Multi-instance deep learning

Razvan Ranca - tractable.io

Why deep learning?

Traditional approach:

1. Take High-dimensional input (eg: images)
2. Extract hand-crafted features
3. Train (standard-ish) classifier on top
4. Predict class

Why deep learning?

Traditional approach:

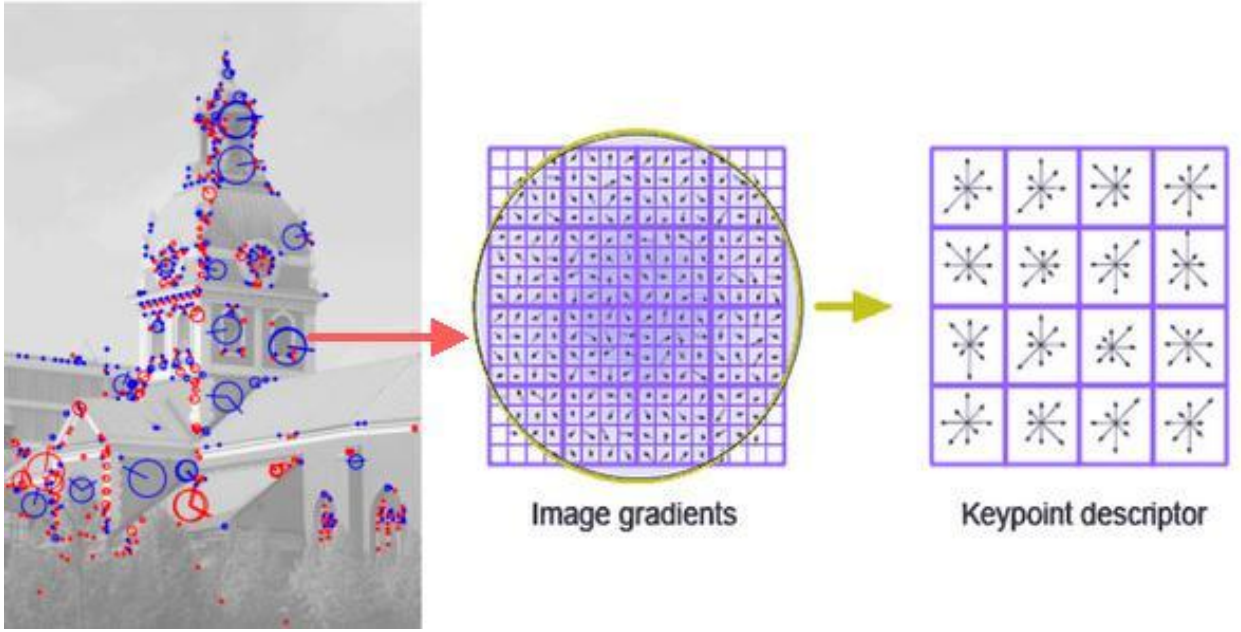
1. Take High-dimensional input
2. Extract hand-crafted features
3. Train (standard-ish) classifier on top
4. Predict class

Quality of hand-crafted features largely determines performance

Why deep learning?

Eg: SIFT

1 Detect Scale-space extrema
2 Filter keypoint candidates
  2.1 Interpolate nearby data
  2.2 Discard low-contrast keypoints
  2.3 Eliminate edge responses
3 Assign orientation
4 Create keypoint descriptor



Quality of hand-crafted features largely determines performance

Why deep learning?

Idea: Learn the feature representation instead

- create feature hierarchy, from raw pixels to output class
- train whole system

Why deep learning?

From

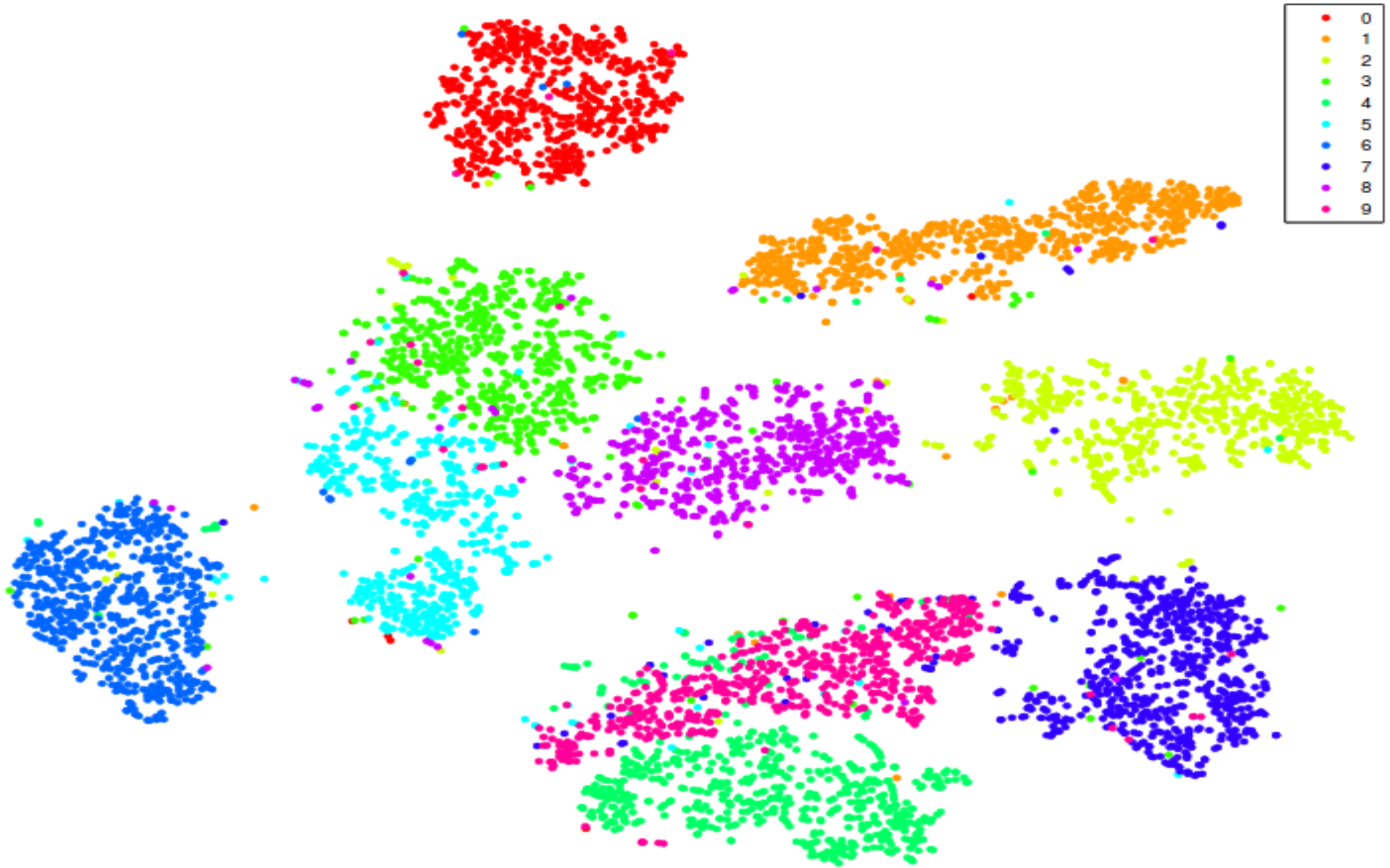
Pixels -> Handcrafted features -> Classifier -> Class

