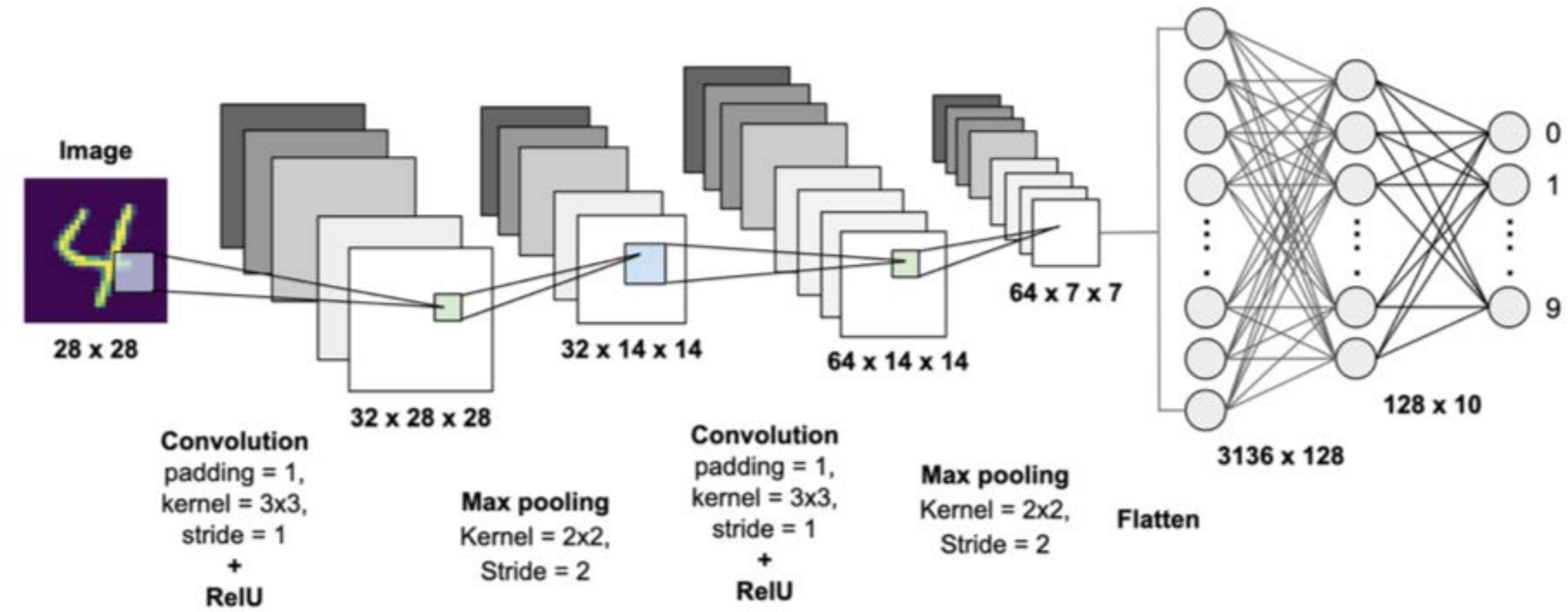
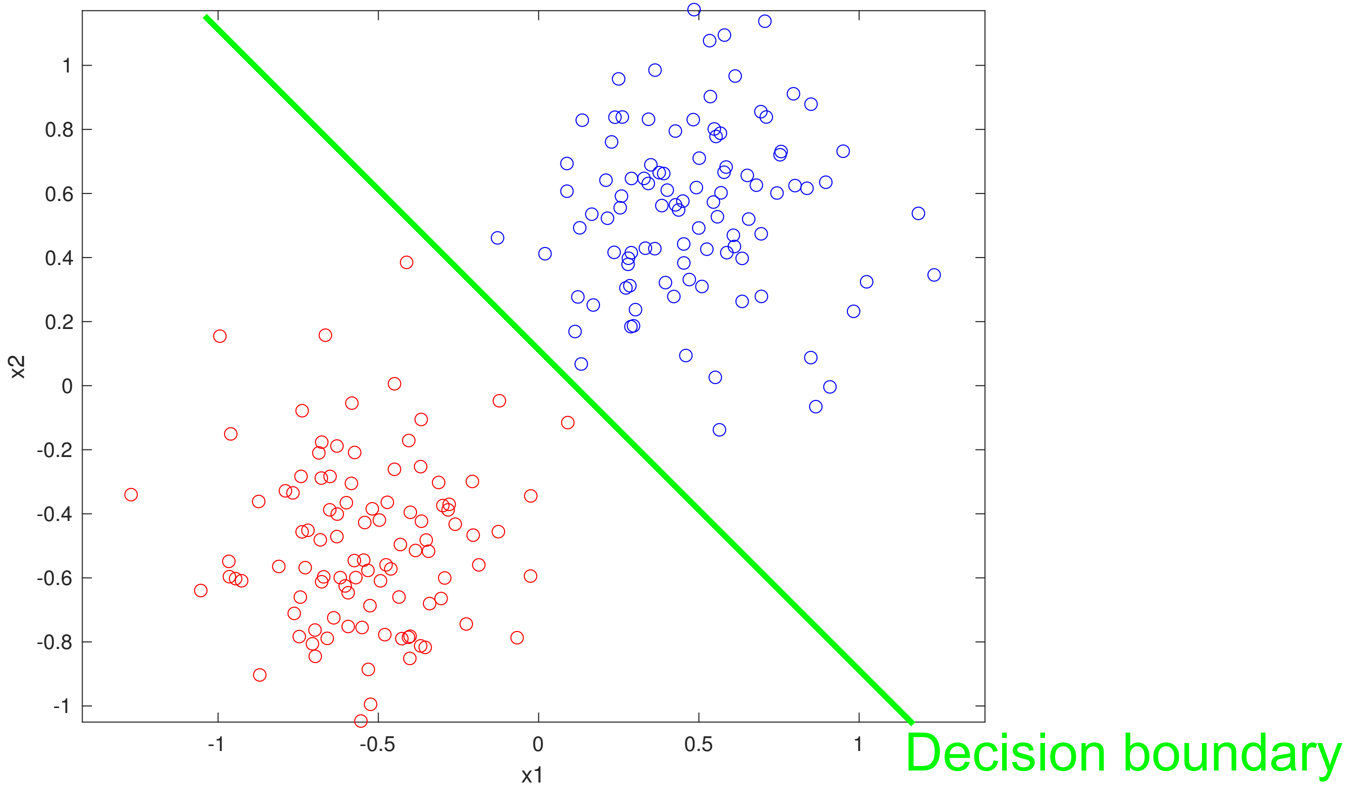
Deep Learning Crash Course



- Single Layer Perceptrons
- Multiple Layer Perceptrons
- Convolutional Neural Nets

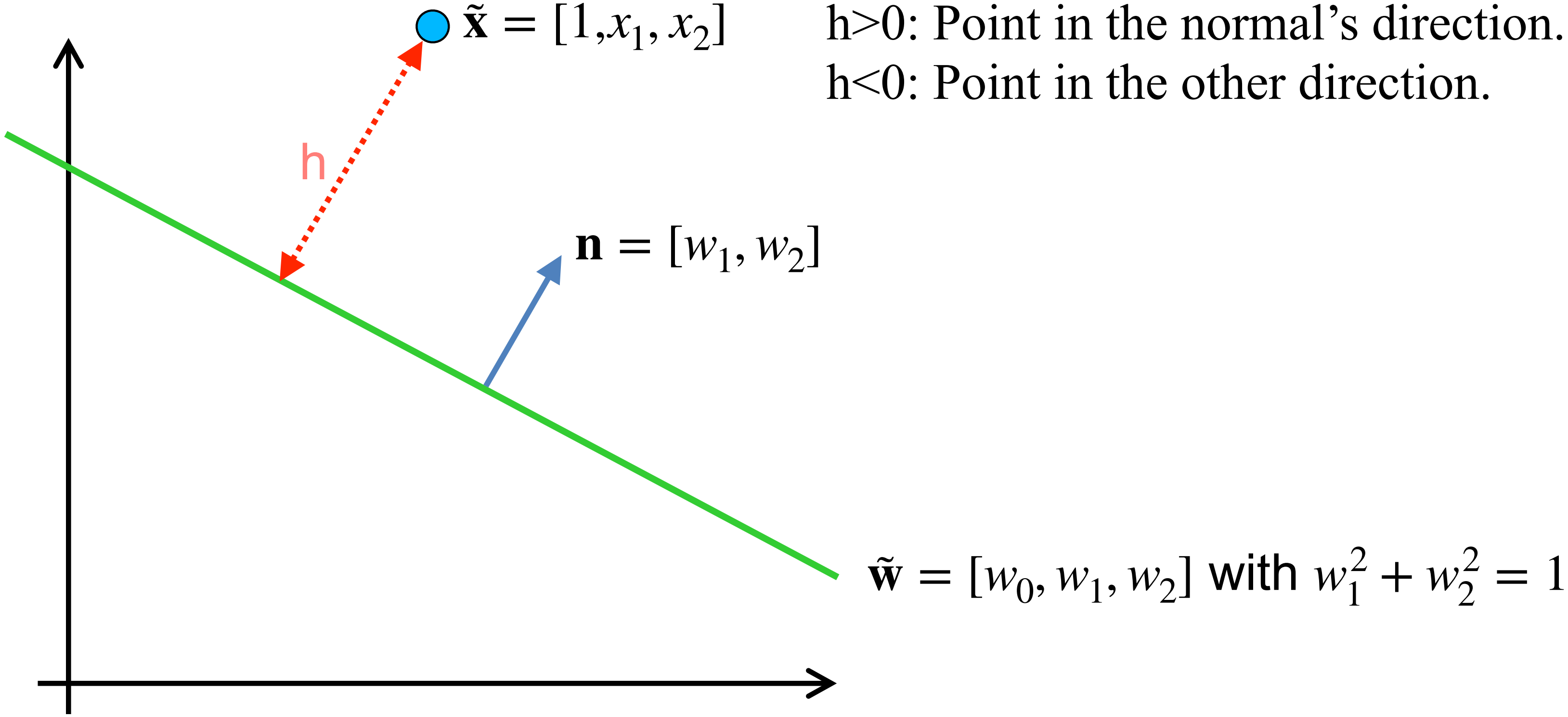
Linear Binary Classification



Two classes shown as different colors:

- The label $y \in \{-1,1\}$ or $y \in \{0,1\}$.
- The samples with label 1 are called positive samples.
- The samples with label -1 or 0 are called negative samples.

Signed Distance



h=0: Point is on the line.
h>0: Point in the normal's direction.
h<0: Point in the other direction.

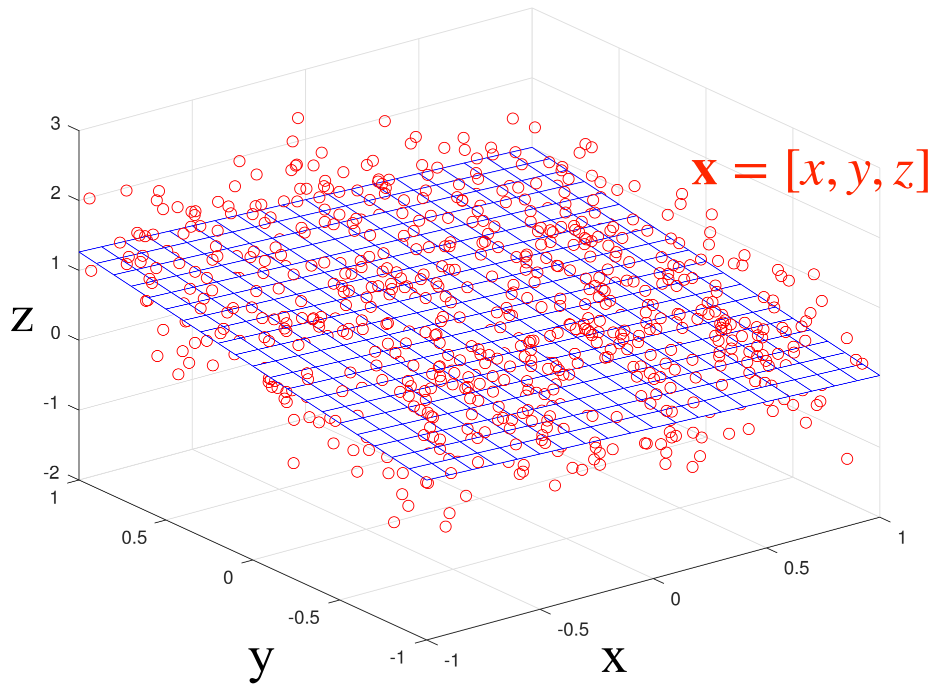
Notation:

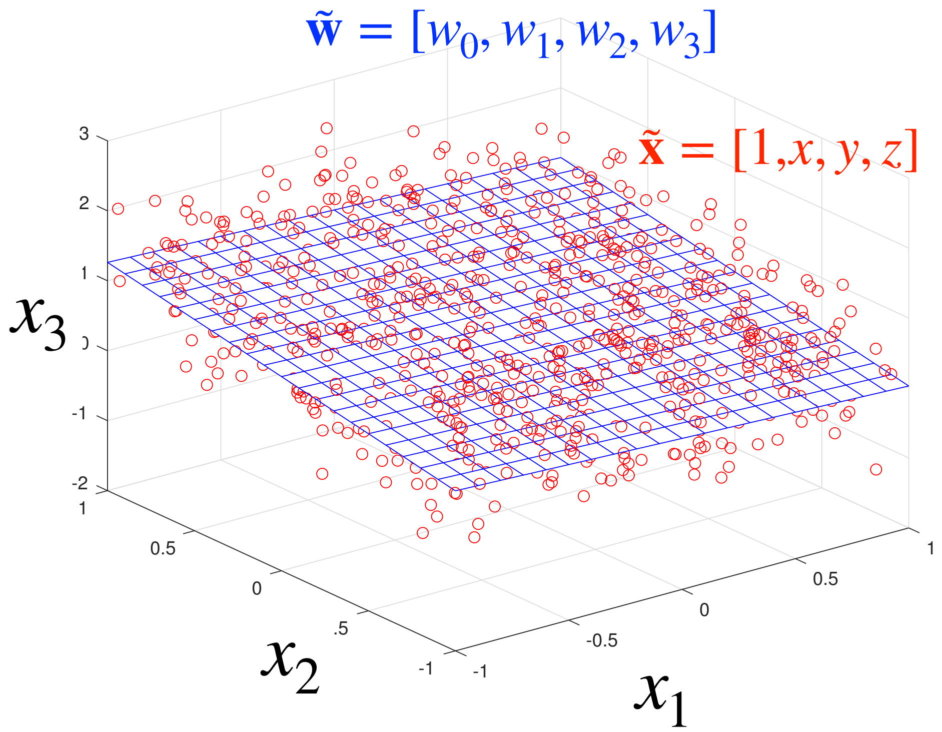
$$\mathbf{x} = [x_1, x_2]$$
$$\tilde{\mathbf{x}} = [1, x_1, x_2]$$

Signed distance:

$$h = w_0 + w_1 x_1 + w_2 x_2$$
$$= \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

Signed Distance in 3D





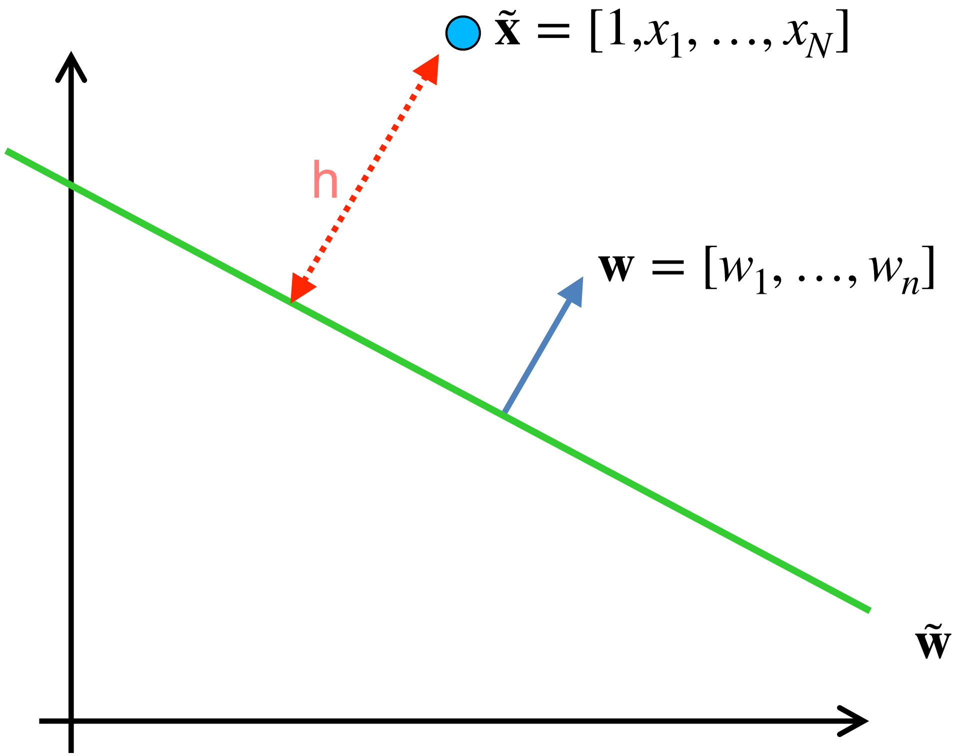
$$\mathbf{x} \in R^3, 0 = ax + by + cz + d$$

$$\tilde{\mathbf{x}} \in R^4, \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$$

Signed distance $h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$ if $w_1^2 + w_2^2 + w_3^2 = 1$.

EPFL

Signed Distance in N Dimensions



h=0: Point is on the decision boundary.
h>0: Point on one side.
h<0: Point on the other side.

$\tilde{\mathbf{w}} = [w_0, w_1, \ldots, w_n]$ with $\sum_{i=1}^{N} w_i^2 = 1$

Notation:

$$\mathbf{x} = [x_1, \ldots, x_n]$$
$$\tilde{\mathbf{x}} = [1, x_1, \ldots, x_n]$$

Hyperplane:

$$\mathbf{x} \in R^n, \quad 0 = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$
$$= w_0 + w_1 x_1 + \ldots w_n x_n$$

Signed distance:

$$h = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$$

Binary Classification in N Dimensions

Hyperplane: $\mathbf{x} \in R^N, \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0$, with $\tilde{\mathbf{x}} = [1 \,|\, \mathbf{x}]$.

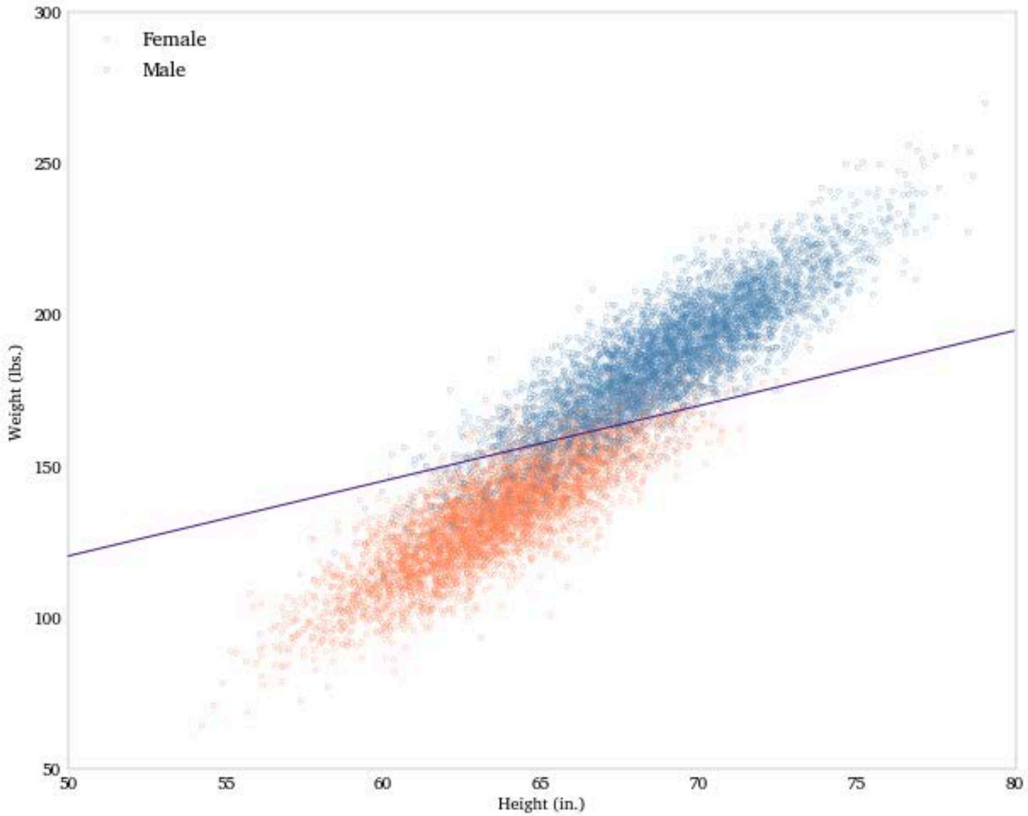
Signed distance: $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$, with $\tilde{\mathbf{w}} = [w_0 \,|\, \mathbf{w}]$ and $||\mathbf{w}|| = 1$.

Problem statement: Find $\tilde{\mathbf{w}}$ such that

- for all or most positive samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$,
- for all or most negative samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$.

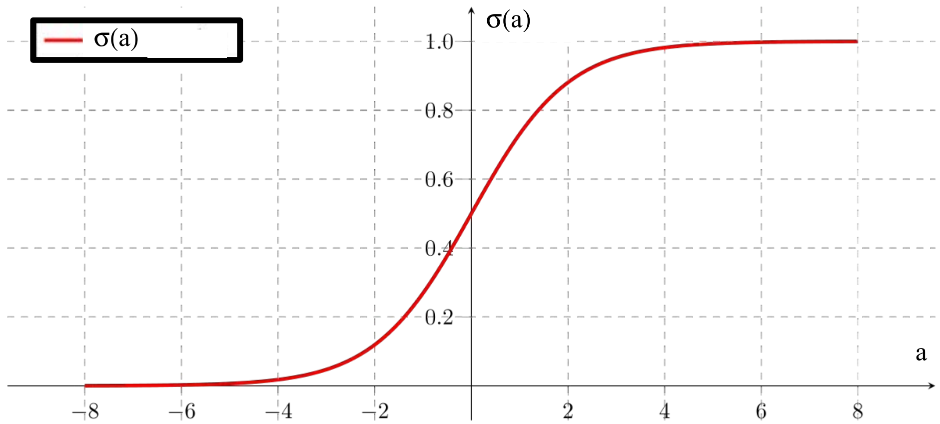
EPFL

Logistic Regression



$$y(\mathbf{x}; \tilde{\mathbf{w}}) = \sigma(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})$$

$$= \frac{1}{1 + \exp(-\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}})}$$



Given a **training** set $\{(\mathbf{x}_n, t_n)_{1 \le n \le N}\}$ minimize

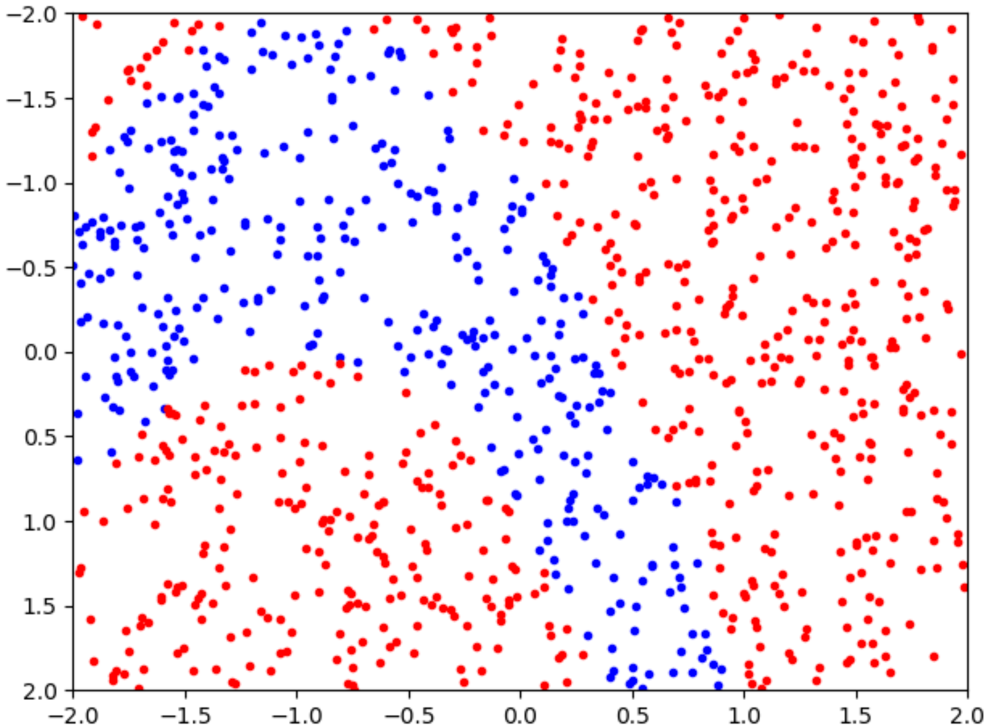
$$-\sum_n (t_n \ln y(\mathbf{x}_n) + (1 - t_n) \ln(1 - y(\mathbf{x}_n))$$

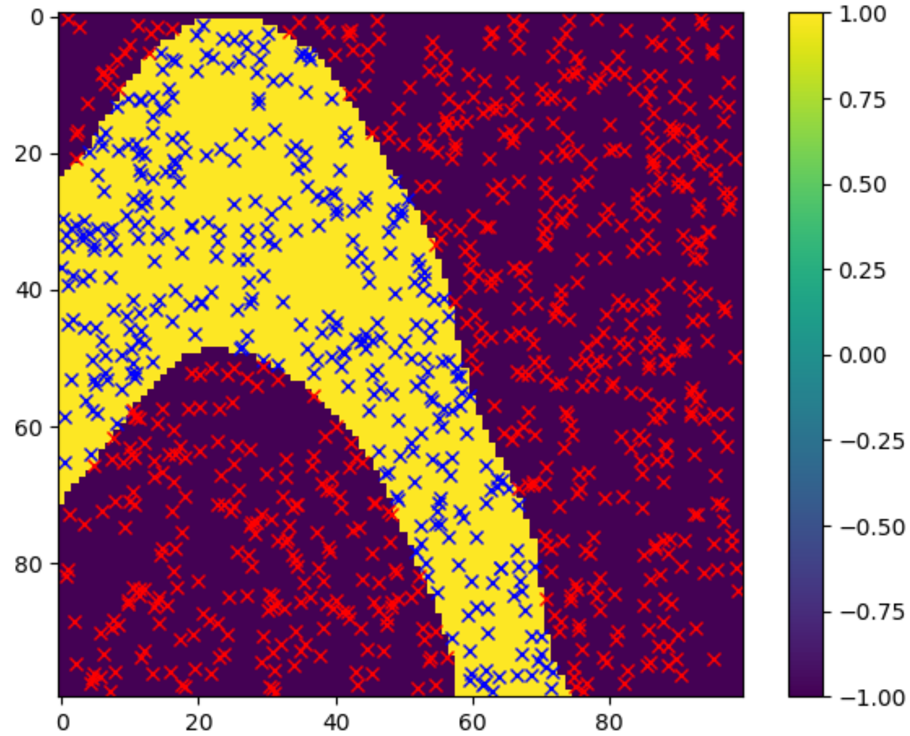
with respect to $\tilde{\mathbf{w}}$.

- When the noise is Gaussian, this is the maximum likelihood solution.
- $y(\mathbf{x}; \tilde{\mathbf{w}})$ can be interpreted at the probability that $\mathbf{x}$ belongs to positive class.

