue}{\mathrm{sum}}_0(C)/N$$
$$\boxed{R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)}$$



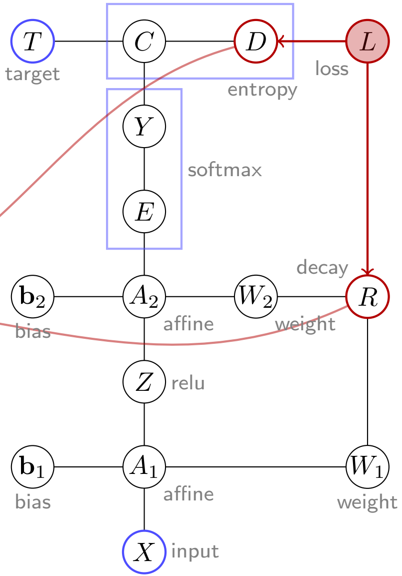
backpropagation

$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$
$Z = \max(0, A_1)$
$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\text{sum}_1(E)$
$C = -\text{sum}_1(T * \log(Y))$
$D = \text{sum}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$\boxed{L = D + R}$



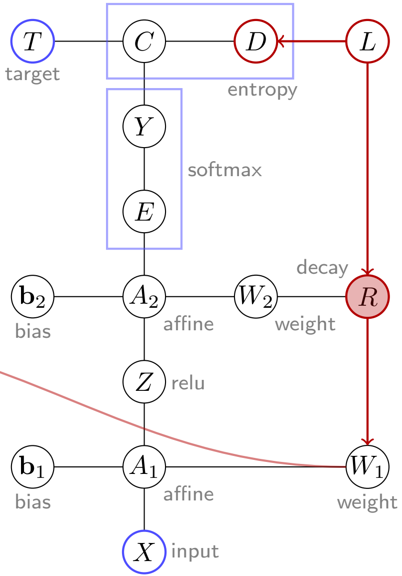
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$
$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$\boxed{(dD, dR) = (dL, dL)}$



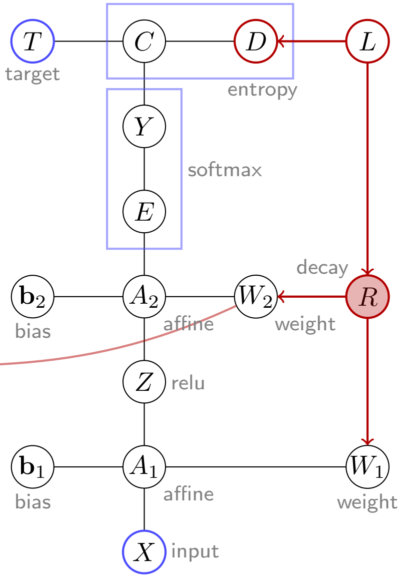
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$
$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$\boxed{dW_1 = dR * \lambda * W_1}$
$\boxed{dW_2 = dR * \lambda * W_2}$



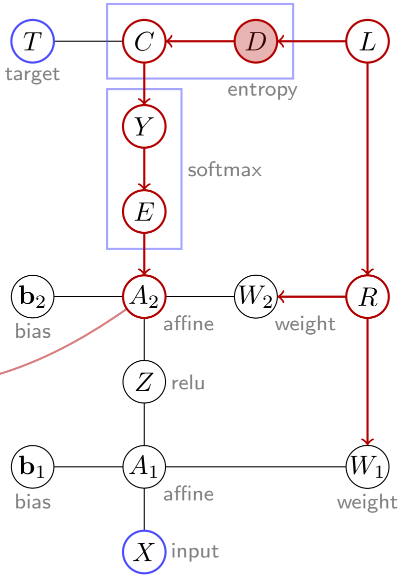
backpropagation

$$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$$
$$Z = \textcolor{blue}{\max}(0, A_1)$$
$$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$$
$$E = \textcolor{blue}{\exp}(A_2)$$
$$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$$
$$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$$
$$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$$
$$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$$
$$L = D + R$$
$$(dD, dR) = (dL, dL)$$
$$\boxed{dW_1 = dR * \lambda * W_1}$$
$$\boxed{dW_2 = dR * \lambda * W_2}$$



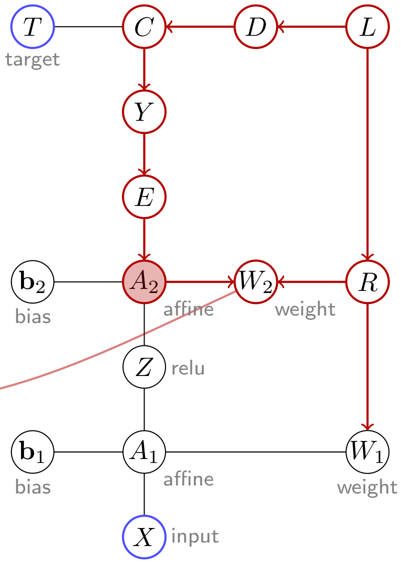
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$
$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$\boxed{dA_2 = dD * (Y - T)/N}$



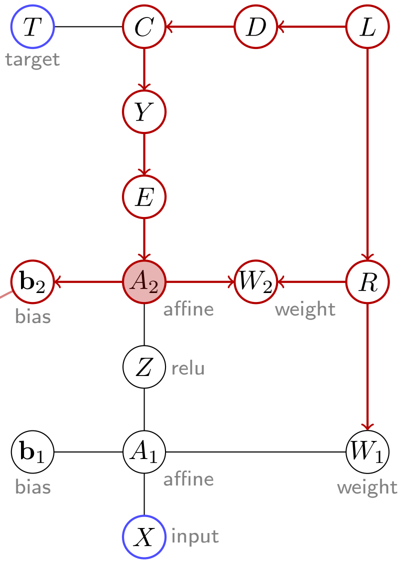
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$
$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$\boxed{dW_2 \mathrel{+}= \textcolor{blue}{\mathrm{dot}}(Z^\top, dA_2)}$
$d\mathbf{b}_2 = \textcolor{blue}{\mathrm{sum}_0}(dA_2)$
$dZ = \textcolor{blue}{\mathrm{dot}}(dA_2, W_2^\top)$



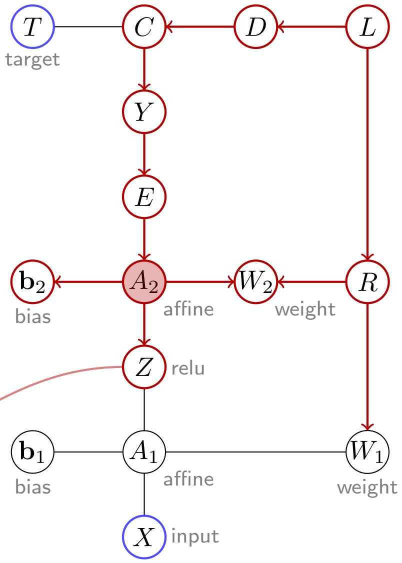
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \mathrm{max}(0, A_1)$
$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E/\mathrm{sum}_1(E)$
$C = -\mathrm{sum}_1(T * \log(Y))$
$D = \mathrm{sum}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$dW_2 \mathrel{+}= \mathrm{dot}(Z^\top, dA_2)$
$\boxed{d\mathbf{b}_2 = \mathrm{sum}_0(dA_2)}$
$dZ = \mathrm{dot}(dA_2, W_2^\top)$



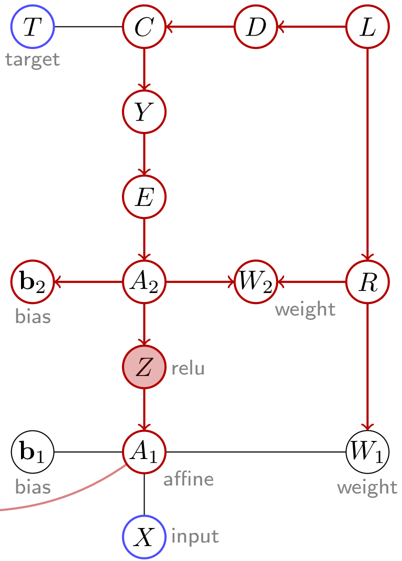
backpropagation

$$A_1 = \text{dot}(X, W_1) + \mathbf{b}_1$$
$$Z = \max(0, A_1)$$
$$A_2 = \text{dot}(Z, W_2) + \mathbf{b}_2$$
$$E = \exp(A_2)$$
$$Y = E/\text{sum}_1(E)$$
$$C = -\text{sum}_1(T * \log(Y))$$
$$D = \text{sum}_0(C)/N$$
$$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$$
$$L = D + R$$
$$(dD, dR) = (dL, dL)$$
$$dW_1 = dR * \lambda * W_1$$
$$dW_2 = dR * \lambda * W_2$$
$$dA_2 = dD * (Y - T)/N$$
$$dW_2 \mathrel{+}= \text{dot}(Z^\top, dA_2)$$
$$d\mathbf{b}_2 = \text{sum}_0(dA_2)$$
$$\boxed{dZ = \text{dot}(dA_2, W_2^\top)}$$



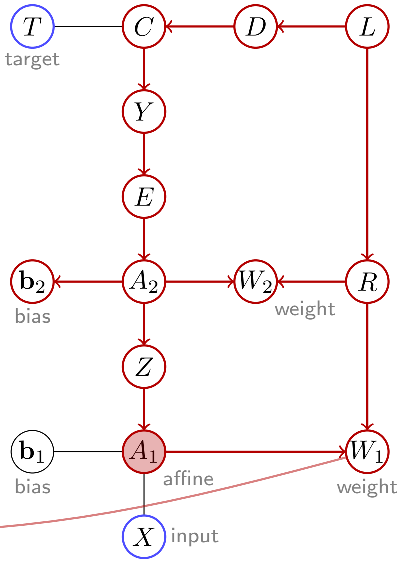
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}}_1(E)$
$C = -\textcolor{blue}{\mathrm{sum}}_1(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$dW_2 \mathrel{+}= \textcolor{blue}{\mathrm{dot}}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \textcolor{blue}{\mathrm{sum}}_0(dA_2)$
$dZ = \textcolor{blue}{\mathrm{dot}}(dA_2, W_2^\top)$
$\boxed{dA_1 = dZ * (Z > 0)}$



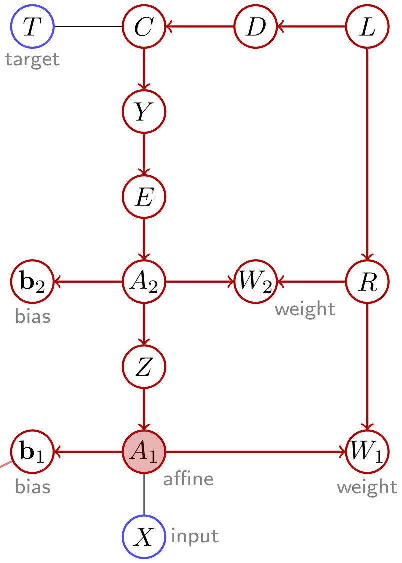
backpropagation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$Z = \textcolor{blue}{\max}(0, A_1)$
$A_2 = \textcolor{blue}{\mathrm{dot}}(Z, W_2) + \mathbf{b}_2$
$E = \textcolor{blue}{\exp}(A_2)$
$Y = E/\textcolor{blue}{\mathrm{sum}_1}(E)$
$C = -\textcolor{blue}{\mathrm{sum}_1}(T * \textcolor{blue}{\log}(Y))$
$D = \textcolor{blue}{\mathrm{sum}_0}(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$dW_2 \mathrel{+}= \textcolor{blue}{\mathrm{dot}}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \textcolor{blue}{\mathrm{sum}_0}(dA_2)$
$dZ = \textcolor{blue}{\mathrm{dot}}(dA_2, W_2^\top)$
$dA_1 = dZ * (Z > 0)$
$\boxed{dW_1 \mathrel{+}= \textcolor{blue}{\mathrm{dot}}(X^\top, dA_1)}$
$\boxed{d\mathbf{b}_1 = \textcolor{blue}{\mathrm{sum}_0}(dA_1)}$



backpropagation

$$
\begin{aligned}
A_1 &= \mathrm{dot}(X, W_1) + \mathbf{b}_1 \\
Z &= \max(0, A_1) \\
A_2 &= \mathrm{dot}(Z, W_2) + \mathbf{b}_2 \\
E &= \exp(A_2) \\
Y &= E/\mathrm{sum}_1(E) \\
C &= -\mathrm{sum}_1(T * \log(Y)) \\
D &= \mathrm{sum}_0(C)/N \\
R &= \tfrac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2) \\
L &= D + R \\
(dD, dR) &= (dL, dL) \\
dW_1 &= dR * \lambda * W_1 \\
dW_2 &= dR * \lambda * W_2 \\
dA_2 &= dD * (Y - T)/N \\
dW_2 &\mathrel{+}= \mathrm{dot}(Z^\top, dA_2) \\
d\mathbf{b}_2 &= \mathrm{sum}_0(dA_2) \\
dZ &= \mathrm{dot}(dA_2, W_2^\top) \\
dA_1 &= dZ * (Z > 0) \\
dW_1 &\mathrel{+}= \mathrm{dot}(X^\top, dA_1) \\
d\mathbf{b}_1 &= \boxed{\mathrm{sum}_0(dA_1)}
\end{aligned}
$$



automatic differentiation

$$
\begin{aligned}
A_1 &= \mathrm{dot}(X, W_1) + \mathbf{b}_1 \\
Z &= \max(0, A_1) \\
A_2 &= \mathrm{dot}(Z, W_2) + \mathbf{b}_2 \\
E &= \exp(A_2) \\
Y &= E / \mathrm{sum}_1(E) \\
C &= -\mathrm{sum}_1(T * \log(Y)) \\
D &= \mathrm{sum}_0(C)/N \\
R &= \tfrac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2) \\
L &= D + R
\end{aligned}
$$

$$
\begin{aligned}
(dD, dR) &= (dL, dL) \\
dW_1 &= dR * \lambda * W_1 \\
dW_2 &= dR * \lambda * W_2 \\
dA_2 &= dD * (Y - T)/N \\
dW_2 &\mathrel{+}= \mathrm{dot}(Z^\top, dA_2) \\
d\mathbf{b}_2 &= \mathrm{sum}_0(dA_2) \\
dZ &= \mathrm{dot}(dA_2, W_2^\top) \\
dA_1 &= dZ * (Z > 0) \\
dW_1 &\mathrel{+}= \mathrm{dot}(X^\top, dA_1) \\
d\mathbf{b}_1 &= \mathrm{sum}_0(dA_1)
\end{aligned}
$$

what is an easy way to automatically generate the backward code from the forward one?

automatic differentiation

$$
\begin{aligned}
&A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1 \\
&\boxed{Z = \max(0, A_1)} \\
&A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2 \\
&E = \exp(A_2) \\
&Y = E/\mathrm{sum}_1(E) \\
&C = -\mathrm{sum}_1(T * \log(Y)) \\
&D = \mathrm{sum}_0(C)/N \\
&R = \tfrac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2) \\
&L = D + R \\
&(dD, dR) = (dL, dL) \\
&dW_1 = dR * \lambda * W_1 \\
&dW_2 = dR * \lambda * W_2 \\
&dA_2 = dD * (Y - T)/N \\
&dW_2 \mathrel{+}= \mathrm{dot}(Z^\top, dA_2) \\
&d\mathbf{b}_2 = \mathrm{sum}_0(dA_2) \\
&dZ = \mathrm{dot}(dA_2, W_2^\top) \\
&\boxed{dA_1 = dZ * (Z > 0)} \\
&dW_1 \mathrel{+}= \mathrm{dot}(X^\top, dA_1) \\
&d\mathbf{b}_1 = \mathrm{sum}_0(dA_1)
\end{aligned}
$$

def relu(A):
    $\boxed{Z = \max(0, A)}$
    def back(dZ, dA):
        $\boxed{dA \mathrel{+}= dZ * (Z > 0)}$
    return *node*(Z, back)

automatic differentiation

$A_1 = \textcolor{blue}{\mathrm{dot}}(X, W_1) + \mathbf{b}_1$
$\boxed{Z = \mathrm{relu}(A_1)}$
$A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2$
$E = \exp(A_2)$
$Y = E / \mathrm{sum}_1(E)$
$C = -\mathrm{sum}_1(T * \log(Y))$
$D = \mathrm{sum}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$dW_2 \mathrel{+}= \mathrm{dot}(Z^\top, dA_2)$
$d\mathbf{b}_2 = \mathrm{sum}_0(dA_2)$
$dZ = \mathrm{dot}(dA_2, W_2^\top)$
$\boxed{Z.\mathrm{back}(A_1)}$
$dW_1 \mathrel{+}= \mathrm{dot}(X^\top, dA_1)$
$d\mathbf{b}_1 = \mathrm{sum}_0(dA_1)$

def $\mathrm{relu}(A)$:
  $Z = \max(0, A)$
  def $\mathrm{back}(dZ, dA)$:
    $dA \mathrel{+}= dZ * (Z > 0)$
  return $\mathit{node}(Z, \mathrm{back})$

automatic differentiation

$\boxed{A_1 = \mathrm{dot}(X, W_1) + \mathbf{b}_1}$
$Z = \mathrm{relu}(A_1)$
$\boxed{A_2 = \mathrm{dot}(Z, W_2) + \mathbf{b}_2}$
$E = \exp(A_2)$
$Y = E / \mathrm{sum}_1(E)$
$C = -\mathrm{sum}_1(T * \log(Y))$
$D = \mathrm{sum}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$$\boxed{\begin{aligned} dW_2 &\mathrel{+}= \mathrm{dot}(Z^\top, dA_2) \\ d\mathbf{b}_2 &= \mathrm{sum}_0(dA_2) \\ dZ &= \mathrm{dot}(dA_2, W_2^\top) \end{aligned}}$$
$Z.\,\mathrm{back}(A_1)$
$$\boxed{\begin{aligned} dW_1 &\mathrel{+}= \mathrm{dot}(X^\top, dA_1) \\ d\mathbf{b}_1 &= \mathrm{sum}_0(dA_1) \end{aligned}}$$

def $\mathrm{affine}(X, (W, \mathbf{b}))$:
$\boxed{A = \mathrm{dot}(X, W) + \mathbf{b}}$
def $\mathrm{back}(dA, dX, (dW, d\mathbf{b}))$:
$$\boxed{\begin{aligned} dW &\mathrel{+}= \mathrm{dot}(X^\top, dA) \\ d\mathbf{b} &\mathrel{+}= \mathrm{sum}_0(dA) \\ dX &\mathrel{+}= \mathrm{dot}(dA, W^\top) \end{aligned}}$$
return $node(A, \mathrm{back})$

automatic differentiation

$\boxed{A_1 = \mathrm{affine}(X, (W_1, \mathbf{b}_1))}$
$Z = \mathrm{relu}(A_1)$
$\boxed{A_2 = \mathrm{affine}(Z, (W_2, \mathbf{b}_2))}$
$E = \exp(A_2)$
$Y = E / \mathrm{sum}_1(E)$
$C = -\mathrm{sum}_1(T * \log(Y))$
$D = \mathrm{sum}_0(C)/N$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$dA_2 = dD * (Y - T)/N$
$\boxed{A_2.\mathrm{back}(Z, (W_2, \mathbf{b}_2))}$

$Z.\mathrm{back}(A_1)$
$\boxed{A_1.\mathrm{back}(X, (W_1, \mathbf{b}_1))}$

def $\mathrm{affine}(X, (W, \mathbf{b}))$:
  $A = \mathrm{dot}(X, W) + \mathbf{b}$
  def $\mathrm{back}(dA, dX, (dW, d\mathbf{b}))$:
    $dW \mathrel{+}= \mathrm{dot}(X^\top, dA)$
    $d\mathbf{b} \mathrel{+}= \mathrm{sum}_0(dA)$
    $dX \mathrel{+}= \mathrm{dot}(dA, W^\top)$
  return $\mathit{node}(A, \mathrm{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$\boxed{\begin{aligned} E &= \exp(A_2) \\ Y &= E / \text{sum}_1(E) \\ C &= -\text{sum}_1(T * \log(Y)) \\ D &= \text{sum}_0(C)/N \end{aligned}}$
$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$\boxed{dA_2 = dD * (Y - T)/N}$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{entropy}(A, T)$:
$\boxed{\begin{aligned} E &= \exp(A) \\ Y &= E / \text{sum}_1(E) \\ C &= -\text{sum}_1(T * \log(Y)) \\ D &= \text{sum}_0(C)/N \end{aligned}}$
def $\text{back}(dD, dA, \_)$:
$\boxed{dA \mathrel{+}= dD * (Y - T)/N}$
return $\textit{node}(D, \text{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$\boxed{D = \text{entropy}(A_2, T)}$

$R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$\boxed{D.\text{back}(A_2, T)}$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{entropy}(A, T)$:
  $E = \exp(A)$
  $Y = E/\text{sum}_1(E)$
  $C = -\text{sum}_1(T * \log(Y))$
  $D = \text{sum}_0(C)/N$
def $\text{back}(dD, dA, \_)$:
  $dA \mathrel{+}= dD * (Y - T)/N$
return $\mathit{node}(D, \text{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{entropy}(A_2, T)$

