Umělá inteligence a strojové učení

Michal Hradiš: Neural networks

Where to get more information

Ian Goodfellow and Yoshua Bengio and Aaron Courville**: Deep Learning,** MIT Press, 2016. http://www.deeplearningbook.org/

Andrew Ng and Kian Katanforoosh: CS230 **Deep Learning**, Stanford, https://cs230.stanford.edu/

CS231n: **Convolutional Neural Networks for Visual Recognition**, Stanford, http://cs231n.stanford.edu/index.html

Andrew Ng, CS229 Machine Learning, Stanford, https://www.youtube.com/watch?v=UzxYlbK2c7E&list=PLEBC422EC5973B4D8

Python Notebooks

Basic Pytorch - https://colab.research.google.com/drive/1ytNsahwdl4FJDkXmaw7CCQQJqQDiAxJt?usp=sharing

Image classification -

https://colab.research.google.com/drive/1oh-iicfuclPJRudYZVIUWRRLtJiMdM4k?usp=sharing#scrollTo=5PRaK3DSPzQw

https://colab.research.google.com/drive/1a8iZeo-pmrRDAHB8cK0TrEmaulMnaKNC#scrollTo=7k37ifwroNnN

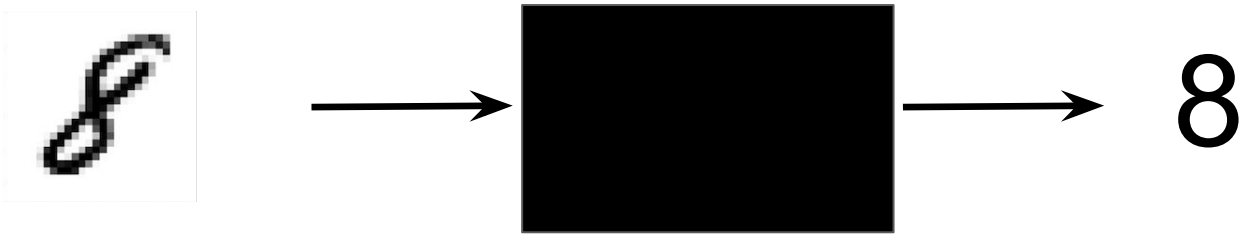
Pytorch tutorials - https://pytorch.org/tutorials/


AI PLAYS BASKETBALL


HUMAN DIGITIZATION

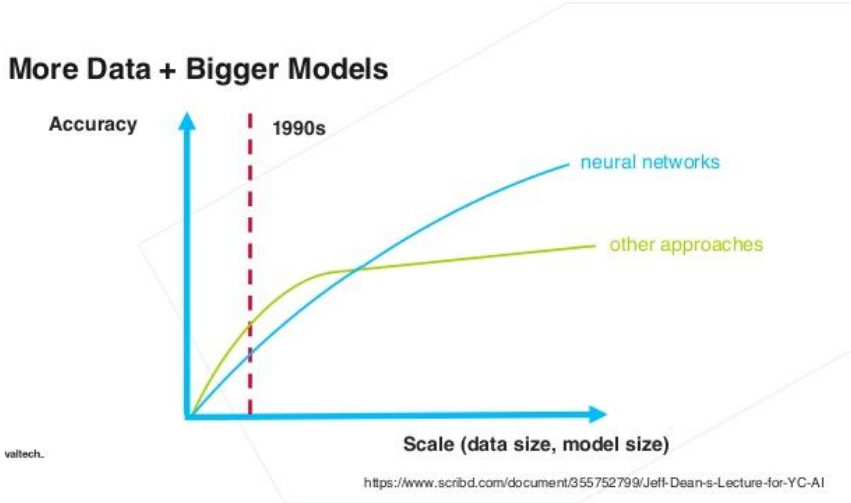
What are neural networks?

- Complex and flexible functions (and computationally expensive)
- Can learn to approximate a desired function from training data
- Can accept large range of input types and produce large range of input types
- Can be “customized” to solve very different tasks.



Why?

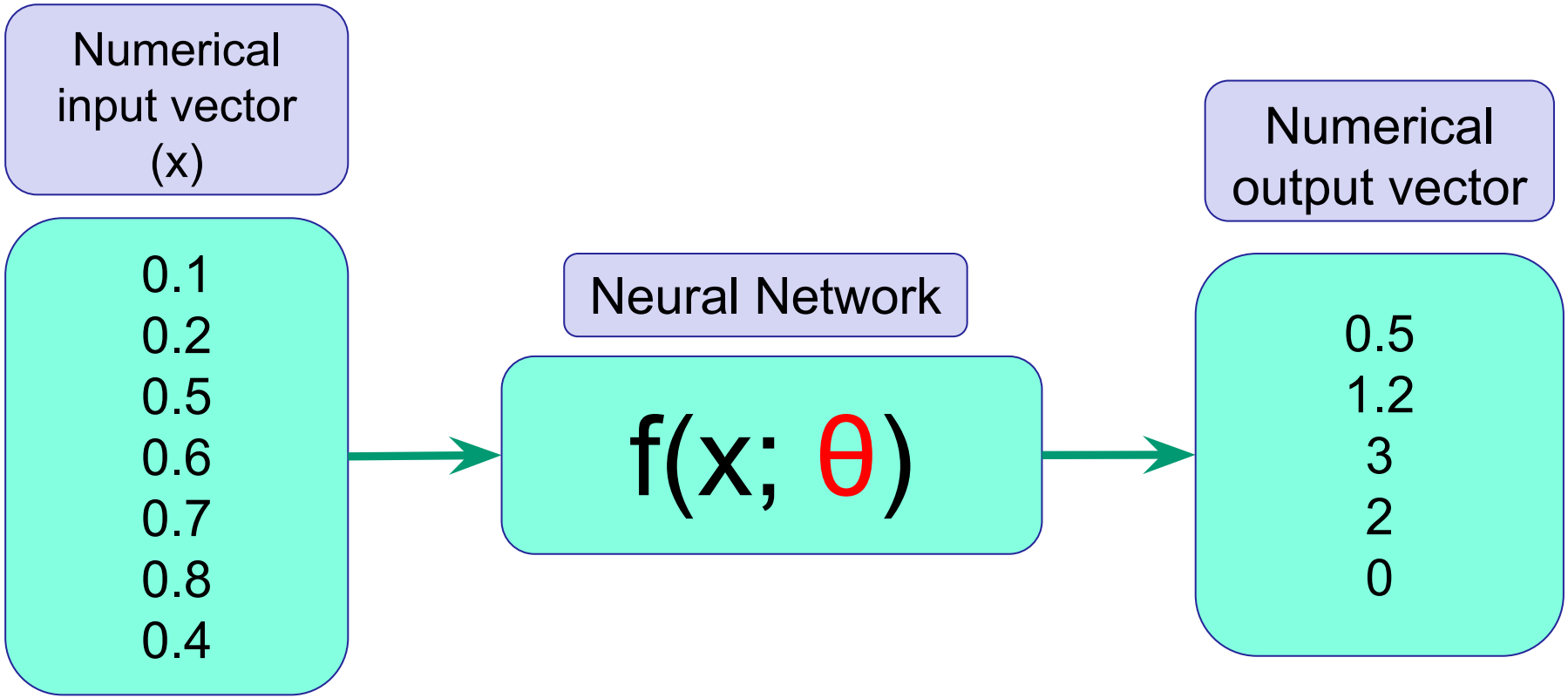
- Scales with computation
- Scales with data
- General framework
- State of the art
- Massive development
- Mature tools



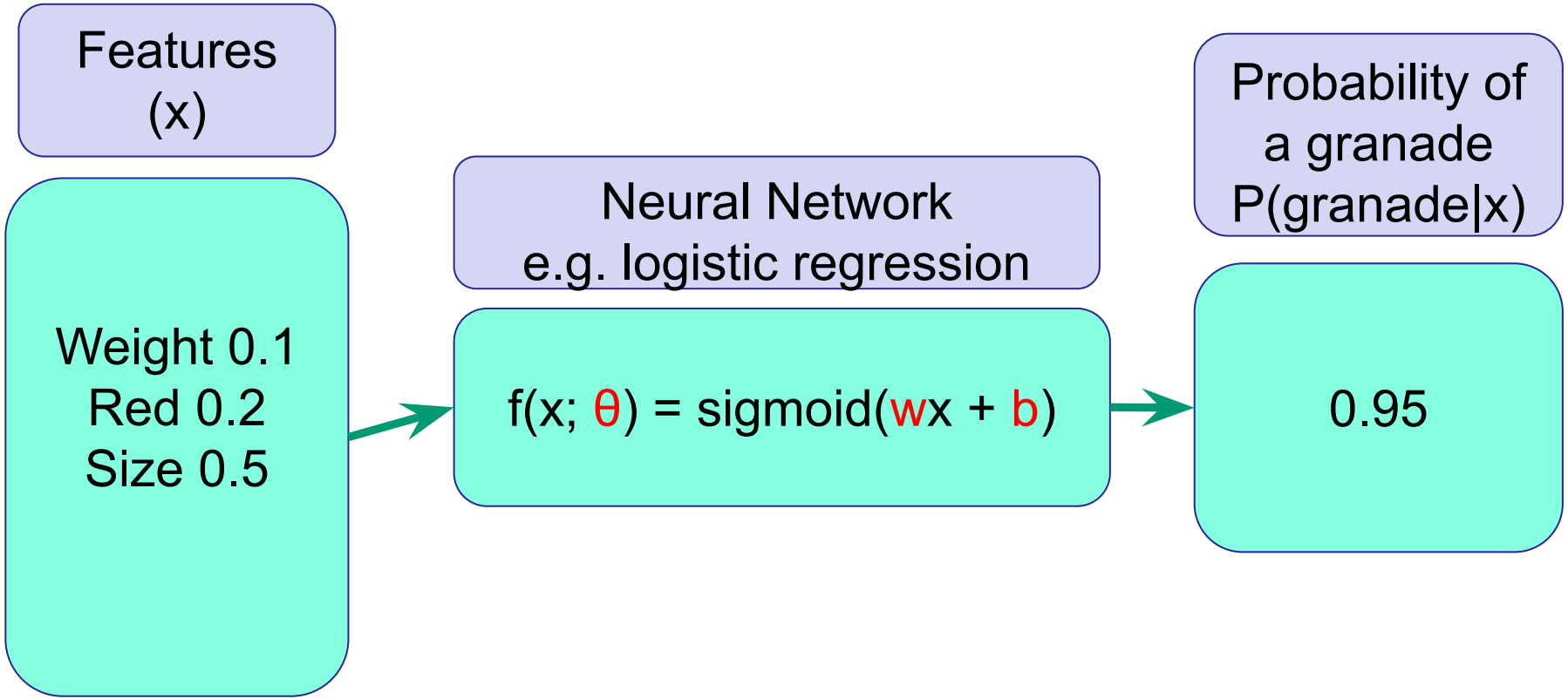
Goals

- Understand basics of machine learning with neural networks
- Understand the flexibility of neural networks
- Have an idea what tools are available
- Complex models and their applications
- Basic intuitions when solving practical tasks
  - Data
  - Generalization
  - Regularization
  - What can go wrong
  - Transfer learning

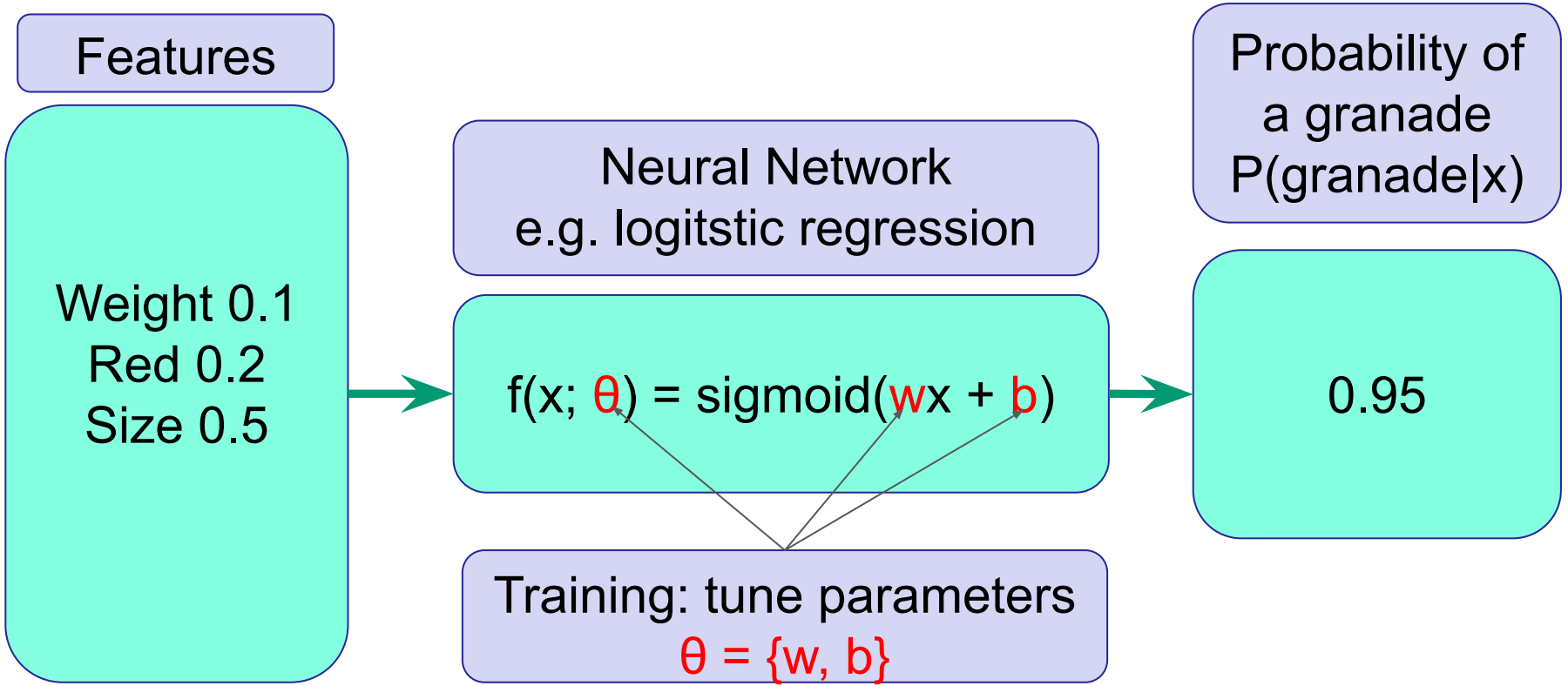
What is a neural network? (in this lecture)



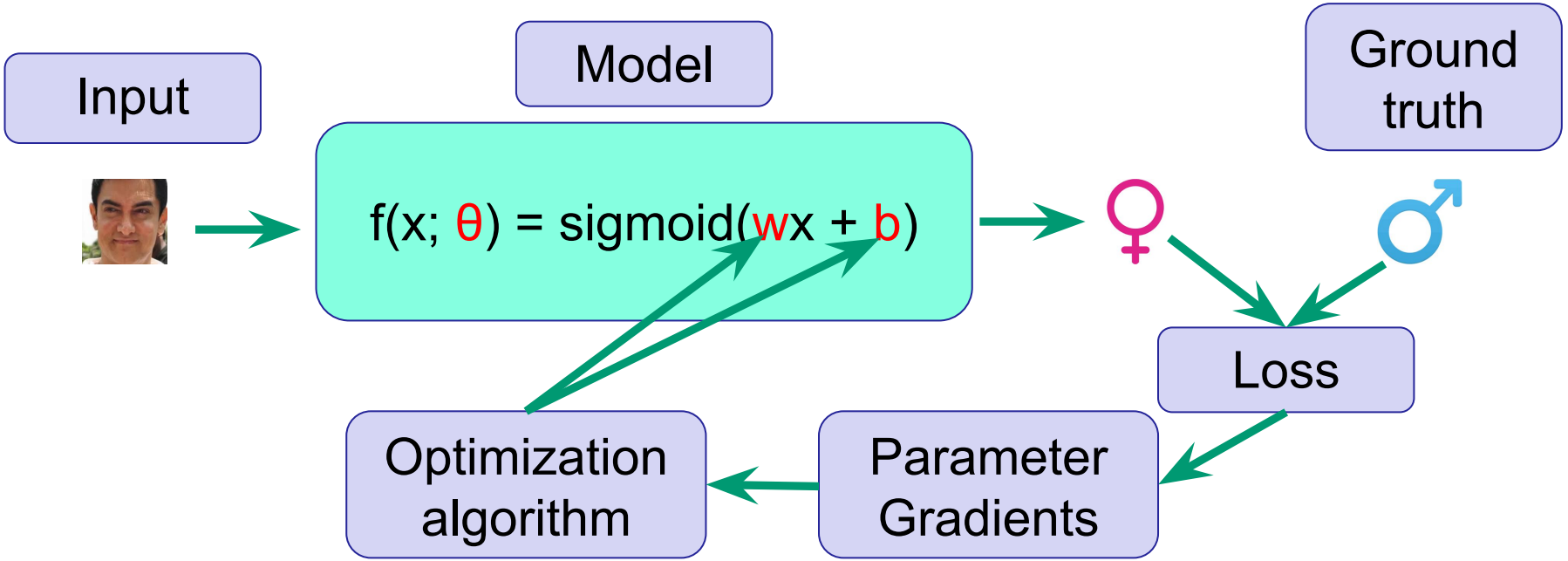
Binary classification network



Binary classification network



Training


Input
Model
Ground truth
f(x; θ) = sigmoid(wx + b)
Loss
Optimization algorithm
Parameter Gradients

Supervised learning components

Training data - inputs and desired network outputs

D = {( ♂), ( ♀), ( ♂), ( ♀)}

Model with parameters (network)