EPFL

Non Separable Distribution





Positive: $100(x_2 - x_1^2)^2 + (1 - x_1)^2 < 0.5$

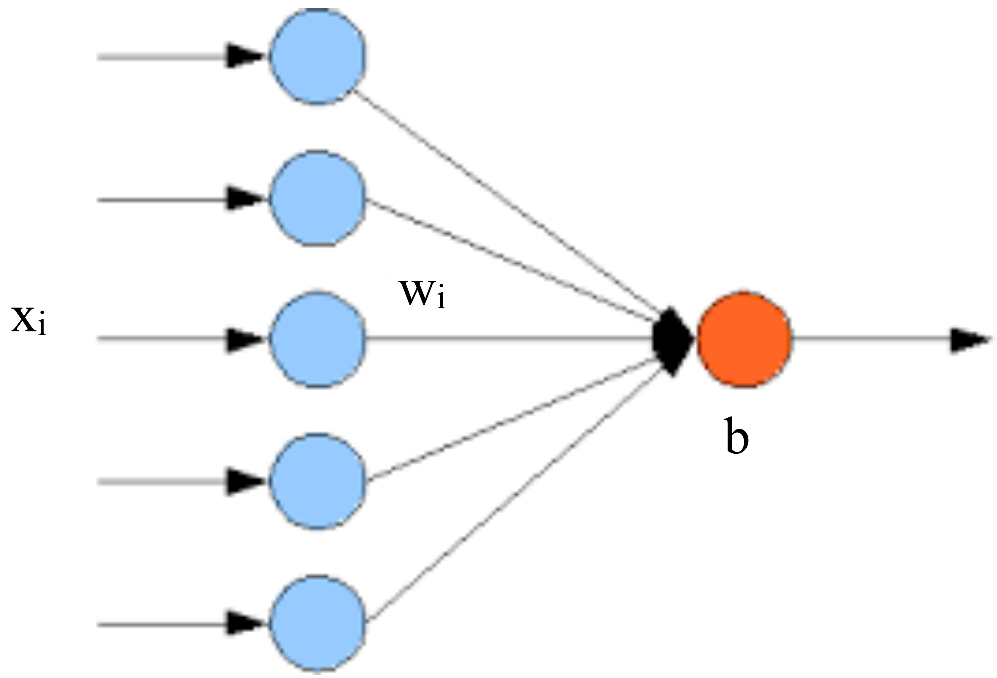
Negative: Otherwise

$y(\mathbf{x}; \tilde{\mathbf{w}})$ must be a non-linear function.

- Logistic regression can handle a few outliers but not a complex non-linear boundary.
- How can we learn a function y such that $y(\mathbf{x}; \tilde{\mathbf{w}})$ is close to 1 for positive samples and close to 0 or -1 for negative ones?

—> Use LOTS of hyperplanes.

Reformulating Logistic Regression

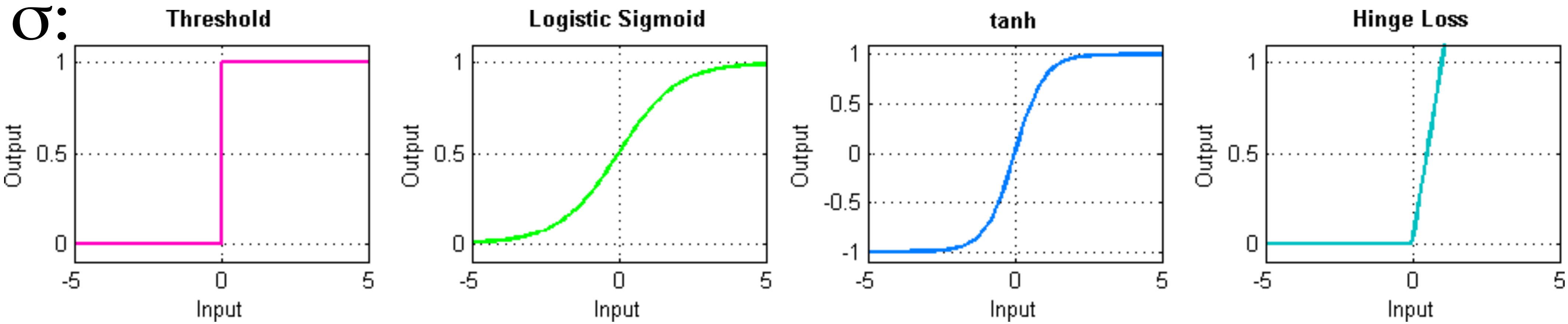


$$y(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

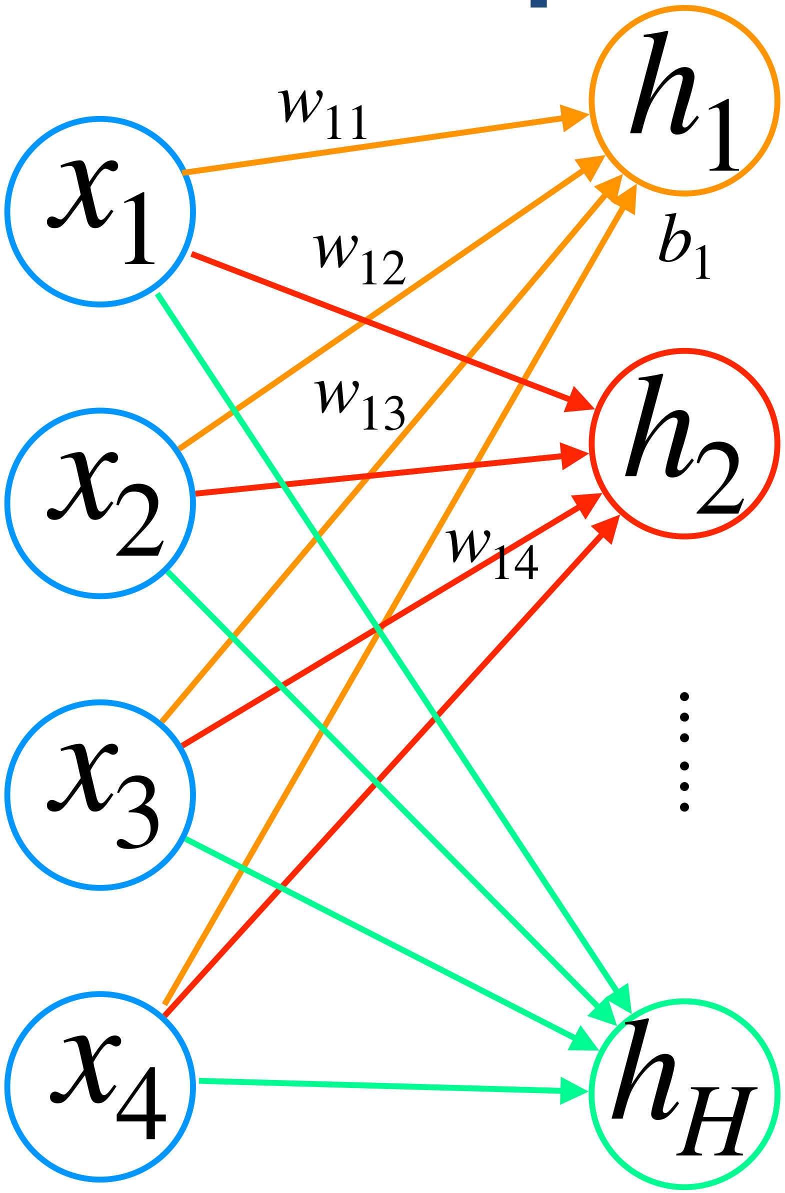
$$\mathbf{x} = \left[x_1, x_2, \ldots, x_n\right]^T$$

$$\mathbf{w} = \left[w_1, w_2, \ldots, w_n\right]^T$$

σ:



Repeating the Process



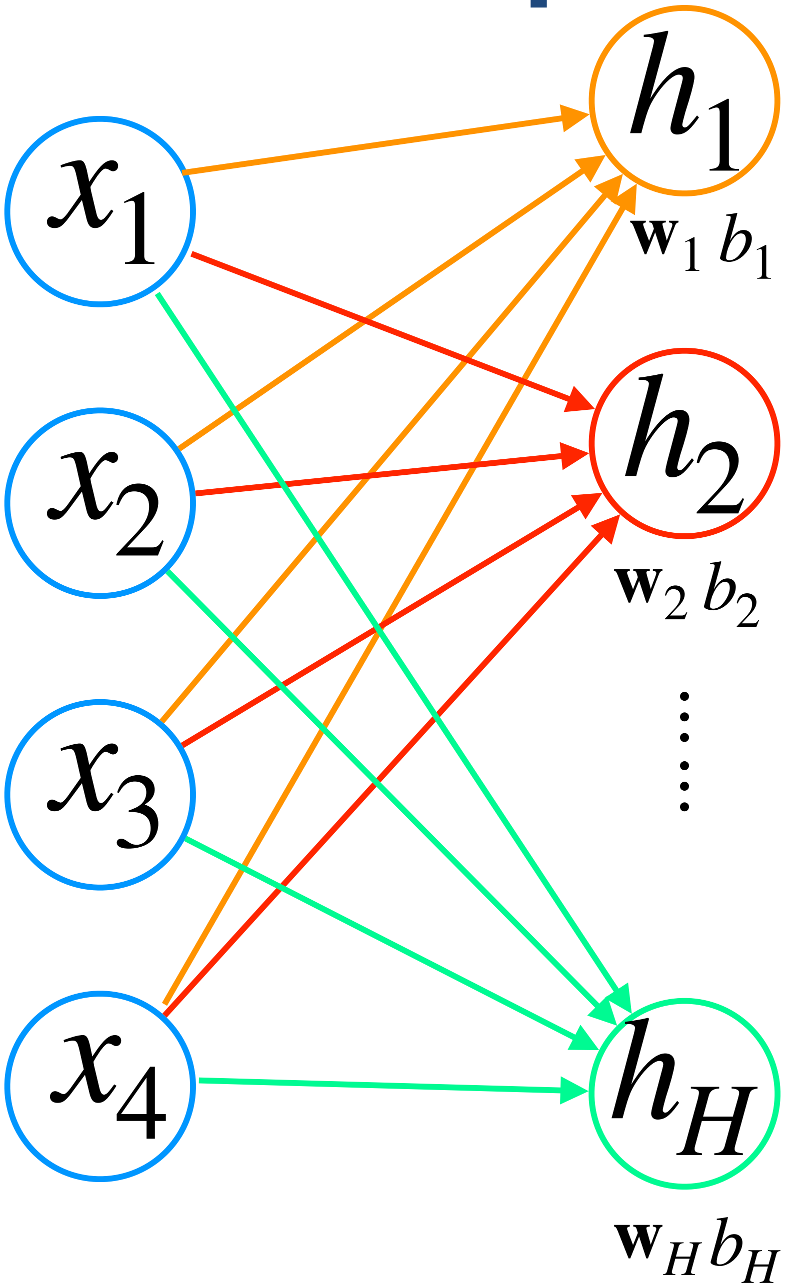
$$h_1 = \sigma(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$$
$$\mathbf{w}_1 = \left[w_{11}, w_{12}, w_{13}, w_{14}\right]^T$$

$$h_2 = \sigma(\mathbf{w}_2 \cdot \mathbf{x} + b_2)$$
$$\mathbf{w}_2 = \left[w_{21}, w_{22}, w_{23}, w_{24}\right]^T$$

$$\vdots$$

$$h_H = \sigma(\mathbf{w}_H \cdot \mathbf{x} + b_H)$$
$$\mathbf{w}_H = \left[w_{H1}, w_{H2}, w_{H3}, w_{H4}\right]^T$$

Repeating the Process

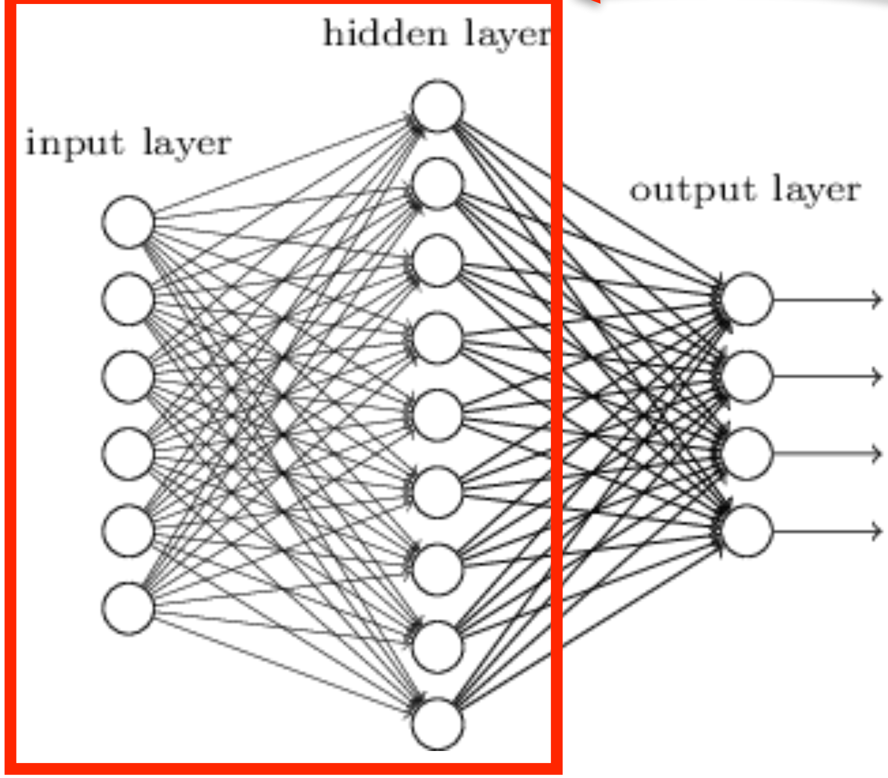


$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \; ,$$

$$\text{with } \mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_H \end{bmatrix}$$

$$\text{and } \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_H \end{bmatrix} .$$

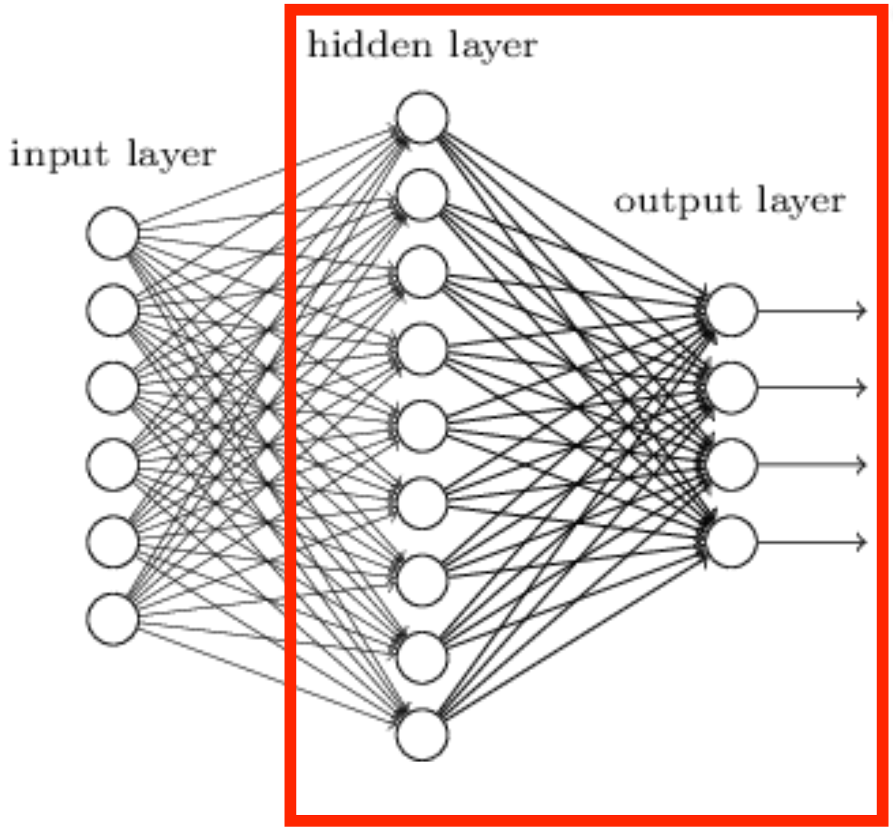
Multi-Layer Perceptron



$$\boxed{\mathbf{h} = \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)}$$
$$\mathbf{y} = \sigma_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

- The process can be repeated several times to create a vector **h**.

Multi-Layer Perceptron



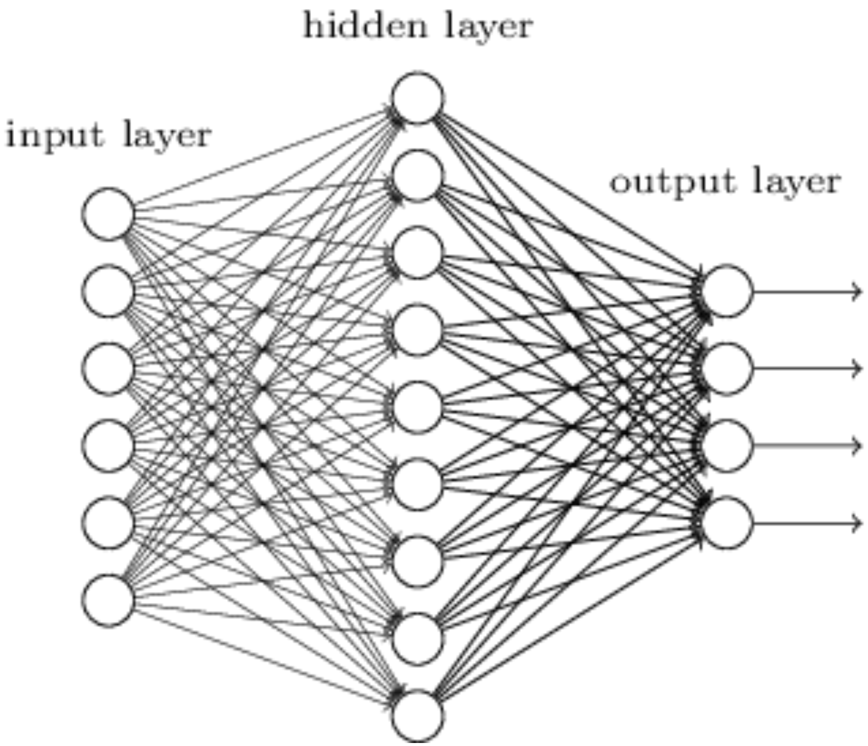
$$\mathbf{h} = \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\boxed{\mathbf{y} = \sigma_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)}$$

- The process can be repeated several times to create a vector **h**.
- It can then be done again to produce an output **y**.

—> This output is a **differentiable** function of the weights.

ReLU



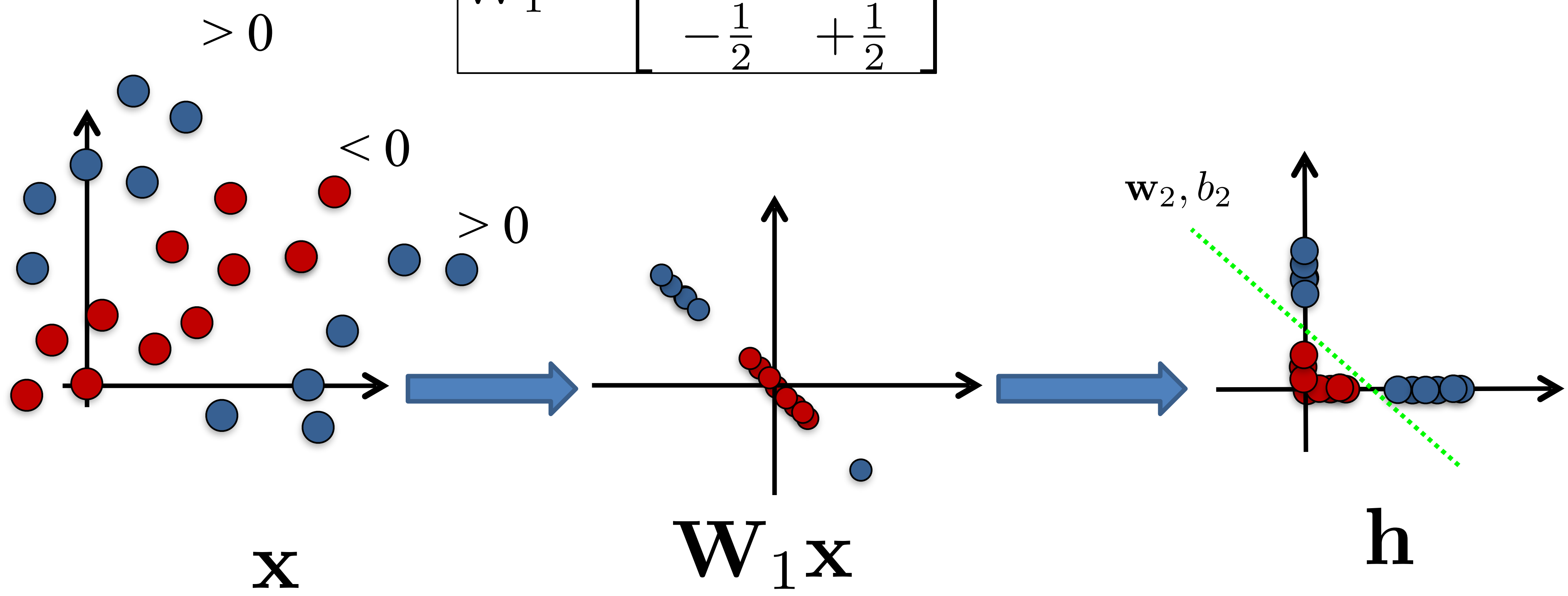
$$
\begin{aligned}
\mathbf{h} &= \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\
\mathbf{y} &= \sigma_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)
\end{aligned}
$$

$$\sigma(\mathbf{x}) = \max(0,\mathbf{x})$$

- Each node defines a hyperplane.
- The resulting function is piecewise linear affine and continuous.

ReLu Behavior

$$\mathbf{W}_1 = \begin{bmatrix} +\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & +\frac{1}{2} \end{bmatrix}$$



$$\mathbf{h} = \mathrm{ReLu}(\mathbf{W}_1\mathbf{x})$$

$$\mathbf{y} = \mathbf{w}_2^T\mathbf{h} + b_2$$

EPFL

Binary Case

- Let the training set be $\{(\mathbf{x}_n, t_n)_{1 \le n \le N}\}$ where $t_n \in \{0,1\}$ is the class label and let us consider a neural net with a 1D output.

- We write

In this case w2 is vector.

$$y_n = \sigma(\mathbf{w}_2(\sigma(\mathbf{W}_1 \mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2) \in [0.1]$$

- We want to minimize the binary cross entropy

$$E(\mathbf{W}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2) = \frac{1}{N} \sum_{n=1}^{N} E_n(\mathbf{W}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2) \; ,$$

$$E_n(\mathbf{W}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2) = -\,(t_n \ln(y_n) + (1 - t_n)\ln(1 - y_n)) \; ,$$

  with respect to the coefficients of $\mathbf{W}_1$, $\mathbf{w}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$.

- E can be minimized using a gradient-based technique.

Multi-Class Case

Let the training set be $\{(\mathbf{x}_n, [t_n^1, \ldots, t_n^K])_{1 \leq n \leq N}\}$ where $t_n^k \in \{0,1\}$ is the probability that sample $\mathbf{x}_n$ belongs to class k.

- We write

$$\mathbf{y}_n = \sigma(\mathbf{W}_2(\sigma(\mathbf{W}_1\mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2) \in R^K$$

$$p_n^k = \frac{\exp(\mathbf{y}_n[k])}{\sum_j \exp(\mathbf{y}_n[j])}$$

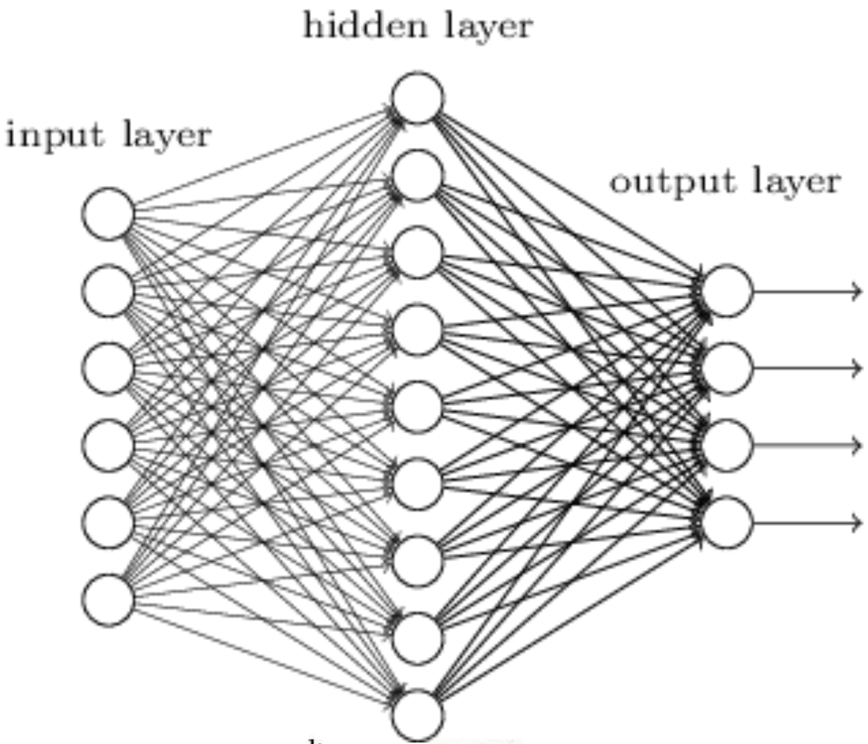
- We minimize the cross entropy

$$E(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = \frac{1}{N}\sum_{n=1}^{N} E_n(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) \ ,$$

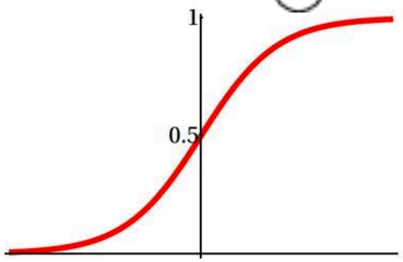
$$E_n(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2) = -\sum t_n^k \ln(p_n^k) \ ,$$

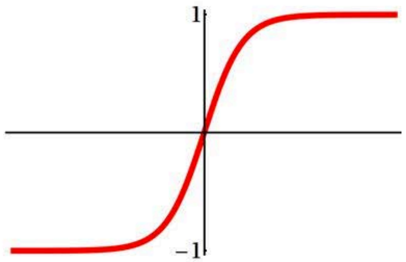
with respect to the coefficients of $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$.