$\boxed{R = \frac{\lambda}{2} * (\|W_1\|_F^2 + \|W_2\|_F^2)}$
$L = D + R$
$(dD, dR) = (dL, dL)$
$dW_1 = dR * \lambda * W_1$
$dW_2 = dR * \lambda * W_2$
$D.\,\text{back}(A_2, T)$
$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{decay}(W)$:
  $\boxed{R = \frac{\lambda}{2} * \text{sum}(\|w\|_F^2 \text{ **for** } w \text{ **in** } W)}$
  def $\text{back}(dR, dW)$:
    for (w, dw) **in** $\text{zip}(W, dW)$:
      $dw \mathrel{+}= dR * \lambda * w$
  return $node(R, \text{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{entropy}(A_2, T)$

$\boxed{R = \text{decay}((W_1, W_2))}$
$L = D + R$
$(dD, dR) = (dL, dL)$
$\boxed{R.\text{back}((W_1, W_2))}$

$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{decay}(W)$:
  $R = \frac{\lambda}{2} * \text{sum}(\|w\|_F^2$ **for** w **in** $W)$
  def $\text{back}(dR, dW)$:
    for (w, dw) **in** $\text{zip}(W, dW)$:
      $dw \mathrel{+}= dR * \lambda * w$
  return $node(R, \text{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{entropy}(A_2, T)$

$R = \text{decay}((W_1, W_2))$
$\boxed{L = D + R}$
$\boxed{(dD, dR) = (dL, dL)}$
$R.\,\text{back}((W_1, W_2))$

$D.\,\text{back}(A_2, T)$
$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{add}(X)$:
  $\boxed{S = \text{sum}(X)}$
  def $\text{back}(dS, dX)$:
    for dx **in** dX:
      $dx \mathrel{+}= dS$
  return $node(S, \text{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \text{entropy}(A_2, T)$

$R = \text{decay}((W_1, W_2))$
$\boxed{L = \text{add}((D, R))}$
$\boxed{L.\text{back}((D, R))}$
$R.\text{back}((W_1, W_2))$

$D.\text{back}(A_2, T)$
$A_2.\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\text{back}(A_1)$
$A_1.\text{back}(X, (W_1, \mathbf{b}_1))$

def $\text{add}(X)$:
  $S = \text{sum}(X)$
  def $\text{back}(dS, dX)$:
    for dx **in** dX:
      $dx \mathrel{+}= dS$
  return $node(S, \text{back})$

automatic differentiation

$A_1 = \mathrm{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \mathrm{relu}(A_1)$
$A_2 = \mathrm{affine}(Z, (W_2, \mathbf{b}_2))$
$D = \mathrm{entropy}(A_2, T)$

$R = \mathrm{decay}((W_1, W_2))$
$L = \mathrm{add}((D, R))$
$L.\mathrm{back}((D, R))$
$R.\mathrm{back}((W_1, W_2))$

$D.\mathrm{back}(A_2, T)$
$A_2.\mathrm{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\mathrm{back}(A_1)$
$A_1.\mathrm{back}(X, (W_1, \mathbf{b}_1))$

def $\mathrm{loss}(A, T, W)$:
    $D = \mathrm{entropy}(A, T)$
    $R = \mathrm{decay}(W)$
    $L = \mathrm{add}((D, R))$
    def $\mathrm{back}(A, T, W)$:
        $L.\mathrm{back}((D, R))$
        $R.\mathrm{back}(W)$
        $D.\mathrm{back}(A, T)$
    return $\mathit{block}(L, \mathrm{back})$

automatic differentiation

$A_1 = \text{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \text{relu}(A_1)$
$A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2))$
$\boxed{L = \text{loss}(A_2, T, (W_1, W_2))}$

$\boxed{L.\,\text{back}(A_2, T, (W_1, W_2))}$

$A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\text{back}(A_1)$
$A_1.\,\text{back}(X, (W_1, \mathbf{b}_1))$

```
def loss(A, T, W):
    D = entropy(A, T)
    R = decay(W)
    L = add((D, R))
    def back(A, T, W):
        L.back((D, R))
        R.back(W)
        D.back(A, T)
    return block(L, back)
```

automatic differentiation

$A_1 = \mathrm{affine}(X, (W_1, \mathbf{b}_1))$
$Z = \mathrm{relu}(A_1)$
$A_2 = \mathrm{affine}(Z, (W_2, \mathbf{b}_2))$
$L = \mathrm{loss}(A_2, T, (W_1, W_2))$

def $\mathrm{loss}(A, T, W)$:
  $L = \mathrm{entropy}(A, T) + \mathrm{decay}(W)$
  return $\mathit{block}(L)$

def $\mathrm{loss}(A, T, W)$:
  $D = \mathrm{entropy}(A, T)$
  $R = \mathrm{decay}(W)$
  $L = \mathrm{add}((D, R))$
  ~~**def** $\mathrm{back}(A, T, W)$:~~
    ~~$L.\,\mathrm{back}((D, R))$~~
    ~~$R.\,\mathrm{back}(W)$~~
    ~~$D.\,\mathrm{back}(A, T)$~~
  return $\mathit{block}(L, \mathrm{back})$

$L.\,\mathrm{back}(A_2, T, (W_1, W_2))$

$A_2.\,\mathrm{back}(Z, (W_2, \mathbf{b}_2))$

$Z.\,\mathrm{back}(A_1)$
$A_1.\,\mathrm{back}(X, (W_1, \mathbf{b}_1))$

automatic differentiation

$$\boxed{\begin{aligned} &A_1 = \text{affine}(X, (W_1, \mathbf{b}_1)) \\ &Z = \text{relu}(A_1) \\ &A_2 = \text{affine}(Z, (W_2, \mathbf{b}_2)) \end{aligned}}$$

$L = \text{loss}(A_2, T, (W_1, W_2))$

$L.\,\text{back}(A_2, T, (W_1, W_2))$

$$\boxed{\begin{aligned} &A_2.\,\text{back}(Z, (W_2, \mathbf{b}_2)) \\ \\ \\ &Z.\,\text{back}(A_1) \\ &A_1.\,\text{back}(X, (W_1, \mathbf{b}_1)) \end{aligned}}$$

def $\text{model}(X, (U_1, U_2))$:

$$\boxed{\begin{aligned} &A_1 = \text{affine}(X, U_1) \\ &Z = \text{relu}(A) \\ &A_2 = \text{affine}(Z, U_2) \end{aligned}}$$

def $\text{back}(X, (U_1, U_2))$:

$$\boxed{\begin{aligned} &A_2.\,\text{back}(Z, U_2) \\ &Z.\,\text{back}(A) \\ &A_1.\,\text{back}(X, U_1) \end{aligned}}$$

return $\mathit{block}(A_2, \text{back})$

automatic differentiation

$\boxed{A_2 = \mathrm{model}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))}$

$L = \mathrm{loss}(A_2, T, (W_1, W_2))$

$L.\,\mathrm{back}(A_2, T, (W_1, W_2))$

$\boxed{A_2.\,\mathrm{back}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))}$

```
def model(X, (U_1, U_2)):
    A_1 = affine(X, U_1)
    Z = relu(A)
    A_2 = affine(Z, U_2)
  def back(X, (U_1, U_2)):
      A_2.back(Z, U_2)
      Z.back(A)
      A_1.back(X, U_1)
  return block(A_2, back)
```

automatic differentiation

$A_2 = \mathrm{model}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

$L = \mathrm{loss}(A_2, T, (W_1, W_2))$

$L.\mathrm{back}(A_2, T, (W_1, W_2))$

$A_2.\mathrm{back}(X, ((W_1, \mathbf{b}_1), (W_2, \mathbf{b}_2)))$

```
def model(X, (U1, U2)):
    A = affine(relu(affine(X, U1)), U2)
    return block(A)
```

```
def model(X, (U1, U2)):
    A1 = affine(X, U1)
    Z = relu(A)
    A2 = affine(Z, U2)
    def back(X, (U1, U2)):
        A2.back(Z, U2)
        Z.back(A)
        A1.back(X, U1)
    return block(A2, back)
```

convolution

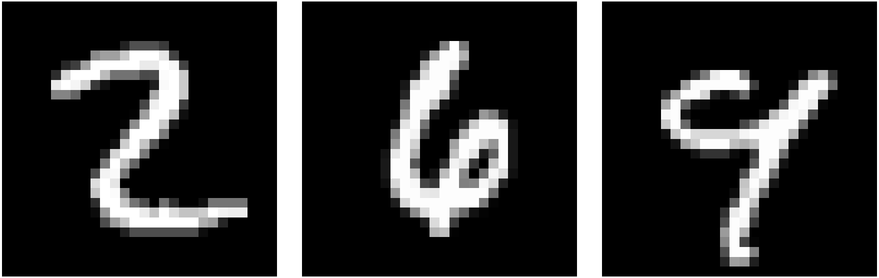
MNIST digits dataset

- 10 classes, 60k training images, 10k test images, 28×28 images

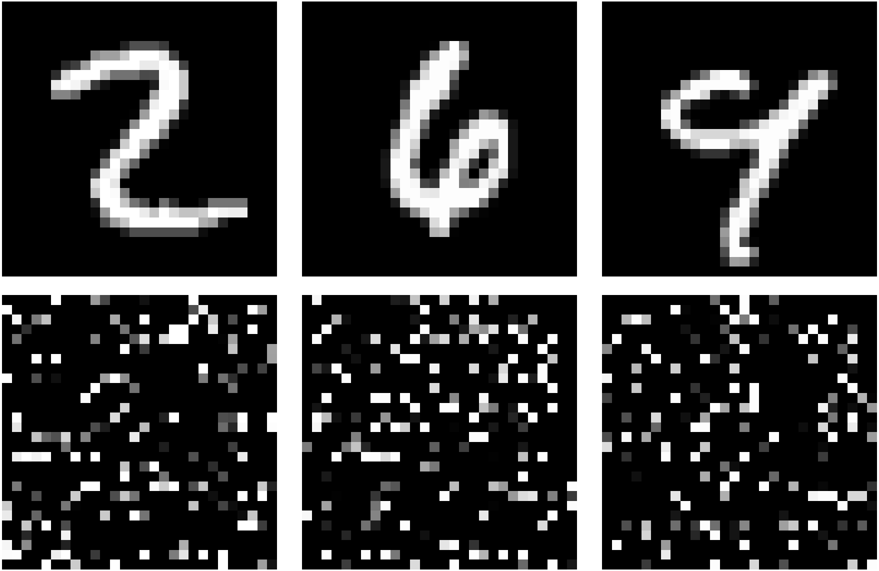
fully connected layers

- a two-layer network with fully connected layers can easily learn to classify MNIST digits (less that 3% error), but learns more than actually required

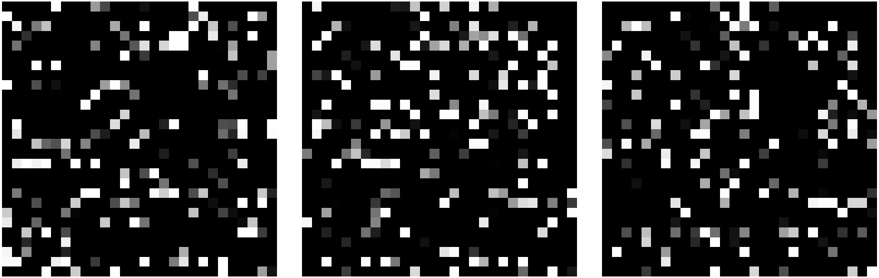
shuffling the dimensions



shuffling the dimensions



shuffling the dimensions



convolution

- convolution results in sparser connections between units, local receptive fields, translation equivariance, shared weights and less parameters to learn
- a convolutional network performs better (less than 1% error), but not on shuffled digits

convolution

- convolution results in sparser connections between units, local receptive fields, translation equivariance, shared weights and less parameters to learn
- a convolutional network performs better (less than 1% error), but not on shuffled digits

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n = 0]$, so that $x[n] = \sum_k x[k]\delta[n-k]$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f\left(\sum_k x[k] t_k(\delta)\right)[n] = \sum_k x[k] t_k(f(\delta))[n] \\ &= \sum_k x[k] h[n-k] \end{aligned}$$

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n=0]$, so that $x[n] = \sum_k x[k]\delta[n-k]$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$f(x)[n] = f\left(\sum_k x[k] t_k(\delta)\right)[n] = \sum_k x[k] t_k(f(\delta))[n]$$

$$= \sum_k x[k] h[n-k]$$

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n=0]$, so that $x[n] = \sum_k x[k]\delta[n-k]$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f\left(\sum_k x[k] t_k(\delta)\right)[n] = \sum_k x[k] t_k(f(\delta))[n] \\ &= \sum_k x[k] h[n-k] \end{aligned}$$

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n=0]$, so that $x[n] = \boxed{\sum_k x[k]\delta[n-k]}$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= f\left(\boxed{\sum_k x[k] t_k(\delta)}\right)[n] = \sum_k x[k] t_k(f(\delta))[n] \\ &= \sum_k x[k] h[n-k] \end{aligned}$$

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n=0]$, so that $x[n] = \sum_k x[k]\delta[n-k]$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = x * h$:

$$\begin{aligned} f(x)[n] &= \boxed{f}\left(\sum_k x[k]t_k(\delta)\right)[n] = \sum_k x[k]t_k\boxed{(f(}\delta))[n] \\ &= \sum_k x[k]h[n-k] \end{aligned}$$

LTI systems and convolution

- discrete-time signal: $x[n]$, $n \in \mathbb{Z}$
- translation (or shift, or delay) $t_k(x)[n] = x[n-k]$, $k \in \mathbb{Z}$
- unit impulse $\delta[n] = \mathbb{1}[n=0]$, so that $x[n] = \sum_k x[k]\delta[n-k]$
- linear system (or filter)

$$f\left(\sum_i a_i x_i\right) = \sum_i a_i f(x_i)$$

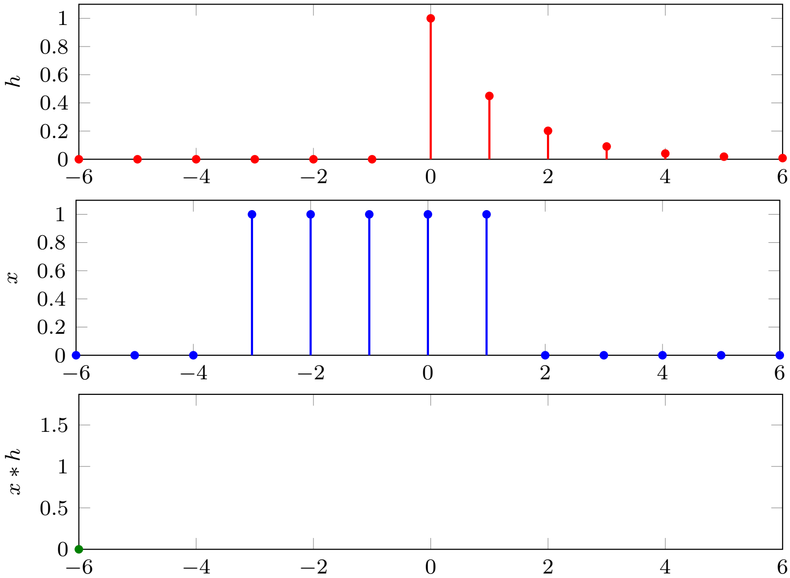
- time-invariant (or translation equivariant) system

$$f(t_k(x)) = t_k(f(x))$$

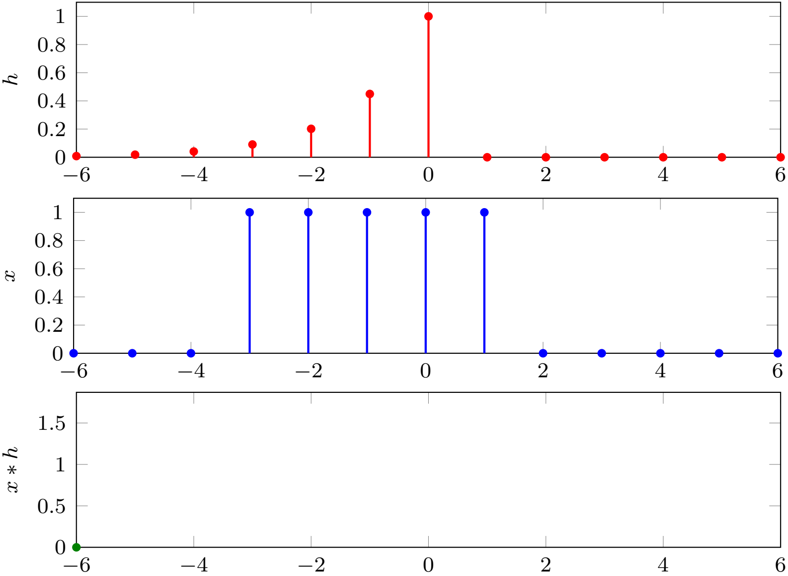
- if f is LTI with impulse response $h = f(\delta)$, then $f(x) = \boxed{x * h}$:

$$f(x)[n] = f\left(\sum_k x[k] t_k(\delta)\right)[n] = \sum_k x[k] t_k(f(\delta))[n]$$
$$= \boxed{\sum_k x[k] h[n-k]}$$