To

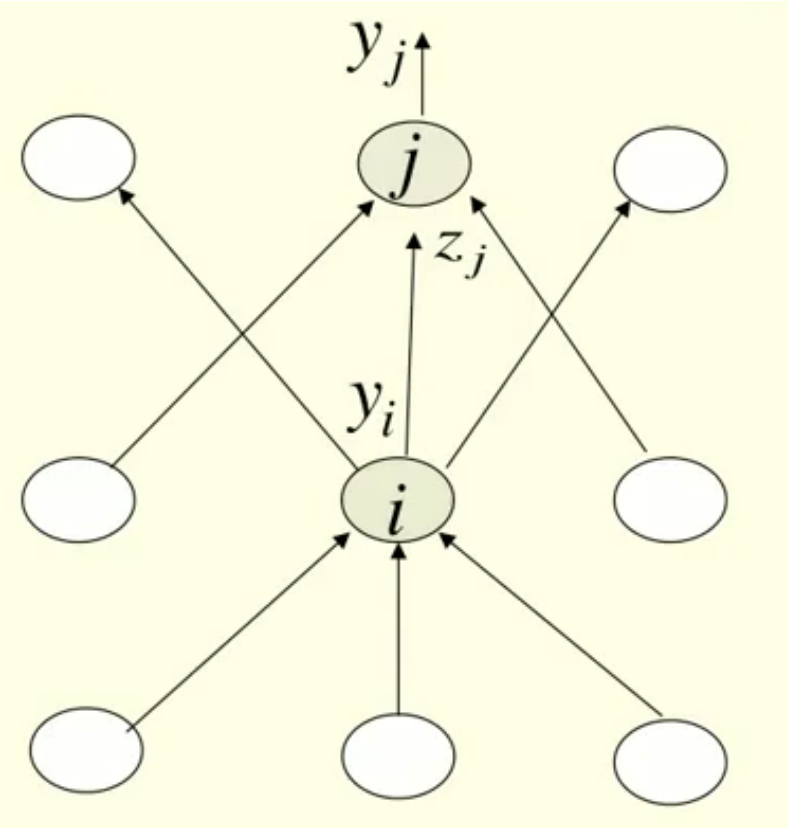
Pixels -> Layer 1 -> Layer 2 ->…->Layer N -> Class

- Can view all layers except last as a feature extractor. If features are sufficiently good, a linear classifier does the job

Sufficiently good features on Mnist



Neural Network Basics - What it is



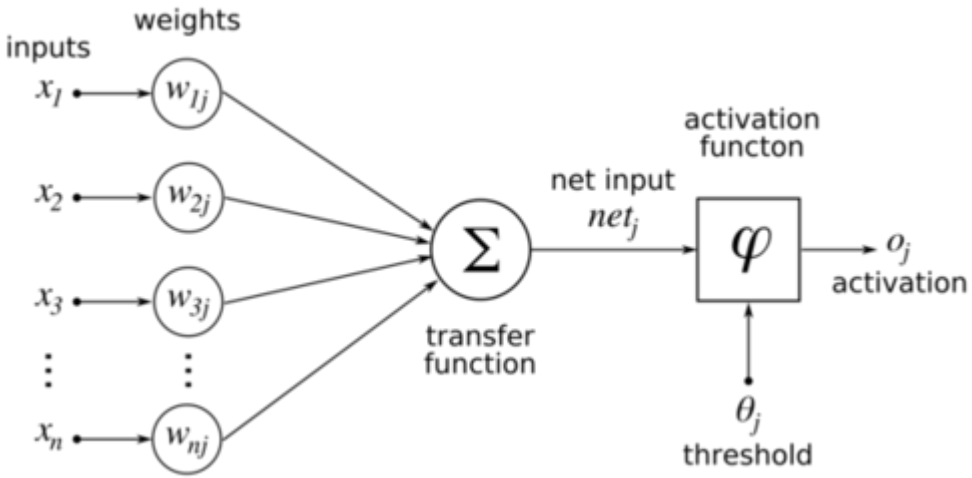
a massively *parameterised* function that takes in raw pixels and spits out label probabilities

neuron: computation

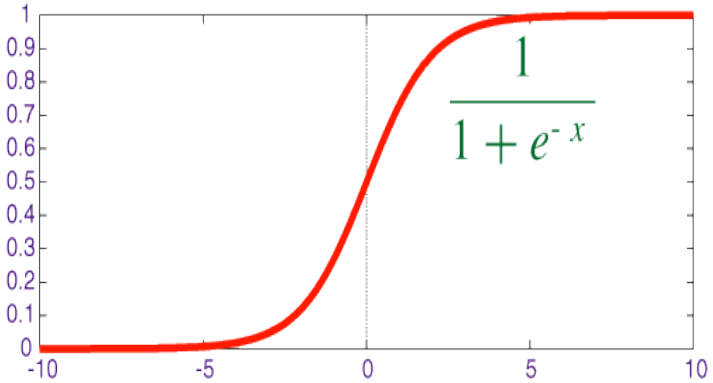
edge weight: parameter

Modern CNNs ~ 100M parameters

Neural Network basics - Perceptron



Sigmoid activation function:

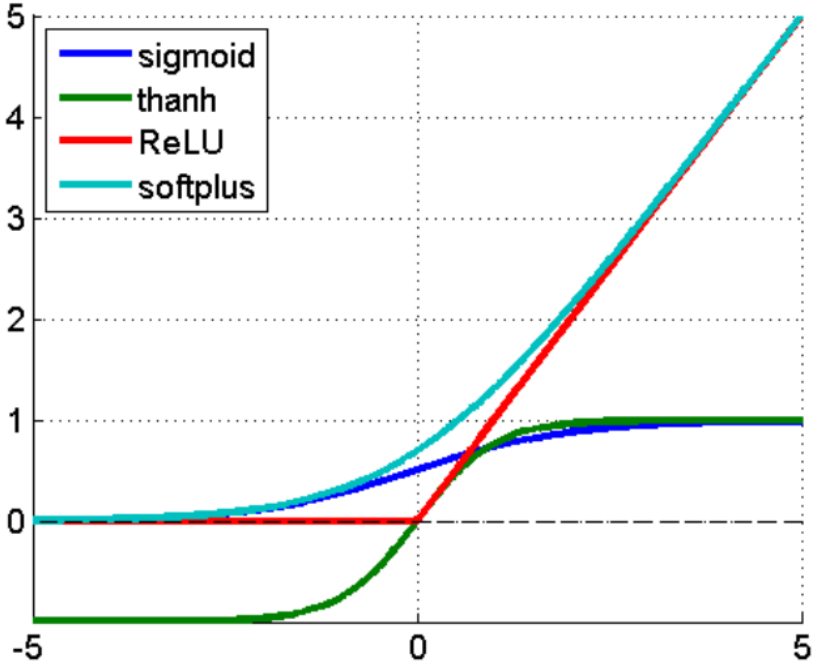


Error = $\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$

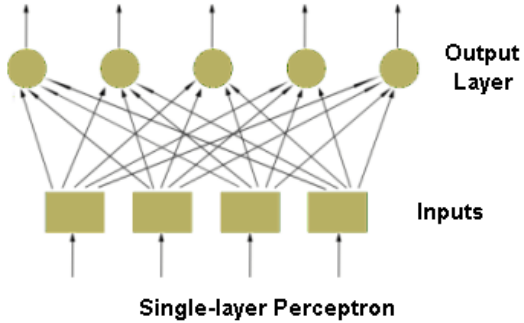
=> $\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{i,d}$

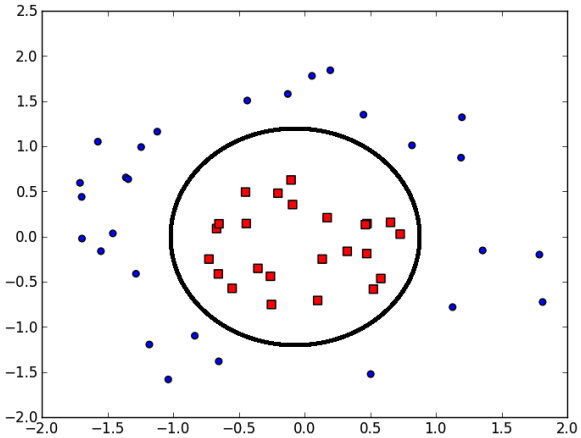
Neural Network basics - non-linearity

Why? To get universal approximators.

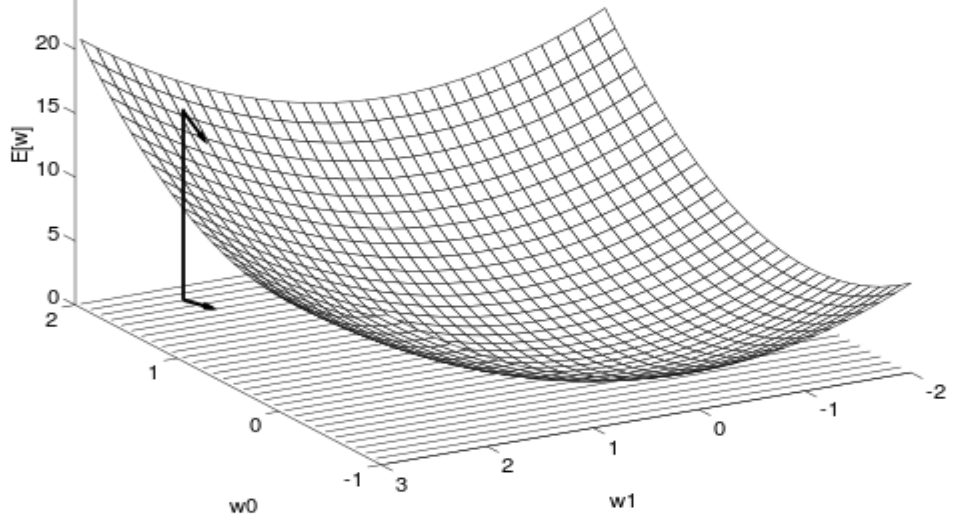


Otherwise (linear) perceptron.





