



Introduction to Deep Learning

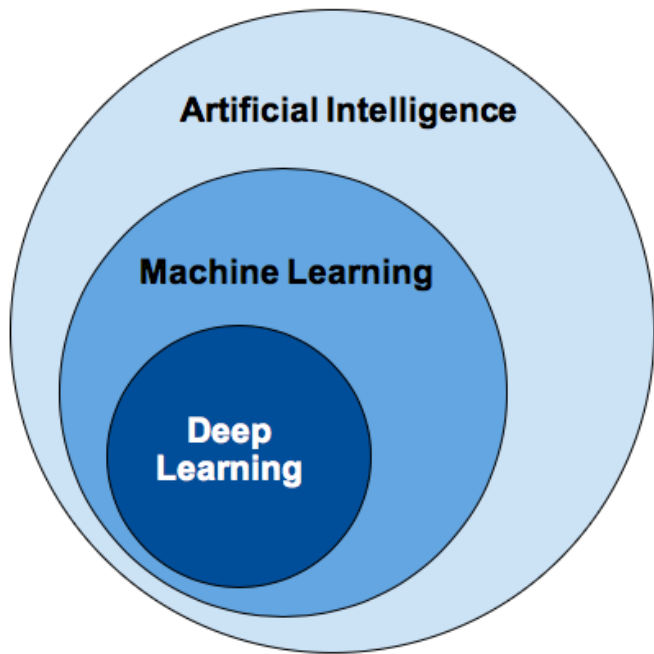
Dmitry Sidnev

Deep Learning R&D Engineer at Intel

Internet of Things Group



Machine Learning & Deep Learning. Definition



Machine learning:

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ”

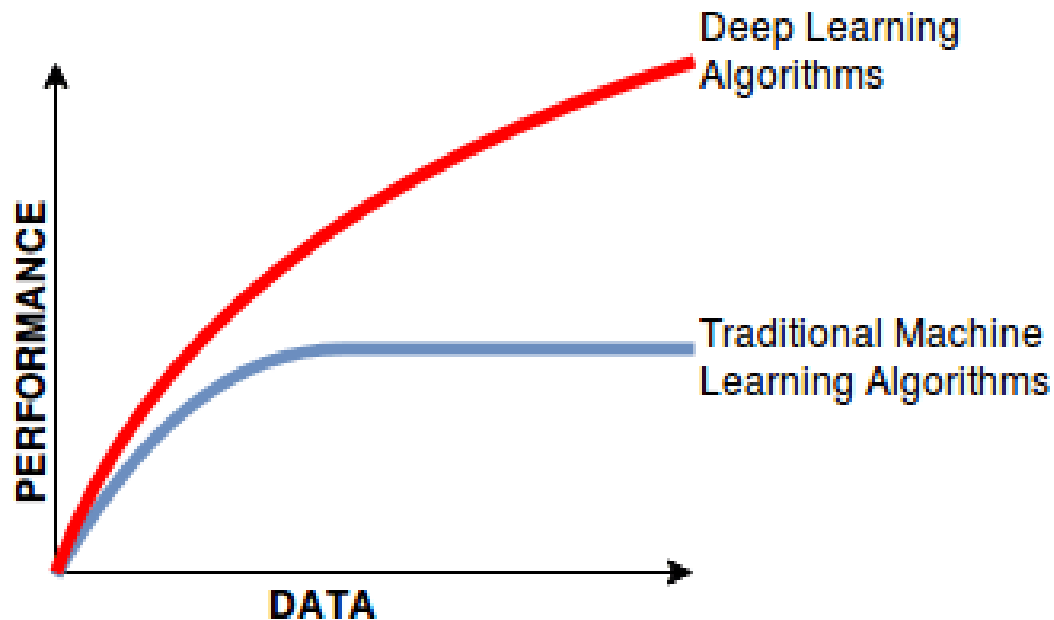
(Mitchell T.M. Machine Learning, 1 edition, New York, NY, USA: McGraw-Hill, Inc., 1997)

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called **artificial neural networks**.

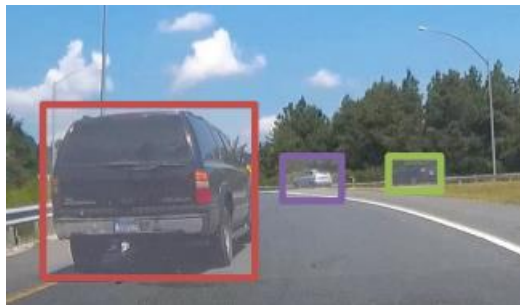
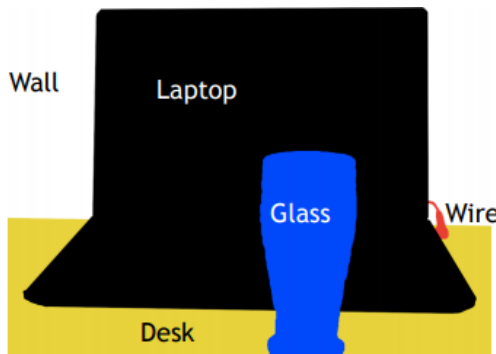
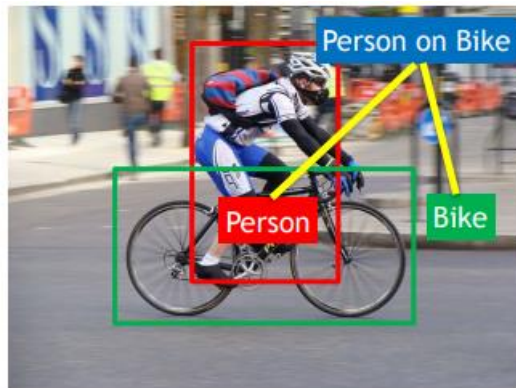
Machine Learning vs Deep Learning

Feature Extraction:

- **Machine Learning** algorithms contain feature engineering phase. In this phase, experts propose the hand-crafted features to facilitate the learning from examples
- **Deep Learning** pipeline: feature extraction is embedded in the learning algorithm where features are extracted in a fully automated way and without any intervention of a human expert



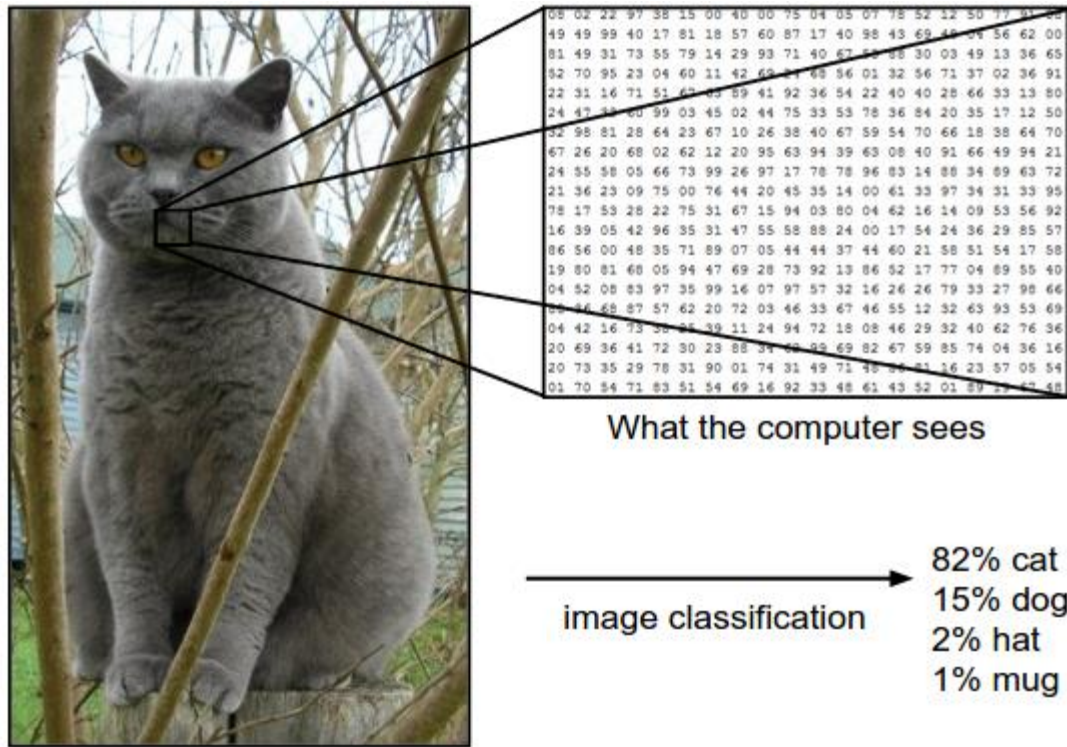
Deep Learning in CV



Deep Learning today successfully solves various tasks in the field of computer vision:

- Image classification
- Object detection
- Object segmentation
- Action recognition
- e.t.c.

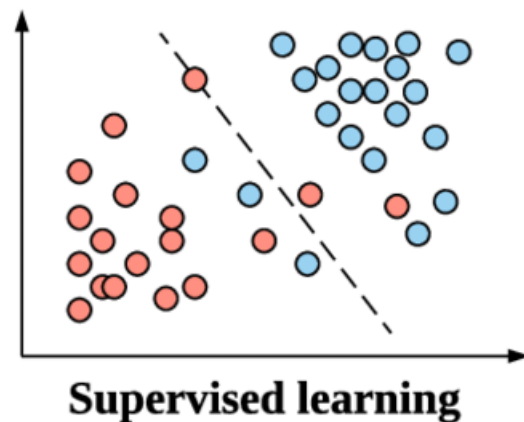
Image classification



- The task is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image.
- Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three color channels Red, Green, Blue

Recap: Supervised Learning

- Space of viable input values \mathcal{X} .
- Space of viable output (target) values \mathcal{Y} .
- Dataset $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$, $|\mathcal{D}| = N$.
- Find a function (a.k.a. model) $h: \mathcal{X} \rightarrow \mathcal{Y}$ that is a good predictor of $y \in \mathcal{Y}$ given $x \in \mathcal{X}$.

