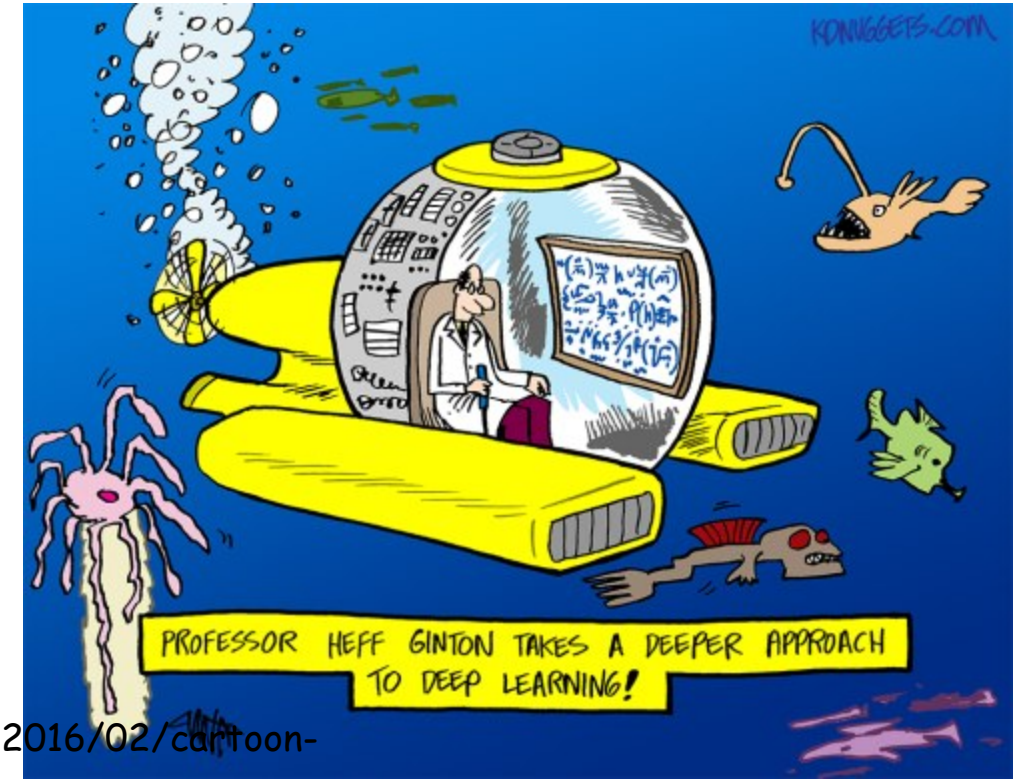
# Deep learning




http://www.kdnuggets.com/2016/02/cartoon-deeper-deep-learning.html

# Deep learning

Some of the most iconic companies are investing heavily in deep learning

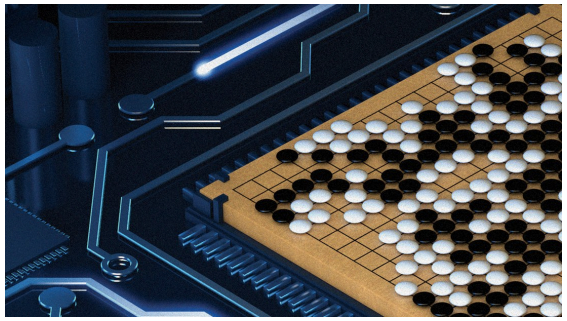

https://www.tensorflow.org/


PyTorch

https://research.facebook.com/ai/


https://deepmind.com/

AlphaGo



http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

# Why this interest in deep learning?

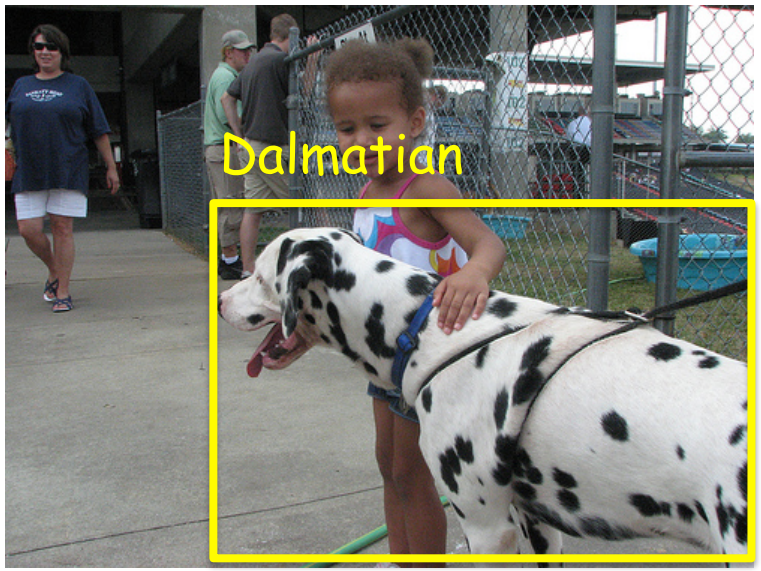
The 2012 ImageNet visual recognition challenge:

- 1000 classes, 1,431,167 images

Geoff Hinton's group: 16% error using convolutional neural networks

Closest competitor: 26%

Current level of error: around 5%




http://www.image-net.org/

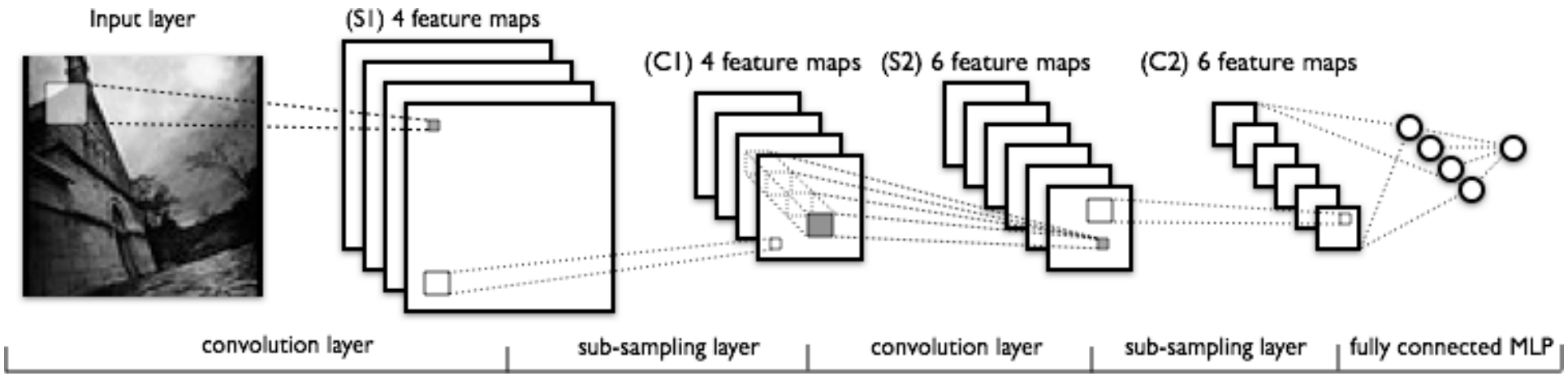
# Deep learning

Deep learning: neural architectures with lots of layers.

An umbrella name for lots of different network architectures

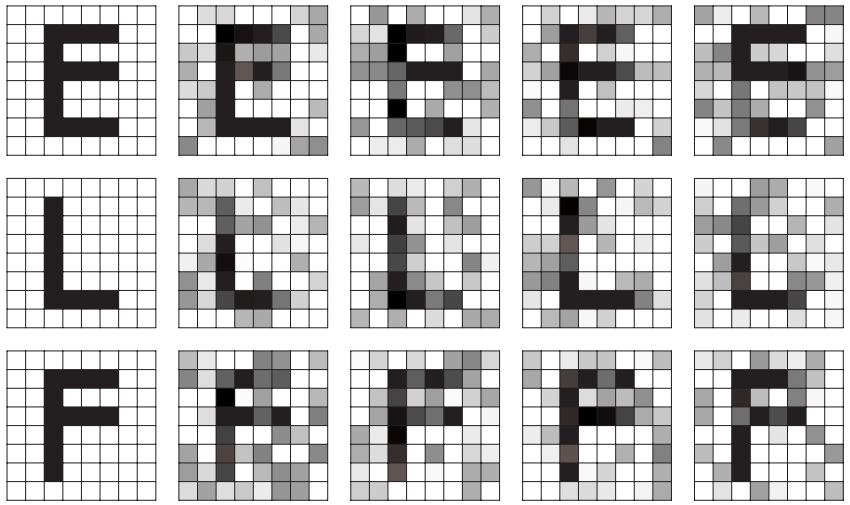
A neural network with a single hidden layer can approximate any function.

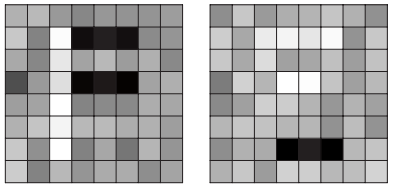
However, a network with multiple layers can represent the target function more efficiently.