Sigmoid and Tanh



$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y} = \sigma(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$



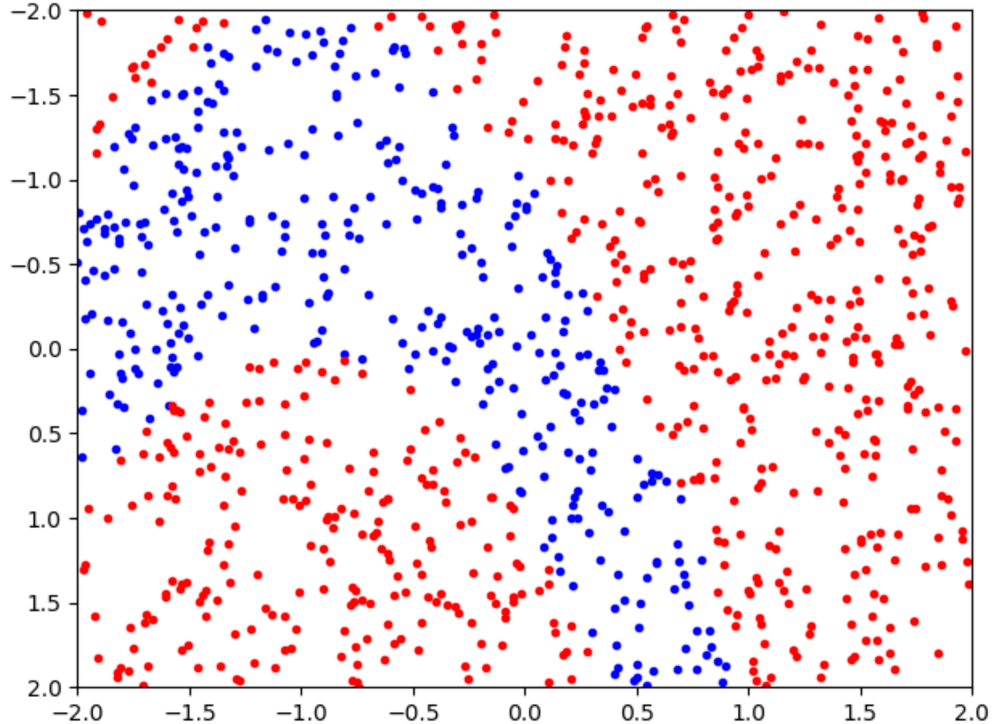


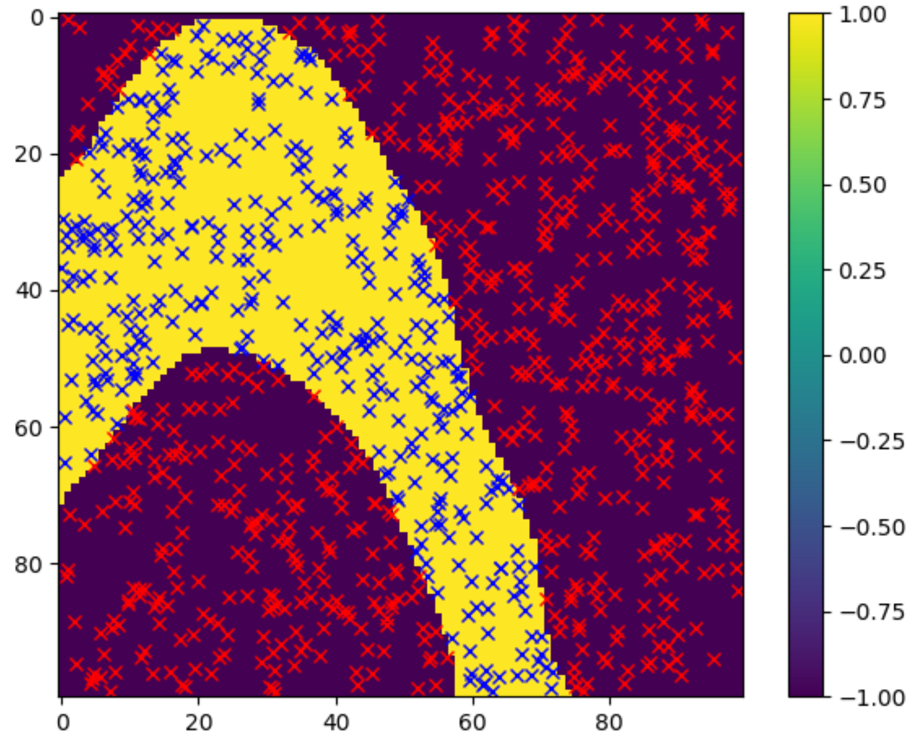
$$\text{sigm:} \quad \sigma(x) = \frac{1}{1+\exp(-x)}$$

$$\text{tanh:} \quad \sigma(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$

- Each node defines a hyperplane.
- The resulting function is continuously differentiable.

Non-Linear Binary Classification





Positive: $100(x_2 - x_1^2)^2 + (1 - x_1)^2 < 0.5$
Negative: Otherwise

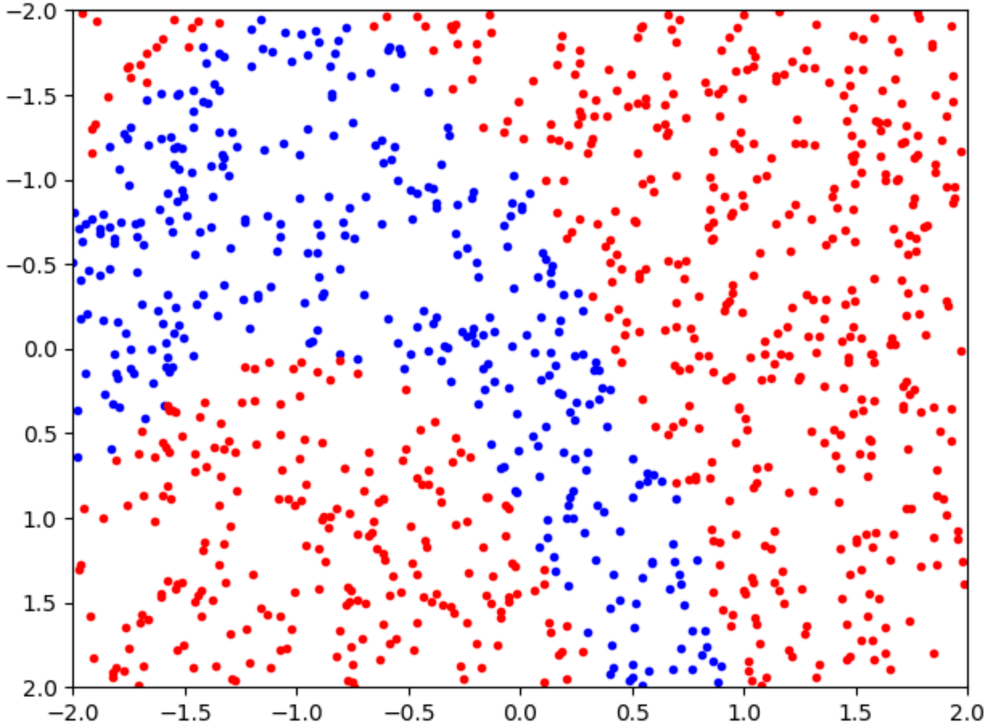
$y(\mathbf{x}; \tilde{\mathbf{w}})$ is now a non-linear function implemented by the network.

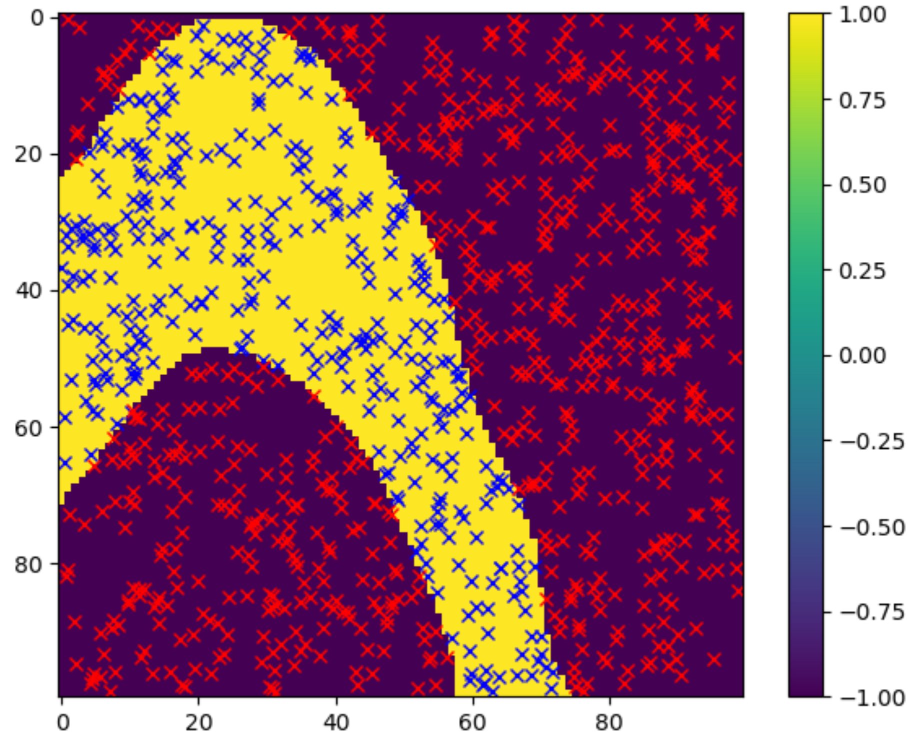
Problem statement: Find $\tilde{\mathbf{w}}$ such that

- for all or most positive samples $y(\tilde{\mathbf{x}}; \tilde{\mathbf{w}}) > 0.5$,
- for all or most negative samples $y(\tilde{\mathbf{x}}; \tilde{\mathbf{w}}) < 0.5$.

EPFL

Regression





Positive: $100(x_2 - x_1^2)^2 + (1 - x_1)^2 < 0.5$

Negative: Otherwise

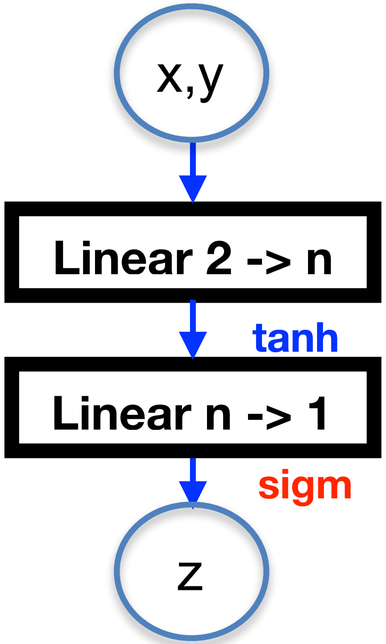
$y(\mathbf{x}; \tilde{\mathbf{w}})$ is now a non-linear function implemented by the network.

Problem statement: Find $\tilde{\mathbf{w}}$ such that

$$y(\mathbf{x}; \tilde{\mathbf{w}}) \approx 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

EPFL

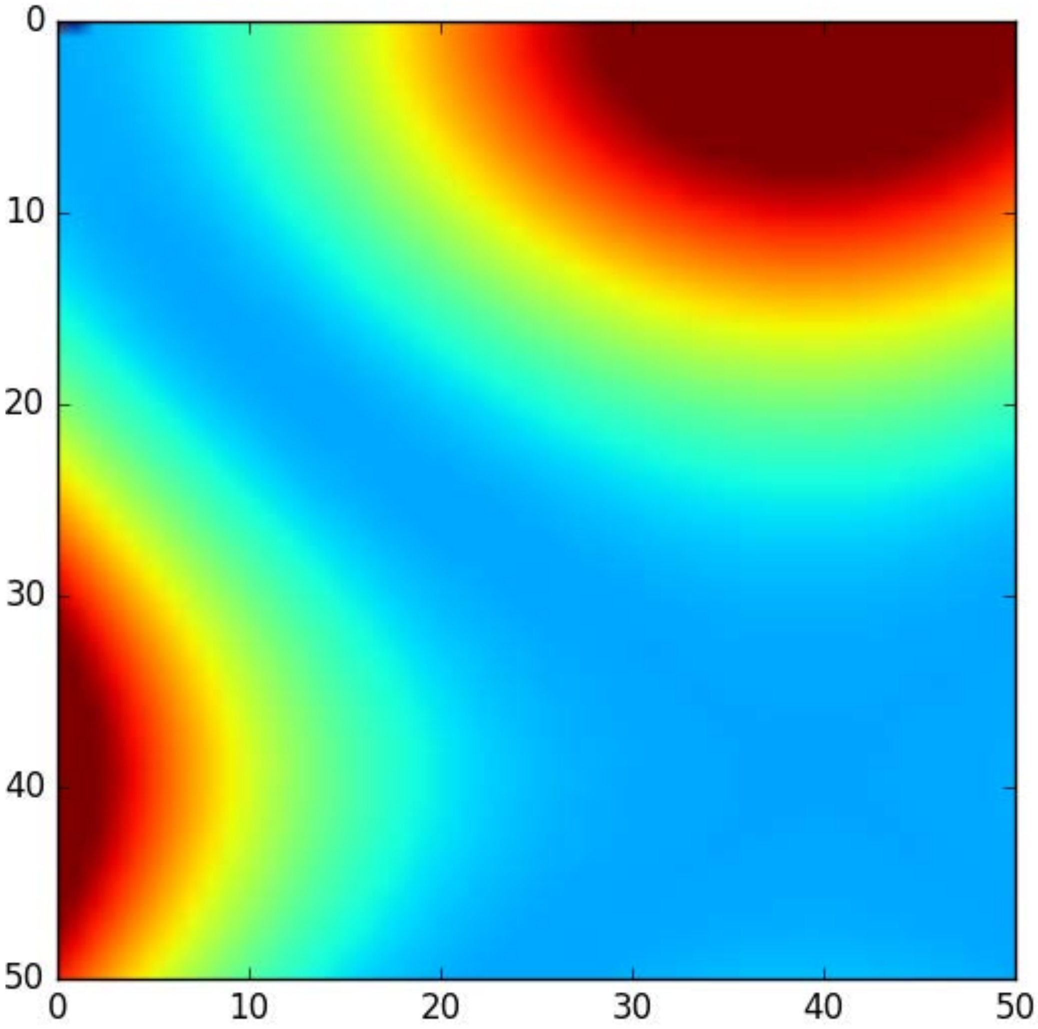
From Classification to Regression



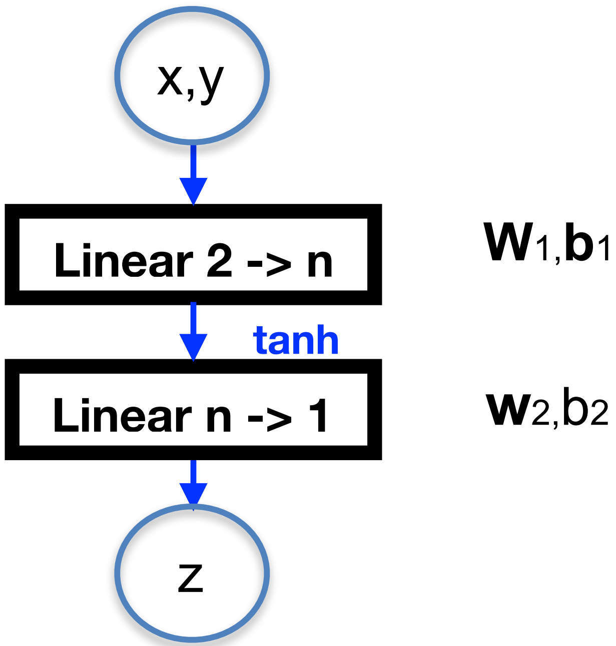
$$\text{Minimize } \sum_i [t_i \log(\mathrm{sigm}(f(x_i, y_i))) + (1 - t_i)\log(1 - \mathrm{sigm}(f(x_i, y_i)))$$
$$\text{with respect to } \mathbf{W}_1, \mathbf{w}_2, b_x, b_y, b_z.$$

$$\text{Minimize } \sum_i (z_i - f(x_i, y_i))^2,$$
$$\text{with respect to } \mathbf{W}_1, \mathbf{w}_2, b_x, b_y, b_z.$$

Approximating a Surface



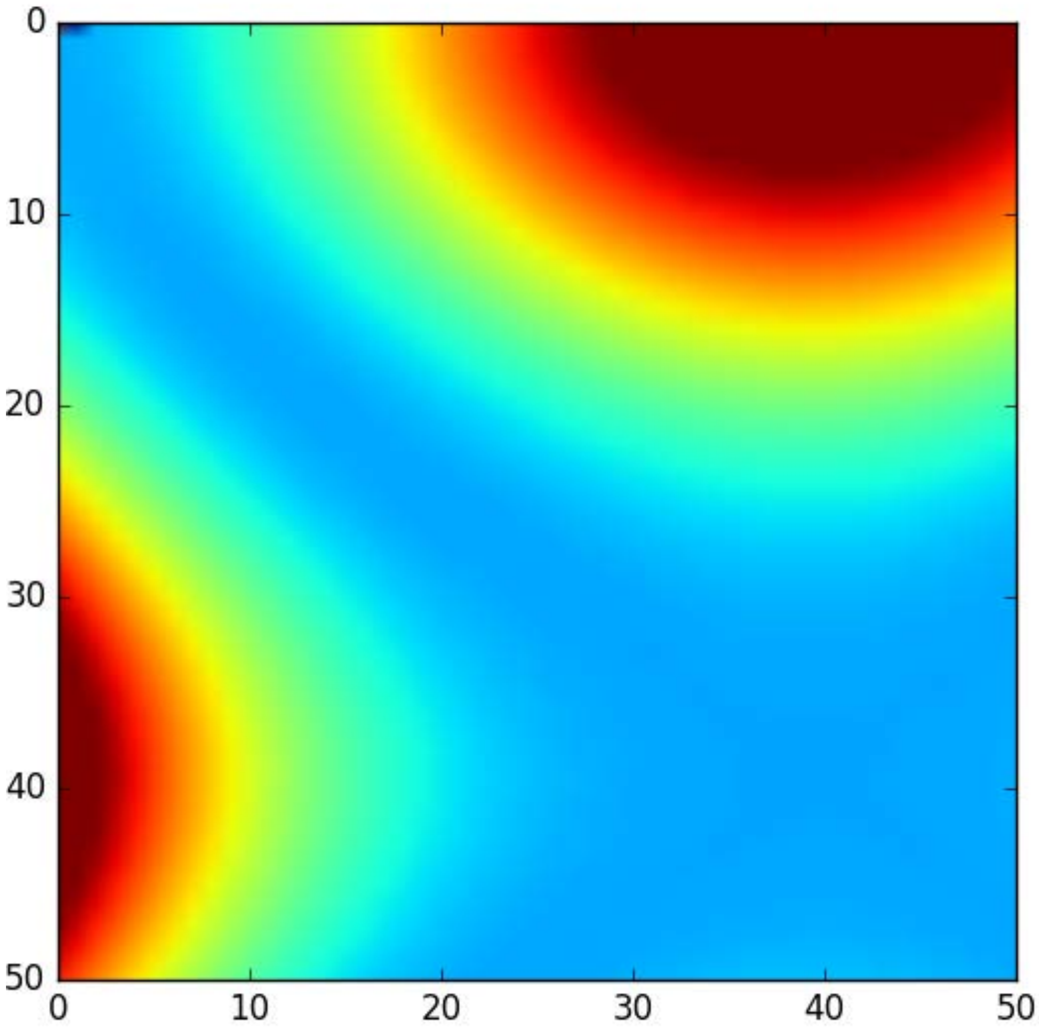
$$z = 100 * (y - x^2)^2 + (1 - x)^2$$



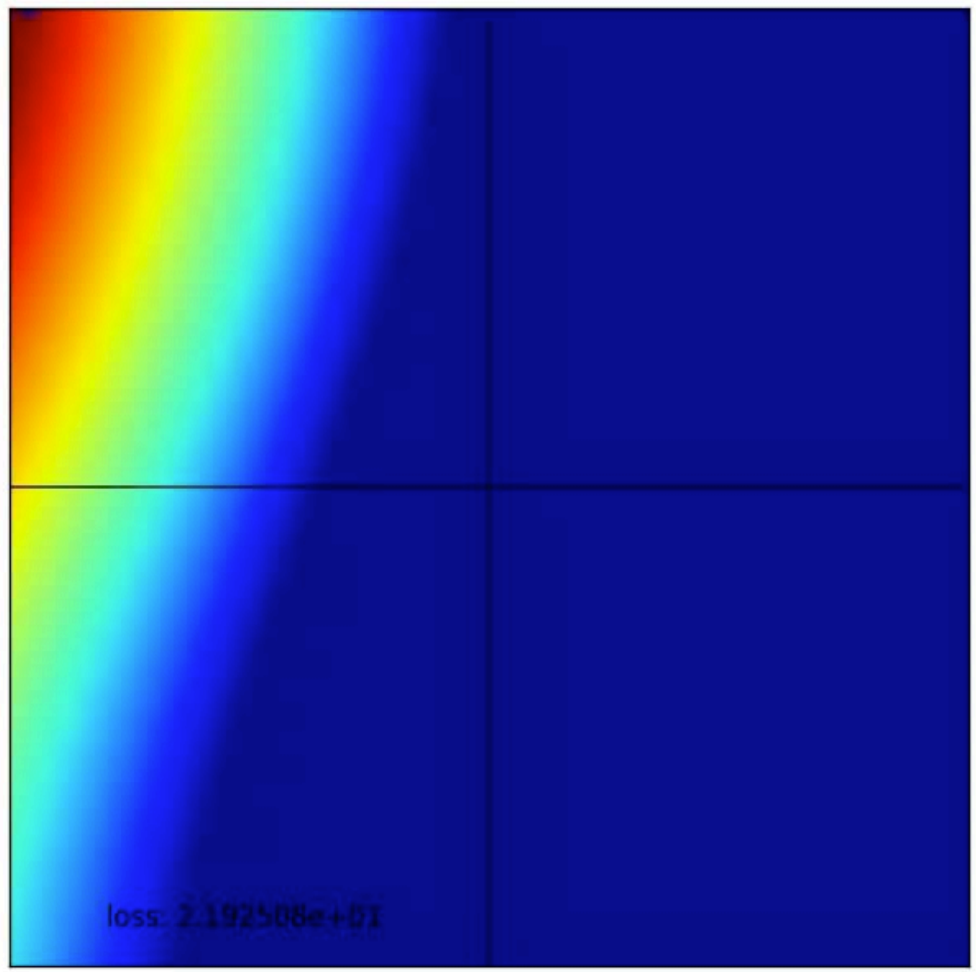
$$\begin{aligned} z &= f(x, y) \\ &= \mathbf{w}_2 \sigma(\mathbf{W}_1 \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{b}_1) + b_2 \end{aligned}$$

Minimize $\sum_i (z_i - f(x_i, y_i))^2$,
with respect to $\mathbf{W}_1, \mathbf{w}_2, b_x, b_y, b_z$.

