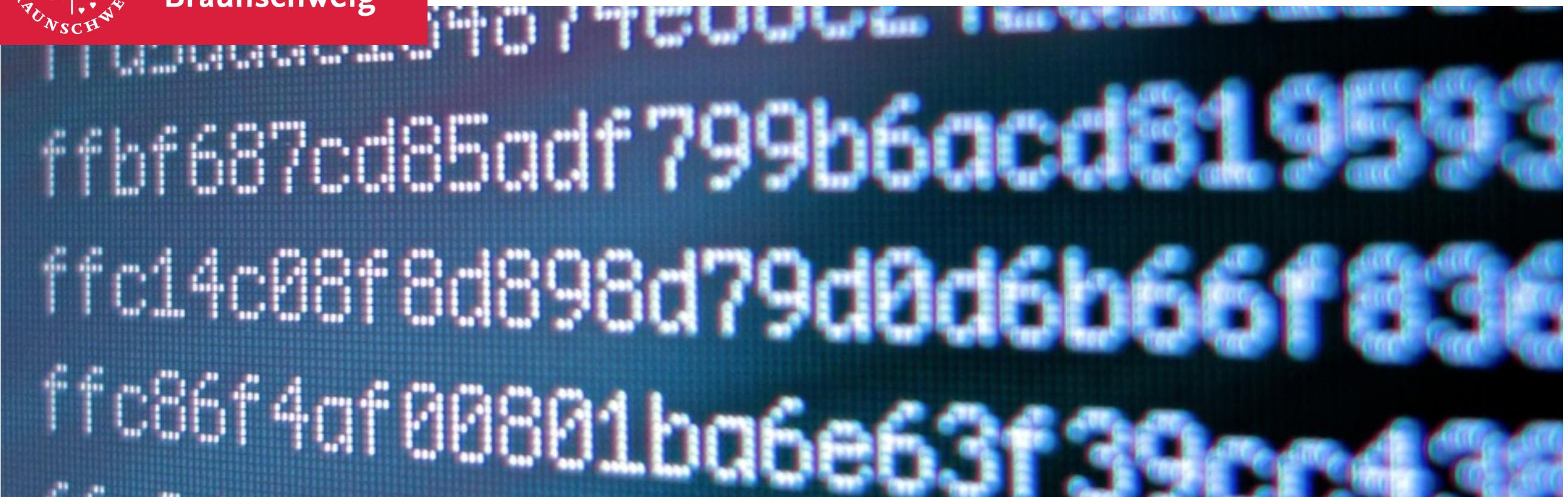




Technische
Universität
Braunschweig

Institute of
System Security



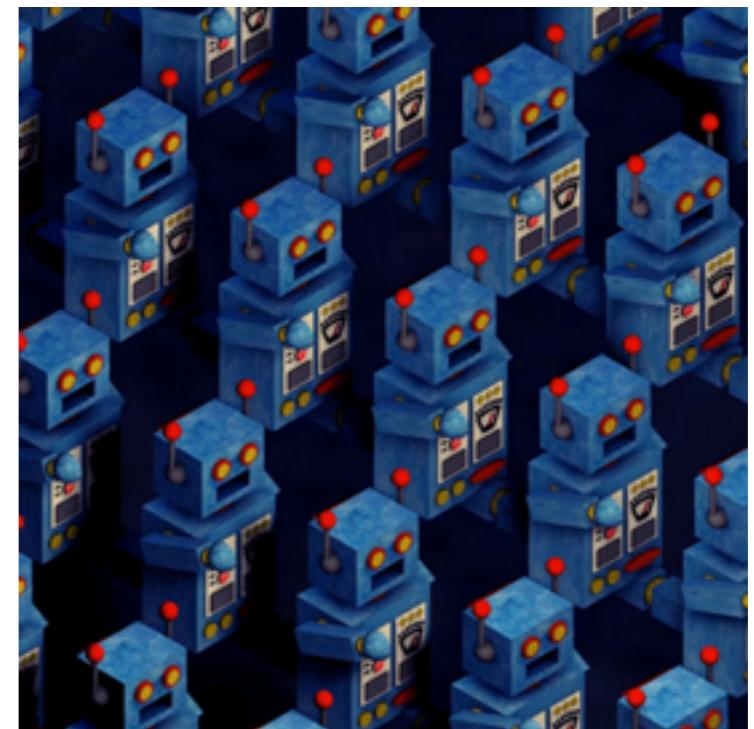
Deep Learning for Malware Analysis

Machine Learning for Computer Security

Hugo Gascón

Overview

- **What you will learn today**
 - A primer on artificial neural networks
 - Modern deep network architectures
 - Some applications to computer security

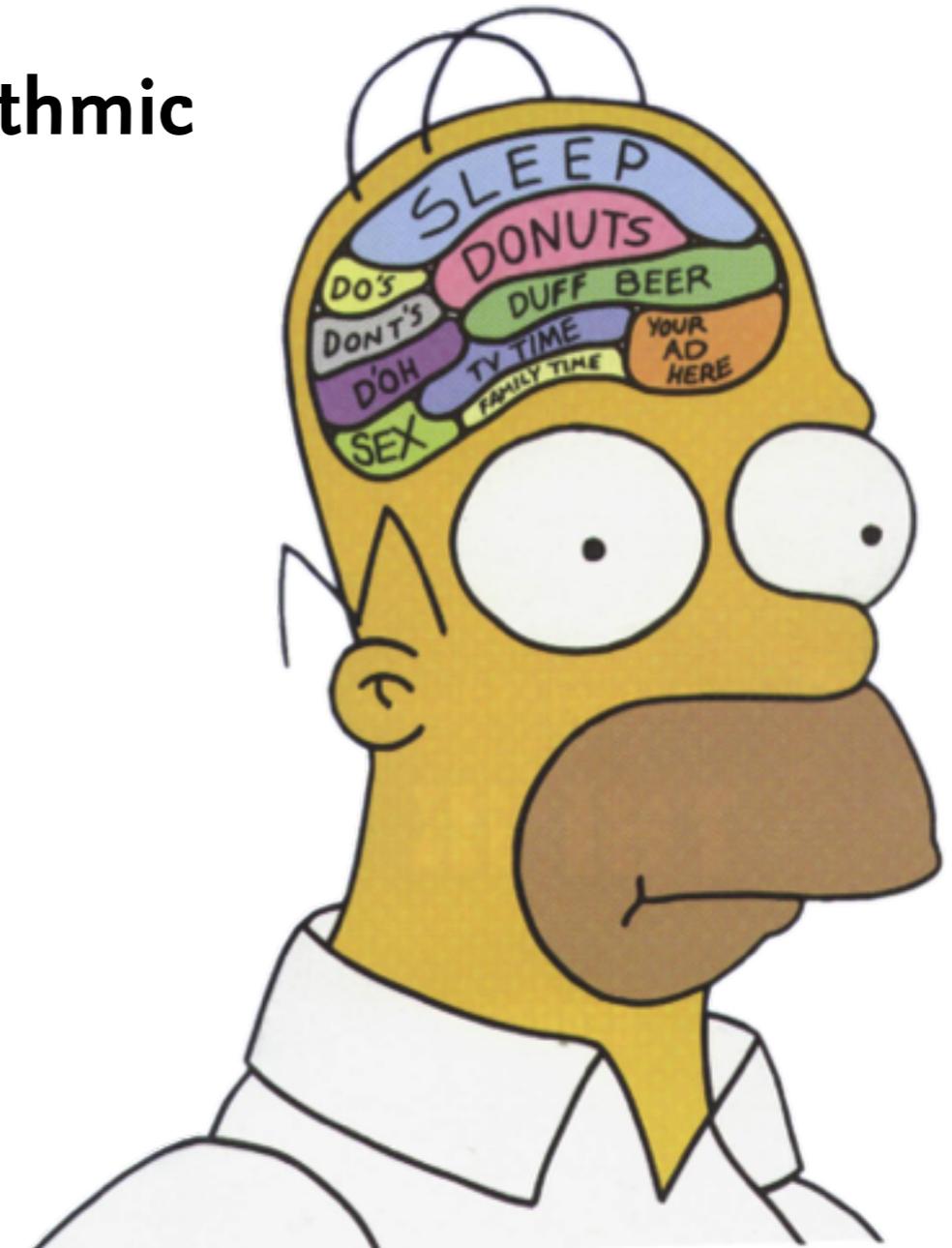


A Primer on Neural Networks



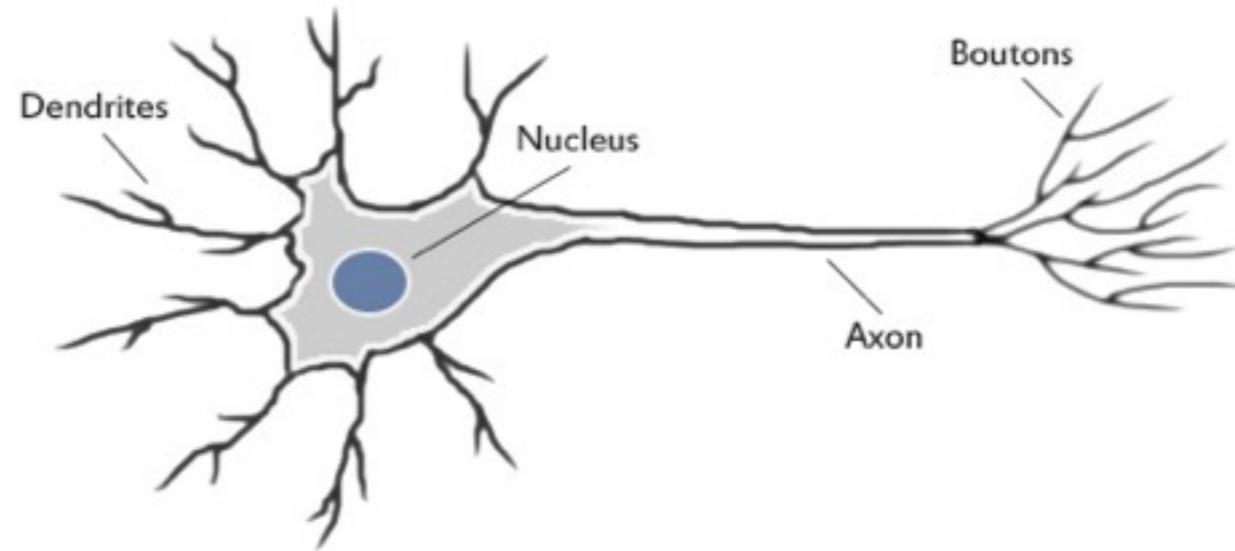
What is an Artificial Neural Network?

- Computers are great at solving algorithmic and math problems
- Real world task difficult to define mathematically, e.g.:
 - Recognising faces
 - Understanding language
- Artificial neural networks model information like the brain does



How does the brain work?

- Extremely large interconnected networks of $\sim 10^{11}$ neurones
- Output a weighted sum of inputs when triggered

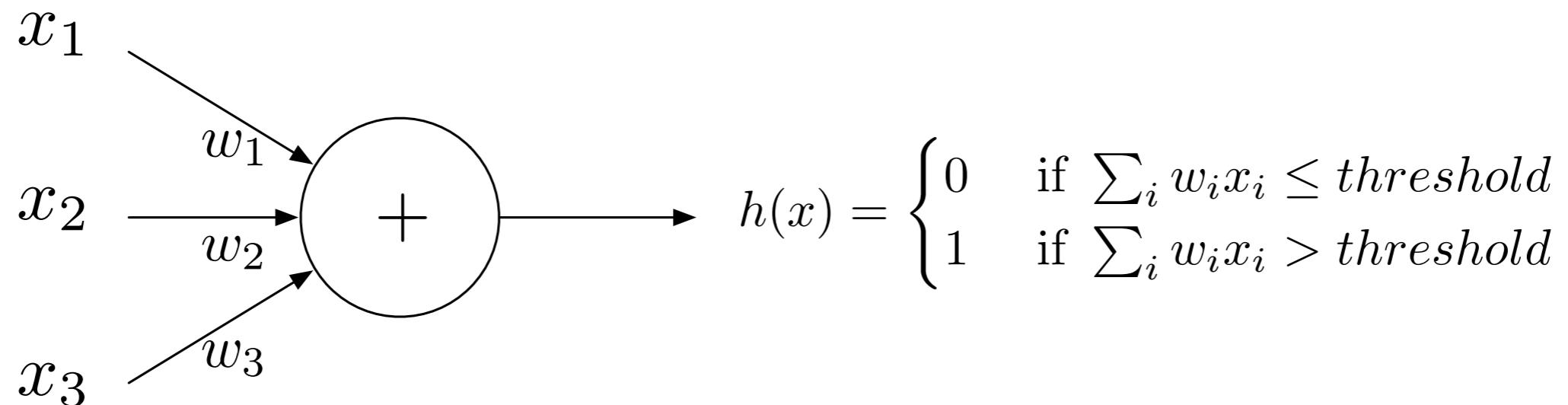


- The artificial model of a neurone is called **perceptron**



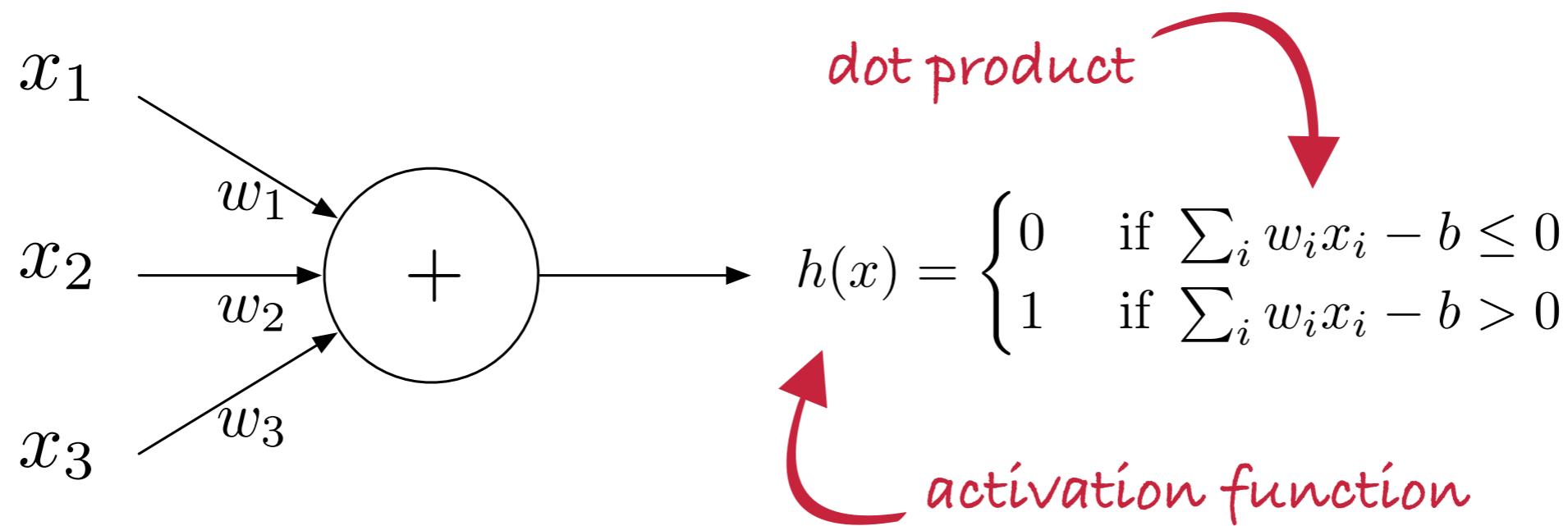
Perceptron

- Developed in the 50s and 60s by the Frank Rosenblatt
- Linear classifier on top of a simple feature extractor