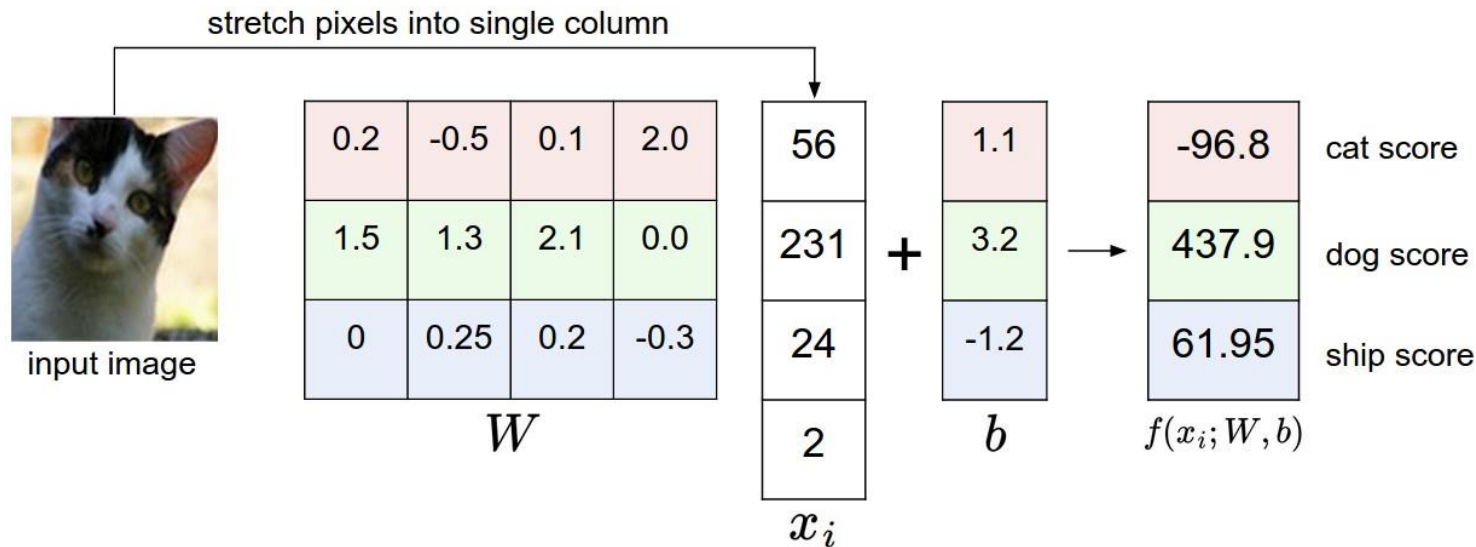


Examples: SVM, k-NN, Random Forest, Logistic Regression, Neural Networks etc.

Linear Classification

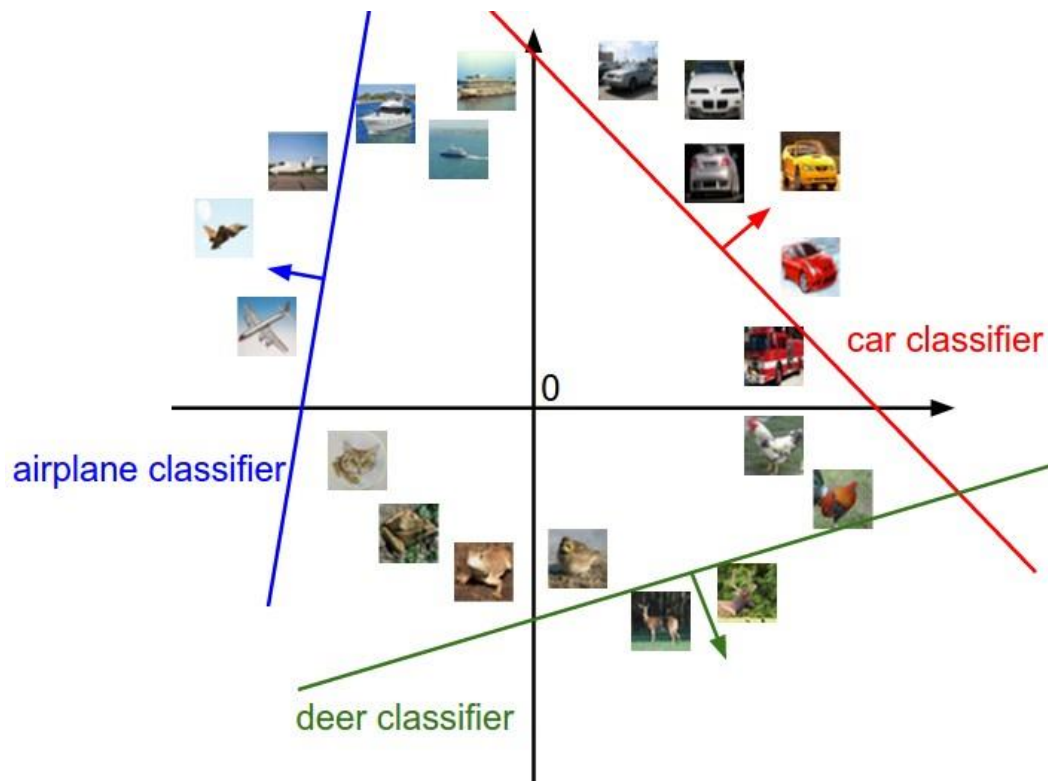
- Let's assume a training dataset of images $x_i \in R^D$, each associated with a label y_i (a.k.a **ground truth**)
- $i = 1 \dots N, y_i \in 1 \dots K$, where N is a number of examples and K is a number of classes
 - For example, in CIFAR-10 we have a training set of $N = 50,000$ images, each with $D = 32 \times 32 \times 3 = 3072$ pixels, and $K = 10$
- Score function $f: R^D \rightarrow R^K$ that maps the raw image pixels to class scores
 - The simplest possible function, a linear mapping: $f(x_i, W, b) = Wx_i + b$
 - **W** is called the **weights** (of size $[K \times D]$)
 - **b** is called the **bias vector** (of size $[K \times 1]$)

Interpreting a linear classifier



An example of mapping an image to class scores. For the sake of visualization, we assume the image **only has 4 pixels** (4 monochrome pixels, we are not considering color channels in this example for brevity), and that we have **3 classes** (red (cat), green (dog), blue (ship) class). We stretch the image pixels into a column and perform matrix multiplication to get the scores for each class. Note that this particular set of weights W is not good at all: the weights assign our cat image a very low cat score. In particular, this set of weights seems convinced that it's looking at a dog.

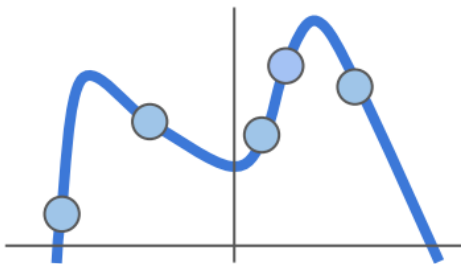
Interpreting a linear classifier



- Cartoon representation of the image space, where each image is a single point, and three classifiers are visualized.
- Using the example of the car classifier (in red), the red line shows all points in the space that get a score of zero for the car class.
- The red arrow shows the direction of increase, so all points to the right of the red line have positive (and linearly increasing) scores, and all points to the left have a negative (and linearly decreasing) scores

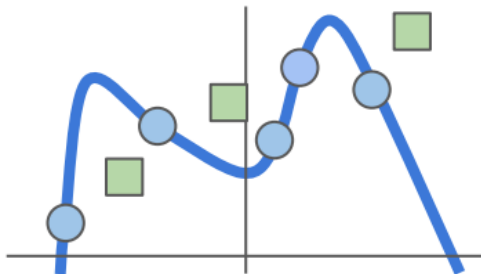
Overfitting problem

Train result



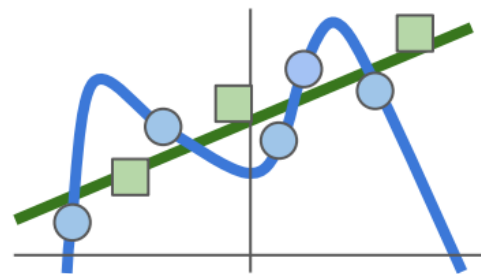
Suppose this is the result of our model. Model ideally fitted to every train example.

Check test examples



OK, try to test our “ideally” model and add test examples. We can see that the result is bad, all test points are outside of blue line.

Expected result

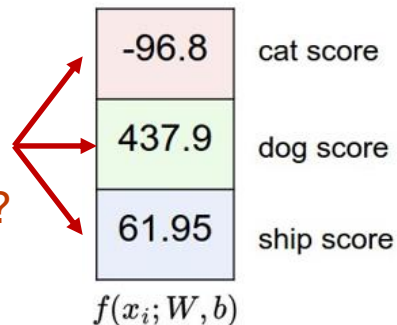


The green line is the best choice where all points (train and test) are equidistant from the line.

Softmax classifier

What does it mean?

How do we need to interpret these values?



May be a probabilistic interpretation is more convenient?

- Softmax classifier gives an intuitive output (normalized class probabilities) and has a probabilistic interpretation
- It takes a vector of arbitrary real-valued scores (in z) and squashes it to a vector of values between zero and one that sum to one

Softmax function:
$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Loss function

What should we do if “cat” is classified as “dog”?

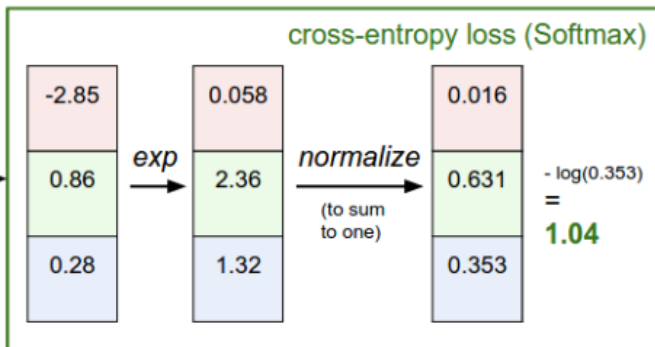
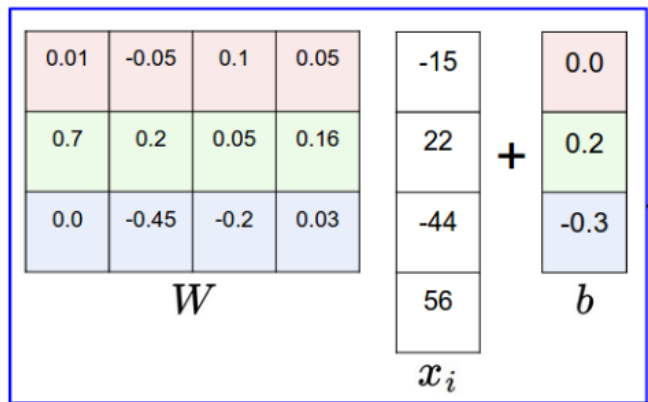


cat	3.2	1.3	2.2
dog	5.1	4.9	2.5
ship	-1.7	2.0	-3.1

- Let's suppose our train dataset: $\{(x_i, y_i)\}_{i=1}^N$, where N – number of train examples, x_i – i -th image and y_i – class of i -th image
- Our classification function: $f(x_i, W) = Wx_i$
- We fed an image with cat but the cat score came out very low (-96.8) compared to the other classes (dog score 437.9 and ship score 61.95).
- We are going to measure our unhappiness with outcomes such as this one with a **loss function** (or sometimes also referred to as the **cost function**)
- Loss function: $L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$
compares our prediction with y_i label