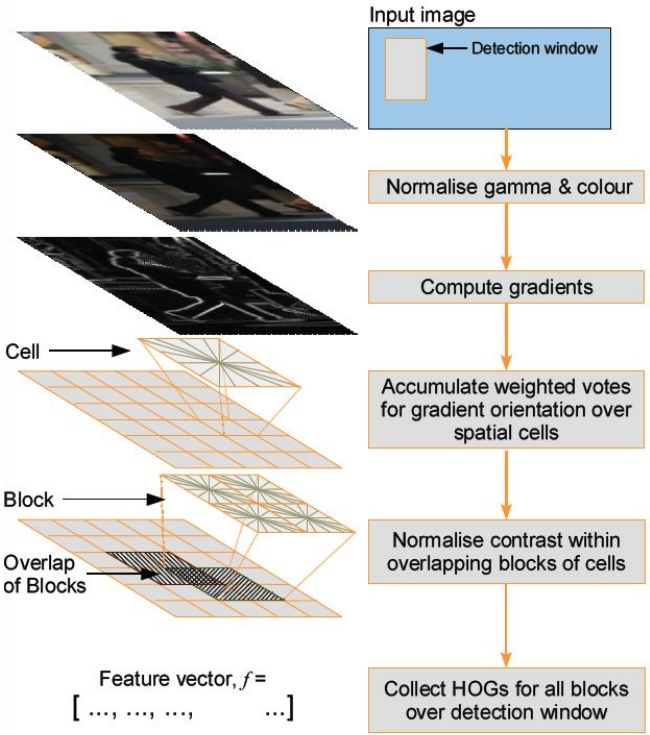
f(x; θ) = sigmoid(wx + b)

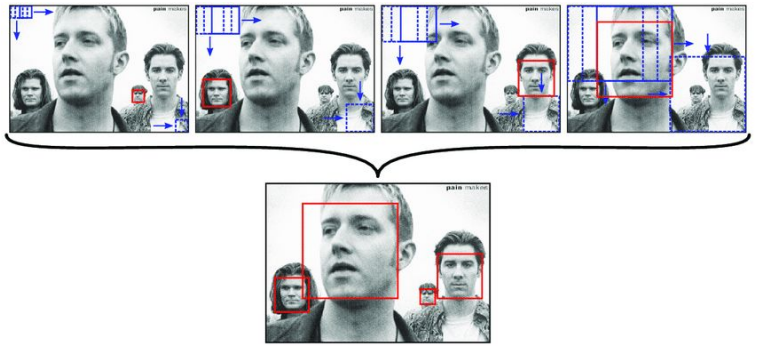
Loss function (e.g. cross entropy)

$$loss(x, y) = \begin{cases} log(f(x;\theta)) & \textbf{if} \quad y = male \\ log(1 - f(x;\theta)) & \textbf{if} \quad y = female \end{cases}$$

Objective function $$J(D, \theta) = \sum_{D=\{(x_i, y_i), \ldots\}} loss\left(f\left(x_i, \theta\right), y_i\right)$$

“Shallow” learning



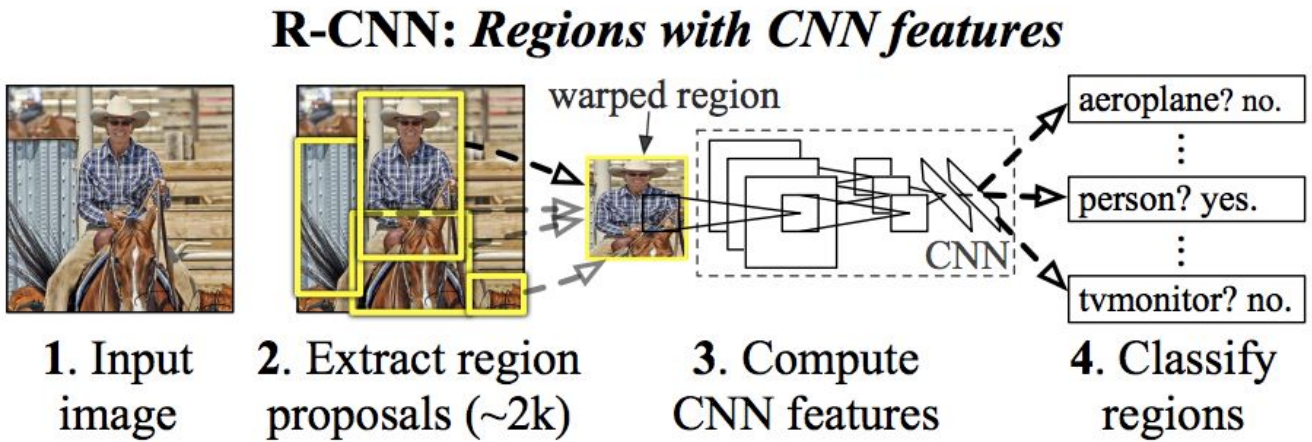


$$f([\text{features}]; \theta) = \text{sigmoid}(w[\text{features}] + b)$$

N. Dalal, and B. Triggs. **Histograms of Oriented Gradients for Human Detection**. In *Proc. CVPR*, 2005.
Šochman. Learning for Sequential Classification. Thesis, CMP, 2009.

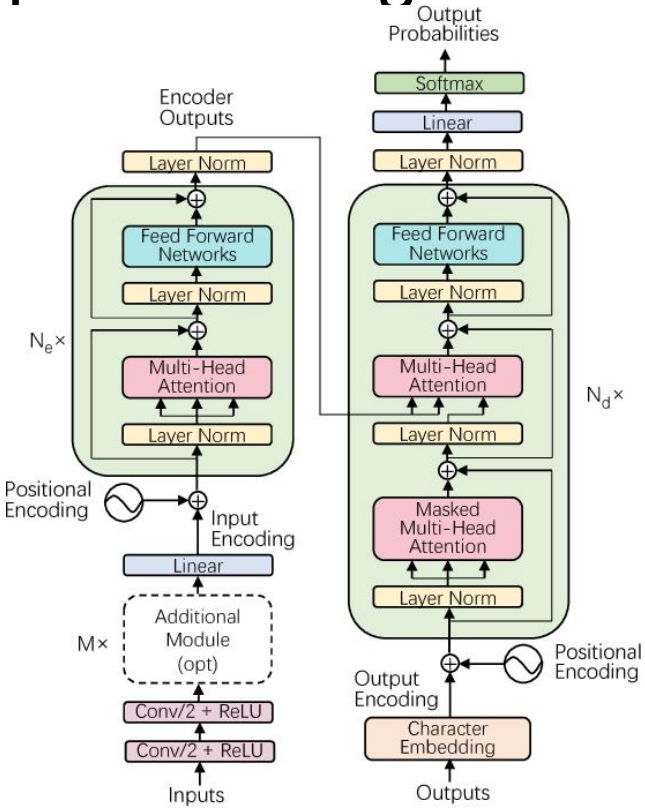
Deep learning

- No hand coded feature extraction
- Our domain knowledge goes into network architecture, regularization, loss, data augmentation, ...



Girshic et al.: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. CVPR, 2014.

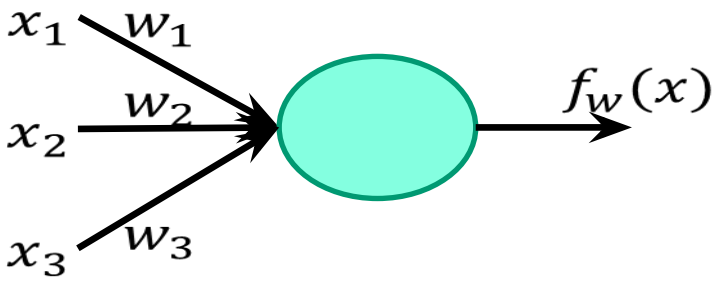
End to end speech recognition/translation



Honza šel na jahody

Dong, L., Xu, S., & Xu, B.. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. ICASSP, 2018.

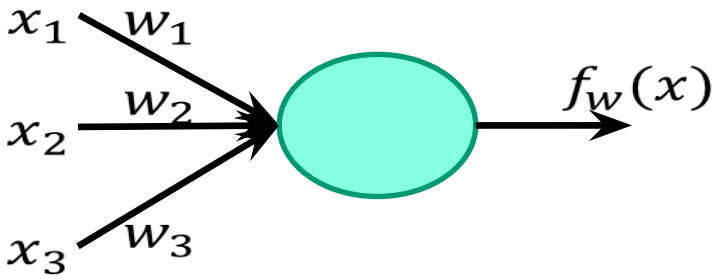
Neural network components (neurons)

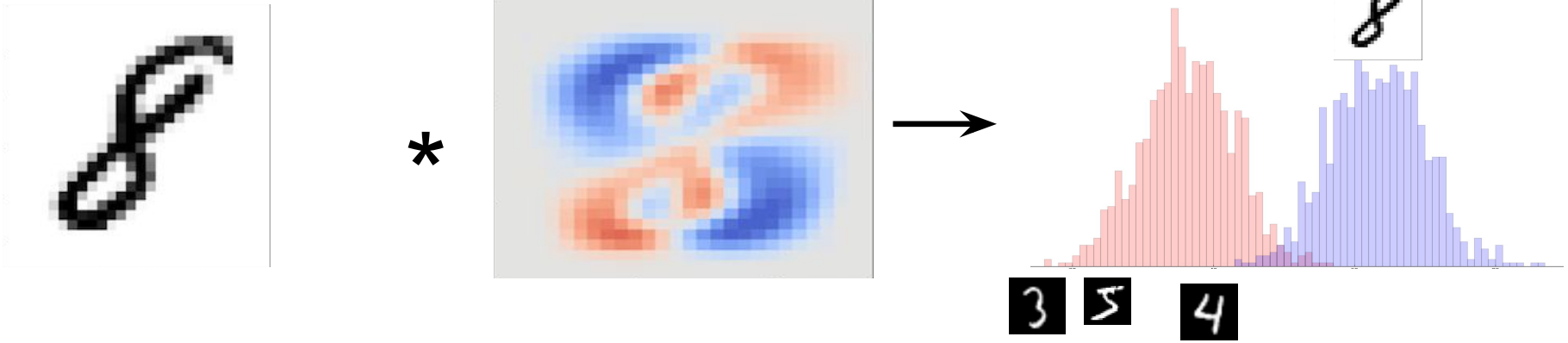


$$f(\mathrm{x};\mathrm{w}) = \mathrm{w}_1\mathrm{x}_1 + \mathrm{w}_2\mathrm{x}_2 + \mathrm{w}_3\mathrm{x}_3$$
$$= \sum_i \mathrm{w}_i\mathrm{x}_i$$

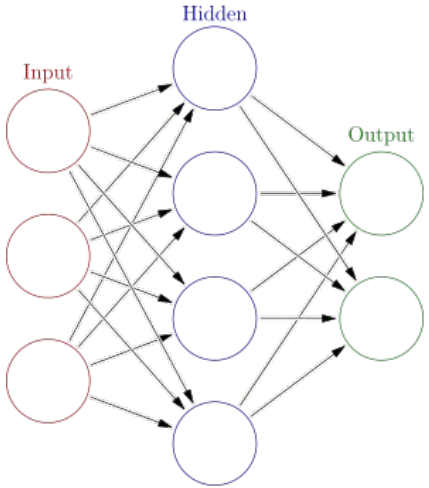
Neurons as templates

Neurons have higher activations if the input “matches” their weights.



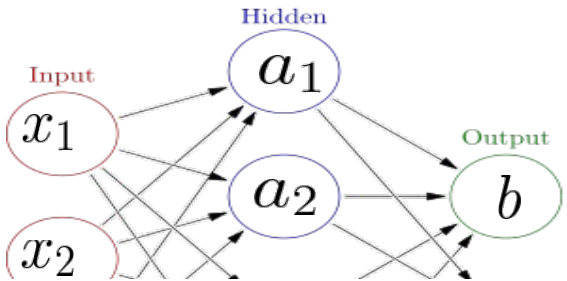


Composition of linear functions?



$$f(\mathrm{x};\mathrm{w}) = \sum_i \mathrm{w}_i \mathrm{x}_i$$

Composition of linear functions?



$$f(\mathrm{x};\mathrm{w}) = \sum_i \mathrm{w}_i \mathrm{x}_i$$

$$a_1 = 1x_1 + 2x_2$$

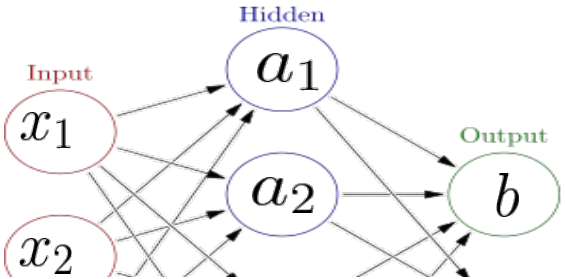
$$a_2 = -1x_1 + 1x_2$$

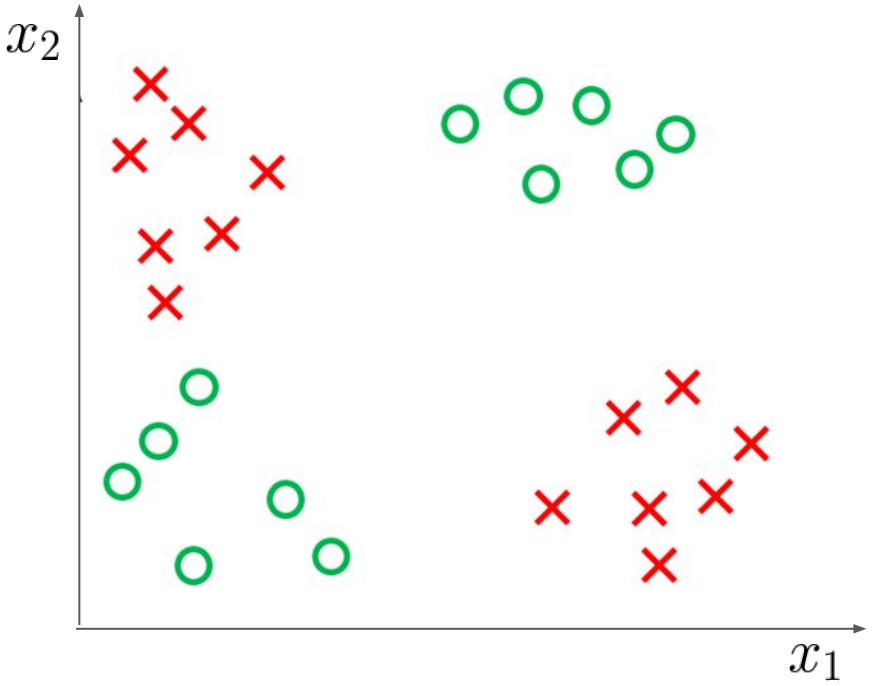
$$b = 4a_1 + 1a_2$$

$$= 4(1x_1 + 2x_2) + 1(-1x_1 + 1x_2)$$

$$= 3x_1 + 3x_2$$

Linear separability and composition of functions

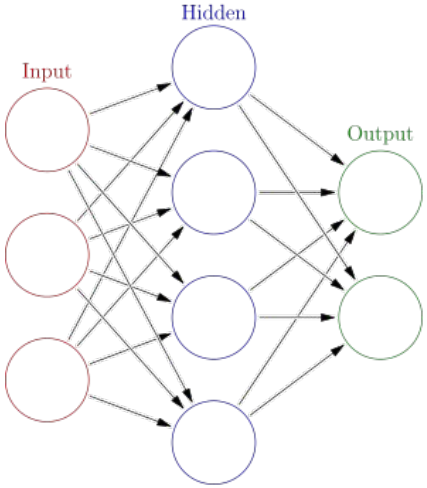




Non-linearity (more at https://en.wikipedia.org/wiki/Activation_function)

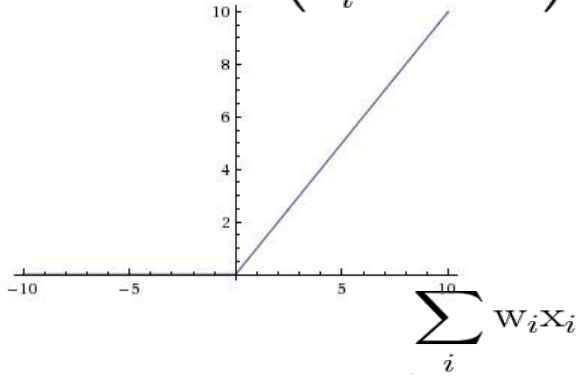
Non-linearities mostly process each activation independently.

One example is ReLU (rectified linear. unit).

