

Machine Learning, Deep Learning

Roadmap

1. AI, Logic, Learning
2. Evaluating performance
3. Model dimensionality and overfitting
4. Artificial Neurons (Perceptrons)
5. Multilayer Neural Networks
6. Activation functions, error functions
7. Deep Learning: ImageNet
8. Dropout
9. DeepBind
10. Implementing Deep Learning: TensorFlow (Mateo Torres)

References

1. AI, Logic, Learning

- AI aims at building intelligent systems
- Logic-based AI focuses on reasoning
- Machine learning-based AI focuses on learning

Classification of machine learning techniques

- **Supervised learning:** for each input we have values for the desired output, i.e. examples are pairs (x, t)
 - **classification problems:** assign inputs to one of a number of discrete classes. *The goal is to learn a function that can discriminate new inputs into a number of classes.*
 - **regression problems:** the outputs represent the values of continuous functions. *The goal here is to learn a model of the continuous functions that can be used to predict the output for new input points.*
- **Unsupervised learning:** goal is to model the probability distribution of the data, or discover structure (e.g. clusters)
- **Semi-supervised learning:** supervised learning which also makes use of unsupervised data
- **Reinforcement learning:** we have a value for the desired output only at the end of a sequence of actions.

2. Evaluating performance: What? How?

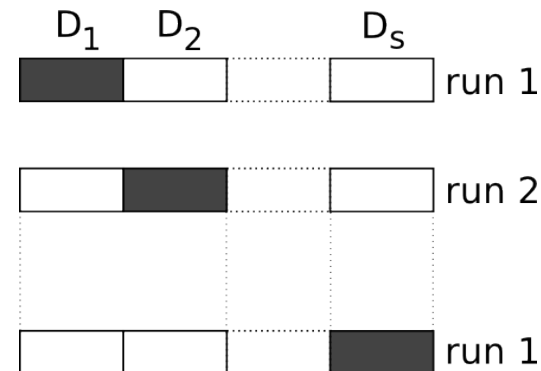
A. What do we want to evaluate?

GENERALIZATION

Therefore it is mandatory to divide your dataset:









Alternatively, use Cross Validation:



B. How do we evaluate performance?

1. Classification problems

	PREDICTED OBJECT	
		
REAL OBJECT	 TP	 FN
	 FP	 TN

Accuracy

$$\frac{TP+TN}{(TP+FP+FN+TN)}$$

Sensitivity (or TPR)

$$\frac{TP}{P} = \frac{TP}{(TP+FN)}$$

Specificity

$$\frac{TN}{N} = \frac{TN}{(TN+FP)}$$

True positive rate

$$\frac{TP}{(TP+FP)}$$

False positive rate

$$\frac{FP}{N} = \frac{FP}{(FP+TN)}$$

*ROC analysis is good for
comparing binary classifiers*

2. Regression problems

Sum of squares error

Root Mean Square error

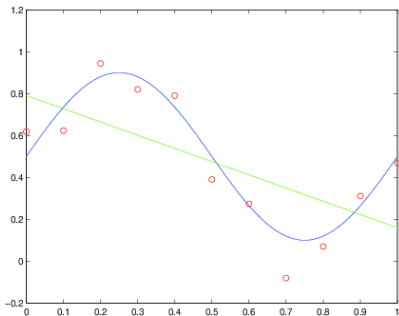
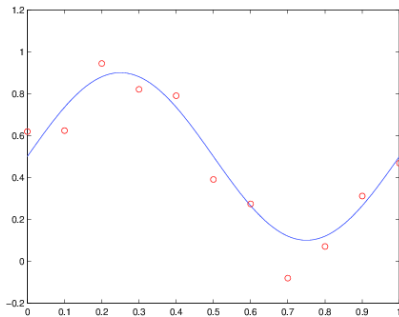
3. Model dimensionality and overfitting

We are given the red dots.

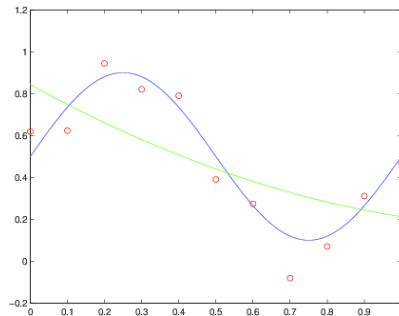
We assume that they are noisy samples from a signal/(function) – the blue curve – which we do not have (we only have the red dots).

We want to predict new points, i.e. the y coordinates for other values of x (e.g. $x > 1$)

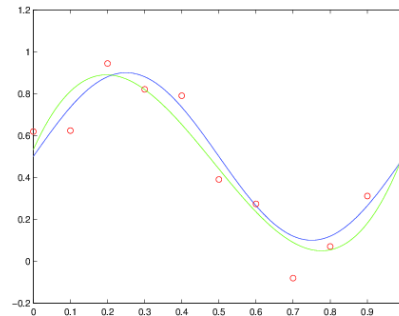
Our model needs to approximate the blue function.
We decide to do it with polynomials.