Cross-entropy loss

matrix multiply + bias offset



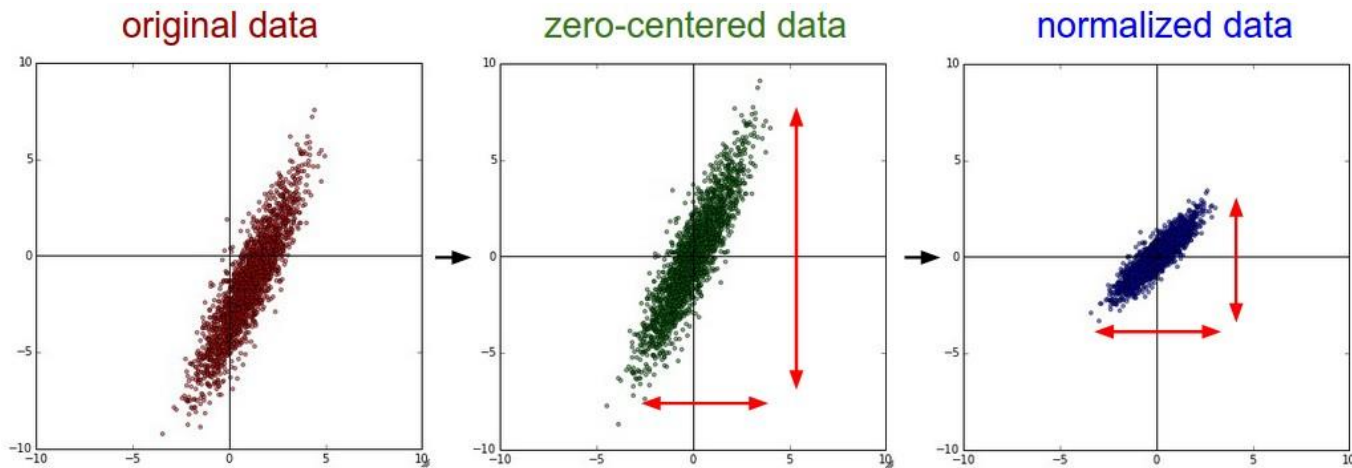
y_i

0
0
1

- Loss function: $L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$
- The **cross-entropy** between a “true” distribution p and an estimated distribution q is defined as: $H(p, q) = -\sum_x p(x) \log q(x)$
- The Softmax classifier is hence minimizing the *cross-entropy* between the estimated class probabilities and the “true” distribution p ($p = [0, \dots, 1, \dots, 0]$ contains a single 1 at the y_i -th position)

Data preprocessing

- **Mean subtraction** involves subtracting the mean across every individual *feature* in the data, and has the geometric interpretation of centering the cloud of data around the origin along every dimension
- **Normalization** refers to normalizing the data dimensions so that they are of approximately the same scale



Multi-layer perceptron

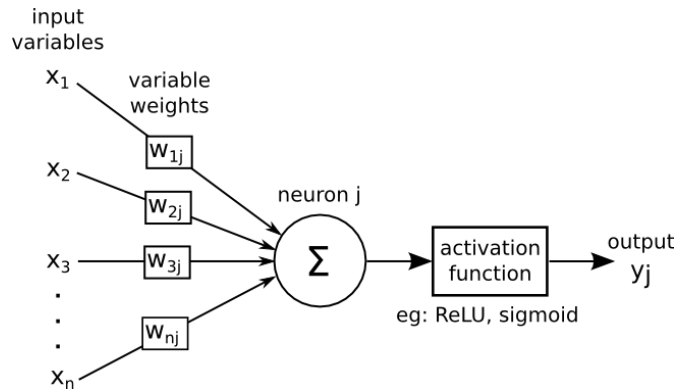
Linear model: $h(x) = w^T x + w_0$.

Generalized linear model: $h(x) = g(w^T x + w_0)$,
where g^{-1} is a non-linear *link* function.

- Logistic regression as a special case.
Let $\mathcal{Y} = \{0, 1\}$, then $h(x) \equiv P(y = 1) = \sigma(w^T x + w_0)$,
where $\sigma(z) = \frac{1}{1+e^{-z}}$ is a sigmoid function.

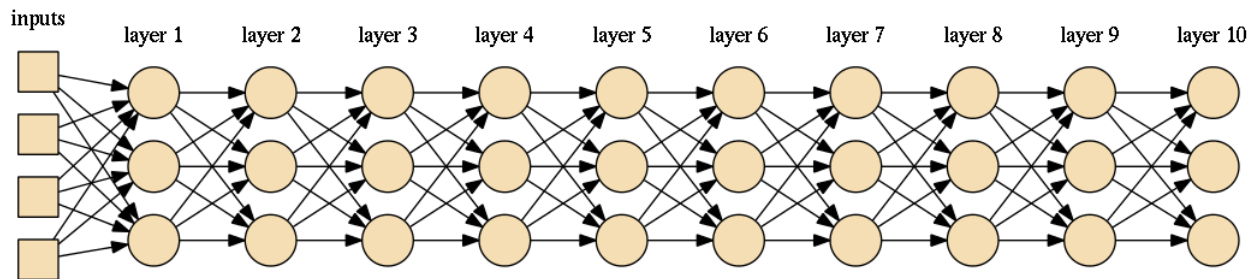
Superposition on **GLMs**: multi-layer perceptron (**MLP**).

- MLP is the simplest feed-forward neural network.
- Naturally represented as a digraph.



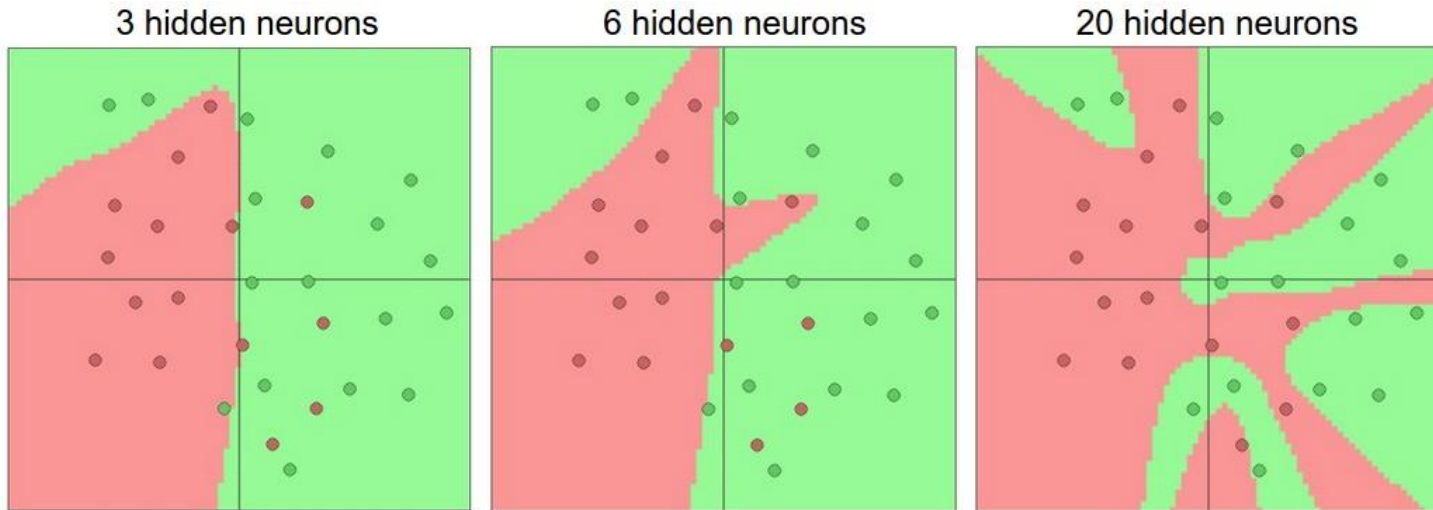
Deep Neural Networks

Main idea: let's stack lots of layers of non-linear transforms.



- No one strict definition of depth, but intuitively it's the number of non-linear data transformation stages involved in the network.
- Hidden layers are not restricted to represent the same non-linear function (class of functions).

Setting number of layers



- Should we use no hidden layers? One hidden layer? Two hidden layers? How large should each layer be?
- As we increase the size and number of layers in a Neural Network, the **capacity** of the network increases
- **Overfitting** occurs when a model with high capacity fits the noise in the data instead of the (assumed) underlying relationship

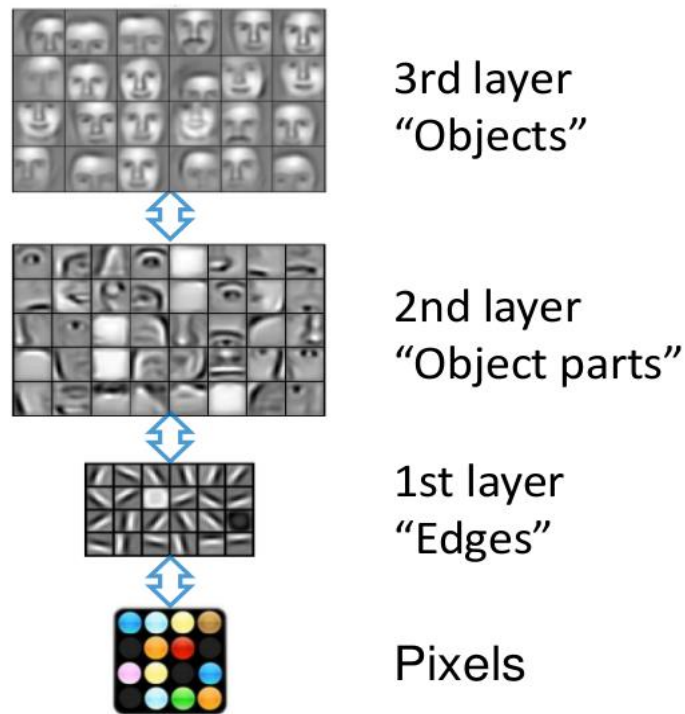
Weights initialization

- **all zero initialization:** there is no source of asymmetry between neurons if their weights are initialized to be the same
- **Small random numbers:** it is common to initialize the weights of the neurons to small numbers and refer to doing so as *symmetry breaking*. The idea is that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network (for example Xavier, MSRA)
- **Initializing the biases:** it is possible and common to initialize the biases to be zero, since the asymmetry breaking is provided by the small random numbers in the weights



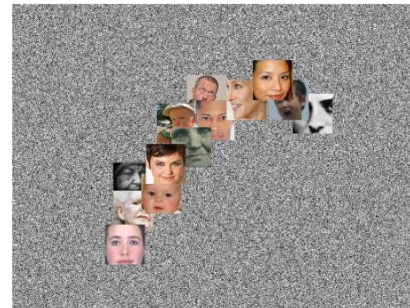
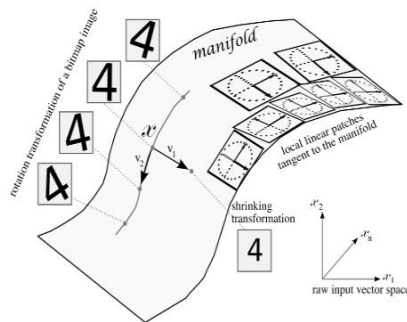
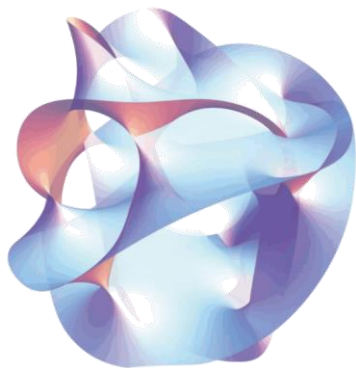
Deep Learning for CV

- Hierarchical (deep) representation is trained automatically given sufficient amount of data.
- Learned representation is hoped to be universal in sense that it can be reused for different tasks involving images.