# Interpreting network weights

sample training patterns



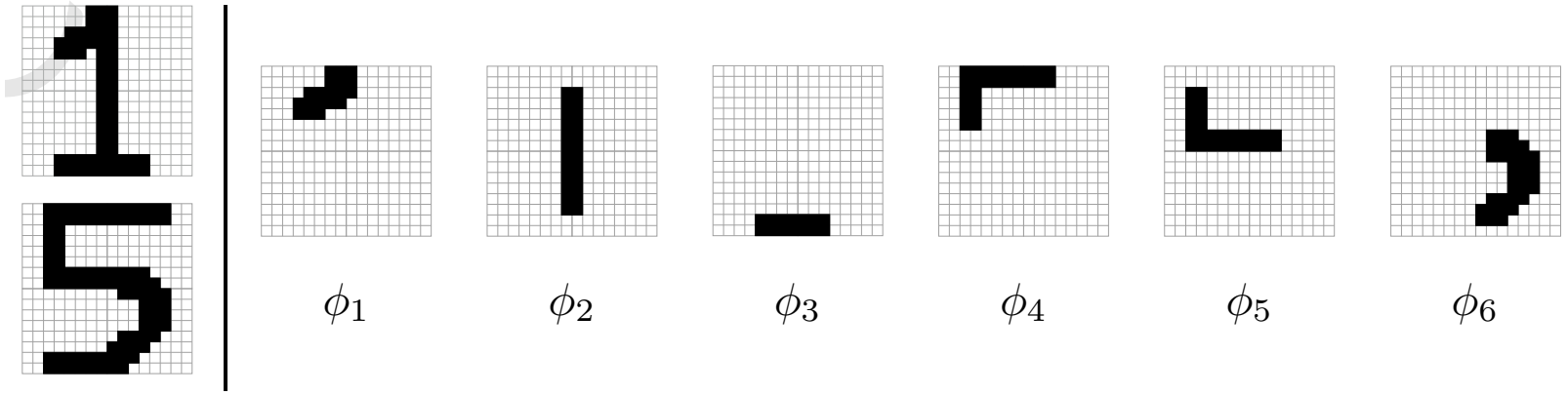


learned input-to-hidden weights

Top: images used to train a 64-2-3 network
Bottom: the weights associated with each of the two hidden units after training.

# Decomposing a learning problem

Suppose we would like to learn to distinguish between the digits '1' and '5'



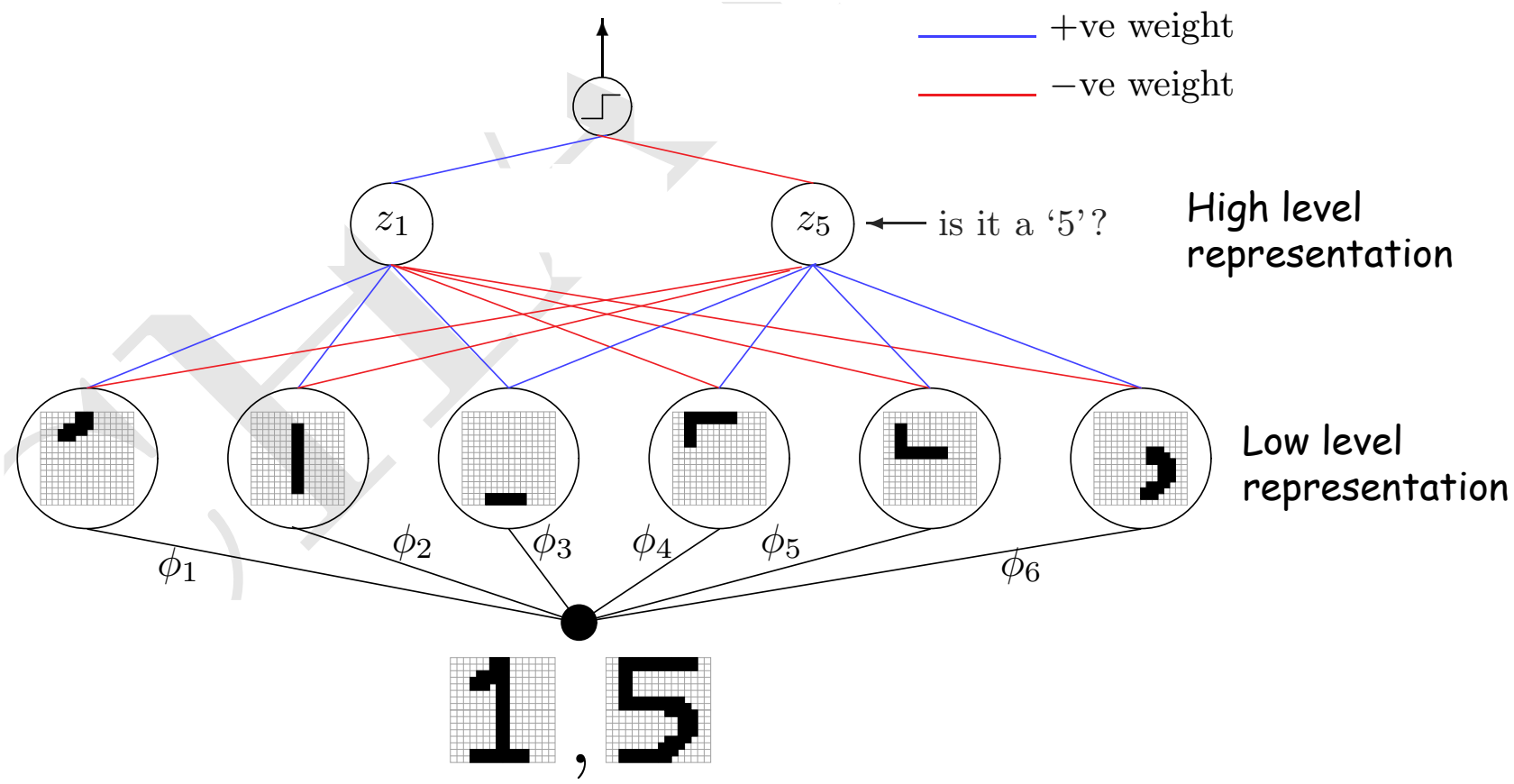

Decompose the digits into small components that characterize them.

A '1' should contain features 1 and 2.

A '5' should contain 3,4,5,6

# Constructing a network

Given the features we can construct a network:

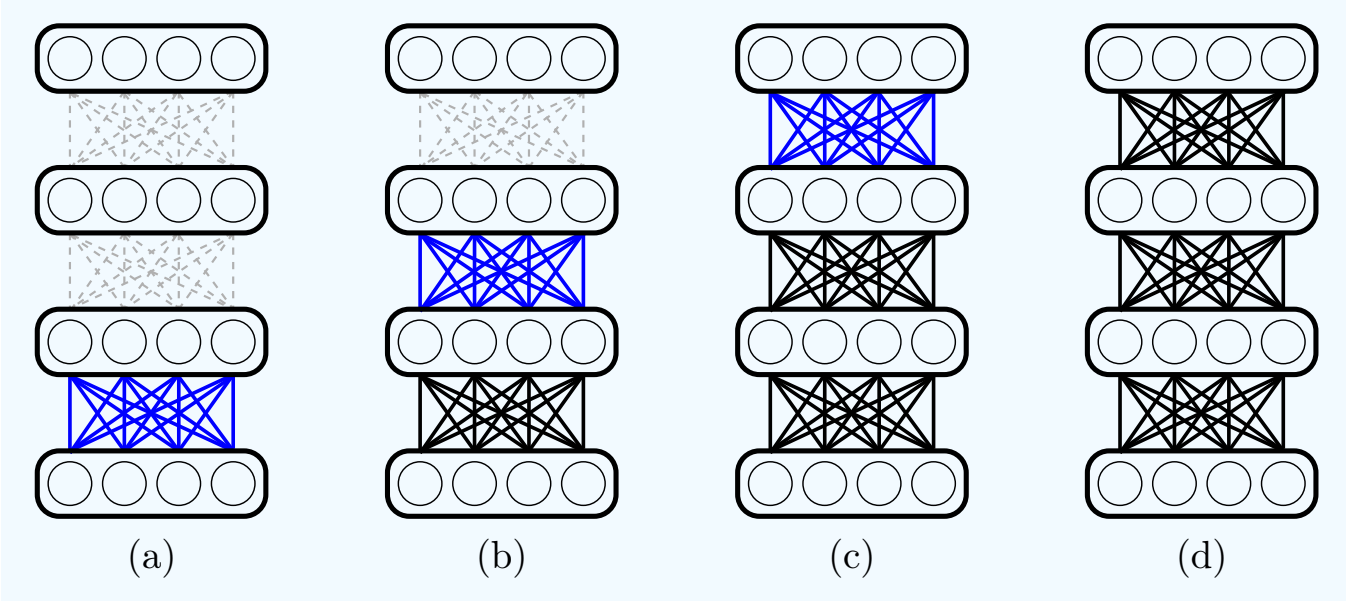



How do we automate the process?