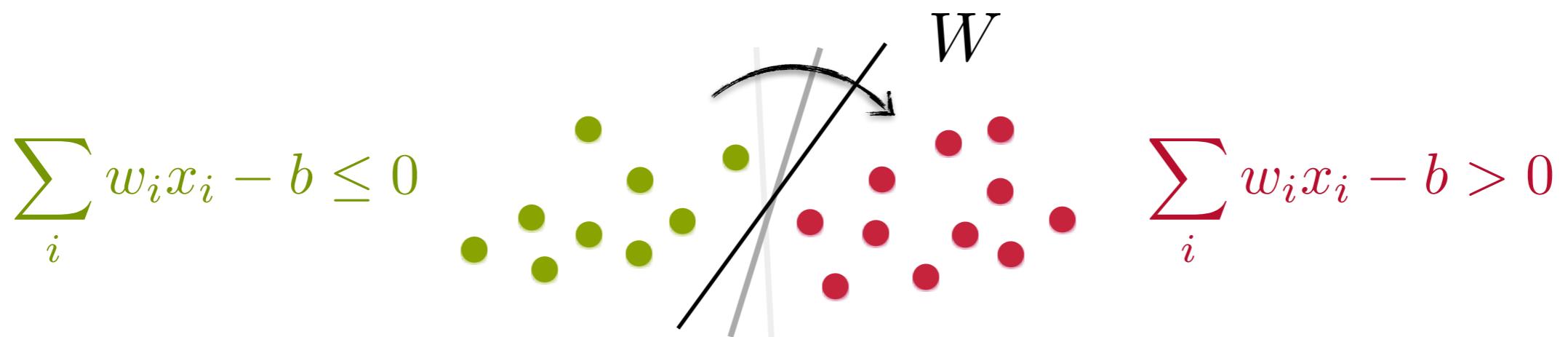
Perceptron

- Developed in the 50s and 60s by the Frank Rosenblatt
- Linear classifier on top of a simple feature extractor



Perceptron as Classifier

- The perceptron is the first learning machine
 - Classic supervised learning algorithm for classification
- Learning model
 - Weight vector $W = \{w_1, w_2, w_3\}$ defines a hyperplane



Perceptron Rule

- Learning by iterative updates of weight vector W
 - Pick x_i from training data and compute the output
 - Update weights following rule

$$w_i^{m+1} = w_i^m + \alpha(t_j - y_j^m)x_{j,i}$$

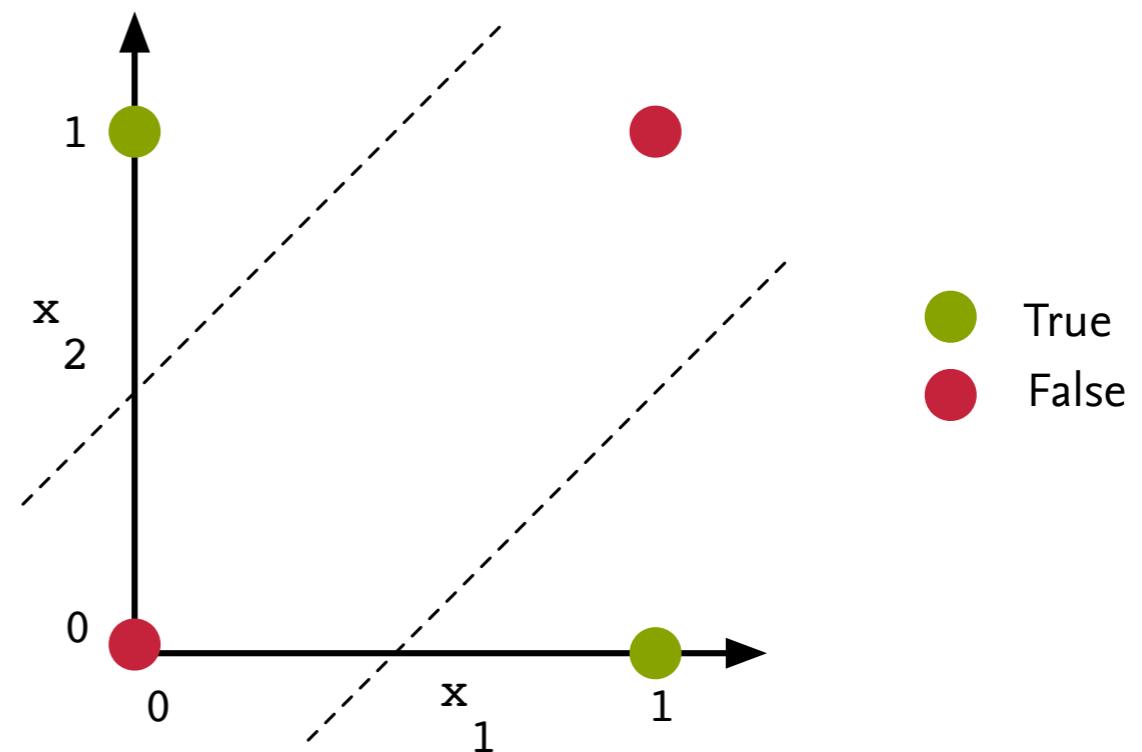
$$0 \leq i \leq n$$

- Algorithm only converges if data is **linearly separable**

Perceptron Limitations

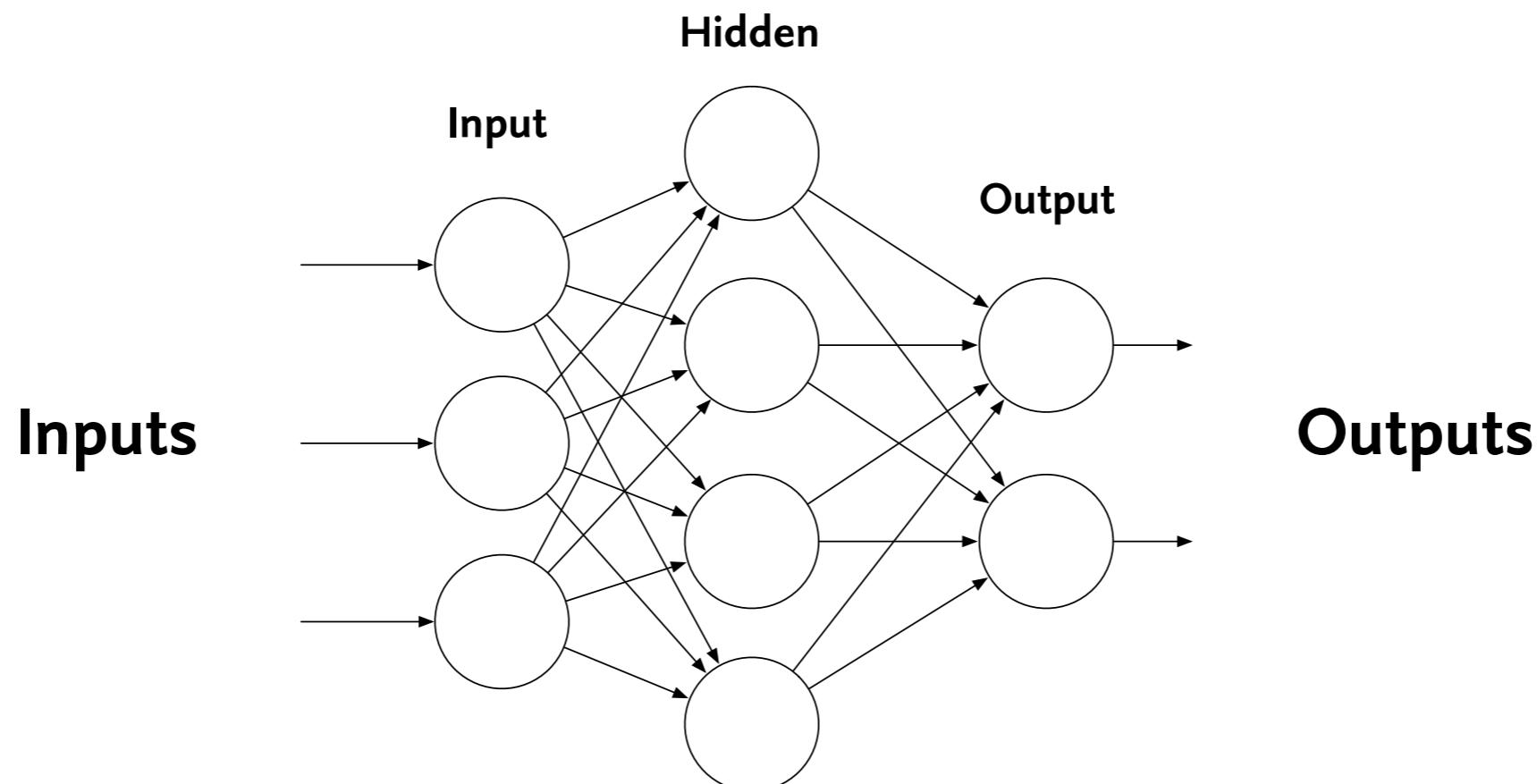
- Simplest problem that can not be solved by a perceptron

XOR	$x_1 = 0$	$x_1 = 1$
$x_2 = 0$	0	0
$x_2 = 1$	1	0



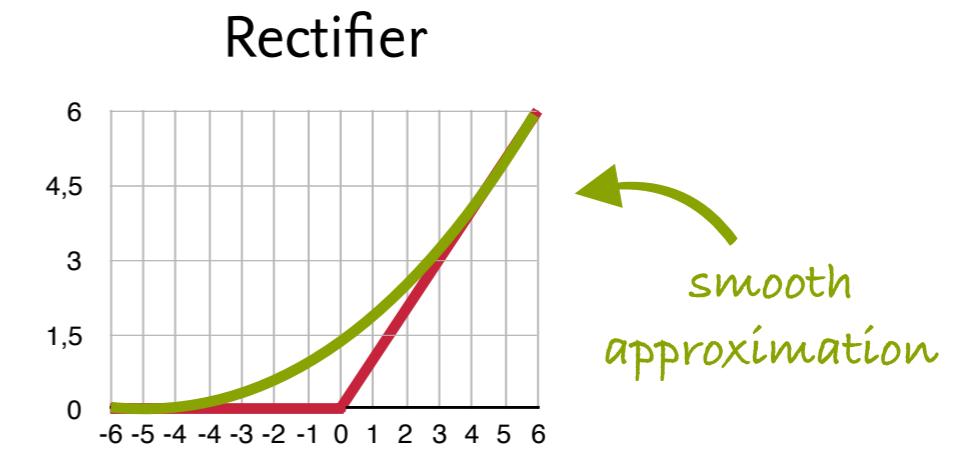
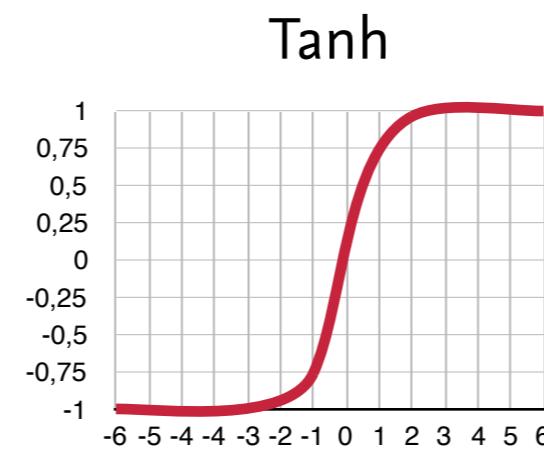
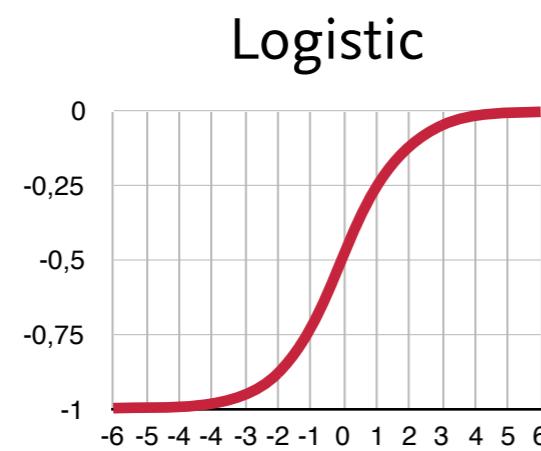
Feedforward Neural Network

- **Perceptron basic building block of neural networks**
 - Each layer weights the decisions from the previous layer
 - Complex and abstract decision making at higher layers



Beyond Linearity

- A linear composition of several linear functions is still linear
- Alternative are **non-linear activation functions or rectifiers**