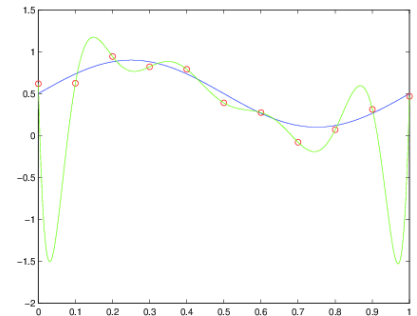
Degree 1 polynomial



Degree 2 polynomial



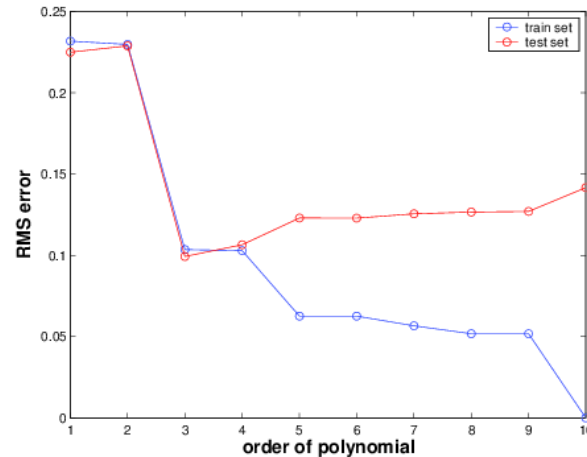
Degree 3 polynomial



Degree 10 polynomial

Which one is best? And why?

How does the GENERALIZATION performance vary, as we increase the complexity of the polynomial?



- Occam's razor (*William of Occam, ~1300*): Accept the simplest explanation that fits the data.

We should prefer simpler models to more complex models, and this preference should be traded off against the extent to which the model fits the data.

- **IMPORTANT:** increasing the number of features may lead to a reduction in performance if the number of datapoints is not increased. Why?

	Feature 1	Feature 2	...	Feature m	Target
Point 1	0.7	0.4		0.1	3.7
Point 2	0.6	0.3		0.2	4.2
⋮			...		
Point n	0.4	0.3		0.6	2.8

This is related to the “Curse of Dimensionality” Bellman, 1961.

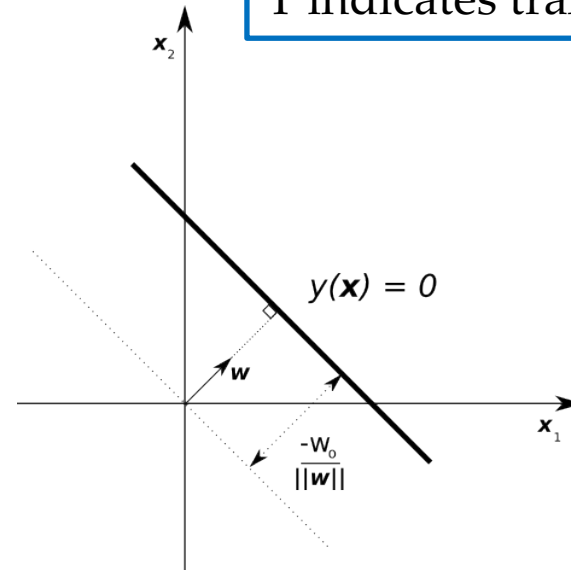
4. Artificial Neurons

NOTATION:
All vectors are columns
Bold indicates vectors
T indicates transpose

Linear function, in d dimensions:

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

w defines the orientation of the plane
 w_0 defines the position of the plane
(it is often called *bias*)



Linear boundary for $y(\mathbf{x}) = 0$, in 2 dimensional input space (x_1, x_2) .

We can define a new $(d+1)$ dimensional vector $\tilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\tilde{\mathbf{x}} = (1, \mathbf{x})$ and rewrite the linear function as:

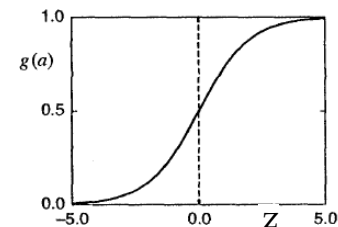
$$y(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$$

We can add a non-linearity (*activation function*):

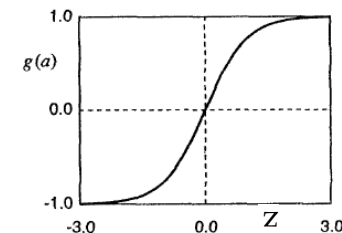
$$y(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x} + w_0)$$

where g can be, for example:

$$g(z) = \frac{1}{1+e^{-z}} \quad \text{Sigmoid or logistic function, } \sigma(a)$$