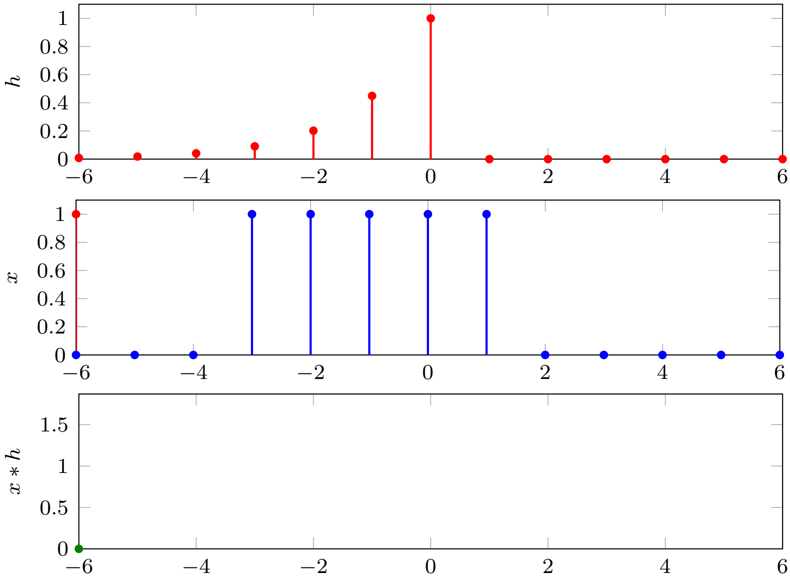
convolution



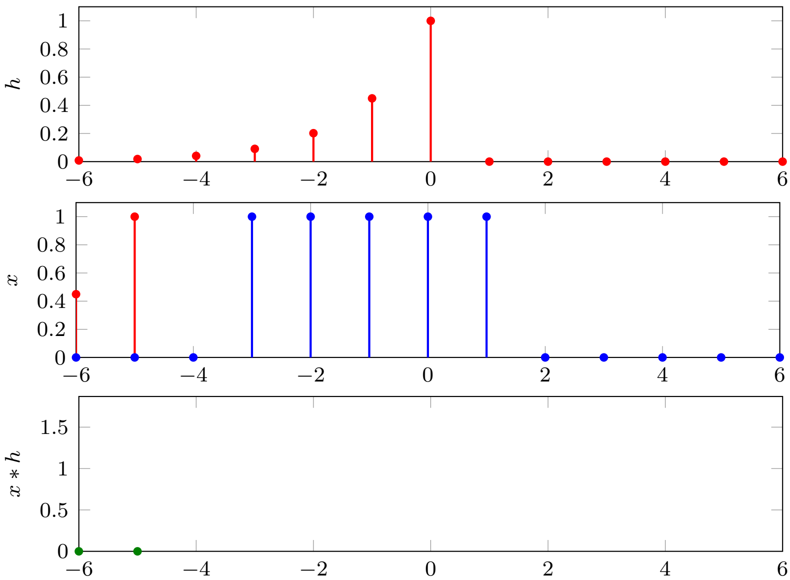
convolution



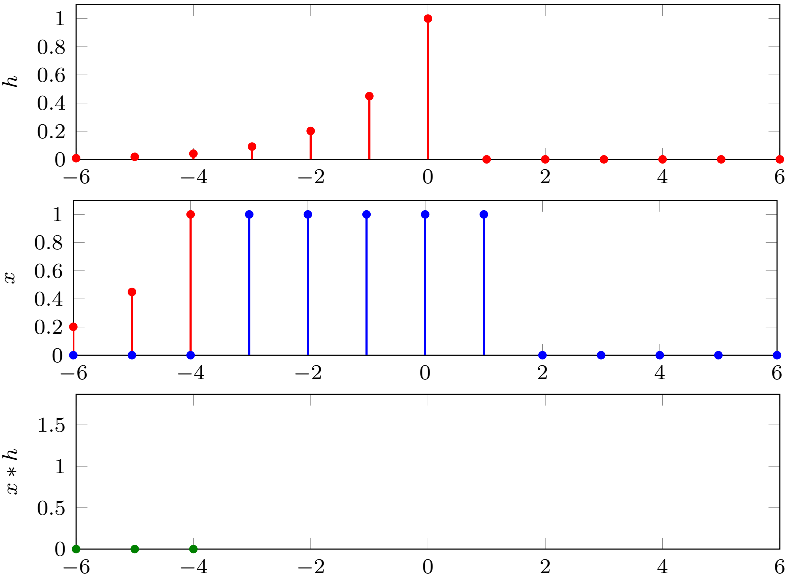
convolution



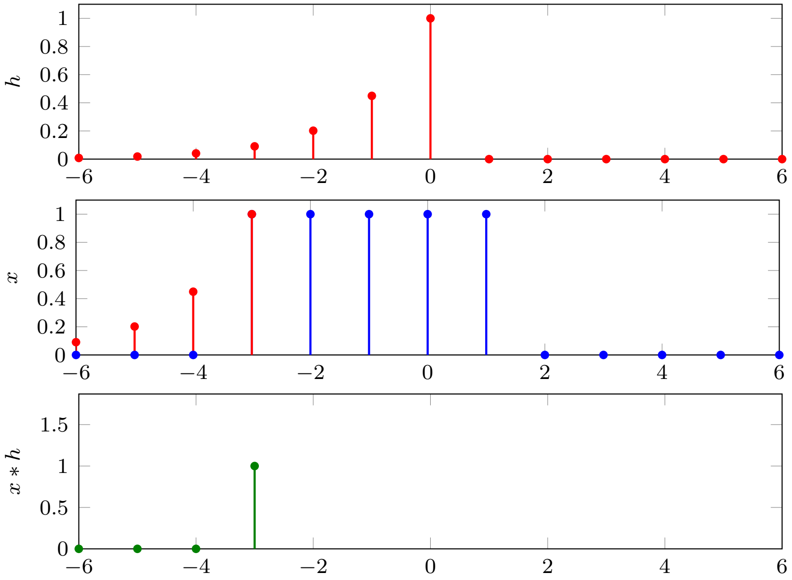
convolution



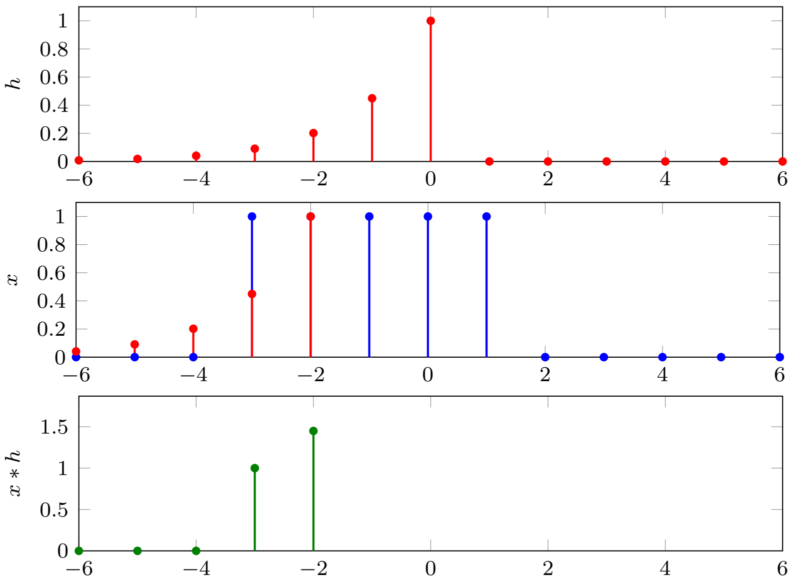
convolution



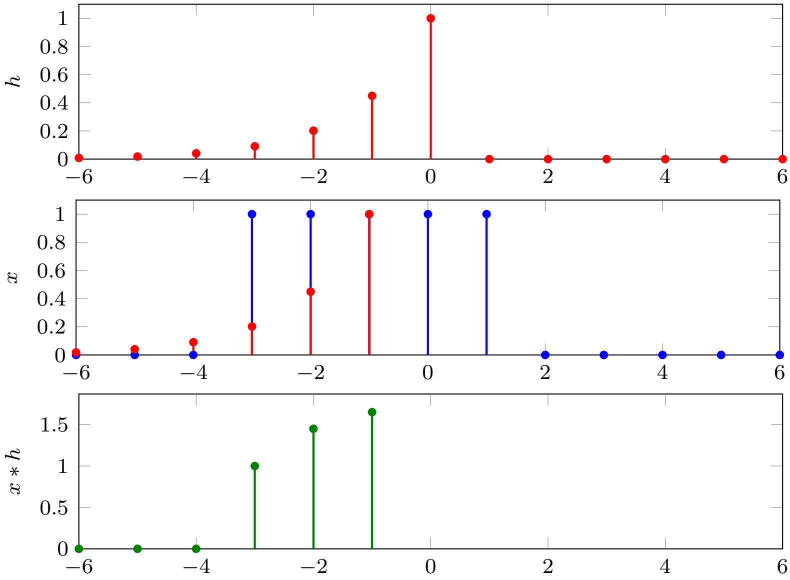
convolution



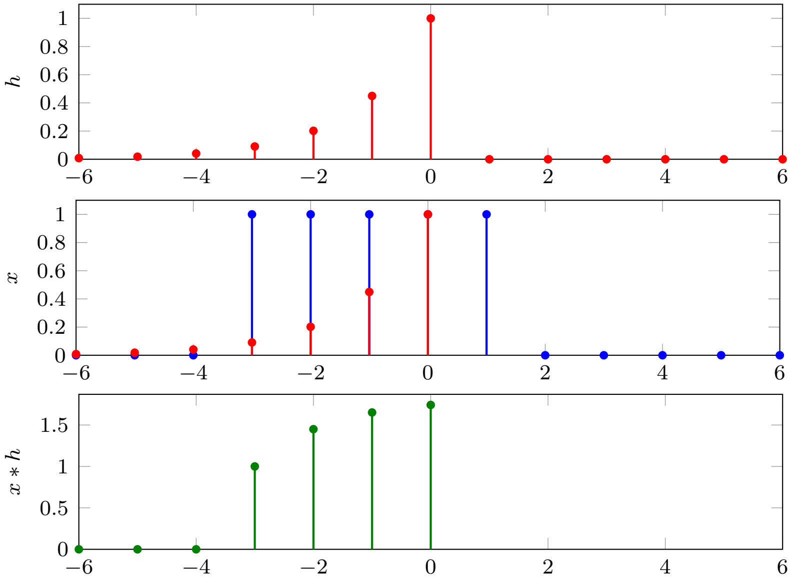
convolution



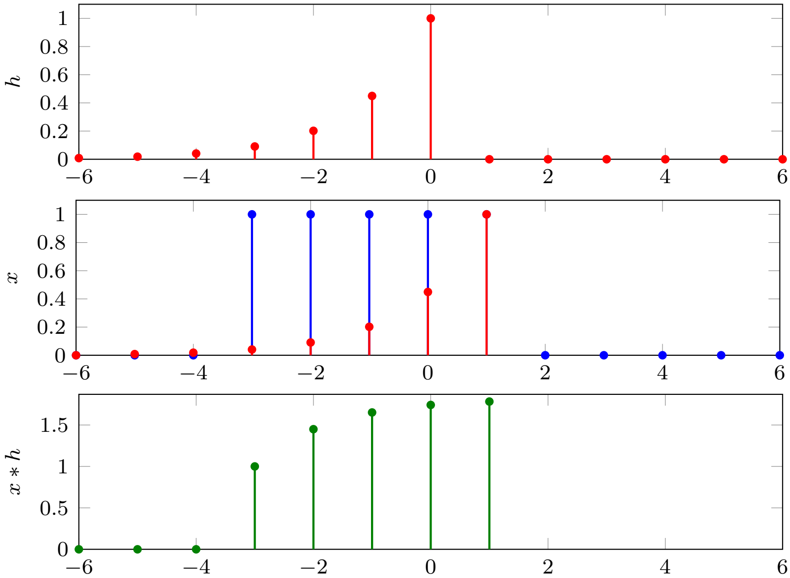
convolution



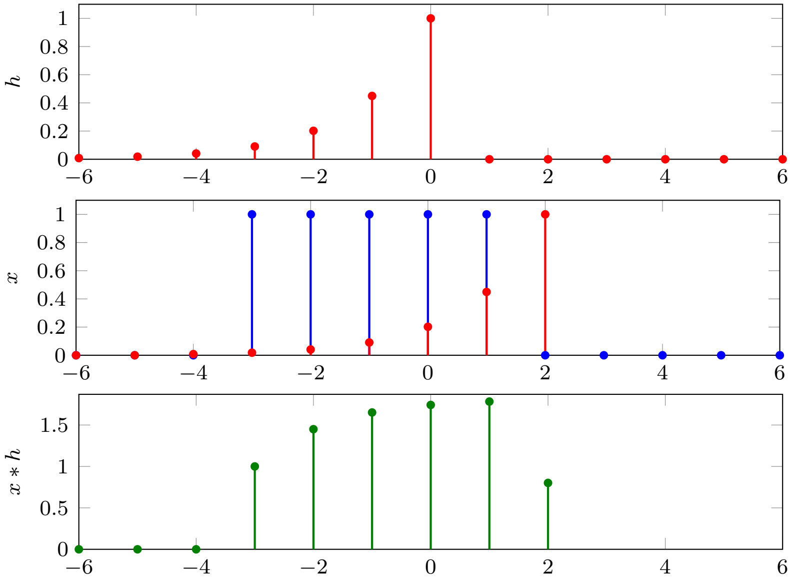
convolution



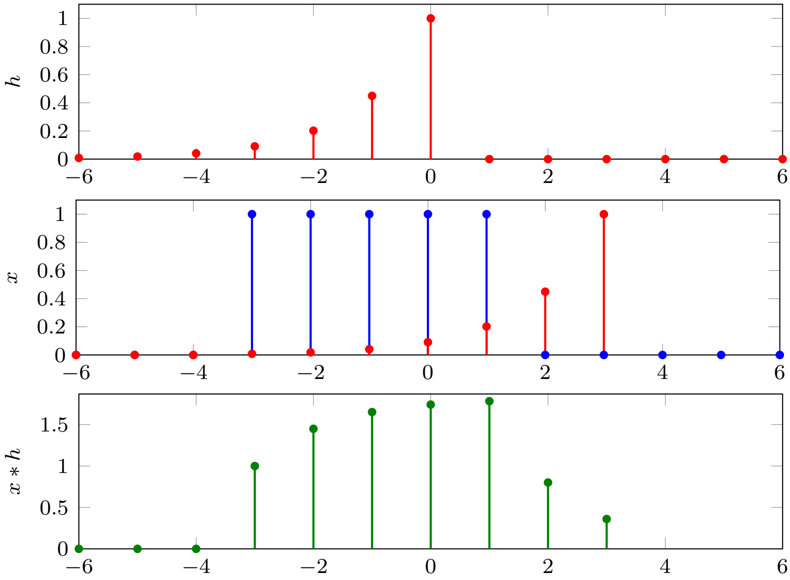
convolution



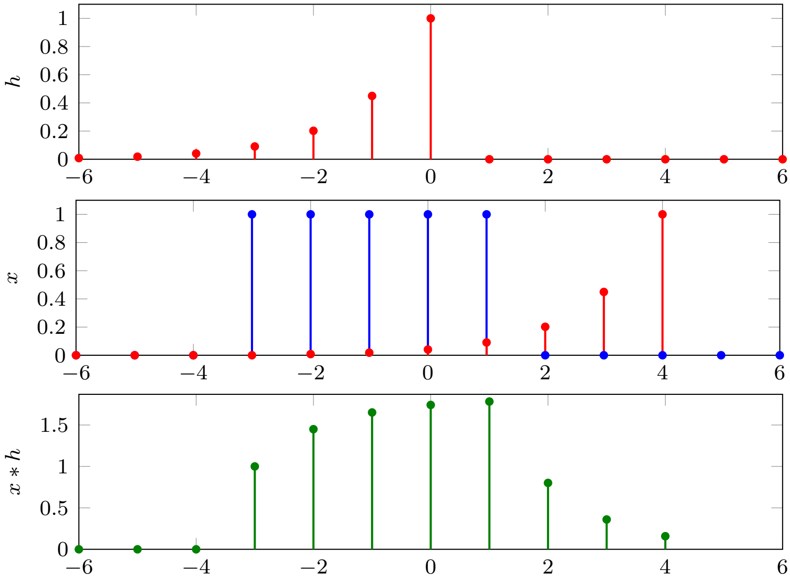
convolution



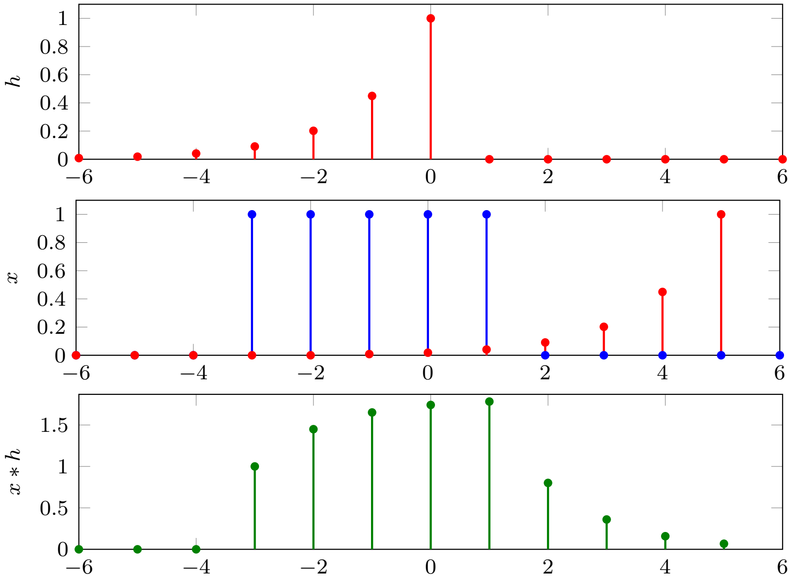
convolution



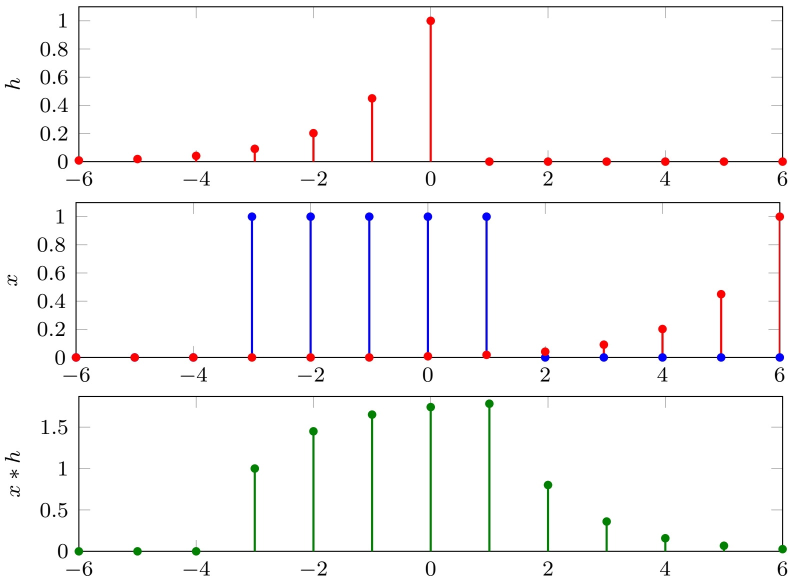
convolution



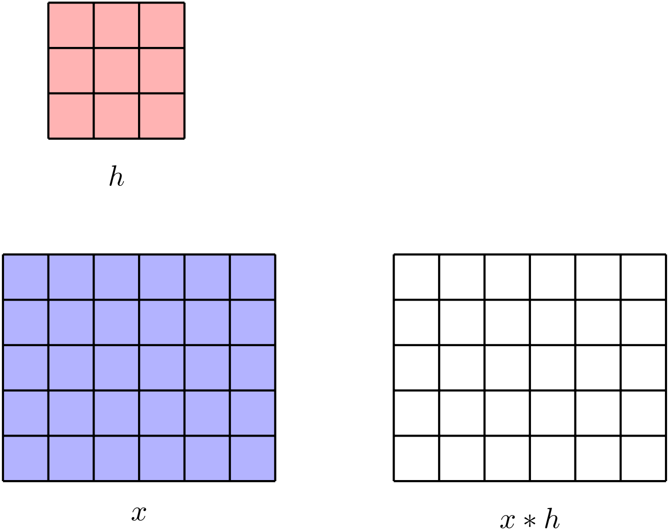
convolution



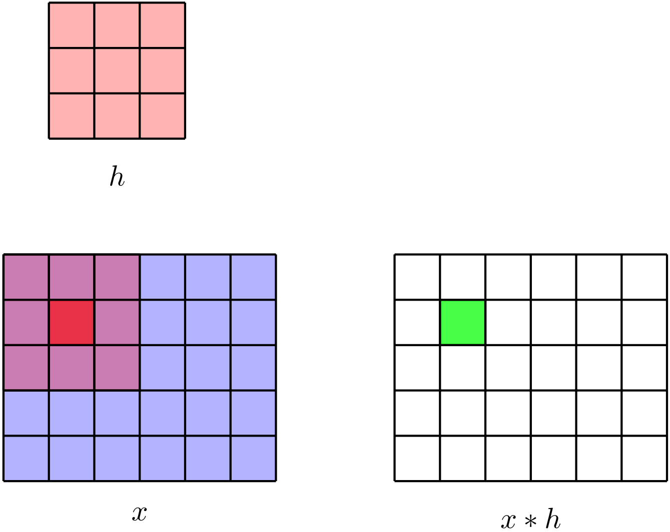
convolution



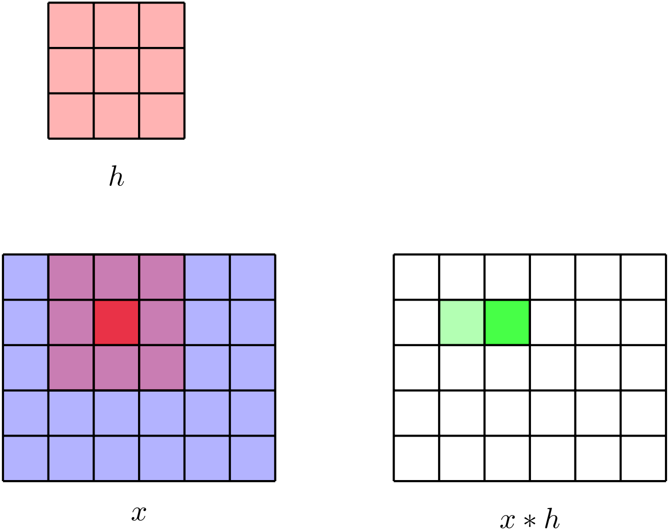
2d convolution



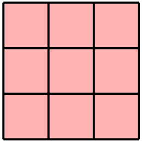
2d convolution



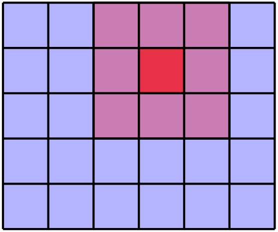
2d convolution



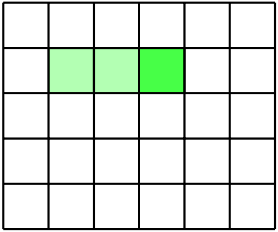
2d convolution



h

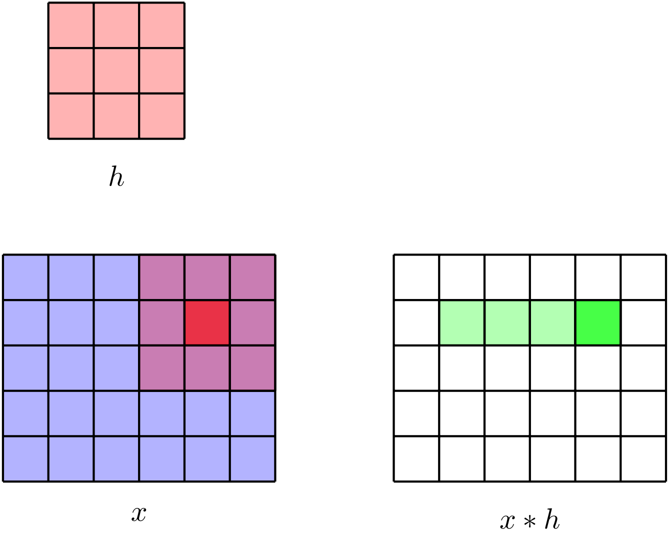


x

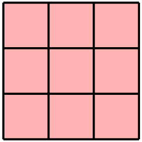


$x * h$

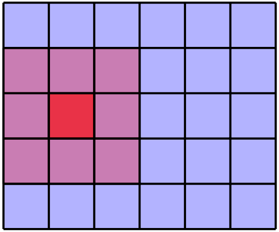
2d convolution



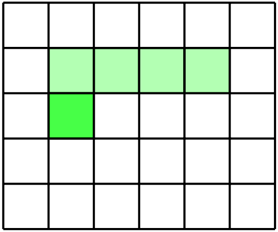
2d convolution



h

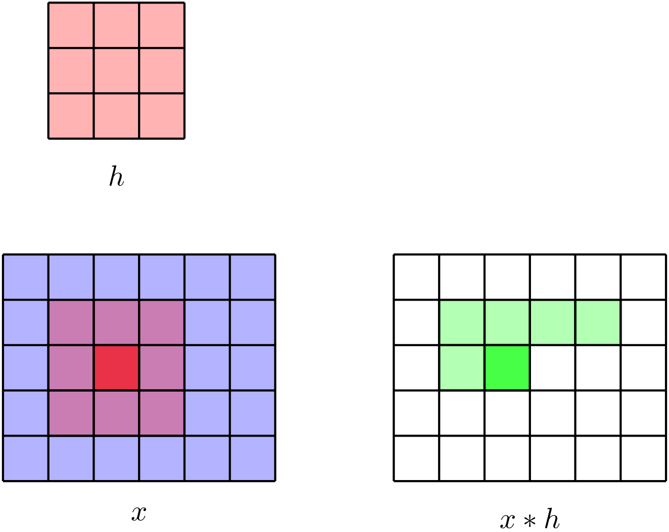


x

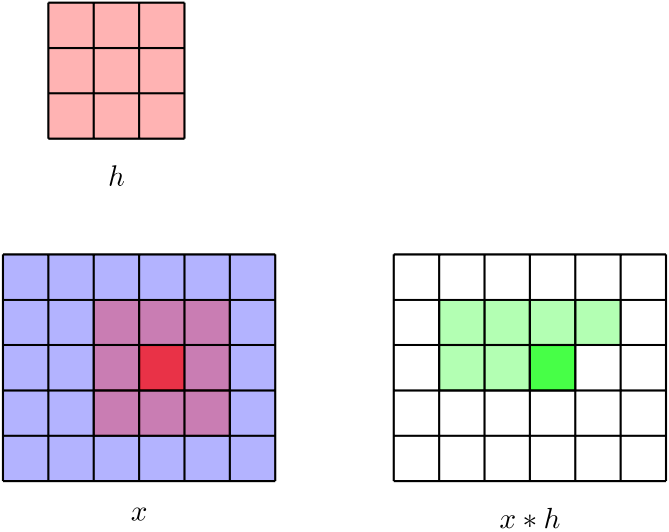


$x * h$

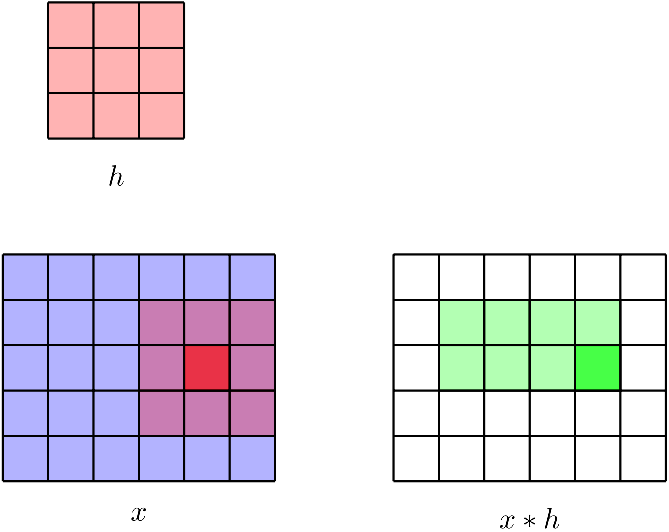
2d convolution



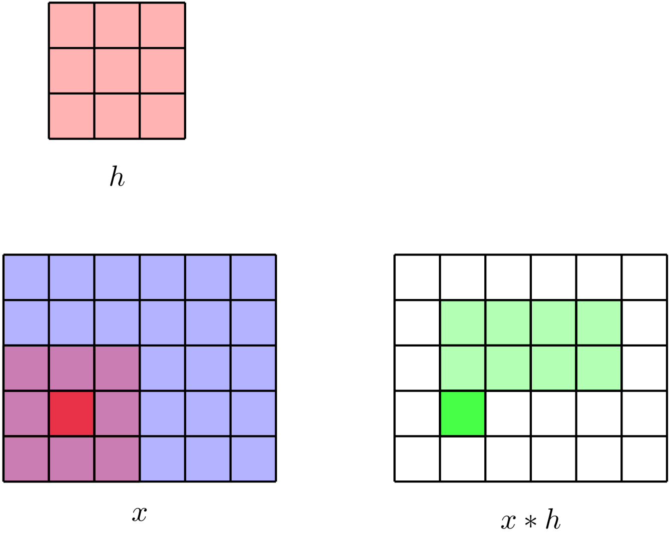
2d convolution