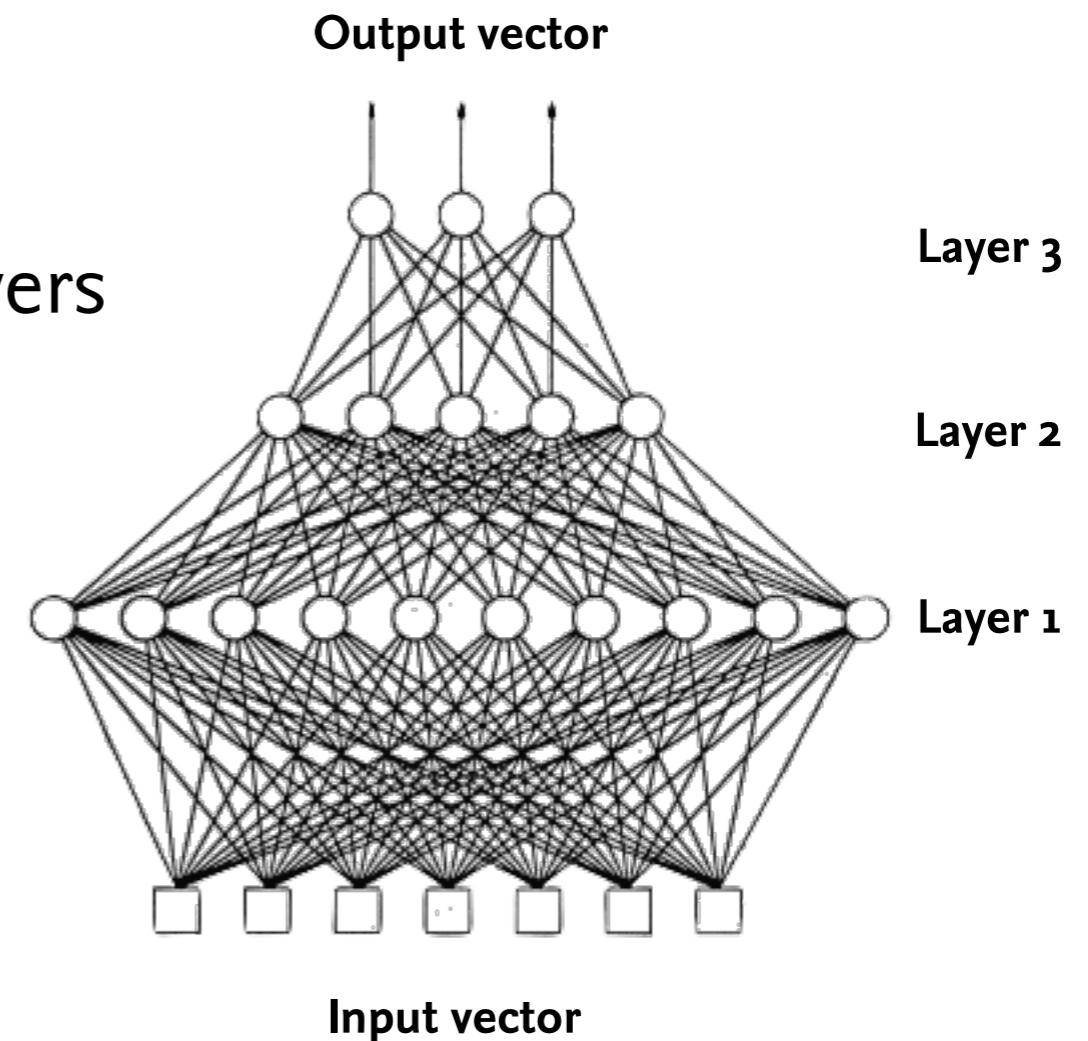
$$h(x) = \frac{1}{1 + e^{-x}}$$

$$h(x) = \tanh(x)$$

$$h(x) = \max(0, x)$$
$$h(x) = \ln(1 + e^x)$$

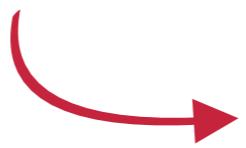
Multilayer Perceptron

- Feedforward neural network with sigmoid neurons
 - Three or more fully connected layers
 - Can distinguish data that are not linearly separable
- Supervised learning with backpropagation



Backpropagation

- Find internal representations to allow an arbitrary mapping
- Basic strategy
 - Backward propagation of errors
 - Gradient of **loss function** respect to all weights
 - Update of weights
- Requires
 - Known desired output value per input
 - **Differentiable** activation functions



e.g. mean square error $E = \frac{1}{2}(t - y)^2$



Backpropagation Phases

1. Propagation

- Forward propagation of a training pattern's input
- Backward propagation of outputs to compute deltas

2. For each weight

- Multiply output delta and input activation to get gradient
- Subtract a percentage of the gradient from the weight

Minimize error with **stochastic gradient descent**

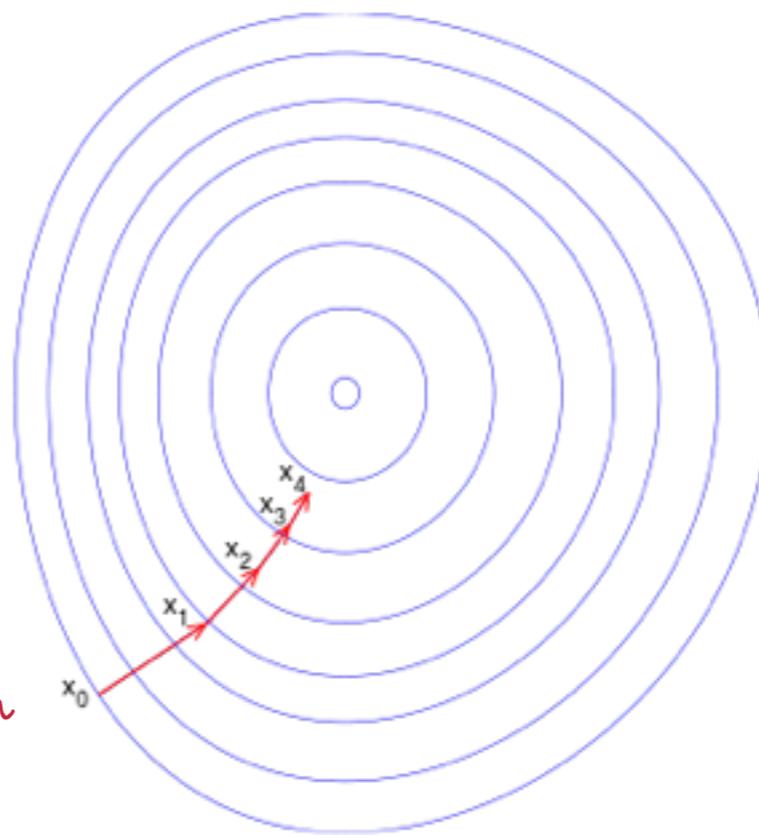


Gradient Descent

- **Most common optimization algorithm for neural networks**
 - Iterative update of parameters to minimize E
 - All training samples required for a single weight update

$$w = w - \eta \sum_i^n \frac{\partial E(w)}{\partial w_i}$$

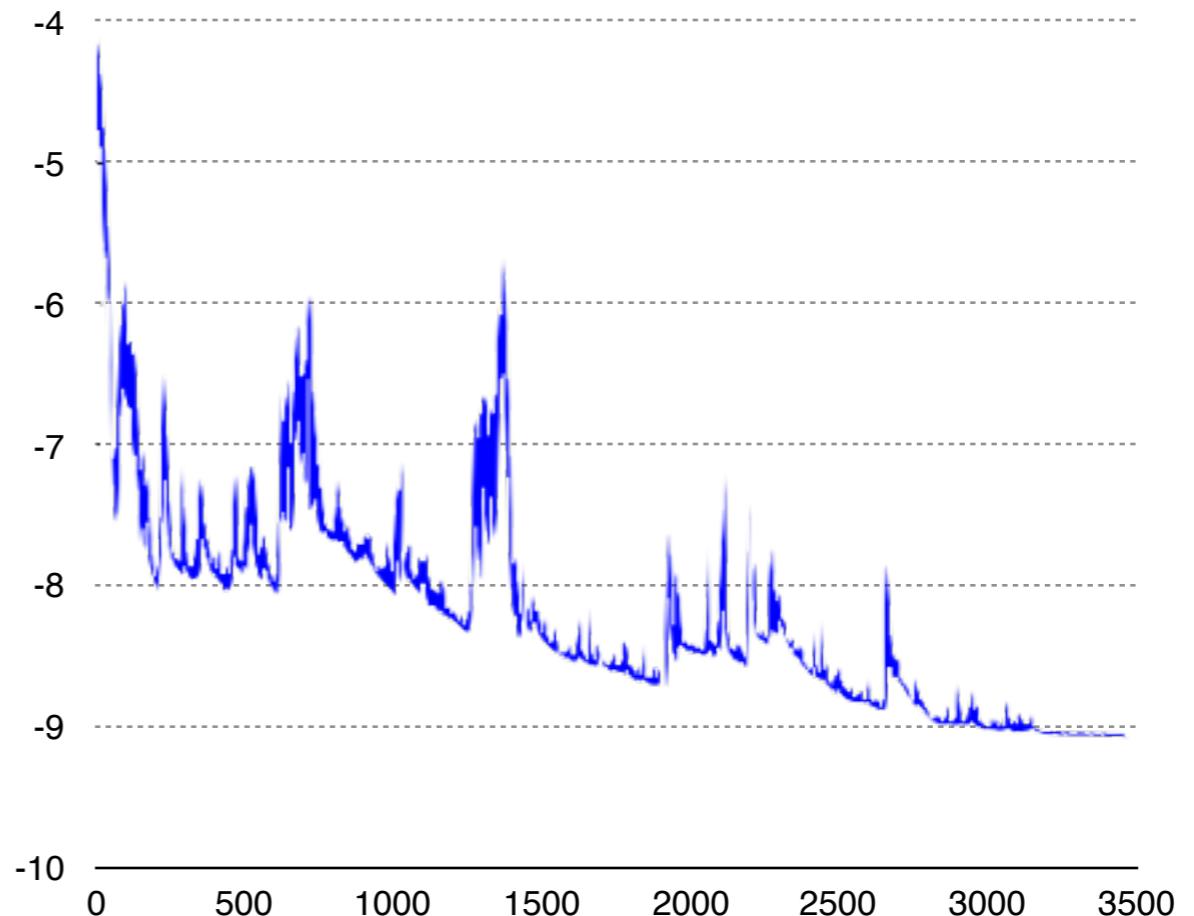
2-dimensional function



Stochastic Gradient Descent

- **Improvement over gradient descent**
 - One training sample and label per weight update
 - Much faster convergence

$$w = w - \eta \frac{\partial E(w)}{\partial w_i}$$

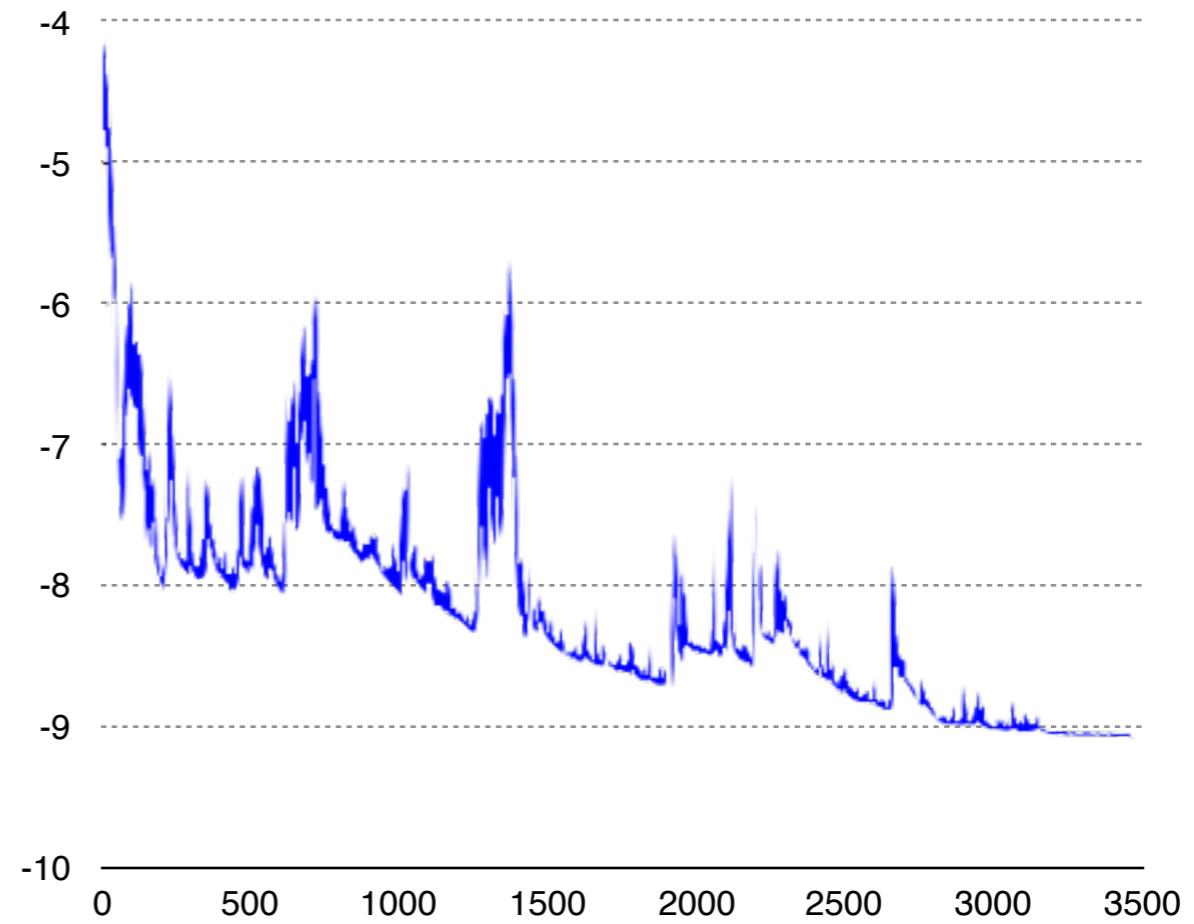


Stochastic Gradient Descent

- **Improvement over gradient descent**
 - One training sample and label per weight update
 - Much faster convergence

Thinking in code...

```
define w, rate
while E > e:
    shuffle training set
    for i in N:
        w = w - rate*dEi(w)
```



Stochastic Gradient Descent Example

- Fit $y = w_1 + w_2x$ to a set of points $(x_1, y_1), \dots, (x_n, y_n)$
- Error function using least squares

$$E(w) = \sum_{i=1}^n E_i(w) = \sum_{i=1}^n (w_1 + w_2x_i - y_i)^2$$

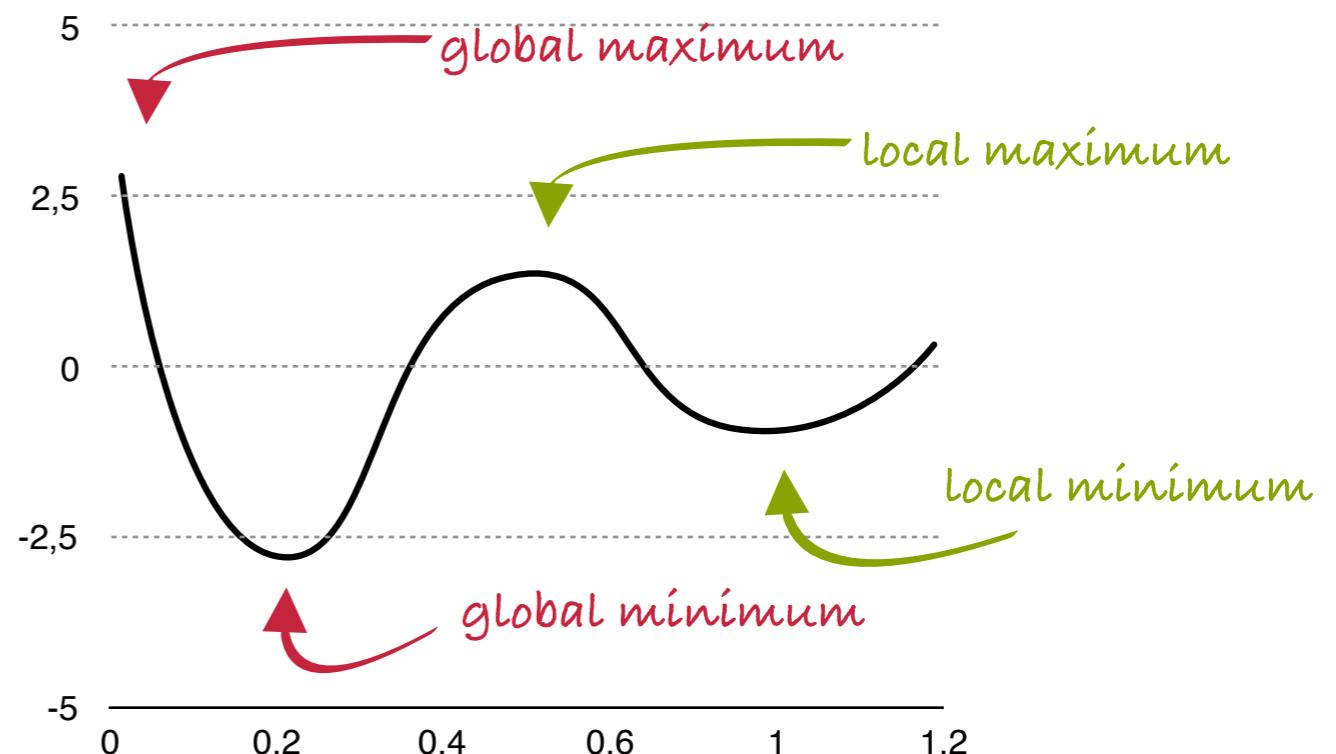
- Adjusted Weights

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} 2(w_1 + w_2x_i - y_i) \\ 2x_i(w_1 + w_2x_i - y_i) \end{bmatrix}$$