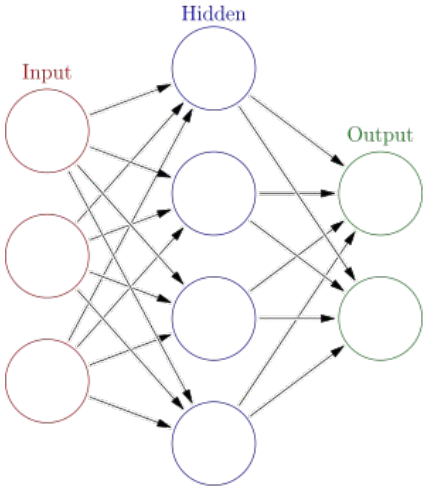


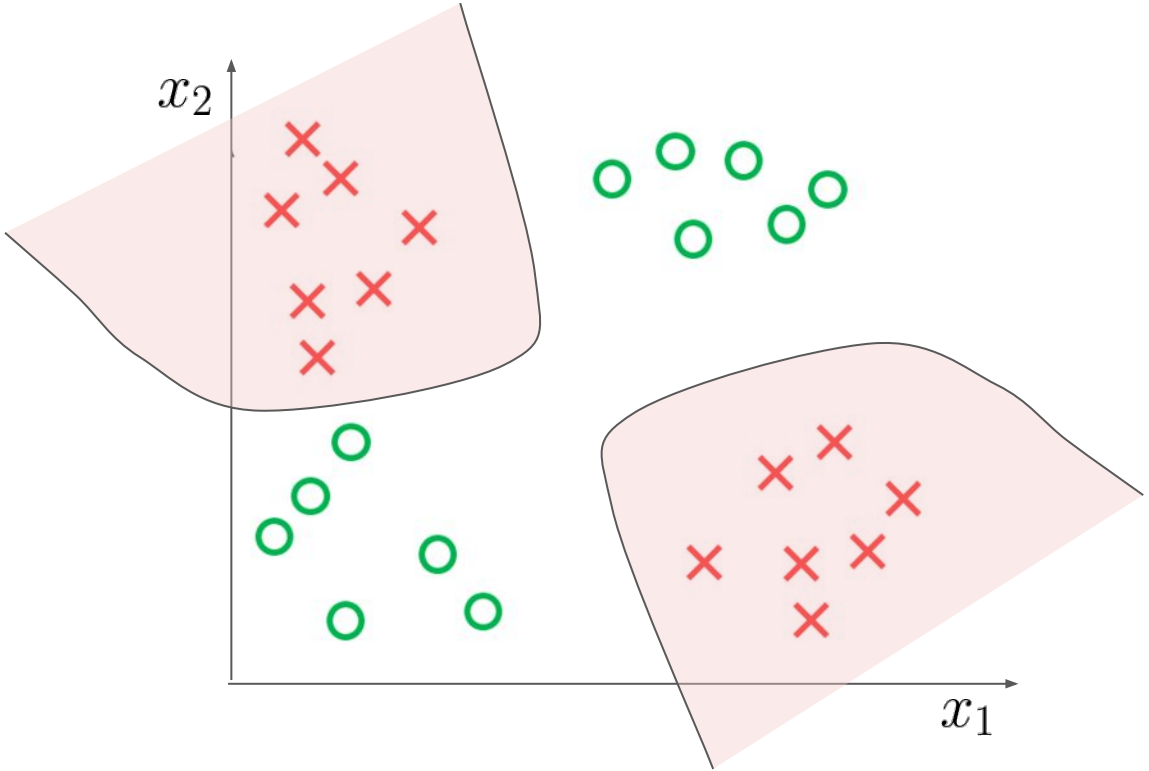
$$f(\mathrm{x};\mathrm{w}) = \sigma\left(\sum_i \mathrm{w}_i \mathrm{x}_i\right)$$

$$f(\mathrm{x};\mathrm{w}) = \max\left(\sum_i \mathrm{w}_i \mathrm{x}_i, 0\right)$$

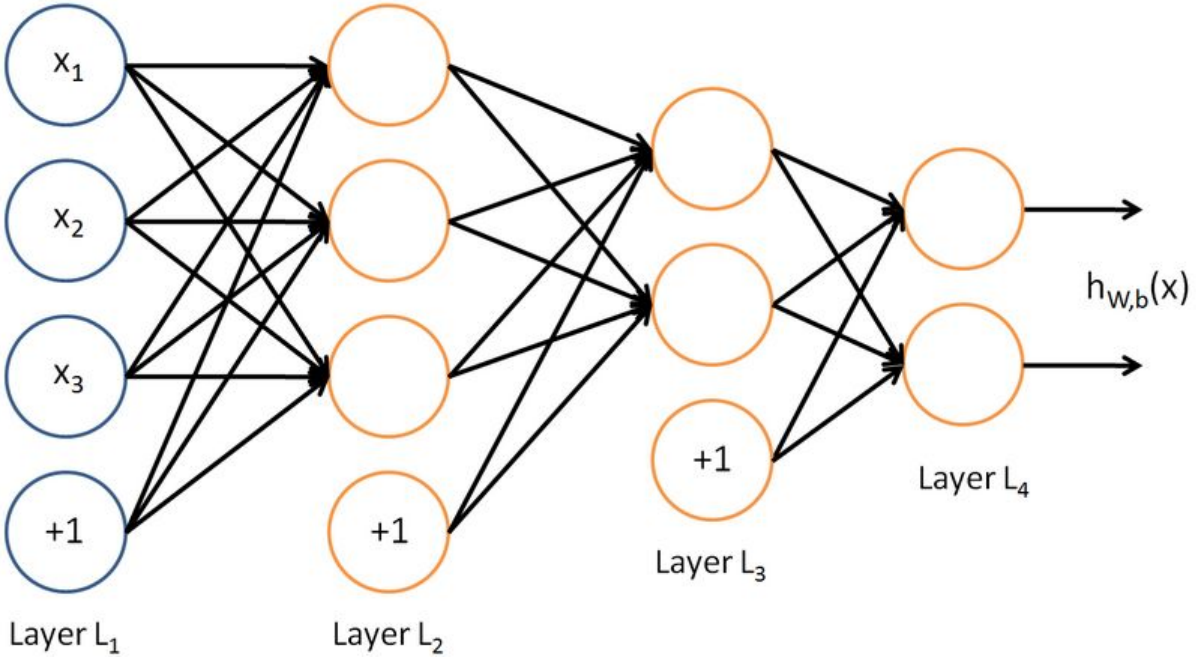


Linear separability and composition of functions



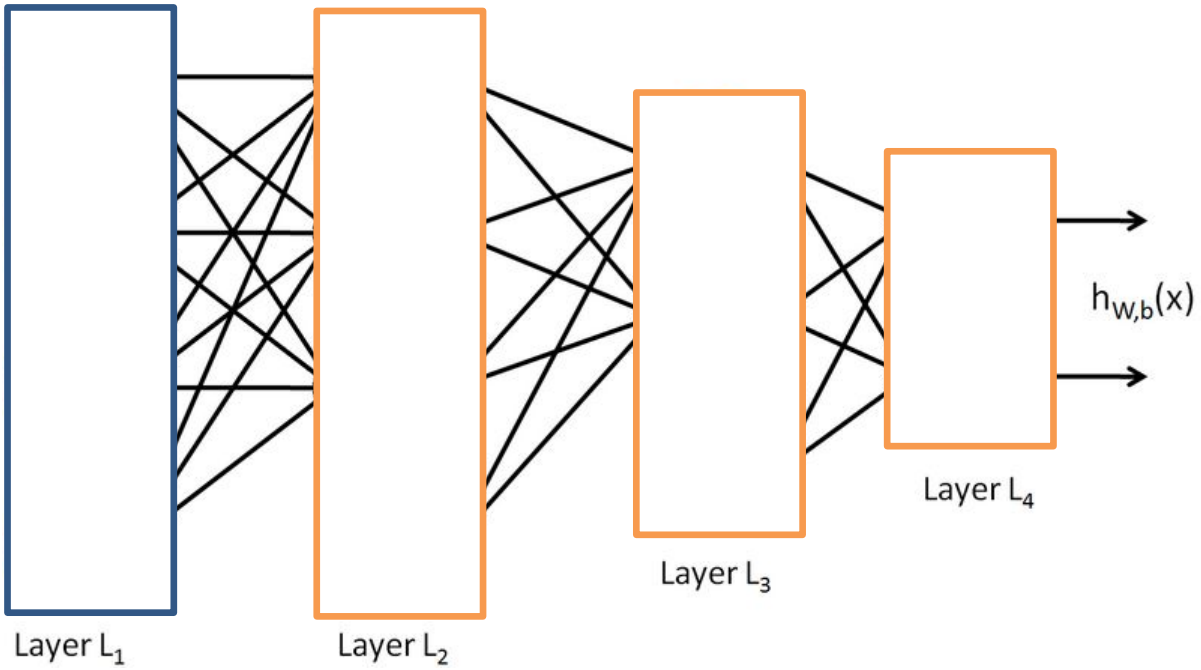


From neurons to layers



Source: UFLDL tutorial, http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

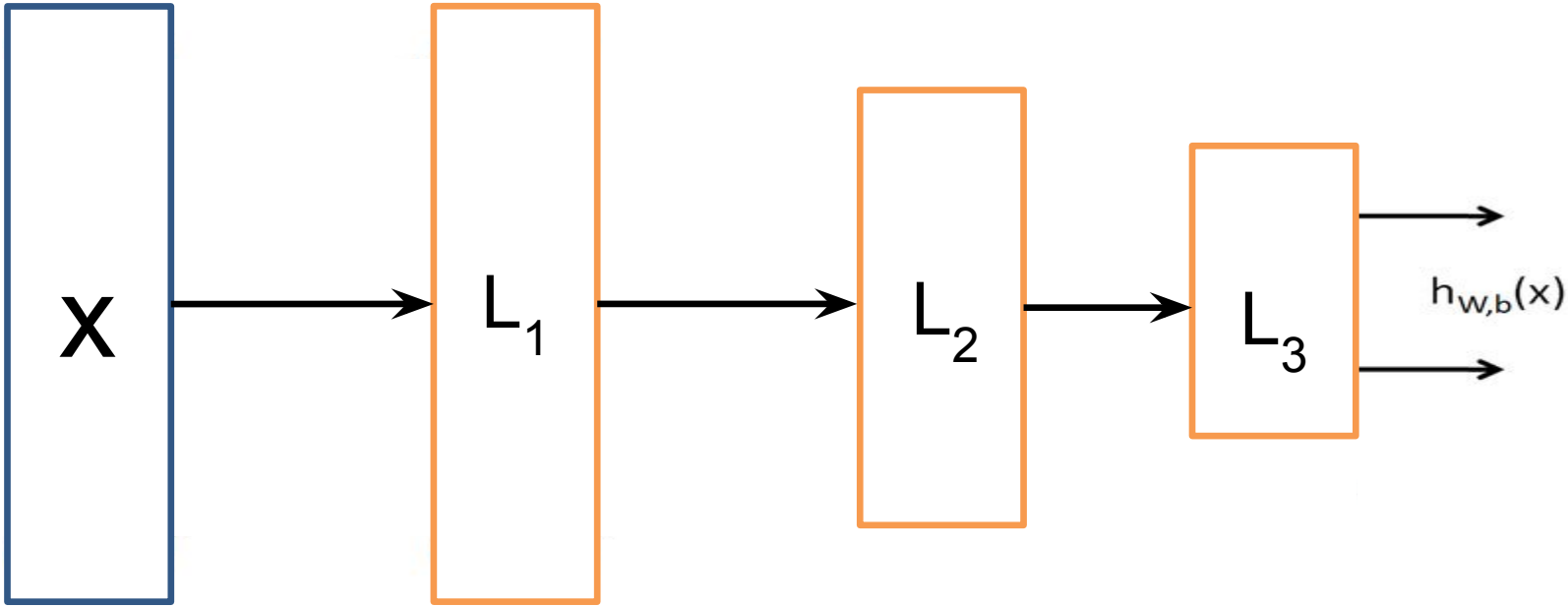
From neurons to layers



Source: UFLDL tutorial, http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

Layers

Often, we don't visualize the layers, but their inputs and outputs (activations).

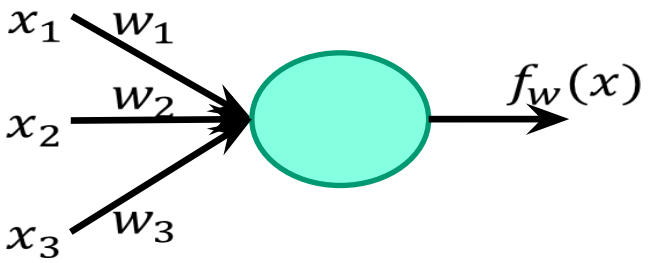


Source: UFLDL tutorial, http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

From neurons to matrices

We would like to define layer of neurons as single mathematical operation (as a linear layer).

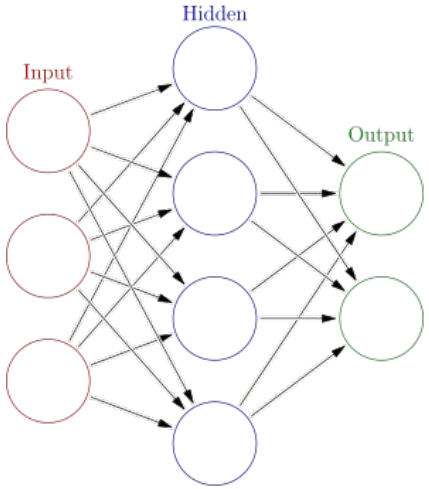
Solution: Linear algebra



$$f(\mathrm{x};\mathrm{w}) = \mathrm{w}_1\mathrm{x}_1 + \mathrm{w}_2\mathrm{x}_2 + \mathrm{w}_3\mathrm{x}_3 = \sum_i \mathrm{w}_i\mathrm{x}_i \quad \rightarrow \quad [1\ 0\ 3] \times \begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix} = -1$$

Linear layer

$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} = -1$$



$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} \begin{bmatrix}0\\2\\0\end{bmatrix} \begin{bmatrix}-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\0\end{bmatrix} = -1, 0, -1, -1$$

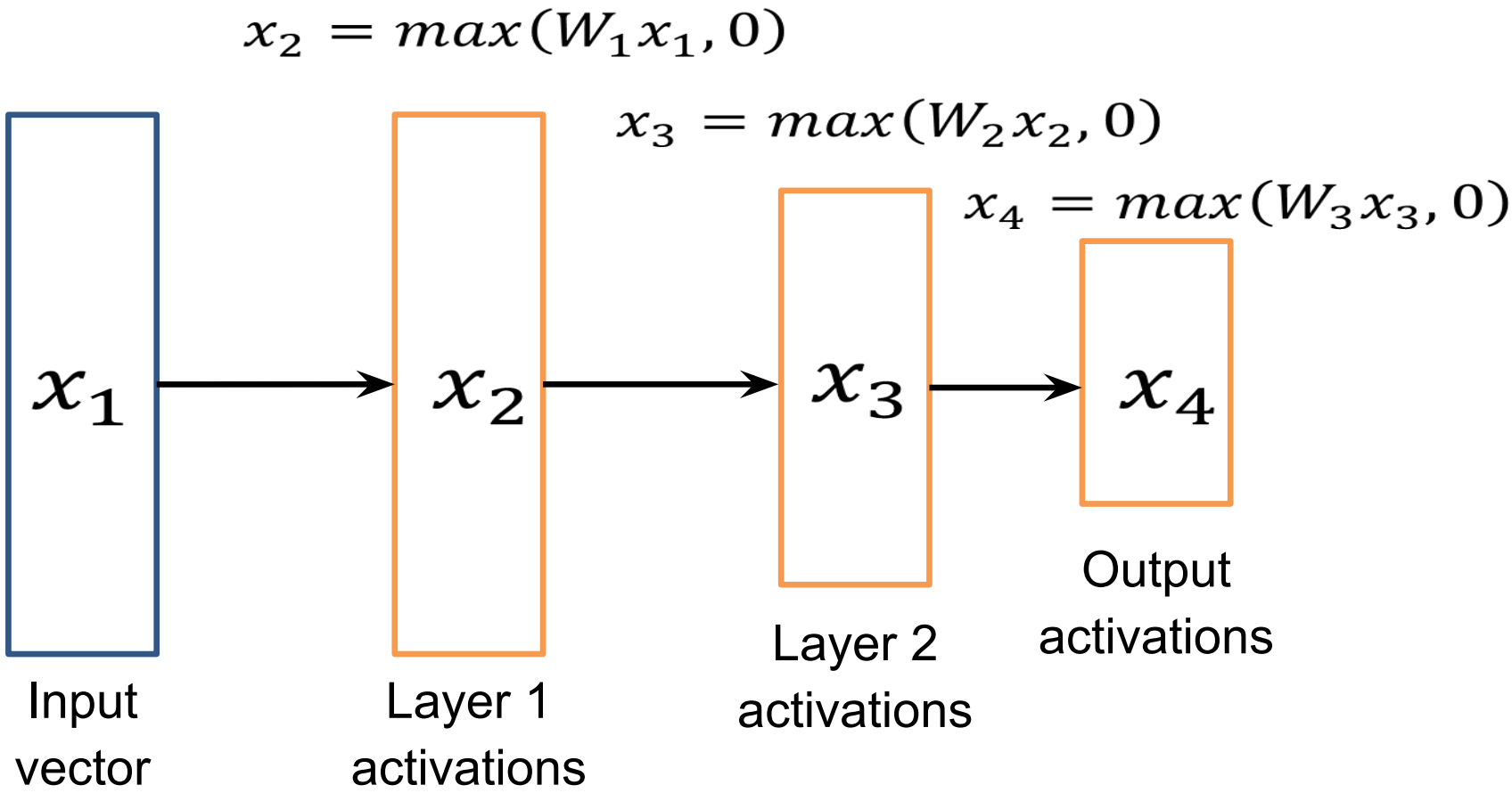
Linear layer

$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2\\-1\\-1\end{bmatrix} \begin{bmatrix}0\\2\\0\end{bmatrix} \begin{bmatrix}-1\\0\\0\end{bmatrix} \begin{bmatrix}1\\1\\0\end{bmatrix} = -1, 0, -1, -1$$