Manifold Learning

- There is strong evidence that natural data lies on low-dimensional manifolds.
- Deep Neural Networks learn how to parameterize these manifolds automatically.



Let's make it more formal

Supervised learning algorithm is defined by:

- Class of functions \mathcal{H} to search model h in.
 - Broader family of functions increases the chance to have a good approximation in, but makes it difficult to find it.
- Quality metrics Q (or loss \mathcal{L}).
 - One should formalize what it means that one function is better than another and quantify the difference.
- Optimization algorithm.
 - Defines a search (optimizing) procedure in \mathcal{H} w.r.t. Q .
 - We are maximizing Q (or minimizing loss \mathcal{L}) on the training dataset \mathcal{D} while tracking Q' (or \mathcal{L}') on a hold-out dataset \mathcal{D}_{test} .

Training procedure

- Having a gradient one could use gradient descent method for optimization of loss.

$$\Theta := \Theta - \varepsilon \nabla_{\Theta} \mathcal{L}(f(X), Y).$$

- Batch gradient descent is still computationally expensive. Mini-batch stochastic gradient descent (SGD) is much more resource friendly.
 - Gradient is estimated on a small random subset of training data:

$$\Theta := \Theta - \varepsilon \nabla_{\Theta} \mathcal{L}(f(X'), Y').$$

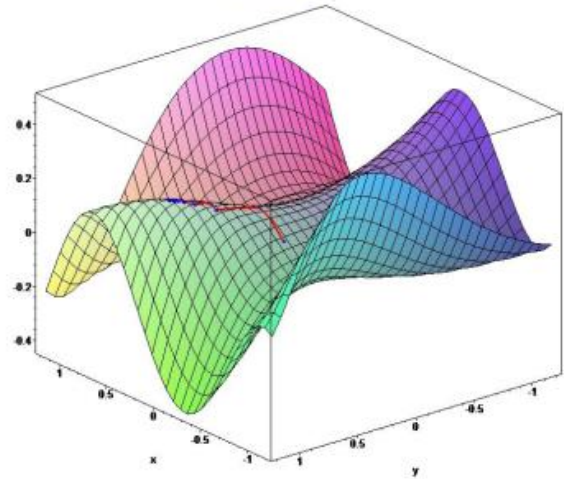
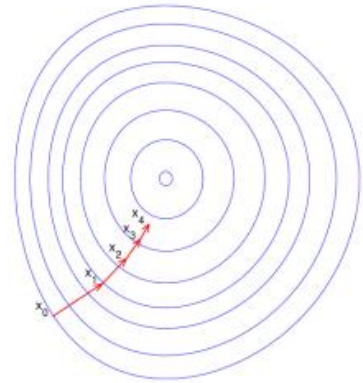
- Random initialization can be used for obtaining a starting point Θ_0 .
- Good initialization is crucial for convergence.

Gradient Descent

Learning rate ε is the most important parameter of training algorithm. Modifications of SGD with adaptive learning rate are widely used:

- AdaGrad;
- AdaDelta;
- RMSProp;
- Adam;
- etc.

How to effectively compute the gradient of the net w.r.t. its parameters?



Backpropagation

Neural network represents a function $y = f(x; w)$ which is superposition of simpler functions corresponding to its layers:

$$f(x; w) = h_n(h_{n-1}(h_{n-2}(\dots (h_1(x); w_1)); w_{n-2}); w_{n-1}); w_n)$$

It is straightforward to compute partial derivative of f w.r.t. any network parameter just applying the chain rule:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

for scalars or the same for vectors:

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j} \frac{dy_j}{dx_i}$$

Backpropagation

Backpropagation is an effective way to evaluate gradient of neural network function w.r.t. all its parameters.

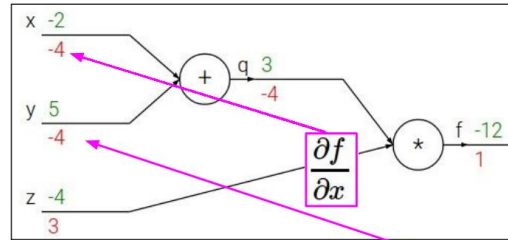
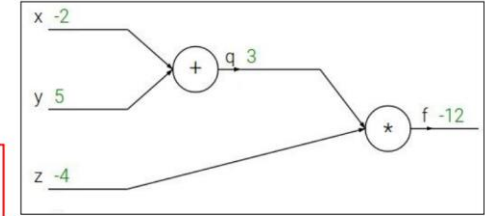
- Neural net nodes are visited in opposite order to the forward pass.
- Gradient of current node w.r.t. its parameters is evaluated.
- Gradient of current node w.r.t. its input is evaluated and the process is repeated.

$$f(x, y, z) = (x + y)z$$

$$\text{e.g. } x = -2, y = 5, z = -4$$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

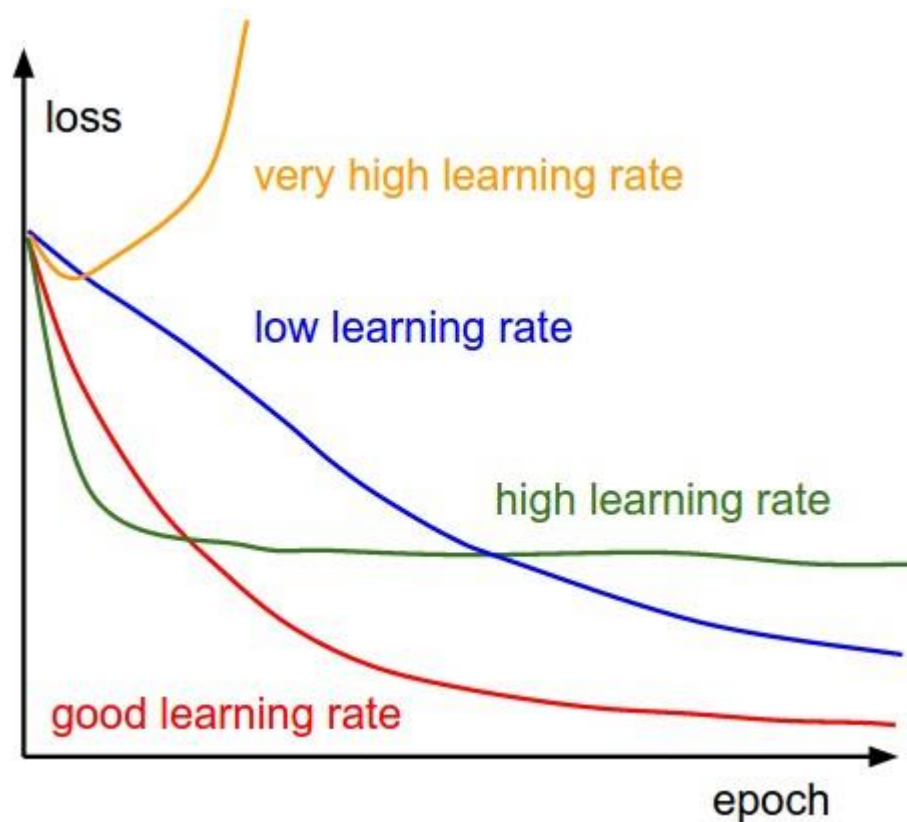


$$\frac{\partial f}{\partial y}$$

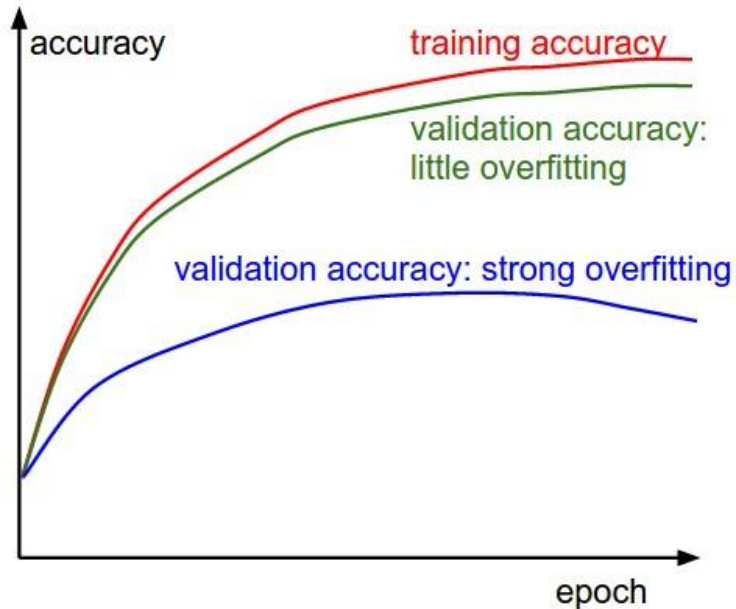
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Learning rate