# Training deep networks

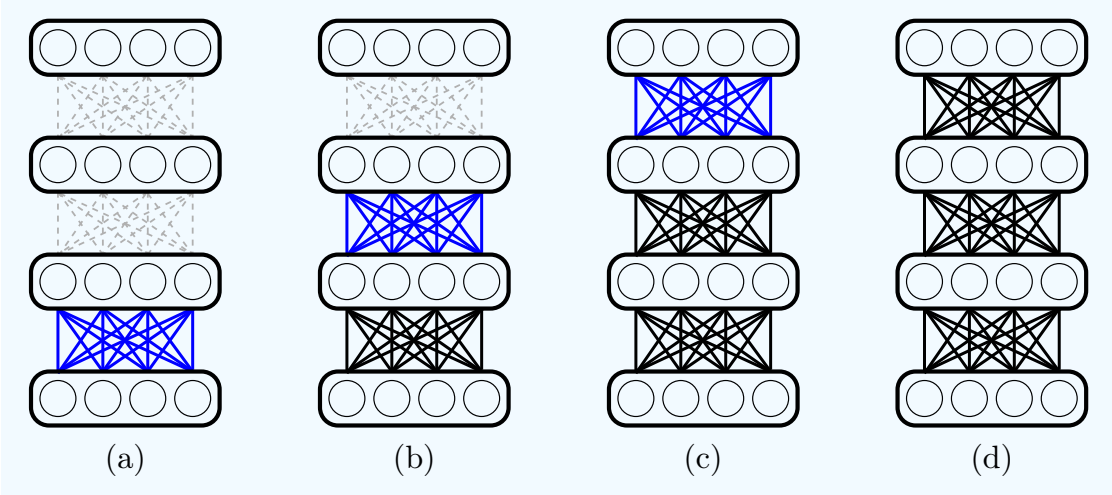
It is difficult to train very deep networks.

Alternative: training layer by layer




At each step only the weights of a single layer are optimized

# Training deep networks




**Greedy Deep Learning Algorithm:**

1: **for** $\ell = 1, \ldots, L$ **do**

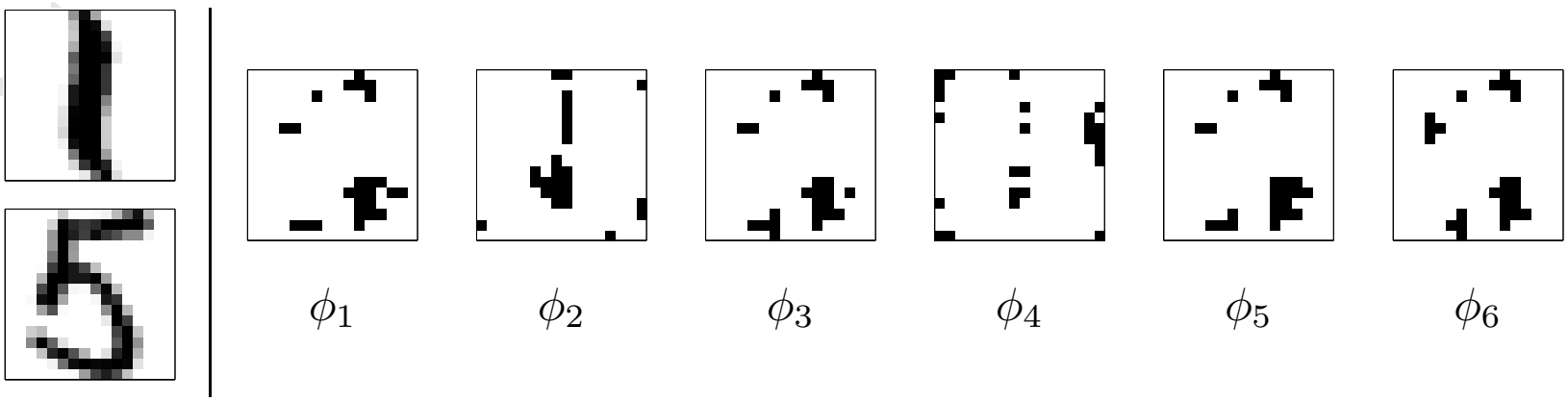
2: $\mathrm{W}^{(1)} \cdots \mathrm{W}^{(\ell-1)}$ are given from previous iterations.

3: Compute layer $\ell - 1$ outputs $\mathbf{x}_n^{(\ell-1)}$ for $n = 1, \ldots, N$.

4: Use $\{\mathbf{x}_n^{(\ell-1)}\}$ to learn weights $\mathrm{W}^{\ell}$ by training a *single* hidden layer neural network. ($\mathrm{W}^{(1)} \cdots \mathrm{W}^{(\ell-1)}$ are fixed.)

# Deep networks for the 1 vs 5 problem

Features learned using a network with 3 layers with 6,2,1 neurons per layer (digits are 16x16 pixels)



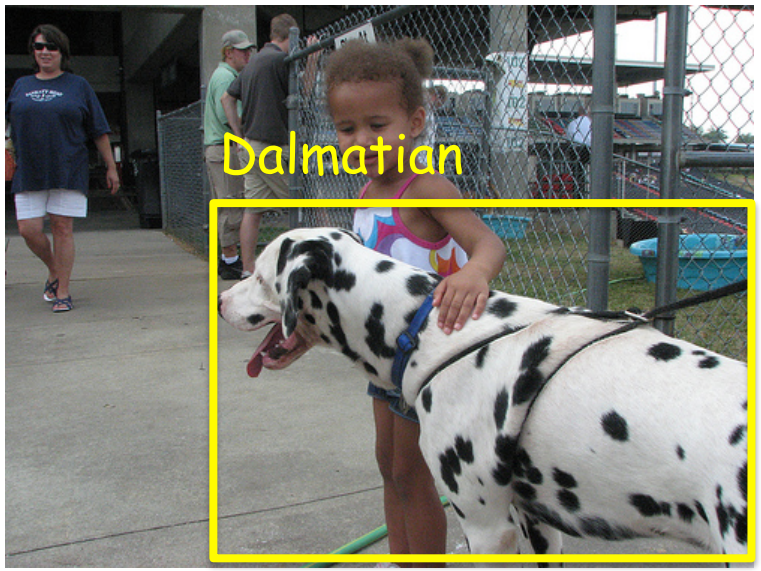
# So have we solved image classification?

Not yet!

The datasets we we looked at were just too easy.

Small images where the digit was centered.

As the images become larger, the network needs to have more and more parameters.




http://www.image-net.org/

# The invariance problem

Our perceptual system is very good at dealing with invariances

- translation, rotation, scaling
- deformation, contrast, lighting, rate

We are so good at this that it's hard to appreciate how difficult it is.

- Dealing with invariances is one of the main difficulties in making computers solve perceptual problems.

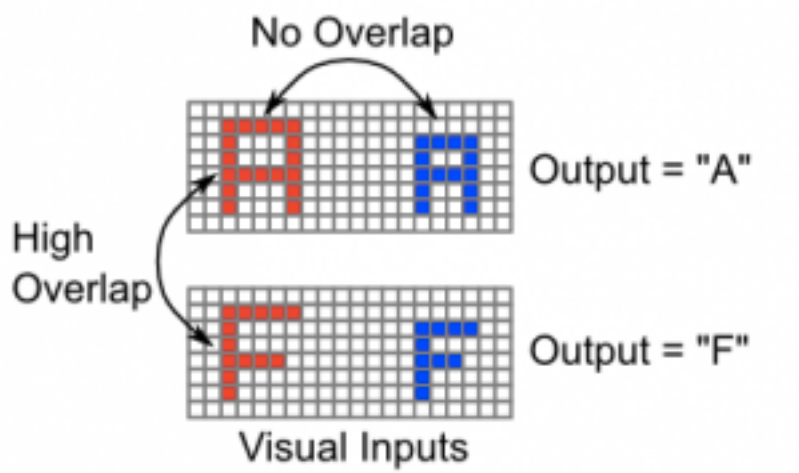



Image from https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Perception

# How to make a classifier invariant

Potential solution: Introduce transformed variants of the data. With sufficient training data the network can learn the invariances by itself.

Alternatives:

- Construct features that have the required invariances
- Build the invariances into the model

# The human visual system

The human visual system performs image processing at increasing levels of abstraction