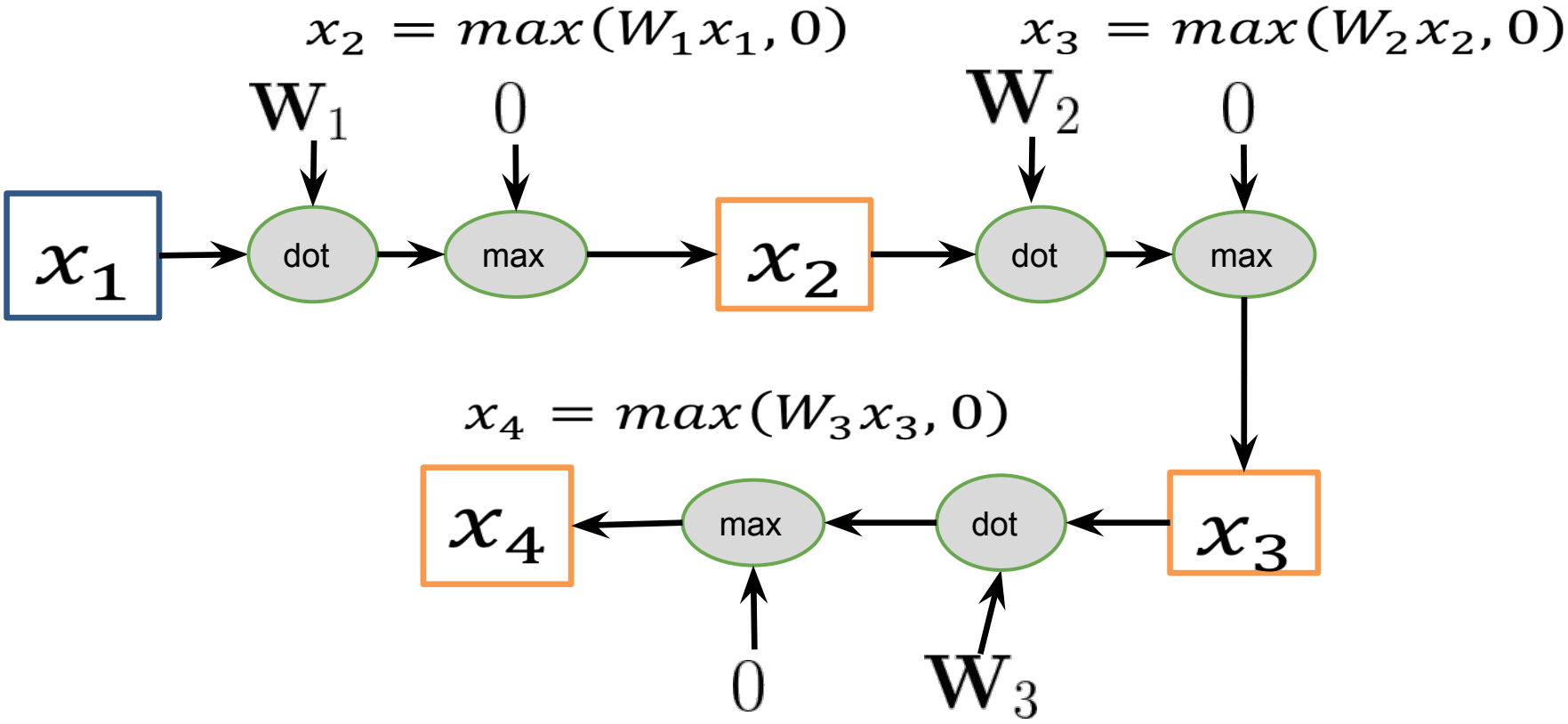
$$\begin{bmatrix}1 & 0 & 3\end{bmatrix} \times \begin{bmatrix}2 & 0 & -1 & 1\\-1 & 2 & 0 & 1\\-1 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}-1 & 0 & -1 & -1\end{bmatrix}$$

$$f(\mathrm{x}; \mathbf{W}) = \mathbf{W}\mathrm{x}$$

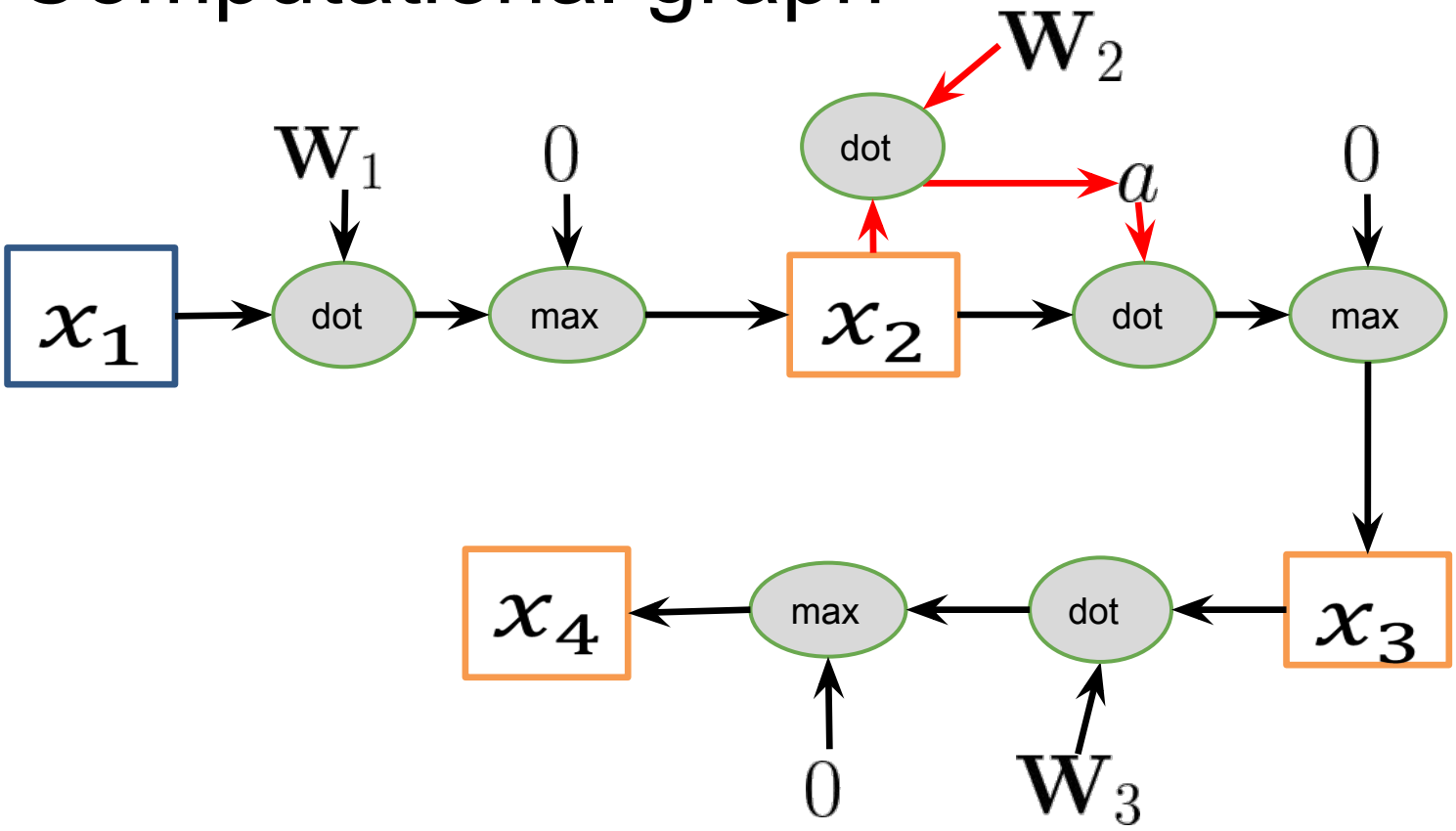
Neural network in code



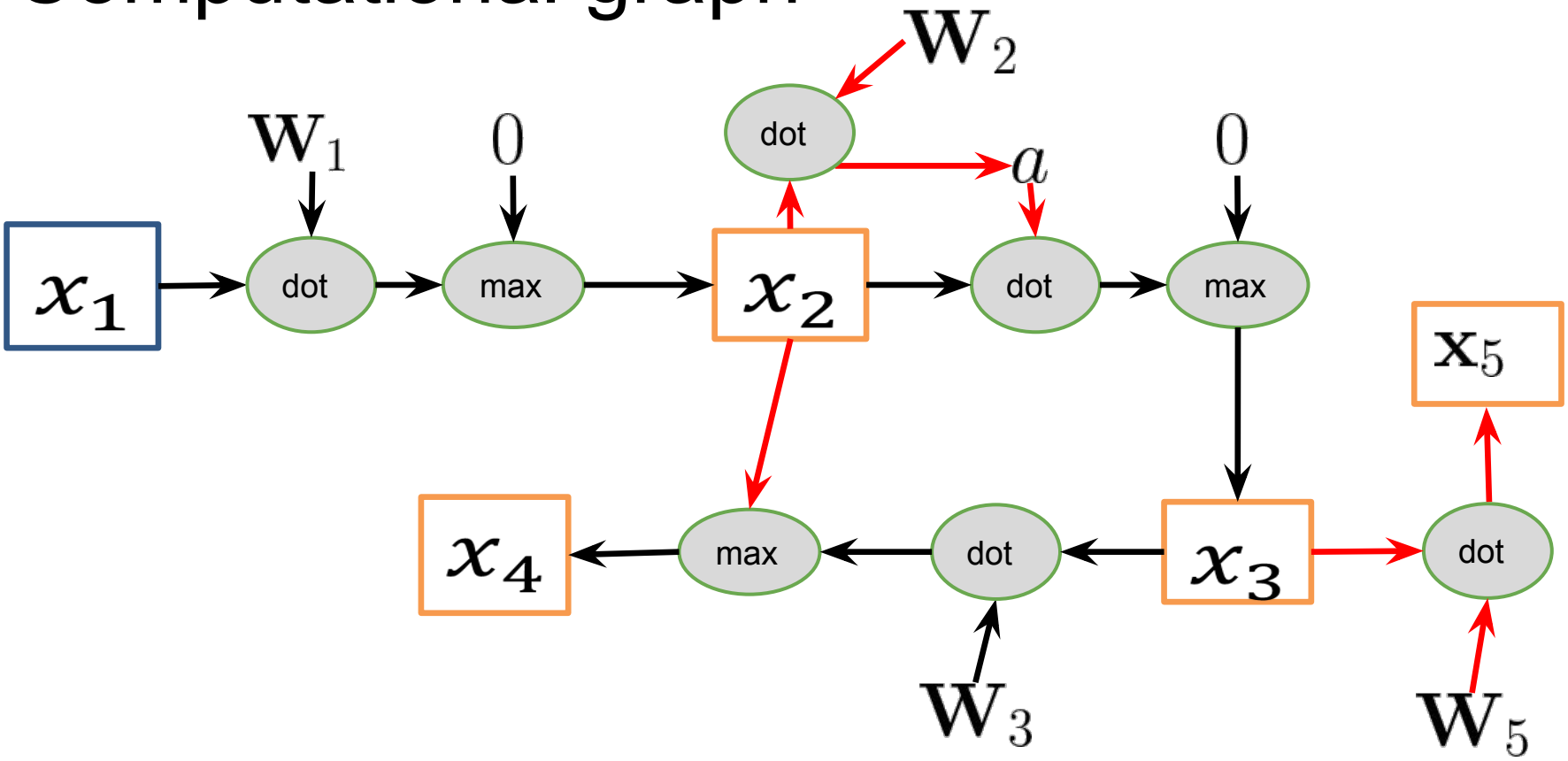
Computational graph



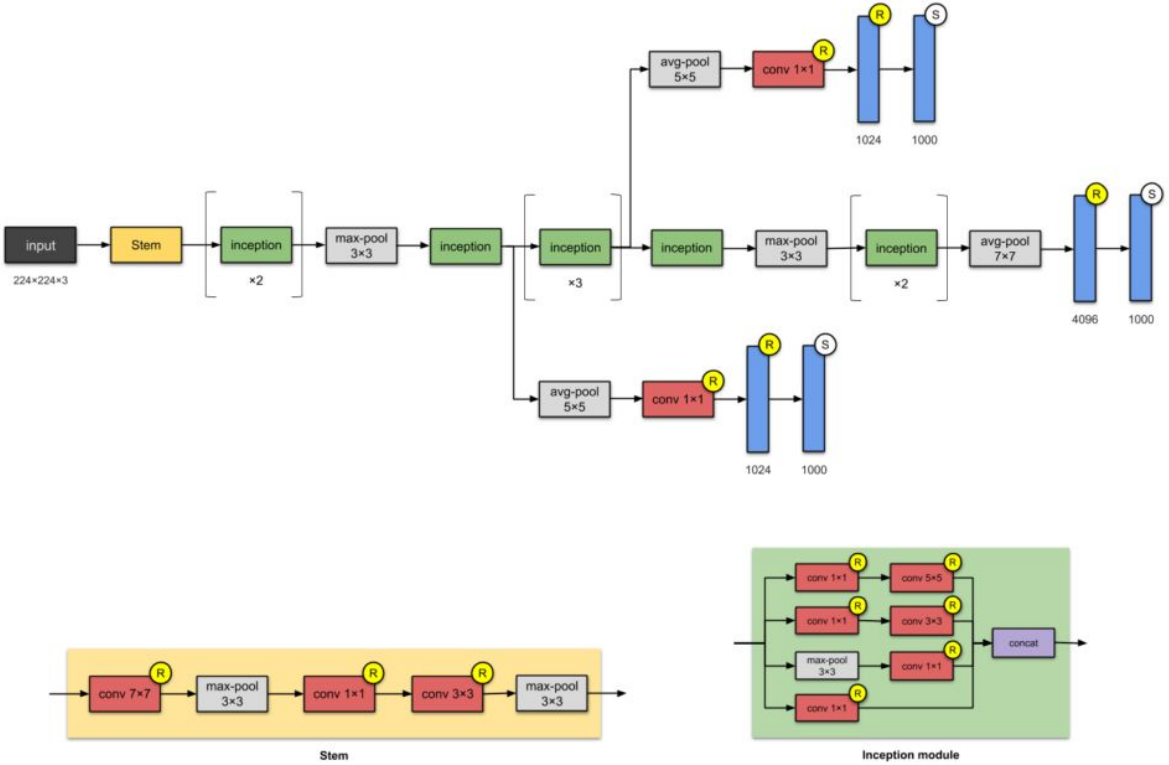
Computational graph



Computational graph



Real world example: Inception-V4



Source: raimibinkarim, Ilustrated: 10 CNN Architectures
https://makerspace.aisingapore.org/2019/09/illustrated-10-cnn-architectures/

Loss functions

- During training, loss function measures how network output differs from the desired output.
- It is important to use the correct loss function for the task.
- It is important to use correct activation function for the taks.
- Multiple loss/activation pairs can usually be used for a single task, but choice affects what the network learns. (You change the tasks definition.)

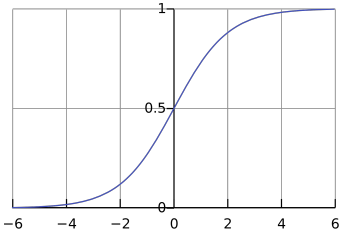
Binary classification

Goal: get p(c | data)

Network output: p(c=1|data)

Binary problem: p(c=2|data) = 1 - p(c=1|data)

Activation function: sigmoid



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

What loss function?

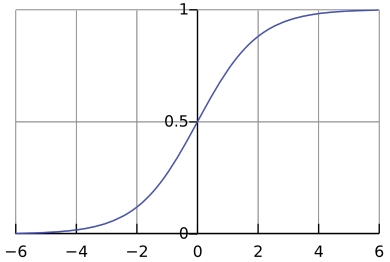
Binary classification (same as in logistic regression)

Goal: get p(c | data)

Network output: p(c=0|data)

Binary problem: p(c=1|data) = 1 - p(c=0|data)

Activation function: sigmoid



Task maximize training set likelihood:

$$P(t|D) = \prod_{D} P(t|x_i) = \prod_{D} f(x_i)^{t_i}(1-f(x_i))^{1-t_i}$$

Minimize binary cross-entropy loss (log-likelihood):

$$J(f, D) = \sum_{i \in D} -\ln(P(t|x_i)) =$$

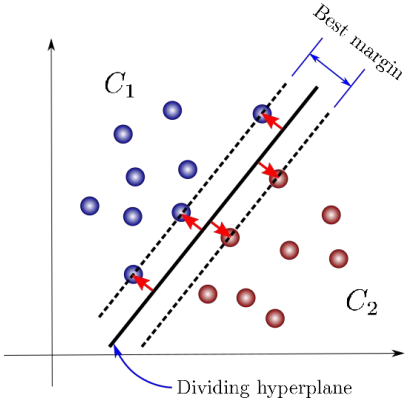
$$\sum_{i \in D} t_i \ln(f(x_i)) + (1 - t_i) \ln(1 - f(x_i))$$

Binary classification - SVM loss

Goal:

f(x) > 1 if c = 1

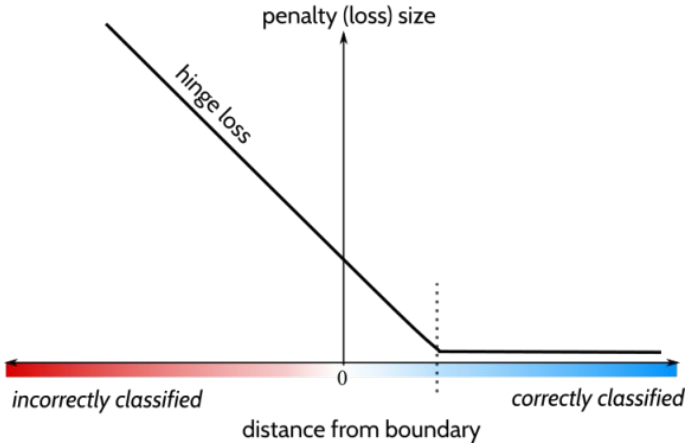
f(x) < -1 if c = -1



Activation function: linear (none)

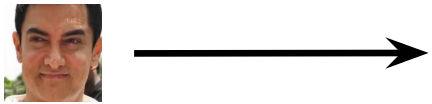
$$J(f, D) = \sum_{i \in D} \max(0, 1 - c_i f(\mathrm{x}_i))$$