Neural Network Basics - Gradient descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

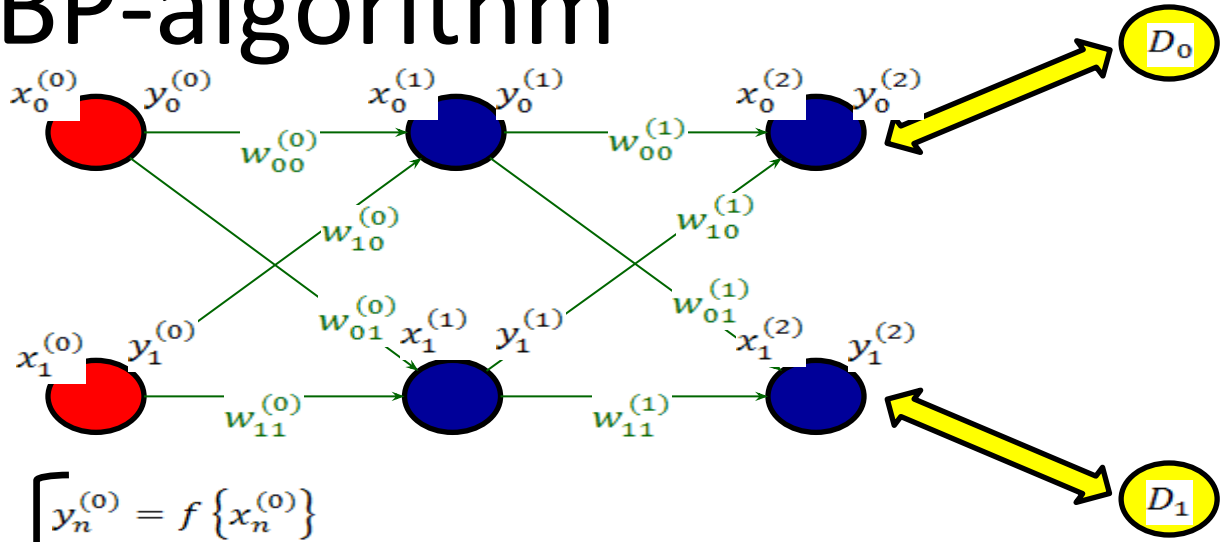
Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

BP-algorithm



Activations

$$\begin{cases} y_n^{(0)} = f\left\{x_n^{(0)}\right\} \\ y_n^{(1)} = f\left\{\sum_{i=0}^{1} y_i^{(0)} w_{in}^{(0)}\right\} \\ y_n^{(2)} = f\left\{\sum_{i=0}^{1} y_i^{(1)} w_{in}^{(1)}\right\} \end{cases}$$

errors

$$\begin{cases} \dfrac{\partial E}{\partial w_{ij}^{(1)}} = -2\left(D_j - y_j^{(2)}\right) f\left\{x_j^{(2)}\right\}\left(1 - f\left\{x_j^{(2)}\right\}\right) y_i^{(1)} \stackrel{\text{def}}{\equiv} \delta_j^{(1)} y_i^{(1)} \\ \dfrac{\partial E}{\partial w_{ij}^{(0)}} = f\left\{x_j^{(1)}\right\}\left(1 - f\left\{x_j^{(1)}\right\}\right) \sum_{n=0}^{1} \delta_n^{(1)} w_{jn}^{(1)} x_i^{(0)} \stackrel{\text{def}}{\equiv} \delta_j^{(0)} x_i^{(0)} \end{cases}$$

The error: $E = \sum_{n=0}^{1} (D_n - y_n^{(2)})^2$

Update Weights: $w_{ij}^{(k)} = w_{ij}^{(k)} + \varepsilon \dfrac{\partial E}{\partial w_{ij}^{(k)}}$

Update

$$\begin{cases} w_{ij}^{(1)} = w_{ij}^{(1)} + \varepsilon \delta_j^{(1)} y_i^{(1)} \\ w_{ij}^{(0)} = w_{ij}^{(0)} + \varepsilon \delta_j^{(0)} x_i^{(0)} \end{cases}$$

Slide credit : Geoffrey Hinton

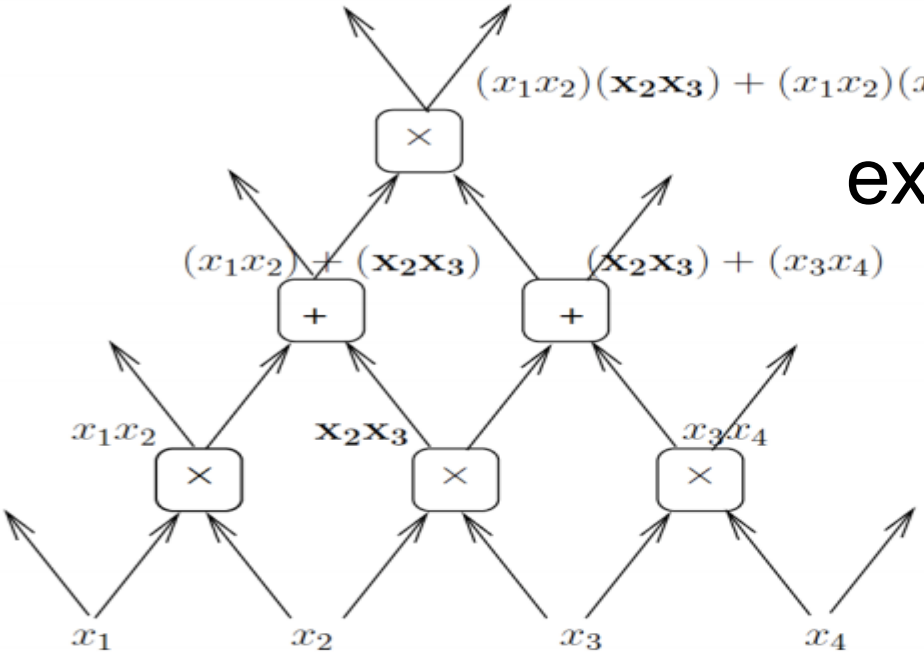
Expressivity of neural networks

- Every boolean function representable by a net with a single hidden layer
- Every bounded continuous function can be approximated, with arbitrarily small error, by a net with a single hidden layer
- Any function can be approximated, with arbitrarily small error, by a net with two hidden layers

Expressivity of neural networks

- Every boolean function representable by a net with a single hidden layer
- Every bounded continuous function can be approximated, with arbitrarily small error, by a net with a single hidden layer
- Any function can be approximated, with arbitrarily small error, by a net with two hidden layers
- But may need a lot of hidden units (exponential in input)

Why deep learning?