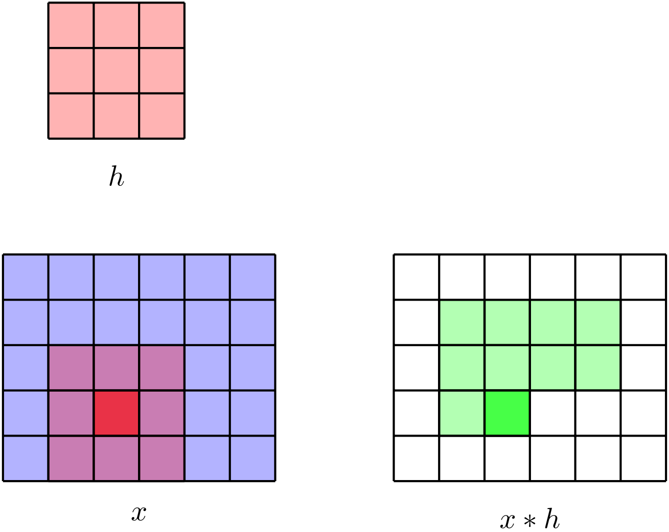
2d convolution



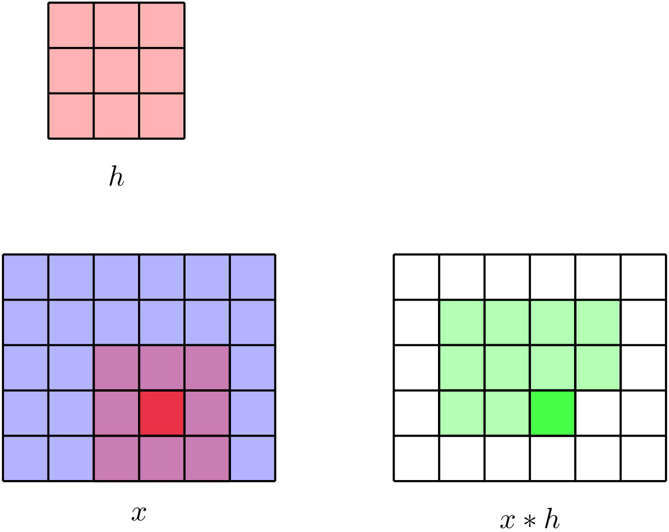
2d convolution



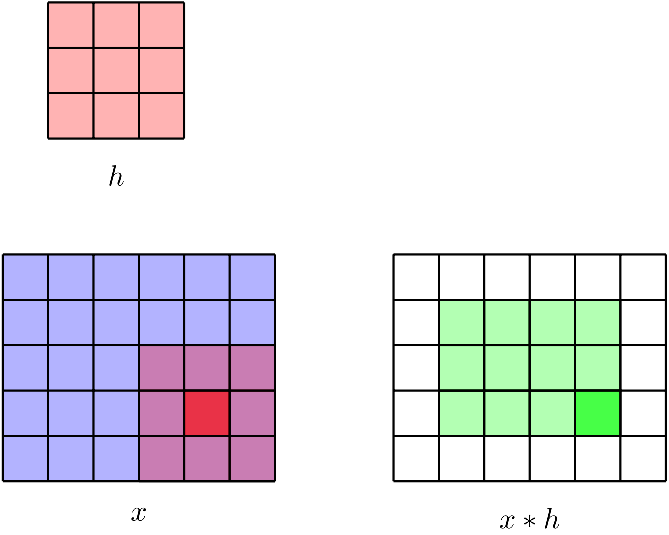
2d convolution



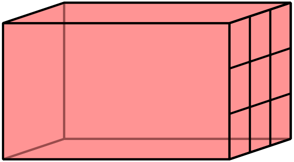
2d convolution



2d convolution

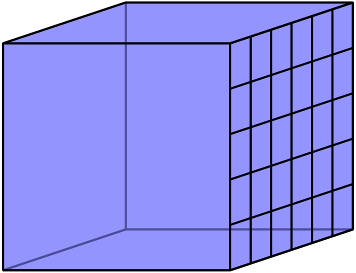


convolution in feature maps

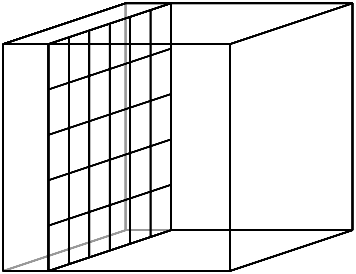


filter 1

filter weights shared
among all spatial positions

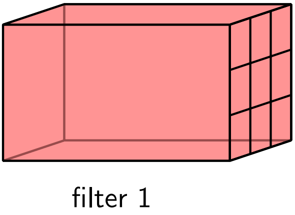


input

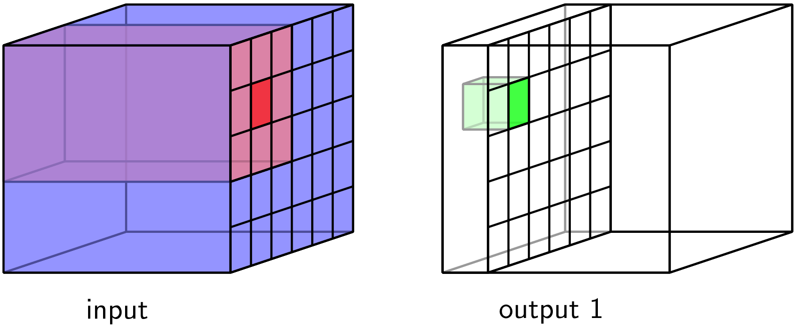


output 1

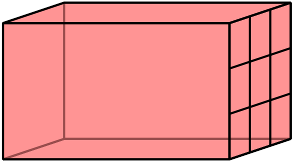
convolution in feature maps



filter weights shared
among all spatial positions

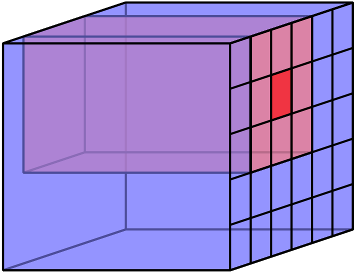


convolution in feature maps

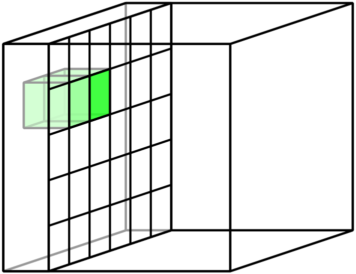


filter 1

filter weights shared
among all spatial positions

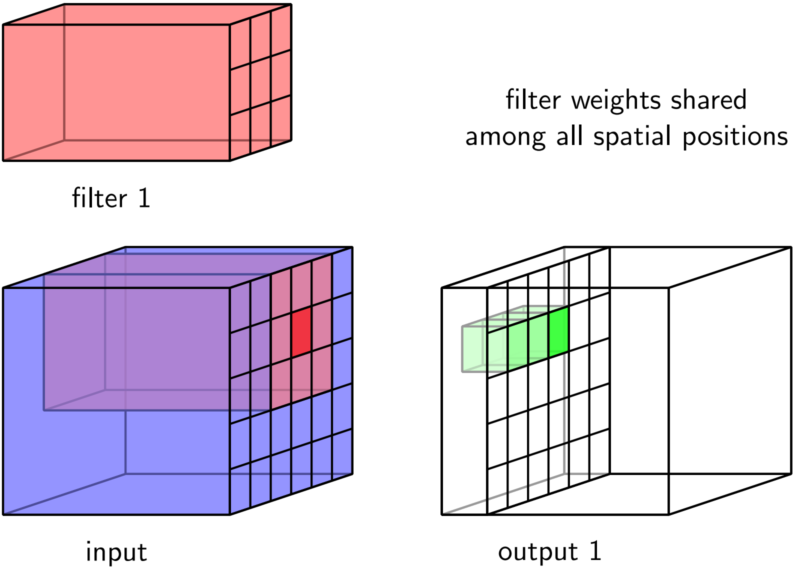


input



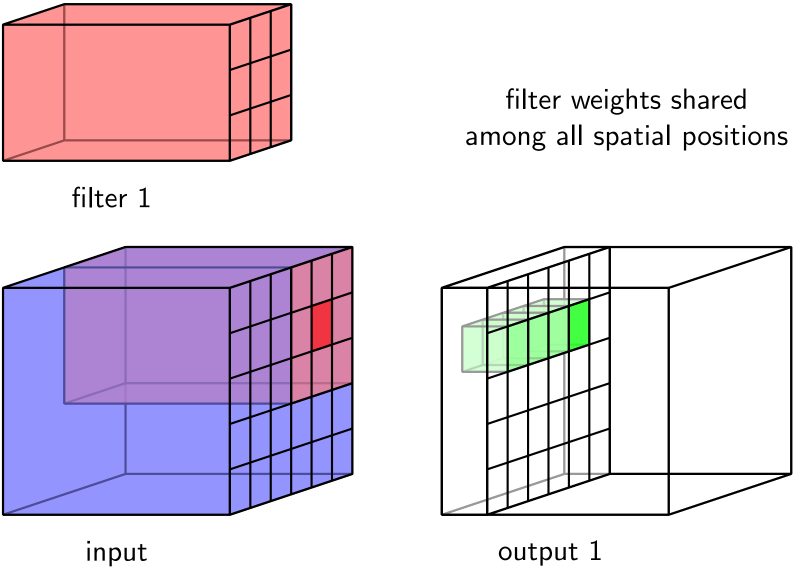
output 1

convolution in feature maps

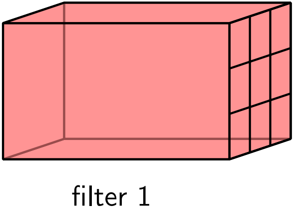


filter weights shared
among all spatial positions

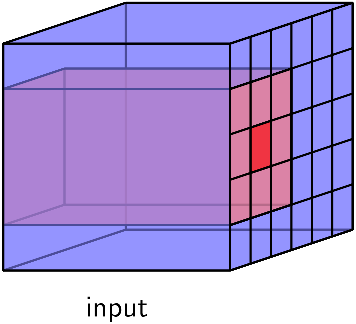
convolution in feature maps

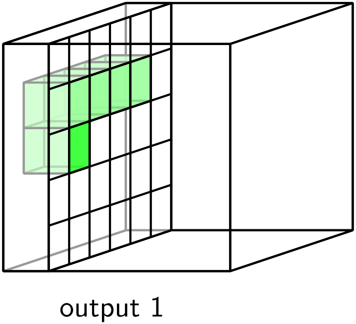


convolution in feature maps

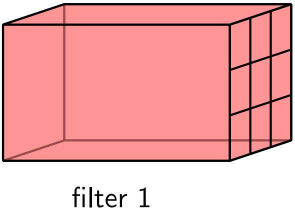


filter weights shared
among all spatial positions

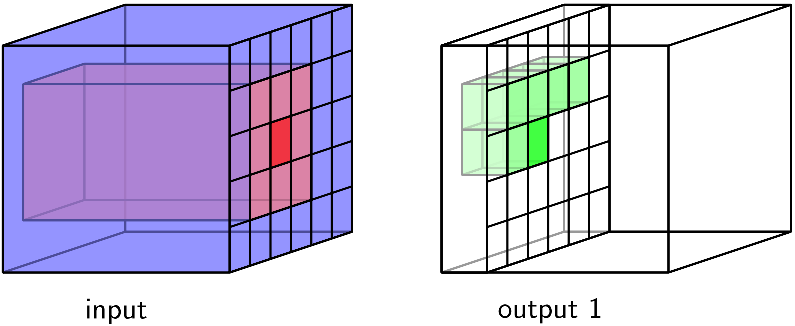




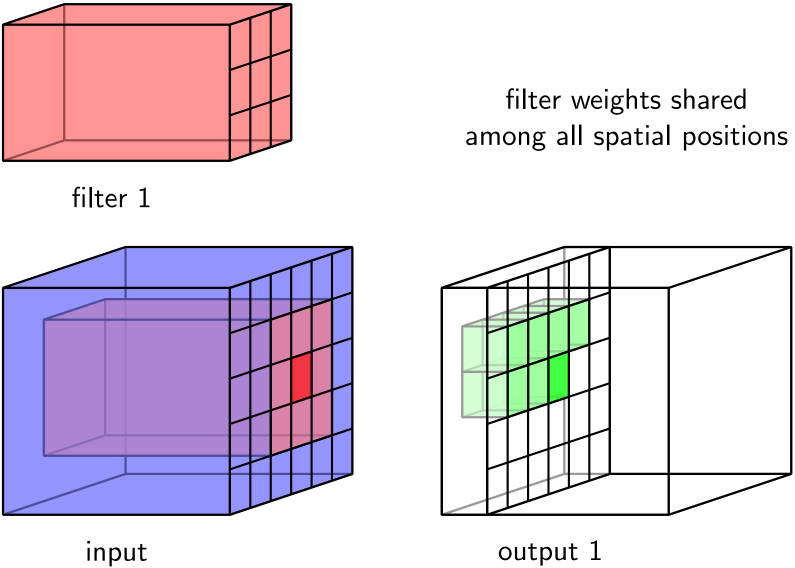
convolution in feature maps



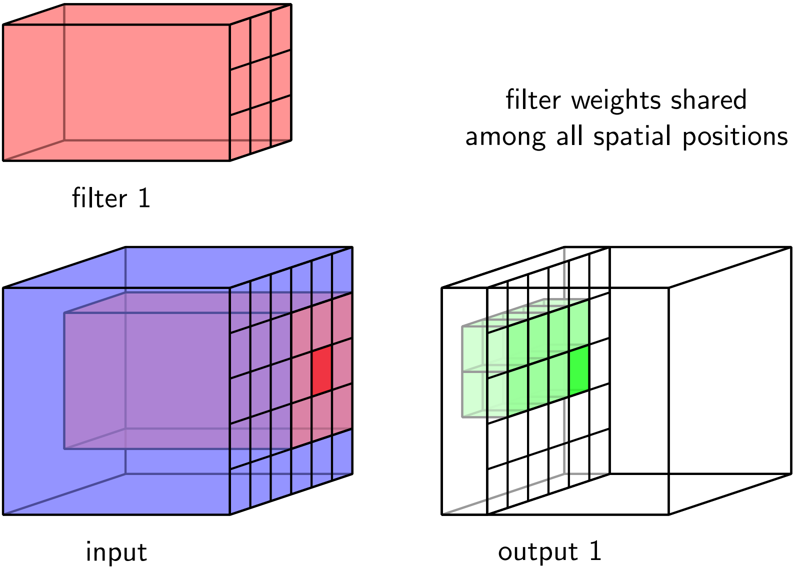
filter weights shared
among all spatial positions