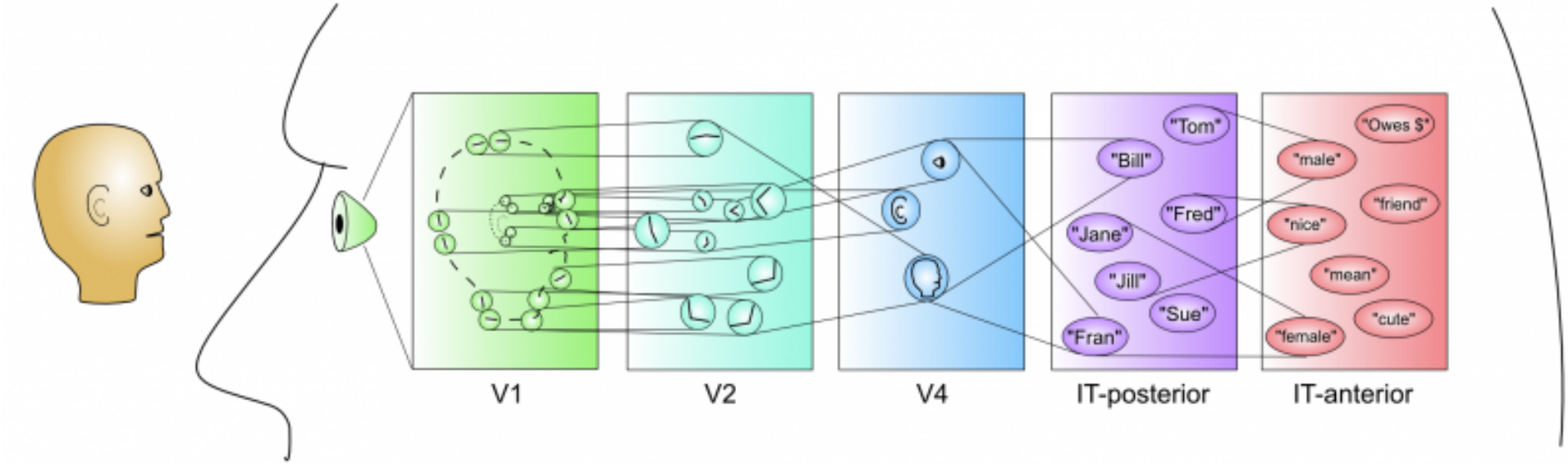
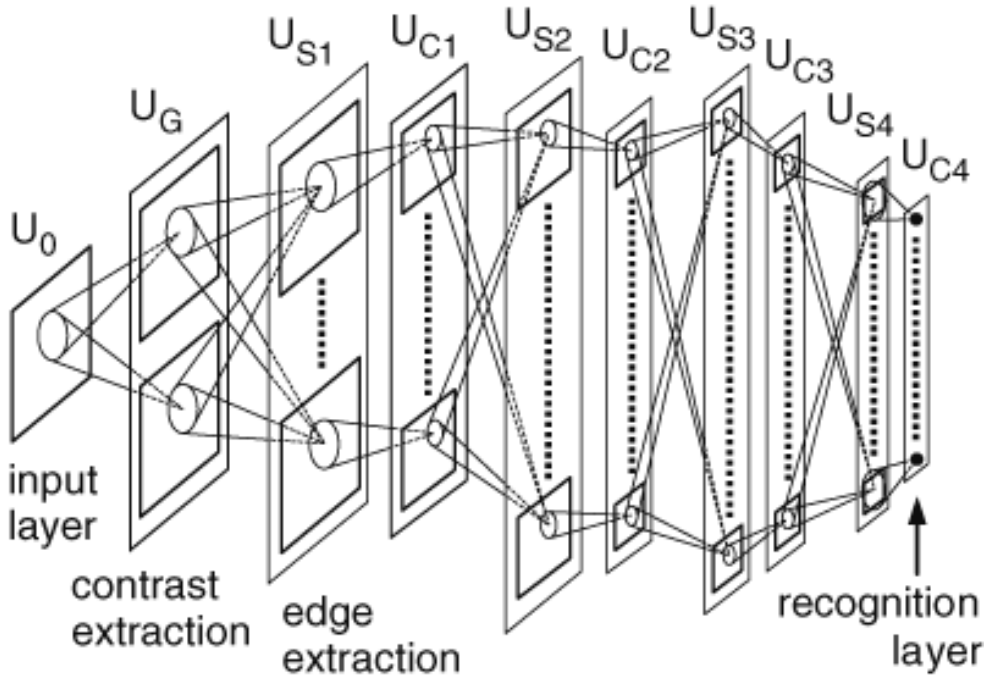



Image from https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Perception

# NN architectures inspired by the brain

For image classification researchers have been exploring architectures that are motivated by the working of the visual system.

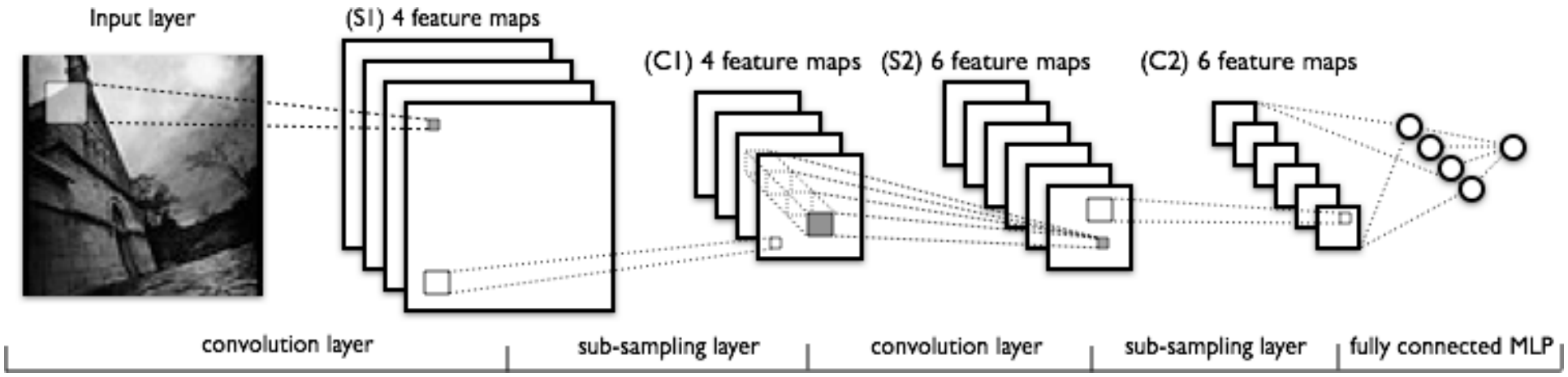



The architecture of the Neocognitron (Fukushima, 1980)

http://www.scholarpedia.org/article/Neocognitron

# convolutional networks

The LeNet-5 network:




Important ideas:

Local features that are useful in one region are likely to be useful elsewhere (weight sharing)

Extract local features (local receptive fields) and combine them to create global features at a more abstract level

http://yann.lecun.com/exdb/lenet/

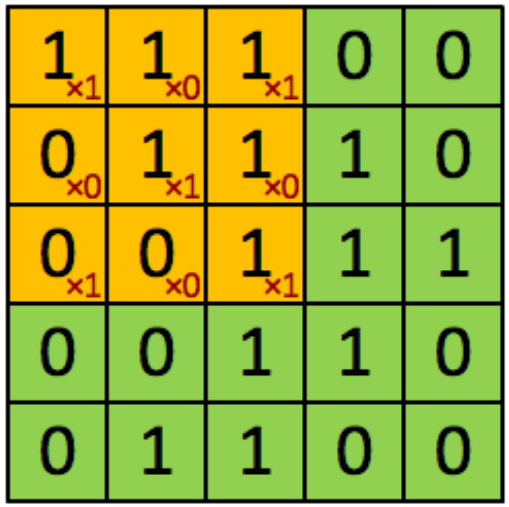
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998.

# convolution

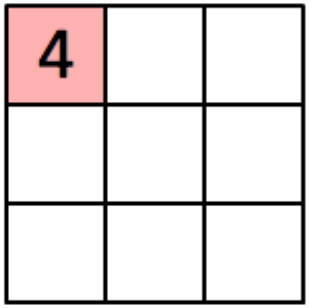
Move a 3x3 receptive field over an image, extracting local features at each position

The matrix associated with the receptive field (feature extractor):

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |




Image

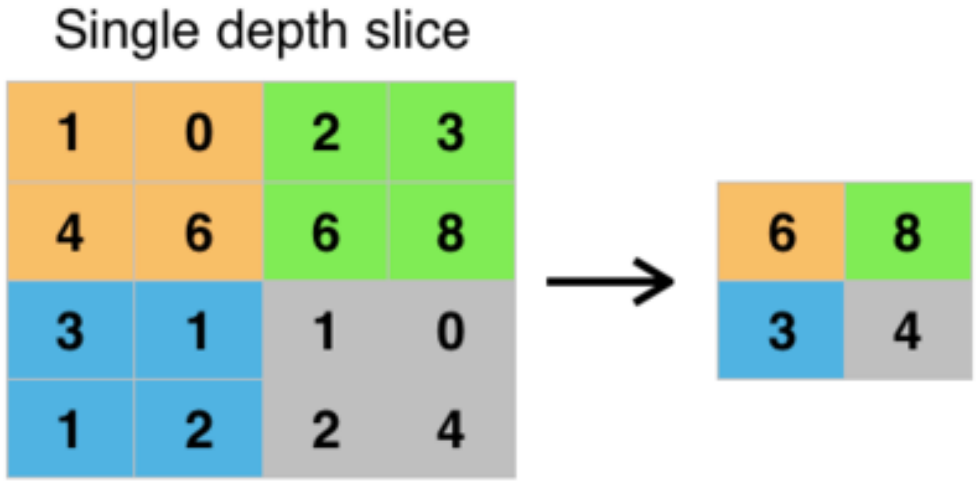



Convolved Feature

Image from http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/

# pooling

Once local features are computed using convolution, they are merged by pooling: combining all the convolved values into a much smaller set (usually just a maximum).

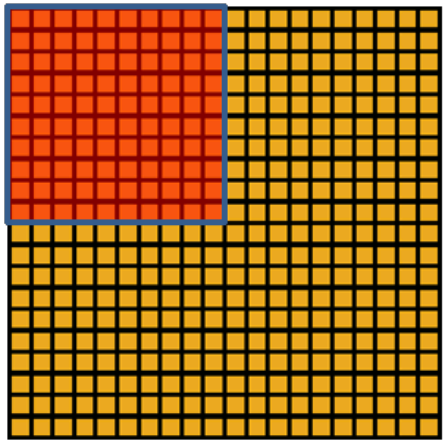


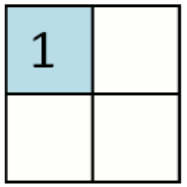

Max pooling with a 2x2 filter and stride = 2

Image from http://ufldl.stanford.edu/tutorial/supervised/Pooling/

# pooling

Once local features are computed using convolution, they are merged by pooling: combining all the convolved values into a much smaller set (usually just a maximum).