Loss function: Hinge loss



Multi-class classification

Goal: get p(c | data) with number of classes > 2



0.1	Michal
0.0	Martin
0.5	Petra
0.3	Katka
0.1	Honza
0.0	Bětka

Activation function: soft-max

$$\sigma_i(\mathbf{z}) = \frac{e^{\mathbf{z_i}}}{\sum_j e^{\mathbf{z_j}}}$$

Multi label classification

Goal: get p(c | data) for multiple binary problems

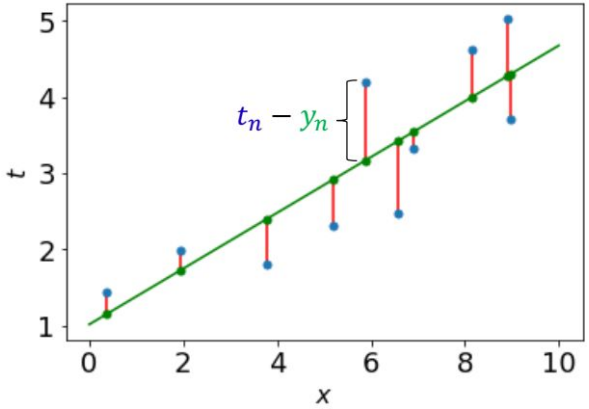
0.1	Asian
1.0	Male
0.9	Black hair
0.3	Actor
0.2	Child
1.0	Photographed outdoors

Activation function: sigmoid

Loss: Binary cross-entropy

Regression

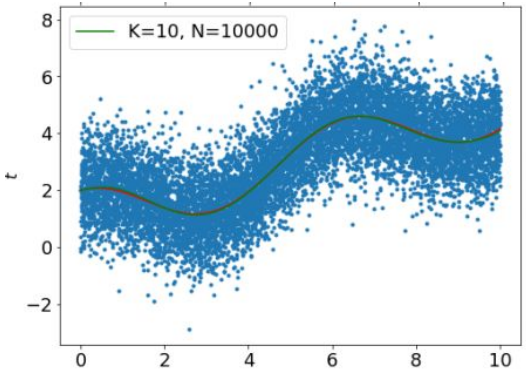
Goal: get E[p(value | data)] + gaussian noise in data



Activation: Linear (none)

Loss: Mean squared error

$$\sum_{i \in D} \frac{1}{2}(f(x_i) - t_i)^2$$



Regression

Goal: get E[p(value | data)] + lot of non-gaussian outliers

Activation: Linear (none)

Loss: Sum of Absolute Difference

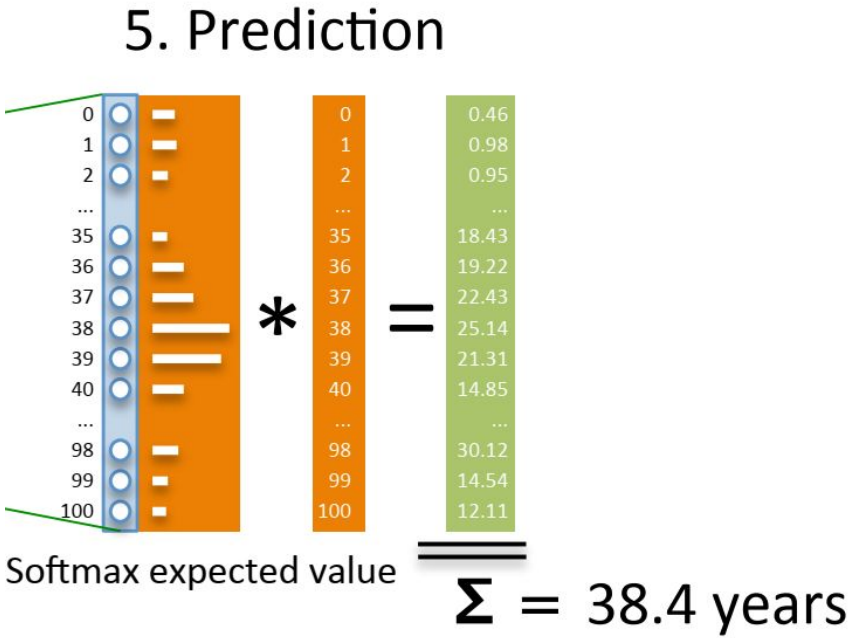
$$\sum_{i \in D} |f(x_i) - t_i|$$

Regression - with discretization

Goal: get p(value | data)

Activation: Softmax

Loss: Cross-entropy

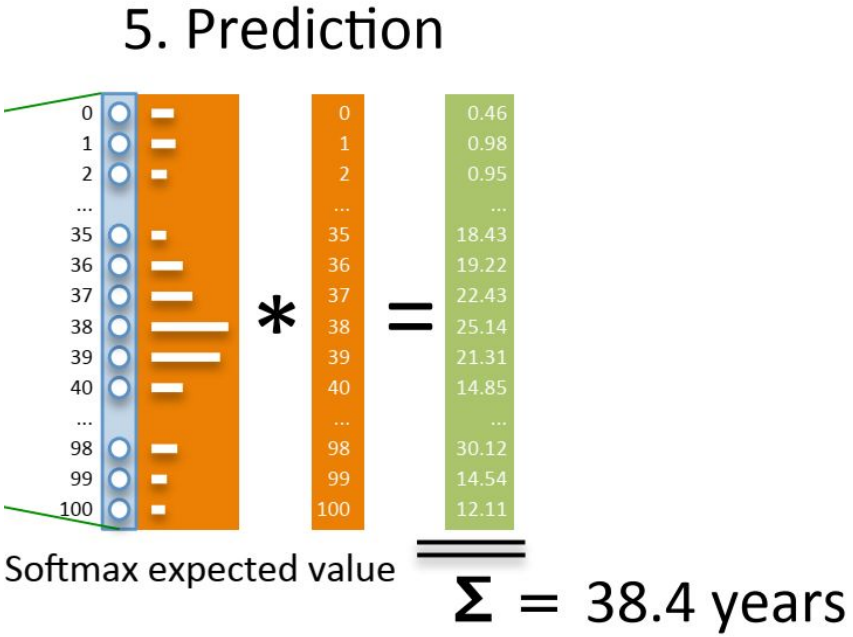


Regression - with discretization

Goal: get E(p(value | data))

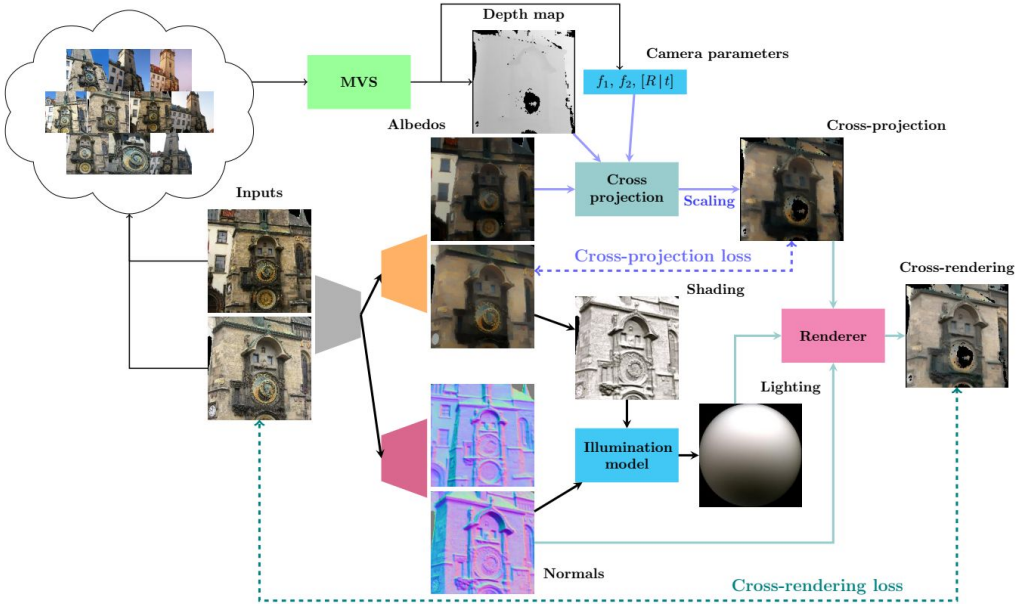
Activation: Softmax (with expectation sum)

Loss: Mean squared error?



Weird loss functions

We train our network to minimise: $\ell = w_1\ell_{\text{appearance}} + w_2\ell_{\text{NM}} + w_3\ell_{\text{albedo}} + w_4\ell_{\text{cross-rend}} + w_5\ell_{\text{albedo-smooth}} + w_6\ell_{\text{albedo-pseudoSup}}$.



Yu, Ye and Smith, William A. P., InverseRenderNet: Learning single image inverse rendering, CVPR, 2019.

Training of neural networks

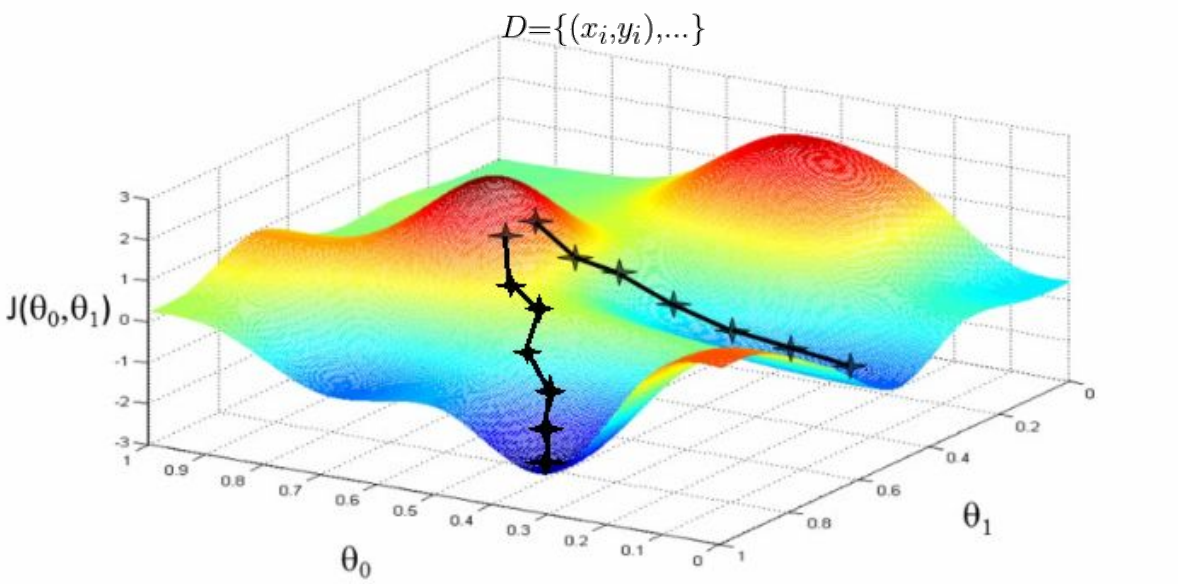
$$J(D,\theta) = \sum_{D=\{(x_i,y_i),\ldots\}} loss\left(f\left(x_i,\theta\right),y_i\right)$$

$$\theta^* = arg\min_{\theta} J(\mathbf{D},\theta)$$

Gradient Descent

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(D,\theta)}{\partial \theta_j}$$

$$J(D,\theta) = \sum_{D=\{(x_i,y_i),\ldots\}} loss\left(f\left(x_i,\theta\right),y_i\right)$$

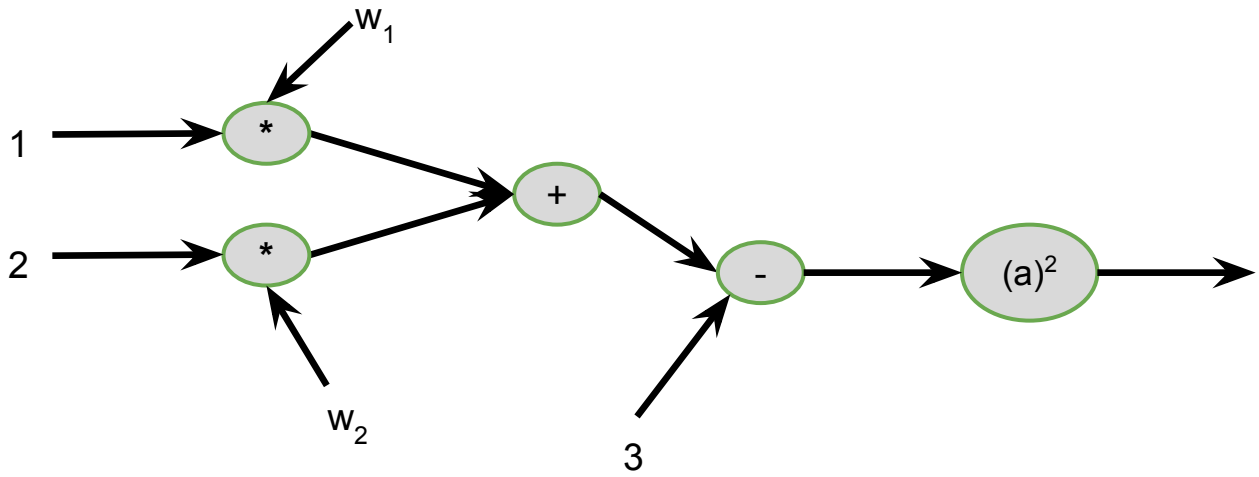


Gradients for training - chain rule

Need $\nabla_w J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}\right]$

$$\frac{\partial (w_1 + 2w_2 - 3)^2}{\partial w_1} = \frac{\partial (a)^2}{\partial a}\frac{\partial (b-3)}{\partial b}\frac{\partial (c + 2w_2)}{\partial c}\frac{\partial (1 * w_1)}{\partial w_1}$$

$$\frac{\partial (w_1 + 2w_2 - 3)^2}{\partial w_2} = \frac{\partial (a)^2}{\partial a}\frac{\partial (b-3)}{\partial b}\frac{\partial (w_1 + c)}{\partial c}\frac{\partial (2 * w_2)}{\partial w_2}$$

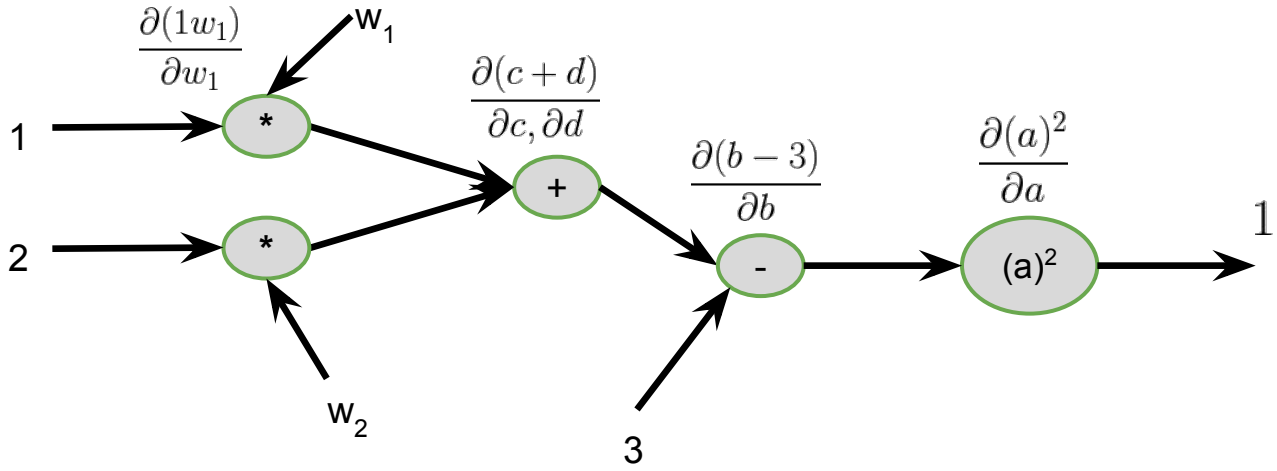


Gradient for training

Need $\nabla_w J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}\right]$

$$\frac{\partial(1w_1 + 2w_2 - 3)^2}{\partial w_1} = \frac{\partial(a)^2}{\partial a}\frac{\partial(b-3)}{\partial b}\frac{\partial(c+2w_2)}{\partial c}\frac{\partial(1*w_1)}{\partial w_1}$$

$$\frac{\partial(1w_1 + 2w_2 - 3)^2}{\partial w_2} = \frac{\partial(a)^2}{\partial a}\frac{\partial(b-3)}{\partial b}\frac{\partial(1w_1+c)}{\partial c}\frac{\partial(2*w_2)}{\partial w_2}$$



Full training

- Forward pass

$$\mathrm{x}_2 = max(\mathrm{W}_1\mathrm{x}_1, 0)$$
$$\mathrm{x}_3 = max(\mathrm{W}_2\mathrm{x}_2, 0)$$
$$\mathrm{x}_4 = max(\mathrm{W}_3\mathrm{x}_3, 0)$$

- Compute loss

$$loss = (\mathrm{x}_4 - 1)^2$$

- Backward pass
  - Compute derivatives of objective fce. with respect to net. parameters using chain rule / backpropagation

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(D, \theta)}{\partial \theta_j}$$

- Update parameters

Pytorch introduction / basics of tensors

https://colab.research.google.com/drive/1ytNsahwdl4FJDkXmaw7CCQQJqQDiAxJt?usp=sharing

The same as Numy with

- Automatic differentiation
- GPU support
- Neural network layers
- Loss functions
- Optimization algorithms for neural networks
- Datasets, data augmentation, predefined networks, pretrained network, ...

Stochastic/mini-batch gradient descent

- Large datasets -> too much computation per network update -> inefficient training
- Estimate gradient on a small random part of the training set (mini-batch) and update network
- Stochastic Gradient Descent

$$J(D,\theta) = \sum_{D=\{(x_i,y_i),\ldots\}} loss\left(f\left(x_i,\theta\right),y_i\right)$$

mini-batch=  Gives approximate $\nabla_w J$

D=  Gives exact $\nabla_w J$

Learning rate and adaptive algorithms

- High learning rate -> divergence
- Low learning rate -> slow training
- For SGD, optimal learning rate depends on
  - Network
  - Objective, consider gradients for J() and 10000*J()
- You have to try different learning rates

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial J(D, \theta)}{\partial \theta_j}$$

- Solution:
  - Adaptive GD algorithms which normalize learning rate
  - Adam, AdaDelta, ...