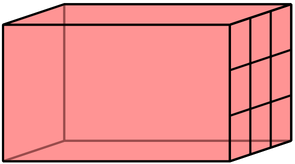
convolution in feature maps



convolution in feature maps

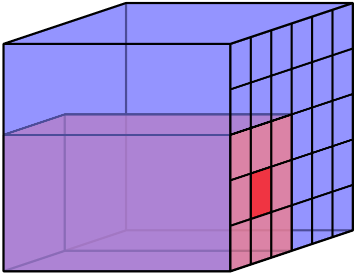


convolution in feature maps

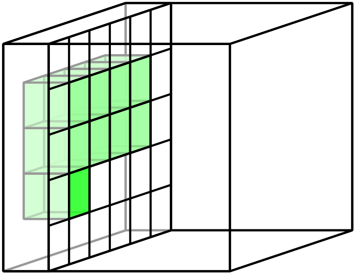


filter 1

filter weights shared
among all spatial positions

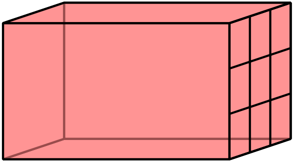


input



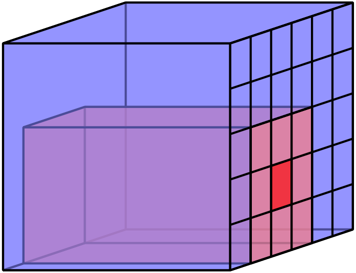
output 1

convolution in feature maps

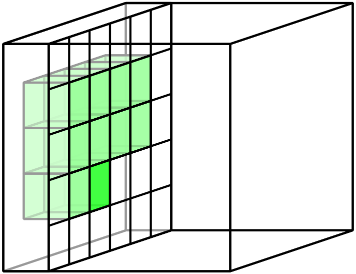


filter 1

filter weights shared
among all spatial positions

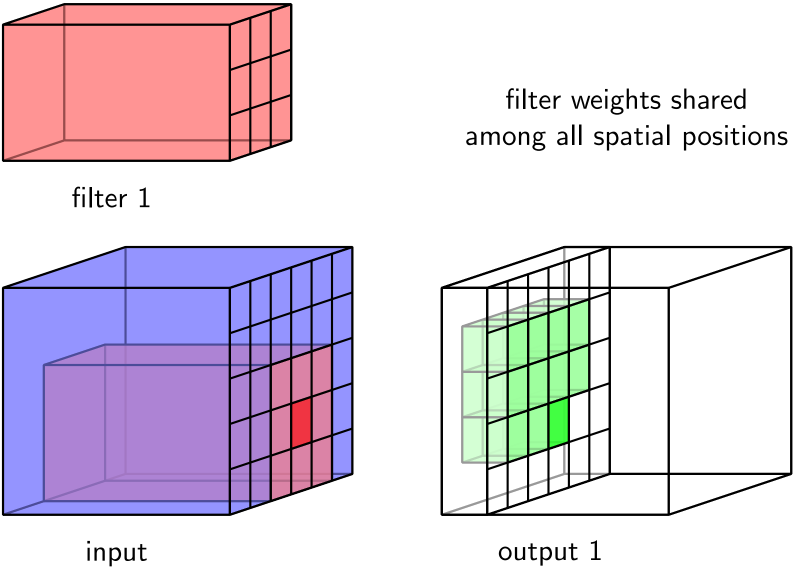


input

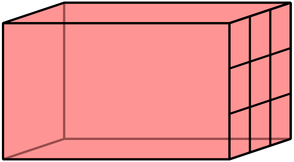


output 1

convolution in feature maps

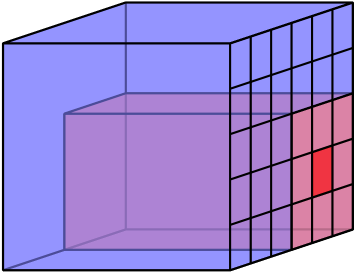


convolution in feature maps

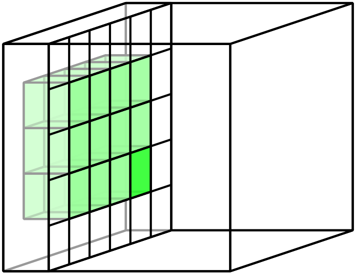


filter 1

filter weights shared
among all spatial positions

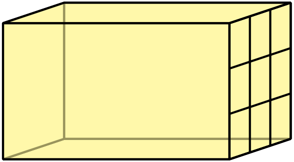


input



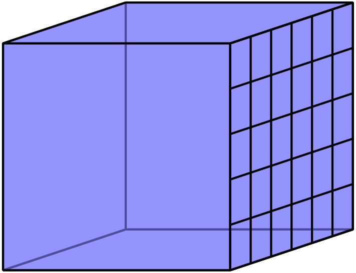
output 1

convolution in feature maps

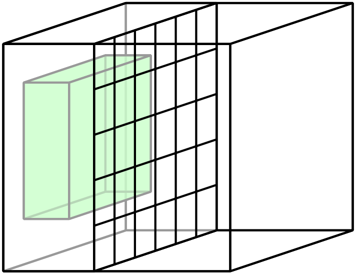


filter 2

new filter, but still shared
among all spatial positions

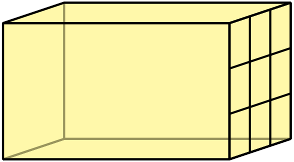


input



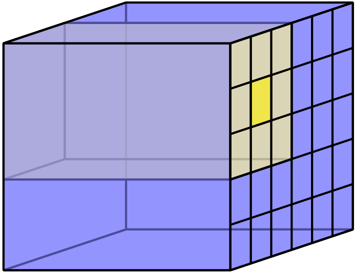
output 2

convolution in feature maps

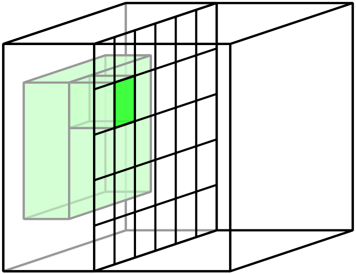


filter 2

new filter, but still shared
among all spatial positions

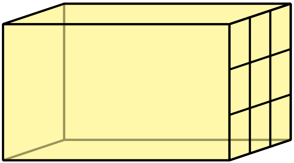


input



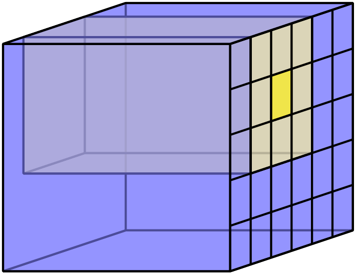
output 2

convolution in feature maps

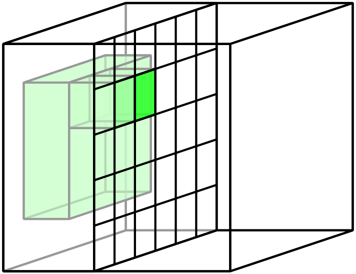


filter 2

new filter, but still shared
among all spatial positions

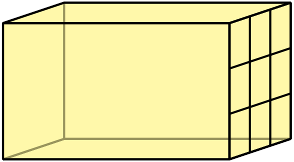


input



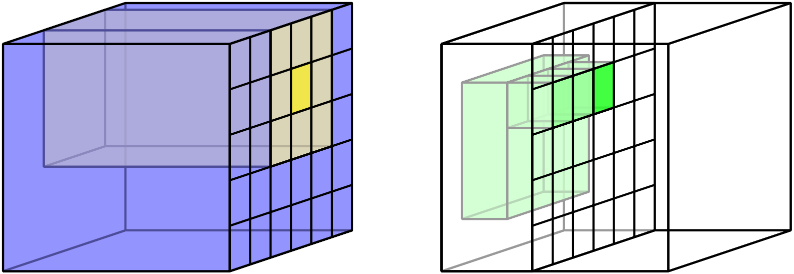
output 2

convolution in feature maps



filter 2

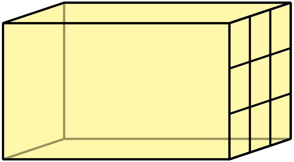
new filter, but still shared
among all spatial positions



input

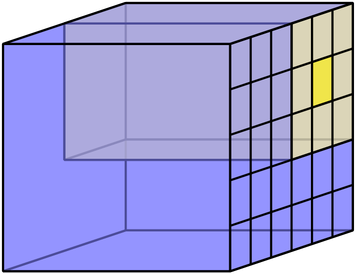
output 2

convolution in feature maps

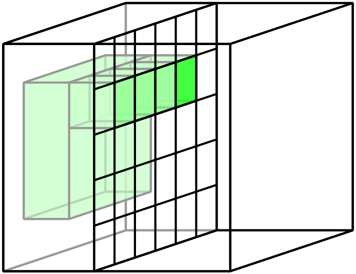


filter 2

new filter, but still shared
among all spatial positions

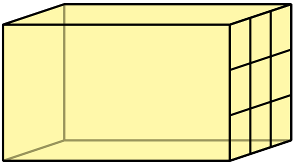


input



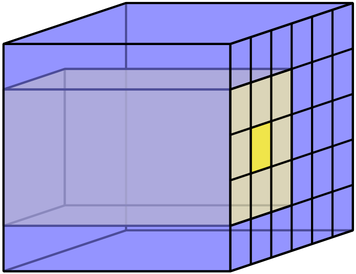
output 2

convolution in feature maps

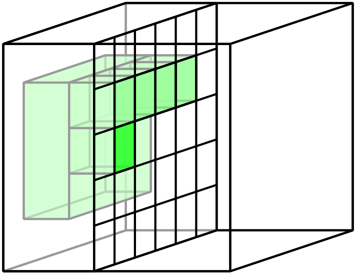


filter 2

new filter, but still shared among all spatial positions

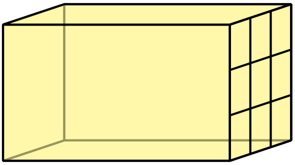


input

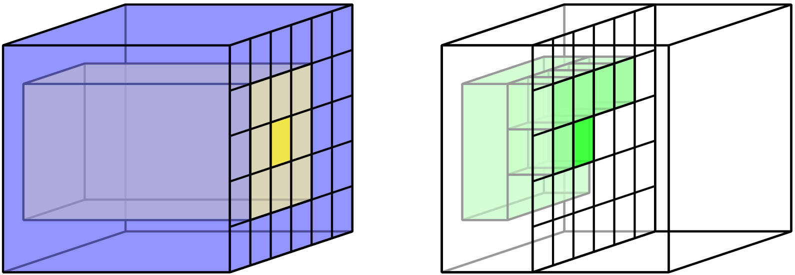


output 2

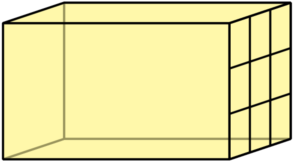
convolution in feature maps



new filter, but still shared
among all spatial positions

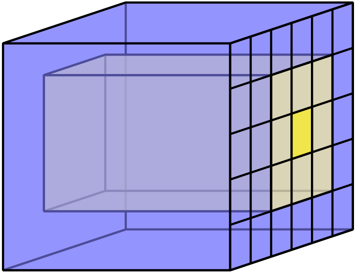


convolution in feature maps

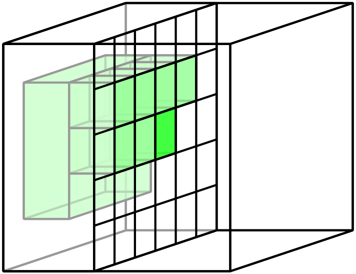


filter 2

new filter, but still shared
among all spatial positions

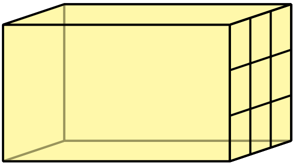


input



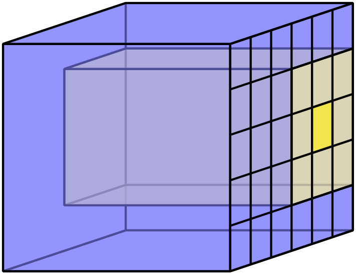
output 2

convolution in feature maps

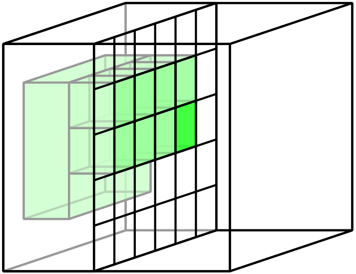


filter 2

new filter, but still shared
among all spatial positions



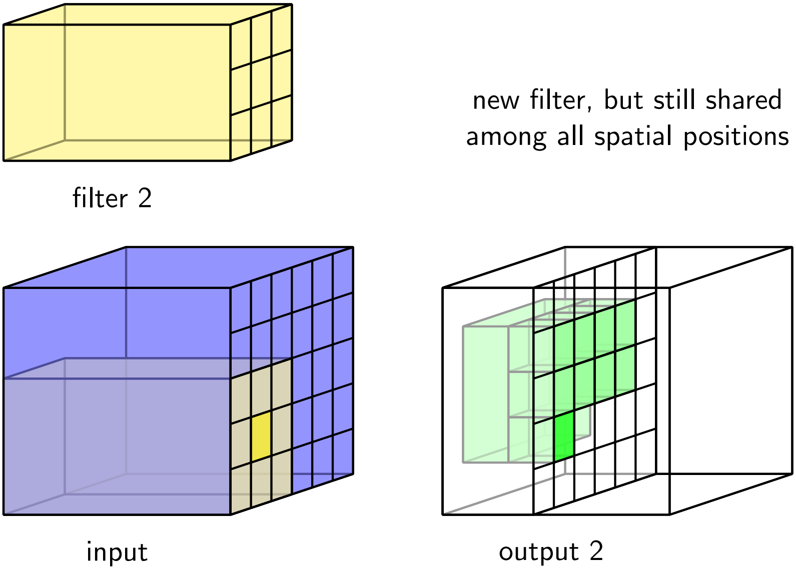
input



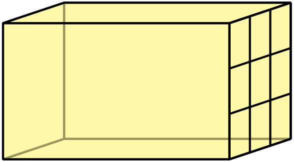
output 2

convolution in feature maps

new filter, but still shared among all spatial positions

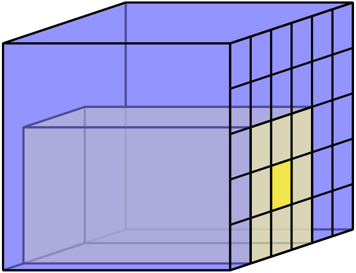


convolution in feature maps

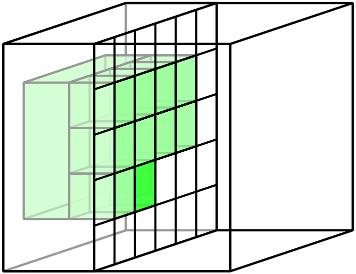


filter 2

new filter, but still shared
among all spatial positions

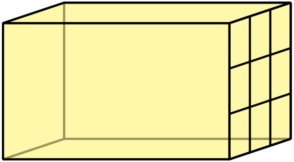


input



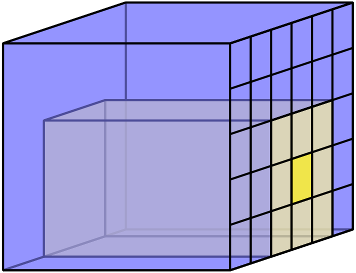
output 2

convolution in feature maps

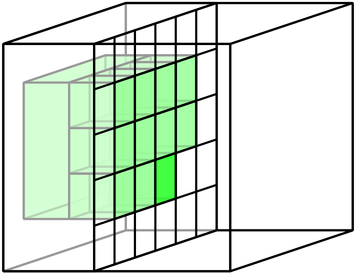


filter 2

new filter, but still shared
among all spatial positions

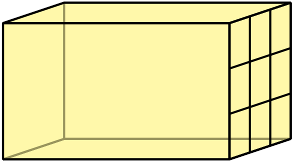


input



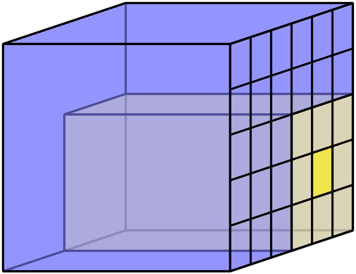
output 2

convolution in feature maps

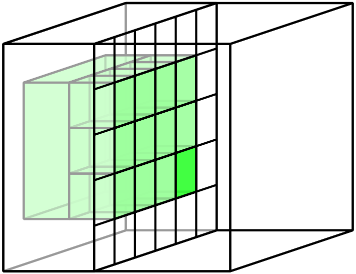


filter 2

new filter, but still shared
among all spatial positions

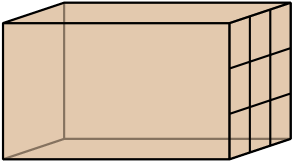


input



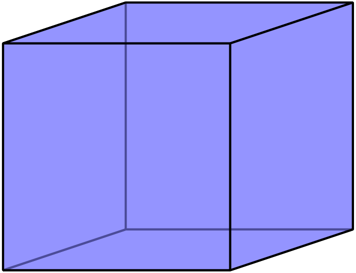
output 2

convolution in feature maps

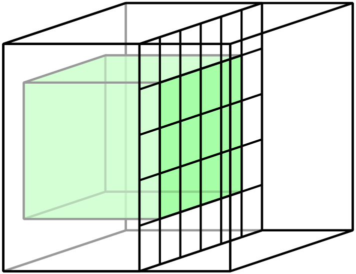


filter 3

different filter for each
output dimension

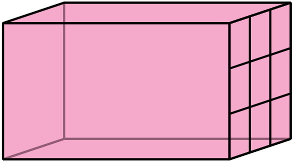


input



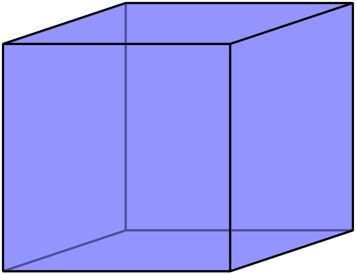
output 3

convolution in feature maps

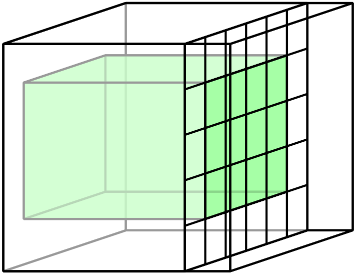


filter 4

different filter for each output dimension

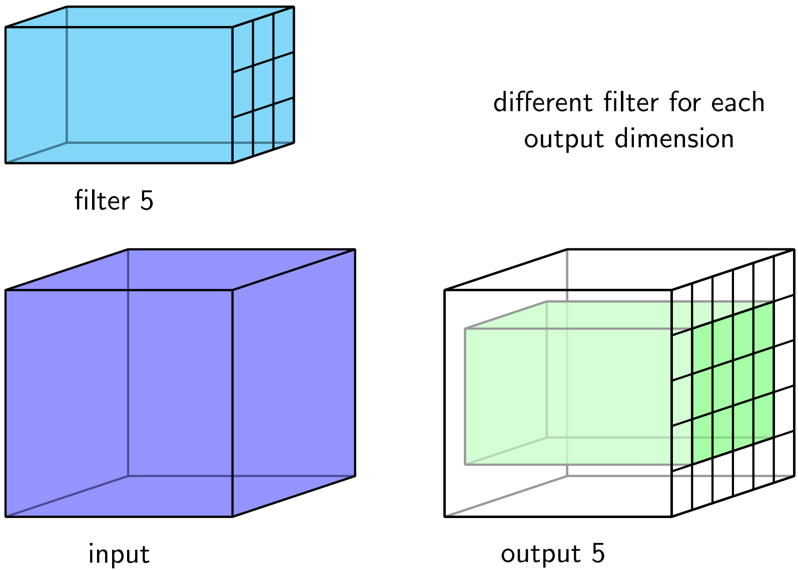


input



output 4

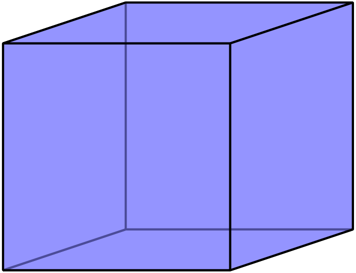
convolution in feature maps



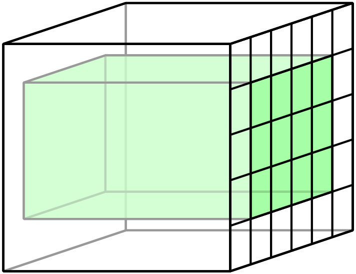
convolution in feature maps

filter 5

1×1 filter is matrix multiplication



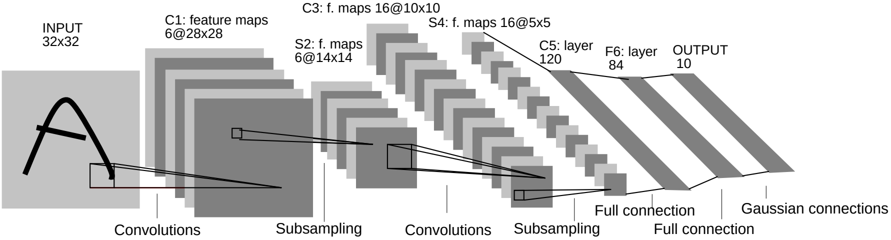
input



output 5

LeNet-5

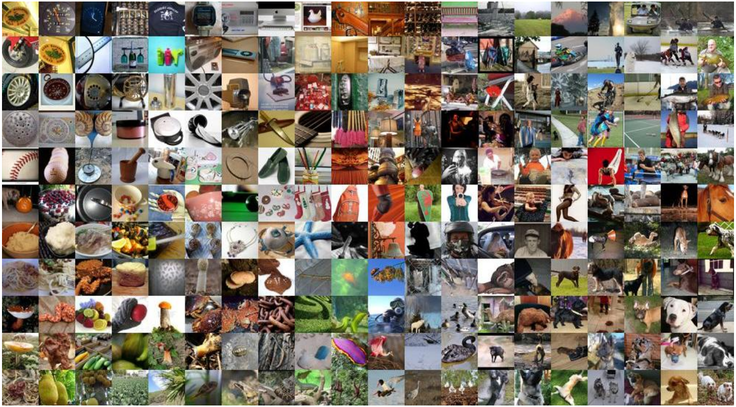
[LeCun et al. 1998]



- sub-sampling gradually introduces translation, scale and distortion invariance
- non-linearity included in sub-sampling layers as feature maps are increasing in dimension

Lecun, Bottou, Bengio, Haffner. IEEE Proc. 1998. Gradient-Based Learning Applied to Document Recognition.