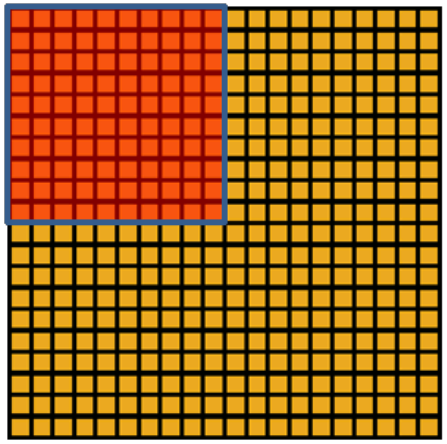

Convolved feature

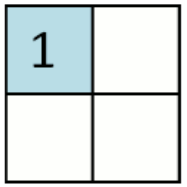
Pooled feature

Image from http://ufldl.stanford.edu/tutorial/supervised/Pooling/

# pooling

Pooling is a way of achieving invariance.





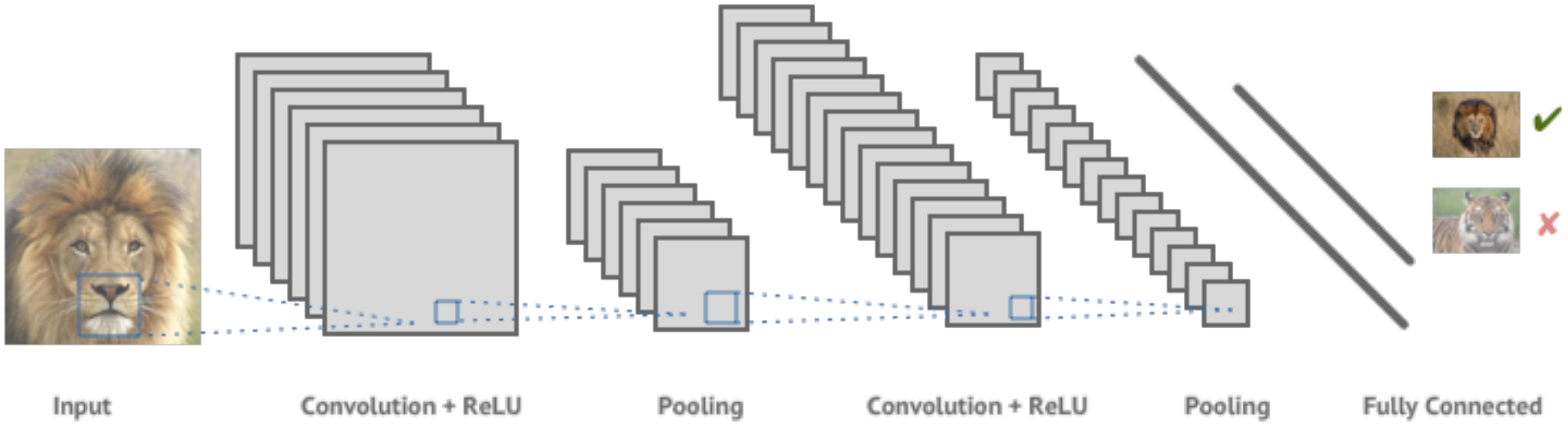

Convolved feature

Pooled feature

Image from http://ufldl.stanford.edu/tutorial/supervised/Pooling/

# Convolutional networks

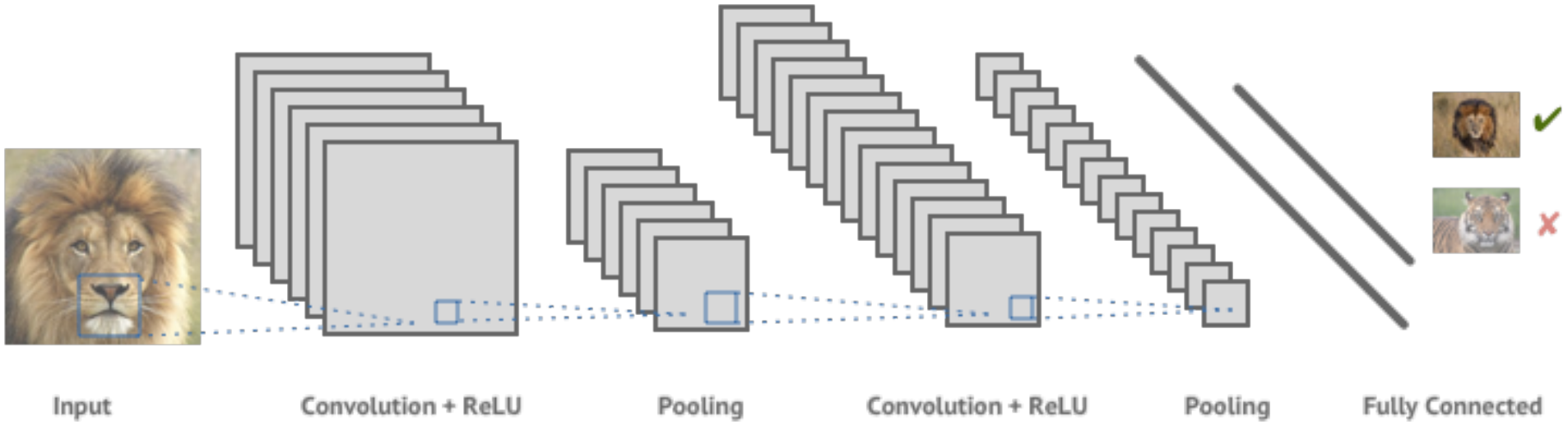
Let's put all the elements together:




Can have multiple convolution/pooling layers
and multiple classification layers

# Training convolutional networks

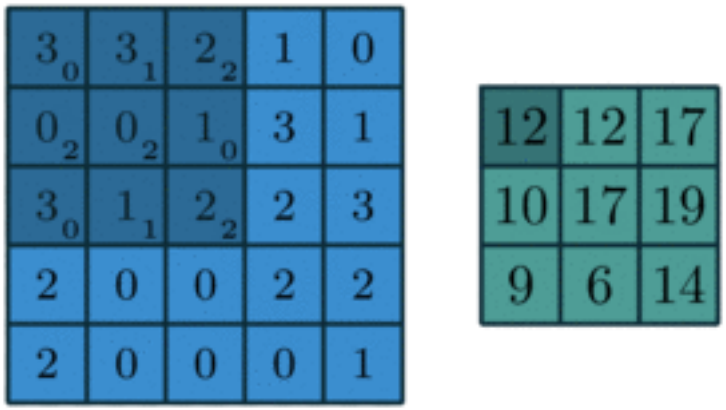
Guess what: backpropagation.




Need to modify the equations to model convolution/pooling

# some details...

When performing convolution we slide a *feature map* or *convolutional filter* across the image:



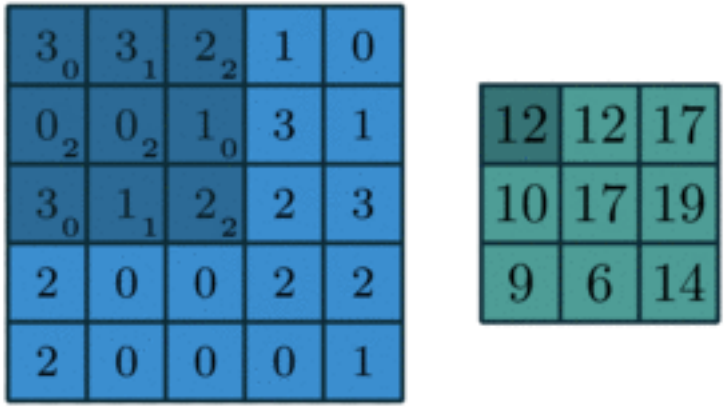

This can be repeated using different feature maps to form as many feature maps as we like as output

Figure from: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

See also: A guide to convolution arithmetic for deep learning
https://arxiv.org/abs/1603.07285

# some details...

When performing convolution we slide a *feature map* or *convolutional filter/kernel* across the image:




Size of the output as a function of size of the input and filter:

$$o = (i - k) + 1$$

- o – size of output
- i – size of input
- k – size of filter

# padding and stride

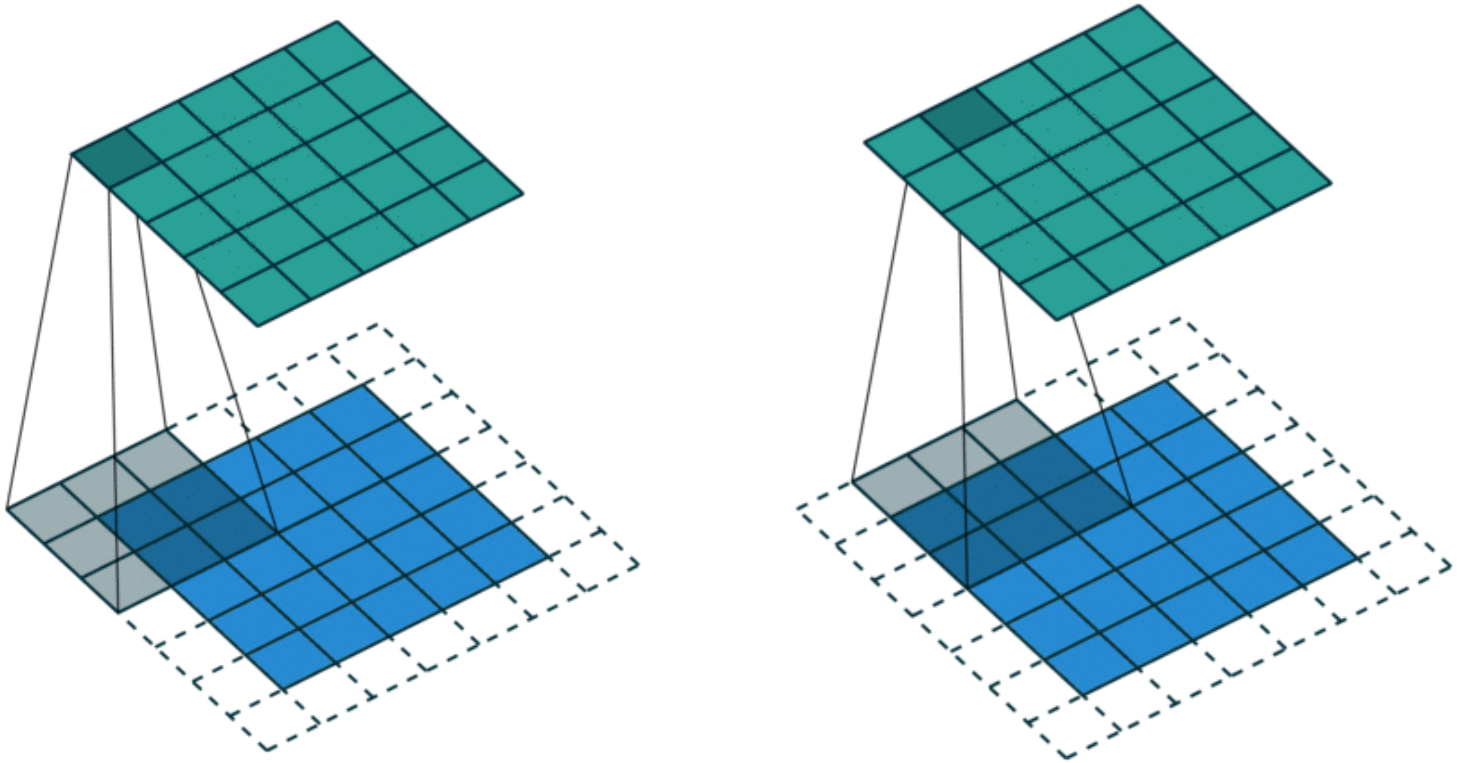
There are several parameters that control how a convolutional filter is moved across an image:

- stride (a form of sub-sampling)
- padding




# padding and stride

The combination of input size, filter size, padding and stride, determine the size of the output



For the special case where the padding equals half the filter size and the stride=1, the output is the same size as the input

# current optimization methods

- AdaGrad: Duchi, John; Hazan, Elad; Singer, Yoram. "Adaptive subgradient methods for online learning and stochastic optimization". JMLR. 12: 2121–2159, 2011.
- RMSProp
- Adam: Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR, 2015.
- Review: Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016). http://ruder.io/optimizing-gradient-descent/

- Bottom line: Adam is a standard approach; often a good idea to compare to SGD+Nesterov momentum

# deep learning software

Theano.

- allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
- tight integration with numpy
- transparent use of GPUs
- efficient symbolic differentiation
- dynamic C code generation

theano

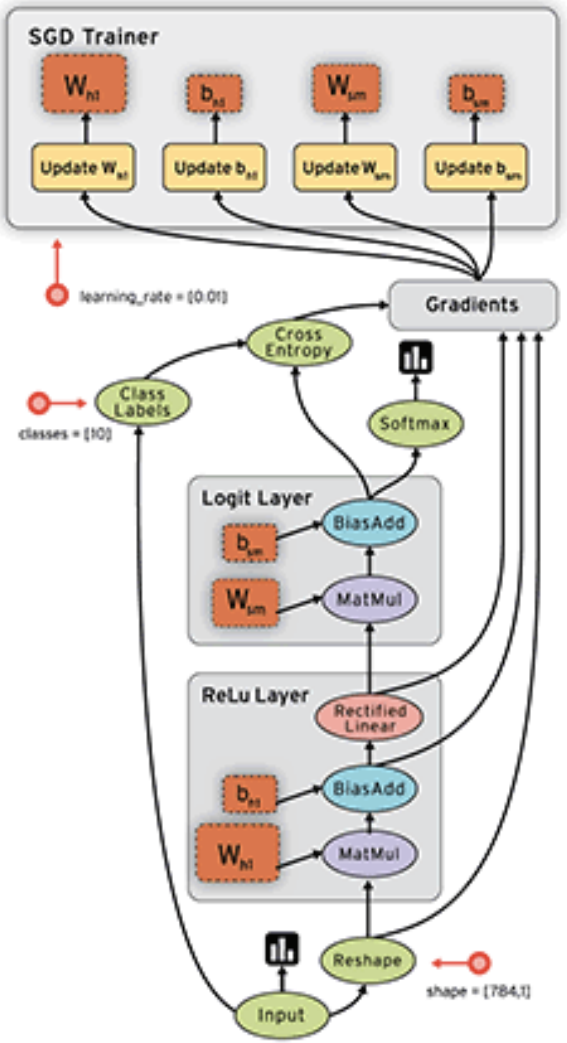
http://deeplearning.net/software/theano/

# deep learning software

TensorFlow (google)


PyTorch (facebook)


Similar features to Theano



# convolutional networks

CNNs can be applied to variable-sized inputs

Provide state-of-the-art performance in several domains:

- Object recognition in images
- Natural language processing (1d convolution)
- Speech processing (1d convolution)

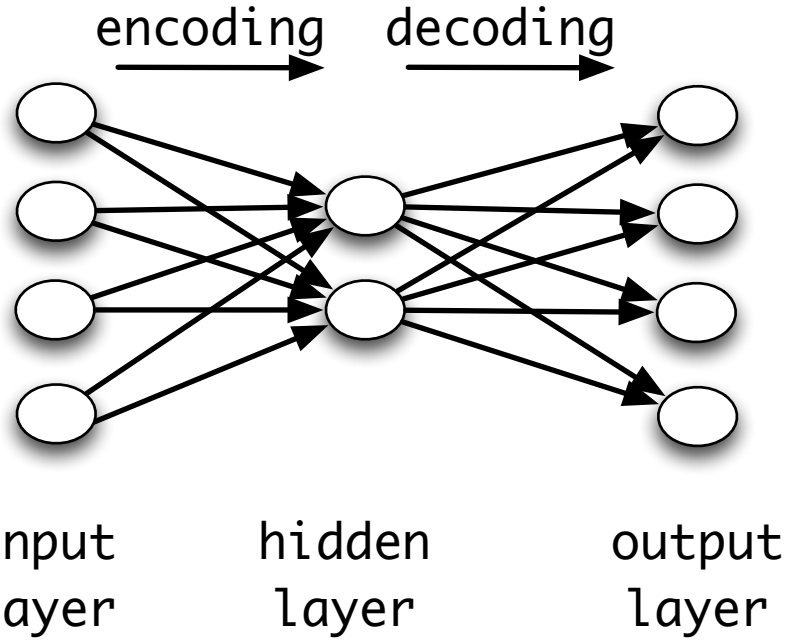
Also being applied in computational biology.

Issues: require lots of training data for good performance.

# auto-encoders

A framework for learning features in an unsupervised manner.

The idea: reconstruct the input using features computed by the hidden layer.

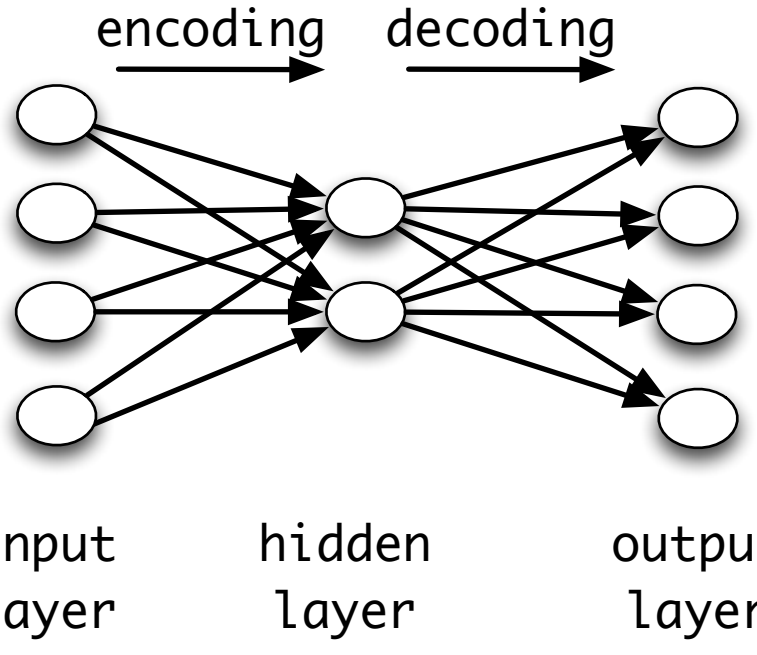



Want to learn a function h such that: $h_{\mathbf{w}}(\mathbf{x}) \approx \mathbf{x}$