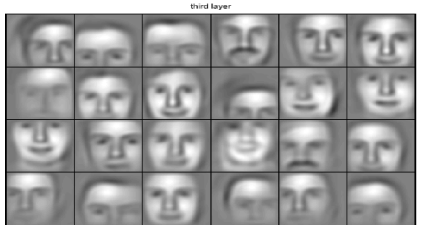


expresivity + abstraction

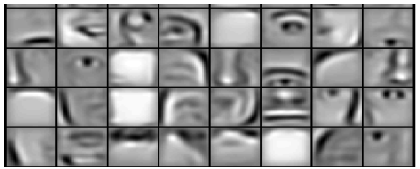
Sum-product network

Slide credit : Yoshua Bengio

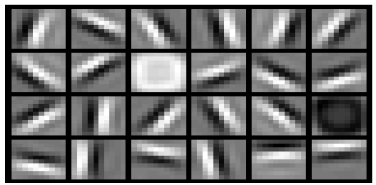
Why Deep learning?



object models

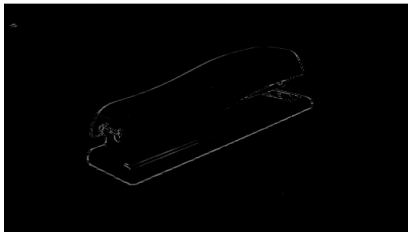


object parts (combination of edges)

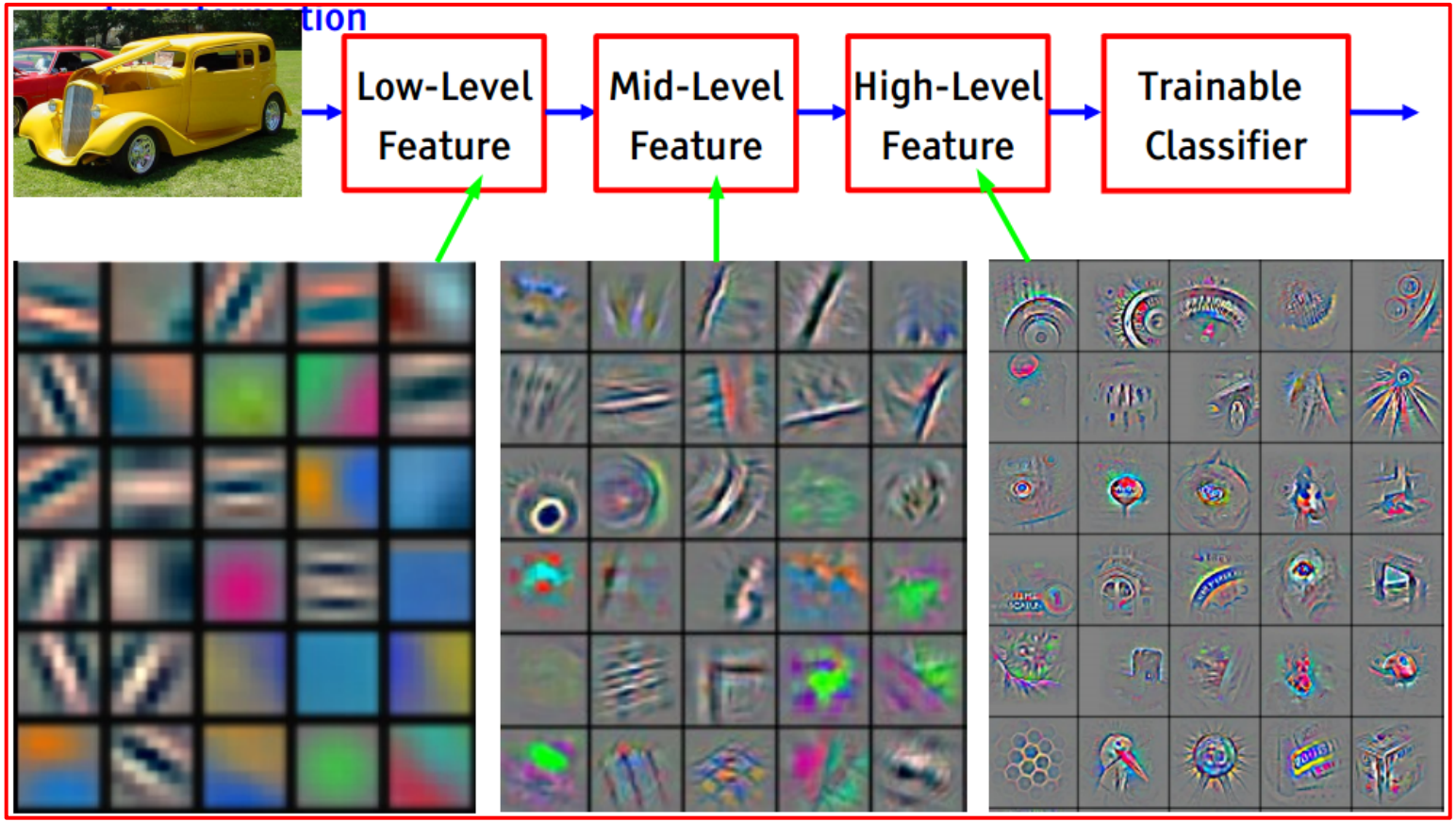


edges

pixels



Deep Learning: Motivations



Deep Neural Networks

- Standard learning strategy
  - Randomly initializing the weights of the network
  - Applying gradient descent using backpropagation
- But, backpropagation does not work well if randomly initialized
  - ANN have to be limited to one or two layers

What's wrong with standard neural networks?

How many parameters does this neural network have?

$$|\theta| = 3D^2 + D$$

For a small 32 by 32 image:

$$|\theta| = 3 \times 32^4 + 32^2 \approx 3 \times 10^6$$

Hard to train

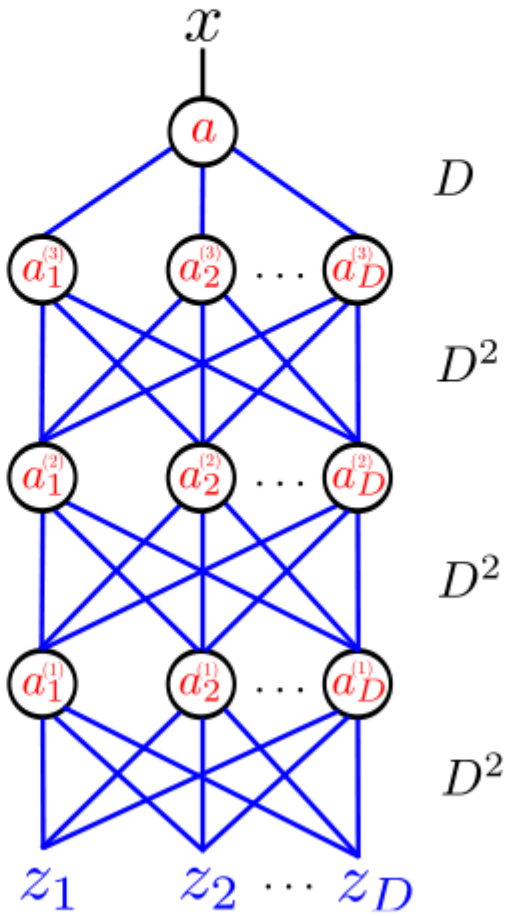
over-fitting and local optima

Need to initialise carefully

layer wise training

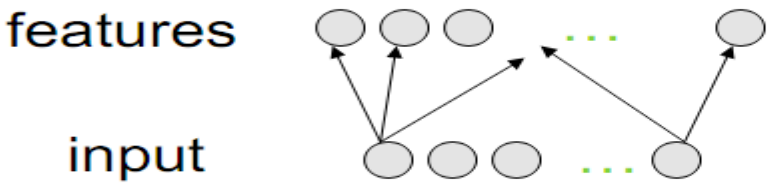
unsupervised schemes

Slide credit: Richard Turner

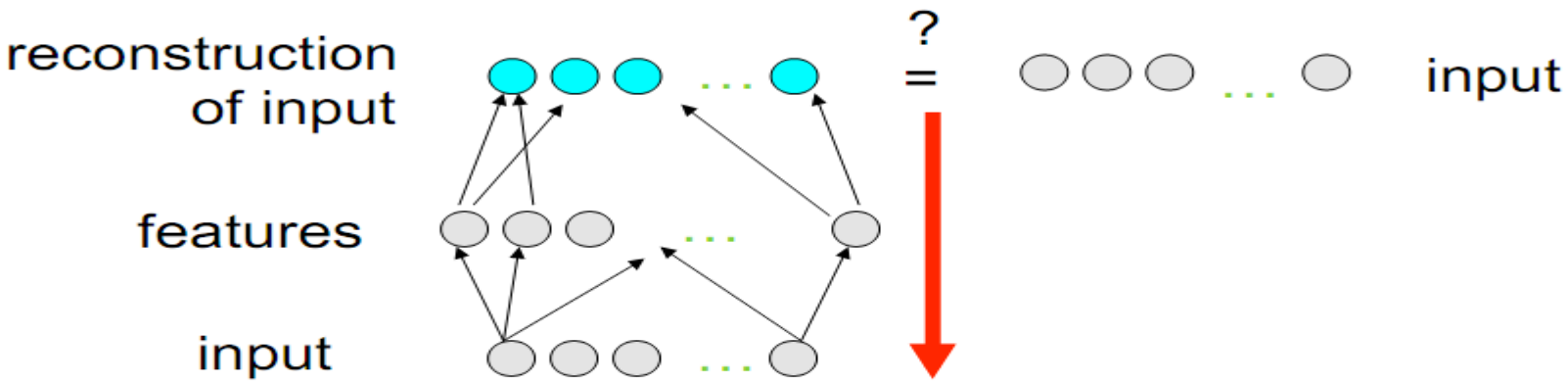


Solution 1: Deep training

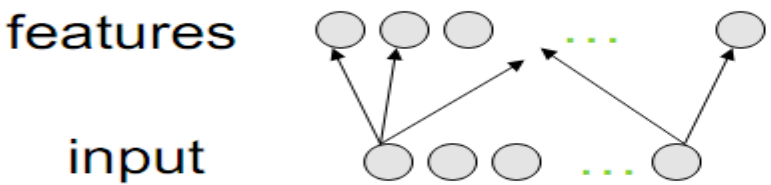
Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training