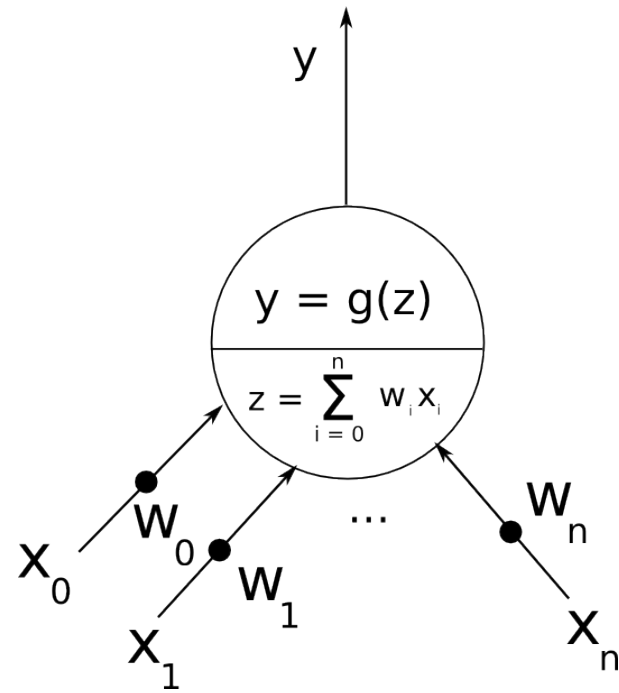
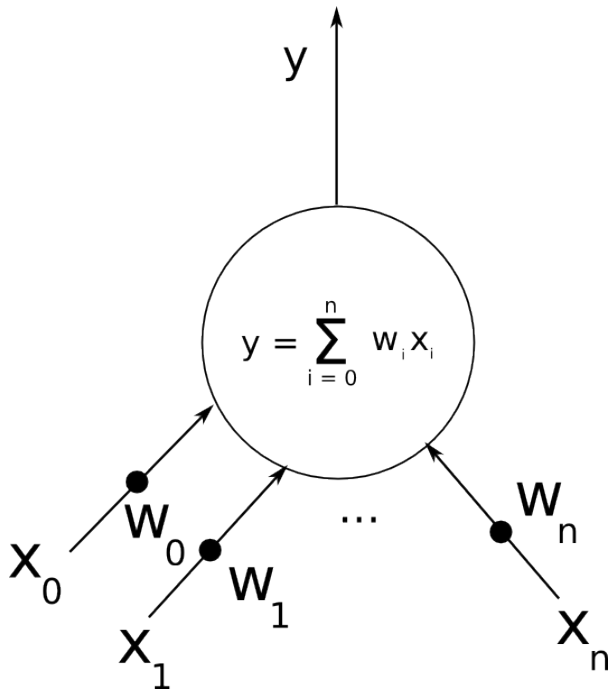
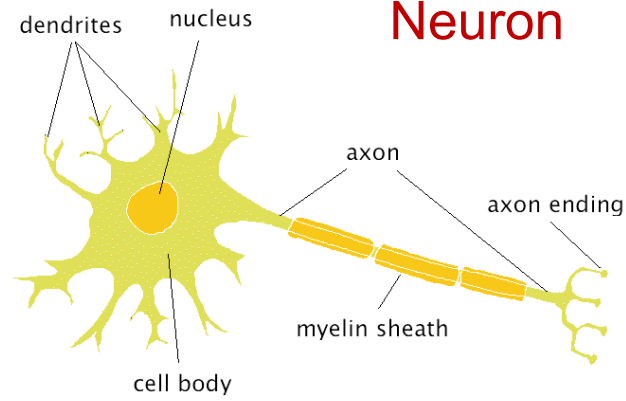
$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad \text{Hyperbolic tangent, } \tanh(a)$$



$$g(z) = \begin{cases} +1 & \text{when } z \geq 0 \\ -1 \text{ (or } 0) & \text{when } z < 0 \text{ otherwise} \end{cases} \quad \text{Step functions}$$

$$g(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{Rectified Linear function}$$

Neuron



Training artificial neurons

How can we set the values of \mathbf{w} in order to solve a classification/regression problem?

Error function: *a function that characterizes the difference between the output of our system and the correct output.*

	Feature 1	Feature 2	...	Feature m	Target
Point 1	0.7	0.4		0.1	3.7
Point 2	0.6	0.3		0.2	4.2
...			...		
Point n	0.4	0.3		0.6	2.8

For example, the sum-of-squares error function:

$$E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n)^2$$

where N is the number of training points, y_n are the output values of the system and t_n are the targets.

- If the **activation function is differentiable** (e.g. sigmoid) **we can calculate the derivatives** of the error function with respect to the weights.
- These derivatives can be used in **gradient-based optimization algorithms** for finding the minimum of the error function.

(Note that if y is a linear function of the weights, and the error function is a quadratic function of the weights, the optimal weights can be found exactly in closed form...)