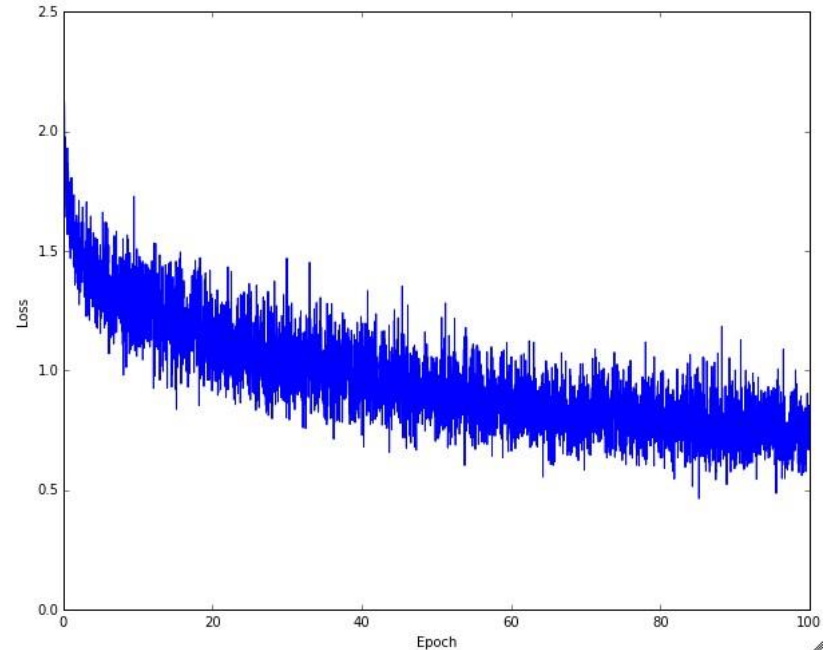
- **Learning rate** is a manually configured hyperparameter
- With *low* learning rate the improvements will be linear
- With *high* learning rate loss will start to look more exponential. Higher learning rate will decay the loss faster, but they get stuck at worse values of loss (green line)
- The best choice probably is some middle learning rate that in the main can be obtained experimentally



Training curves

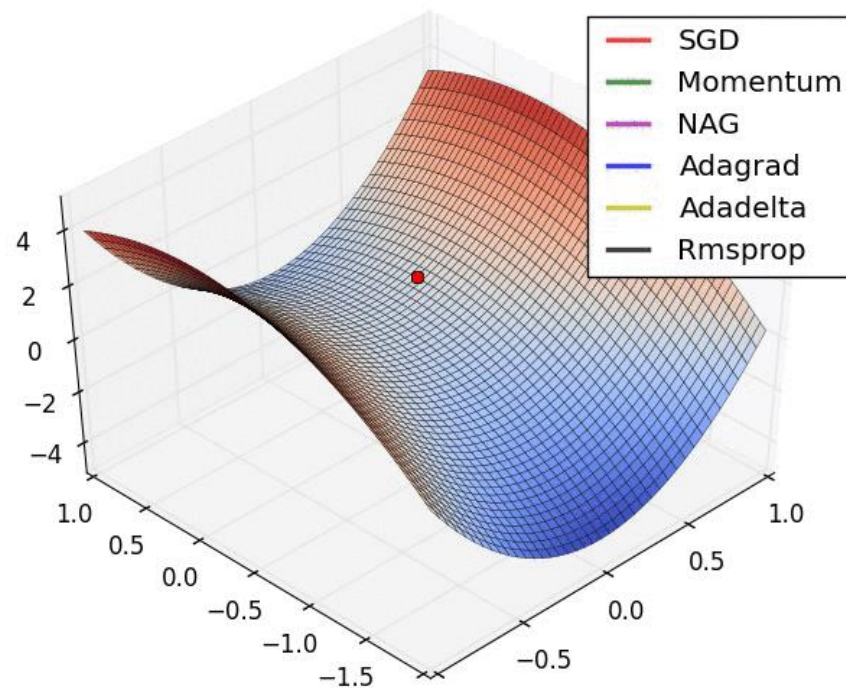
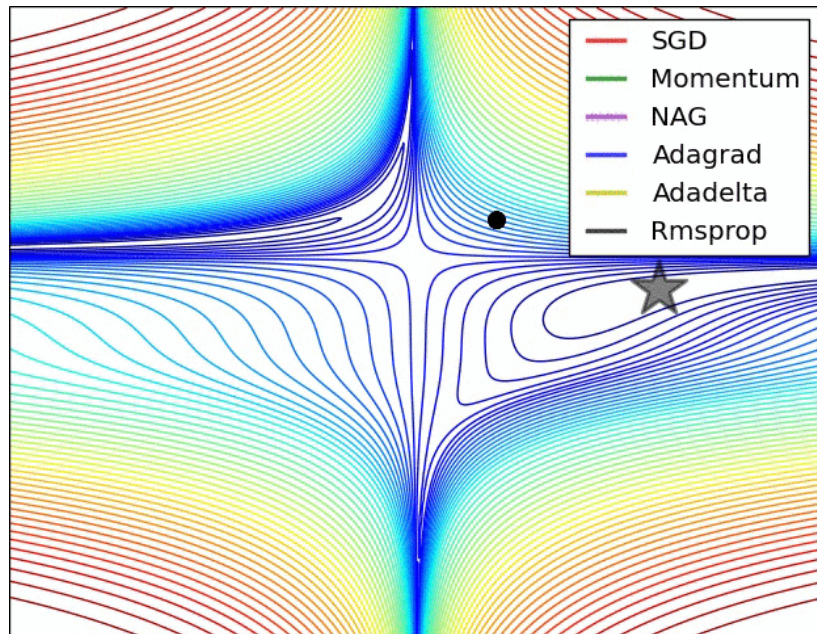


The gap between the training and validation accuracy indicates the amount of overfitting



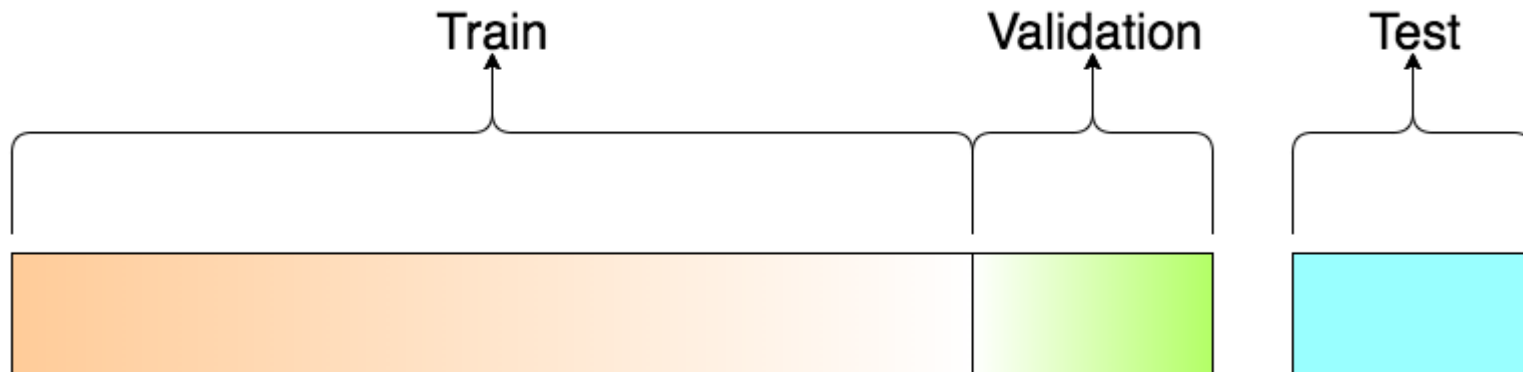
An example of a typical loss function over time, while training a small network on CIFAR-10 dataset

Optimizers



Dataset split

- **Training Dataset:** The sample of data used to fit the model. The model sees and learns from this data
- **Validation Dataset:** The sample of data used to provide an unbiased evaluation of a model fit on the training dataset
- **Test Dataset:** The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset

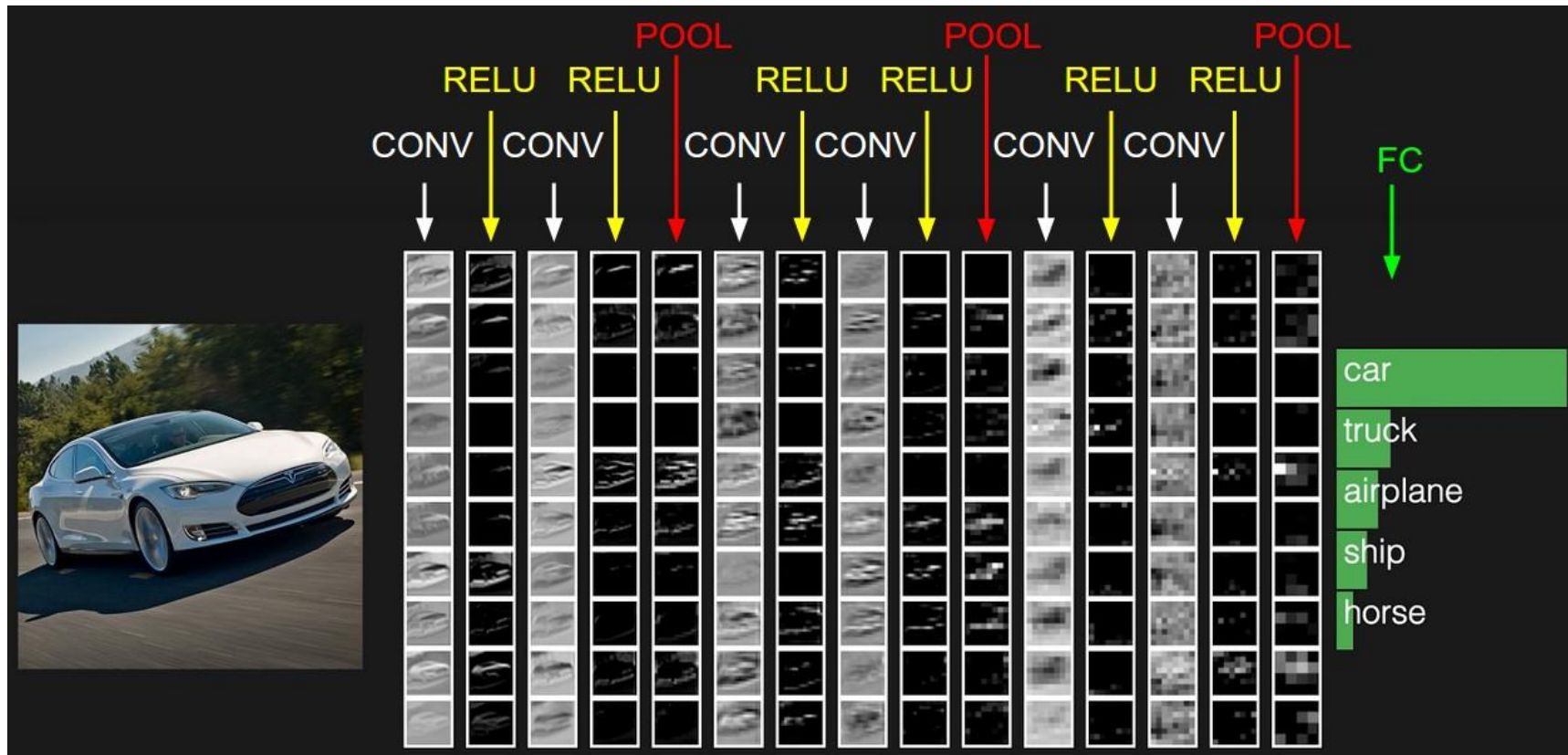


How to create my own DNN?