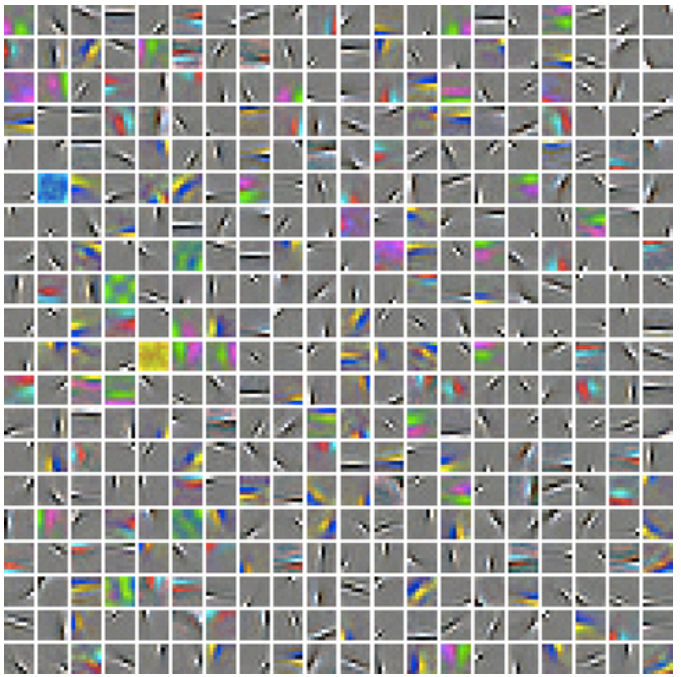
The reconstruction error: $\mathbf{e}_n = ||\hat{\mathbf{x}}_n - \mathbf{x}_n||^2$

# auto-encoders

When trained effectively, auto-encoders learn interesting features that characterize the input



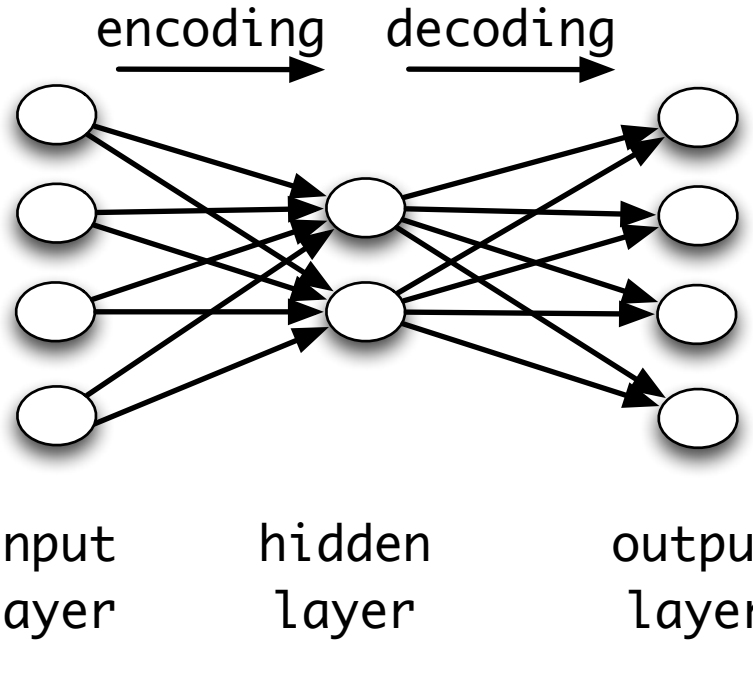

Features learned by an auto-encoder

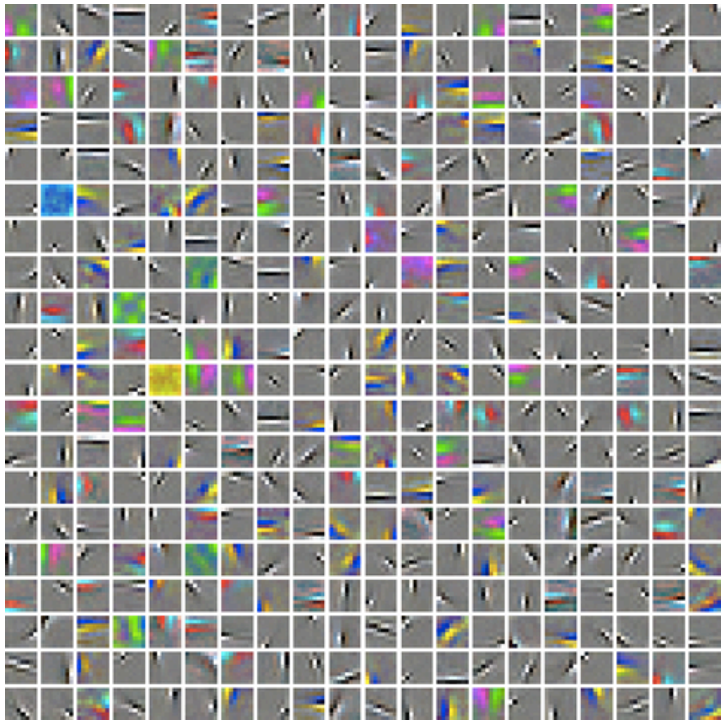


At the end, use the encoding function as features for supervised learning.

# auto-encoders

When trained effectively, auto-encoders learn interesting features that characterize the input

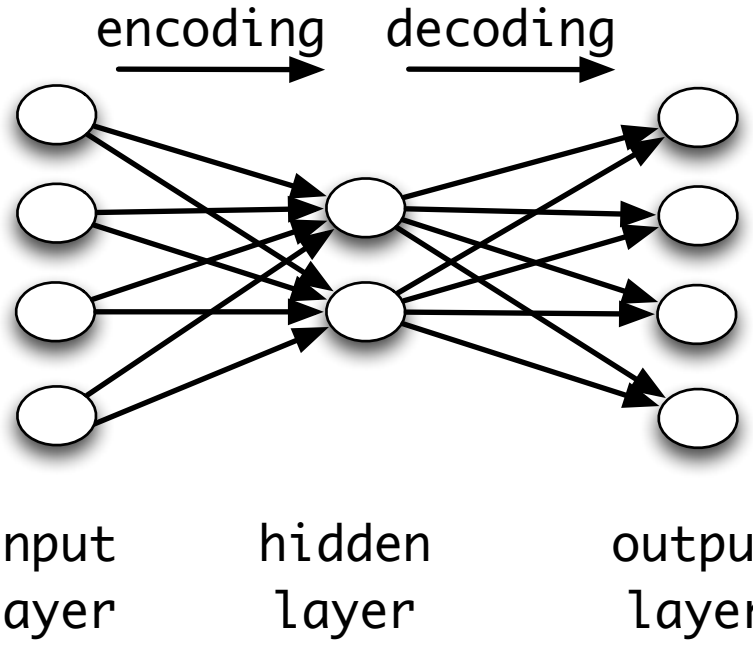


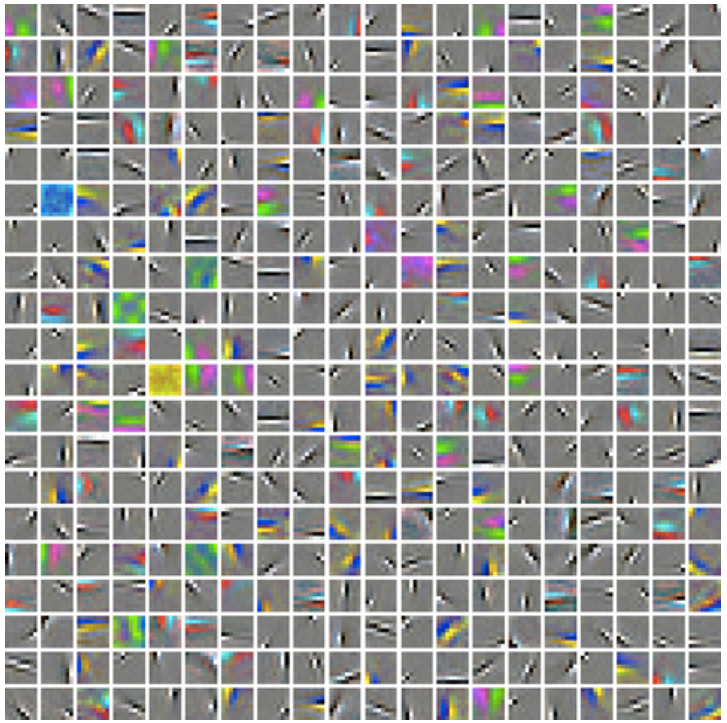



Since there are less neurons in the hidden layer, the network is forced to learn a compressed version of the input.