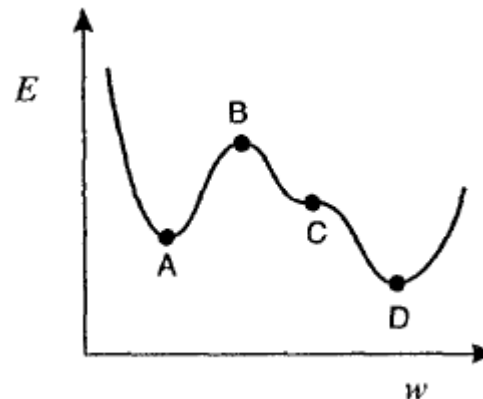
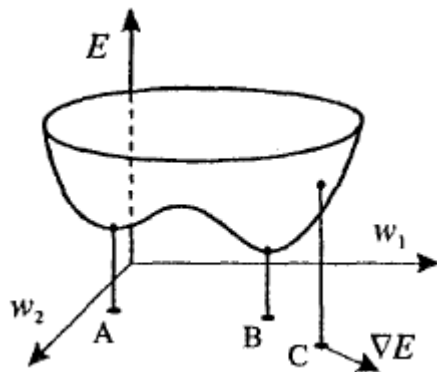
Gradient Descent

1. Begin with an initial guess for w
2. Update the weight vector by moving a small distance in the direction in which E decreases most rapidly, i.e. opposite of its gradient: $-\nabla_w E$. The update rule for the weights is:

$$w_i^{(t+1)} = w_i^{(t)} - \epsilon \frac{\partial E}{\partial w_i}$$

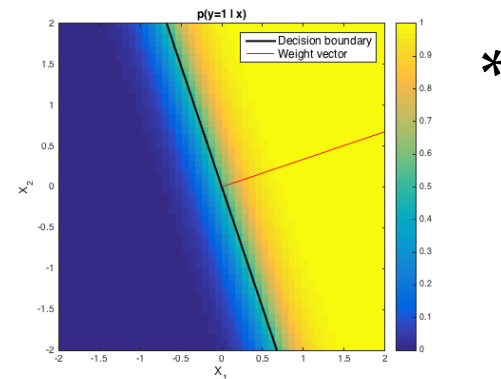
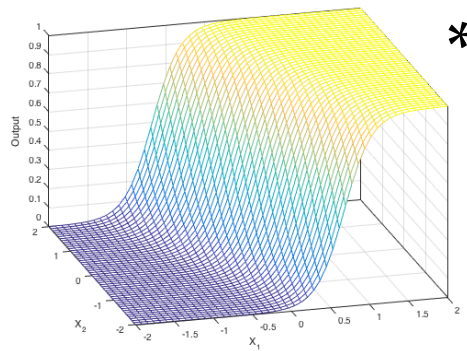
where ϵ is the *learning rate* parameter.

3. Repeat step 2 until the difference in the error between 2 successive iterations is smaller than a predefined threshold.

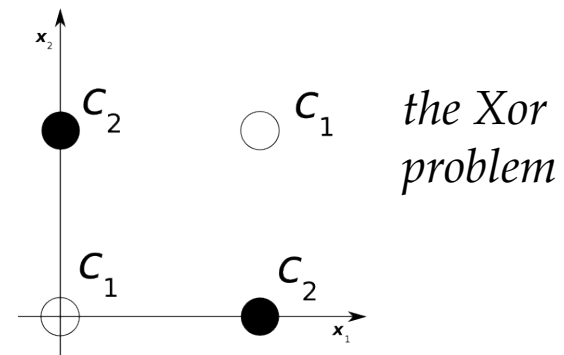


Limitations

- Since the activation functions are monotonic, the decision boundary generated by one artificial neuron is linear.



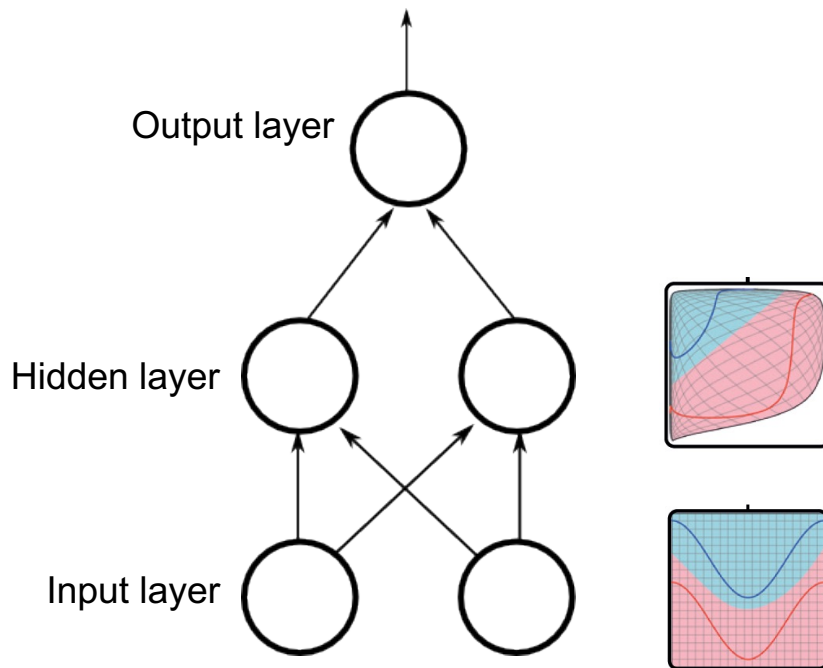
- Therefore it cannot solve some simple problems