ImageNet

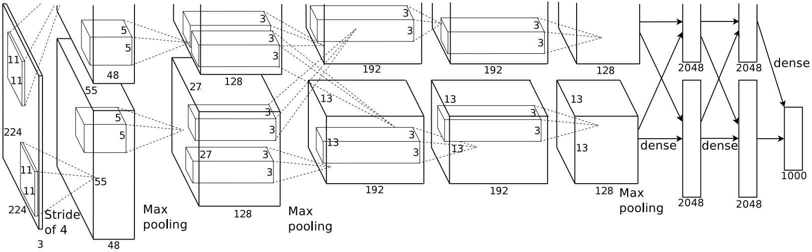


- 22k classes, 15M samples
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1000 classes, 1.2M training images, 50k validation images, 150k test images

Russakovsky, Deng, Su, Krause, *et al.* 2014. Imagenet Large Scale Visual Recognition Challenge.

AlexNet

[Krizhevsky et al. 2012]

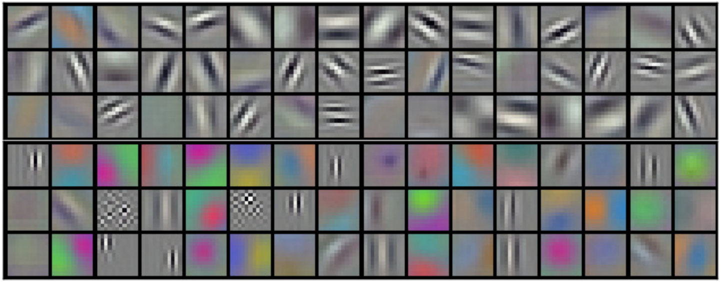


- implementation on two GPUs; connectivity between the two subnetworks is limited
- ReLU, data augmentation, local response normalization, dropout
- outperformed all previous models on ILSVRC by 10%

Krizhevsky, Sutskever, Hinton. NIPS 2012. Imagenet Classification with Deep Convolutional Neural Networks.

learned layer 1 kernels

[Krizhevsky et al. 2012]

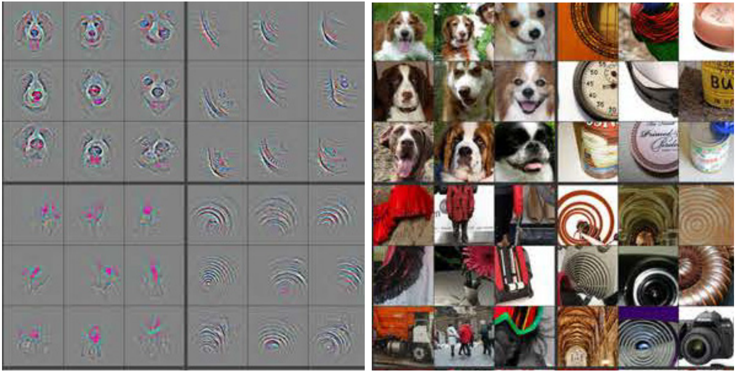


- 96 kernels of size $11 \times 11 \times 3$
- top: 48 GPU 1 kernels; bottom: 48 GPU 2 kernels

Krizhevsky, Sutskever, Hinton. NIPS 2012. Imagenet Classification with Deep Convolutional Neural Networks.

visualizing intermediate layers

[Zeiler and Fergus 2014]

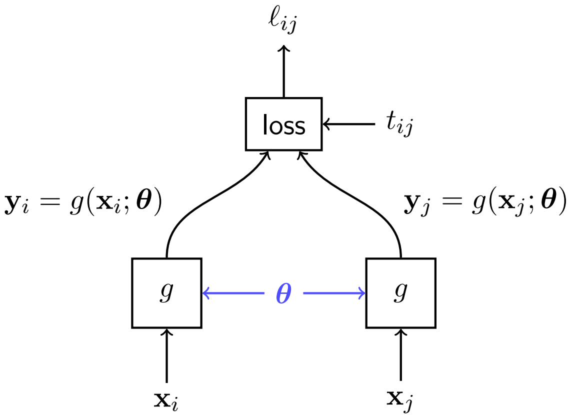


- reconstructed patterns from top 9 activations of selected features of layer 4 and corresponding image patches

unsupervised learning and image retrieval

siamese architecture

[LeCun et al. 2005]



Chopra, Hadsell, Lecun, CVPR 2005. Learning a Similarity Metric Discriminatively, with Application to Face Verification.

manifold learning

[LeCun et al. 2006]

- input samples $\mathbf{x}_i$, output vectors $\mathbf{y}_i = g(\mathbf{x}_i; \boldsymbol{\theta})$
- target variables $t_{ij} = \mathbb{1}[\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- contrastive loss

$$\ell_{ij} = t_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 + (1 - t_{ij})[m - \|\mathbf{y}_i - \mathbf{y}_j\|]_+^2$$

Hadsell, Chopra, Lecun. CVPR 2006. Dimensionality Reduction By Learning an Invariant Mapping.

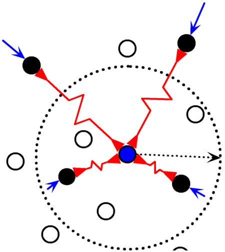
manifold learning

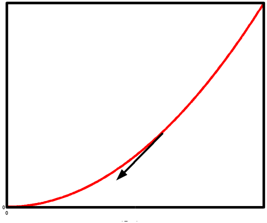
[LeCun et al. 2006]

- input samples $\mathbf{x}_i$, output vectors $\mathbf{y}_i = g(\mathbf{x}_i; \boldsymbol{\theta})$
- target variables $t_{ij} = \mathbb{1}[\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- contrastive loss

$$\ell_{ij} = t_{ij} \boxed{\|\mathbf{y}_i - \mathbf{y}_j\|^2} + (1 - t_{ij})[m - \|\mathbf{y}_i - \mathbf{y}_j\|]_+^2$$

similar





Hadsell, Chopra, Lecun. CVPR 2006. Dimensionality Reduction By Learning an Invariant Mapping.

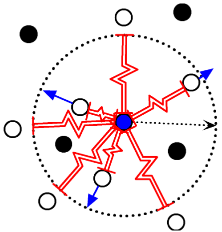
manifold learning

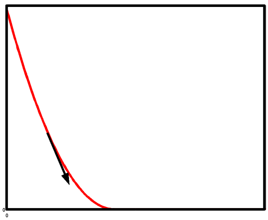
[LeCun et al. 2006]

- input samples $\mathbf{x}_i$, output vectors $\mathbf{y}_i = g(\mathbf{x}_i; \boldsymbol{\theta})$
- target variables $t_{ij} = \mathbb{1}[\mathrm{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- contrastive loss

$$\ell_{ij} = t_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 + (1 - t_{ij})\boxed{[m - \|\mathbf{y}_i - \mathbf{y}_j\|]_+^2}$$

dissimilar

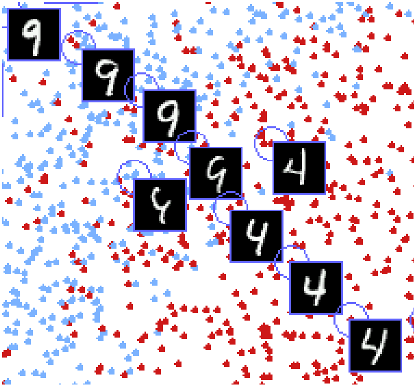


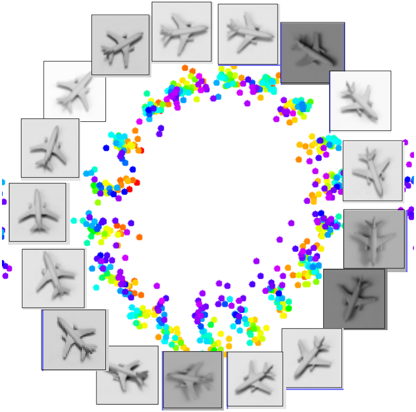


Hadsell, Chopra, Lecun. CVPR 2006. Dimensionality Reduction By Learning an Invariant Mapping.

manifold learning

[LeCun et al. 2006]

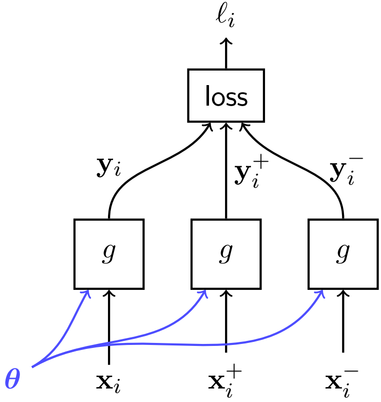




Hadsell, Chopra, Lecun. CVPR 2006. Dimensionality Reduction By Learning an Invariant Mapping.

triplet architecture

[Wang et al. 2014]



Wang, Song, Leung, Rosenberg, Wang, Philbin, Chen, Wu. CVPR 2014. Learning Fine-Grained Image Similarity with Deep Ranking.

learning to rank

[Wang et al. 2014]

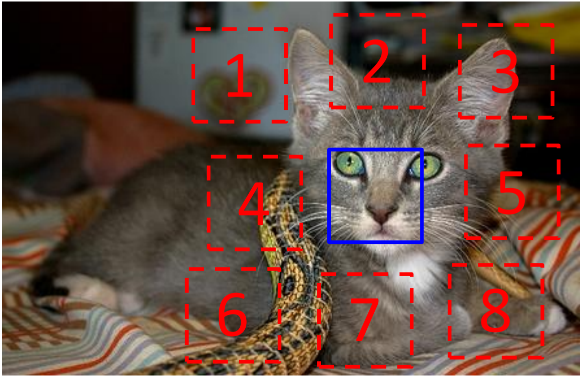
- input "anchor" $\mathbf{x}_i$, output vector $\mathbf{y}_i = g(\mathbf{x}_i; \boldsymbol{\theta})$
- positive $\mathbf{y}_i^+ = g(\mathbf{x}_i^+; \boldsymbol{\theta})$, negative $\mathbf{y}_i^- = g(\mathbf{x}_i^-; \boldsymbol{\theta})$
- triplet loss

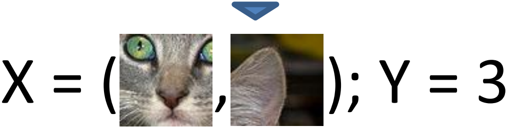
$$\ell_i = \left[m + \|\mathbf{y}_i - \mathbf{y}_i^+\|^2 - \|\mathbf{y}_i - \mathbf{y}_i^-\|^2\right]_+$$

Wang, Song, Leung, Rosenberg, Wang, Philbin, Chen, Wu. CVPR 2014. Learning Fine-Grained Image Similarity with Deep Ranking.

unsupervised learning by solving puzzles

[Doersch et al. 2015]

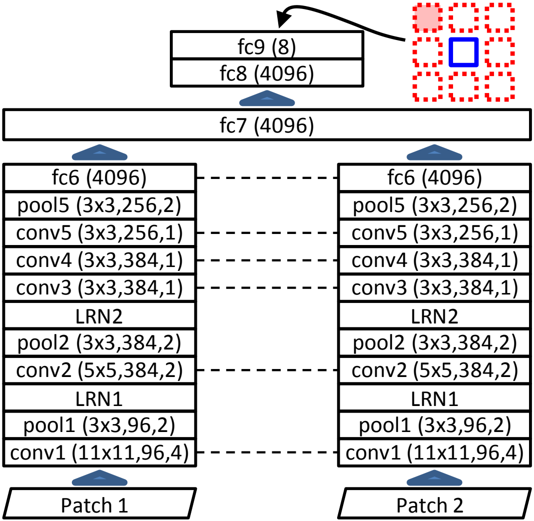




Doersch, Gupta, Efros. ICCV 2015. Unsupervised Visual Representation Learning By Context Prediction.

unsupervised learning by solving puzzles

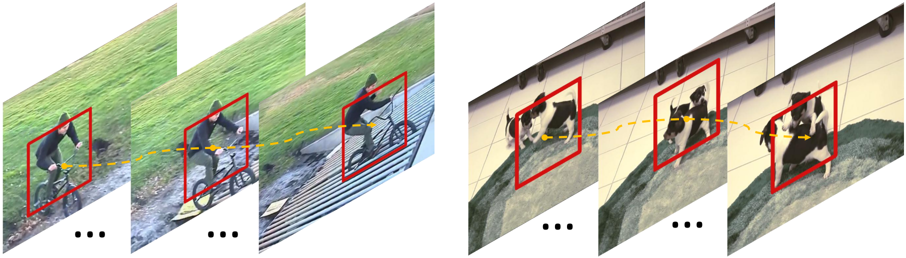
[Doersch et al. 2015]



Doersch, Gupta, Efros. ICCV 2015. Unsupervised Visual Representation Learning By Context Prediction.

unsupervised learning by watching video

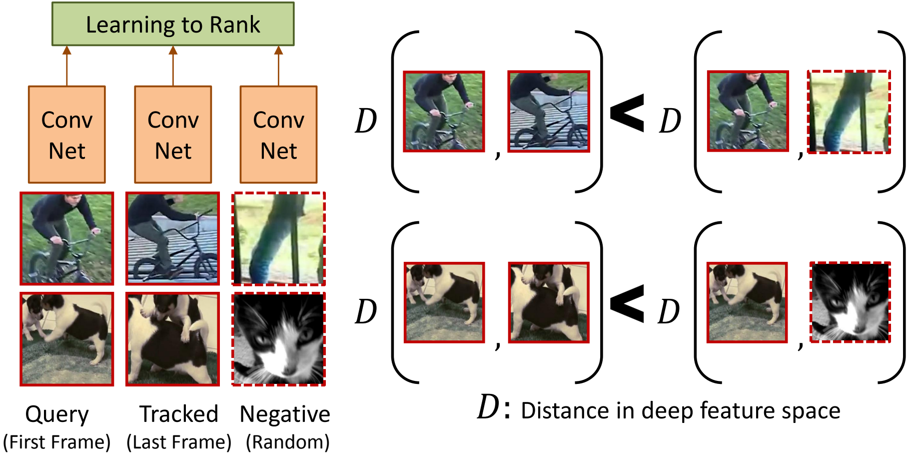
[Wang et al. 2015]



Wang and Gupta. ICCV 2015. Unsupervised Learning of Visual Representations Using Videos.

unsupervised learning by watching video

[Wang et al. 2015]



Wang and Gupta. ICCV 2015. Unsupervised Learning of Visual Representations Using Videos.

unsupervised learning by watching video

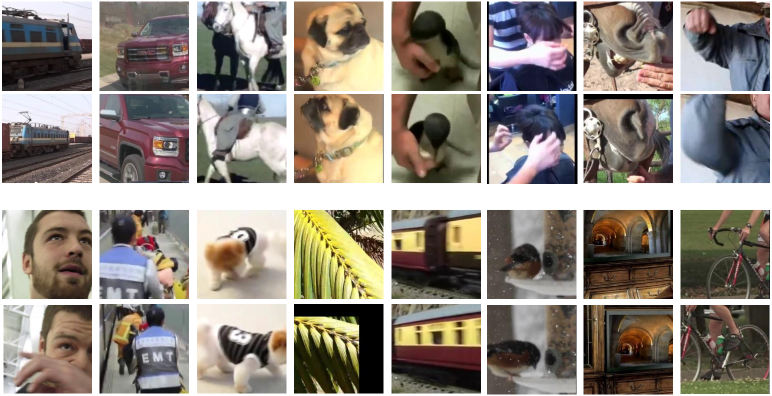
[Wang et al. 2015]

Query (First Frame)

Tracked (Last Frame)

Query (First Frame)

Tracked (Last Frame)



Wang and Gupta. ICCV 2015. Unsupervised Learning of Visual Representations Using Videos.

ranking by CNN features

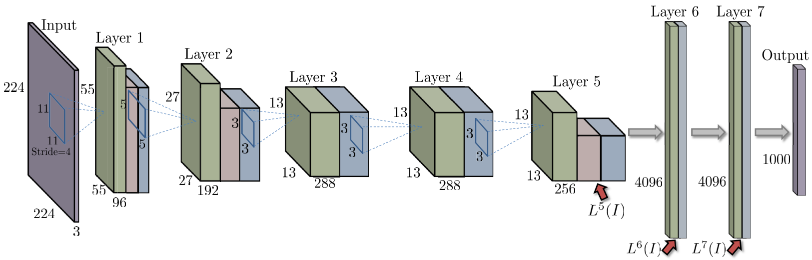
[Krizhevsky et al. 2012]

- use the last fully-connected layer features

Krizhevsky, Sutskever, Hinton. NIPS 2012. Imagenet Classification with Deep Convolutional Neural Networks.

neural codes

[Babenko et al. 2014]



- investigate more than the last fully-connected layer
- fine-tune by softmax on 672 classes of 200k landmark photos

Babenko, Slesarev, Chigorin, Lempitsky. ECCV 2014. Neural Codes for Image Retrieval.

neural codes

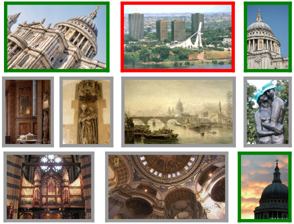
[Babenko et al. 2014]

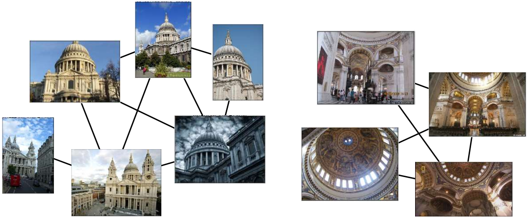
- investigate more than the last fully-connected layer
- fine-tune by softmax on 672 classes of 200k landmark photos

Babenko, Slesarev, Chigorin, Lempitsky. ECCV 2014. Neural Codes for Image Retrieval.

fine-tuning

[Gordo et al. 2016]



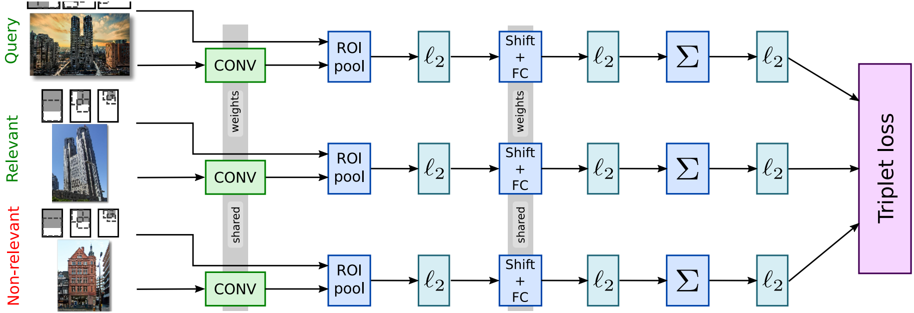


- clean landmark images by pairwise matching
- fine-tune by triplet architecture and regional max-pooling (R-MAC)

Gordo, Almazan, Revaud, Larlus. ECCV 2016. Deep Image Retrieval: Learning Global Representations for Image Search.

fine-tuning

[Gordo et al. 2016]



- clean landmark images by pairwise matching
- fine-tune by triplet architecture and regional max-pooling (R-MAC)

Gordo, Almazan, Revaud, Larlus. ECCV 2016. Deep Image Retrieval: Learning Global Representations for Image Search.

unsupervised fine-tuning

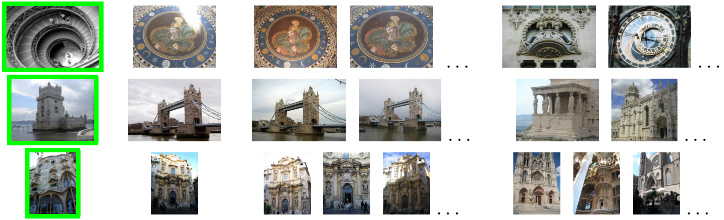
[Radenovic et al. 2016]

(positive)

- reconstruct 700 3d models with 160k images by SfM on 7M images
- fine-tune by siamese architecture and global max-pooling (MAC)

unsupervised fine-tuning

[Radenovic et al. 2016]



(negative)