Interpolating a Surface

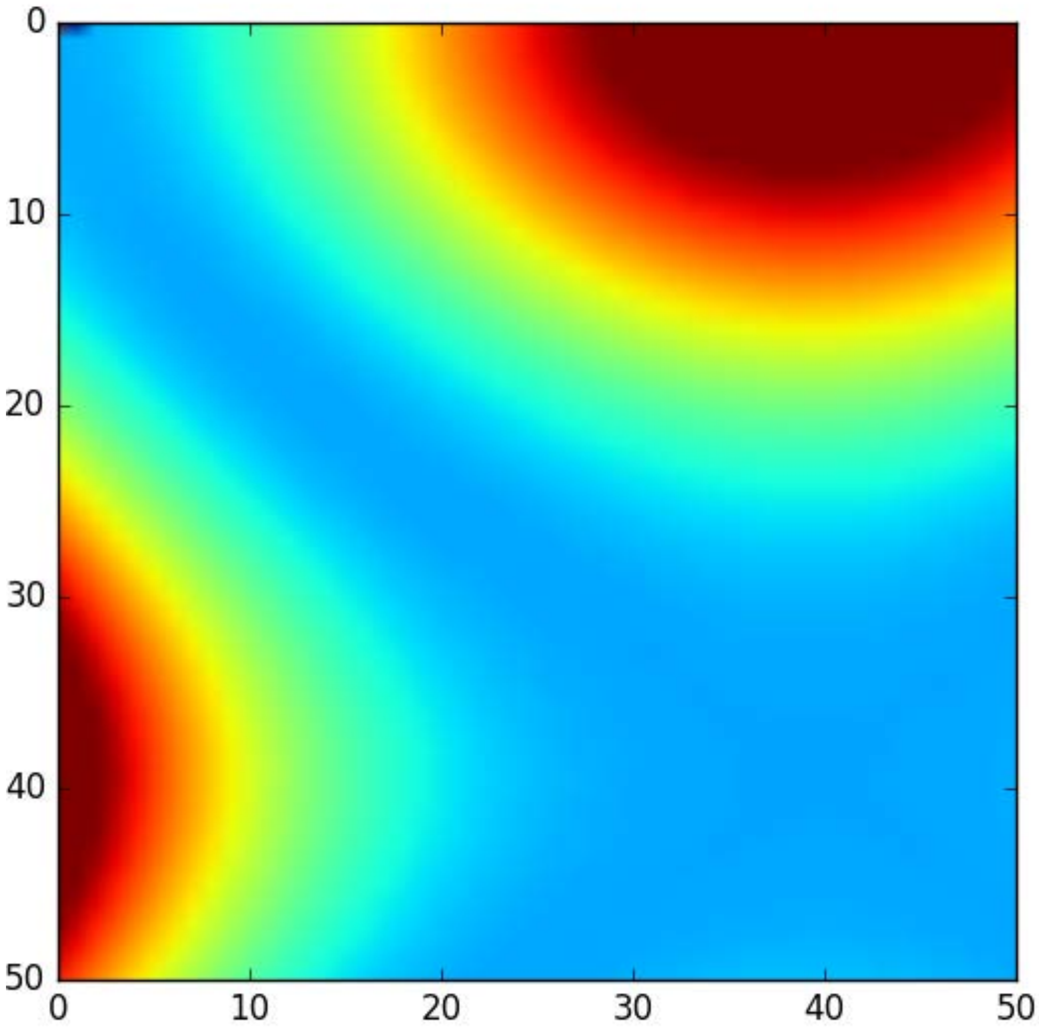


$$z = 100 * (y - x^2)^2 + (1 - x)^2$$

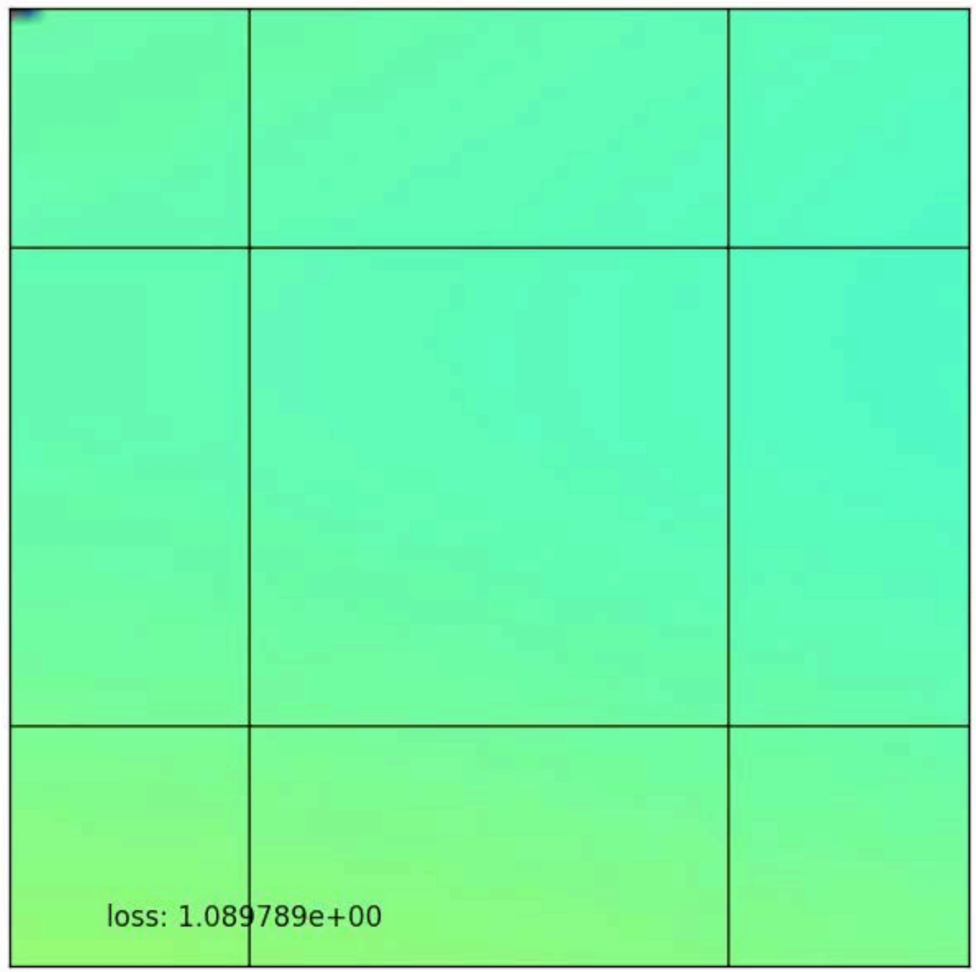


3-node hidden layer

Interpolating a Surface

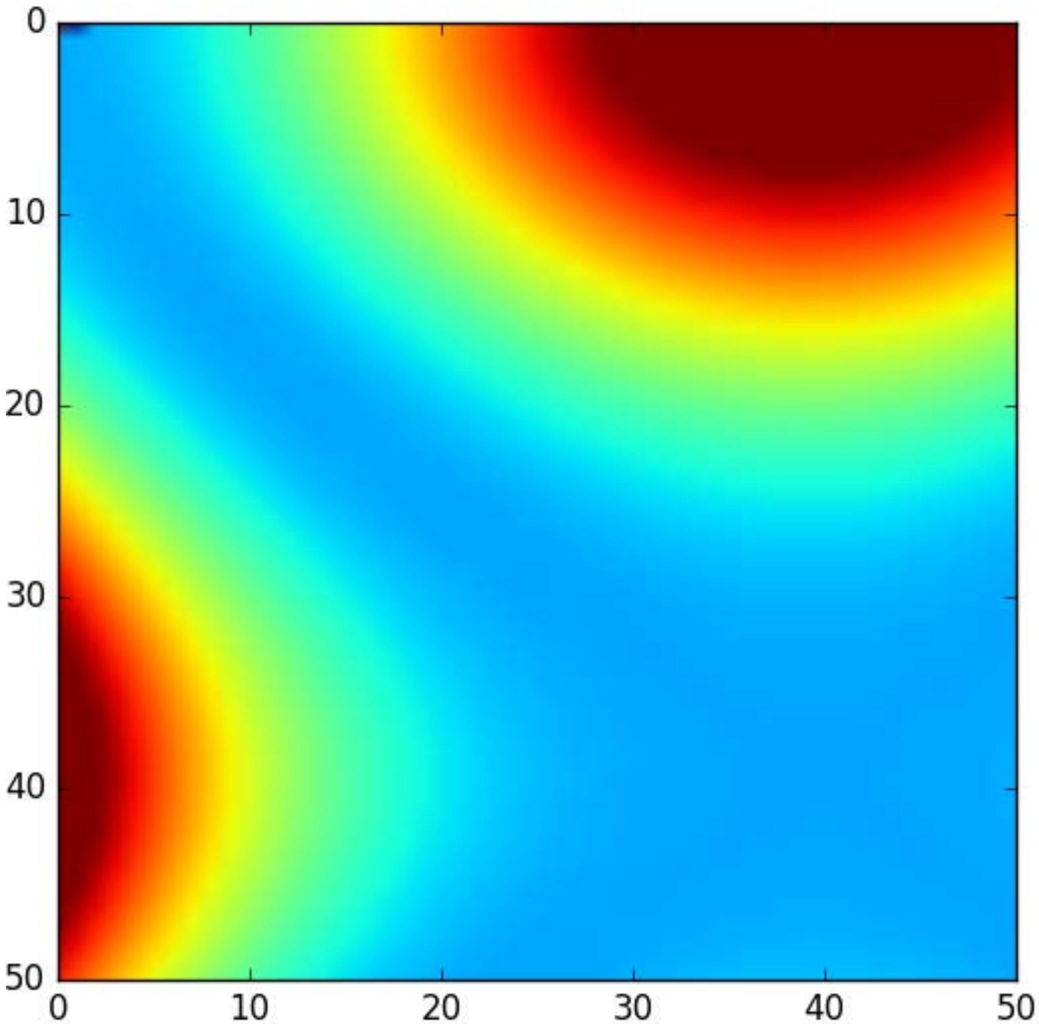


$$z = 100 * (y - x^2)^2 + (1 - x)^2$$

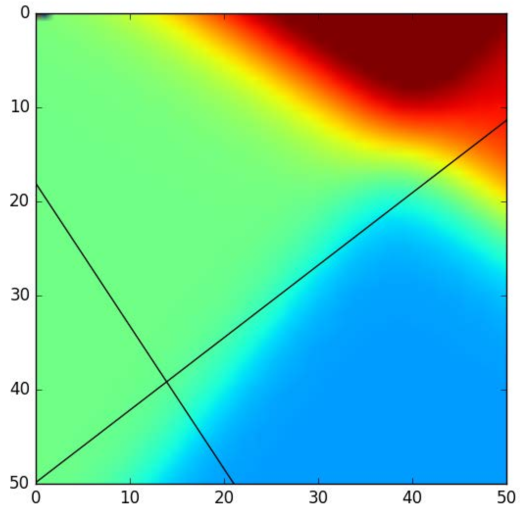


4-node hidden layer

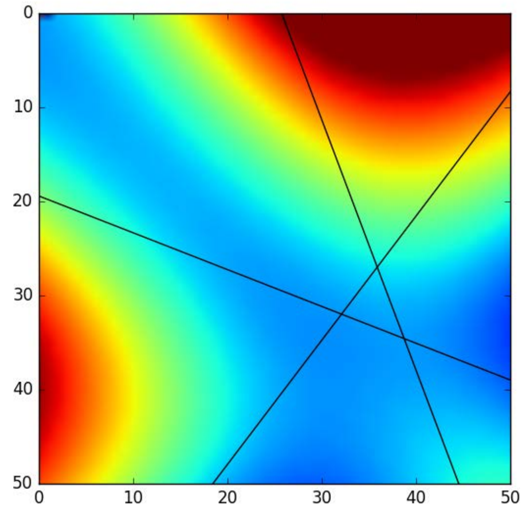
Adding more Nodes



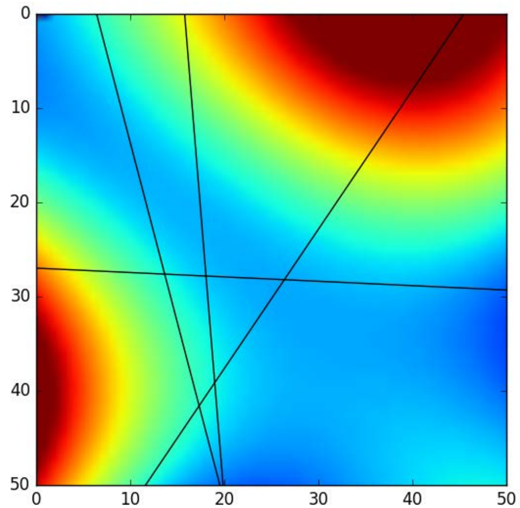
$$z = 100 * (y - x^2)^2 + (1 - x)^2$$



2 nodes -> loss 3.02e-01

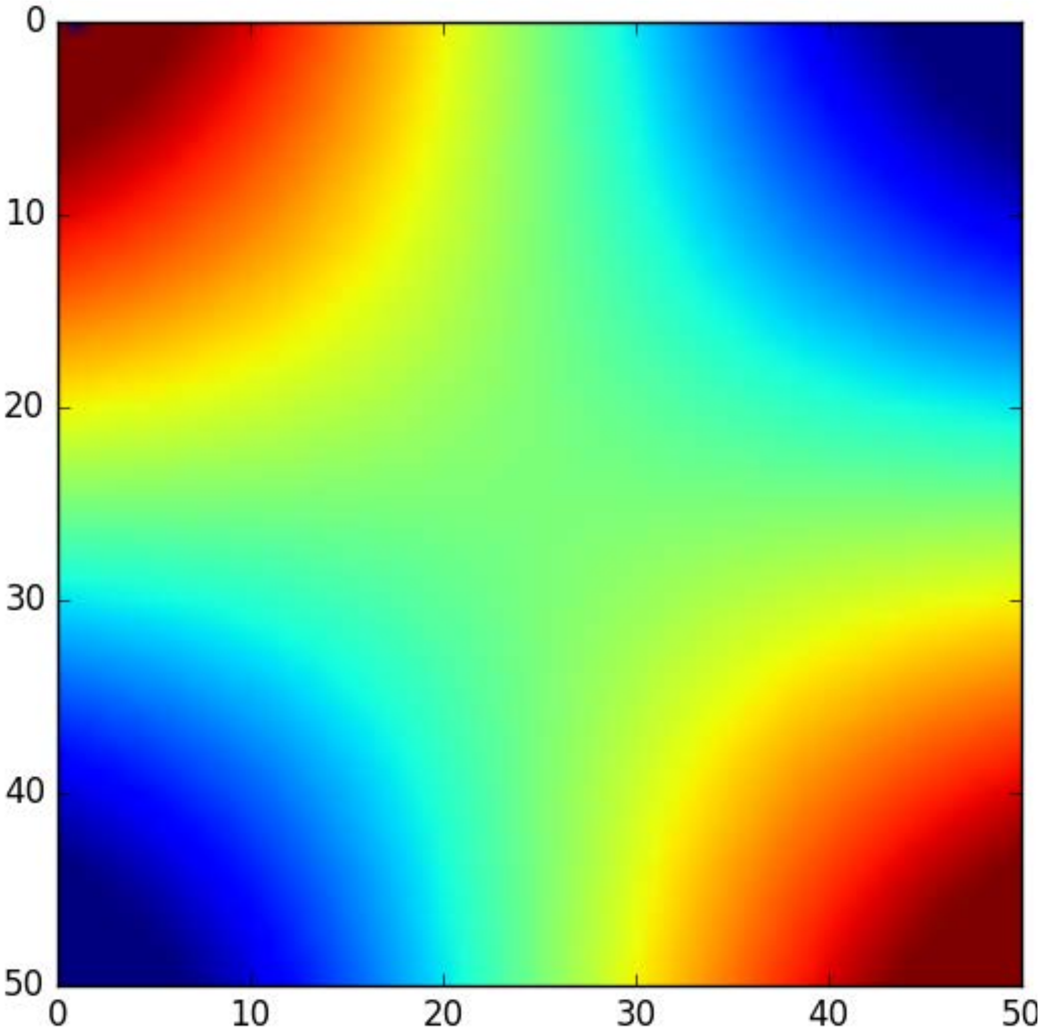


3 nodes -> loss 2.08e-02

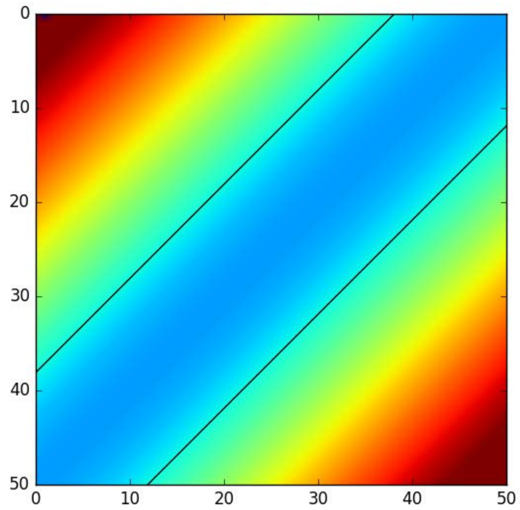


4 nodes -> loss 8.27e-03

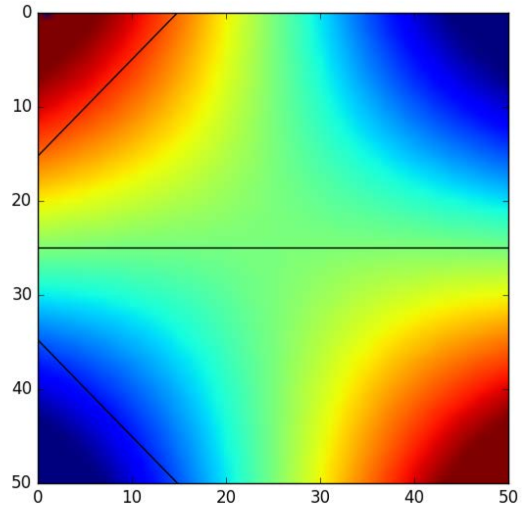
Adding more Nodes



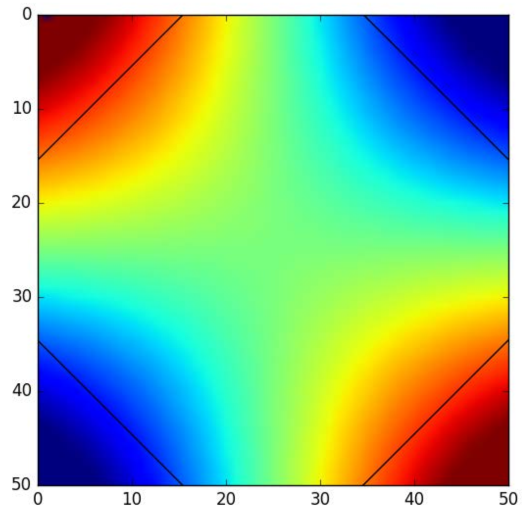
$$z = sin(x)sin(y)$$



2 nodes -> loss 2.61e-01

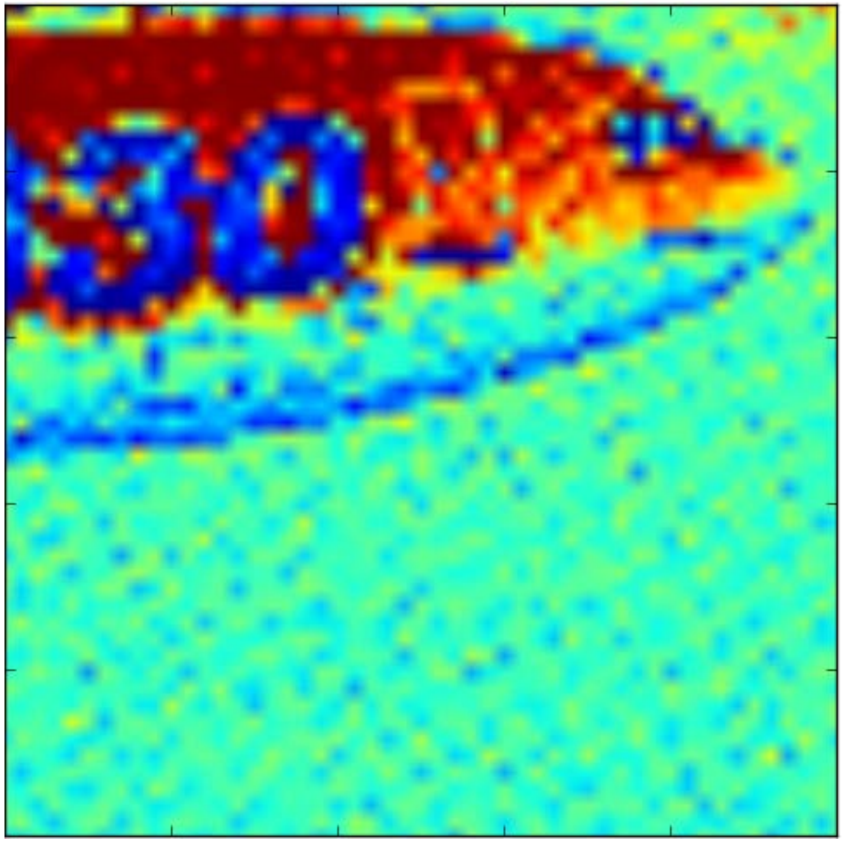


3 nodes -> loss 2.51e-04

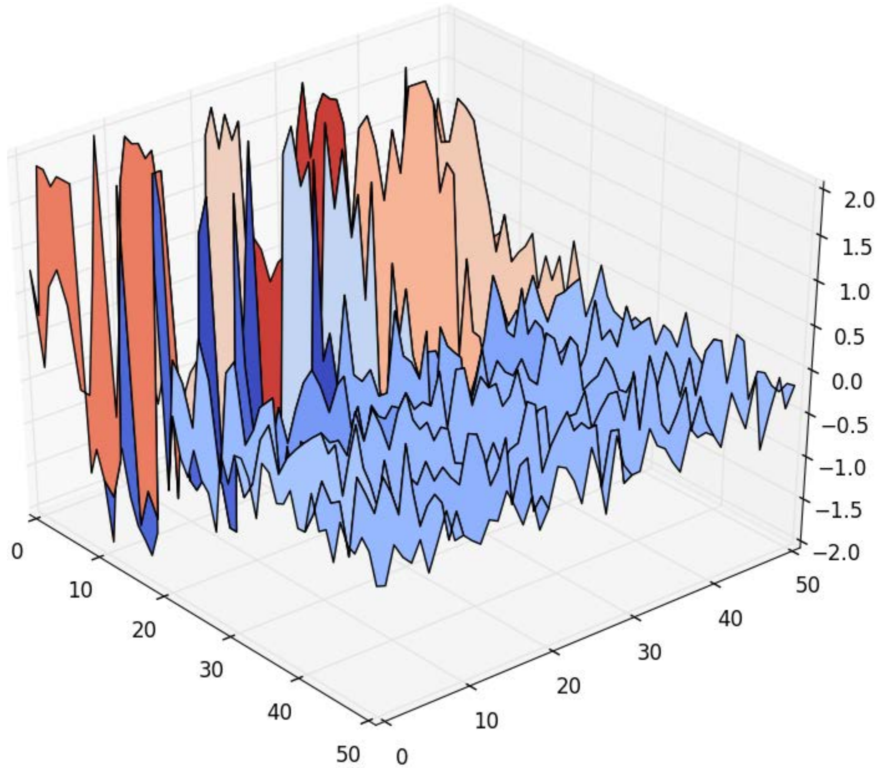


4 nodes -> loss 3.07e-07

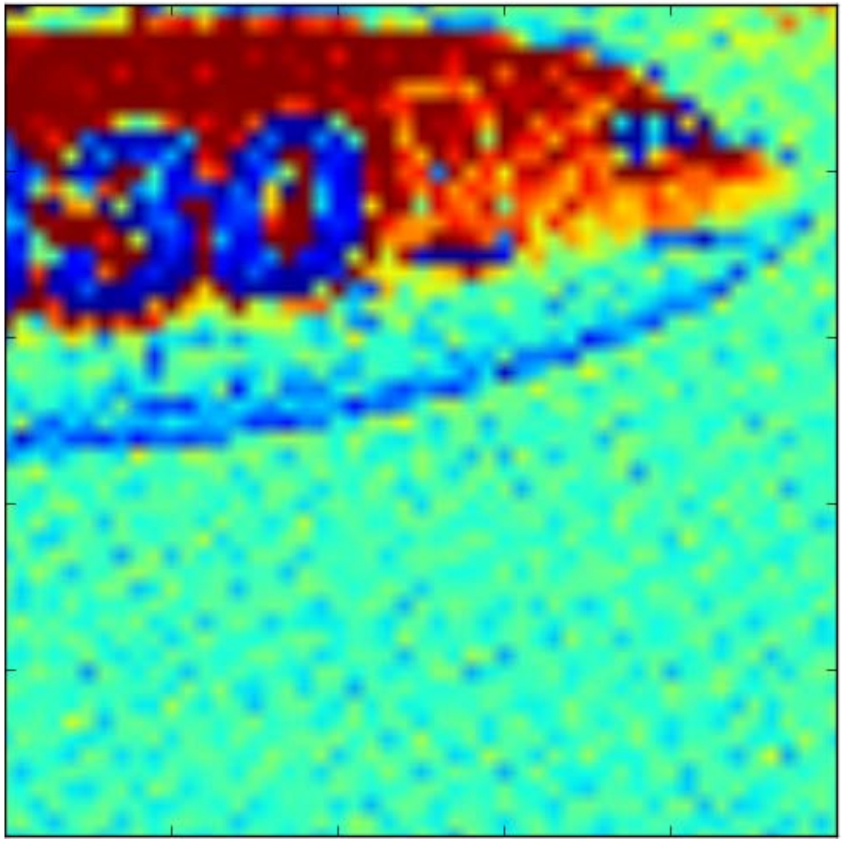
More Complex Surface



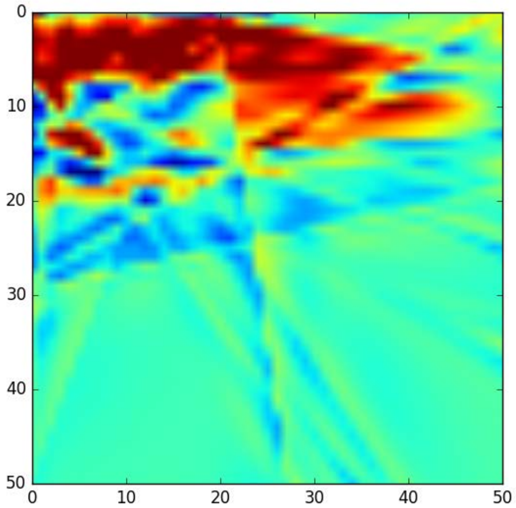
$$I = f(x, y)$$



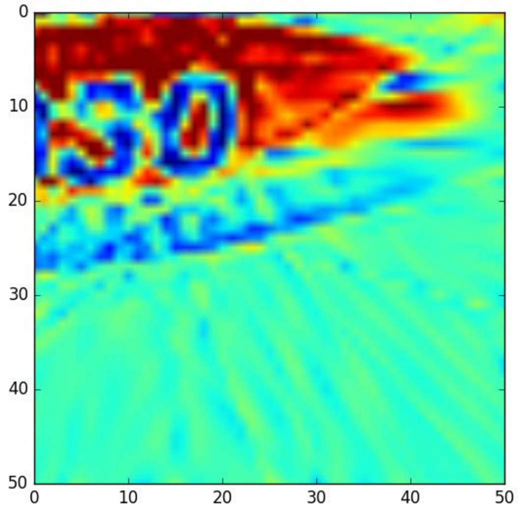
More Complex Surface



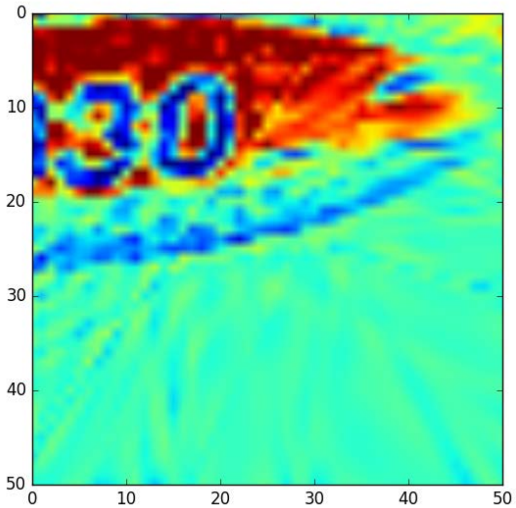
$$I = f(x, y)$$



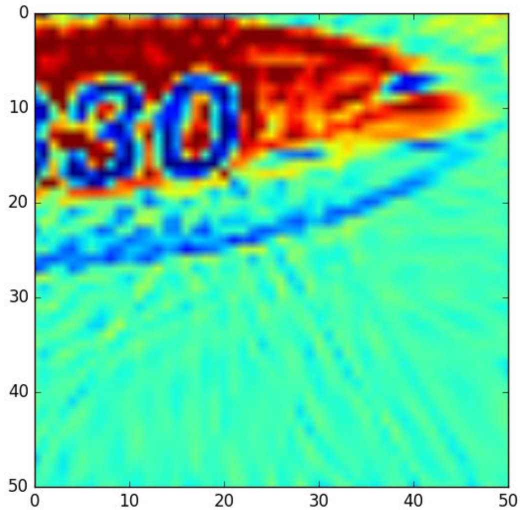
50 nodes -> loss 3.65e-01



100 nodes -> loss 2.50e-01



125 nodes -> loss 2.40e-01



300 nodes -> loss 1.92e-01

Universal Approximation Theorem

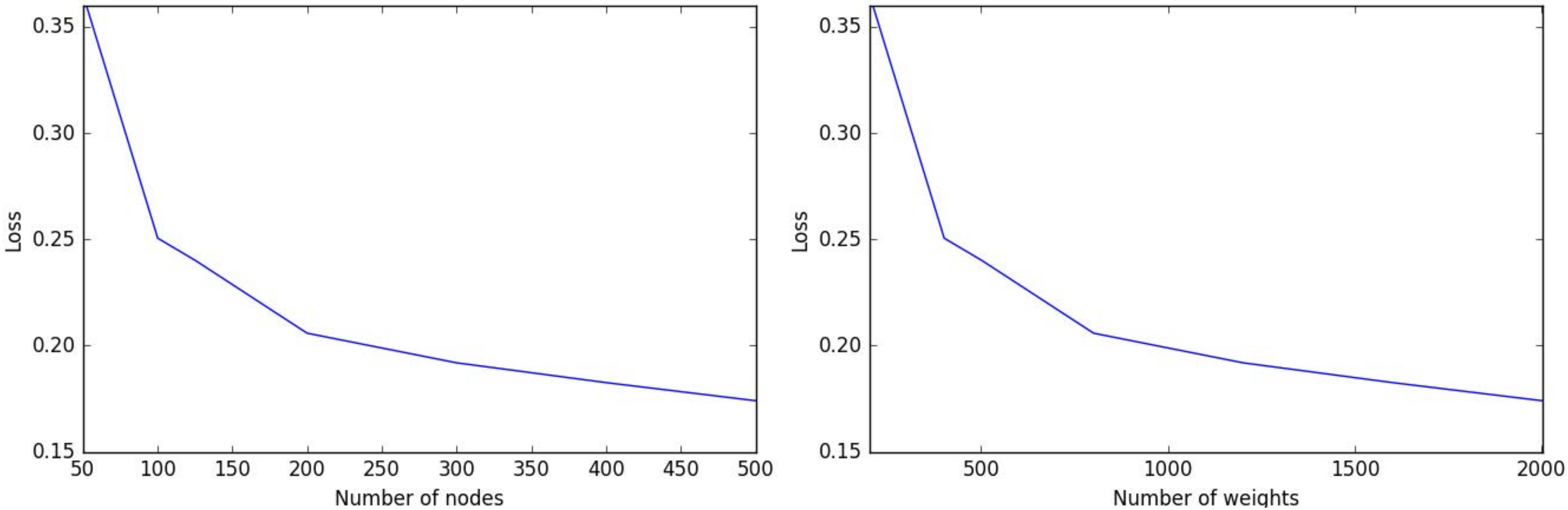
A feedforward network with a linear output layer and at least one hidden layer with any 'squashing' activation function (e.g. logistic sigmoid) can approximate any Borel measurable function (from one finite-dimensional space to another) with any desired nonzero error.

Any continuous function on a closed and bounded set of R^n is Borel-measurable.

—> In theory, any reasonable function can be approximated by a one-hidden layer network as long as it is continuous.

[Hornik et al, 1989; Cybenko, 1989]

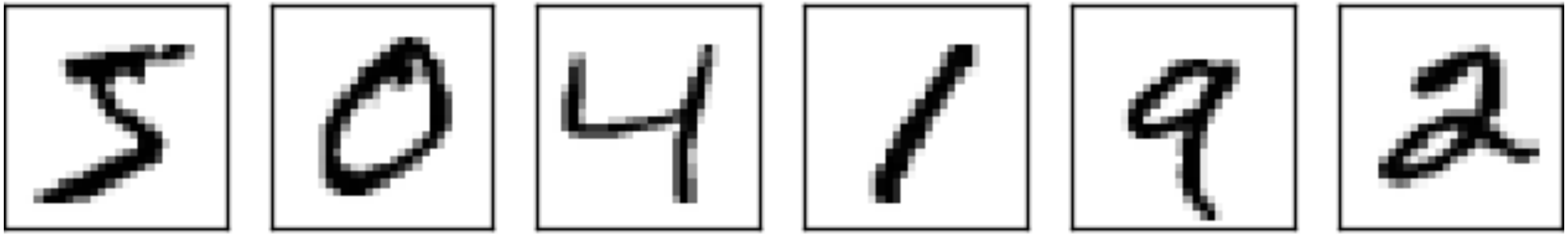
In Practice



- It may take an exponentially large number of parameters for a good approximation.
- The optimization problem becomes increasingly difficult.

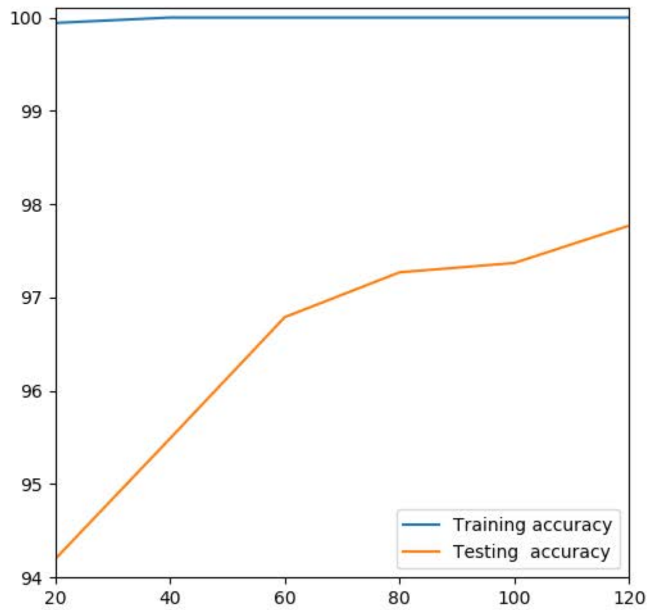
—> The one hidden layer perceptron may not converge to the best solution!