You need to know main building blocks:

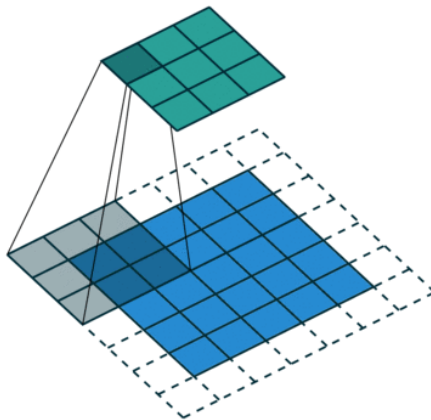
- fully-connected
- activation functions
- convolutions
- sub-sampling (pooling)
- task-specific layers (e.g. classifier or regressors)
- regularizers

Convolutional Neural Networks (CNN)



Building blocks: Convolution

- The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting
- Parameters:
 - kernel size;
 - number of kernels;
 - stride;
 - padding.



0 ₂	0 ₀	0 ₁	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0
0 ₀	0 ₁	1 ₁	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

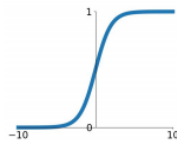
1	6	5
7	10	9
7	10	8

Building blocks: Activation functions

- Prevents deep network collapse to shallow.
- Variants:
 - Logistic sigmoid
 - Hyperbolic tangent
 - ReLU and modifications (Leaky ReLU, PReLU, Concatenated ReLU)

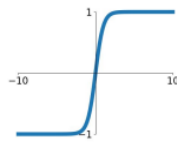
Sigmoid

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



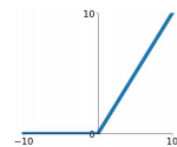
tanh

$$\tanh(x)$$



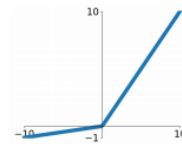
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

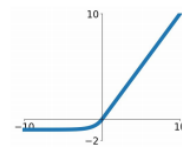


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

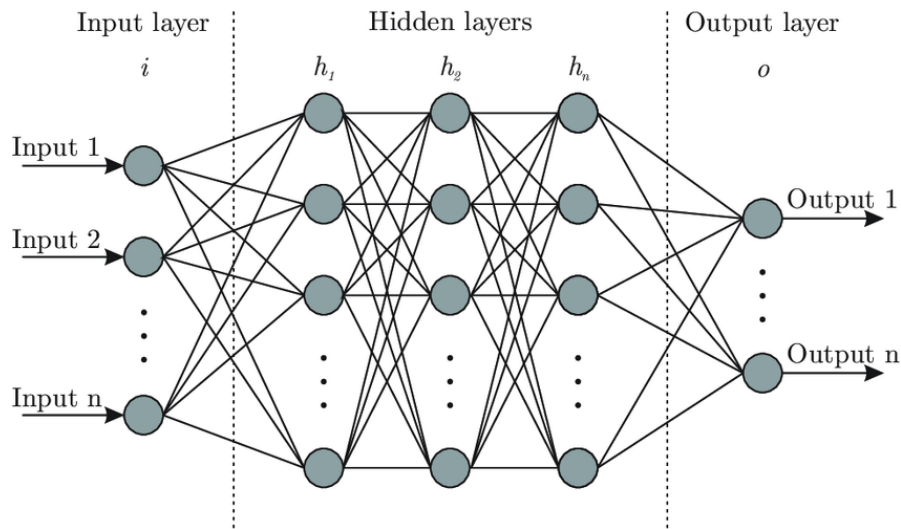


ReLU is the most popular now because it is:

- hardware friendly (fast)
- leads to sparse activations
- sparse gradients
- makes end-to-end training from scratch easier

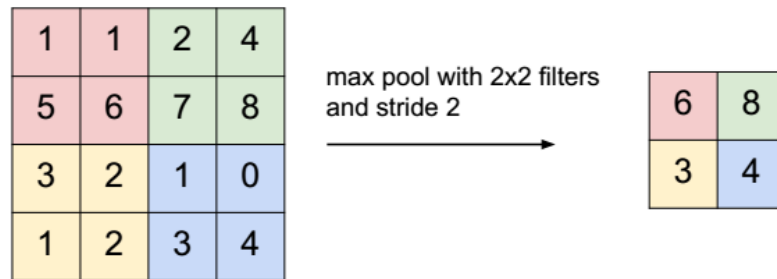
Building blocks: Fully-Connected

- Essentially is a vanilla single-layer perceptron.
- Doesn't preserve spatial information.
- Usually used:
 - right before an output layer, may play a role of a classifier;
 - as bottleneck layer (dimensionality reduction, some kind of PCA);
- Parameters:
 - number of units in the next layer.



Building blocks: Pooling

- Replaces an element of a feature map at certain location by a summary statistics of nearby elements.
- Pros:
 - Adds invariance to local (small) translations.
 - Combined with striding makes representation smaller.
- Popular choices are:
 - max pooling;
 - average pooling.



Parameters:

- type
- kernel size
- stride

Building blocks: Regularizer

Regularization is a common name for techniques that are intended to reduce generalization (test) error rather than simply reduce training loss.

Common regularizers used in context of deep learning:

- weight decay;
- dropout;
- normalization;
- data augmentation;
- multi-task learning.

Regularizers: Weight decay

Places a constraint on weights (elements of a convolutional kernel, parameters of a fully-connected layer, etc.).

Usually in a form of $\|W\|_2^2$ or $\|W\|_1$, which is added to a loss function:

$$\mathcal{L} = \mathcal{L}_{data_fitting}(\mathcal{D}) + \lambda \|W\|.$$

Pros:

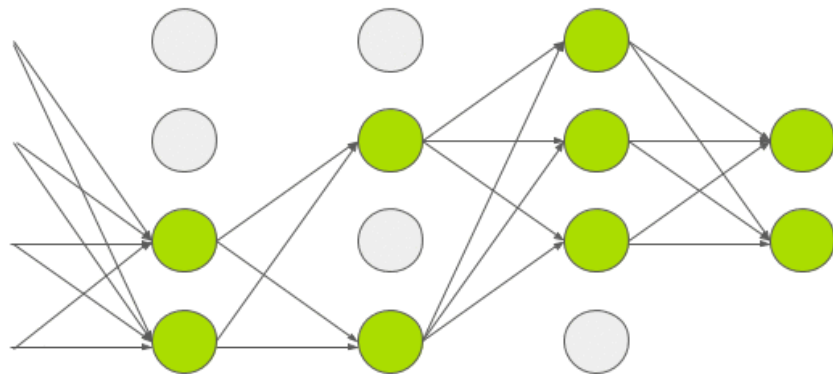
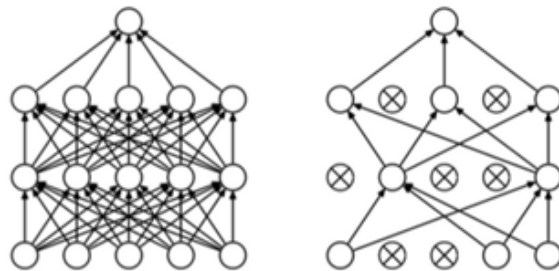
- Makes the model more resistant to random noise.
- May induce weights sparsity.

Parameters:

- weight decay factor λ .

Regularizers: Dropout

- At training stage randomly deactivates (sets to zero) units of the net.
- Can be seen as training an ensemble of networks.
- At test time the whole net is used in a deterministic way, or outputs of several sampled networks are summarized.
- Parameters:
 - dropout ratio.



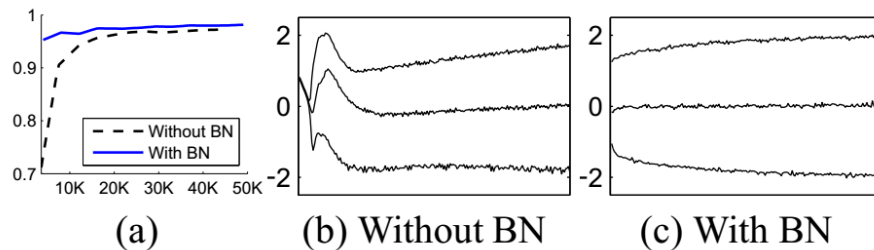
Regularizers: BatchNorm

- Normalizes activations of a layer setting their expected value to zero, and standard deviation to one.
- Pros:
 - Incredibly increases training speed (up to 10x).
 - Makes training process more stable, by reducing the effect of concept drift for deeper layers.
 - Has no parameters.
 - Very cheap at test time.

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

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

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$



Regularizers: Data augmentation

- Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data.
- Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks



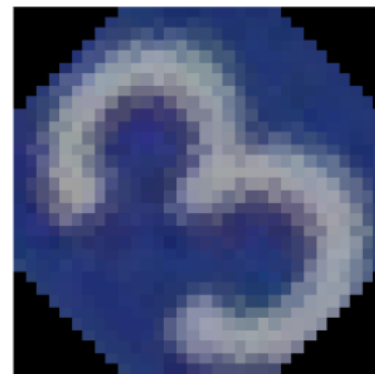
Original



Horizontal Flip



Pad & Crop



Rotate

Building blocks: Classifier

- Solving a classification task we want to get a multinoulli distribution over possible classes conditioned on observed example x :

$$y_c = P(y = c|x),$$

where $c = \overline{1, K}$.

- This can be achieved by applying softmax (softargmax would be a better name) transform on top of the fully-connected layer with K outputs.

$$\widehat{y}_c \equiv \text{softmax}(z)_c = \frac{\exp(z_c)}{\sum_i \exp(z_i)}.$$

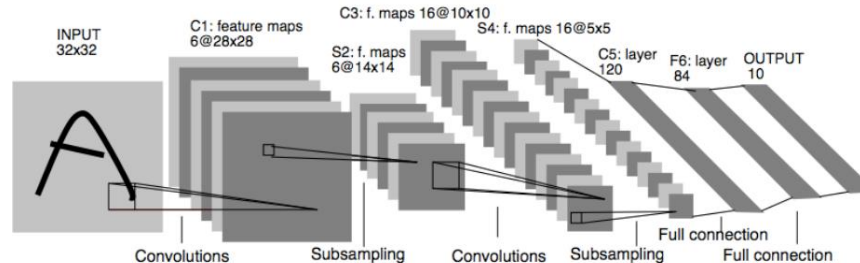
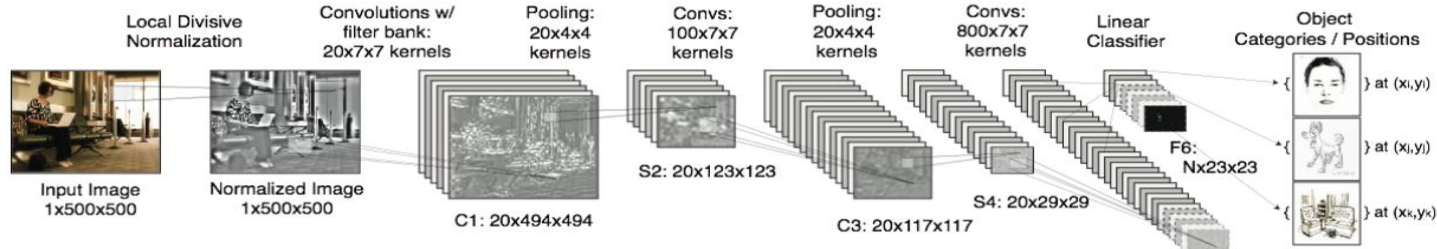
- Combined with a cross-entropy loss is a perfect choice for deep classifier.