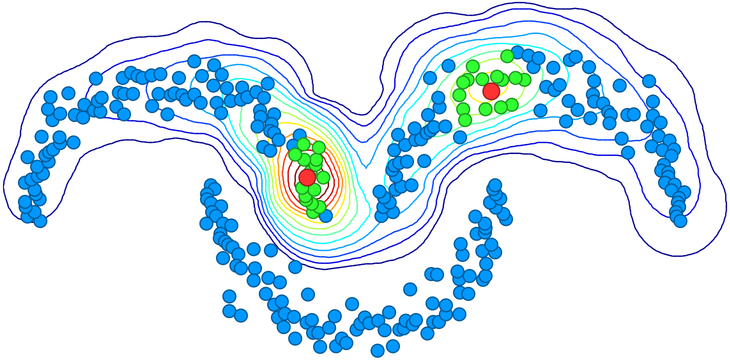
- reconstruct 700 3d models with 160k images by SfM on 7M images
- fine-tune by siamese architecture and global max-pooling (MAC)

Radenovic, Tolias, Chum. ECCV 2016. CNN Image Retrieval Learns From BoW: Unsupervised Fine-Tuning with Hard Examples.

graph-based methods

query expansion and searching on manifolds

[Iscen et al. 2017]

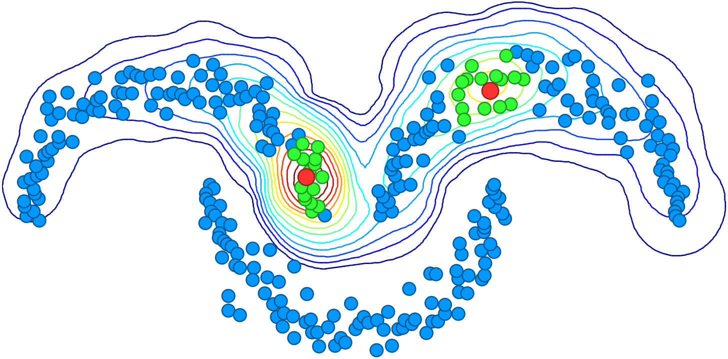


- now that images are represented by a global descriptor or just a few regional descriptors, graph methods are more applicable than ever

Iscen, Tolias, Avrithis, Furon, Chum. CVPR 2017. Efficient Diffusion on Region Manifolds: Recovering Small Objects With Compact CNN Representations.

query expansion and searching on manifolds

[Iscen et al. 2017]



- now that images are represented by a global descriptor or just a few regional descriptors, graph methods are more applicable than ever

Iscen, Tolias, Avrithis, Furon, Chum. CVPR 2017. Efficient Diffusion on Region Manifolds: Recovering Small Objects With Compact CNN Representations.

query expansion as a linear system

[Iscen et al. 2017]

- reciprocal nearest neighbor graph on images or regions
- symmetrically normalized adjacency matrix $\mathcal{W}$
- regularized Laplacian

$$\mathcal{L}_\alpha = \frac{I - \alpha\mathcal{W}}{1 - \alpha}$$

- initial query: sparse observation vector
  $y_i = \mathbb{1}[i \text{ is query (or neighbor)}]$
- query expansion: solve linear system

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

Iscen, Tolias, Avrithis, Furon, Chum. CVPR 2017. Efficient Diffusion on Region Manifolds: Recovering Small Objects With Compact CNN Representations.

query expansion as a linear system

[Iscen et al. 2017]

- reciprocal nearest neighbor graph on images or regions
- symmetrically normalized adjacency matrix $\mathcal{W}$
- regularized Laplacian

$$\mathcal{L}_\alpha = \frac{I - \alpha \mathcal{W}}{1 - \alpha}$$

- initial query: sparse observation vector
  $y_i = \mathbb{1}[i \text{ is query (or neighbor)}]$
- query expansion: solve linear system

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

Iscen, Tolias, Avrithis, Furon, Chum. CVPR 2017. Efficient Diffusion on Region Manifolds: Recovering Small Objects With Compact CNN Representations.

query expansion as a linear system

[Iscen et al. 2017]

- reciprocal nearest neighbor graph on images or regions
- symmetrically normalized adjacency matrix $\mathcal{W}$
- regularized Laplacian

$$\mathcal{L}_\alpha = \frac{I - \alpha\mathcal{W}}{1 - \alpha}$$

- initial query: sparse observation vector
  $y_i = \mathbb{1}[i \text{ is query (or neighbor)}]$
- query expansion: solve linear system

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

Iscen, Tolias, Avrithis, Furon, Chum. CVPR 2017. Efficient Diffusion on Region Manifolds: Recovering Small Objects With Compact CNN Representations.

searching on manifolds as smoothing

[Iscen et al. 2017]

- express $\mathcal{L}_\alpha^{-1}$ using a transfer function

$$\mathcal{L}_\alpha^{-1} = h_\alpha(\mathcal{W}) = (1-\alpha)(I - \alpha\mathcal{W})^{-1}$$

- given any matrix function h, we want to compute

$$\mathbf{x} = h(\mathcal{W})\mathbf{y}$$

without computing $h(\mathcal{W})$

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing

[Iscen et al. 2017]

- express $\mathcal{L}_\alpha^{-1}$ using a transfer function

$$\mathcal{L}_\alpha^{-1} = h_\alpha(\mathcal{W}) = (1-\alpha)(I - \alpha\mathcal{W})^{-1}$$

- given any matrix function h, we want to compute

$$\mathbf{x} = h(\mathcal{W})\mathbf{y}$$

without computing $h(\mathcal{W})$

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing

$$\boxed{\mathbf{x}} = h\left(\boxed{\mathcal{W}}\right)\boxed{\mathbf{y}}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing

$$\mathbf{x} = h\left(U \Lambda U^\top \right) \mathbf{y}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

searching on manifolds as smoothing

$$\mathbf{x} \approx h\left(U \; \Lambda \; U^\top \right) \mathbf{y}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing

$$\mathbf{x} \approx U \, h\left(\Lambda \right) U^\top \mathbf{y}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing

$$\mathbf{x} \approx U \, h\left(\underbrace{\Lambda}_{\text{diagonal}}\right) U^\top \underbrace{\mathbf{y}}_{\text{sparse}}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

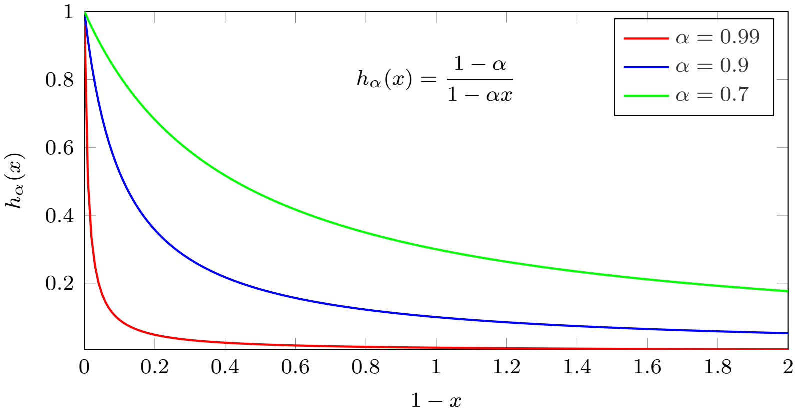
searching on manifolds as smoothing

$$\mathbf{x} \approx \mathcal{F}^{-1} \, h \left(\Lambda \right) \mathcal{F} \, \mathbf{y}$$

- eigenvalue decomposition of $\mathcal{W}$
- low-rank approximation
- (under conditions on h and Λ)
- dot-product search
- linear graph filter in frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

searching on manifolds as smoothing



- low-pass filtering in the frequency domain

Iscen, Tolias, Avrithis, Furon, Chum. arXiv 2017. Fast Spectral Ranking for Similarity Search.

unsupervised object discovery

[Siméoni et al. 2016]

Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

[Siméoni et al. 2016]

Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

[Siméoni et al. 2016]

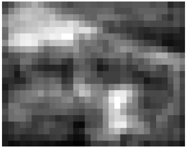
dataset

Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

[Siméoni et al. 2016]

dataset

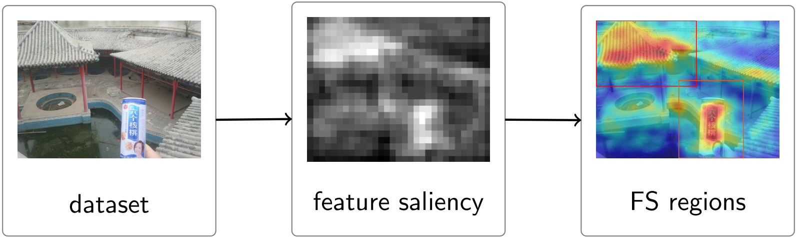


feature saliency

Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

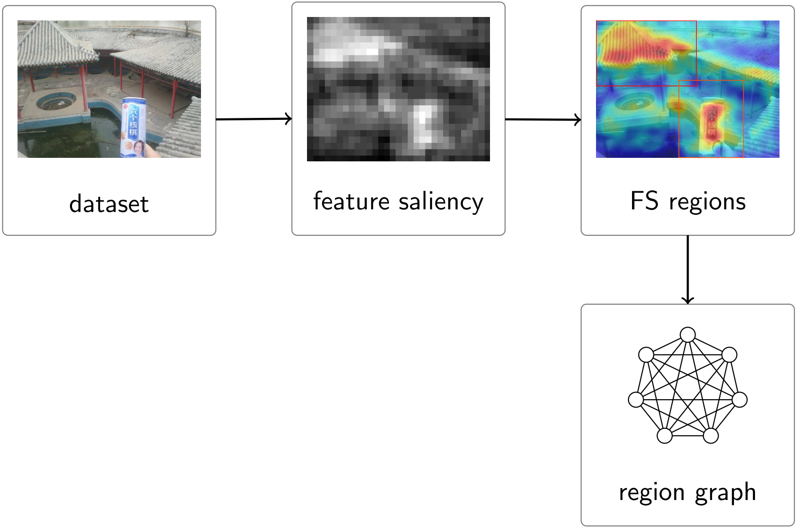
[Siméoni et al. 2016]



Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

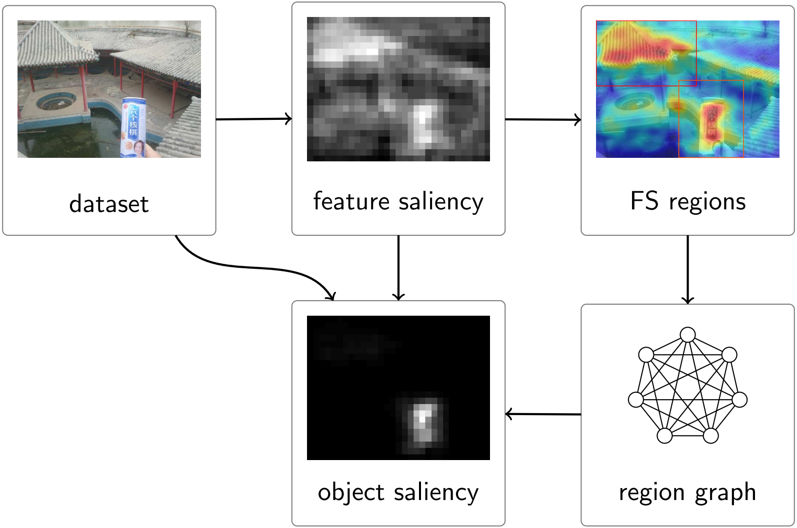
[Siméoni et al. 2016]



Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

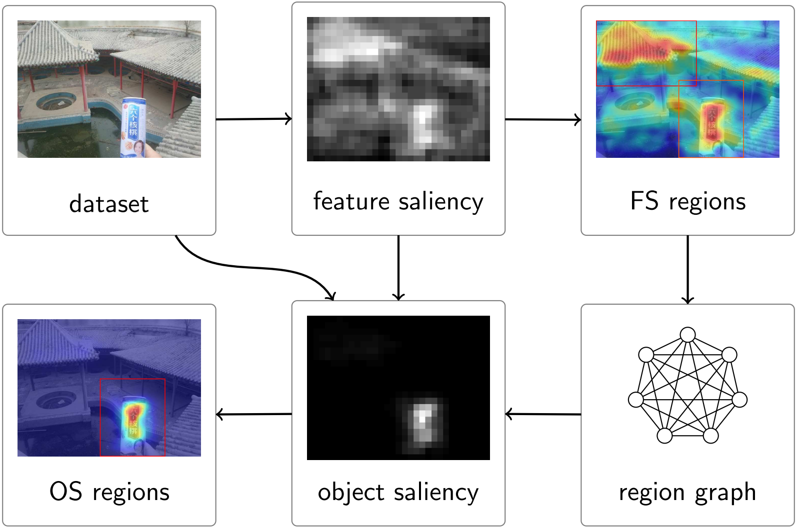
[Siméoni et al. 2016]



Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

unsupervised object discovery

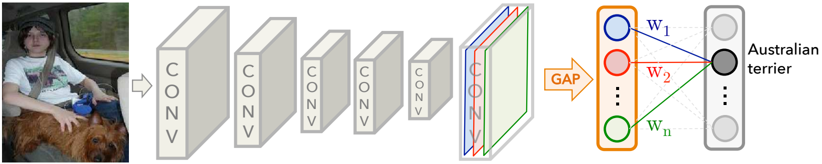
[Siméoni et al. 2016]



Siméoni, Iscen, Tolias, Avrithis, Chum. arXiv 2017. Unsupervised deep object discovery for instance recognition.

class activation mapping (CAM)

[Zhou et al. 2016]



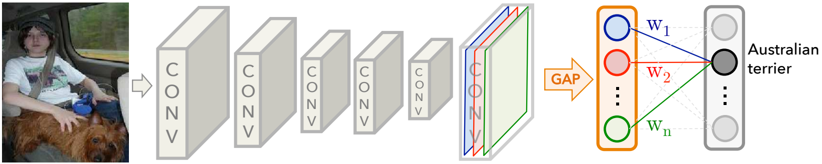
- global average pooling

$$S_c = \sum_k w_k^c \sum_{x,y} A_k(x, y)$$

Zhou, Khosla, Lapedriza, Oliva, Torralba. CVPR 2016. Learning Deep Features for Discriminative Localization.

class activation mapping (CAM)

[Zhou et al. 2016]



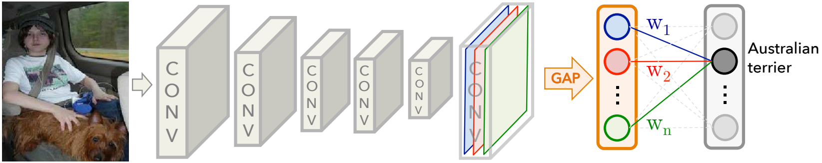
- global average pooling

$$S_c = \sum_k w_k^c \sum_{x,y} A_k(x,y) = \sum_{x,y} \sum_k w_k^c A_k(x,y)$$

Zhou, Khosla, Lapedriza, Oliva, Torralba. CVPR 2016. Learning Deep Features for Discriminative Localization.

class activation mapping (CAM)

[Zhou et al. 2016]



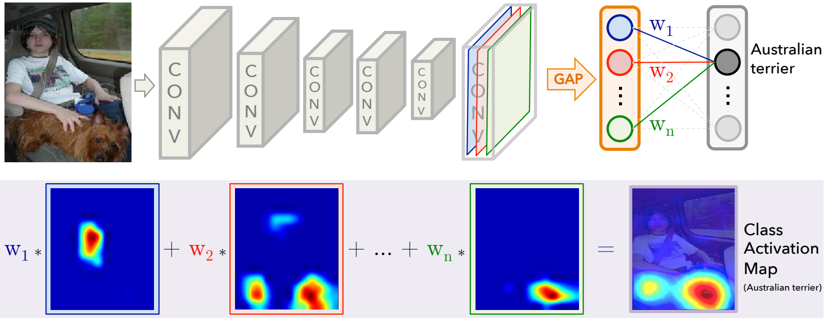
- global average pooling

$$S_c = \sum_k w_k^c \sum_{x,y} A_k(x,y) = \sum_{x,y} \boxed{\sum_k w_k^c A_k(x,y)} = \sum_{x,y} \boxed{M_c(x,y)}$$

Zhou, Khosla, Lapedriza, Oliva, Torralba. CVPR 2016. Learning Deep Features for Discriminative Localization.

class activation mapping (CAM)

[Zhou et al. 2016]

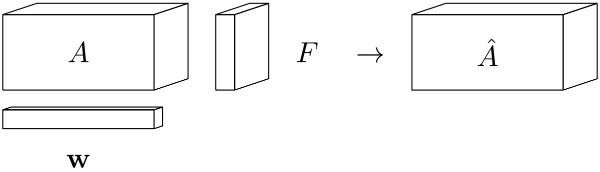


- global average pooling

$$S_c = \sum_k w_k^c \sum_{x,y} A_k(x,y) = \sum_{x,y} \boxed{\sum_k w_k^c A_k(x,y)} = \sum_{x,y} \boxed{M_c(x,y)}$$

cross-dimensional weighting (CroW)

[Kalantidis et al. 2016]



- spatial weights (visual saliency)

$$F(x,y) = \sum_k A_k(x,y)$$

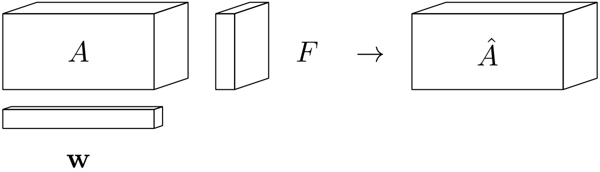
- channel weights (sparsity sensitive)

$$w_k = -\log\left(\epsilon + \sum_{x,y} \mathbb{1}[A_k(x,y)]\right)$$

Kalantidis, Mellina, Osindero. ECCVW 2016. Cross-Dimensional Weighting for Aggregated Deep Convolutional Features.

cross-dimensional weighting (CroW)

[Kalantidis et al. 2016]



- spatial weights (visual saliency)

$$F(x, y) = \sum_k A_k(x, y)$$

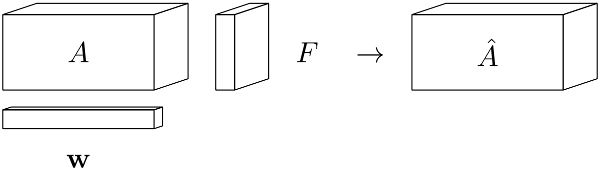
- channel weights (sparsity sensitive)

$$w_k = -\log\left(\epsilon + \sum_{x,y} \mathbb{1}[A_k(x, y)]\right)$$

Kalantidis, Mellina, Osindero. ECCVW 2016. Cross-Dimensional Weighting for Aggregated Deep Convolutional Features.

cross-dimensional weighting (CroW)

[Kalantidis et al. 2016]



- spatial weights (visual saliency)

$$F(x,y) = \sum_k A_k(x,y)$$

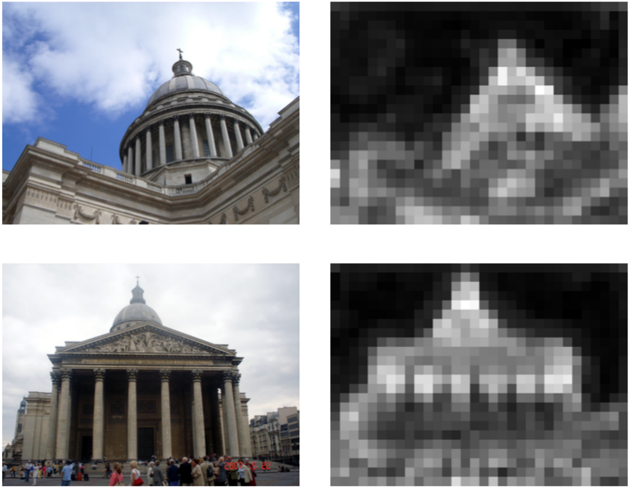
- channel weights (sparsity sensitive)

$$w_k = -\log\left(\epsilon + \sum_{x,y} \mathbb{1}[A_k(x,y)]\right)$$

Kalantidis, Mellina, Osindero. ECCVW 2016. Cross-Dimensional Weighting for Aggregated Deep Convolutional Features.

cross-dimensional weighting (CroW)

[Kalantidis et al. 2016]



Kalantidis, Mellina, Osindero. ECCVW 2016. Cross-Dimensional Weighting for Aggregated Deep Convolutional Features.

feature saliency (FS) map

- channel weights (sparsity sensitive, as in CroW)

$$w_k = -\log\left(\epsilon + \sum_{x,y} \mathbb{1}[A_k(x,y)]\right)$$

- feature saliency map (as in CAM)

$$F(x,y) = \sum_k w_k A_k(x,y)$$

feature saliency (FS) map

- channel weights (sparsity sensitive, as in CroW)