Backpropagation Limitations

- Slow for large networks
- Learning rate trade-off between **speed** and **stability**
- Not guaranteed to find global minimum of error function



Basic Network Architectures



Autoencoder

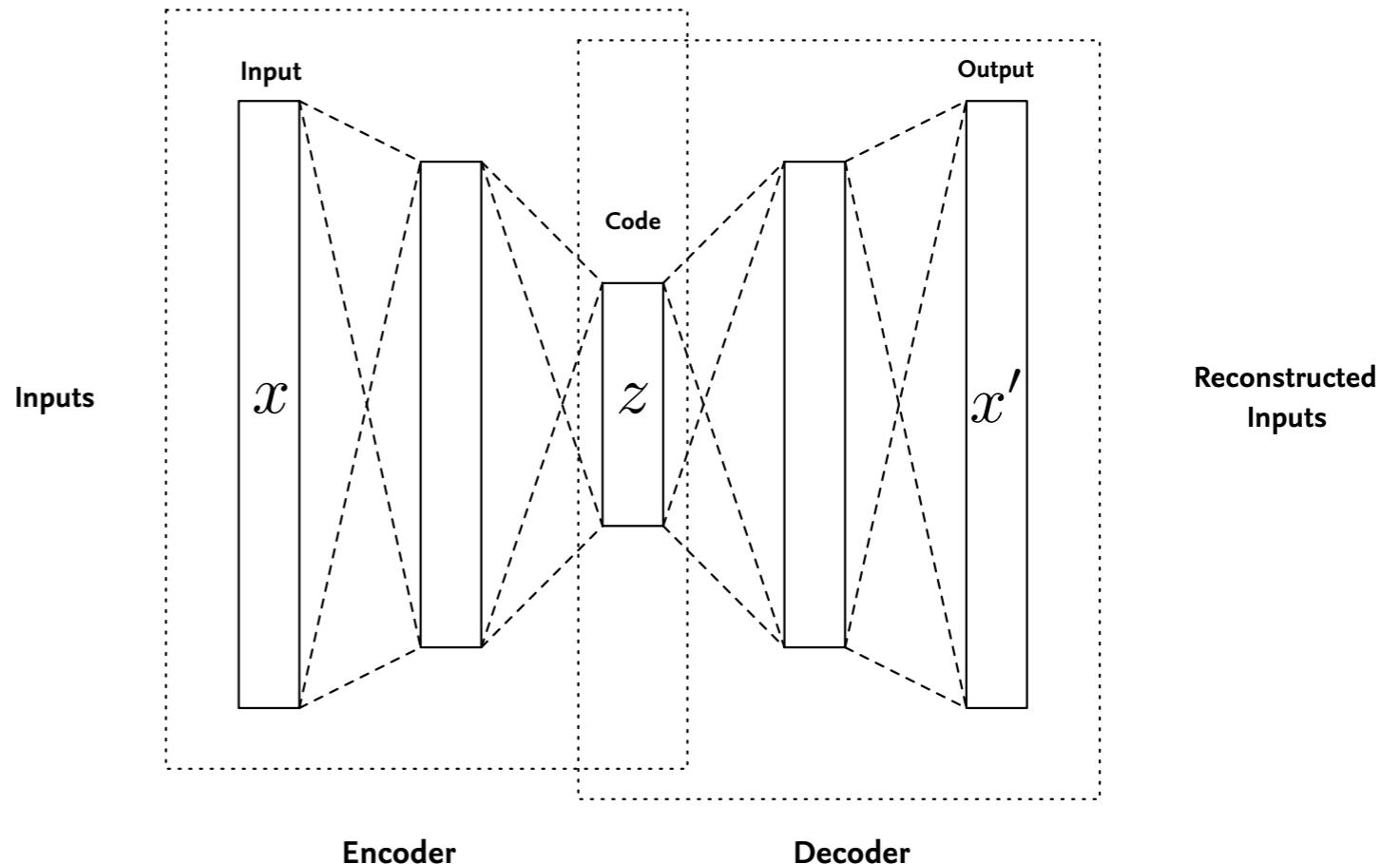
- **Feedforward non-recurrent neural network**
 - Input, output and one or more hidden layers
 - Input and output layers have **same size**
- **Learn a representation (encoding) of the data**
 - Dimensionality reduction
 - Learning generative models
- **Autoencoders build unsupervised learning models**

Autoencoder Architecture

- **2 Parts**
 - Encoder $\phi : \chi \rightarrow \mathcal{F}$
 - Decoder $\psi : \mathcal{F} \rightarrow \chi$
- **Map** $x \in \mathbb{R}^d$ **onto** $z \in \mathbb{R}^p$ **and** z **onto** x'

$$z = \sigma_1(Wx + b)$$

$$z' = \sigma_2(W'z + b')$$



- **Minimise reconstruction errors**

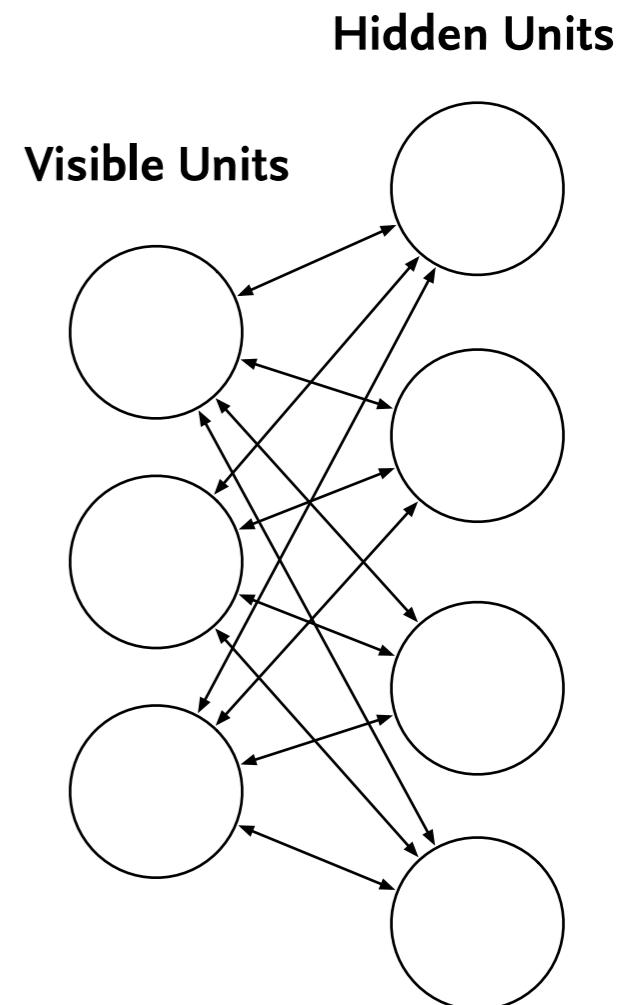
$$\mathcal{L}(x, x') = \|x - x'\|^2 = \|x - \sigma_2(W'(\sigma_1(Wx + b)) + b')\|^2$$

Autoencoder Variations

- **Denoising Autoencoders**
 - Goal is to obtain a good representation
 - Partially corrupted input → train to recover the undistorted input
- **Sparse Autoencoders**
 - Learn useful structures in input data with larger hidden layers
 - Impose sparsity hidden units → learn sparse input representations

Restricted Boltzmann Machine

- Learn a probability distribution over inputs
- Composed of visible and hidden layers
 - Undirected connections
 - Fully connected
- Binary unit activation with Bernoulli distribution
- Trained through **contrastive divergence**



Contrastive Divergence

1. Positive phase

- Input sample x is clamped to the input layer
- Propagate x to hidden layer resulting in activations h

2. Negative phase

- Propagate h back to visible layer with result x'
- Propagate x' to hidden layer resulting in activations h'

3. Weight Update

$$w_{t+1} = w_t + a(xh^T - x'h'^T)$$



Deep Network Architectures



Shallow vs. Deep Architectures

- Shallow networks have only **one** hidden layer
- **Multiple** hidden layers required to learn complex functions
- Drawbacks
 - Overfitting
 - Fit training data too well but perform poorly on test data
 - Vanishing gradients
 - Gradients become small in relation to weights
 - Hard for backpropagation to pass info to lower layers



Recent Advancements

- **To prevent overfitting → Dropout**
 - Randomly drop units during training
 - Prevents units from co-adapting too much
- **To prevent vanishing gradients → Greedy layer-wise training**
 - Trains parameters of each layer individually
 - Freeze parameters for the remainder of the model
 - Fine-tuning using backpropagation



Deep Networks

- Autoencoders and RBMs are effective feature detectors
- They can be **stacked** to form **deep networks**
 - Greedily training to avoid vanishing gradient and overfitting
- Result in powerful learning structures
 - e.g.: Google deep autoencoders learnt...



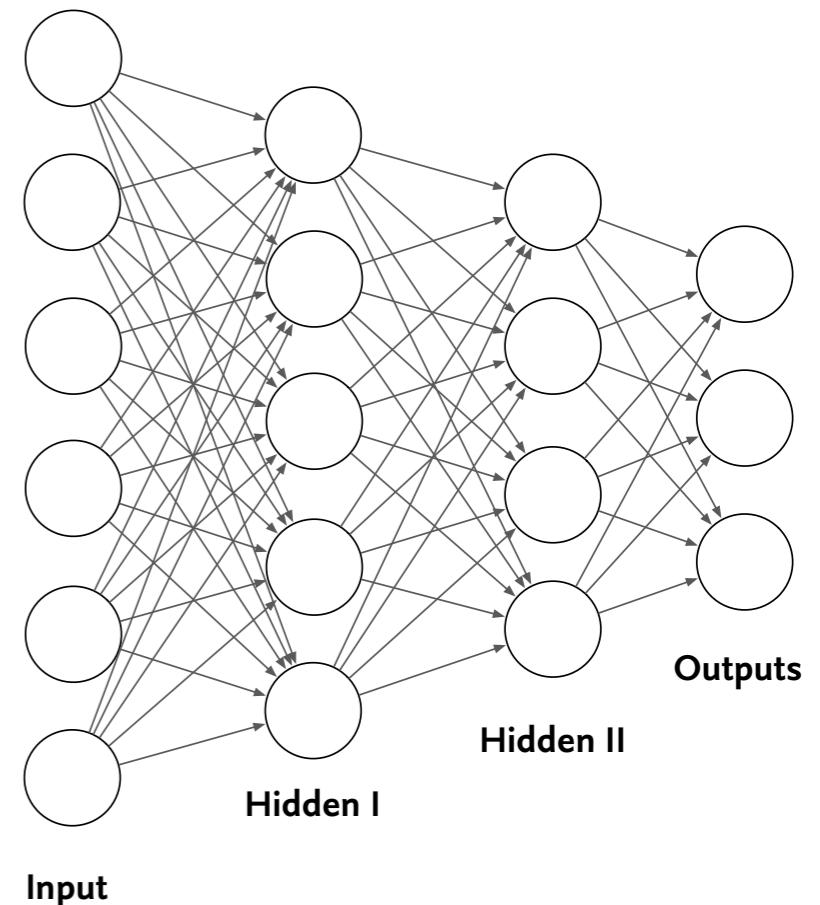
Deep Networks

- Autoencoders and RBMs are effective feature detectors
- They can be **stacked** to form **deep networks**
 - Greedily training to avoid vanishing gradient and overfitting
- Result in powerful learning structures
 - e.g.: Google deep autoencoders learnt...
...high-level features from unlabeled data



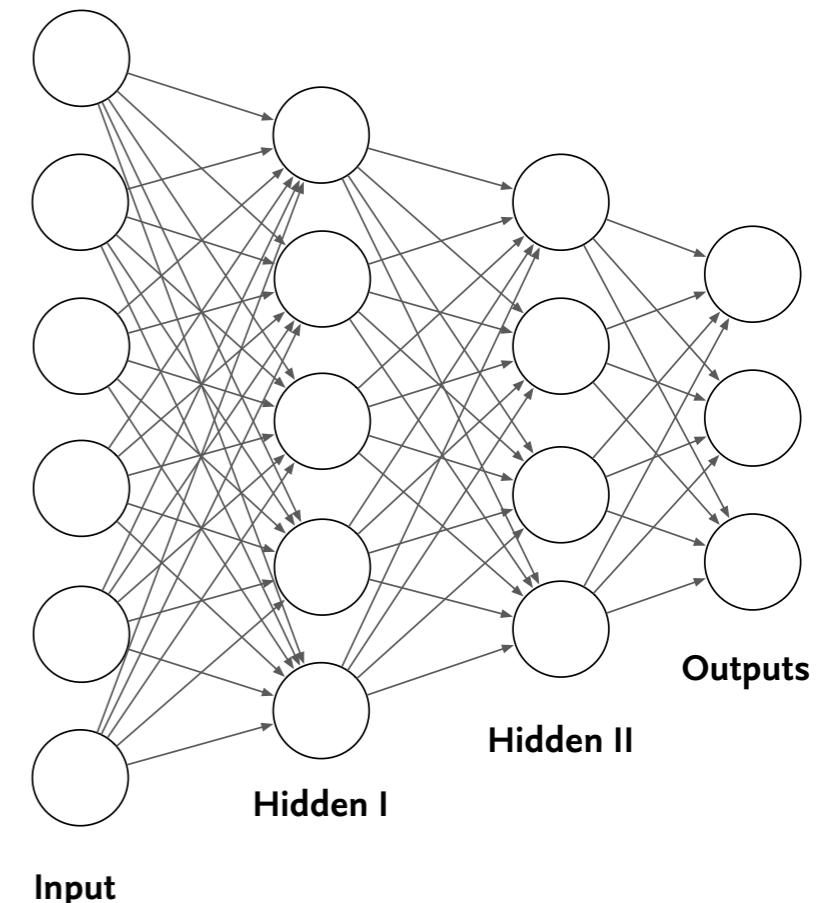
Stacked Autoencoders

- **Input layer of first AE is the input layer for the network**
- **Hidden layer of AE t as input layer to AE t + 1**