* from <http://stats.stackexchange.com/questions/263768/can-a-perceptron-with-sigmoid-activation-function-perform-nonlinear-classificati>

5. Multilayer Neural Networks

Let's stack up many neurons. A multilayer Neural Network can make the classes of data linearly separable!



Data are on the red and blue lines. Note how a regular grid in input space is transformed by hidden units.

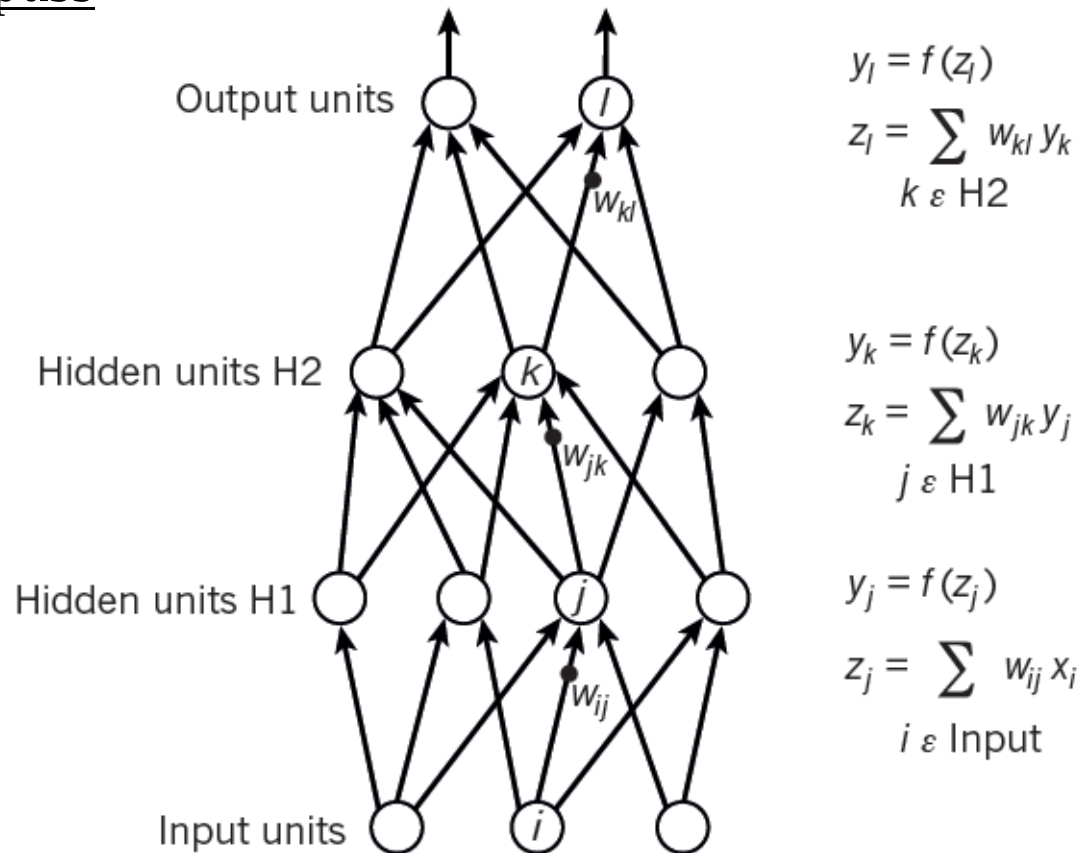
(from LeCun, Bengio, Hinton, Nature, 2015)

Important: non-linear activation functions are necessary – a multilayer network of linear units is always equivalent to a one layer network with linear units.

The Backpropagation algorithm

It calculates the derivatives of the error function with respect to the network weights. These can be used to modify the weights in order to minimize the error function.

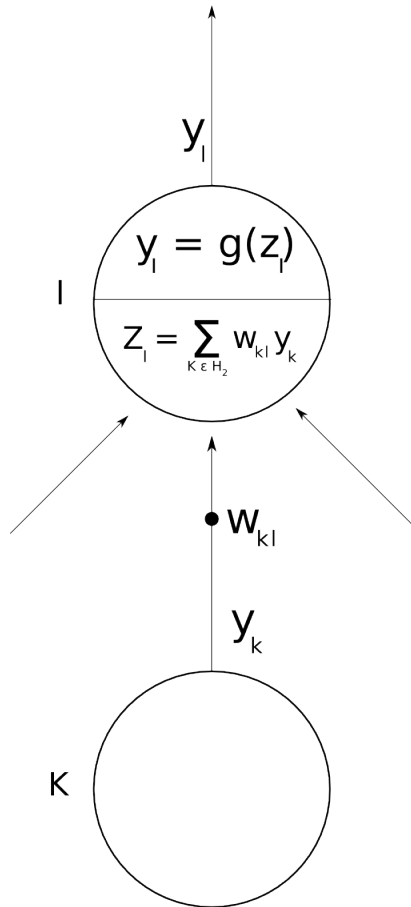
1. Forward pass



Note that bias terms are omitted.