Train good model and guess real-world performance

- Training set
  - Train models on this set
  - In domain data, similar problems, synthetic, generated, augmented
- Validation set
  - Evaluate models, tune hyperparameters, choose alg. and architecture
  - Same (iid) as training set?
- Test set
  - Estimate real-world performance
  - Data ideally from real world
- Loss is usually different than the target metric
  - Loss can improve while “accuracy” gets worse

Review

- What is a neural network?
- What are “trainable/learnable” parameters?
- What is loss function?
- What is gradient?
- GD and SGD?
- What is a training set in supervised learning?
- Shallow learning vs deep learning?
- Linear neuron and matrix multiplication.
- Non-linear activations.
- Layers.
- Computational graph.

Review

- Basic loss functions for classification and regression.
- What is training - objective function minimization.
- Backward pass and gradient descent rule.
- SGD and its benefits.
- What are train set, validation set and test set?

Neural networks for structured data

Local cues

Local cues

Image representations

Layers for images?

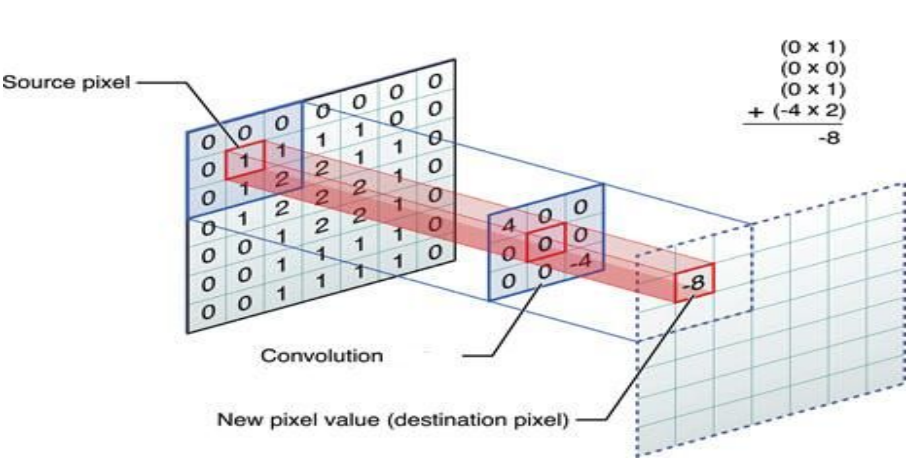
- Which mathematical operation fits?
  - Is local
  - Processes each part of input the same way
  - Is linear
  - Is the building block of neural networks for images, sound, ...

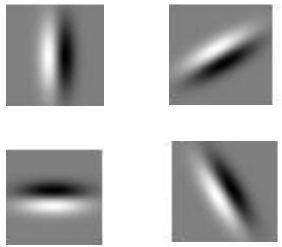
History of Convolutional Networks

- **1980** – Kunihiko Fukushima – **Neocognitron**: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position
- **1986** – Hinton, Rumelhart, Williams, McClelland - **Backpropagation**
- **1998** – LeCun et al.: Gradient-Based Learning Applied to Document Recognition
- **2012** – Krizhevsky et al.: ImageNet Classification with Deep Convolutional Neural Networks
- **2013-today** – explosion of applications

Convolution

- Linear operation
  - (matrix multiplication on small neighborhood)
- Output: Map of local similarity of input and conv. kernel





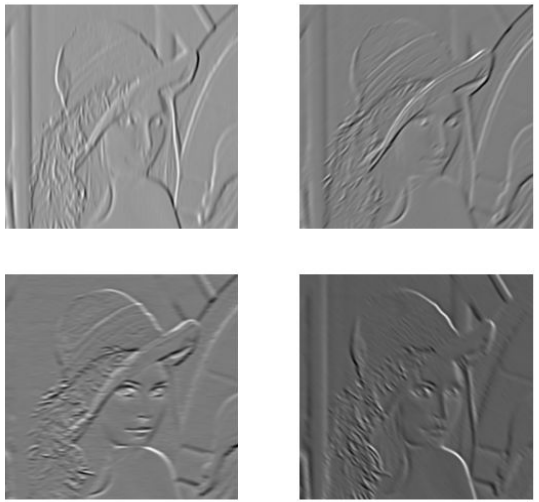
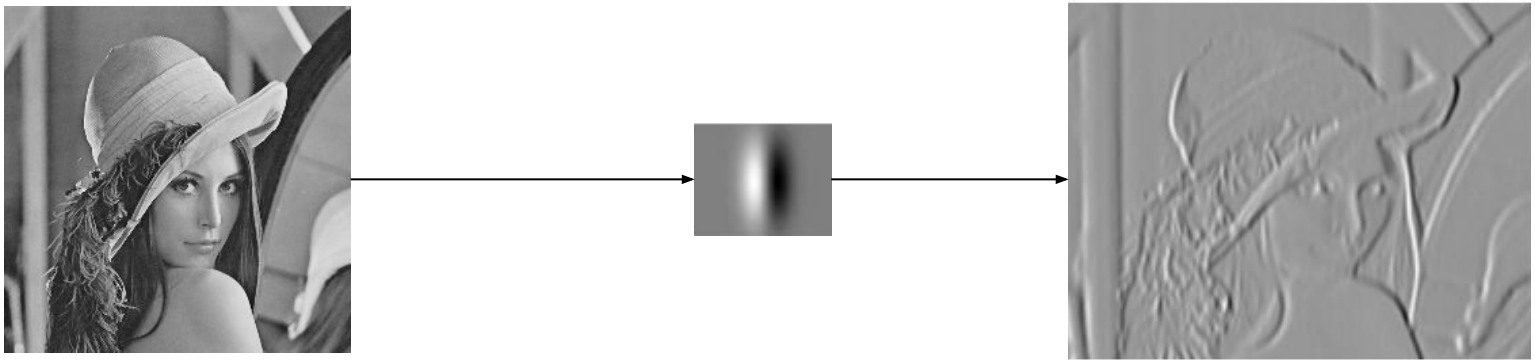


Image from:Basavarajaiah. 6 basic things to know about Convolution. https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Convolutional layer



Convolutional layer

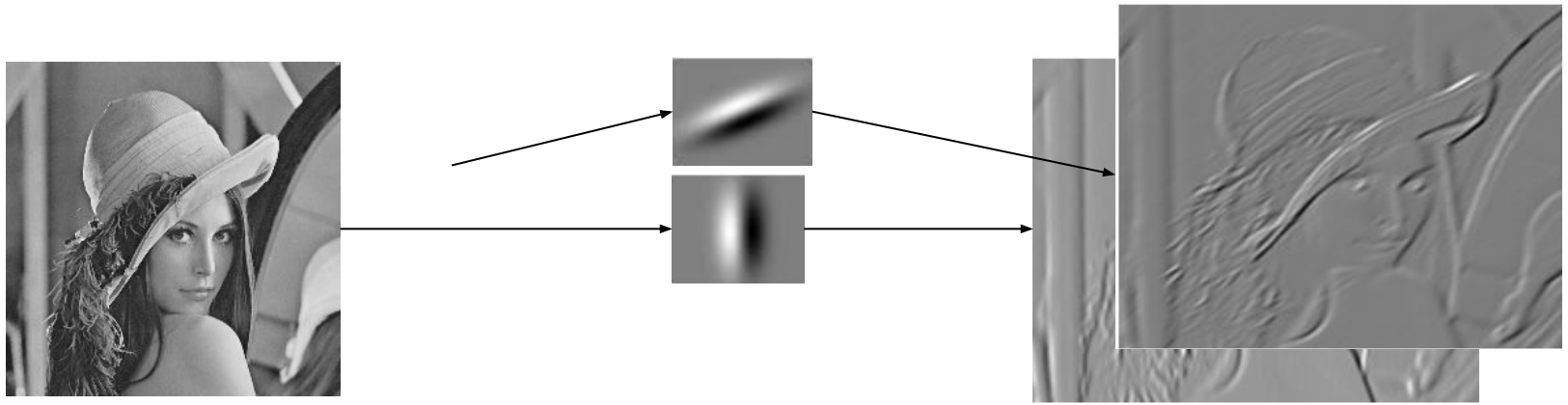


Image from:Basavarajaiah. 6 basic things to know about Convolution. https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Convolutional layer

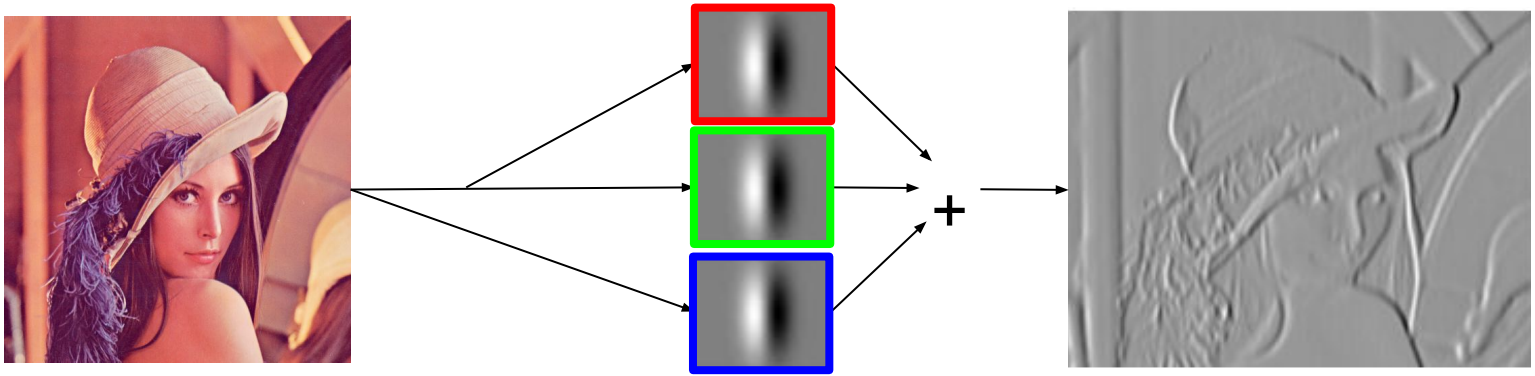
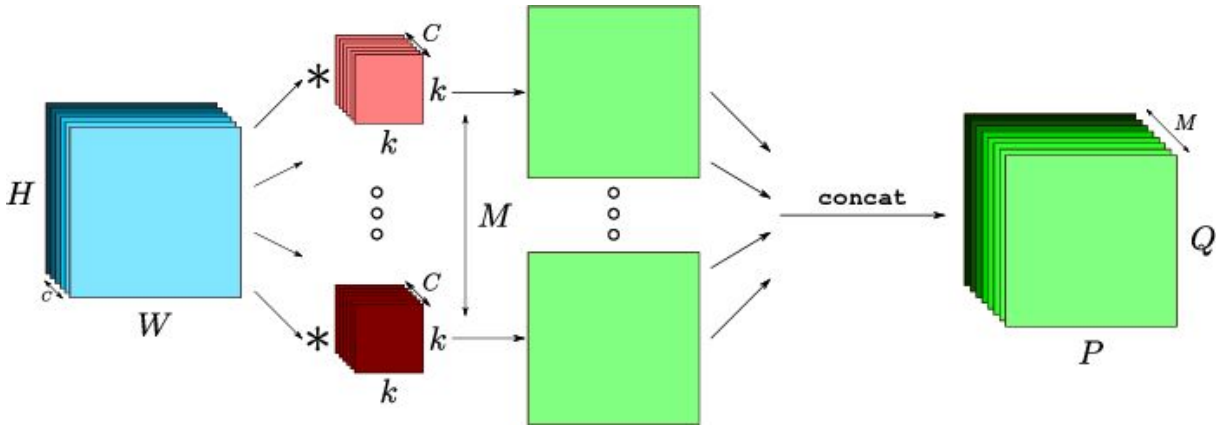


Image from:Basavarajaiah. 6 basic things to know about Convolution. https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Convolution

- Channels -> activation map



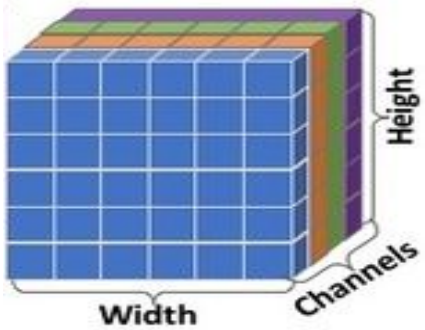


Image from:Basavarajaiah. Vasudevan, Aravind et al. “Parallel Multi Channel convolution using General Matrix Multiplication.” *ASAP.* 2017.

Pooling layer

- Downsampling
  - Local pixels are aggregated into a single pixel
  - Spatial resolution is reduced, channels stay the same
  - Reduction by AVERAGE or MAX functions.
  - MAX-pooling directly encourages translation invariance

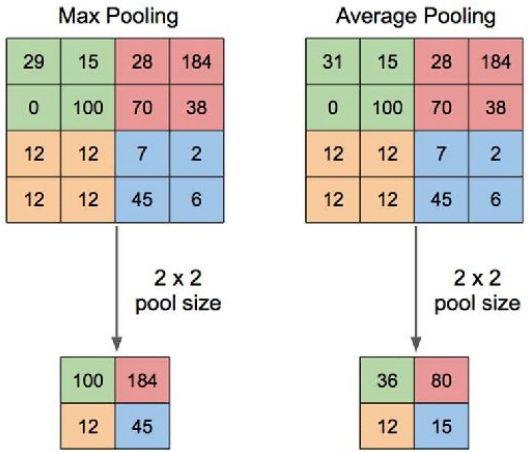
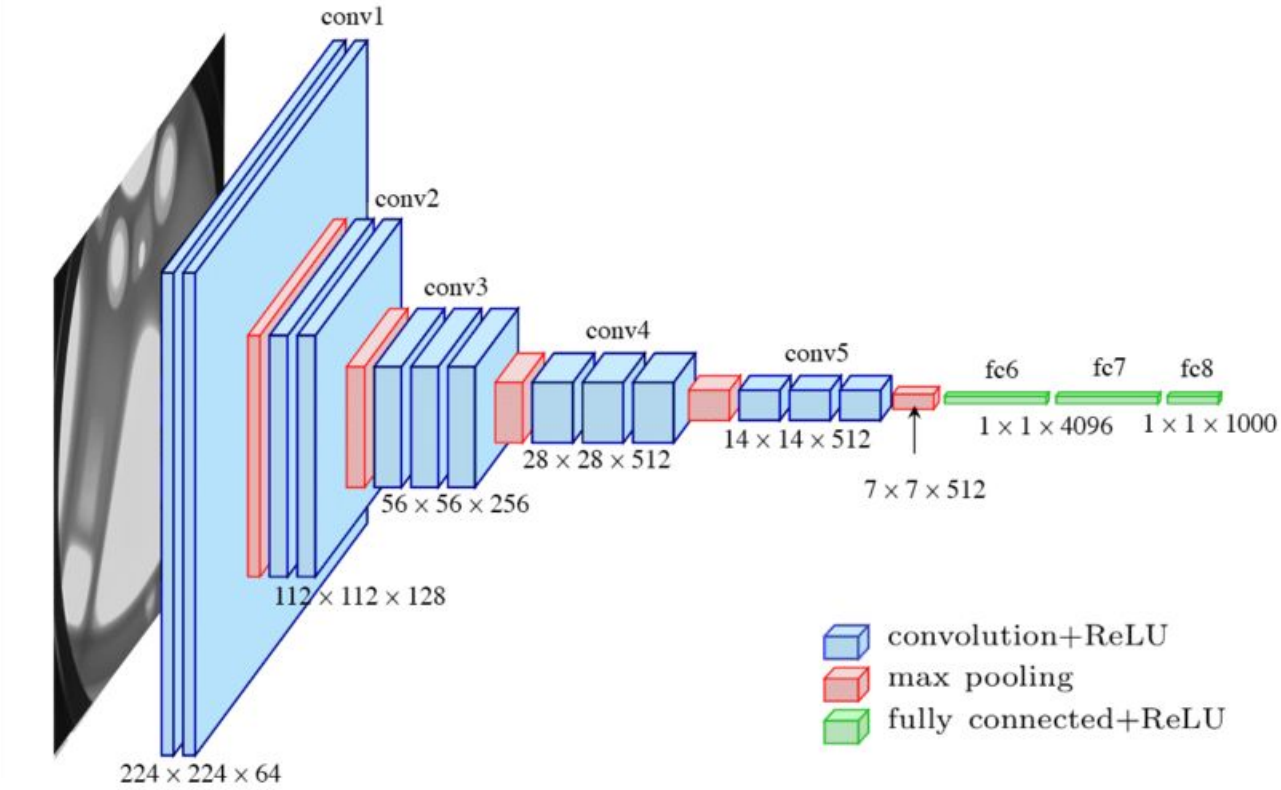


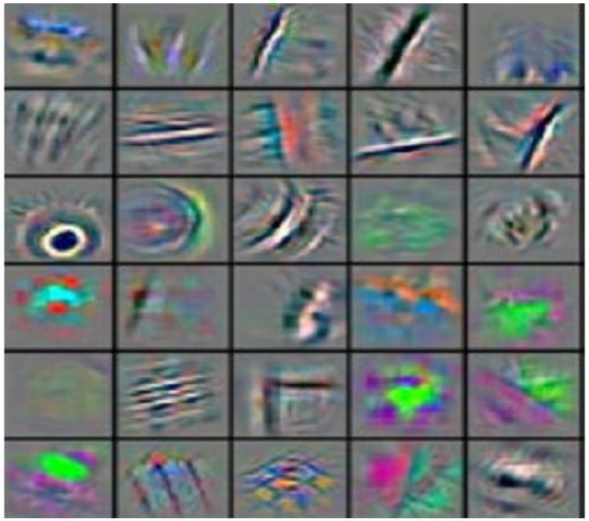
Image from: Yani et al., Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. J. Phys. Conf. 2019.