Training Stacked Autoencoders

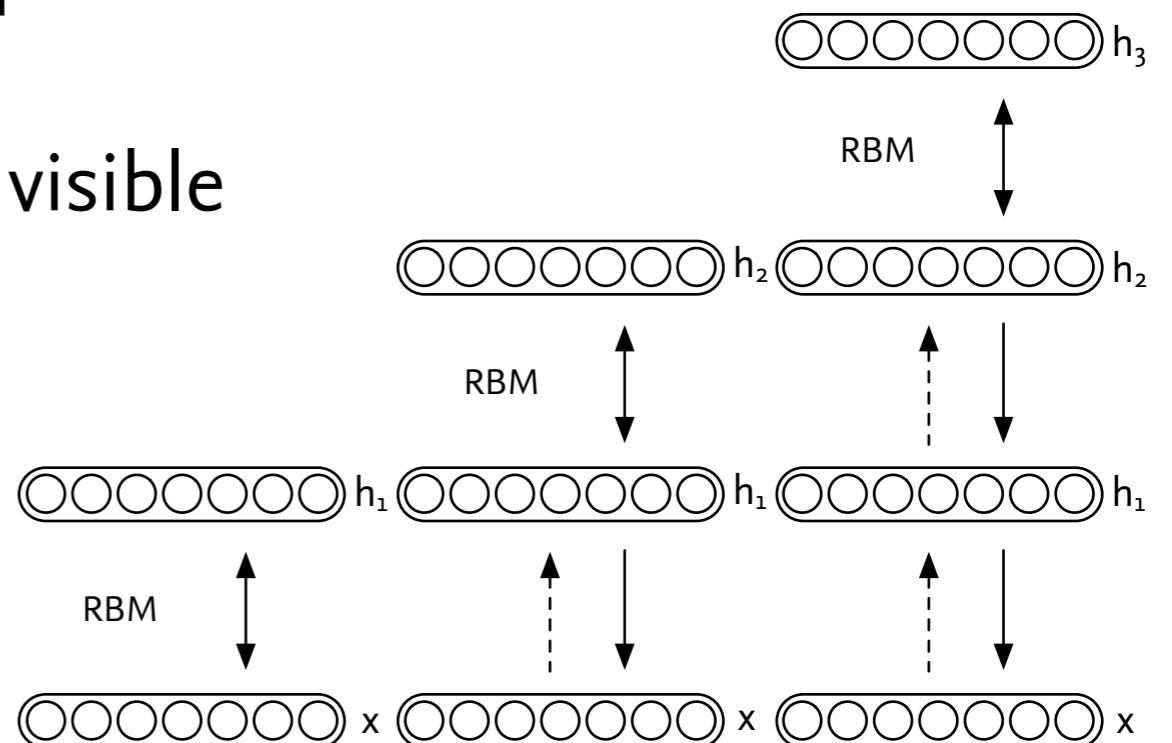
- 1. Train first AE with additional output layer**
 - All training data + backpropagation
- 2. Train second AE with hidden I as input**
 - Propagate sample from input of first AE
 - Update weights using backpropagation
- 3. Repeat 1-2 for all layers → pre-training**
- 4. Add one fully connected layer and train with BP → fine-tuning**



Deep Belief Networks

- **Stacked Restricted Boltzmann Machines**

- Input layer of first RBM is the input layer for the network
- Hidden layer of RBM t becomes visible layer of RBM $t + 1$



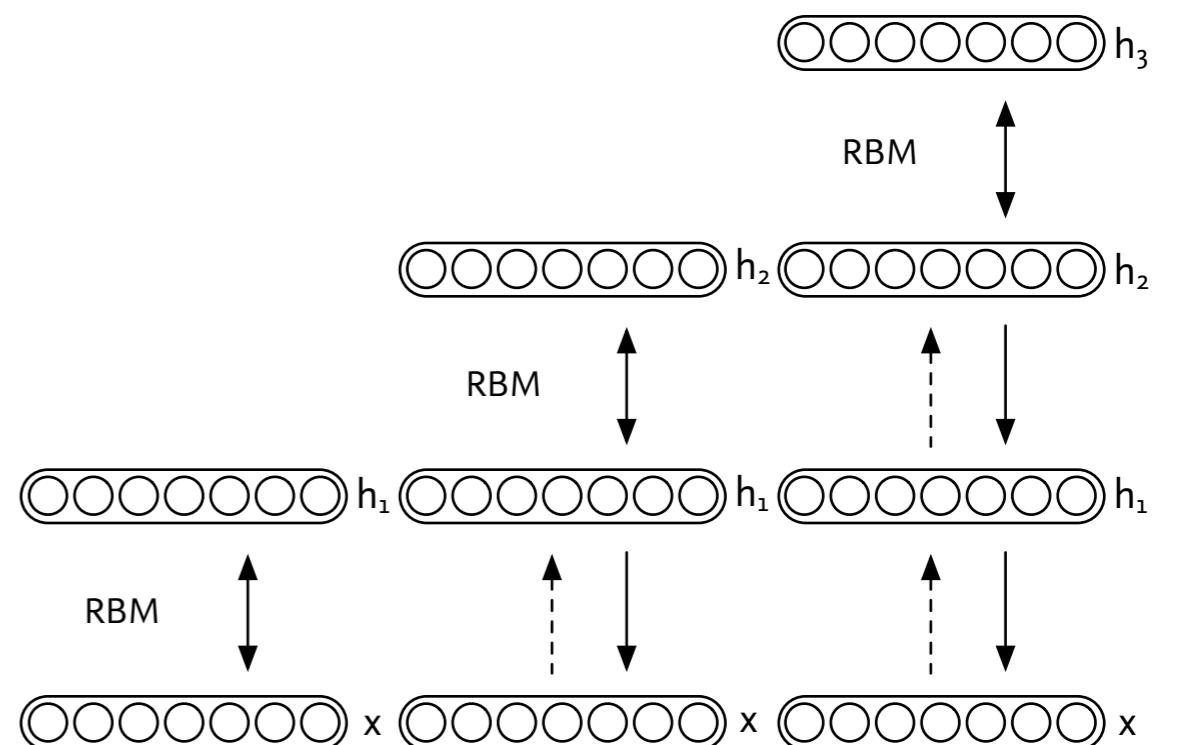
Training Deep Belief Networks

- 1. Train first RBM using contrastive divergence**

- 2. Train second RBM with hidden 1 as visible layer**

- 3. Repeat 1-2 for all layers**

- 4. Add one fully connected layer and train with BP**

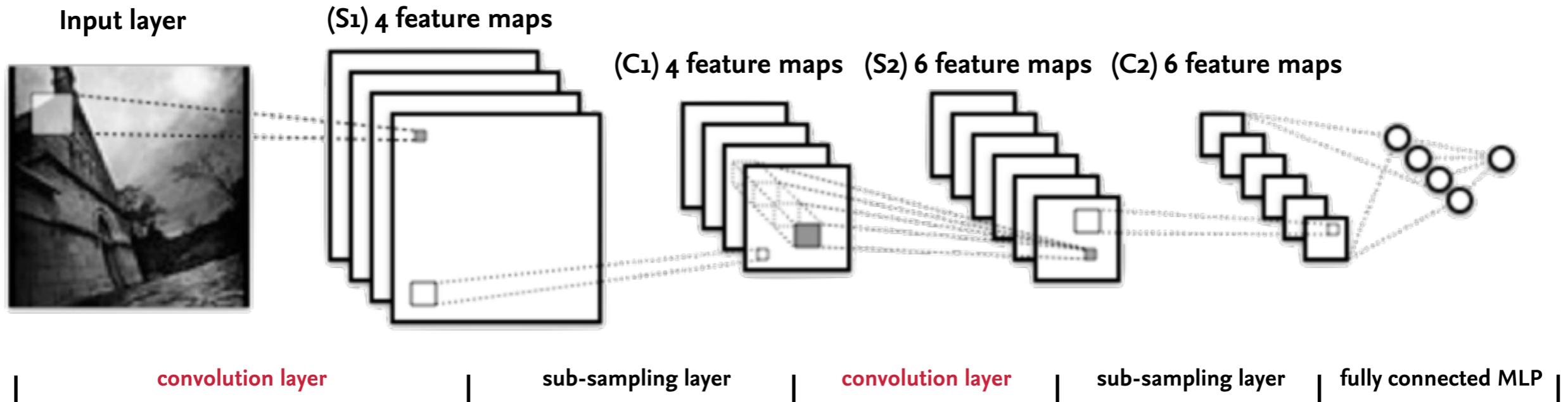


Convolutional Networks

- Feedforward non-recurrent neural network
- Specially suited for image recognition
- Based on the concept of **filters**



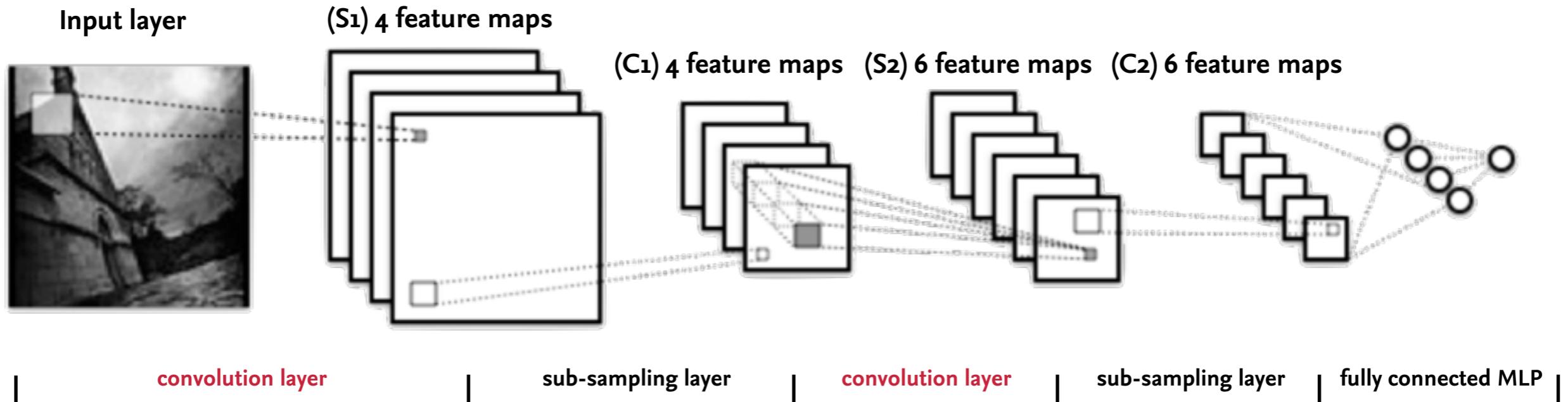
Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

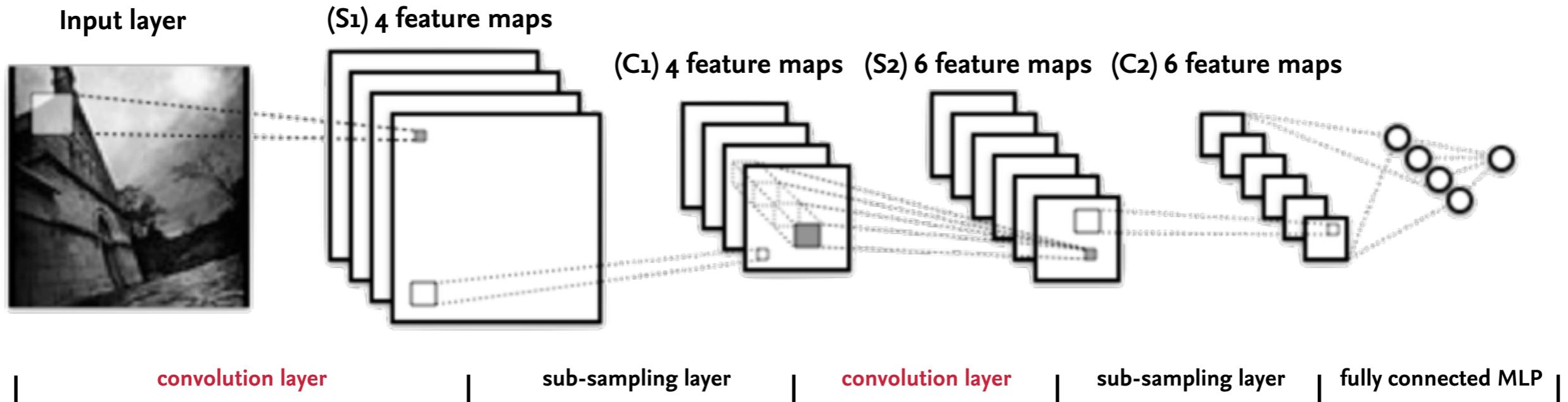
Image

4		

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1 _{x1}	1 _{x0}	0 _{x1}	0
0	1 _{x0}	1 _{x1}	1 _{x0}	0
0	0 _{x1}	1 _{x0}	1 _{x1}	1
0	0	1	1	0
0	1	1	0	0

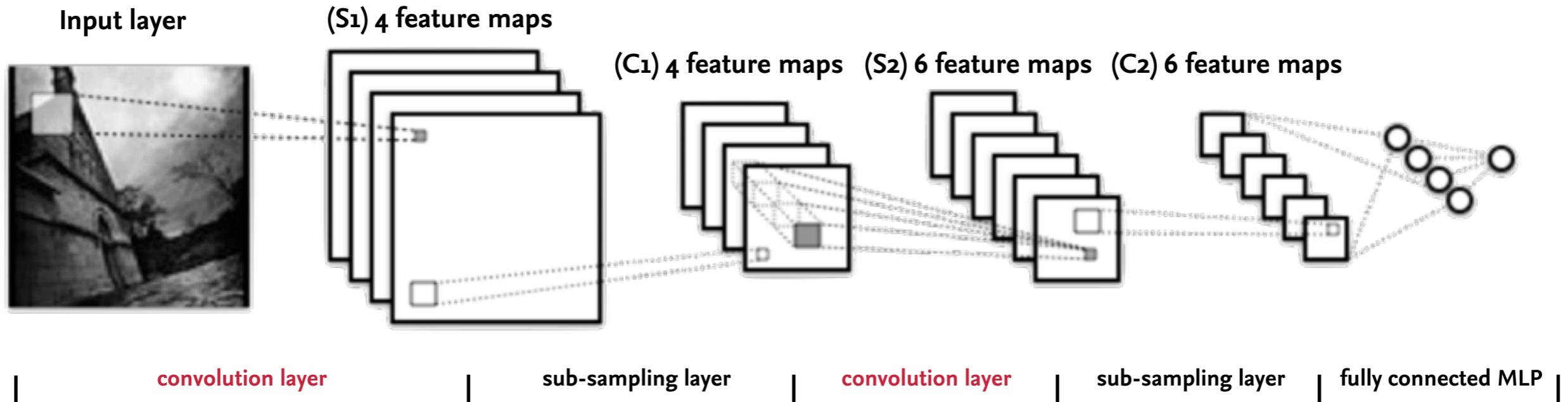
Image

4	3	

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	$\times 1$	0	$\times 0$	0	$\times 1$
0	1	1	$\times 0$	1	$\times 1$	0	$\times 0$
0	0	1	$\times 1$	1	$\times 0$	1	$\times 1$
0	0	1	1	1	0		
0	1	1	0	0			