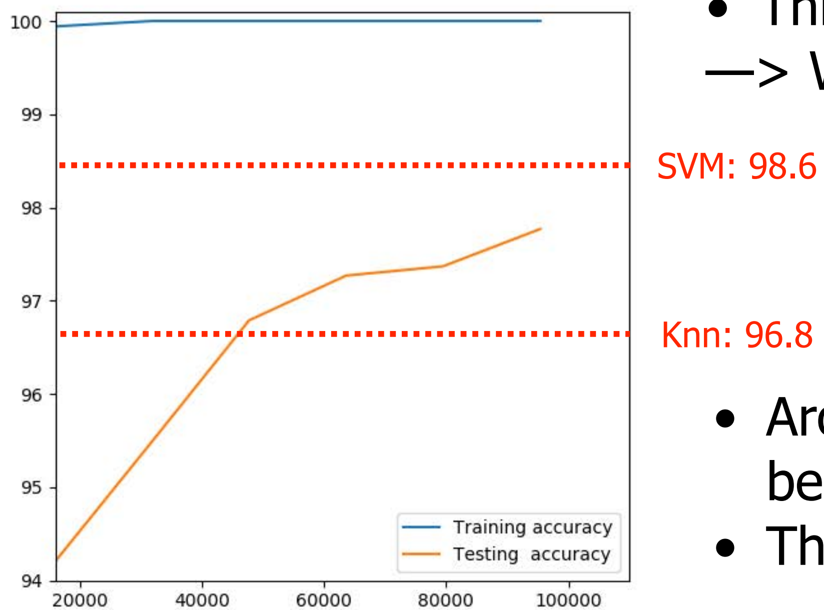
MNIST



- The network takes as input 28x28 images represented as 784D vectors.
- The output is a 10D vector giving the probability of the image representing any of the 10 digits.
- There are 50’000 training pairs of images and the corresponding label, 10’000 validation pairs, and 5’000 testing pairs.

MNIST Results





nIn =784

nOut = 10

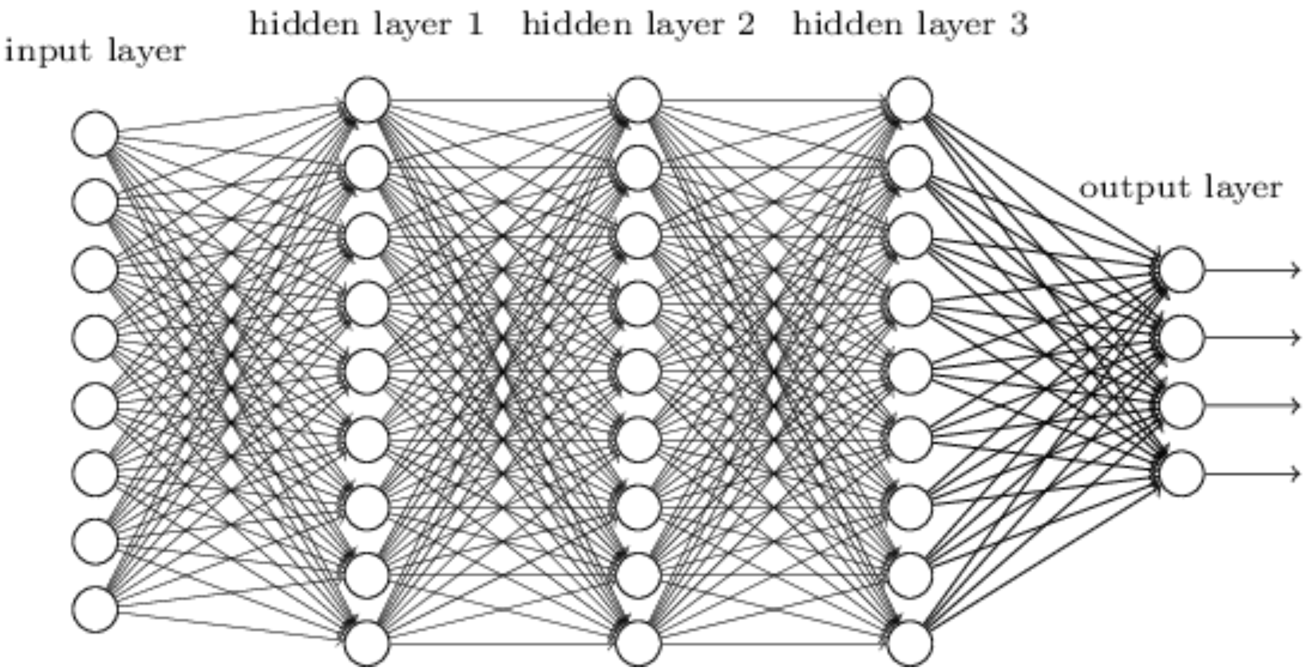
20 < hidden layer size < 120

- MLPs have **many** parameters.
- This has long been a major problem.

—> Was eventually solved by using GPUs.

- Around 2005, SVMs were often felt to be superior to neural nets.
- This is no longer the case ….

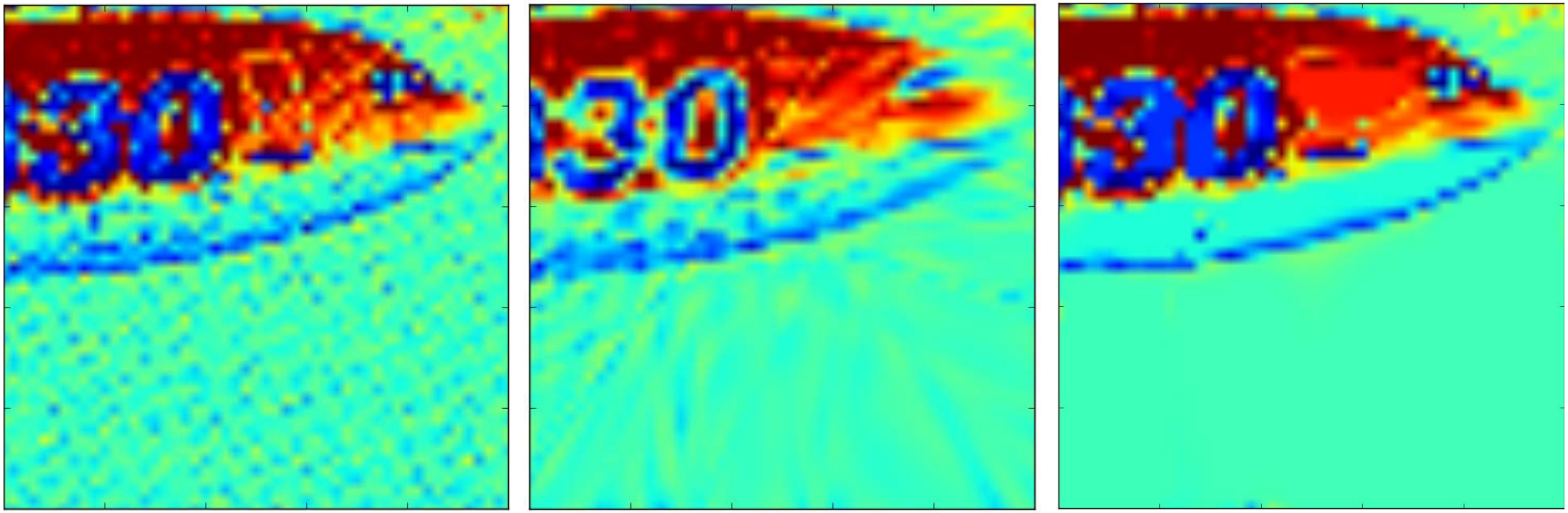
Deep Learning



$$
\begin{aligned}
\mathbf{h}_1 &= \sigma_1(\mathbf{W}_1\mathbf{x}_1 + \mathbf{b}_1) \\
\mathbf{h}_2 &= \sigma_2(\mathbf{W}_2\mathbf{h}_2 + \mathbf{b}_2) \\
&\dots \\
\mathbf{y} &= \sigma_n(\mathbf{W}_n\mathbf{h}_n + \mathbf{b}_n)
\end{aligned}
$$

- The descriptive power of the net increases with the number of layers.
- In the case of a 1D signal, it is roughly proportional to $\prod_n W_n$ where w_n is the width of layer n.

Second Layer for Approximation



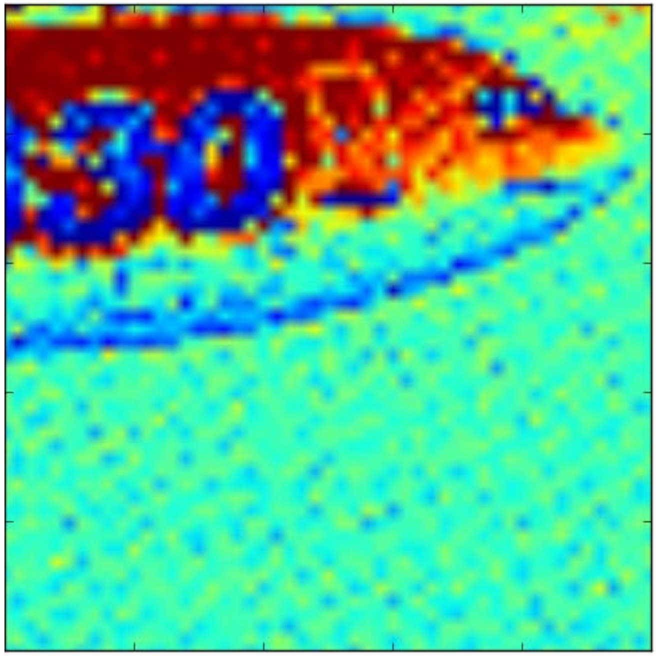
$$I = f(x, y)$$

1 Layer: 125 nodes -> loss 2.40e-01   2 Layers: 20 nodes -> loss 8.31e-02

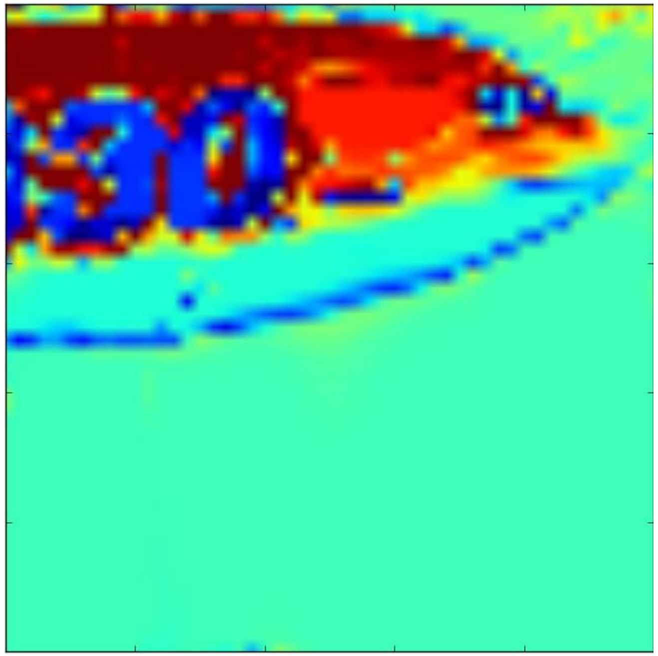
501 weights in both cases

EPFL

Adding a Third Layer

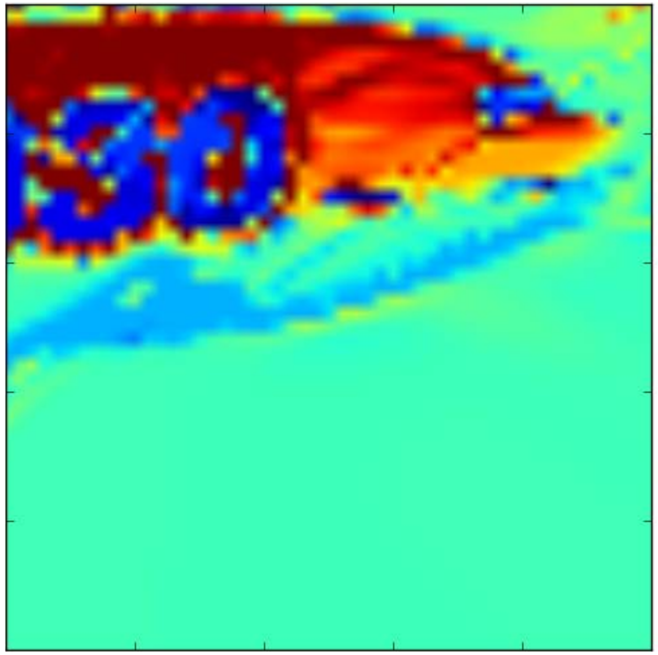


$$I = f(x, y)$$



2 Layers: 20 nodes -> loss 8.31e-02

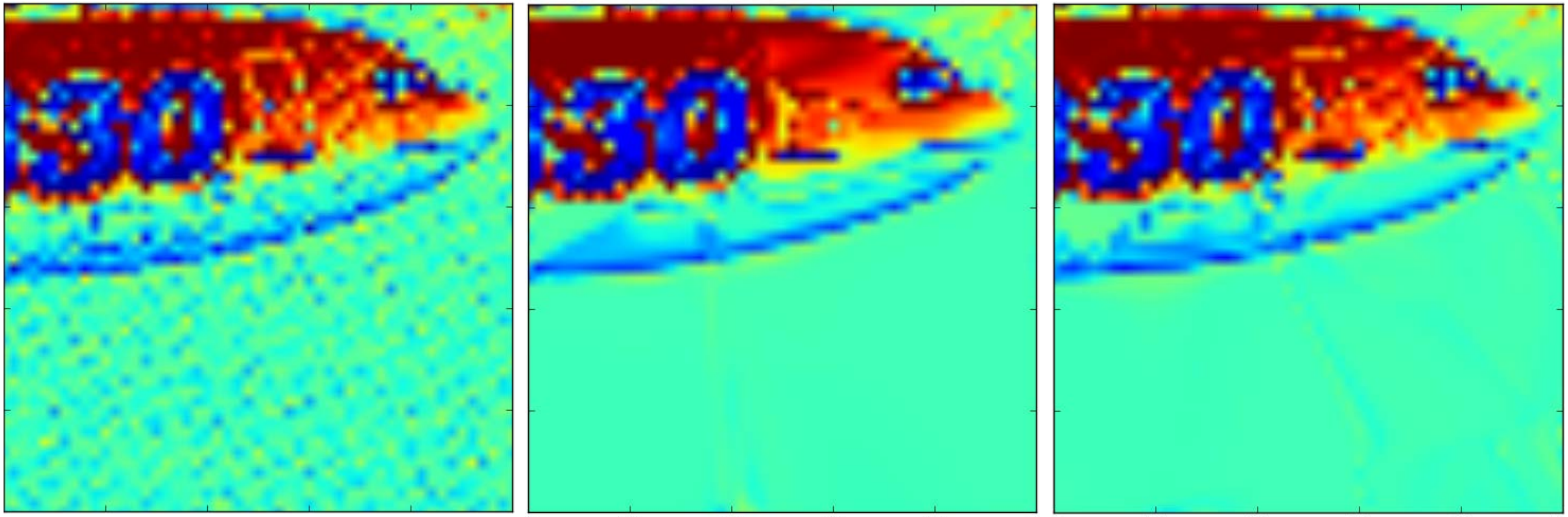
501 weights



3 Layers: 14 nodes -> loss 7.55e-02

477 weights

Adding a Third Layer



$$I = f(x, y)$$

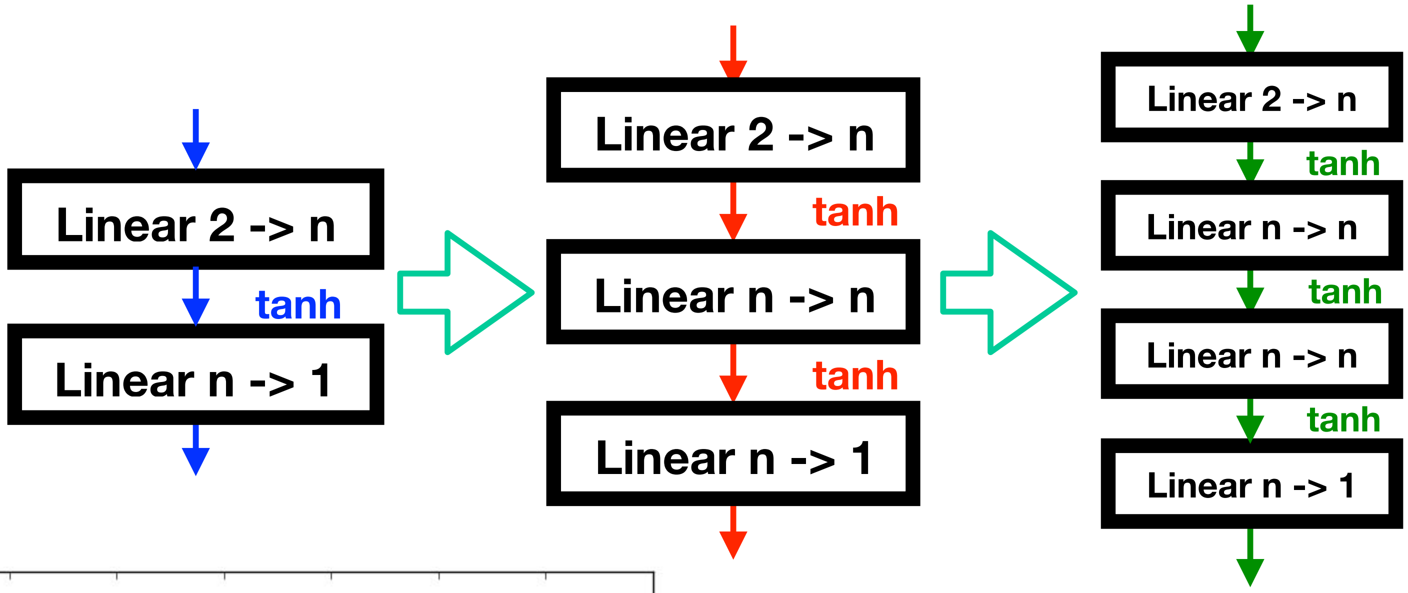
3 Layers: 15 nodes -> loss 5.93e-02

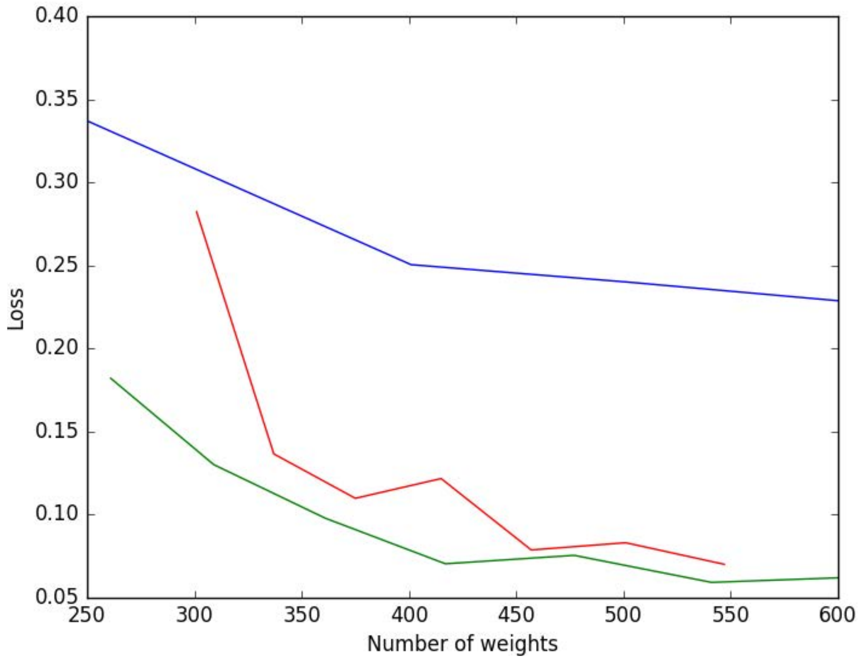
541 weights

3 Layers: 19 nodes -> loss 4.38e-02

837 weights

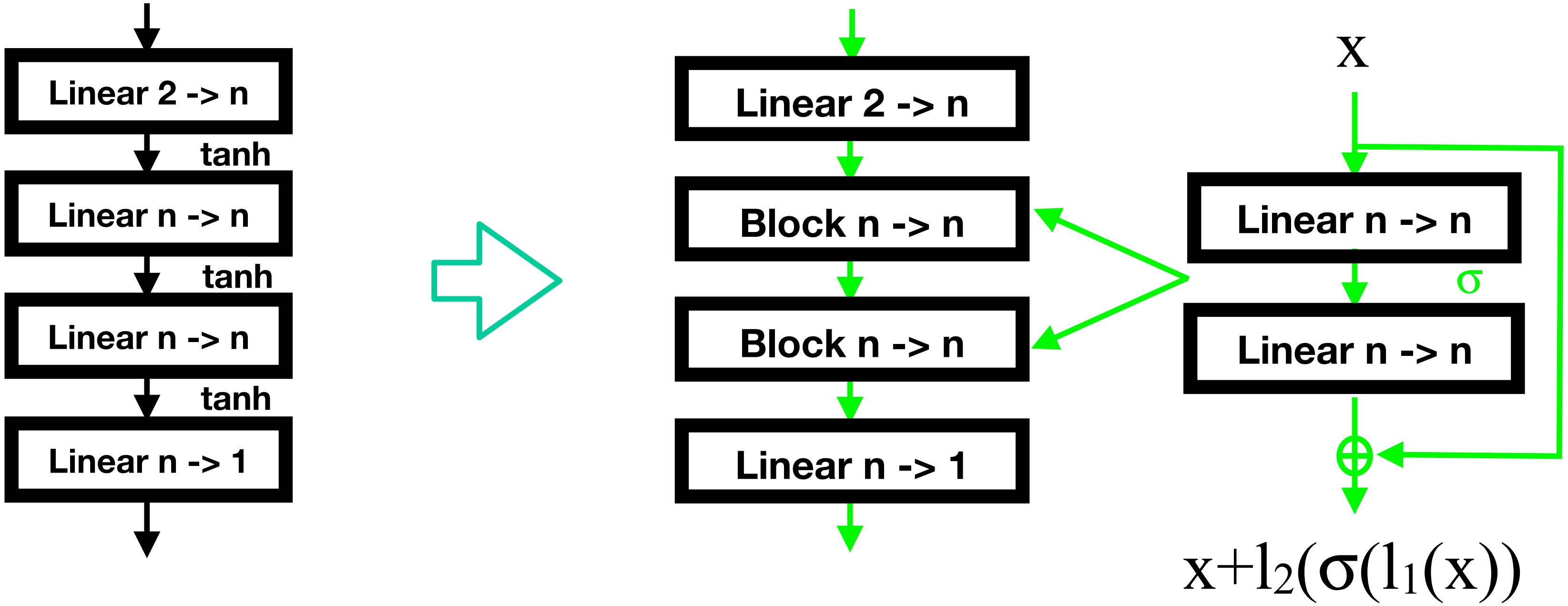
Multi Layer Perceptrons





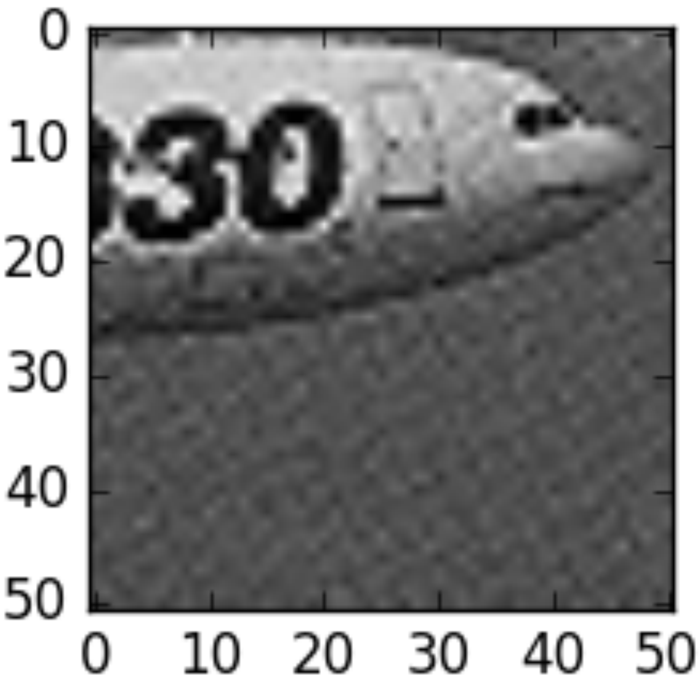
- Adding layers often yields better convergence properties.
- In current practice, deeper is usually better.

MLP to ResNet

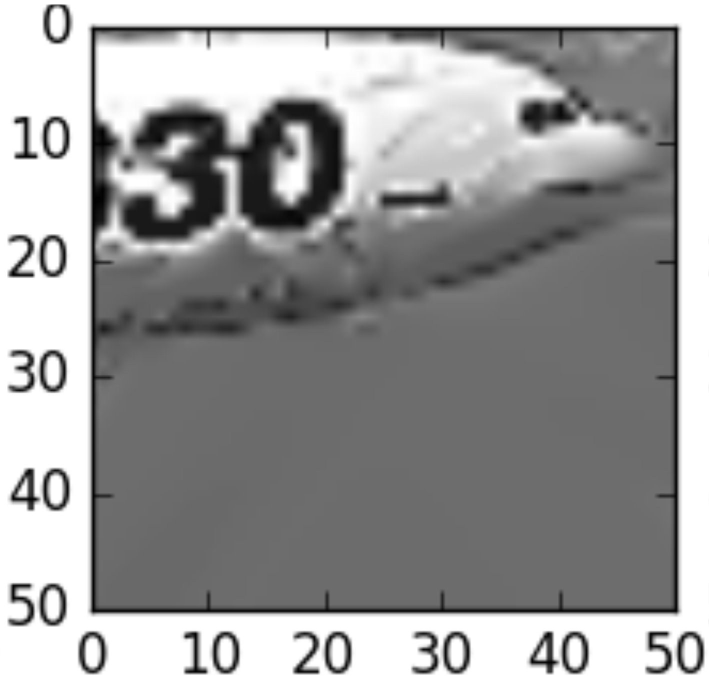


Further improvements in the convergence properties have been obtained by adding a bypass, which allows the final layers to only compute residuals.

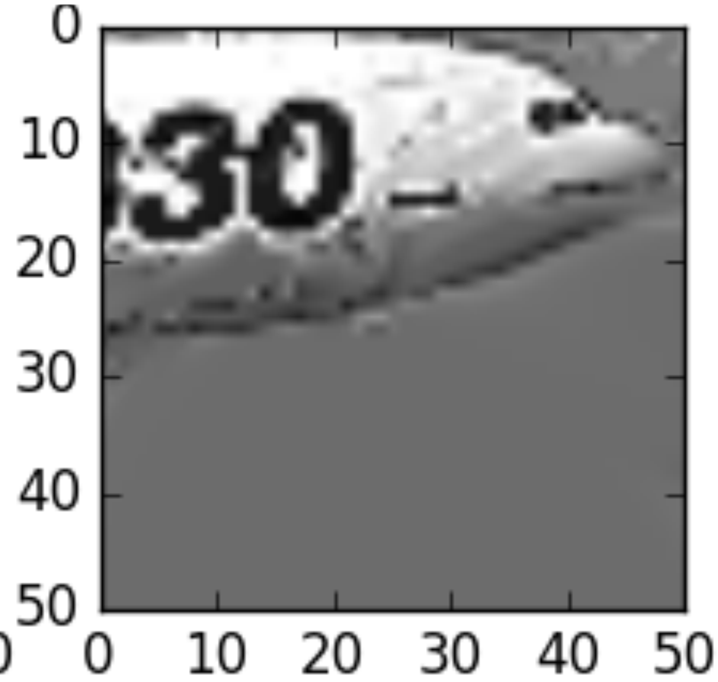
Improving the Network



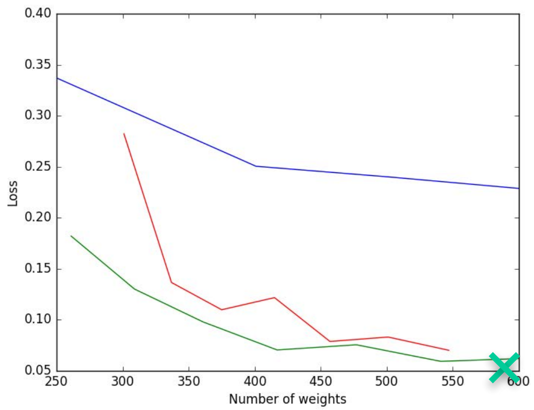
Original 51x51 image:
2601 gray level values.