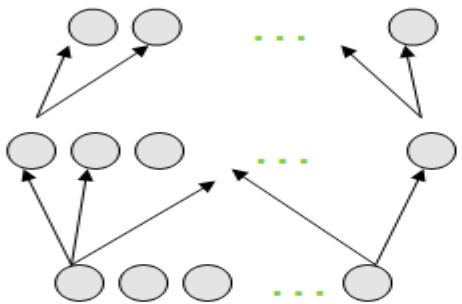


Layer-Wise Unsupervised Pre-training

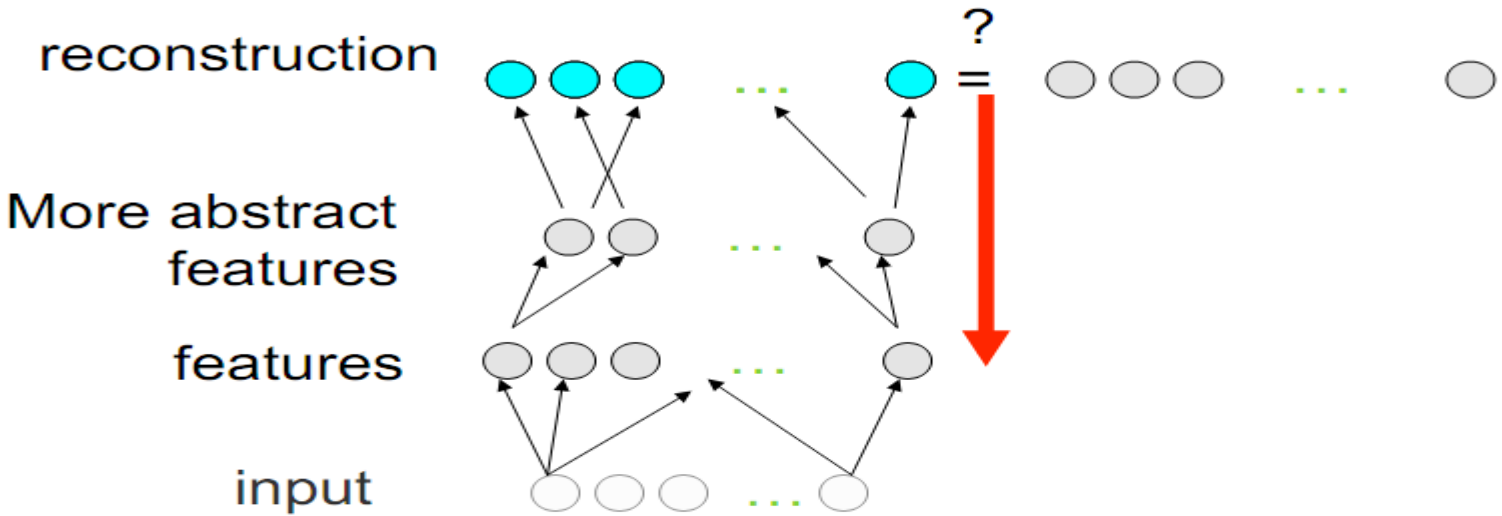
More abstract features

features

input



Layer-Wise Unsupervised Pre-training

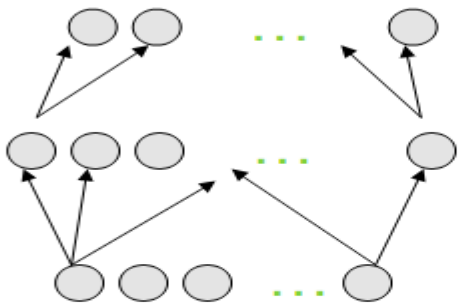


Layer-Wise Unsupervised Pre-training

More abstract features

features

input



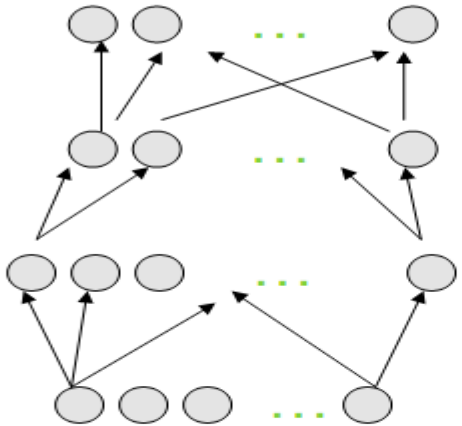
Layer-Wise Unsupervised Pre-training

Even more abstract features

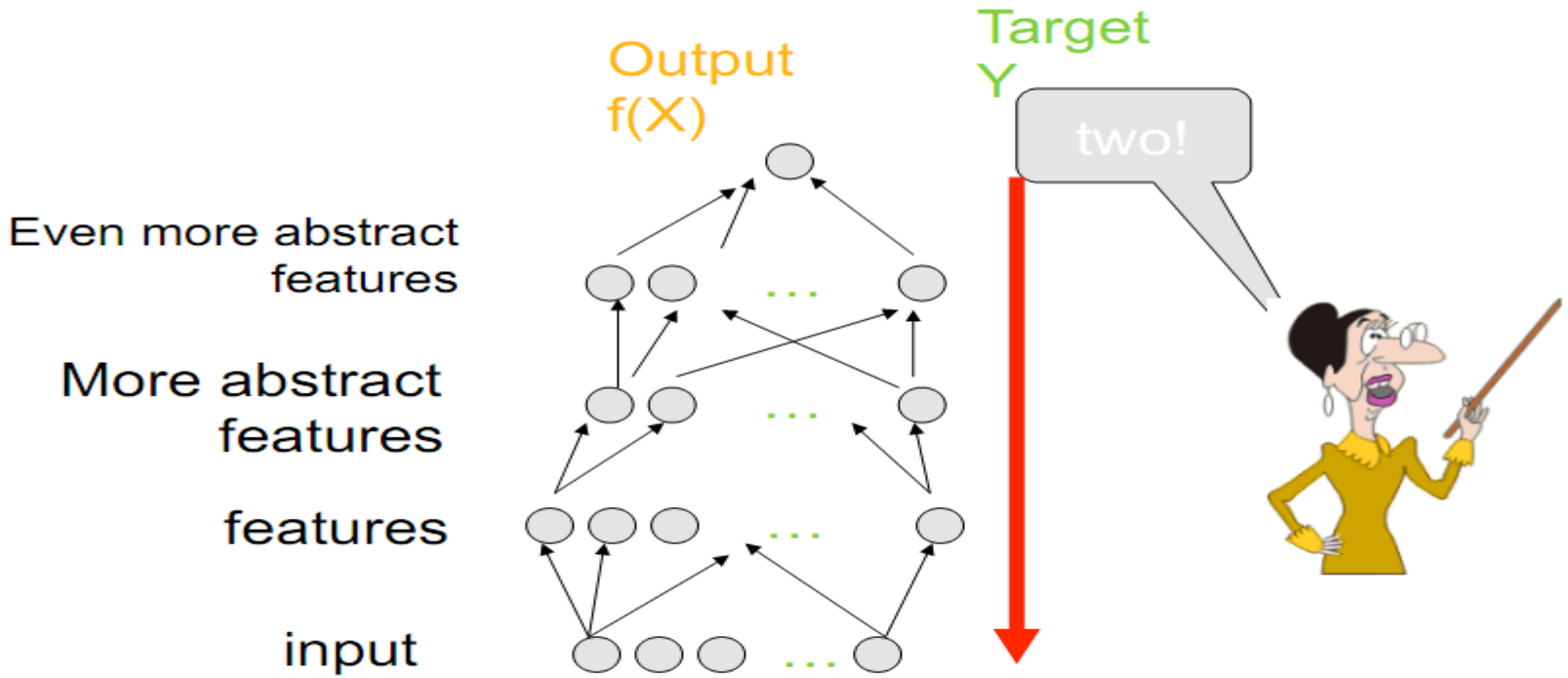
More abstract features

features

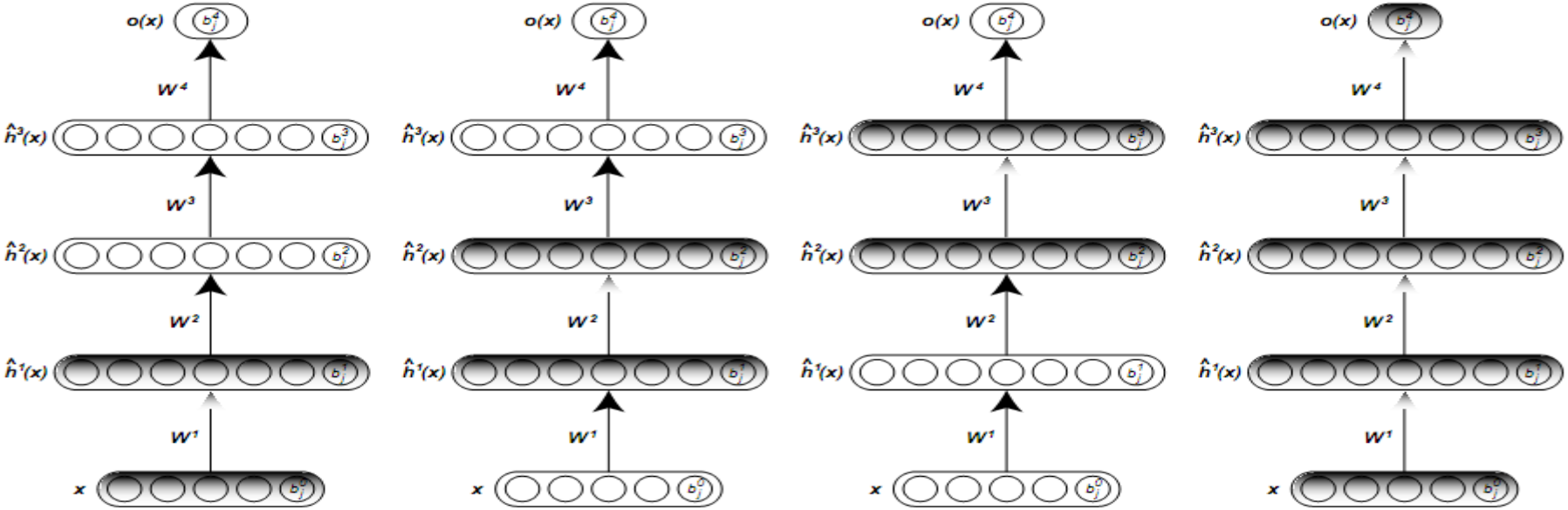
input



Supervised Fine-Tuning



Solution 1: Unsupervised greedy layer-wise training procedure



(a) First hidden layer pre-training

(b) Second hidden layer pre-training

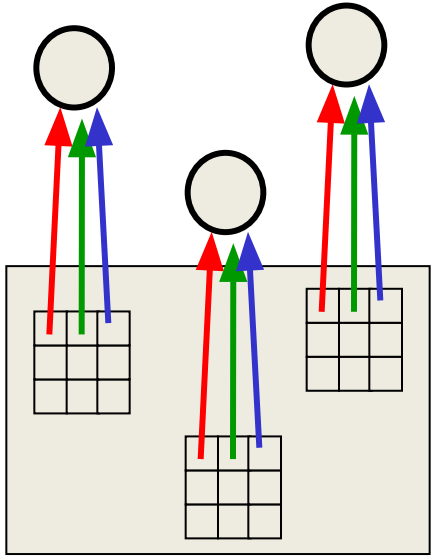
(c) Third hidden layer pre-training

(d) Fine-tuning of whole network

Solution2: The replicated feature approach
(currently the dominant approach for neural networks)

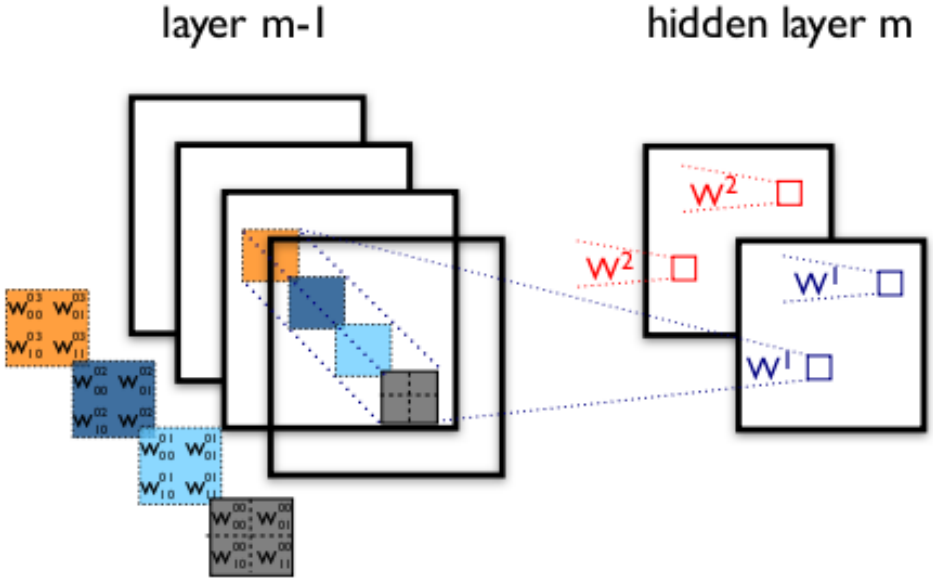
- Use many different copies of the same feature detector with different positions.
  - Could also replicate across scale and orientation (tricky and expensive)
  - Replication greatly reduces the number of free parameters to be learned.
- Use several different feature types, each with its own map of replicated detectors.
  - Allows each patch of image to be represented in several ways.

The red connections all have the same weight.