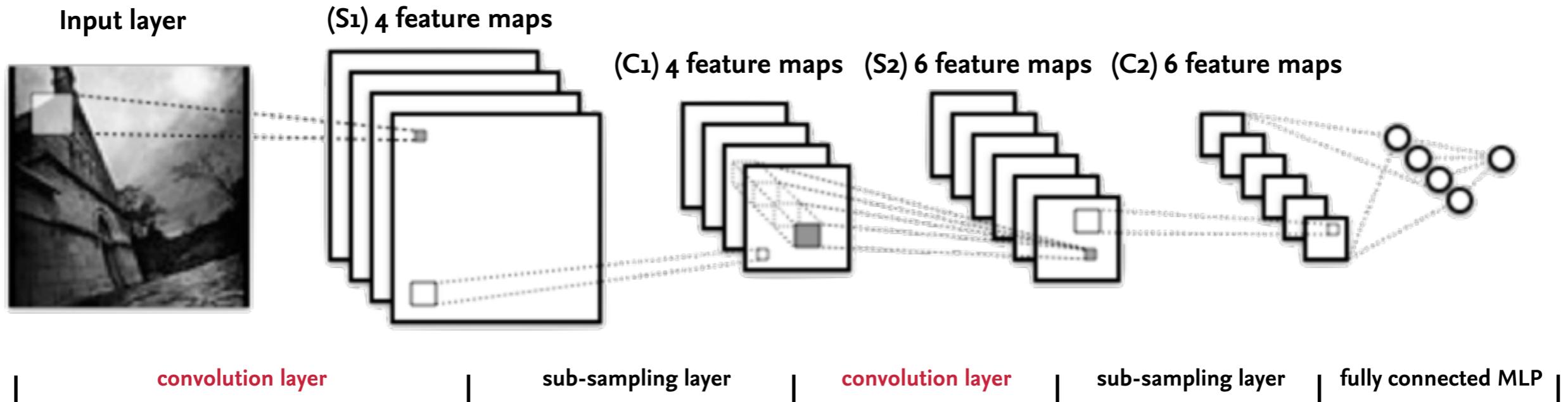
Image

4	3	4

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	1	0
0 <small>x0</small>	0 <small>x1</small>	1 <small>x0</small>	1	1
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	0
0	1	1	0	0

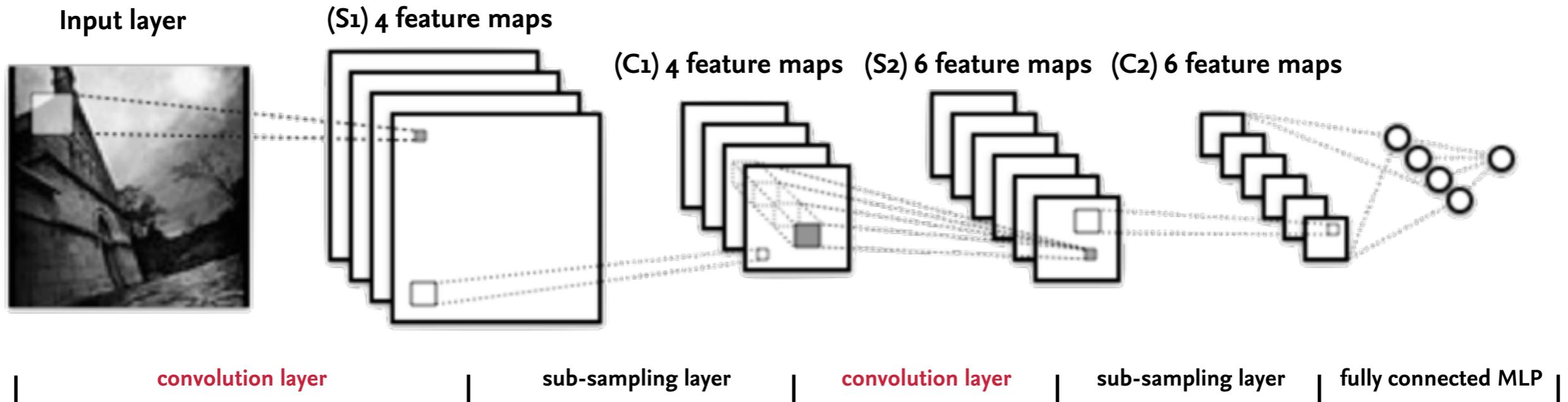
Image

4	3	4
2		

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1 _{x1}	1 _{x0}	1 _{x1}	0
0	0 _{x0}	1 _{x1}	1 _{x0}	1
0	0 _{x1}	1 _{x0}	1 _{x1}	0
0	1	1	0	0

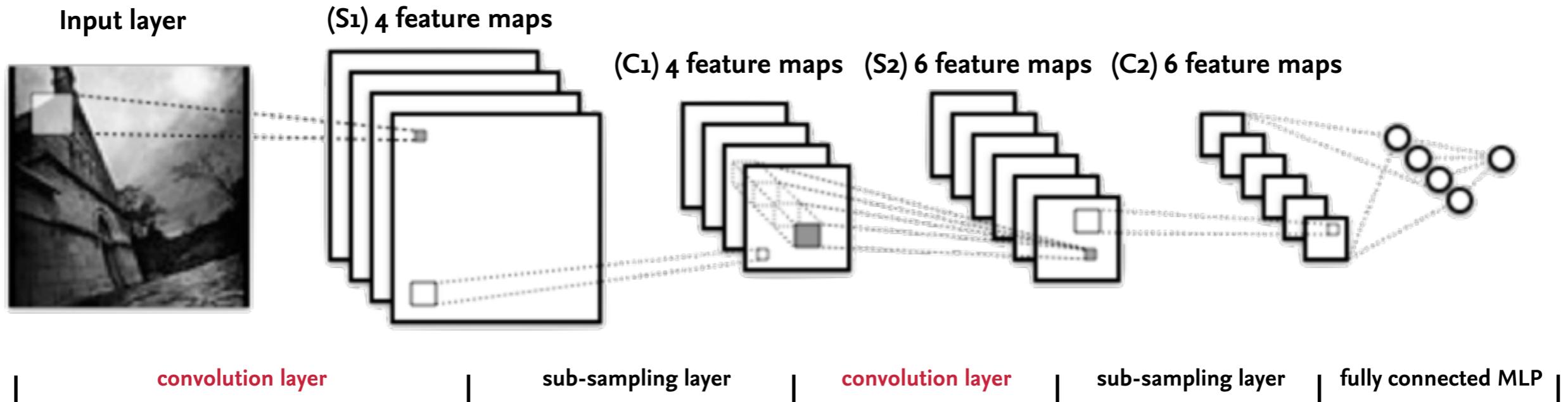
Image

4	3	4
2	4	

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1	1 _{x1}	1 _{x0}	0 _{x1}
0	0	1 _{x0}	1 _{x1}	1 _{x0}
0	0	1 _{x1}	1 _{x0}	0 _{x1}
0	1	1	0	0

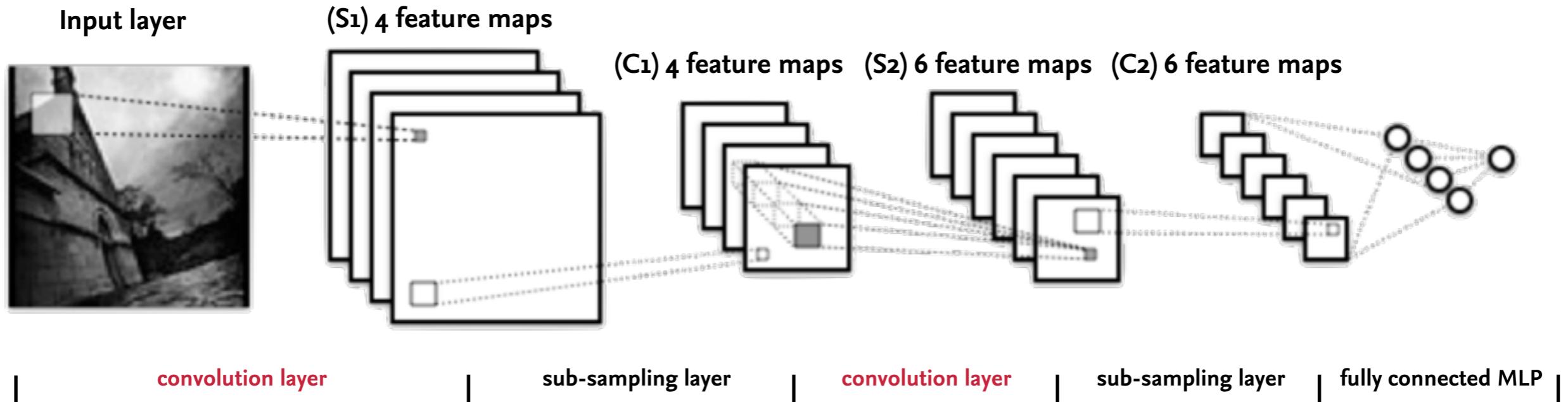
Image

4	3	4
2	4	3

Convolved
Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

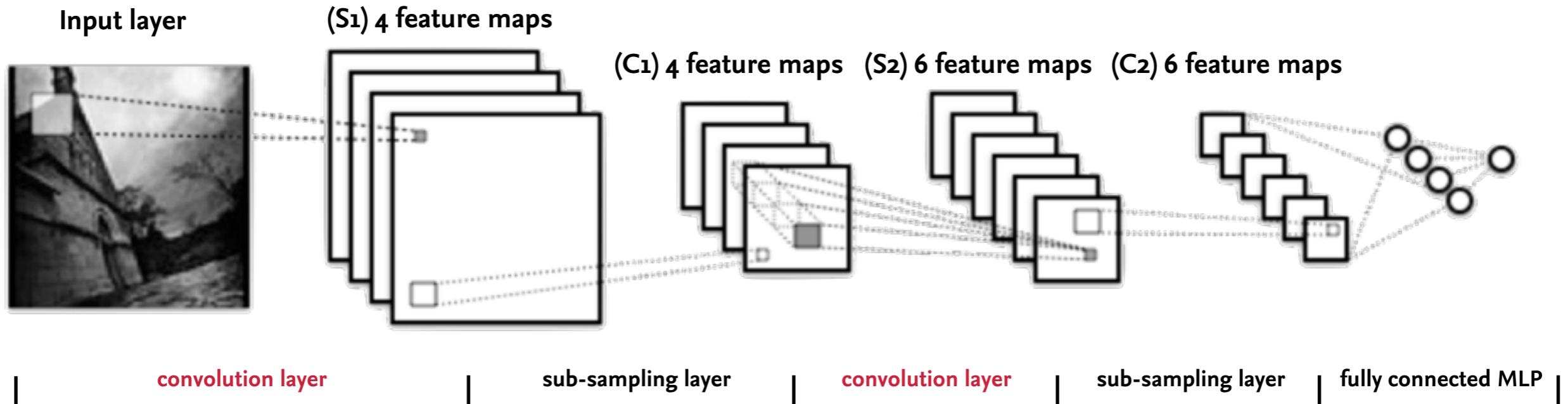
Image

4	3	4
2	4	3
2		

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

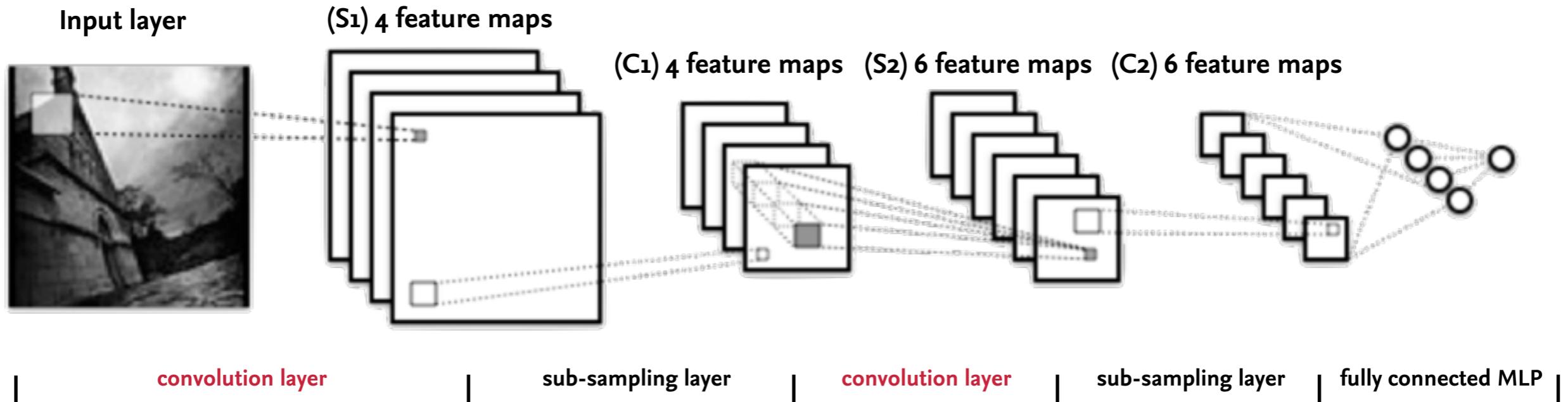
Image

4	3	4
2	4	3
2	3	

Convolved Feature



Convolution Layers



- Apply a number of filters to input
- Results of a filter applied to the image is a **feature map**

1	1	1	0	0
0	1	1	1	0
0	0	1	$\times 1$	$\times 0$
0	0	1	$\times 0$	$\times 1$
0	1	1	$\times 1$	$\times 0$