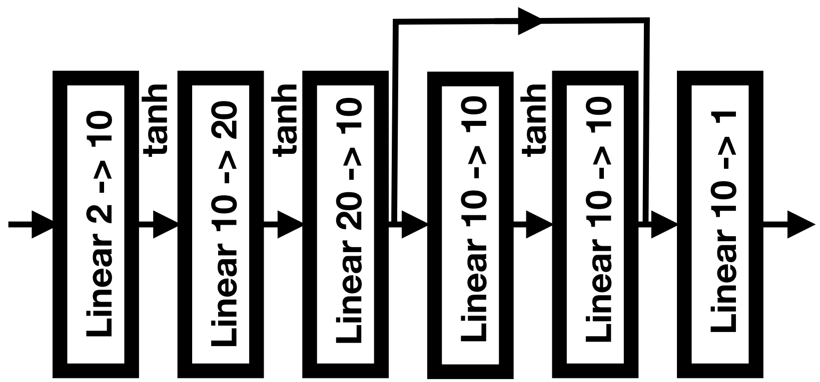


MLP 10/20/10 Interpolation:
471 weights, loss 6.43e-02.



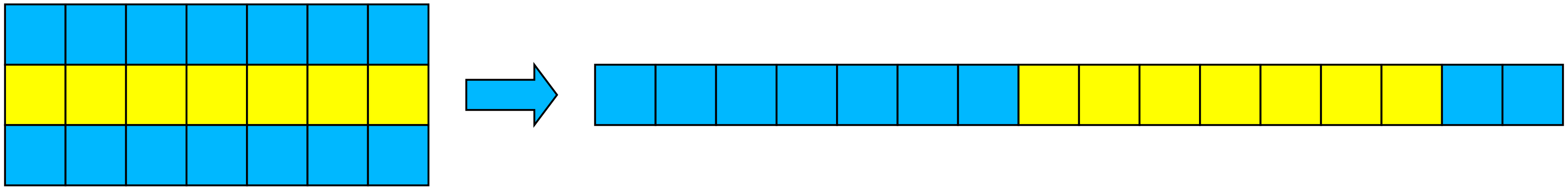
MLP 10/20/10/10 Interpolation:
581 weights, loss 5.30e-2.





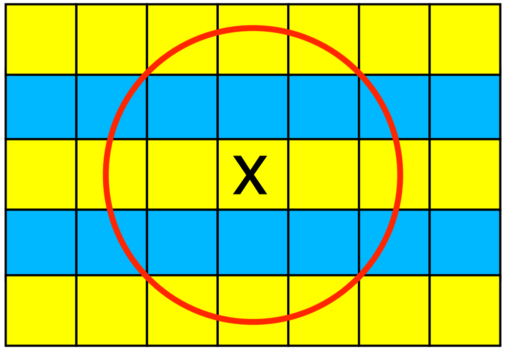
Digital Images

136	134	161	159	163	168	171	173	173	171	166	159	157	155
152	145	136	130	151	149	151	154	158	161	163	163	159	151
145	149	149	145	140	133	145	143	145	145	145	146	148	148
148	143	141	145	145	145	141	136	136	135	135	136	135	133
131	131	129	129	133	136	140	142	142	138	130	128	126	120
115	111	108	106	106	110	120	130	137	142	144	141	129	123
117	109	098	094	094	094	100	110	125	136	141	147	147	145
136	124	116	105	096	096	100	107	116	131	141	147	150	152
152	152	137	124	113	108	105	108	117	129	139	150	157	159
159	157	157	159	135	121	120	120	121	127	136	147	158	163
165	165	163	163	163	166	136	131	135	138	140	145	154	163
166	168	170	168	166	168	170	173	145	143	147	148	152	159
168	173	173	175	173	171	170	173	177	178	151	151	153	156
161	170	176	177	177	179	176	174	174	176	177	179	155	157
161	162	168	176	180	180	180	182	180	175	175	178	180	180



- A MxN image can be represented as an MN vector, in which case neighborhood relationships are lost.
- By contrast, treating it as a 2D array preserves neighborhood relationships.

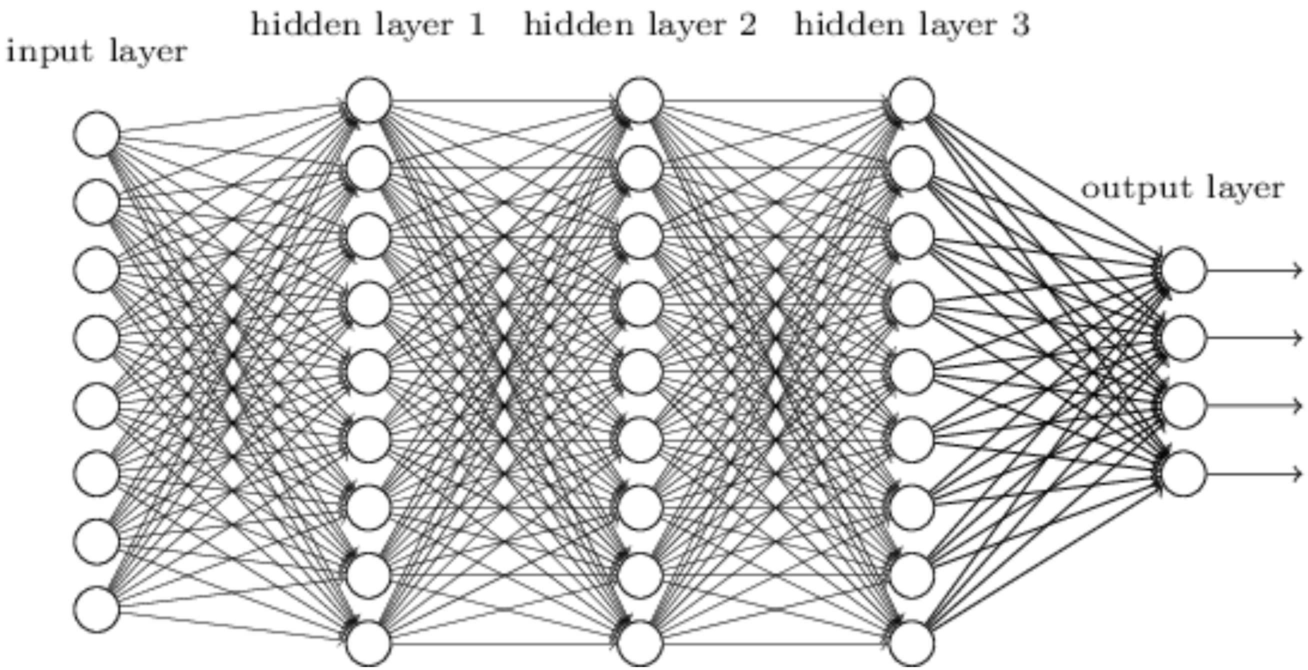
Image Specificities



- In a typical image, the values of neighboring pixels tend to be more highly correlated than those of distant ones.
- An image filter should be translation invariant.

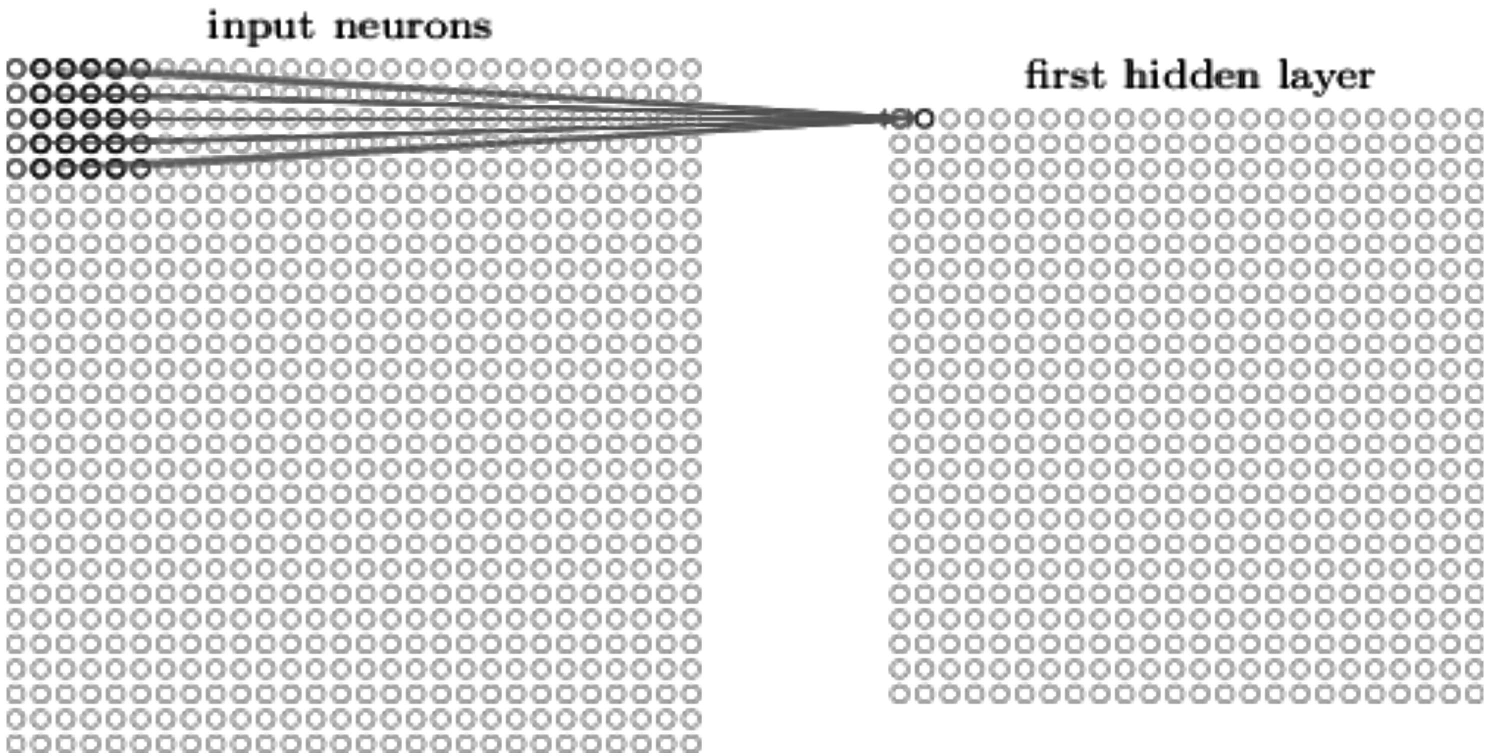
—> These two properties can be exploited to drastically reduce the number of weights required by CNNs using so-called convolutional layers.

Fully Connected Layers



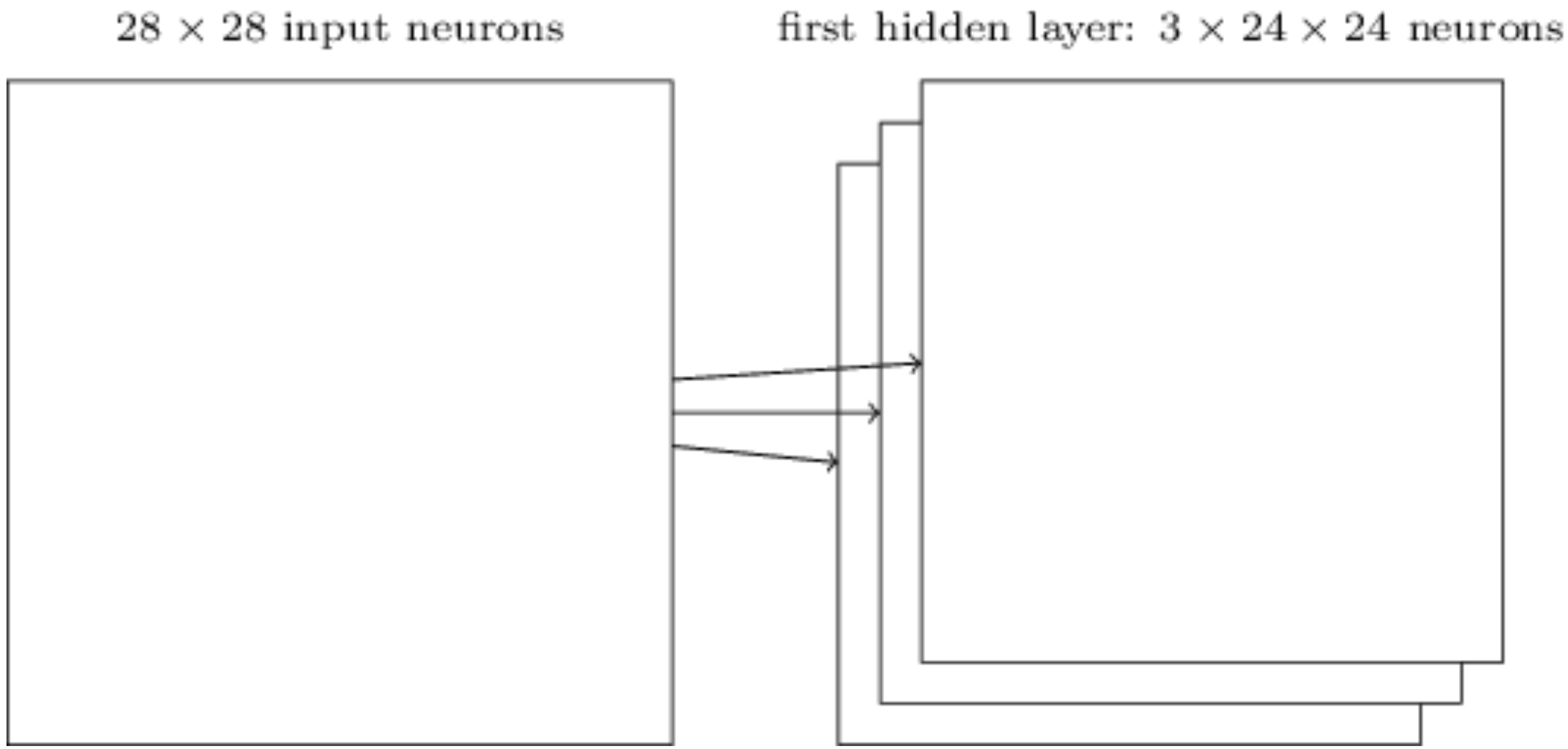
- The descriptive power of the net increases with the number of layers.
- In the case of a 1D signal, it is roughly proportional to $\prod_n W_n$ where W_n represents the width of a layer.

Convolutional Layer

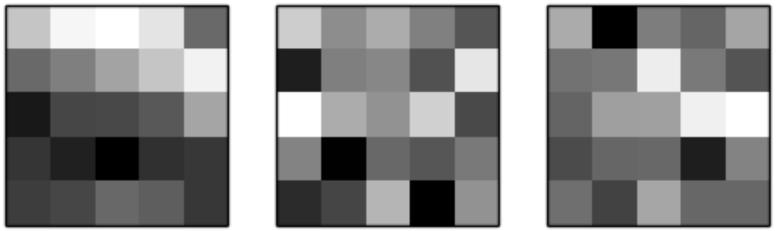


$$\sigma\left(b + \sum_{x=0}^{n_x} \sum_{y=0}^{n_y} w_{x,y} a_{i+x,j+y}\right)$$

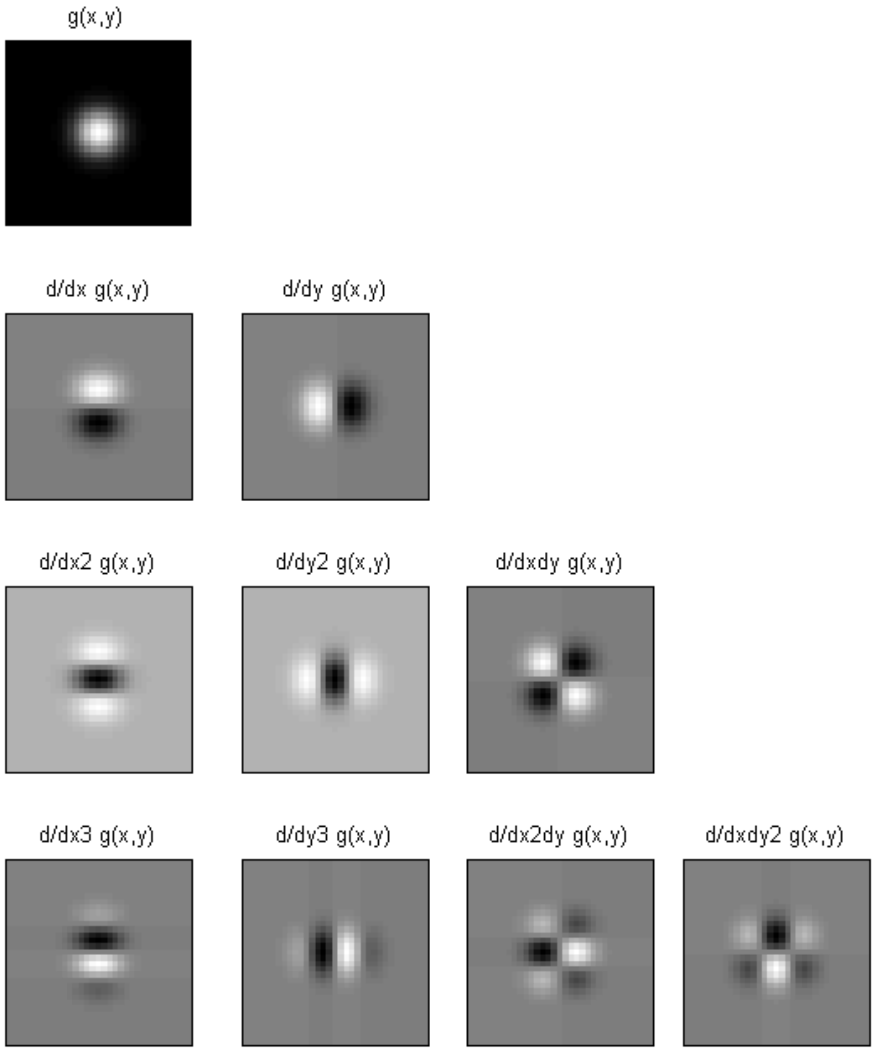
Feature Maps

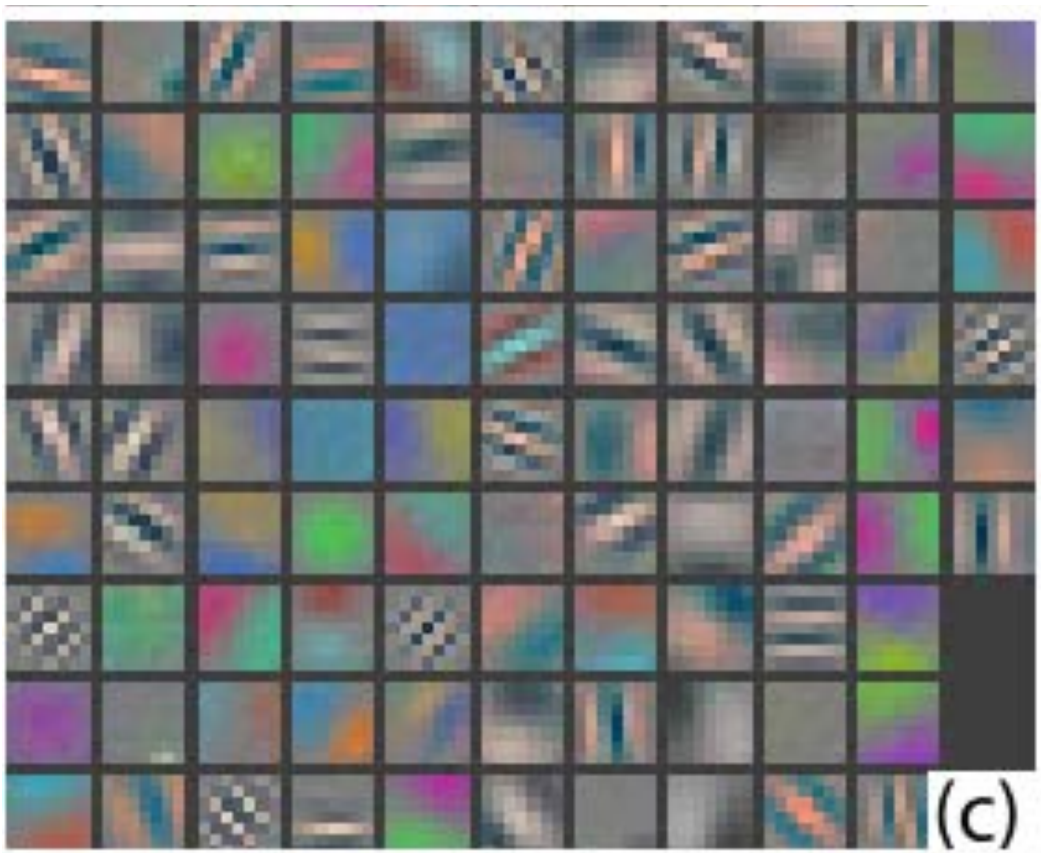


Filters:



Filters

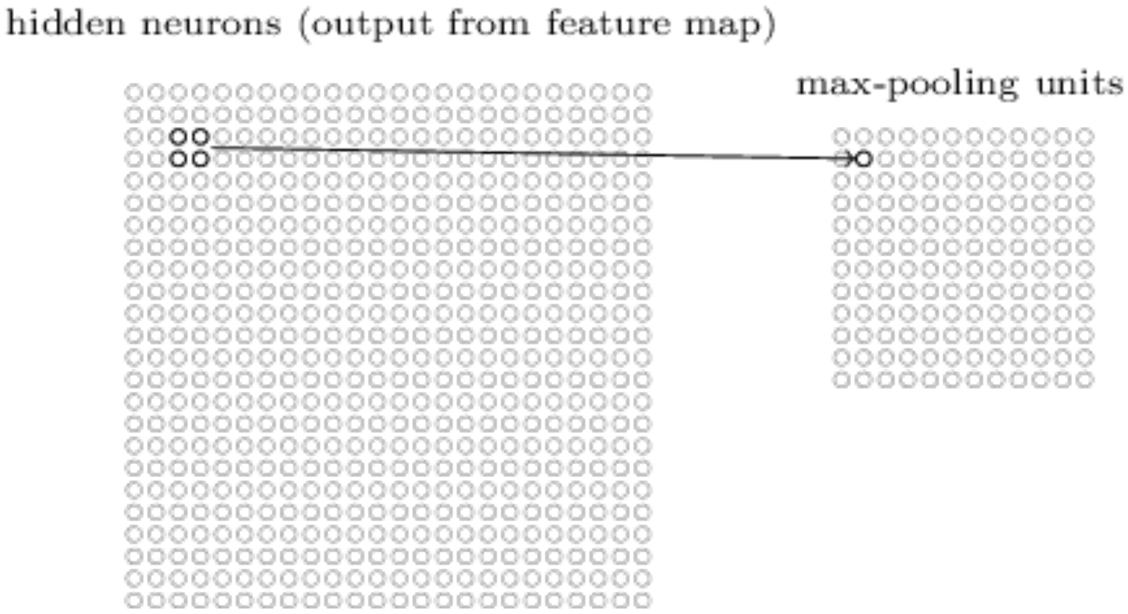




Derivatives

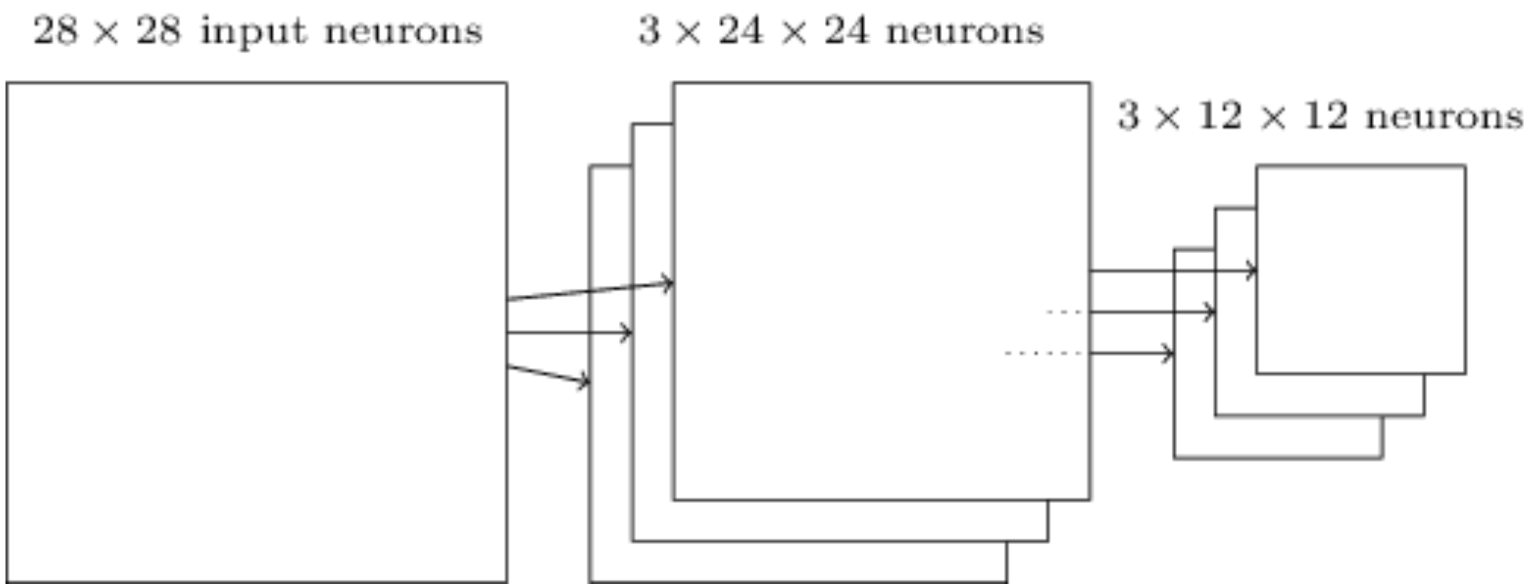
Learned filters

Pooling Layer



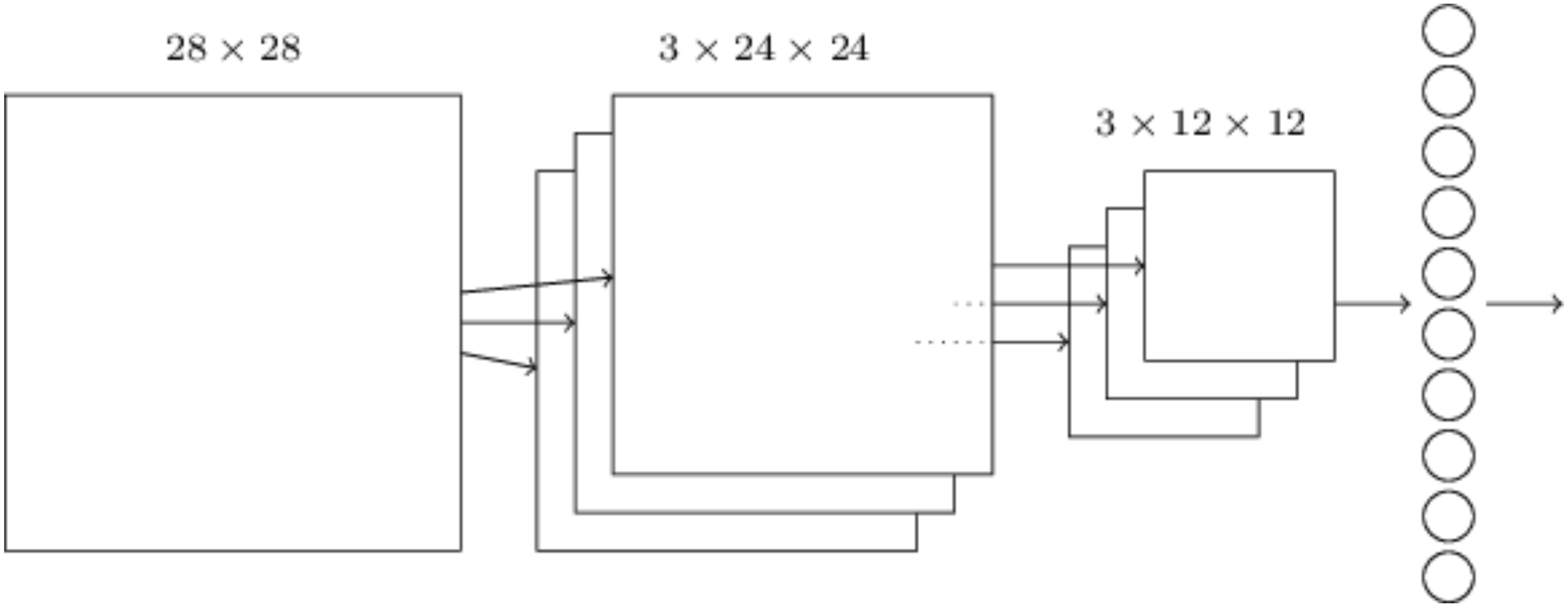
- Reduces the number of inputs by replacing all activations in a neighborhood by a single one.
- Can be thought as asking if a particular feature is present in that neighborhood while ignoring the exact location.