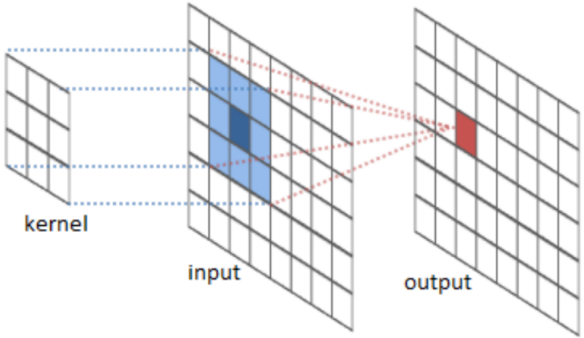
Slide credit: Geoffrey Hinton

Solution 2: Convolutional neural networks



W_{ij}^{kl} = weight connecting each pixel of the k-th feature map at layer m, with the pixel at coordinates (i,j) of the l-th feature map of layer (m-1).

Solution 2: Learned Filters



	0	0	0	
	-1	1	0	
	0	0	0	

The key ideas behind convolutional neural networks

1. Image statistics are translation invariant (objects and viewpoint translates)
   - build this translation invariance into the model (rather than learning it)
   - tie lots of the weights together in the network
   - reduces number of parameters
2. Expect learned low-level features to be local (e.g. edge detector)
   - build this into the model by allowing only local connectivity
   - reduces the numbers of parameters further
3. Expect high-level features learned to be coarser (c.f. biology)
   - build this into the model by subsampling more and more up the hierarchy
   - reduces the number of parameters again

Backpropagation with weight constraints

- It's easy to modify the backpropagation algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$To\ \ constrain: \quad w_1 = w_2$$

$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

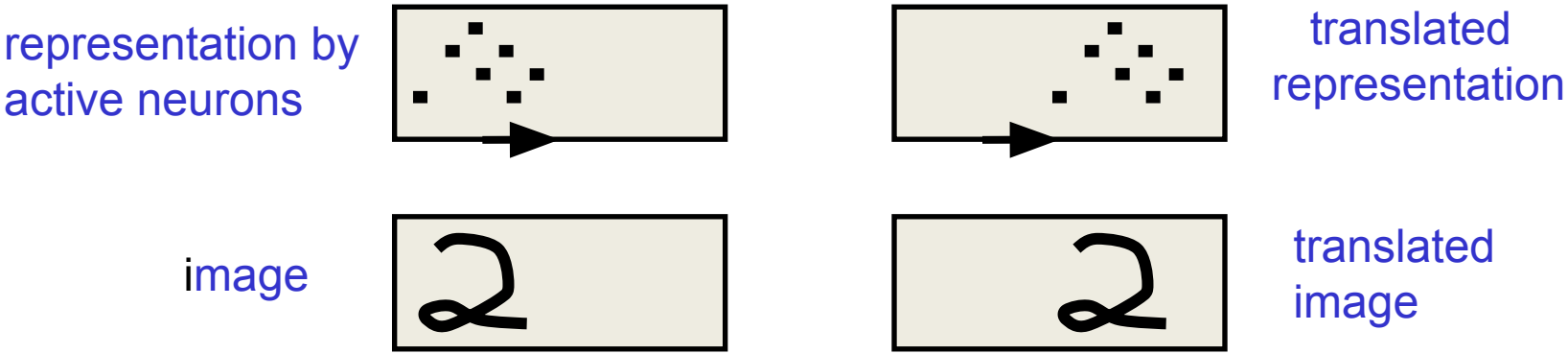
$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

Slide credit: Geoffrey Hinton

What does replicating the feature detectors achieve?