(from LeCun, Bengio, Hinton, *Nature*, 2015) 18

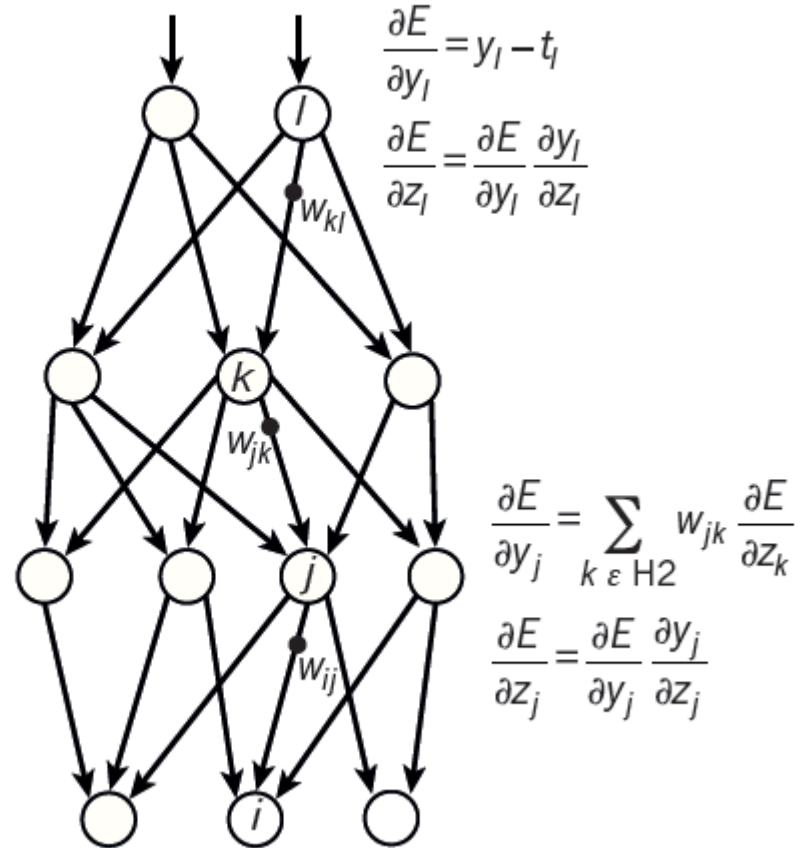
2. Backward pass



$$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

Compare outputs with correct answer to get error derivatives



(from LeCun, Bengio, Hinton, Nature, 2015)

Important result

Neural networks having 2 layers of weights and a sigmoidal activation unit can approximate arbitrarily well any functional continuous mapping, provided the number of hidden units is large enough

(this was proved by several authors towards the end of the '80s)

Nevertheless, NN with several hidden layers will turn out to be very useful...

6. Error Functions, activation functions

- Minimizing a sum-of-square error, amounts to maximizing the likelihood of the data under a Gaussian noise assumption → very good for **Regression problems**
- For **Classification problems**, the targets are binary variables, and the Gaussian noise model is not ideal.
 - The appropriate error function to use is the cross-entropy error function
 - For binary classification, the use of the sigmoid activation function allows the outputs of the network to be interpreted as posterior probabilities of class membership.
 - For multiple exclusive classes, the corresponding activation function is the *softmax*.

7. Deep Learning

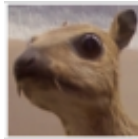
- **Representation learning:** To discover from raw data the representations needed for regression or classification.
- **Deep learning methods have multiple levels of representation.** Each level is obtained by composing simple but non-linear modules that each transform the representation at one level into one at a higher, more abstract level.
- Typically, **very large** systems:
 - ❑ hundreds of hundreds of thousands of artificial neurons
 - ❑ millions of adjustable weights
 - ❑ hundreds of millions of labelled examples with which to train the machine.
- Learning made possible by the advent of fast graphics processing units (**GPUs**)
- Software libraries have been developed for the creation of these systems (e.g. **TensorFlow** the Google Brain Team within Google's Machine Intelligence)

Deep convolutional networks (ConvNets)