# auto-encoders

When trained effectively, auto-encoders learn interesting features that characterize the input

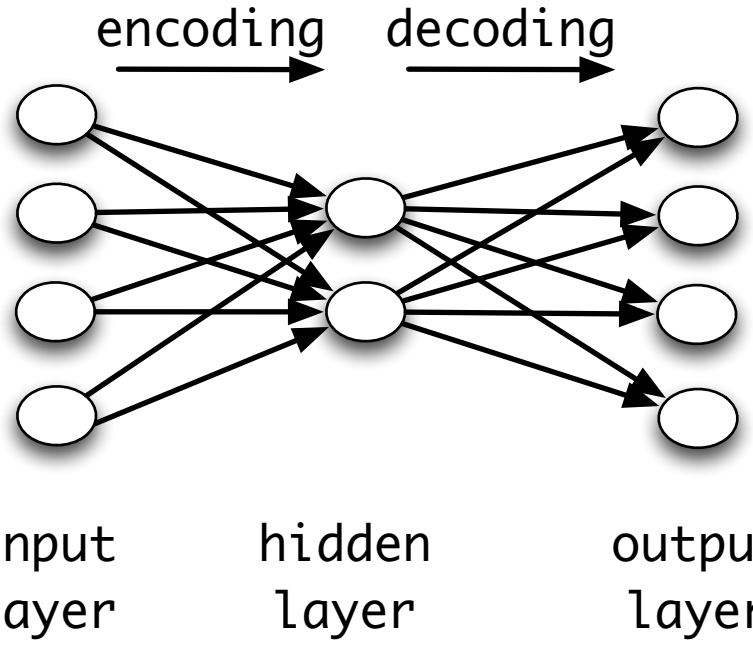


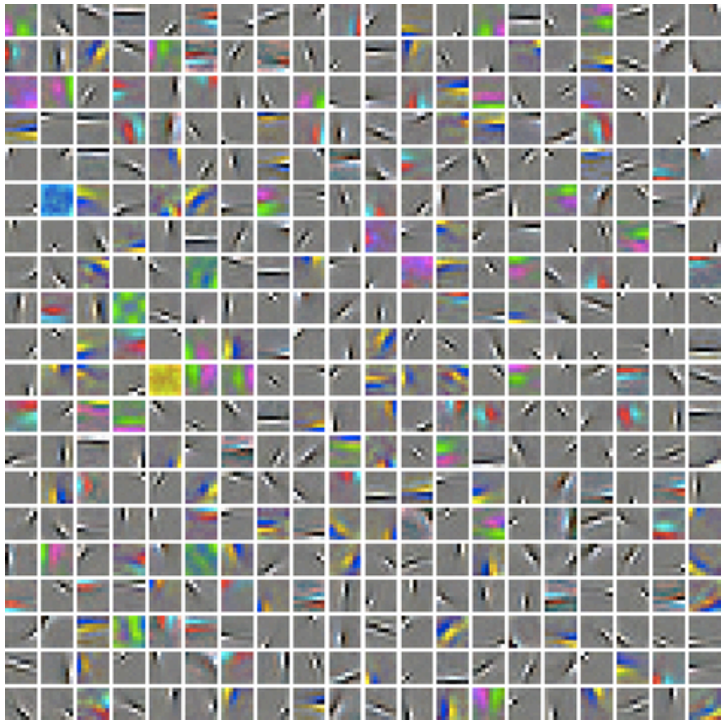



Can also achieve that by adding a sparsity-inducing penalty to the neural network error function.

# auto-encoders

When trained effectively, auto-encoders learn interesting features that characterize the input



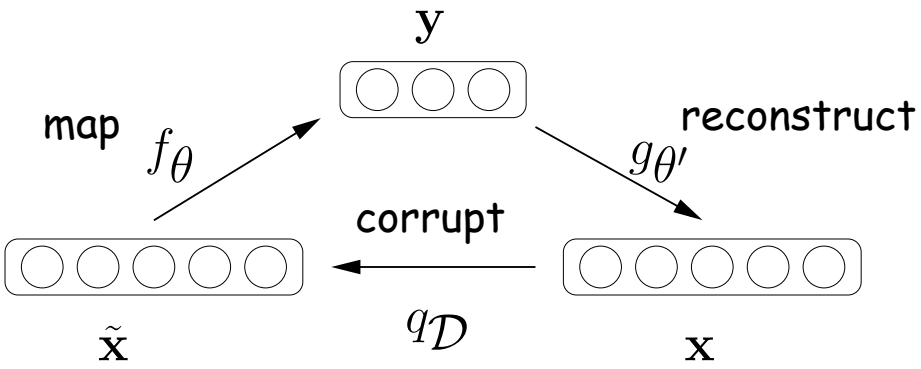



Stacked auto-encoders: multiple layers, trained by the greedy layer-wise training algorithm

# denoising auto-encoders

Learn to reconstruct a corrupted version of the input

Intuition: introduces robustness into the learned representation.



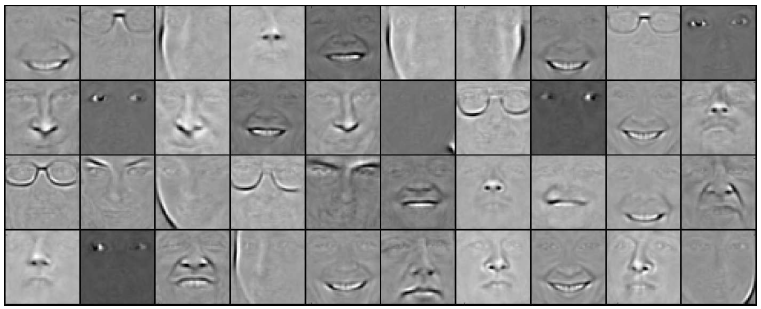

**x** is corrupted; the autoencoder maps it to **y**, and attempts to reconstruct **x**

Note: corruption introduced only in training

Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

# auto-encoders and CNNs

The ideas of auto-encoders and convolutional networks can be combined: convolutional auto-encoders



Masci, Jonathan, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber.
"Stacked convolutional auto-encoders for hierarchical feature extraction."
In Artificial Neural Networks and Machine Learning–ICANN 2011, pp. 52-59, 2011.

Makhzani, Alireza, and Brendan Frey. "A Winner-Take-All Method for Training Sparse Convolutional Autoencoders." Accepted, NIPS 2015.

# speeding up deep networks

Deep networks are computationally expensive to train

GPUs to the rescue:

Designed to speed up the computations performed in video games: conversion of a 3-d specification to what should be displayed on the screen.

They handle simple computations without much branching and process large memory buffers in parallel.

Exactly what's needed for deep networks!

NVIDIA: general purpose GPUs - C-like programming using CUDA. But still not easy to program.

# speeding up deep networks

Alternative to GPUs: parallelism using asynchronous stochastic gradient descent

SparkNet: Use Spark (MapReduce-like framework) to distribute the computation

SparkNet: Training Deep Networks in Spark
https://github.com/amplab/SparkNet

http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks

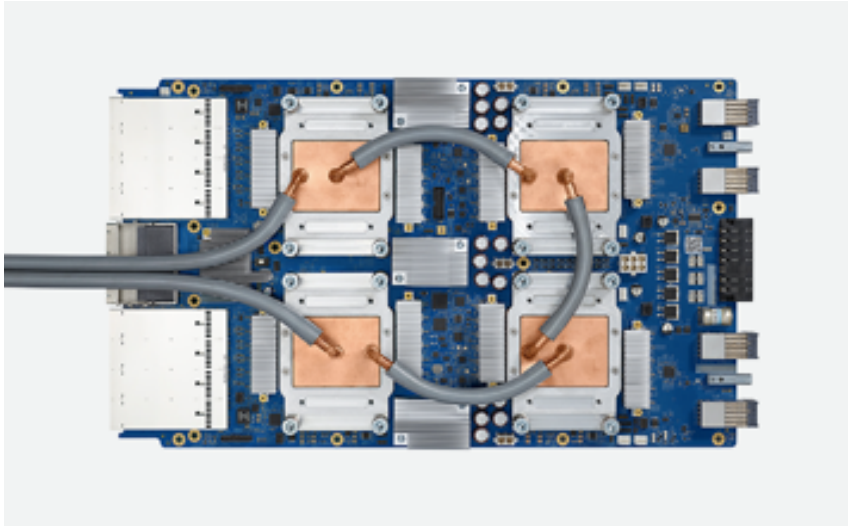
http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent

# From GPUs to TPUs

Google has recently unveiled a chip that is specifically designed for the tensor operations that are used in machine learning.

Idea: let's sacrifice precision for speed!

Claims to be an order of magnitude faster for ml applications.



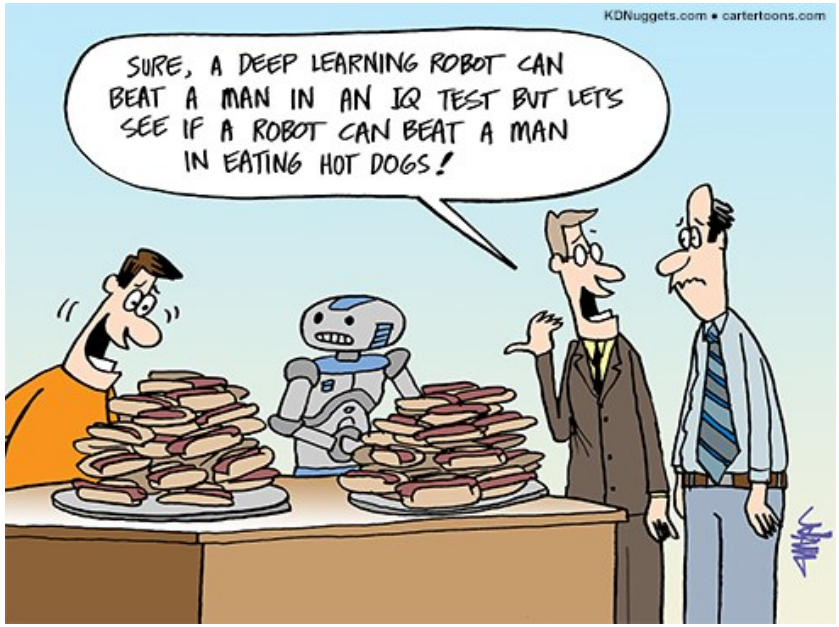
https://cloud.google.com/tpu/

# Deep learning resources

See:

http://deeplearning.net

http://www.deeplearningbook.org/




http://www.kdnuggets.com/2015/07/cartoon-humans-ahead-deep-learning.html