Adding the Pooling Layers



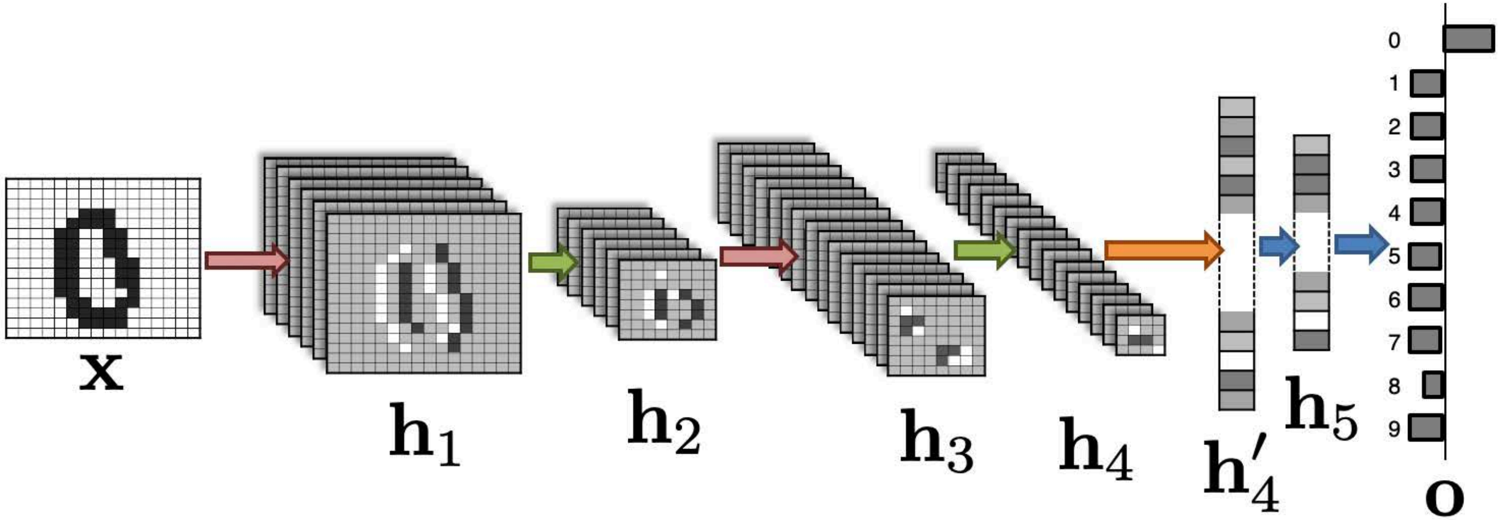
The output size is reduced by the pooling layers.

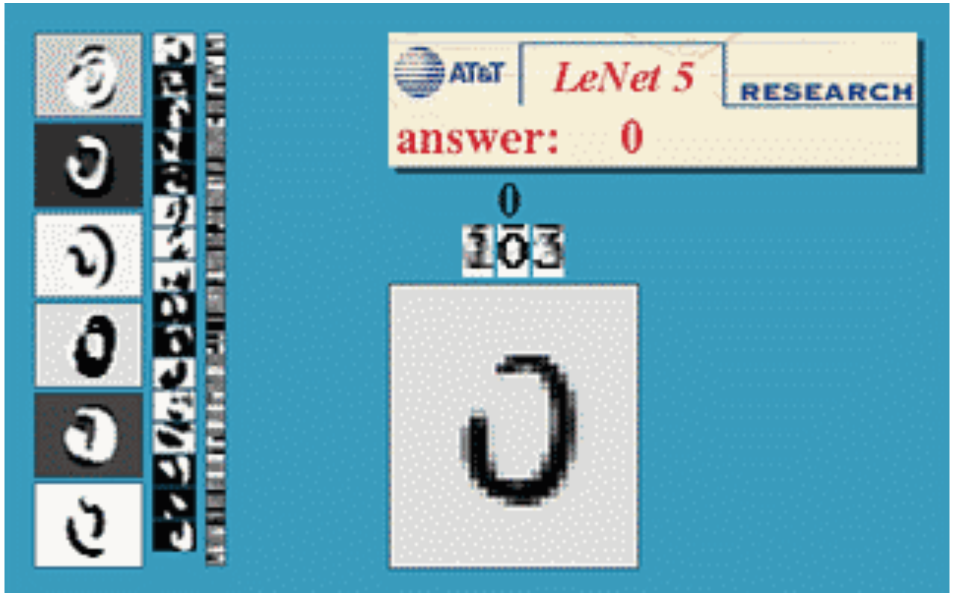
Adding a Fully Connected Layer



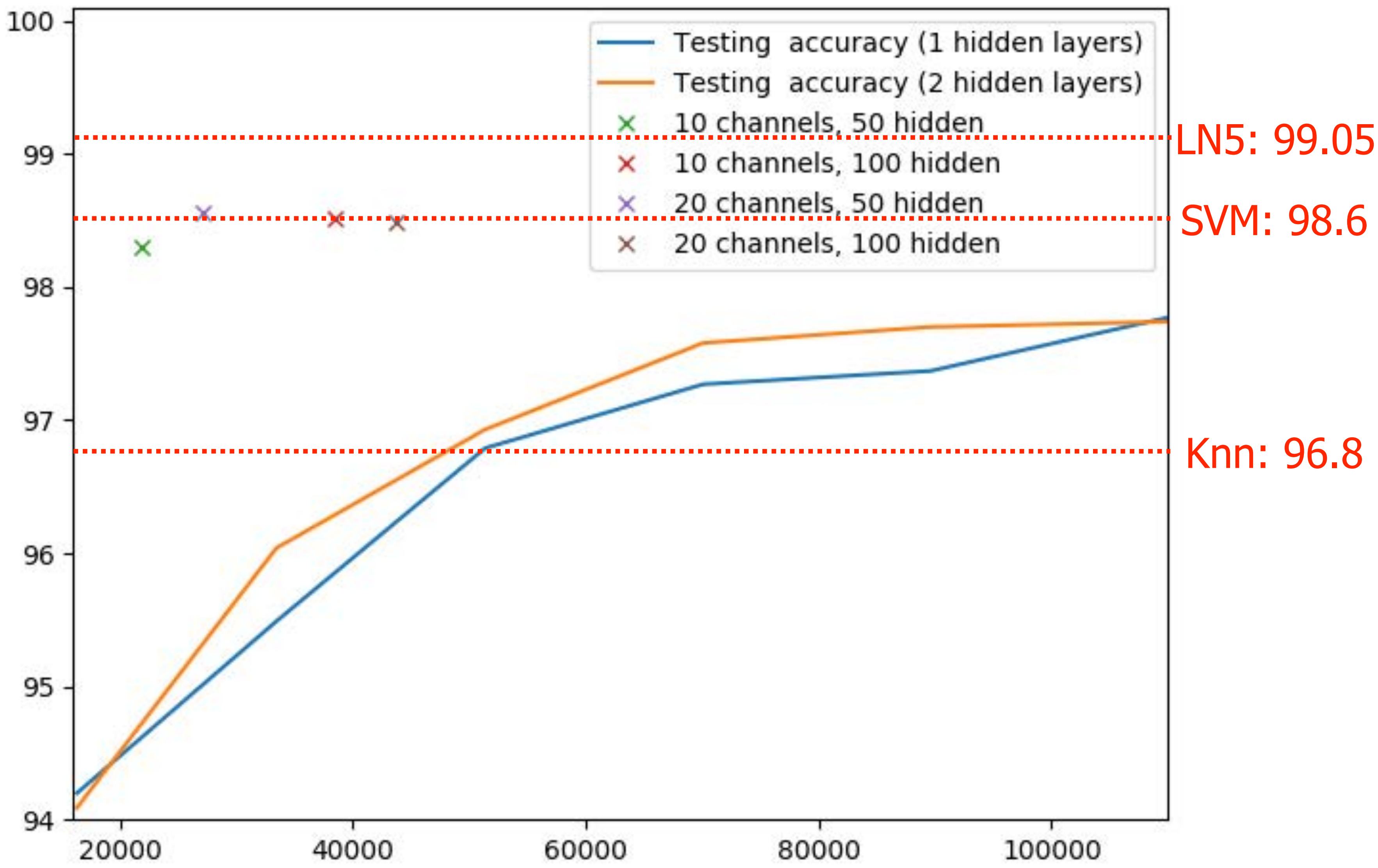
- Each neutron in the final fully connected layer is connected to all neurons in the preceding one.
- Deep architecture with many parameters to learn but still far fewer than an equivalent multilayer perceptron.

LeNet (1989-1999)

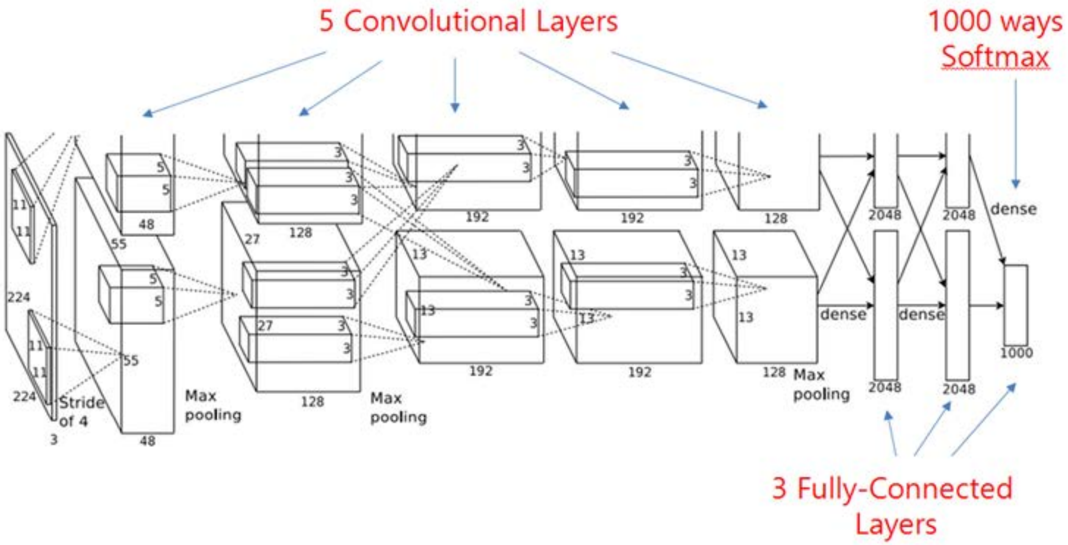




Lenet Results



AlexNet (2012)



Task: Image classification

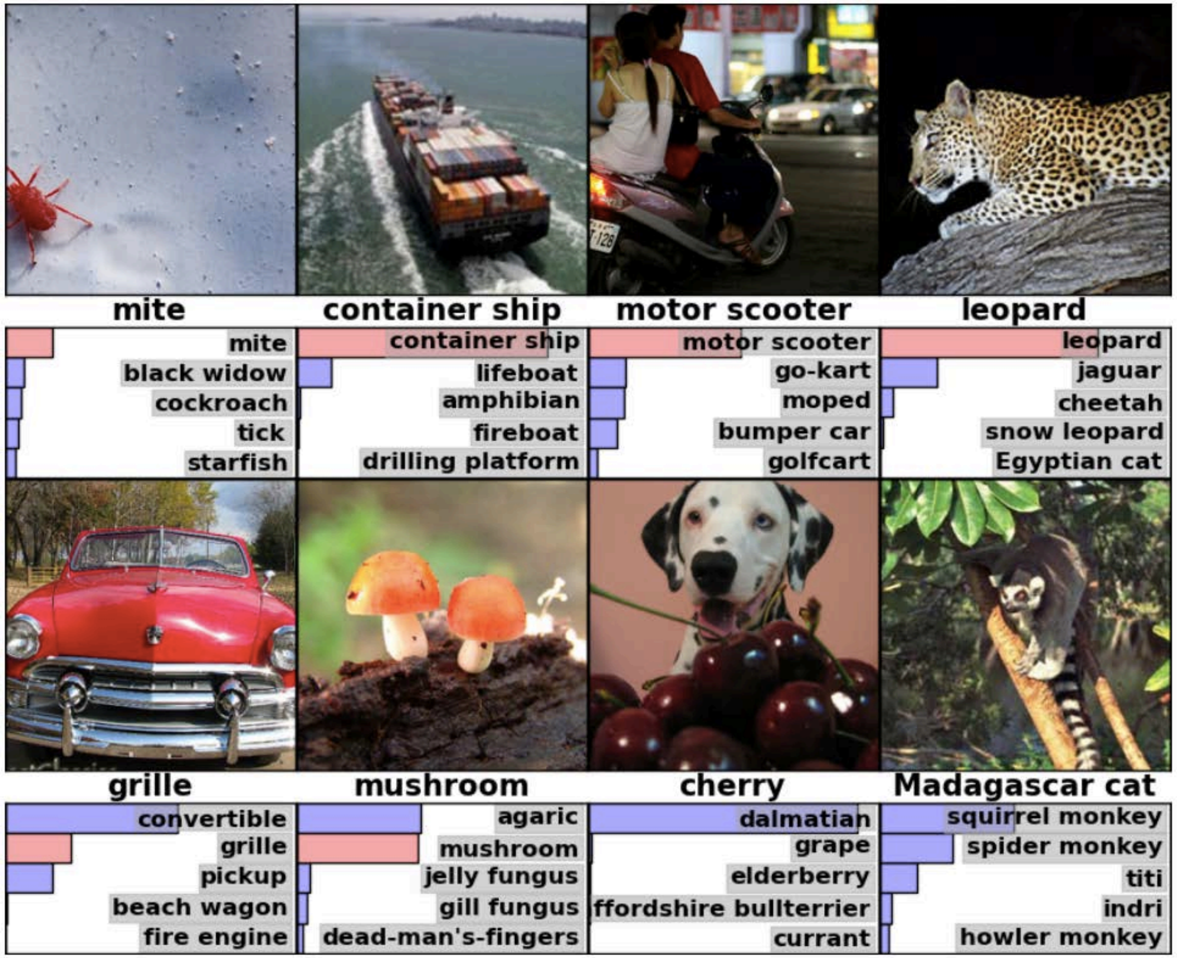
Training images: Large Scale Visual Recognition Challenge 2010

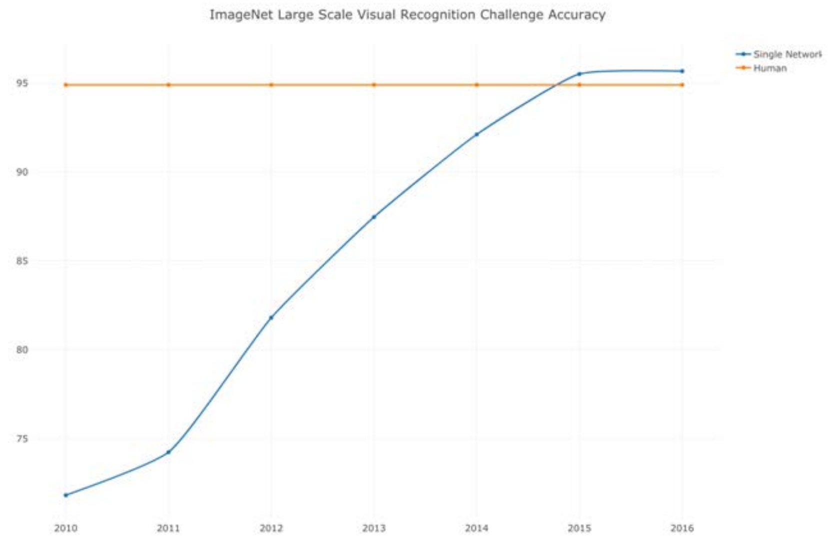
Training time: 2 weeks on 2 GPUs

Major Breakthrough: Training large networks has now been shown to be practical!!

Krizhevsky, NIPS'12

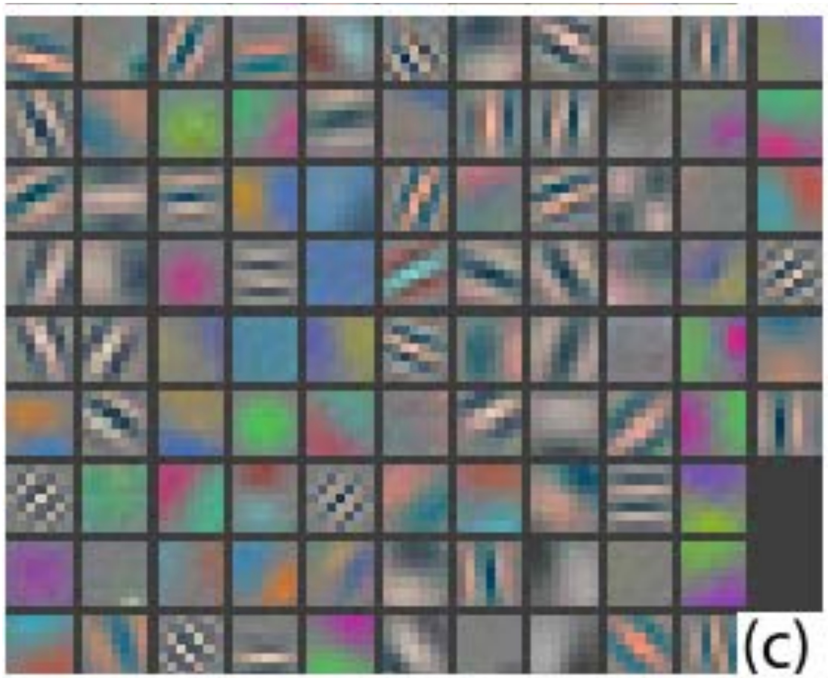
AlexNet Results



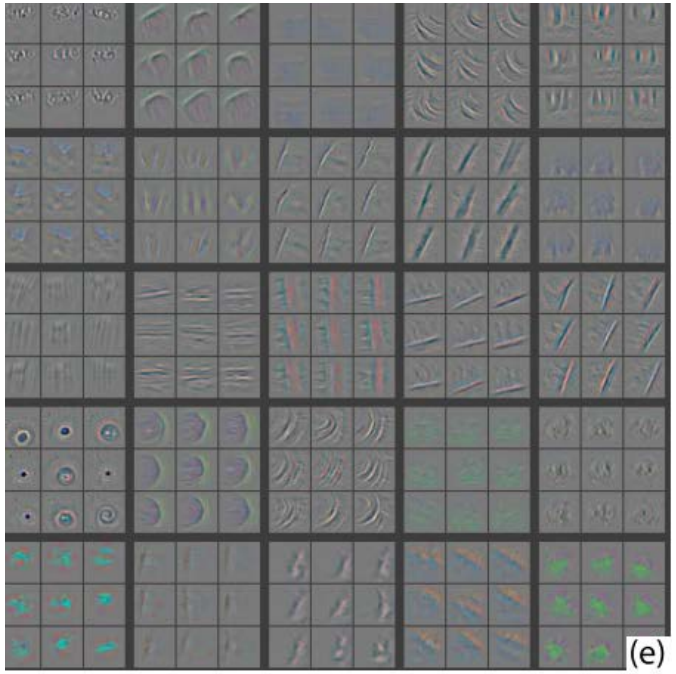


- At the 2012 ImageNet Large Scale Visual Recognition Challenge, AlexNet achieved a top-5 error of 15.3%, more than 10.8% lower than the runner up.
- Since 2015, networks outperform humans on this task.

Feature Maps



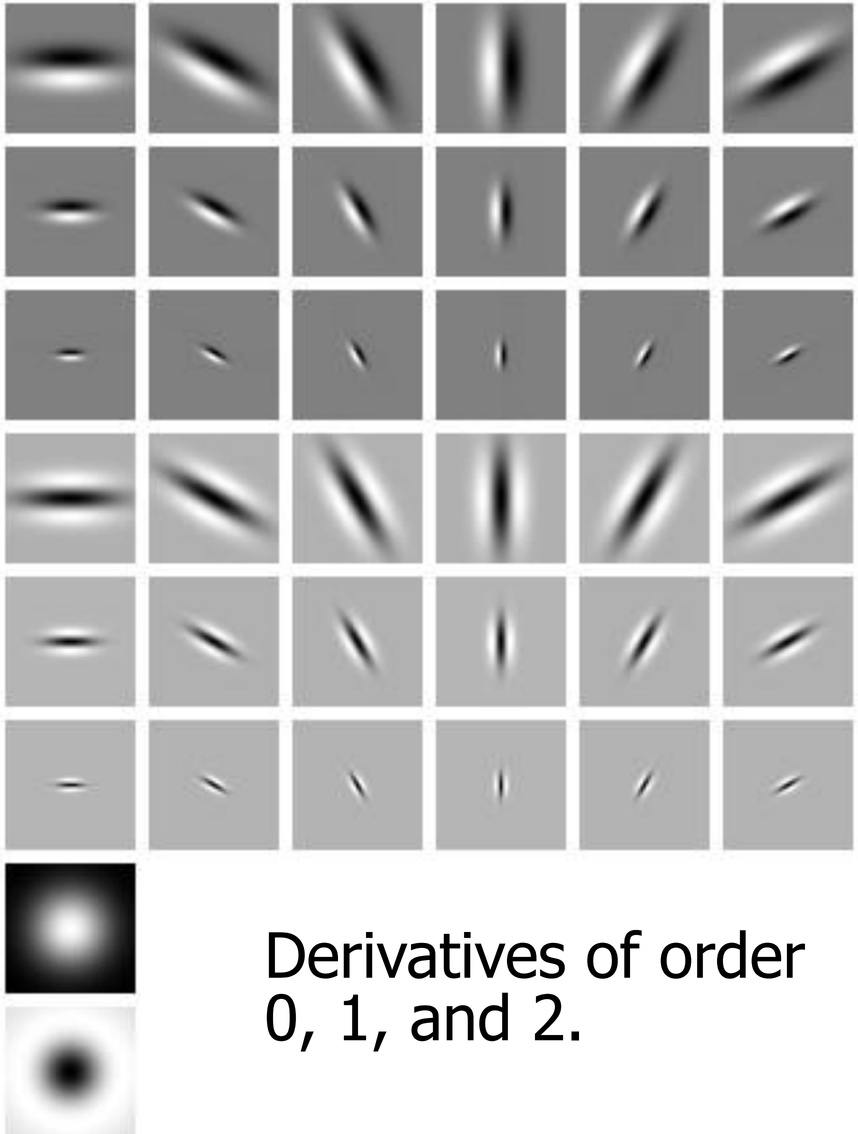
First convolutional layer



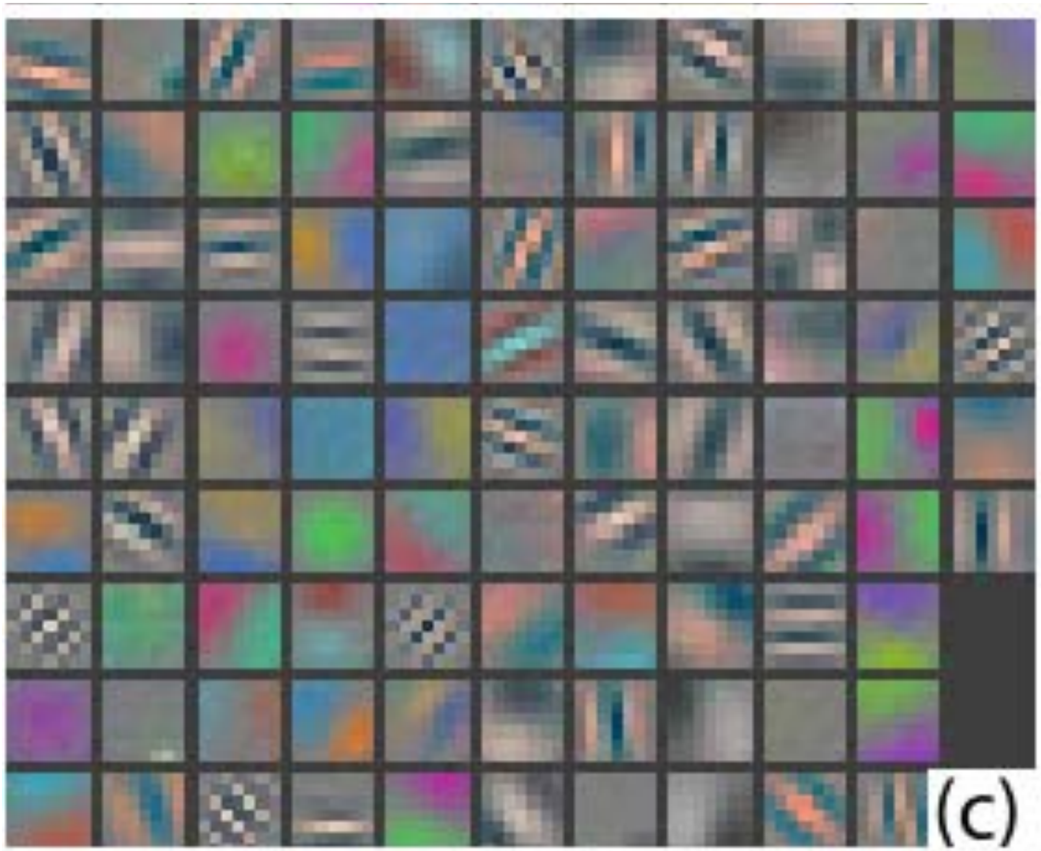
Second convolutional layer

- Some of the convolutional masks are very similar to oriented Gaussian or Gabor filters.
- The trained neural nets compute oriented derivatives, which the brain is also **believed** to do.

Filter Banks



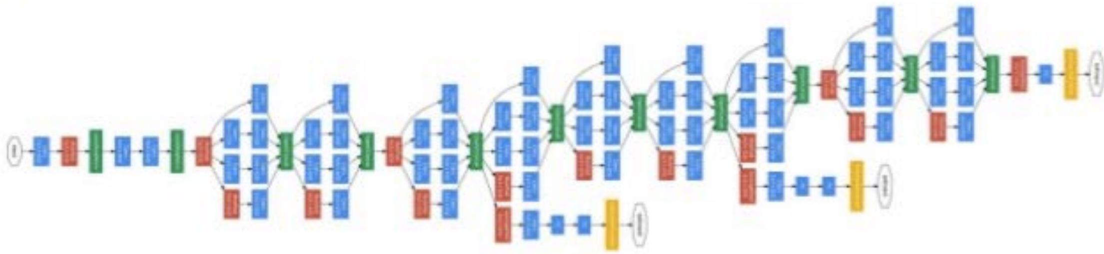
Derivatives of order 0, 1, and 2.



Learned

Bigger and Deeper

VGG19, 3 weeks of training.



"hibiscus"

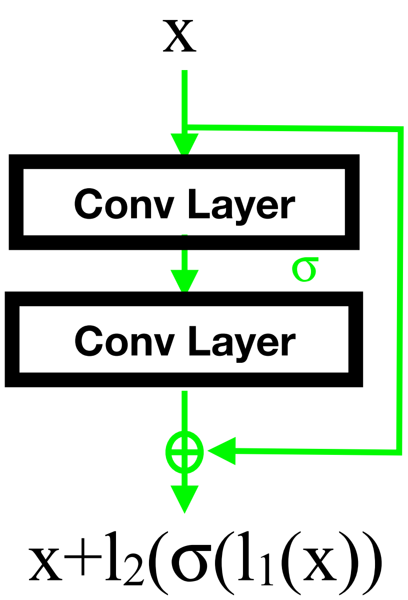
"dahlia"

GoogleLeNet.