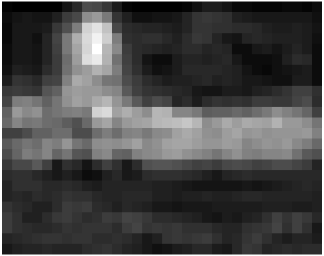
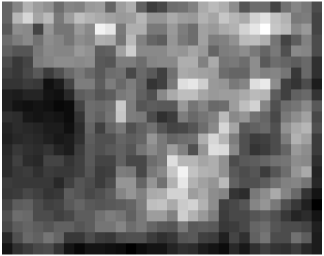
$$w_k = -\log \left(\epsilon + \sum_{x,y} \mathbb{1}[A_k(x,y)] \right)$$

- feature saliency map (as in CAM)

$$F(x,y) = \sum_k w_k A_k(x,y)$$

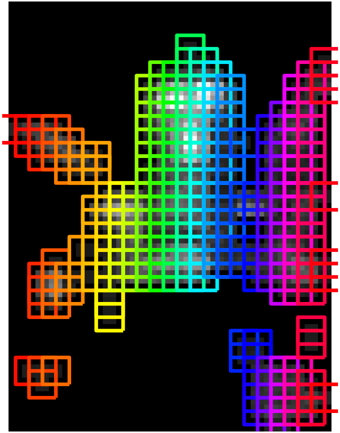
feature saliency (FS) map



region detection with EGM

[Avrithis and Kalantidis 2012]

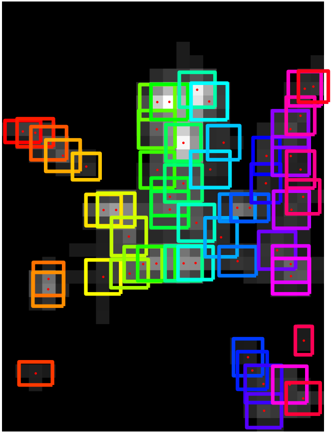


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

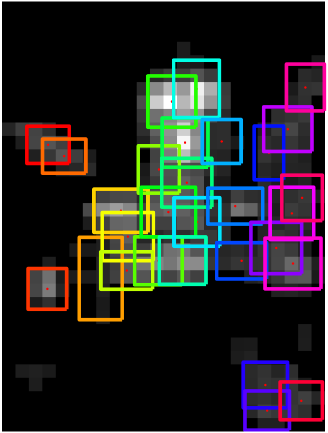


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

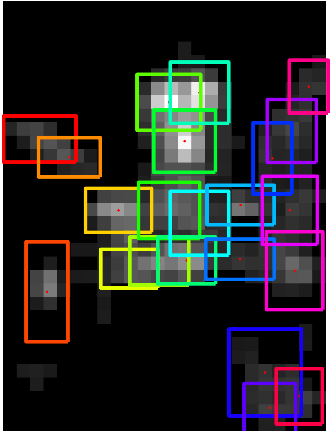


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

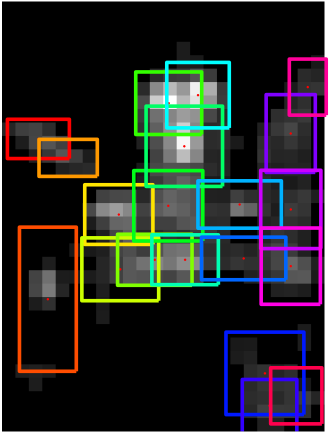


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

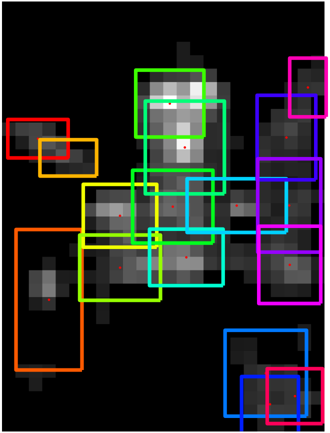


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

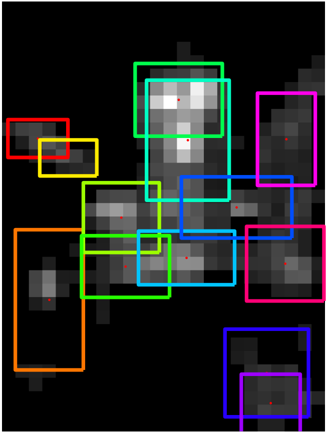


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

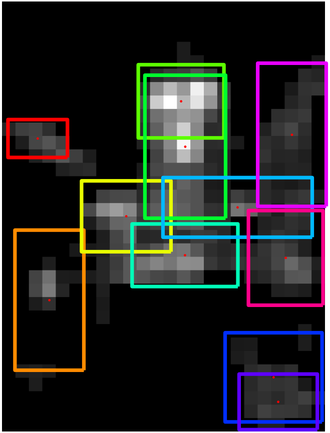


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

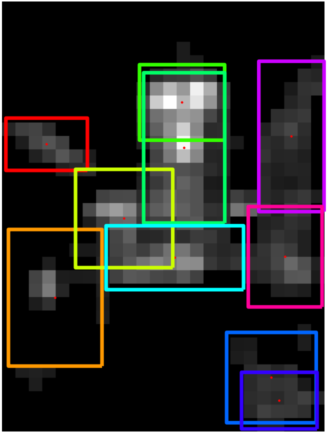


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

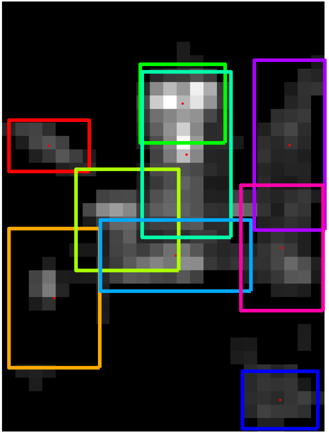


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

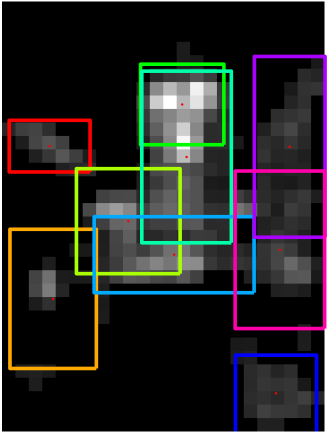


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

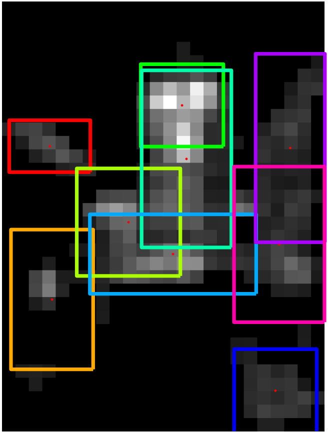


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

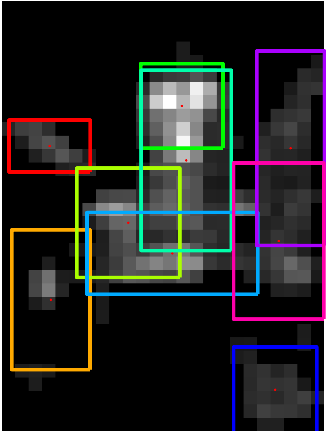


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

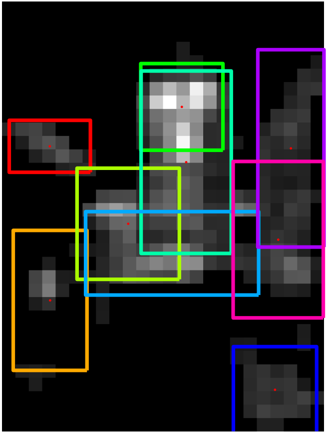


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]

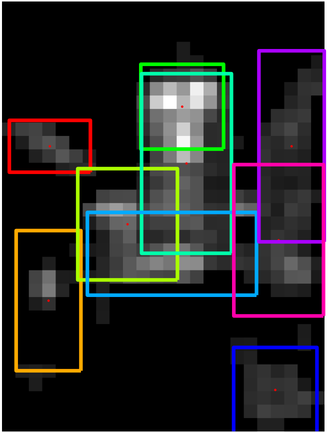


- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

region detection with EGM

[Avrithis and Kalantidis 2012]



- expanding Gaussian mixtures (EGM)
- generalized from points to 2d functions (images)

Avrithis and Kalantidis. ECCV 2012. Approximate Gaussian Mixtures for Large Scale Vocabularies.

graph centrality

- construct graph from detected regions
- local search

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

  where $y_i = \mathbb{1}[i \text{ is query}]$
- global centrality (Katz)

$$\mathcal{L}_\alpha \mathbf{g} = \mathbf{1}$$

Katz. Psychometrika 1953. A New Status Index Derived From Sociometric Analysis.

graph centrality

- construct graph from detected regions
- local search

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

  where $y_i = \mathbb{1}[i \text{ is query}]$
- global centrality (Katz)

$$\mathcal{L}_\alpha \mathbf{g} = \mathbf{1}$$

Katz. Psychometrika 1953. A New Status Index Derived From Sociometric Analysis.

graph centrality

- construct graph from detected regions
- local search

$$\mathcal{L}_\alpha \mathbf{x} = \mathbf{y}$$

  where $y_i = \mathbb{1}[i \text{ is query}]$
- global centrality (Katz)

$$\mathcal{L}_\alpha \mathbf{g} = \mathbf{1}$$

Katz. Psychometrika 1953. A New Status Index Derived From Sociometric Analysis.

object saliency (OS) map

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

object saliency (OS) map

$$S(\underbrace{p}_{\text{patch}}) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

object saliency (OS) map

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

patch

descriptor

object saliency (OS) map

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

patch

neighbors

descriptor

object saliency (OS) map

similarity

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

patch

neighbors

descriptor

object saliency (OS) map

similarity

FS

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

patch

neighbors

descriptor

object saliency (OS) map

similarity

FS

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

centrality

patch

descriptor

neighbors

object saliency (OS) map

FS similarity FS

$$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$

centrality

patch neighbors descriptor

object saliency (OS) map

FS similarity FS

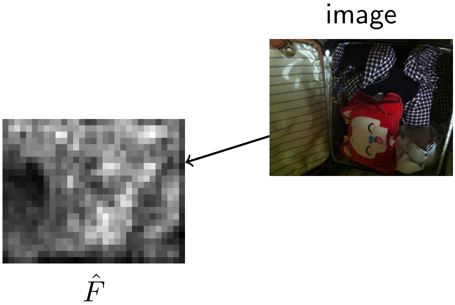
OS $$S(p) = \hat{F}(p) \sum_{i \in N_p} \mathrm{sim}(\mathbf{v}_i, \mathbf{u}_p) f_i g_i$$ centrality

patch neighbors descriptor

object saliency (OS) map

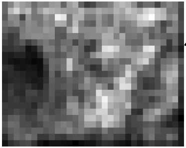
image

object saliency (OS) map

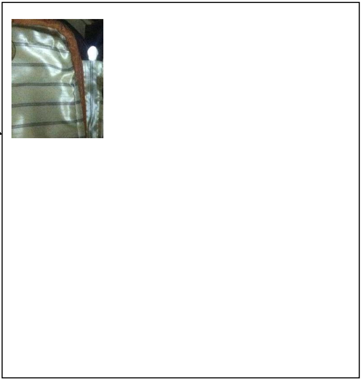


object saliency (OS) map

image

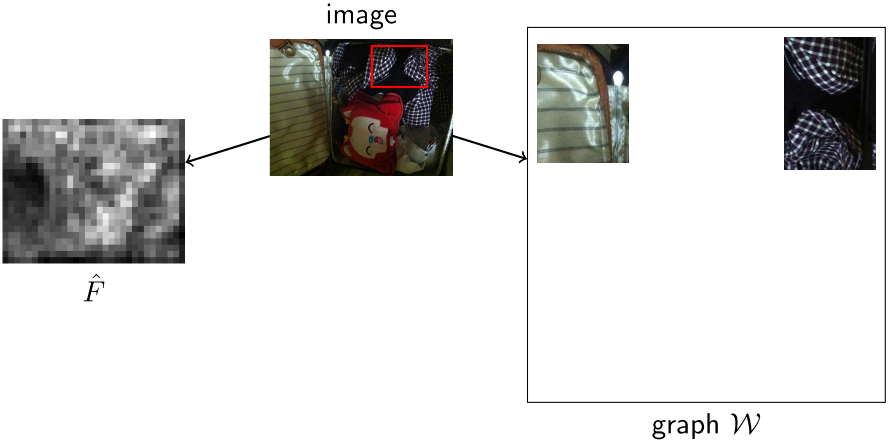


$\hat{F}$

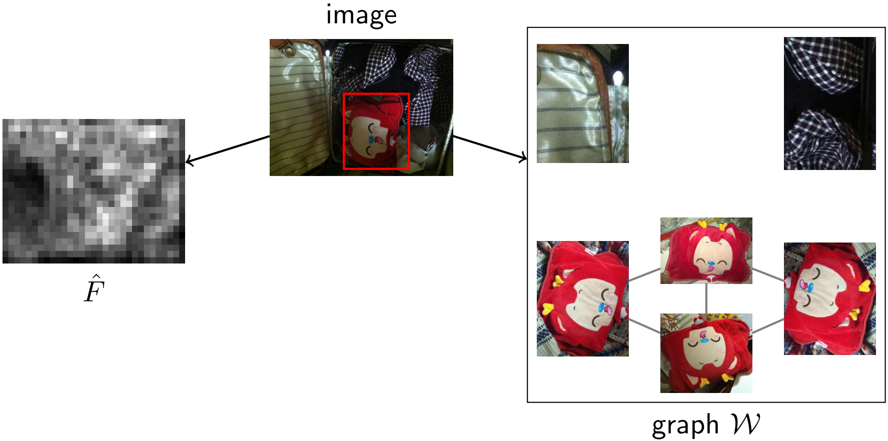


graph $\mathcal{W}$

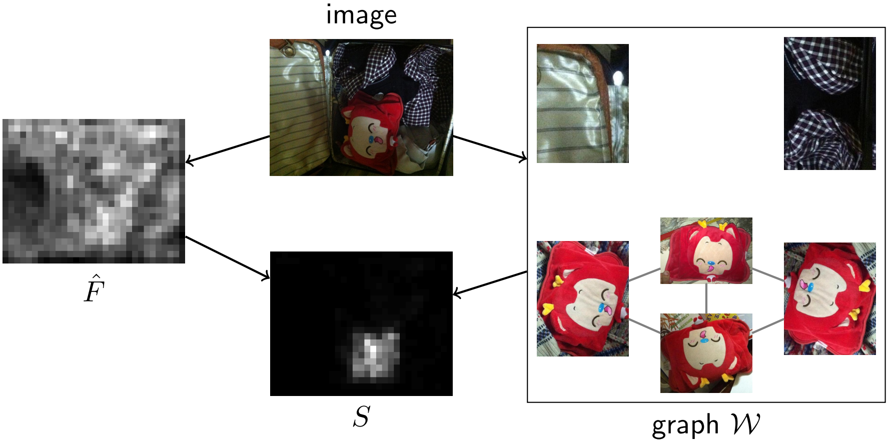
object saliency (OS) map



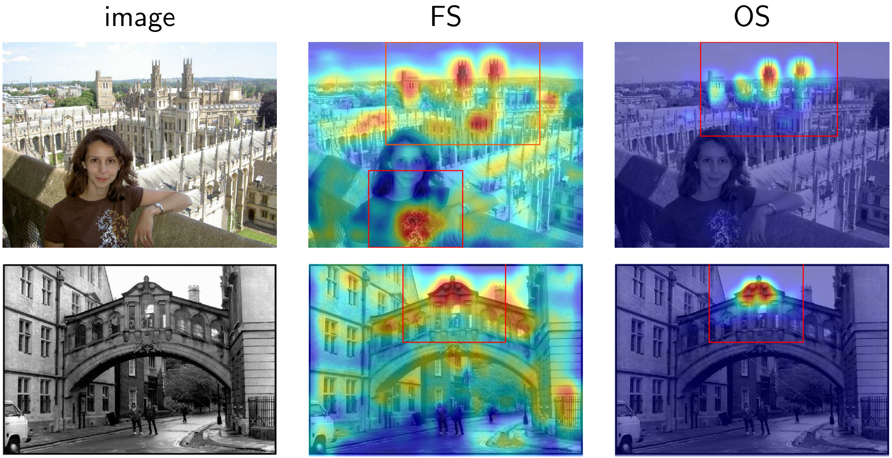
object saliency (OS) map



object saliency (OS) map

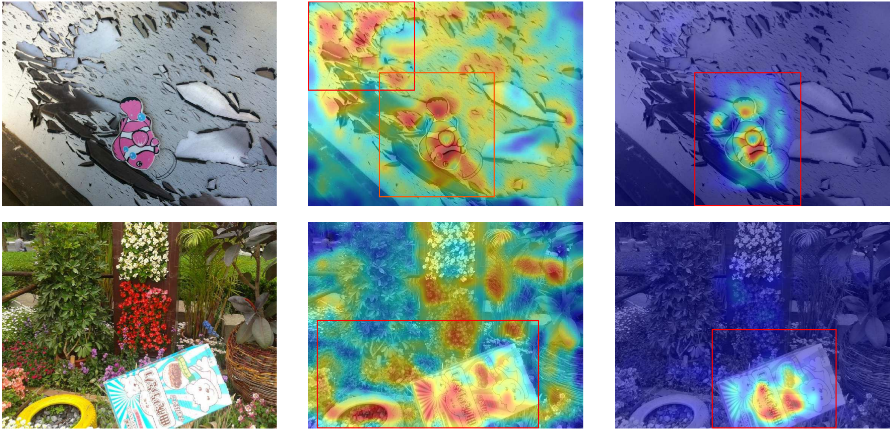


FS versus OS (Oxford 5k)

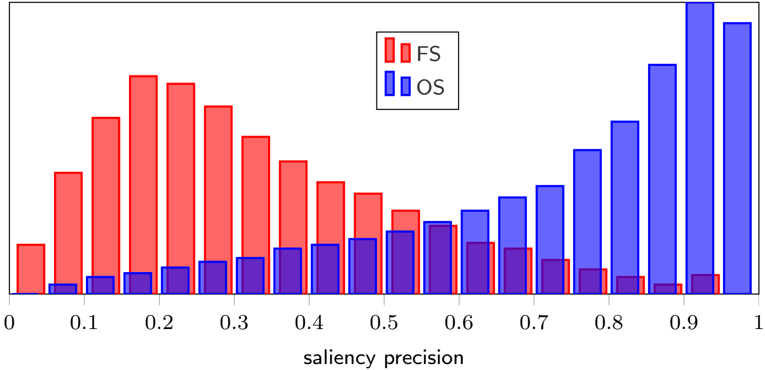


FS versus OS (INSTRE)

image FS OS



what does OS find?

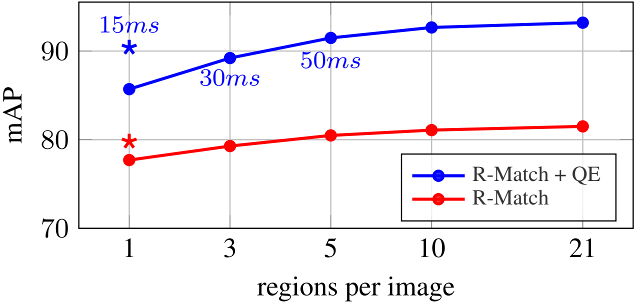


- precision: sum of saliency over ground truth regions, normalized by the sum over the entire image

global image representation

- fine-tuned VGG features [Radenovic *et al.* 2016]
- compute FS, detect regions with EGM and construct graph
- compute OS for each image in the dataset
- re-detect regions with EGM
- max-pool over regions, sum-pool globally as in R-MAC

global versus regional



- regional search: $O(n)$ space and $O(n^2)$ query time, where n is the number of regions (descriptors) per image
- same performance with 5 times less memory and ≈ 4 times faster

state of the art (global)

Method	QE	Instre	Oxford	Oxford105k
MAC	-	48.5	79.7	73.9
R-MAC	-	47.7	77.7	70.1
FS.EGM *	-	48.4	77.5	70.2
OS.EGM *	-	50.1	79.6	71.8
OS.EGM-$\triangle^\star$	-	53.7	79.8	71.4
MAC	✓	71.8	87.4	86.0
R-MAC	✓	70.3	85.7	82.7
FS.EGM *	✓	71.2	89.8	87.9
OS.EGM *	✓	72.7	**90.4**	**88.0**
OS.EGM-$\triangle^\star$	✓	**75.4**	90.1	84.3

- always better than R-MAC, up to 6% at large scale
- compete MAC, even though network was optimized for that
- most gain with QE

summary

- let's go and learn with as little supervision as possible!

joint work with

Oriane Siméoni

Ahmet Iscen

Giorgos Tolias

Teddy Furon

Ondrej Chum

unsupervised object discovery

https://arxiv.org/abs/1709.04725

fast spectral ranking

https://arxiv.org/abs/1703.06935

diffusion on region manifolds

https://arxiv.org/abs/1611.05113

thank you!