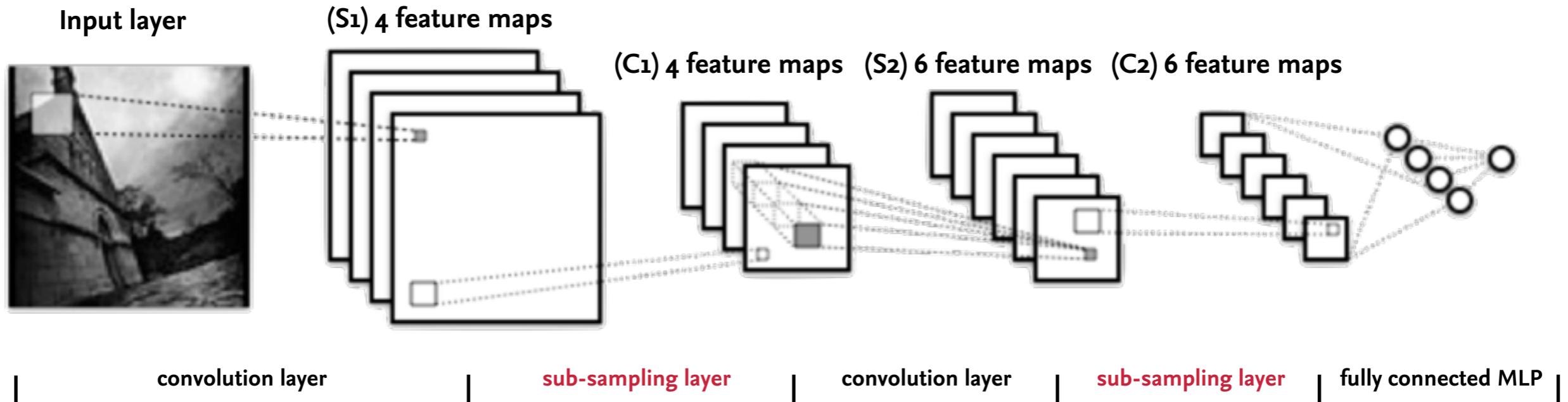
Image

4	3	4
2	4	3
2	3	4

Convolved
Feature



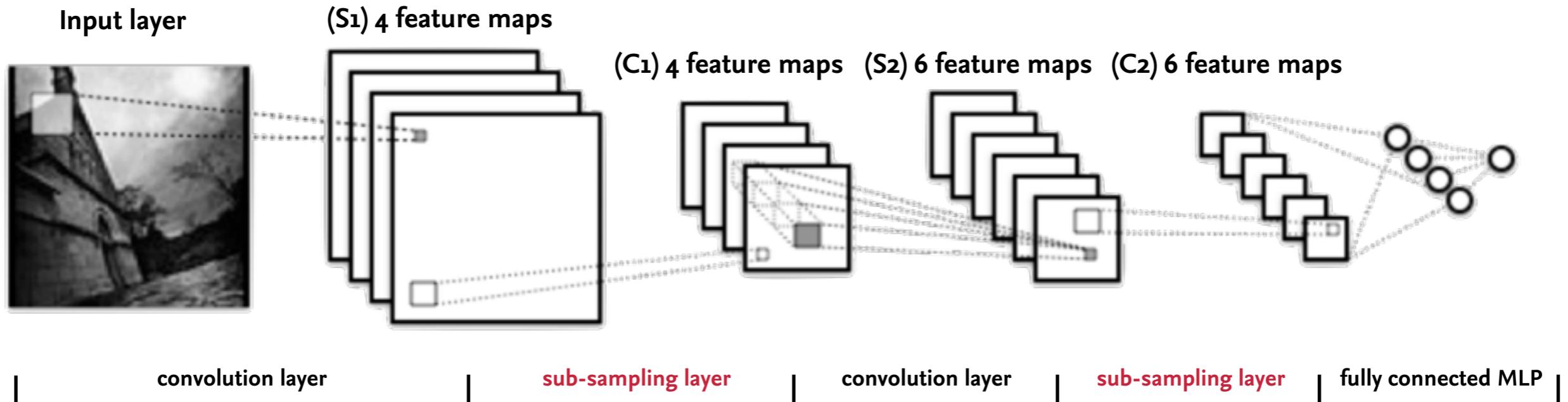
Sub-sampling Layers



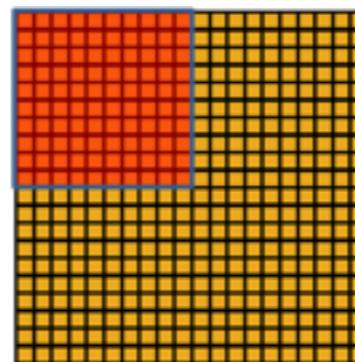
- Reduce the size of the input
- Max pooling, average pooling, stochastic pooling, etc



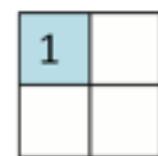
Sub-sampling Layers



- **Reduce the size of the input**
- **Max pooling, average pooling, stochastic pooling, etc**

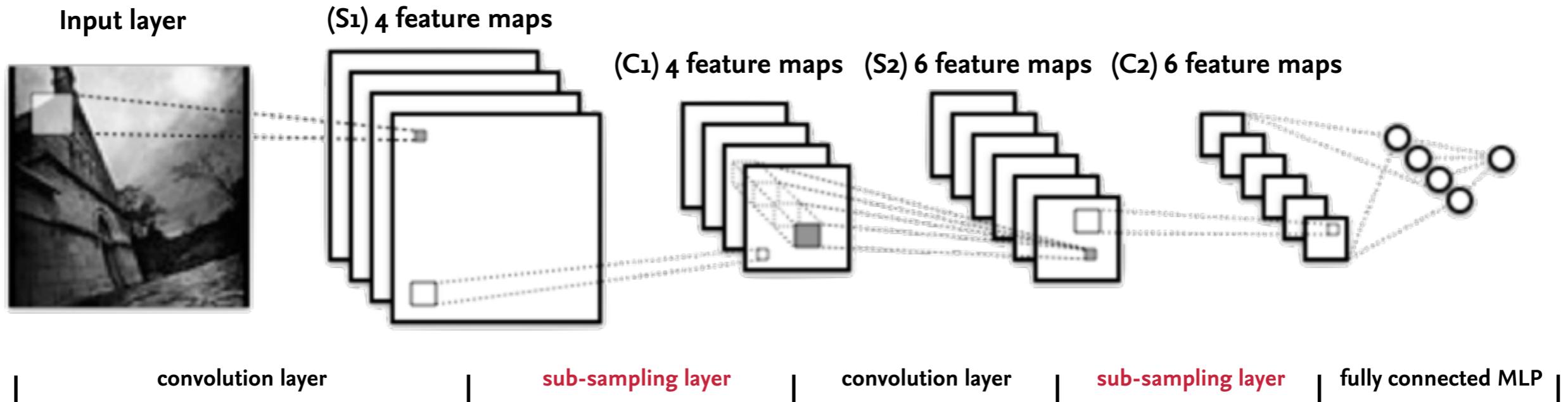


Convolved
feature

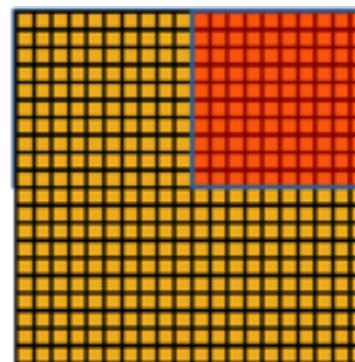


Pooled
feature

Sub-sampling Layers



- **Reduce the size of the input**
- **Max pooling, average pooling, stochastic pooling, etc**

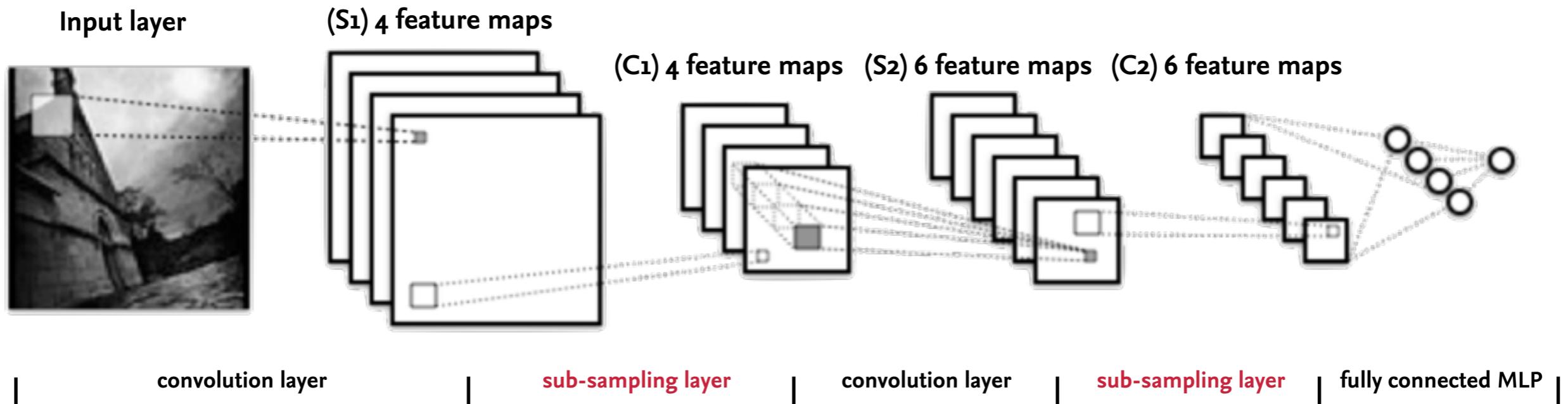


Convolved
feature

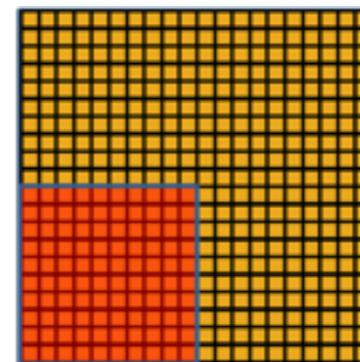


Pooled
feature

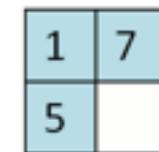
Sub-sampling Layers



- **Reduce the size of the input**
- **Max pooling, average pooling, stochastic pooling, etc**

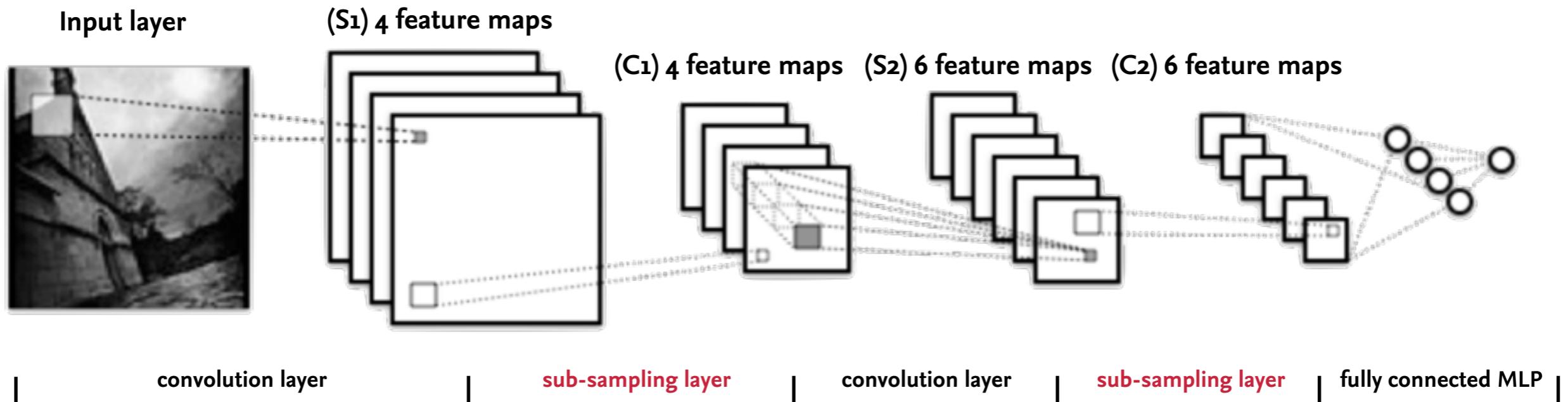


Convolved
feature

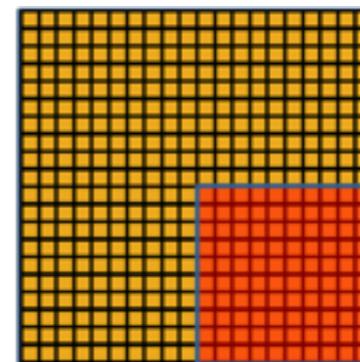


Pooled
feature

Sub-sampling Layers



- **Reduce the size of the input**
- **Max pooling, average pooling, stochastic pooling, etc**



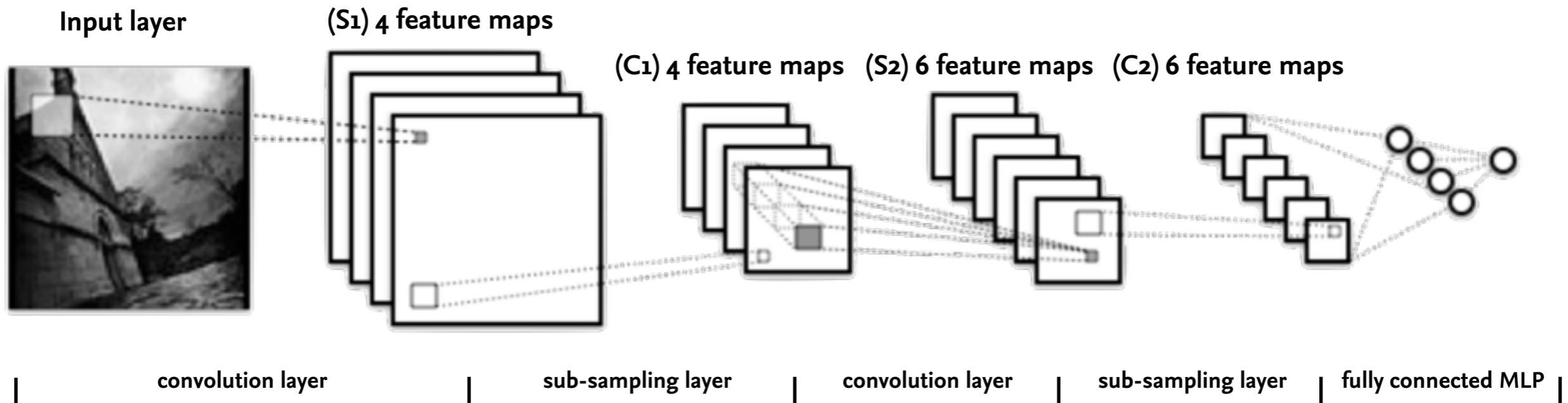
1	7
5	9

Convolved
feature

Pooled
feature



Convolutional Networks



- **Training**
 - Backpropagation
 - Consider sub-sampling values
 - Update weights of convolutional filters



Deep Learning Security Applications



Current Research

- **Actually, almost non-existent**
 - SVMs default ML algorithm for security problems
 - General renewed interest in deep NN only since 2006
- **However, some preliminary attempts exist, e.g.:**
 - Convolutional networks for binary classification
 - Stacked autoencoders for protocol identification
 - Recognizing functions in binaries using Recurrent NN



Deep Convolutional Networks for Malware Detection

- **Convnets assume **stationarity** and **compositionality** in data**
 - Spatial structure → pixels in image data
 - Temporal structure → audio samples in spoken language
- **Possible approach for malware analysis**
 - Represent binary code as image data or graphs
 - Find fixed size input vector representation
 - Train convolutional network