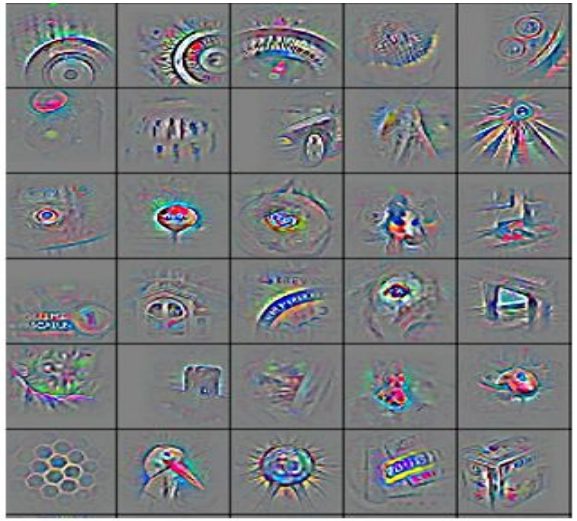
Network architecture (VGG net)



Hierarchy of features

(and human visual system analogy)



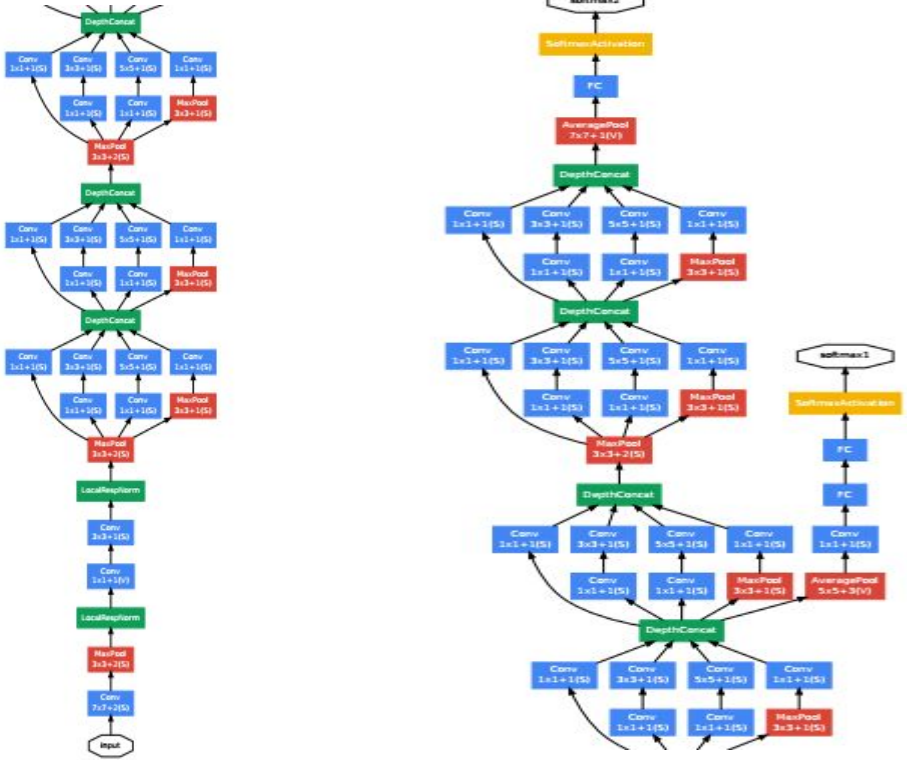


Very Deep Convolutional Networks

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Karen Simonyan, Andrew Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition,

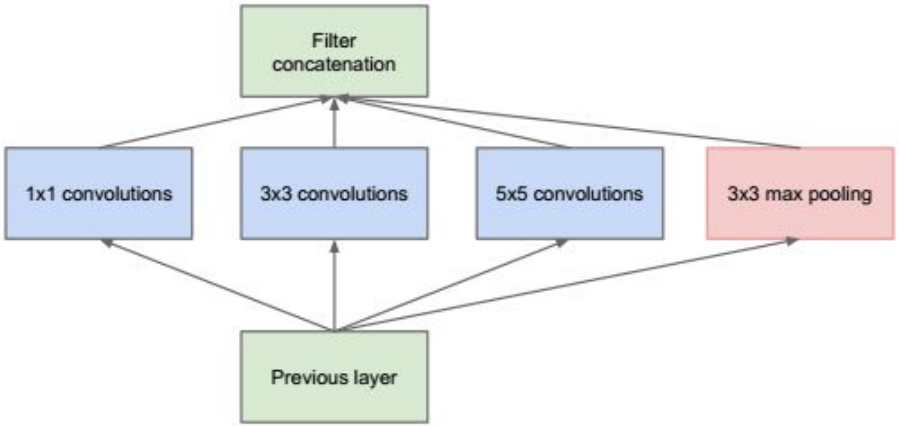
GoogLeNet

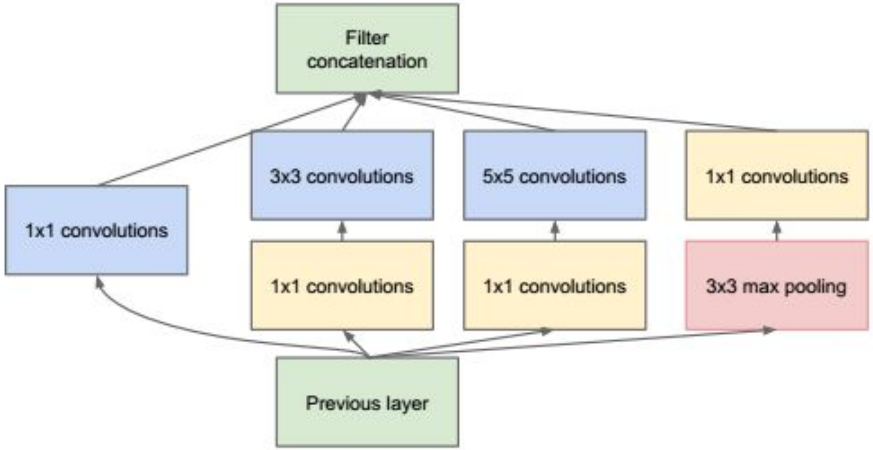


Szegedy et al.: Going deeper with convolutions

GoogLeNet - Inception Layer

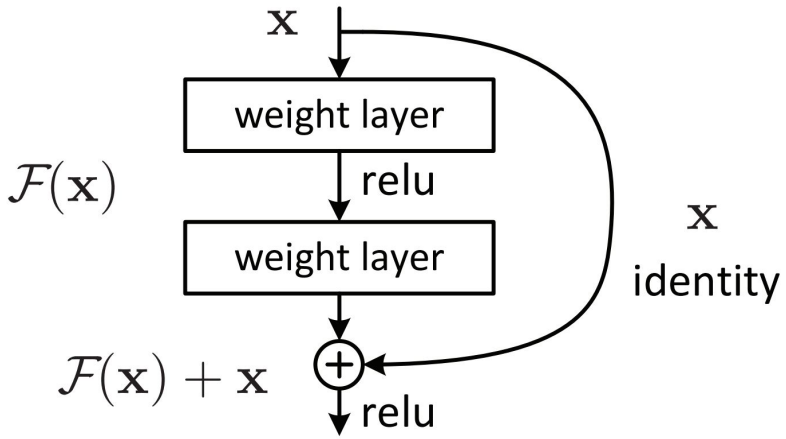
- Winner of ILSVRC 2014

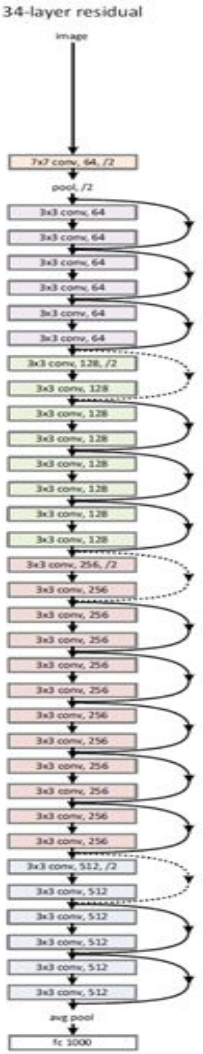




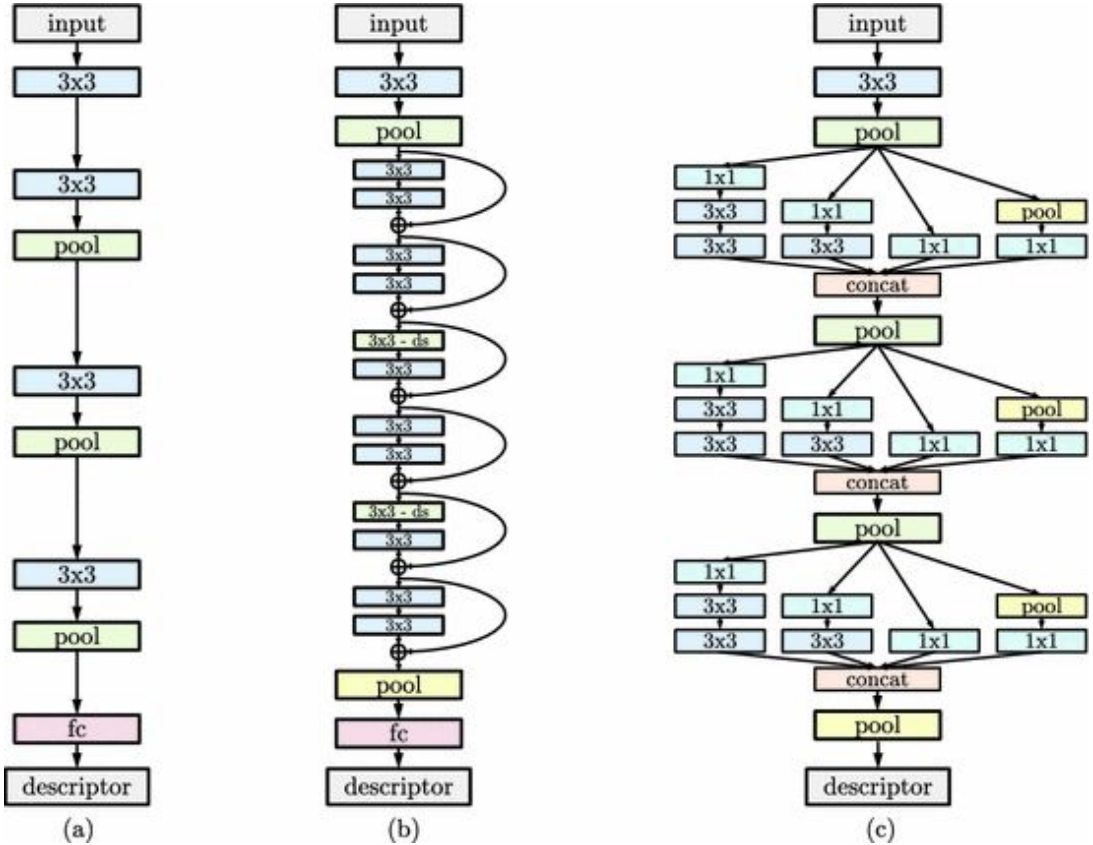
Szegedy et al.: Going deeper with convolutions

Residual networks





Combinations



Generalization

- **Good model structure**
- Multi-task learning
- Pre-training
- Domain adaptation
- More data - augmentation, synthetic data
- Model size

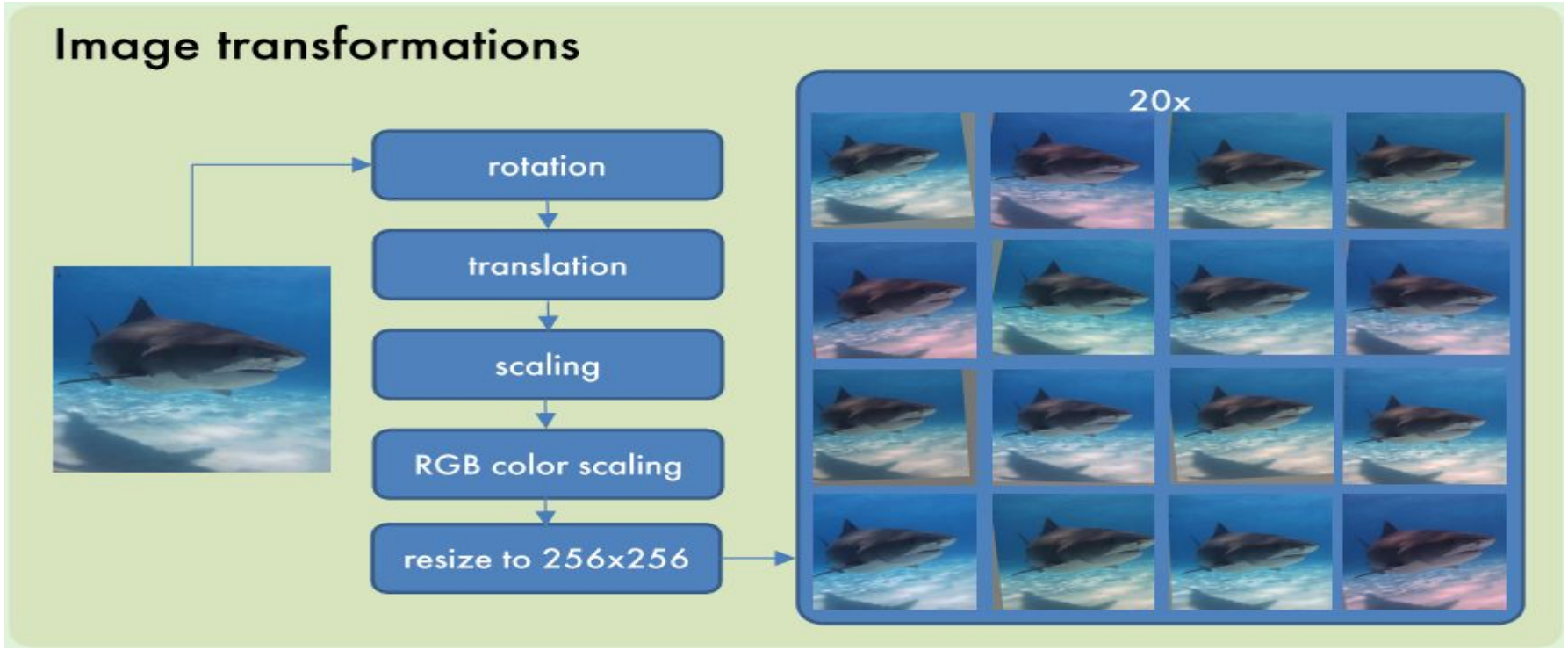
Regularization and overfitting

- Deep convolutional networks overfit surprisingly late
- Early stopping (when training)
- Weight regularization (weight decay)
  - does not help much (with ReLU)
- Activation regularization

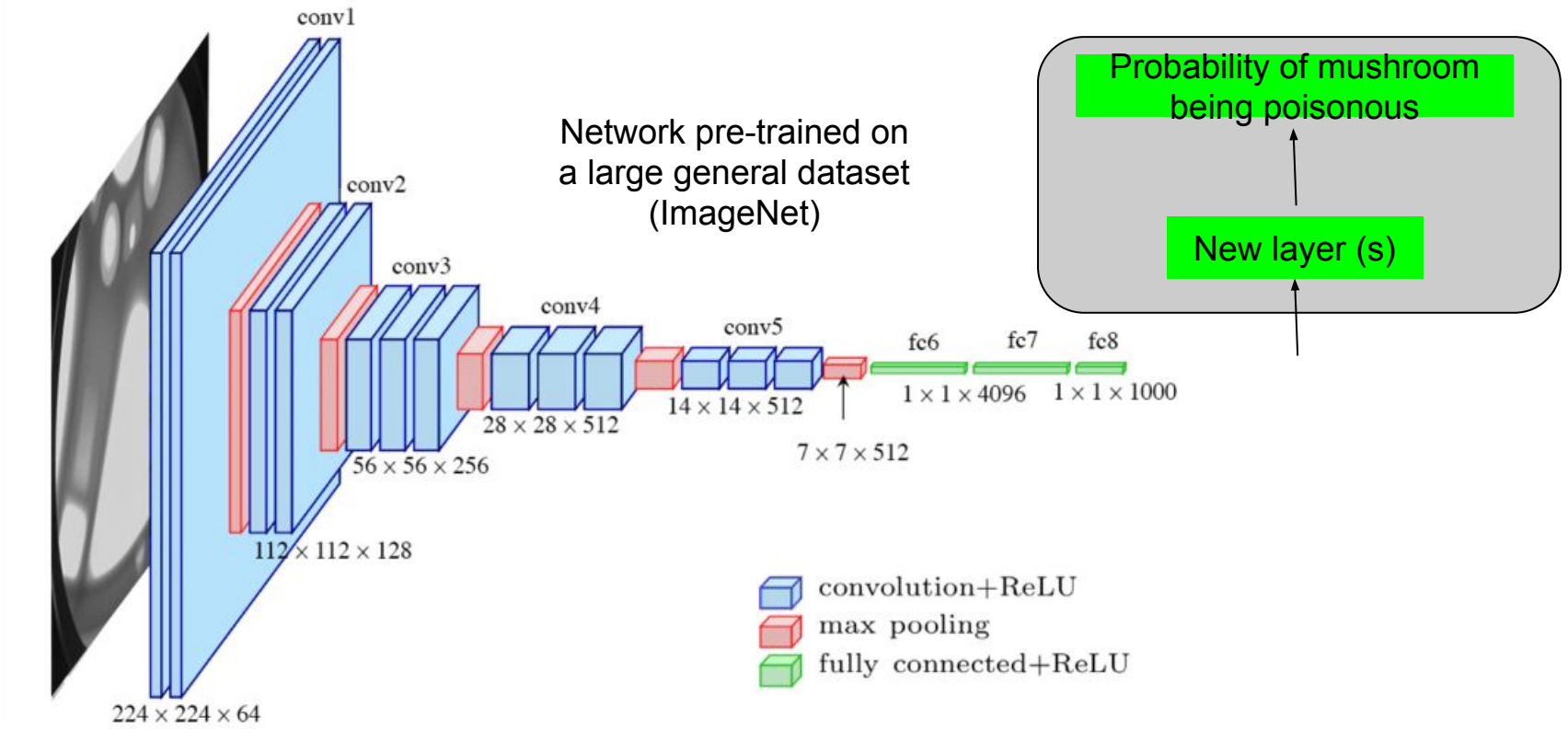
$$J(D,\theta) = \left(\sum_{D} loss(f_\theta(x_i), y_i) \right) + \lambda ||\theta||^2 + \beta ||activation||^2$$