- Designed to extract features from the data
- Four key ideas:
 1. local connections
 2. shared weights
 3. pooling
 4. use of many layers
- Units in a convolutional layer are organized in feature maps.
- **Feature maps**: each unit is connected to local patches in the feature maps of the previous layer through a set of weights. All units in a feature map share the same weights. Different feature maps in a layer use different weights.
- **Pooling units** compute the maximum of a local patch of units in one (or in a few) feature maps.
- **IMPORTANT**: ConvNets have fewer connections, hence they are easier to train... (less overfitting)

- Convolving an image with a numerical matrix we obtain variations of the image:

original
image



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

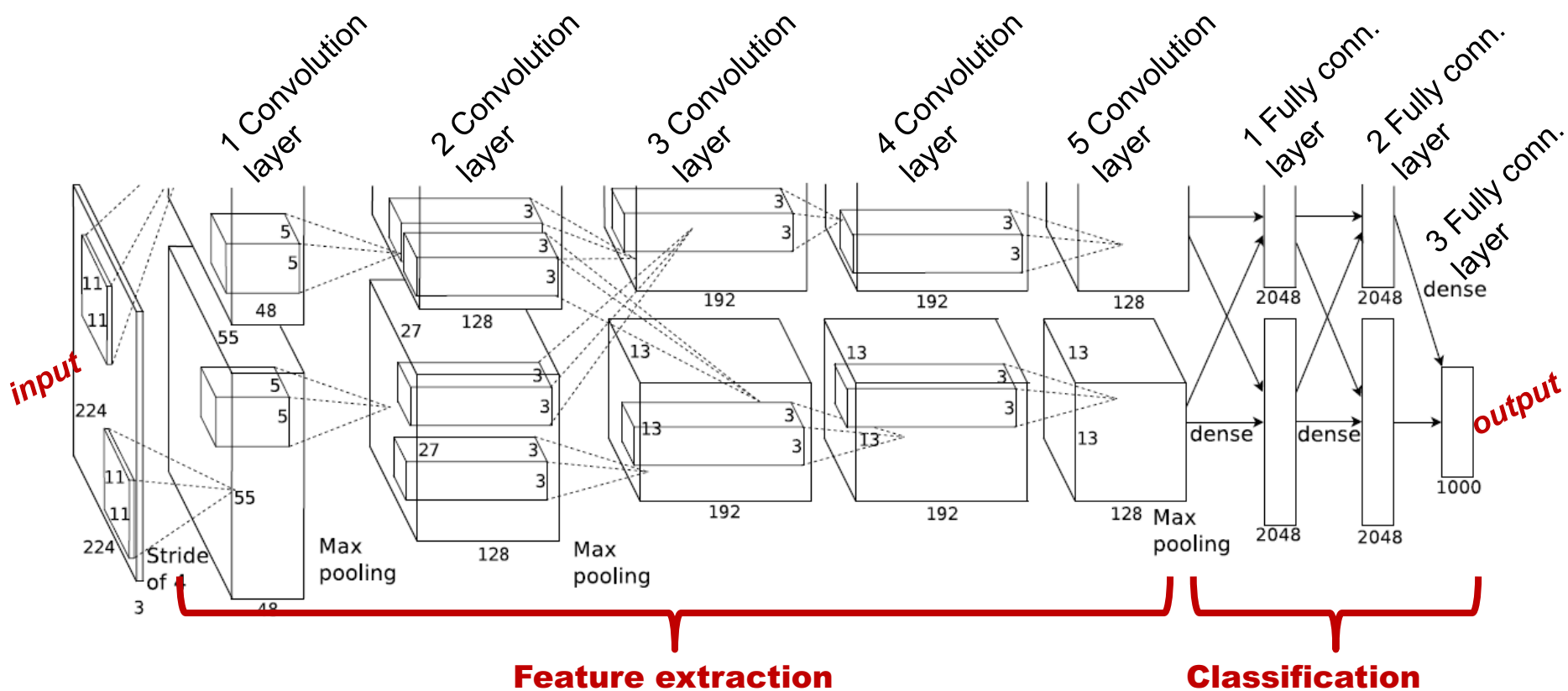
Example: two different filters (with red/green outline) slides over the input image (convolution operation) to produce two different **feature maps**.



ImageNet

[Krizhevsky, Sutskever, Hinton, NIPS, 2012]

- Classification problem: classify object in image -- 1000 different classes.
- Best results ever: Halved the error rate of the best competitor.
- Training with **1 million high-resolution images**; validation 50,000 images; testing 100,000 images
- NN with **60 million parameters and 650,000 neurons**
- 5 convolutional layers, some followed by max-pooling layers
- 3 fully-connected layers with a final 1000-way softmax.
- Very efficient **GPU implementation** of the convolution operation.
- *Architecture inspired by LGN-V1-V2-V4-IT hierarchy in the visual cortex.*



(from Krizhevsky, Sutskever, Hinton, NIPS, 2012)

- ReLU non-linear units
- 2 GPUs (top and bottom parts) that communicate only in certain layers
- Local response normalization (implements a form of lateral inhibition) – 1st and 2nd layer only
- Overlapping max pooling -- 1st, 2nd and 5th layer only
- Output layer: 1000 units, softmax activation, cross entropy error function
- Dropout to reduce overfitting (1st and 2nd fully connected layers).