Image classification. Datasets overview

Name	Images number	Classes number	Description
ImageNet	~1.5M	1K	The most popular dataset in image classification
CIFAR-10	~60K	10	
MNIST	~70K	10	Contains digits
MS_COCO	~330K	80	The most popular dataset for object detection and segmentation
OpenImages	~5M	5K	

Prehistoric

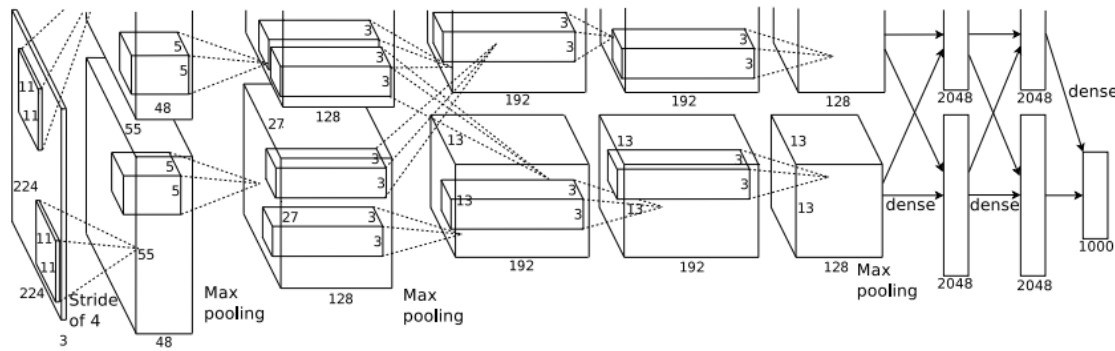
- Hubel & Wiesel 1962: simple cells (local features) + complex cells (“pool”)
- Fukushima 1974-1982: Cognitron & Neocognitron
- LeCun et al. 1989-1998: LeNet



AlexNet (2012)

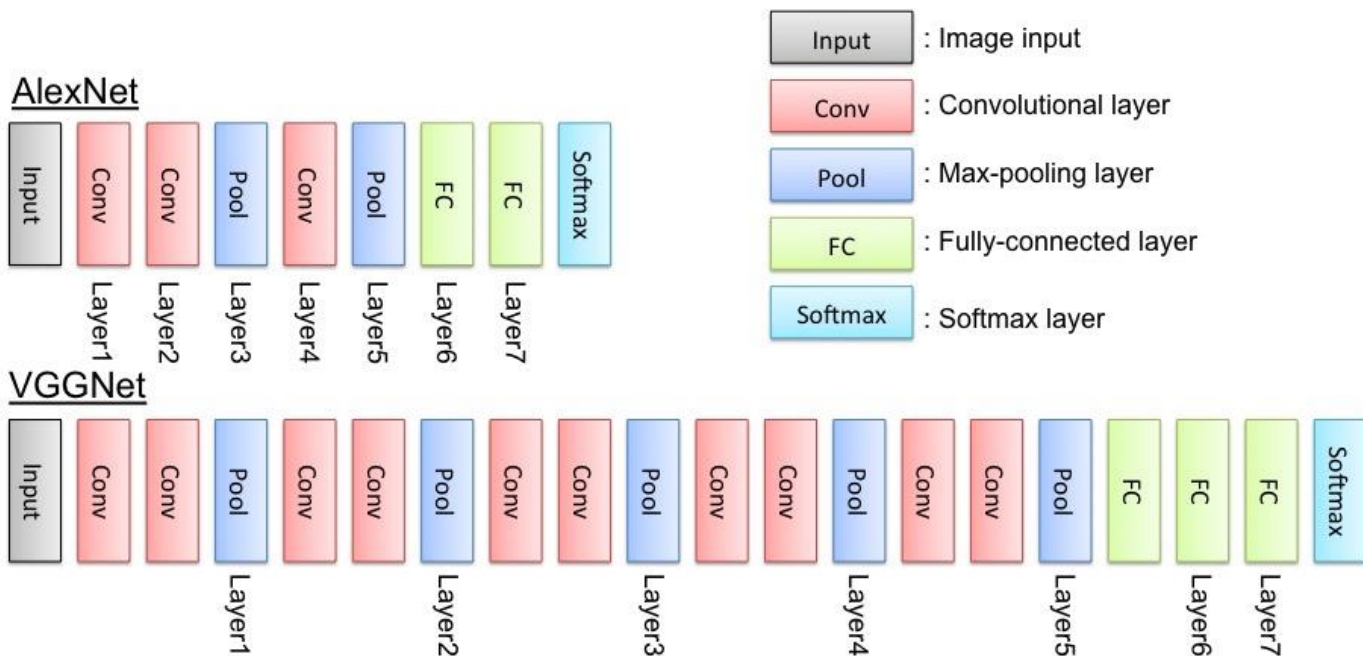
Won the 2012 ImageNet LSVRC [Krizhevsky, Sutskever, Hinton 2012]

- Error rate: 15% (top 5), previous state of the art: 26% error.
- 650K neurons, 832M synapses, 60M parameters.
 - 95% of weights in fully connected, 5% in conv.
 - 95% computations in conv, 5% in fully connected.



VGG (2014)

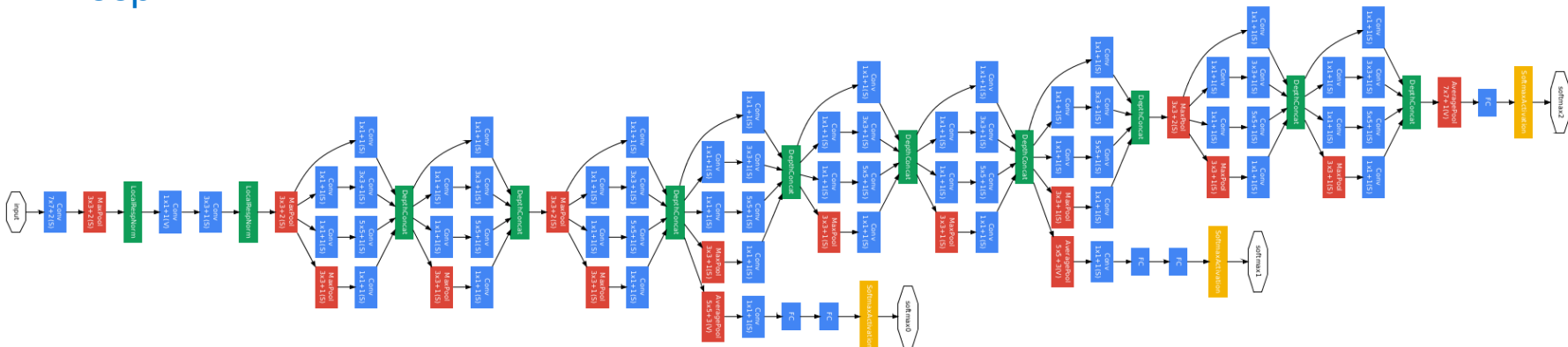
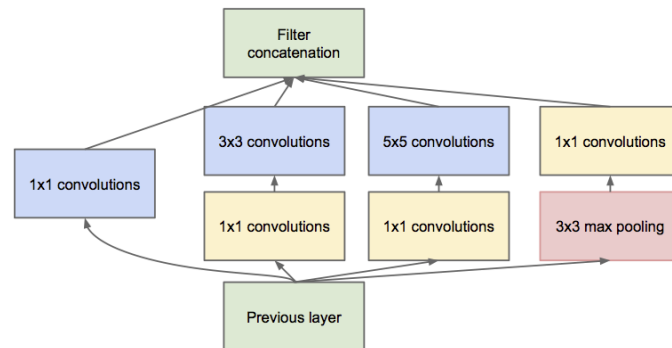
- Use several 3x3 layers instead of 11x11 or 7x7



GoogLeNet (2014)

Won the 2014 ImageNet LSVRC
(~6.6% Top-5 error)

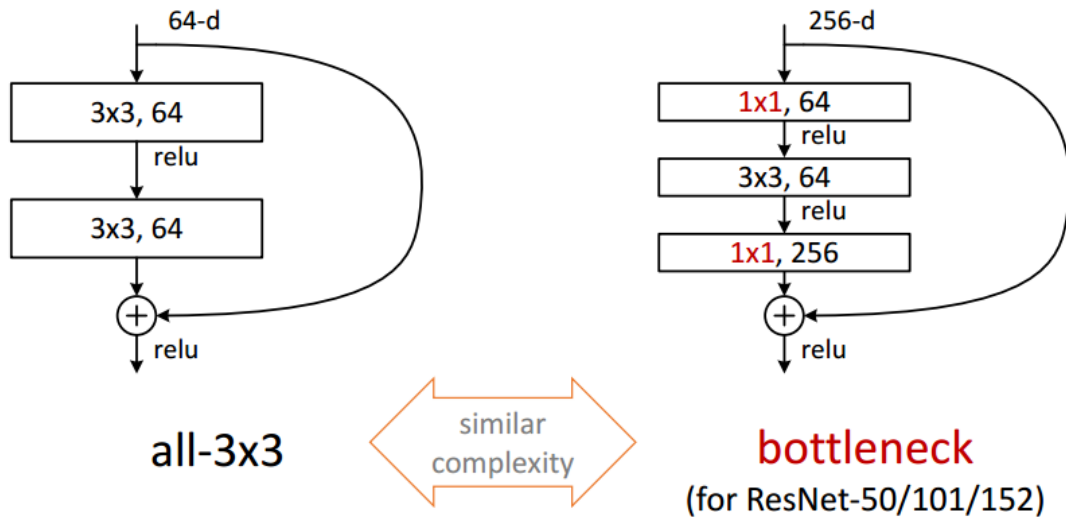
- More scale invariance (inception block)
- Small filters
- Deep