- Equivariant activities: Replicated features do not make the neural activities invariant to translation. The activities are equivariant.



- Invariant knowledge: If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.

Slide credit: Geoffrey Hinton

Priors and Prejudice

- We can put our prior knowledge about the task into the network by designing appropriate:
  - Connectivity.
  - Weight constraints.
  - Neuron activation functions
- This is less intrusive than hand-designing the features.
  - But it still prejudices the network towards the particular way of solving the problem that we had in mind.
- Alternatively, we can use our prior knowledge to create a whole lot more training data.
  - This may require a lot of work (Hofman&Tresp, 1993)
  - It may make learning take much longer.
- It allows optimization to discover clever ways of using the multi-layer network that we did not think of.
  - And we may never fully understand how it does it.

Slide credit: Geoffrey Hinton

ConvNets - Results

ImageNet - 1 million images, 1000 categories.

2012 -> SuperVision 16.4% error (top-5), Next best (non-convnet) 26.2% error

2013 -> Clarifai 11.7% error

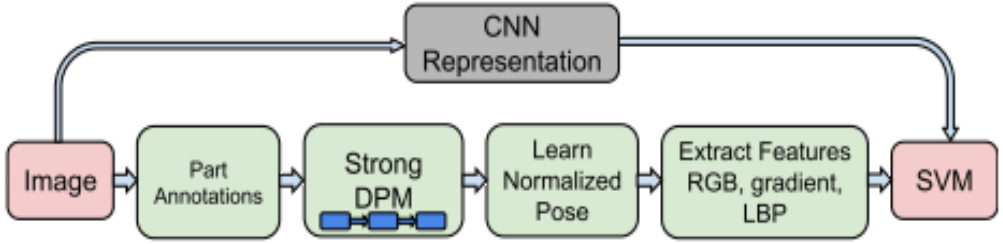
2014 -> GoogLeNet 6.7%

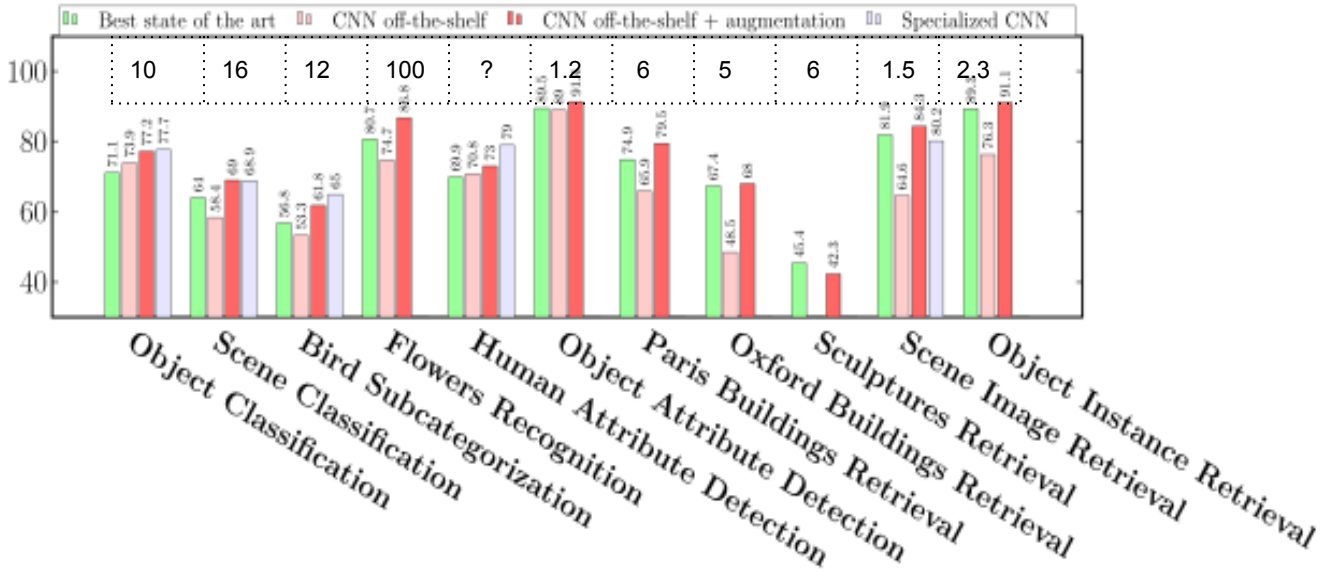
-------> Human Performance 5.1%

1 month ago -> ConvNet achieves 4.8%

ConvNet - Results

CNN Features off-the-shelf:
an Astounding Baseline for Recognition





OverFeat trained on 1M labelled images of 1000 object classes.

Project target dataset (10k, 15k, 11k, 80k,) images into OverFeat feature space.

Supervisedly train a linear classifier on feature vectors.

Why multi-instance learning?

It is not always possible to provide labeled data for training:

- Requires substantial human effort
- Requires expensive tests
- Disagreement among experts
- Labeling is not possible at instance level

Objective: present a learning algorithm that can learn from ambiguously labeled training data

Standard Learning vs. Multiple Instance Learning

Standard supervised learning

Optimize some model (or learn a target concept) given training samples and corresponding labels

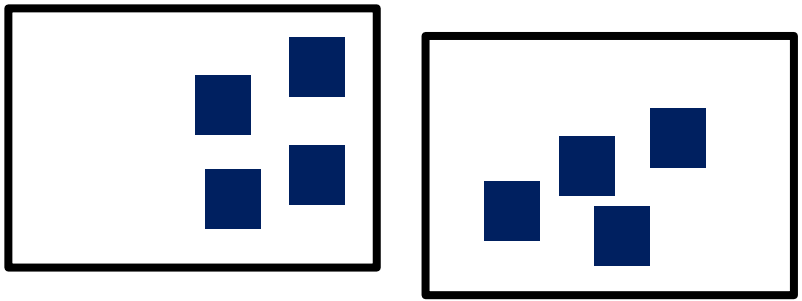
MIL

Learn a target concept given **multiple sets** of samples and corresponding labels for the **sets**.
Interpretation: Learning with uncertain labels / noisy teacher

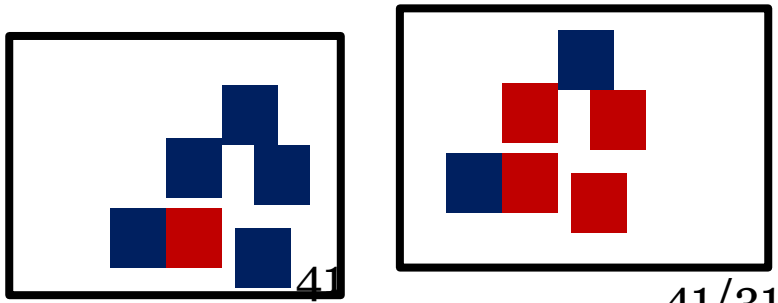
Learning from Bags

- In MIL, a label is attached to a bag
- A bag is labeled as positive if and only if at least one of its instances is positive.