8. Dropout

Technique which sets to 0 the output of each hidden neuron with probability 0.5. “Dropped out” neurons do not contribute to the forward pass or backpropagation.

- It reduces complex co-adaptations of neurons → each neuron is forced to learn more robust features
- It is equivalent to sampling a different architecture every time an input is presented. But all these architectures share weights
- We know that combining predictions of many different models is very good at reducing test error (Netflix price challenge)

9. DeepBind

*[Alipanahi et al, Nat Biotech. 2015
Zeng et al, ISMB, 2016]*

Modeling of DNA sequence protein binding specificity.

1. **Motif discovery:** classify sequences that are bound by a transcription factor from negative sequences that are dinucleotide shuffles of the positively bound sequences.
2. **Motif occupancy:** discriminate genomic motif instances that are bound by a transcription factor (positive set) from motif instances that are not bound by the same transcription factor (negative set) in the same cell type, where GC-content and motif strength are matched between the positive and negative set.

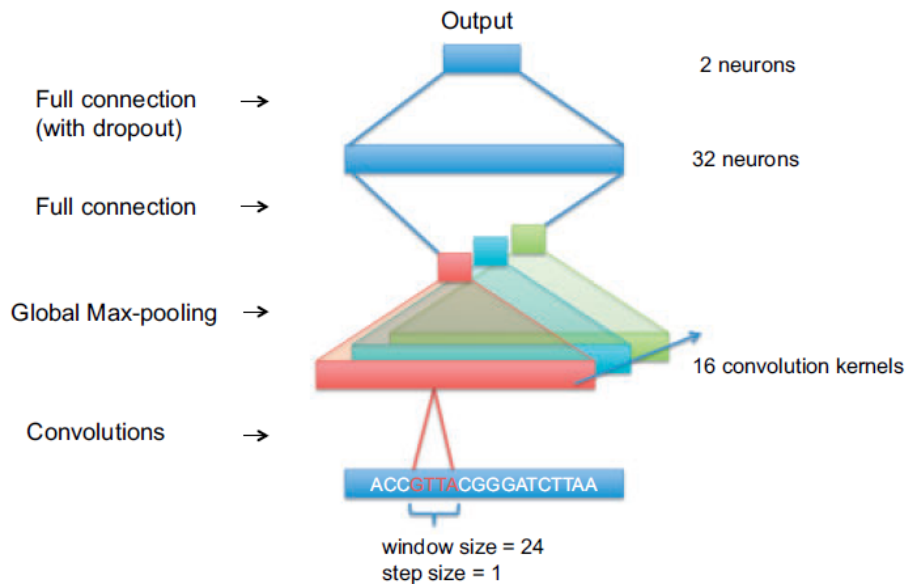


Table 1. The code and brief description of the 9 variants of CNN models compared in this work

Code	Architecture (relative to 1layer structure)
1layer	The basic structure as depicted in Figure 1
1layer_1motif	Use 1 convolutional kernels
1layer_64motif	Use 64 convolutional kernels
1layer_128motif	Use 128 convolutional kernels
1layer_local_win9	Use local maxpooling of window size 9 at top
1layer_local_win3	Use local maxpooling of window size 3 at top
2layer	2 layers with 16/32 kernels
3layer	3 layers with 16/32/64 kernels
2layer_local_win3	2 layers with 16/32 kernels, use local maxpooling of window size 3 at top
3layer_local_win3	3 layers with 16/32/64 kernels, use local maxpooling of window size 3 at top

(from Zeng et al, ISMB, 2016)

- **1st layer: convolutional layer**, which can be thought of as a motif scanner.
- **2nd layer: global max-pooling layer**. Its role is to call whether the motif modelled by the respective convolutional layer exists in the input sequence or not.
- **3rd and 4th layer: fully connected**
- Many variations were analysed later

References

- Books

- Christopher M. Bishop, *Pattern Recognition and Machine Learning*, 2007
- Trevor Hastie, Rob Tibshirani, Jerome Friedman *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Second Edition, 2009
- Ian Goodfellow, Yoshua Bengio, Aaron Courville *Deep Learning*, 2016

- Papers

- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*, in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*
- LeCun, Y., Bengio, Y. and Hinton, G. E. (2015), *Deep Learning* Nature, Vol. 521, pp 436-444
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) *Dropout: A simple way to prevent neural networks from overfitting* The Journal of Machine Learning Research, 15(1), pp 1929-1958.
- Babak Alipanahi, Andrew DeLong, Matthew Weirauch, Brendan J Frey *Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning*, Nature Biotechnology 33, 831–838 (2015)
- Haoyang Zeng, Matthew Edwards, Ge Liu, David Gifford *Convolutional neural network architectures for predicting DNA–protein binding* Bioinformatics (2016) 32 (12): i121-i127.