- **Dropout** helps a lot
  - Sets random activations to 0

Data augmentation



Finetuning

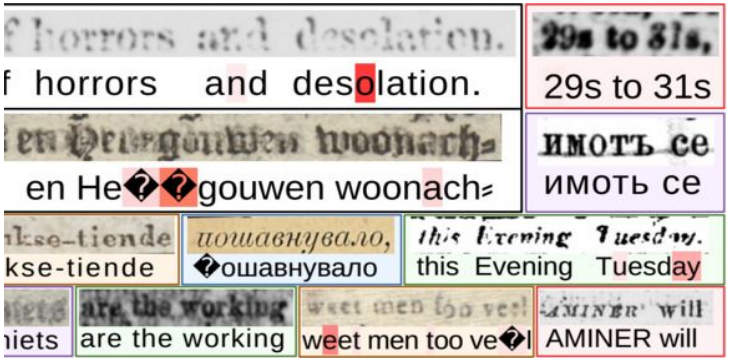

conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
Network pre-trained on a large general dataset (ImageNet)
Probability of mushroom being poisonous
New layer (s)
convolution+ReLU
max pooling
fully connected+ReLU

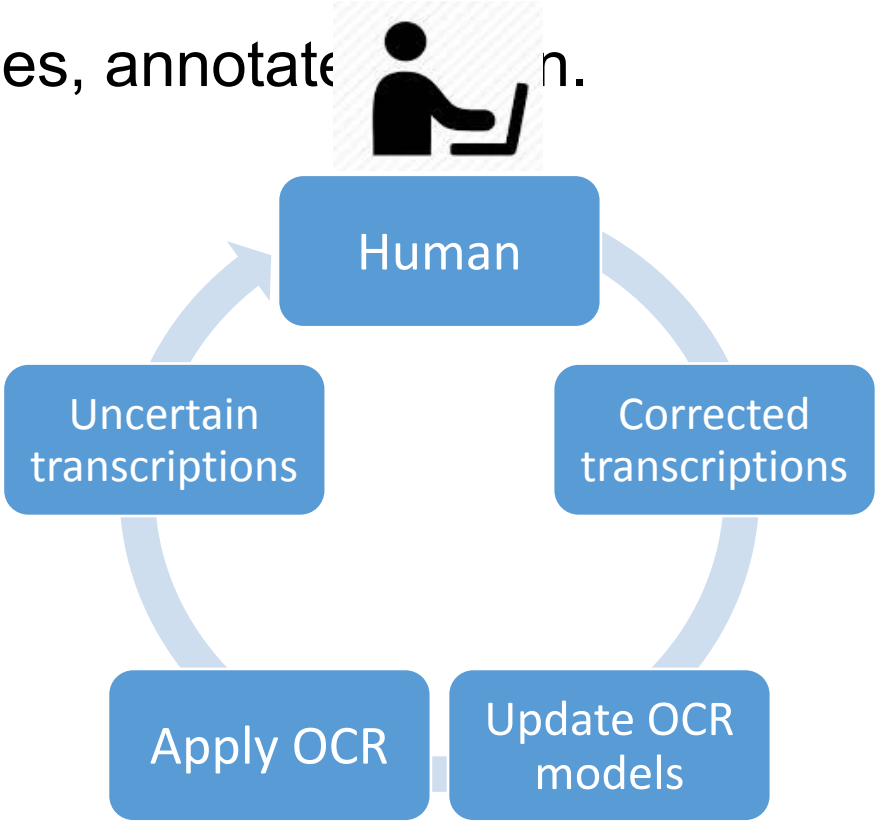
Active learning

Make annotations effective.

Use model to select hard examples, annotate n.

Use prediction confidence.





Weird learning

Semi supervised learning - annotated and unannotated data

Active learning - pick good data to annotate

Self-supervised learning - automatically generate labels (mostly pre-training)

Transfer learning - network for one task is fine-tuned on another task

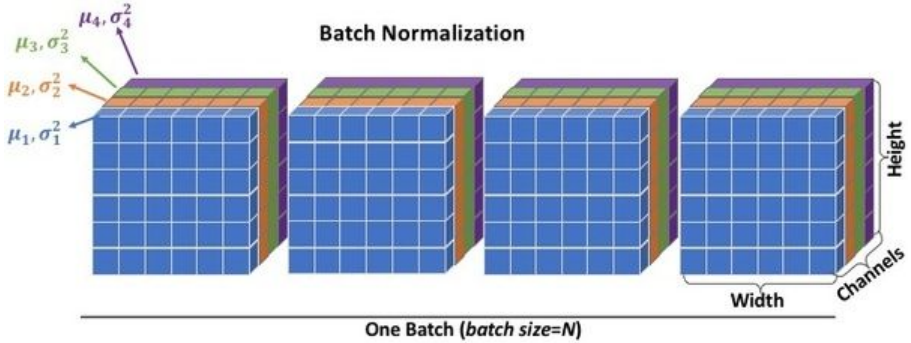
Knowledge distillation - usually a good large model generates “labels” for smaller one

Domain adaptation - the task remains the same, but data changes (day/night)

Self training - model generates labels and is trained on them (semi-supervised)

Federated learning - distributed learning on private data

Batch normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

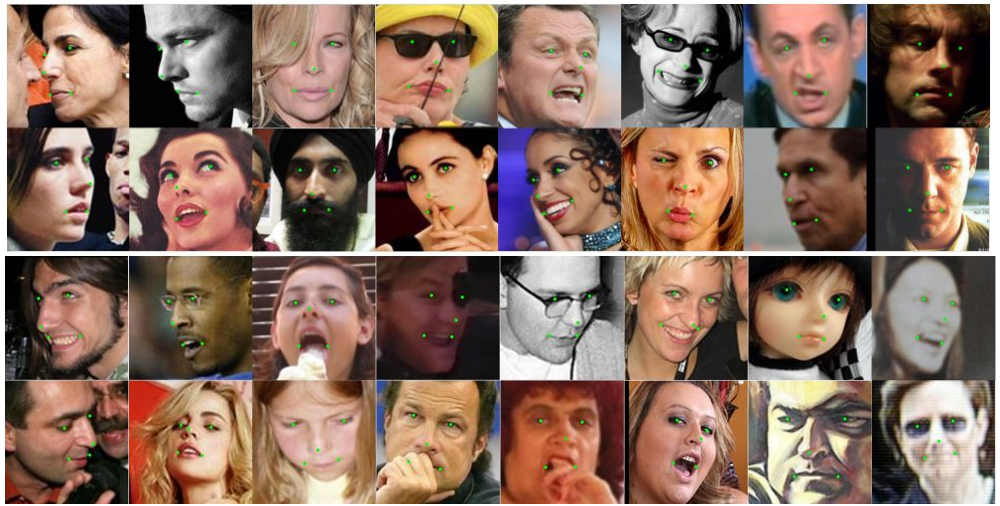
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe, Szegedy: Batch normalization: accelerating deep network training by reducing internal covariate shift, ICML, 2015.

Facial alignment

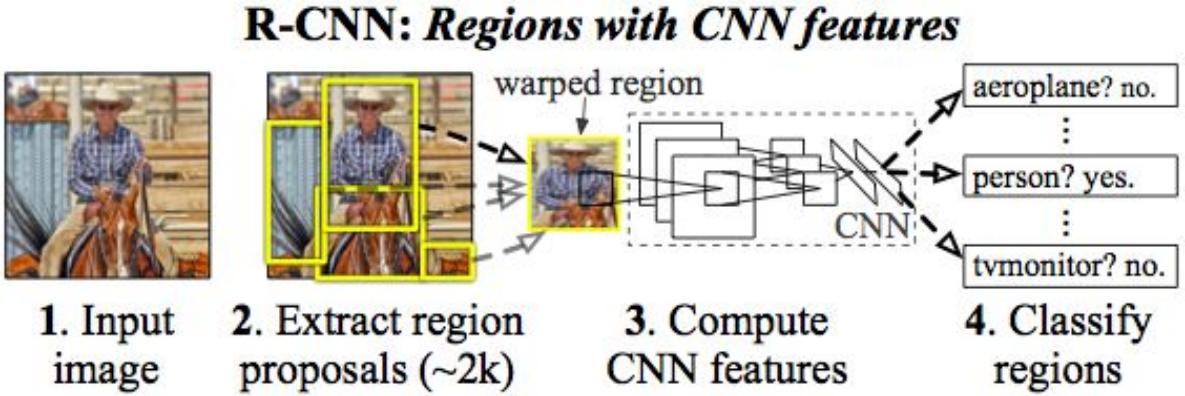
- Input - Facial crop
- Output: x,y coordinates -> regression loss (e.g. MSE)



Sun et al.: Deep Convolutional Network Cascade for Facial Point Detection, CVPR 2013

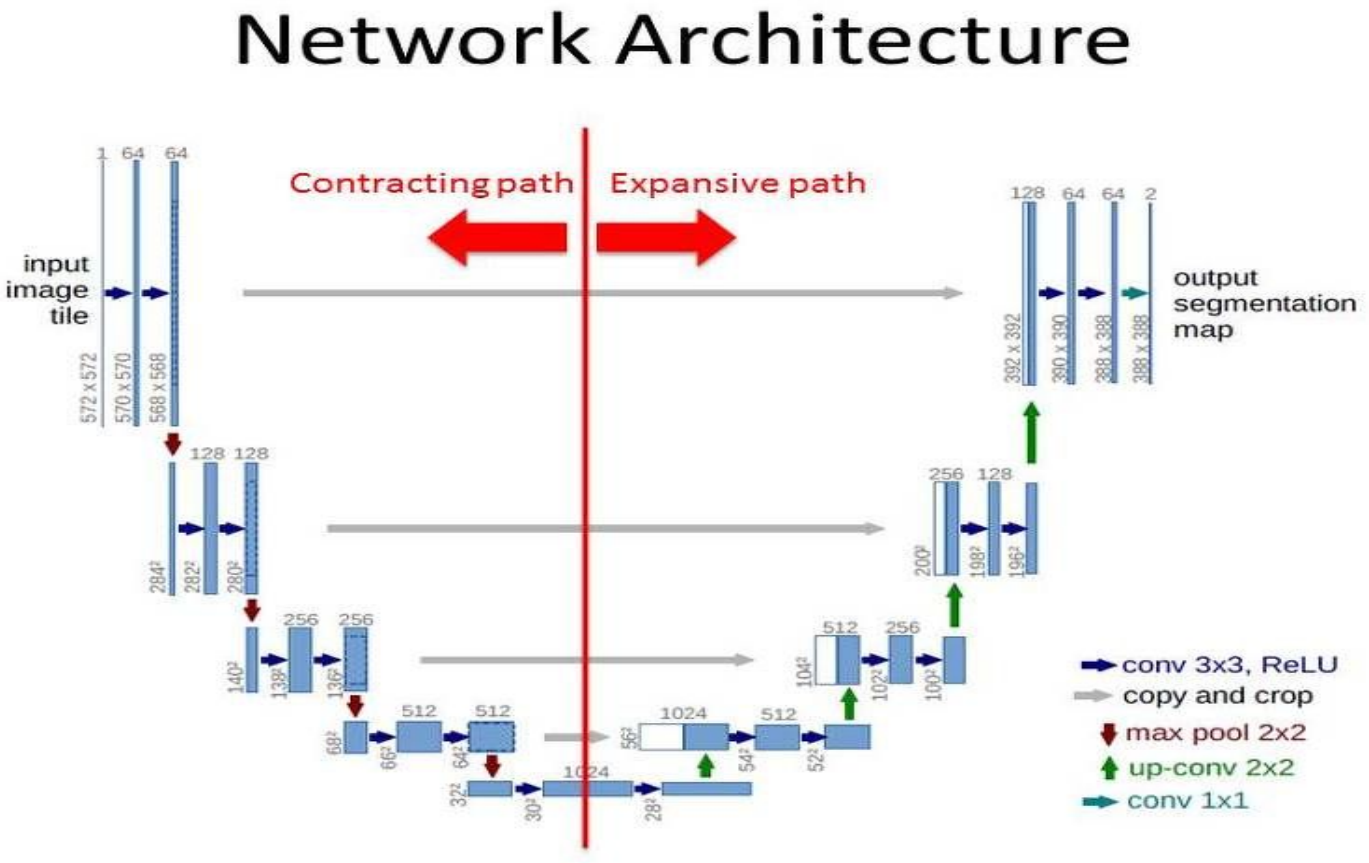
Object detection

- R-CNN



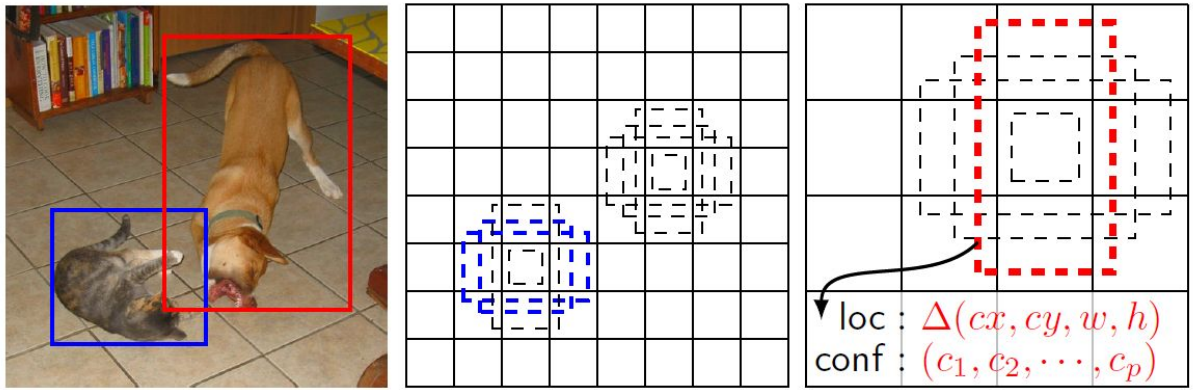
Girshick, R., Donahue, J., Darrell, T., & Malik, J., "Rich feature hierarchies for accurate object detection and semantic segmentation"

Fully Convolutional Networks: FCN



Convolutional object detection

- Output per pixel:
  - For each anchor (base bounding box)
  - Object class (e.g. softmax)

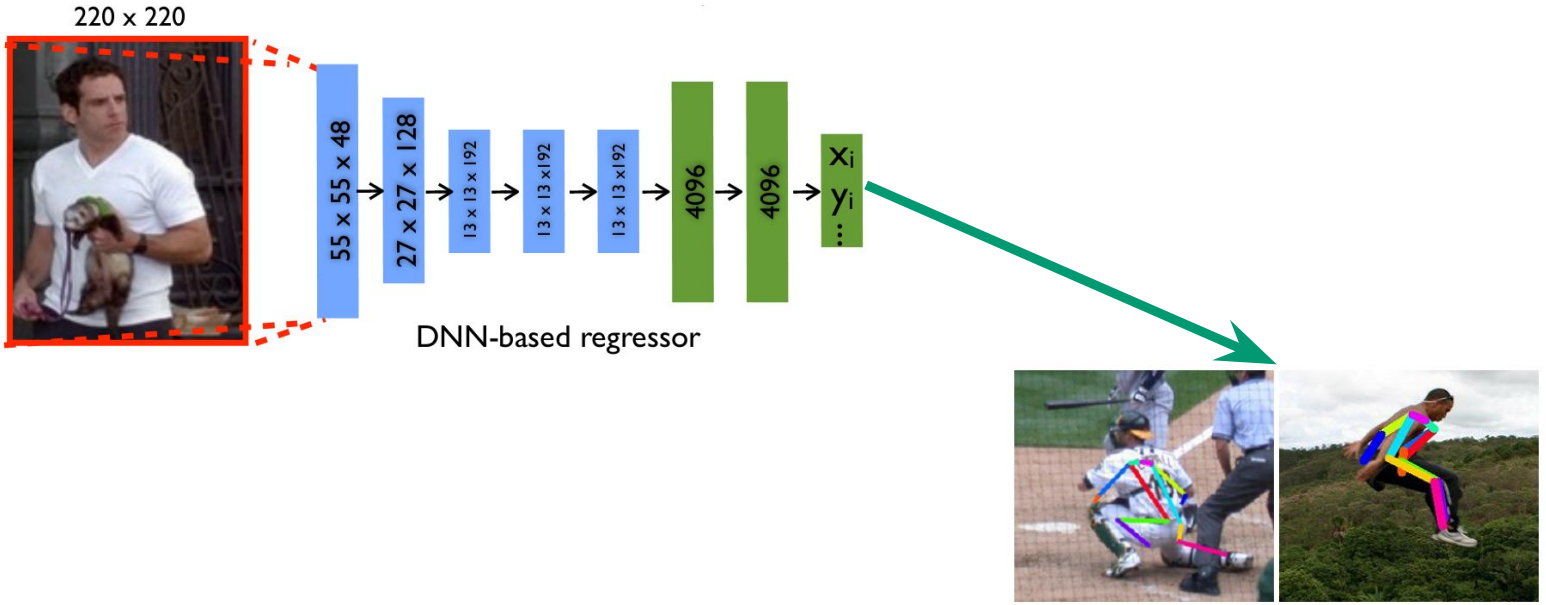


(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

ssion x, y, w, h)

Liu et al.: SSD: Single Shot MultiBox Detector, ECCV 2016.

Pose estimation



Toshev et al.: DeepPose: Human Pose Estimation via Deep Neural Networks. CVPR 2014