NEGATIVE BAGS
(Each bag is an image)



POSITIVE BAGS
(Each bag is an image)



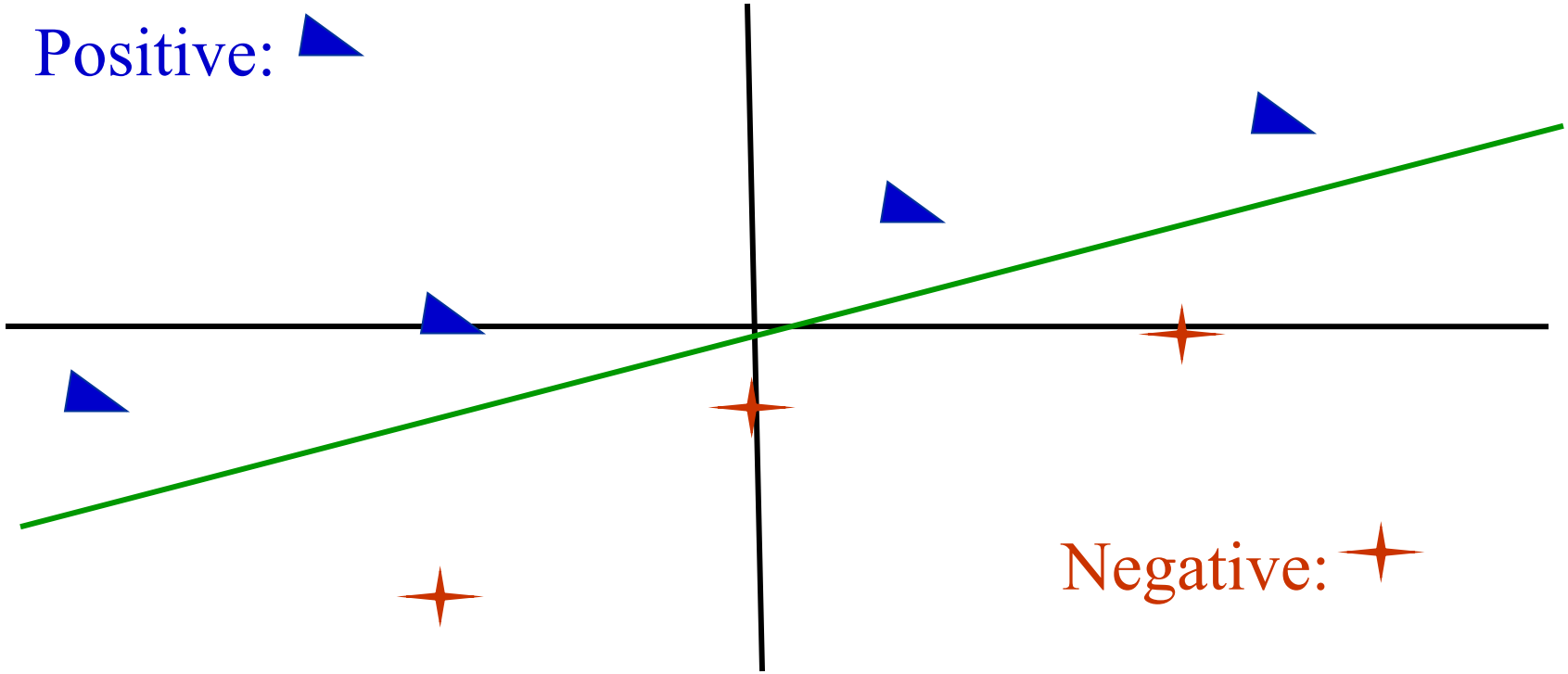
Noisy-Or for Estimating the Density

- It is assumed that the event can only happen if at least one of the causations occurred
- It is also assumed that the probability of any cause failing to trigger the event is independent of any other cause

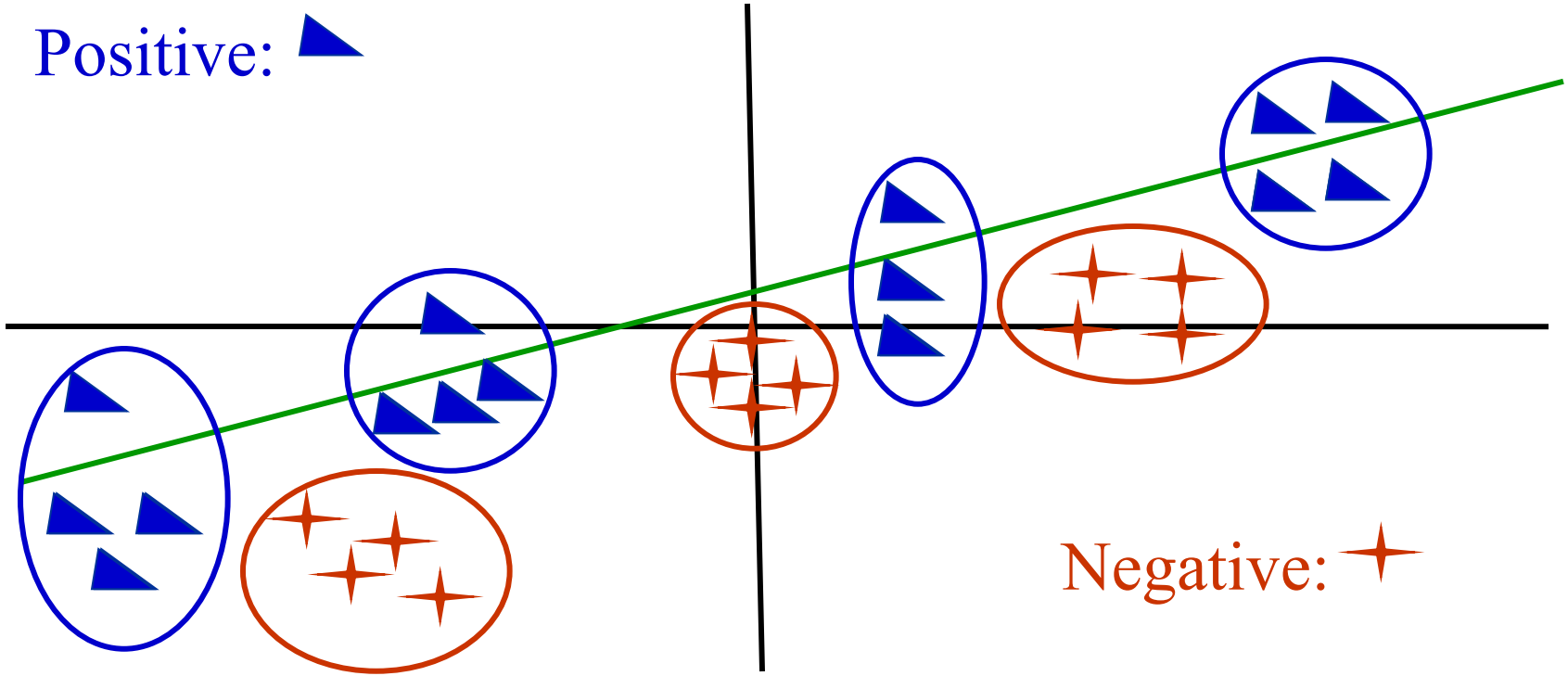
$$\Pr(x = t|B_i^+) = \Pr(x = t|B_{i1}^+, B_{i2}^+, \ldots)$$

$$\Pr(x = t|B_i^+) = 1 - \prod_j (1 - \Pr(x = t|B_{ij}^+))$$

Example: Standard Classification



Example: Multiple Instance Classification



Examine the distribution of the instances

- It looks for a point that is close to instances in different positive bags and that is far from the instances in the negative bags
- Such a point represents the concept that we would like to learn
- .

Single Point Concept

- A concept that corresponds to single point in feature space

- Every *Bi+* has at least one instance that is equal to the true concept corrupted by some Gaussian noise.
- Every *Bi-* has no instances that are equal to the true concept corrupted by some Gaussian noise

Single Point Concept

- A concept that corresponds to single point in feature space

- Every *Bi+* has at least one instance that is equal to the true concept corrupted by some Gaussian noise.
- Every *Bi-* has no instances that are equal to the true concept corrupted by some Gaussian noise

So, finally

Multi Instance Deep Learning

Given sufficiently powerful feature representation, multi instance learning also solved with classifier on top (and a noisy max).

Multi Instance Deep Learning

Given sufficiently powerful feature representation, multi instance learning also solved with classifier on top (and a noisy max).

Two immediate possibilities :

1. Softmax added on top of network (or any differentiable function of the features really) **- Cool option**
2. Create correct feature representation through active learning, given small, fully labelled, validation set **- Might be necessary if dataset too messy for the cool option**

Thank you

Razvan Ranca - tractable.io