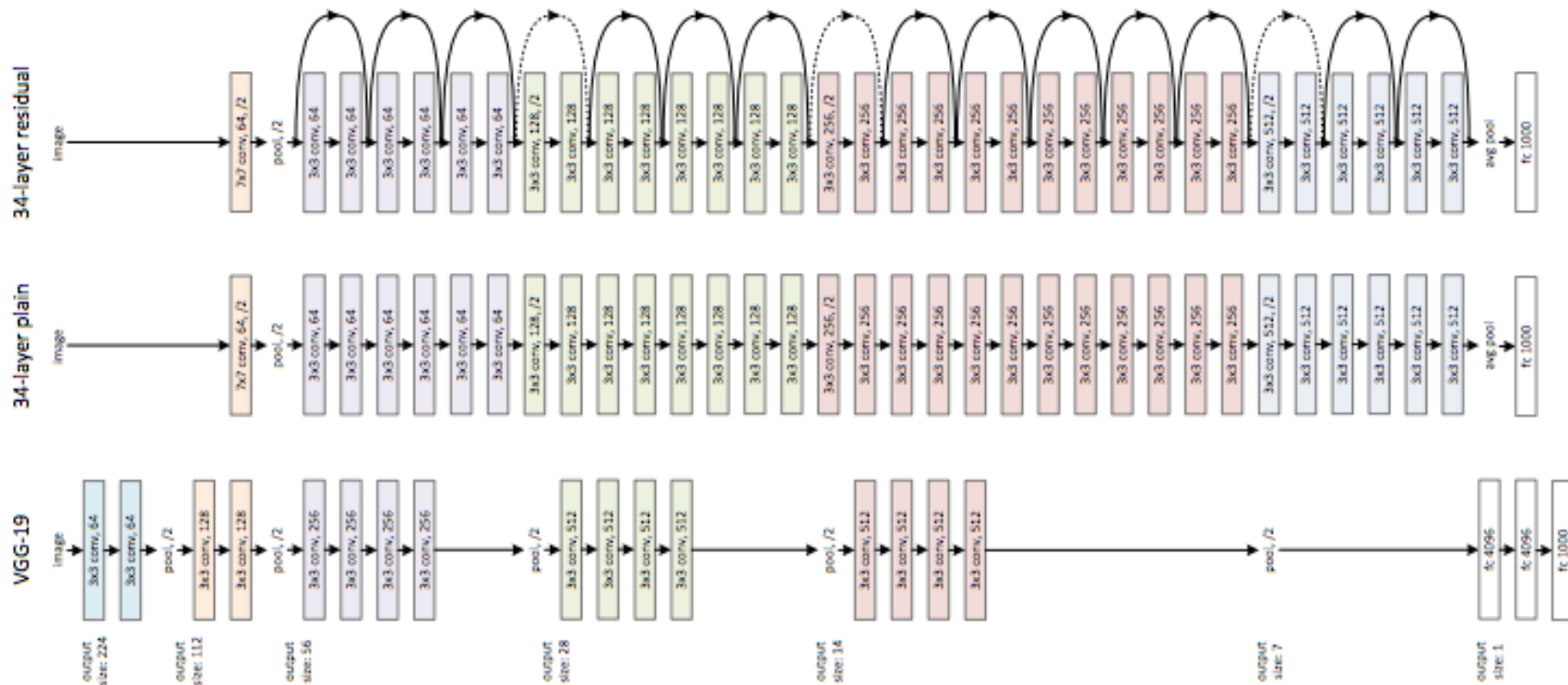
Residual Network (ResNet) (2015)

Won the 2015 ImageNet LSVRC (3.57% ensemble Top-5 error)

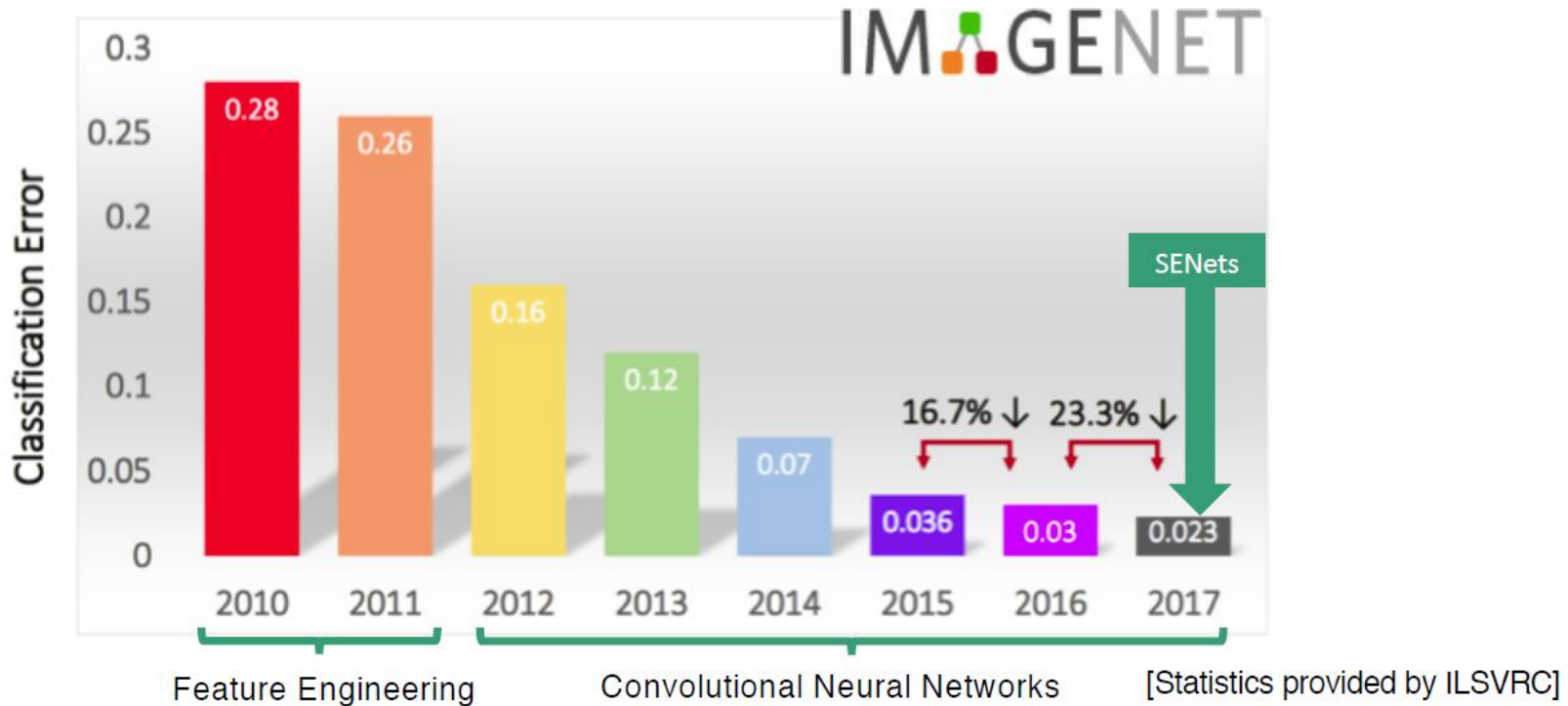
- Identity + residual (~automatic detection of required layers' number)
- Superdeep (152 layers, 1202 for CIFAR-10)



Residual Network (ResNet)

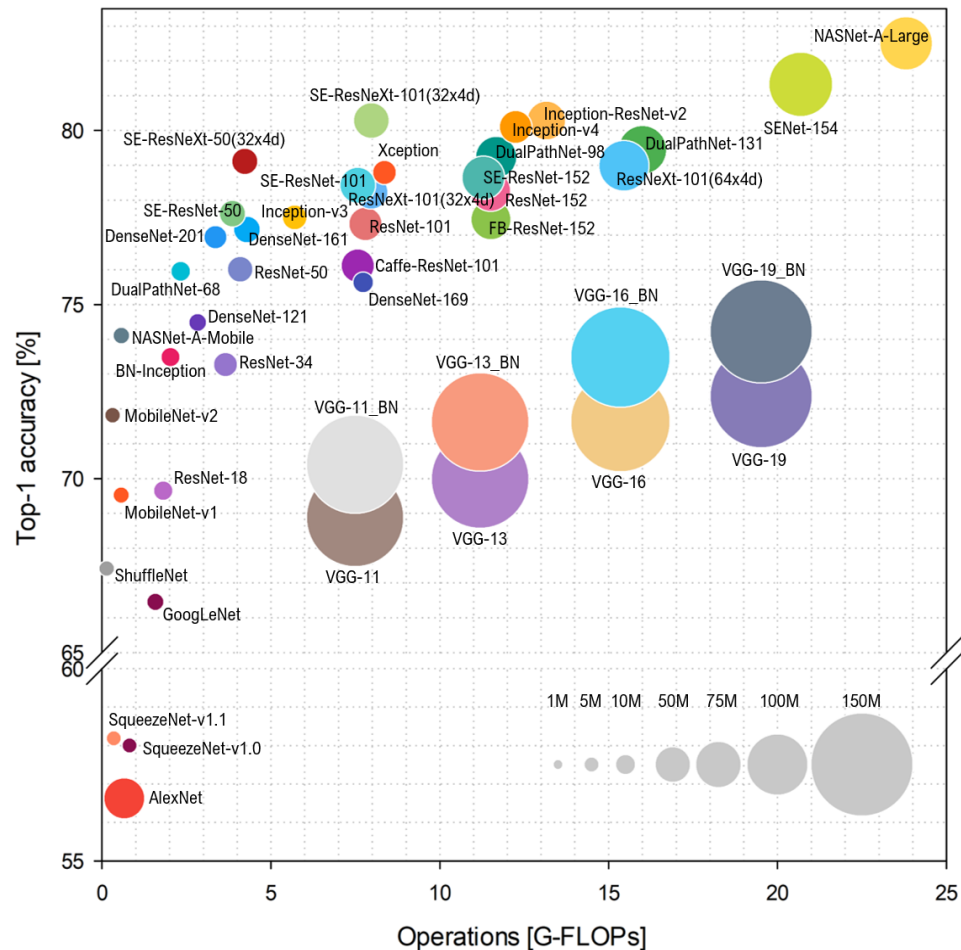


ImageNet progress from 2010 to 2017



Models: not all networks are created equal

- Accuracy
- Operations (G-Ops)
- Model size (parameters)



Simone Bianco , Remi Cadene, Luigi Celona , Paolo Napoletano "Benchmark Analysis of Representative Deep Neural Network Architectures." 2018

Links

- CS231n: Convolutional Neural Networks for Visual Recognition: <http://cs231n.stanford.edu/>
- ImageNet dataset: <http://www.image-net.org/>
- Linear Classification Loss Visualization: <http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/>
- MLP training demo: <http://playground.tensorflow.org/>
- Convolution demo: <https://cs231n.github.io/assets/conv-demo/index.html>

Simone Bianco , Remi Cadene, Luigi Celona , Paolo Napoletano “Benchmark Analysis of Representative Deep Neural Network Architectures.” 2018