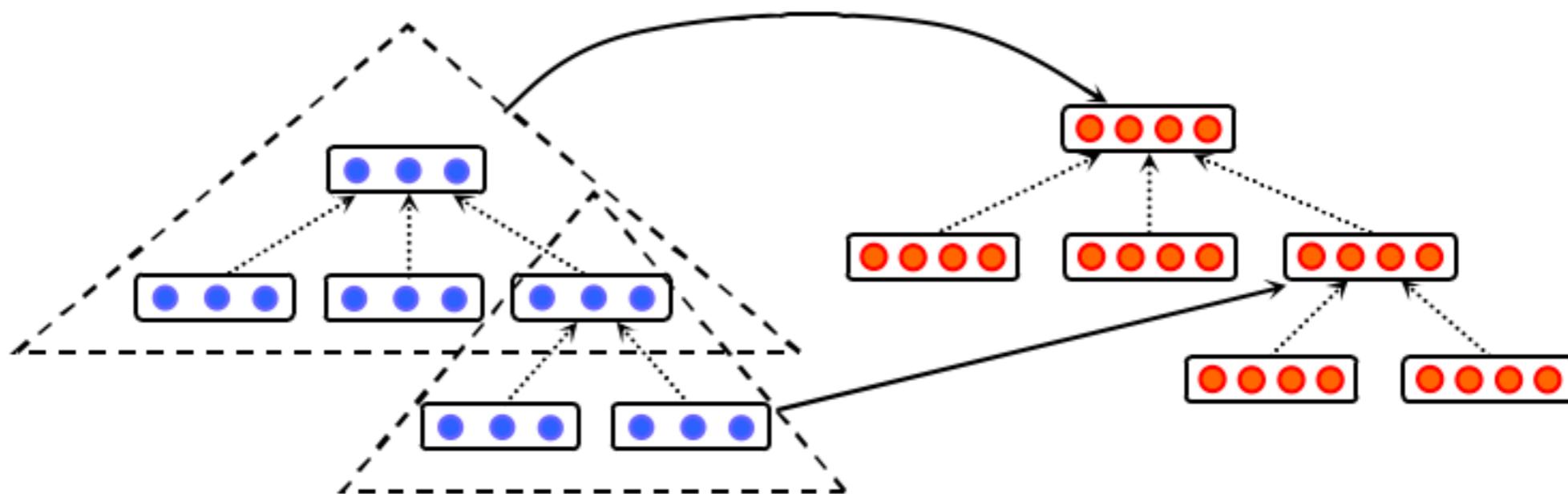
Spatial Structure in x86 Instructions

- Raw binary data as image



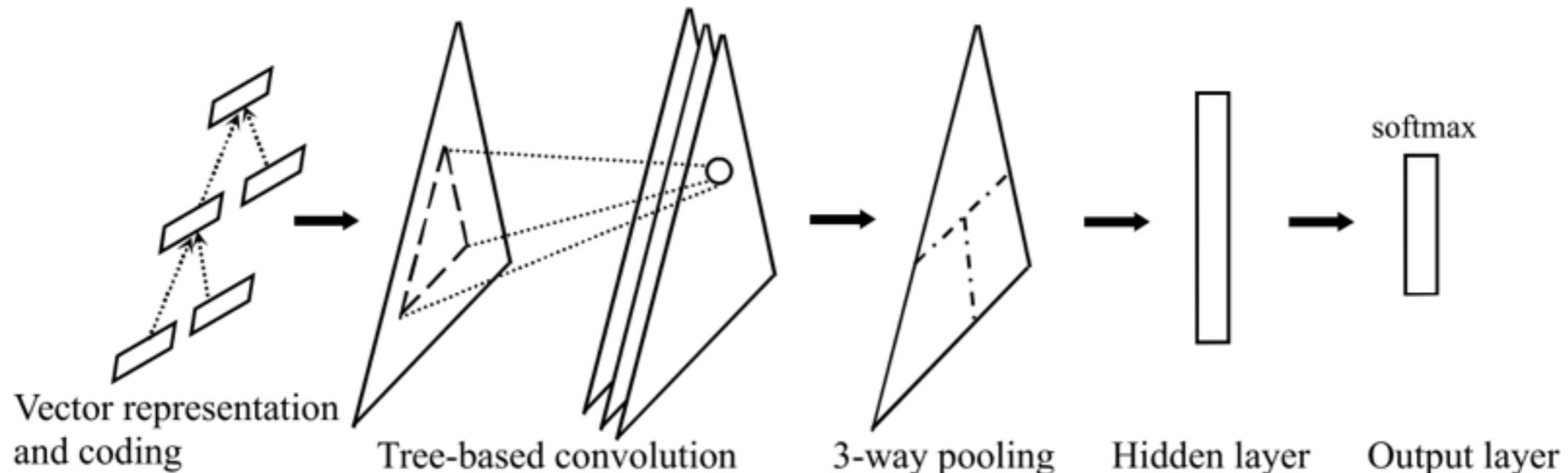
Spatial Structure in Call Graphs

- Tree-based convolution



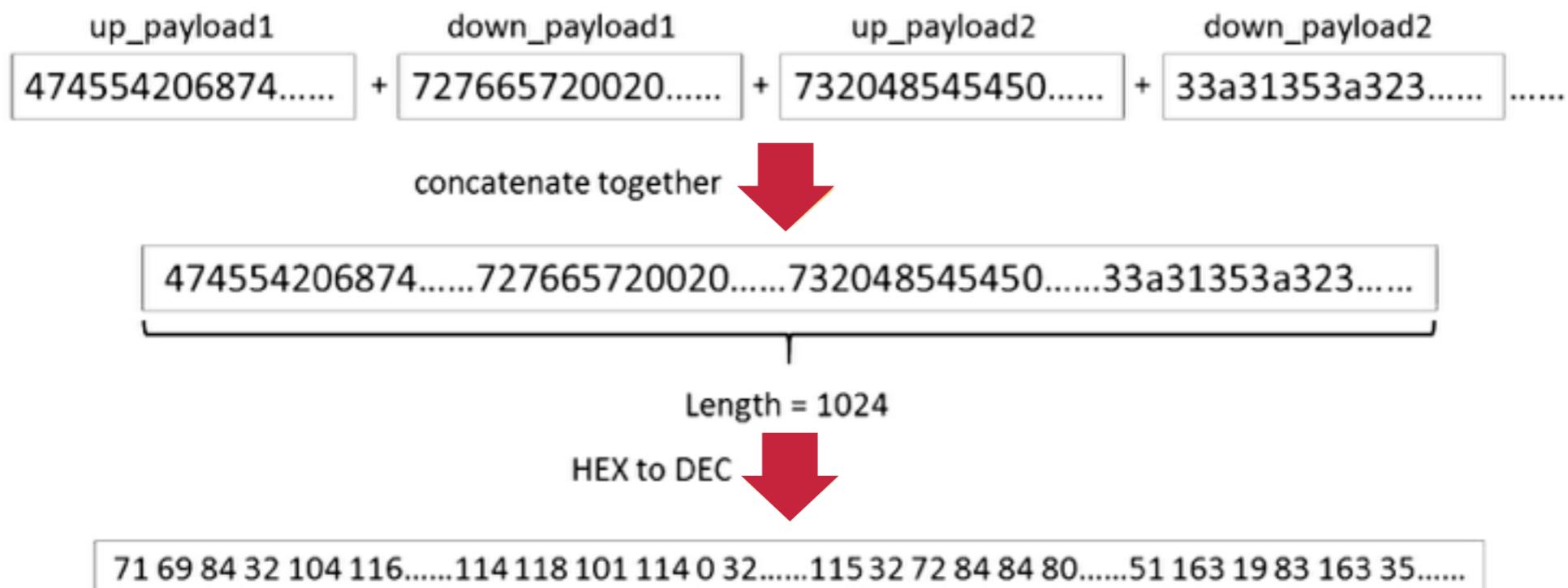
Spatial Structure in Code Graphs

- Tree-based convolutional network



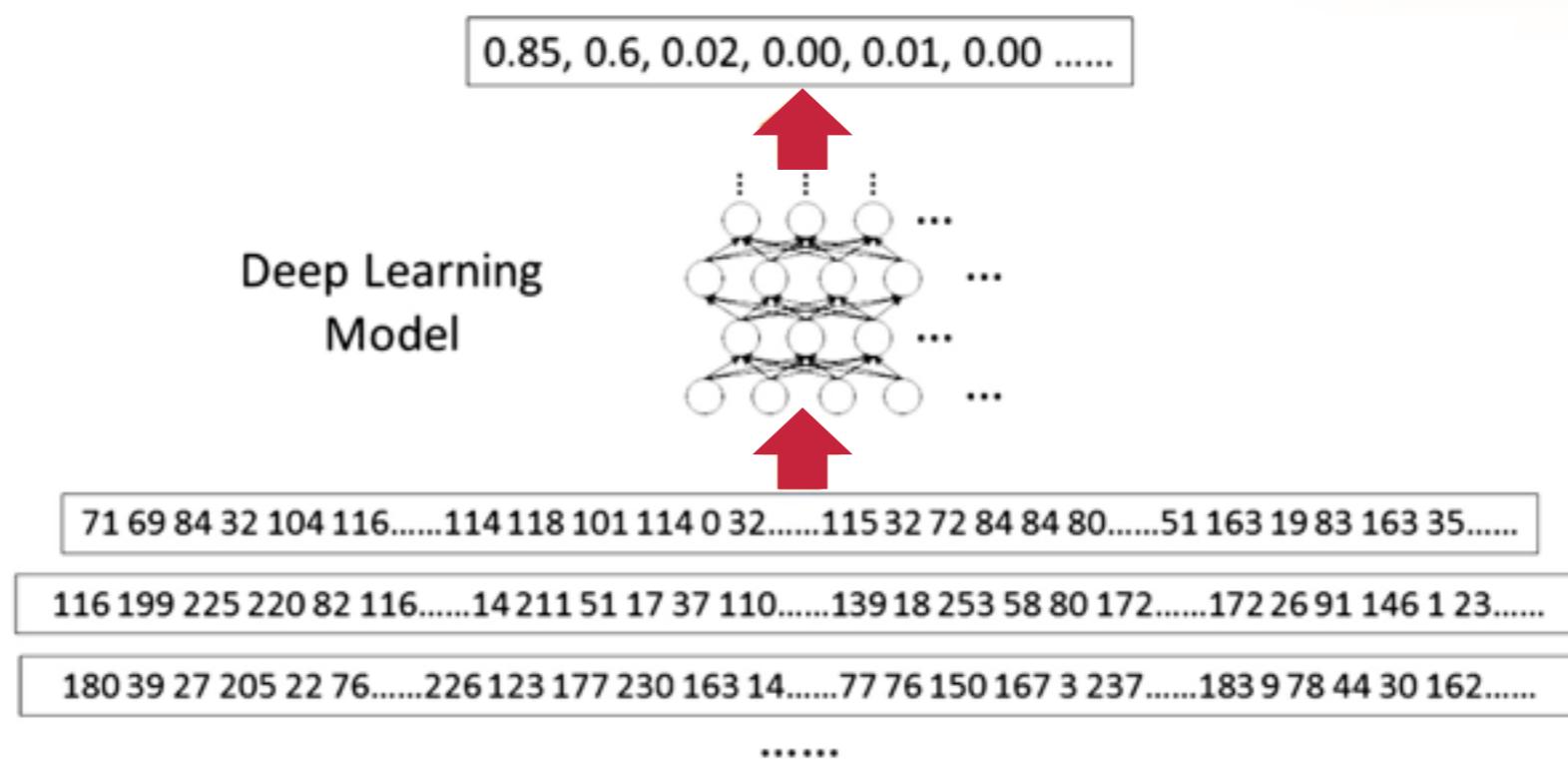
Deep Learning for Traffic Identification

- Concatenate payloads in 1024-dim vectors



Deep Learning for Traffic Identification

- Feed deep stack autoencoders to predict protocol type



Protocol classification → 97.9% avg. precision

Application identification → 96.3% avg. precision



Challenges & Limitations

- Security data is usually generated by non-ergodic processes
 - Deep networks assume **structure in data**
 - Lack of consistent statistical properties
- Security data is difficult to collect and label
 - Relevant security cases are typically unfrequent
 - Deep networks are effective when trained with **lots of data**



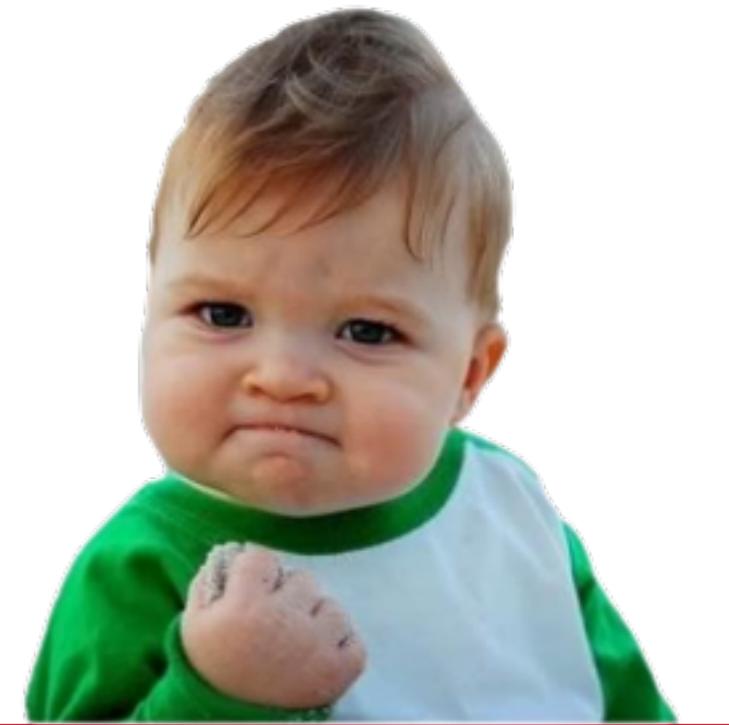
Tutorials & Tools

- Many introductory tutorials and How-To's
 - <http://deeplearning.net/reading-list/>
- Well documented and open source frameworks



Summary

- **Neural Networks & Deep Learning**
 - Deep nets → stacked autoencoders and RBMs
 - Efficient learning with **greedy layer-wise training** and **dropout**
- **Security applications are till in its dawn**
 - Opportunity for smarter **representations**
 - and **large-scale** approaches



References

- Chul, E., Shin, R., Song, D., & Moazzezi, R. (2015). Recognizing Functions in Binaries with Neural Networks. USENIX Security.
- Dahl, G., & Stokes, J. (2013). Large-scale malware classification using random projections and neural networks.
- Jung, W., & Kim, S. (2015). Poster : Deep Learning for Zero-day Flash Malware Detection. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2–3.
- Ma, M., Huang, L., Xiang, B., & Zhou, B. (2015). Dependency-based Convolutional Neural Networks for Sentence Embedding. Acl-2015, (1995), 174–179.
- Mou, L., Li, G., Jin, Z., Zhang, L., & Wang, T. (2014). TBCNN: A Tree-Based Convolutional Neural Network for Programming Language Processing. Retrieved from <http://arxiv.org/abs/1409.5718>
- Mou, L., Li, G., Liu, Y., Peng, H., & Jin, Z. (2014). Building Program Vector Representations for Deep Learning. arXiv Preprint arXiv: Retrieved from <http://arxiv.org/abs/1409.3358>
- Pascanu, R., Stokes, J. W., Sanossian, H., & Anil Thomas, M. M. (2015). Malware Classification With Recurrent Networks. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).
- Wang, Y., Cai, W., & Wei, P. (2016). A deep learning approach for detecting malicious JavaScript code. <http://doi.org/10.1002/sec>
- Yuan, Z., Lu, Y., Wang, Z., & Xue, Y. (2014). Droid-Sec: Deep Learning in Android Malware Detection. Sigcomm 2014, 371–372. <http://doi.org/10.1145/2619239.2631434>