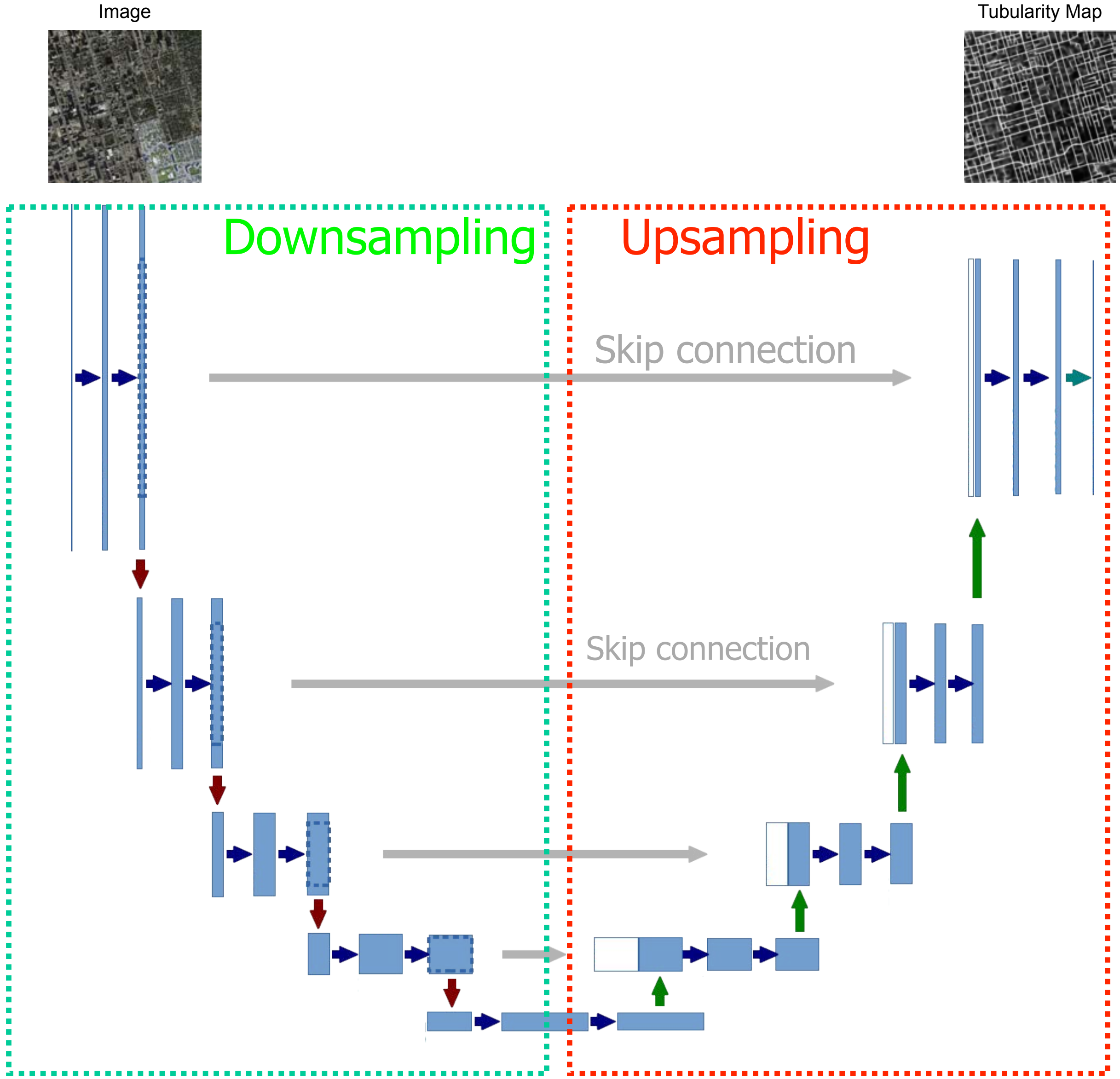
"It was demonstrated that the representation depth is beneficial for the classification accuracy, and that state-of-the-art performance on the ImageNet challenge dataset can be achieved using a conventional ConvNet architecture."

Simonyan & Zisserman, ICLR'15

ResNet to U-Net



ResNet block



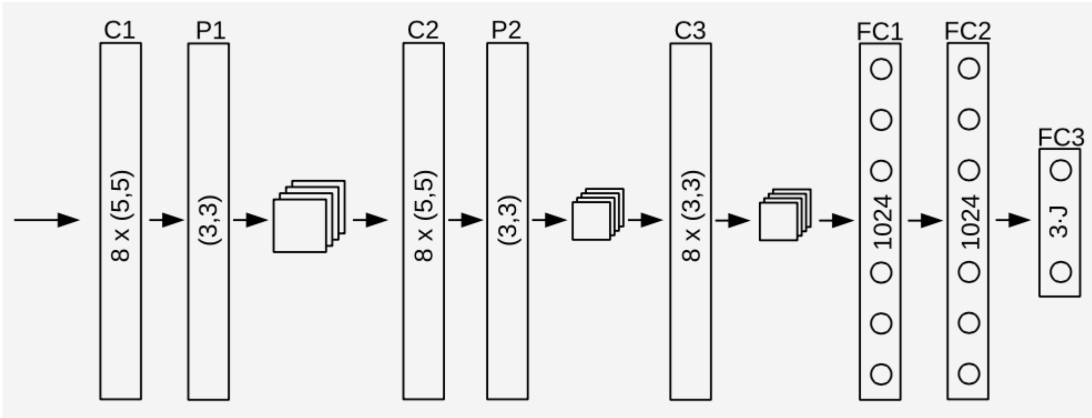
U-Net

—> Add skip connection to produce an output of the same size as the input.

EPFL

Regression

Input



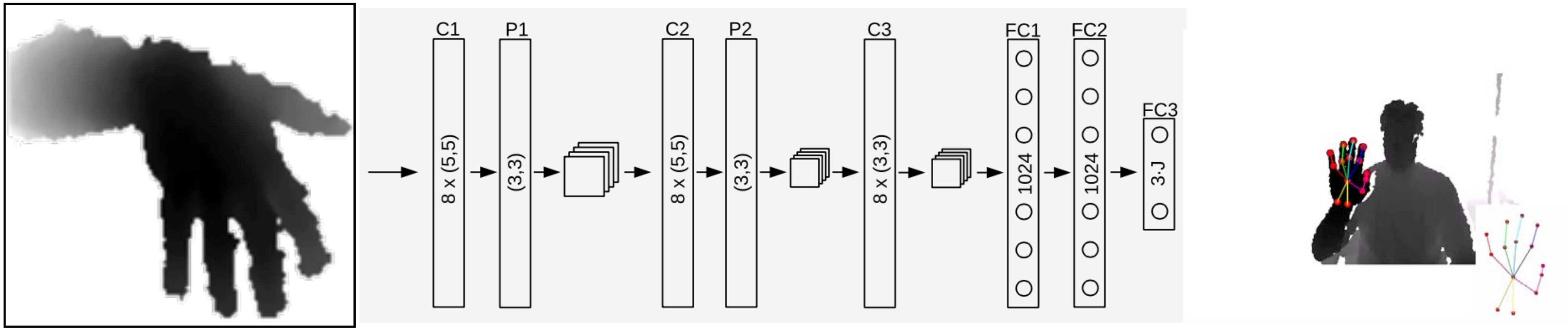
Output

$$\min_{\mathbf{W}_l,\mathbf{B}_l} \sum_i ||\mathbf{F}(\mathbf{x}_i, \mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L) - \mathbf{y}_i||^2$$

using

- stochastic gradient descent on mini-batches,
- dropout,
- hard example mining,
- ………..

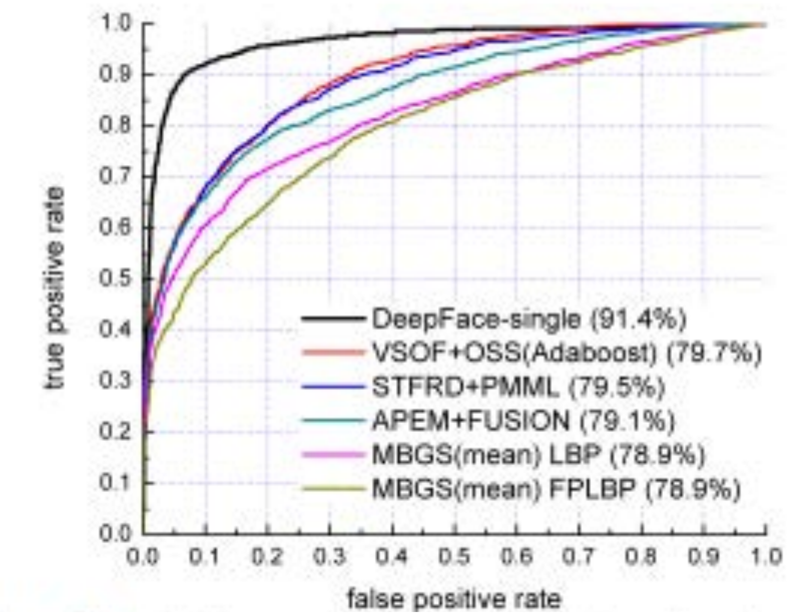
Hand Pose Estimation (2015)



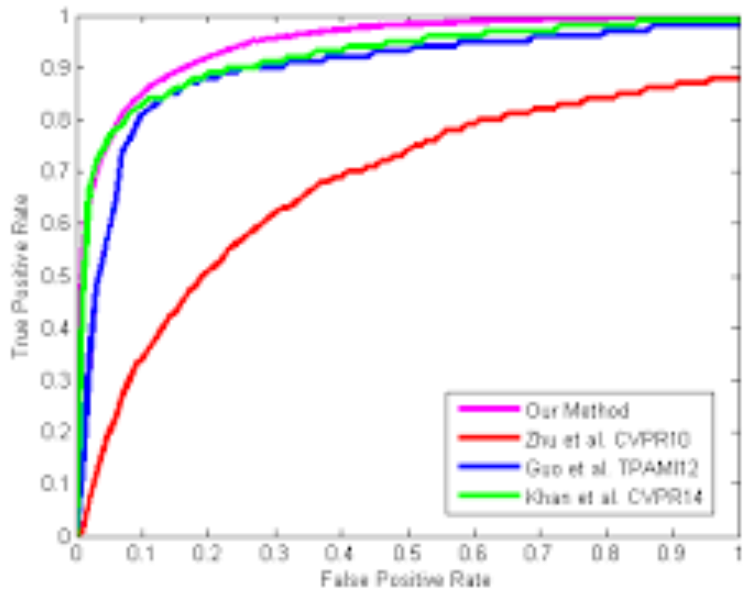
Input: Depth image.

Output: 3D pose vector.

Roc Hunting

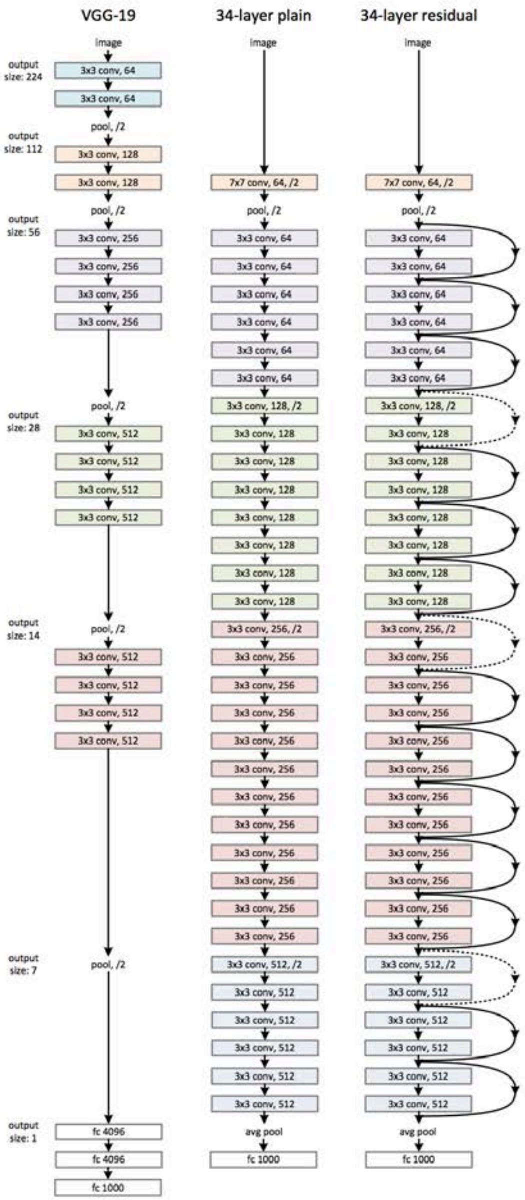


DeepFace
Taigman et al. 2014

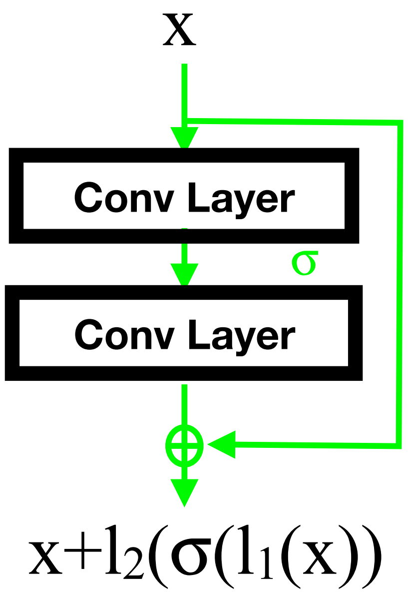


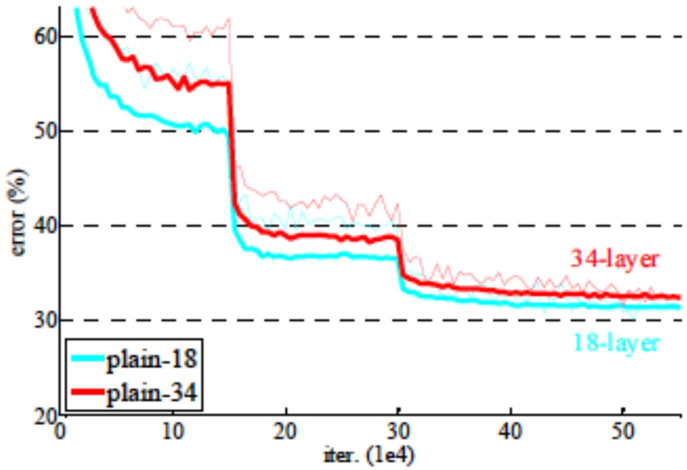
Deep Edge Detection
Shen et al. 2015

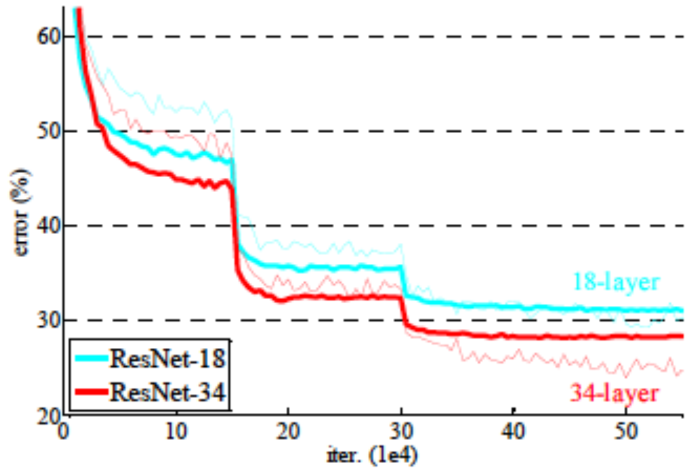
Deeper and Deeper



Resnet

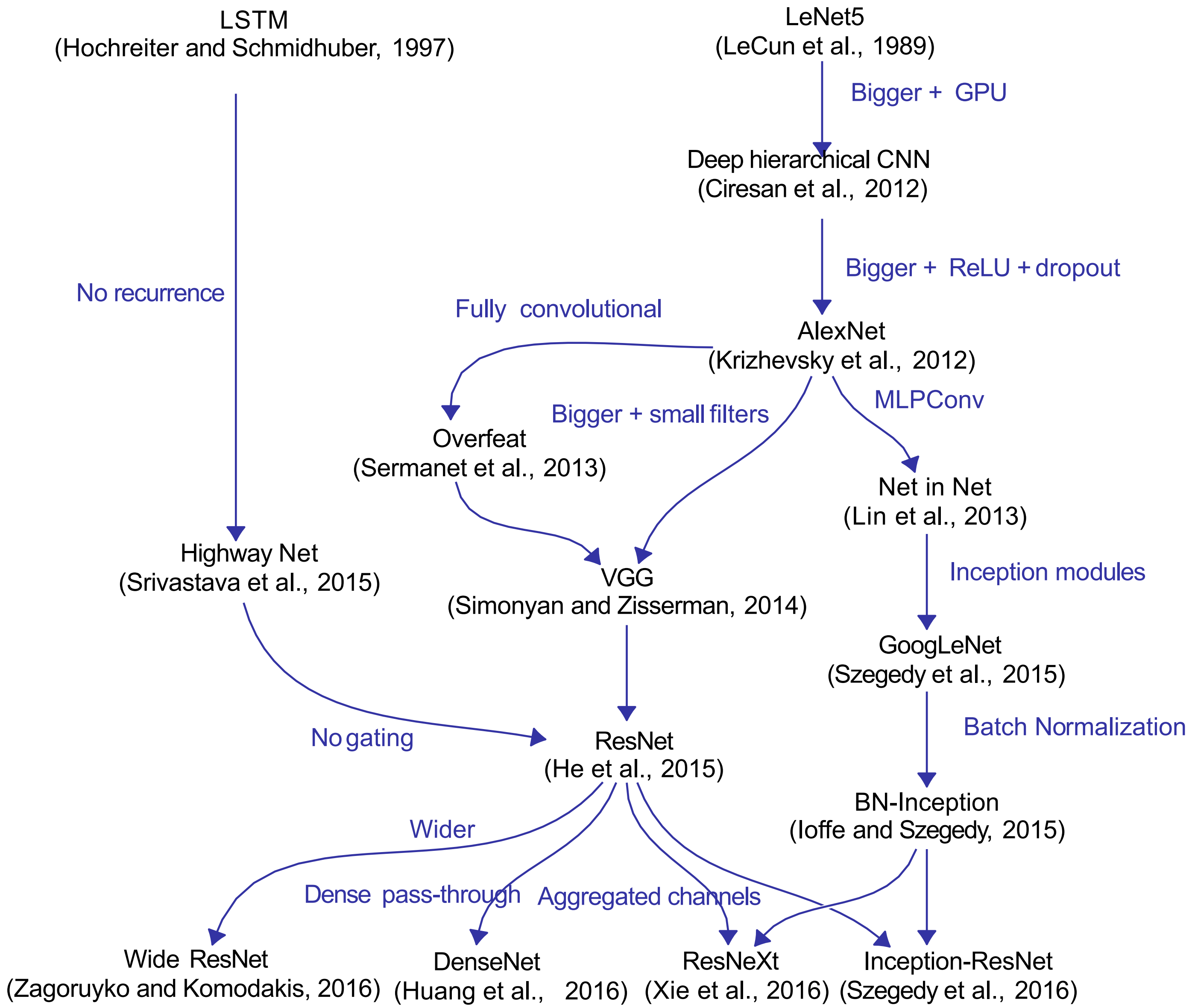




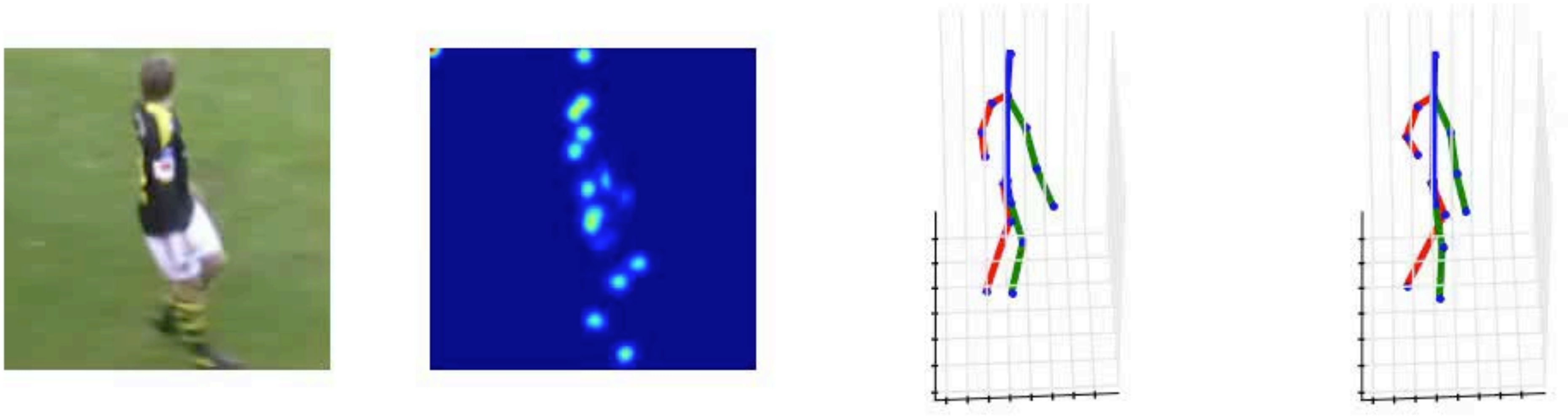


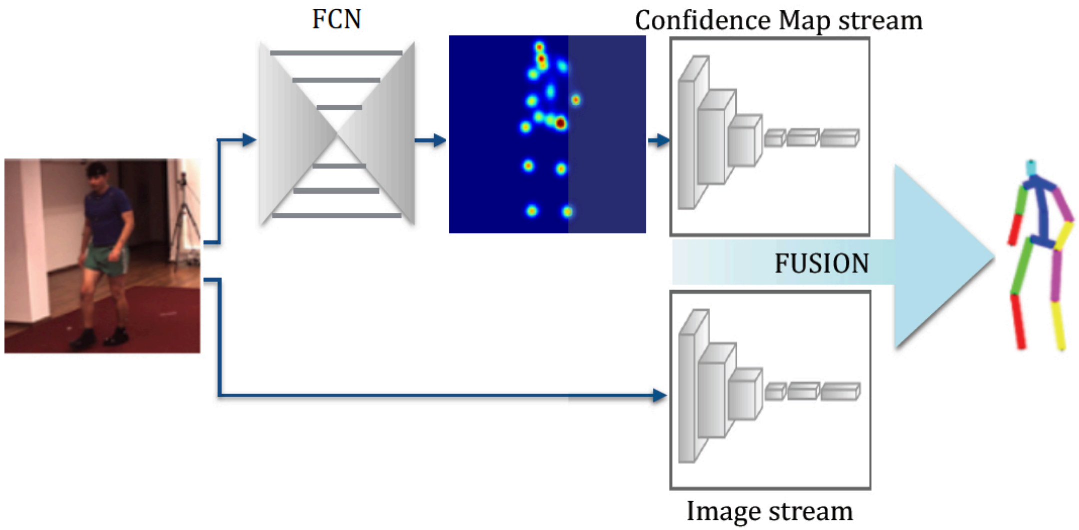
He et al. , CVPR’16

Image Classification Taxonomy

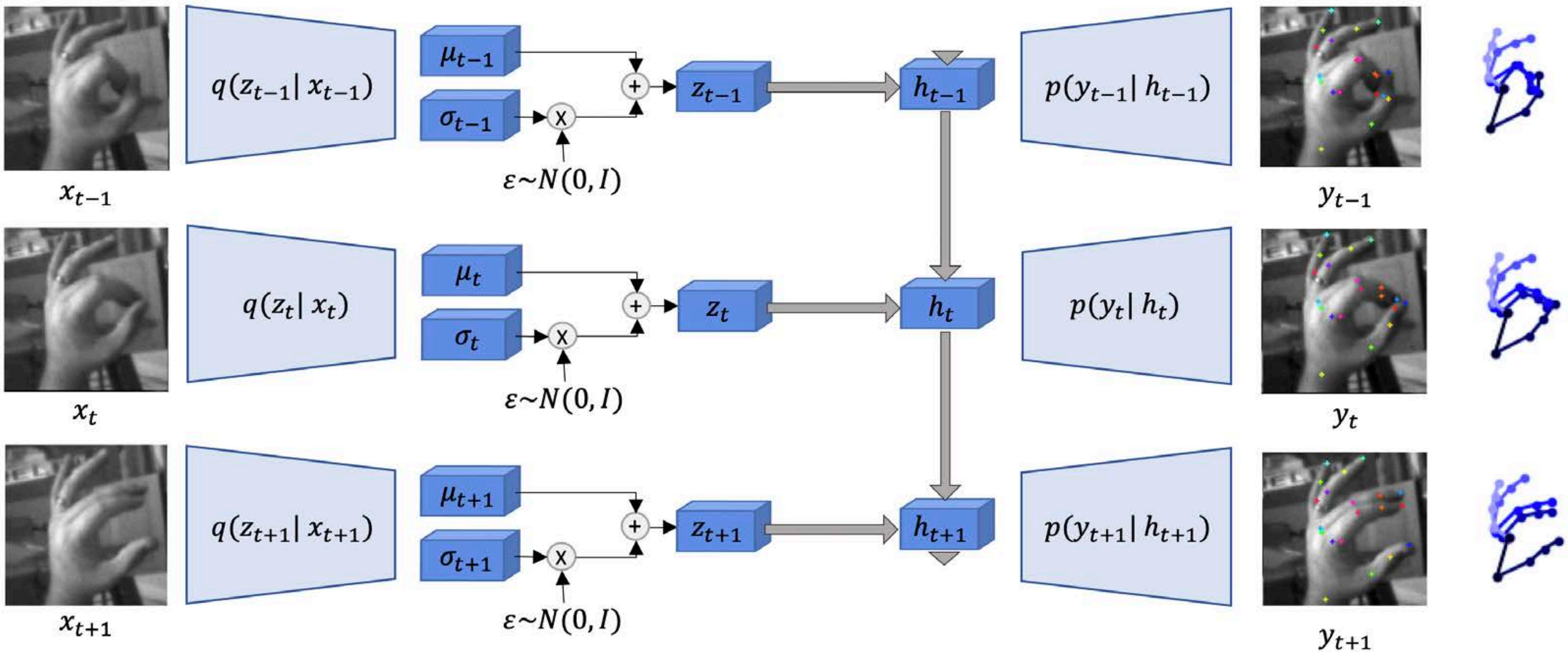


Monocular Pose Estimation



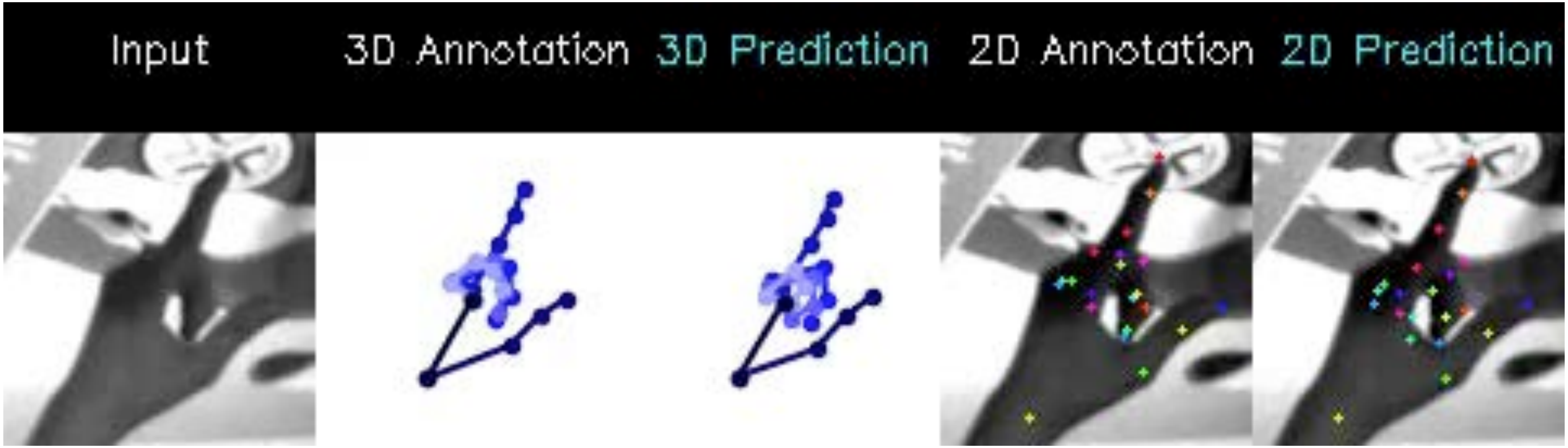


Recurrent Auto Encoder



Remelli et al. , ArXiv'19

Hand Pose Estimation from Video (2019)



- This is considerably more difficult than estimating from range images.
- It requires a large training database.

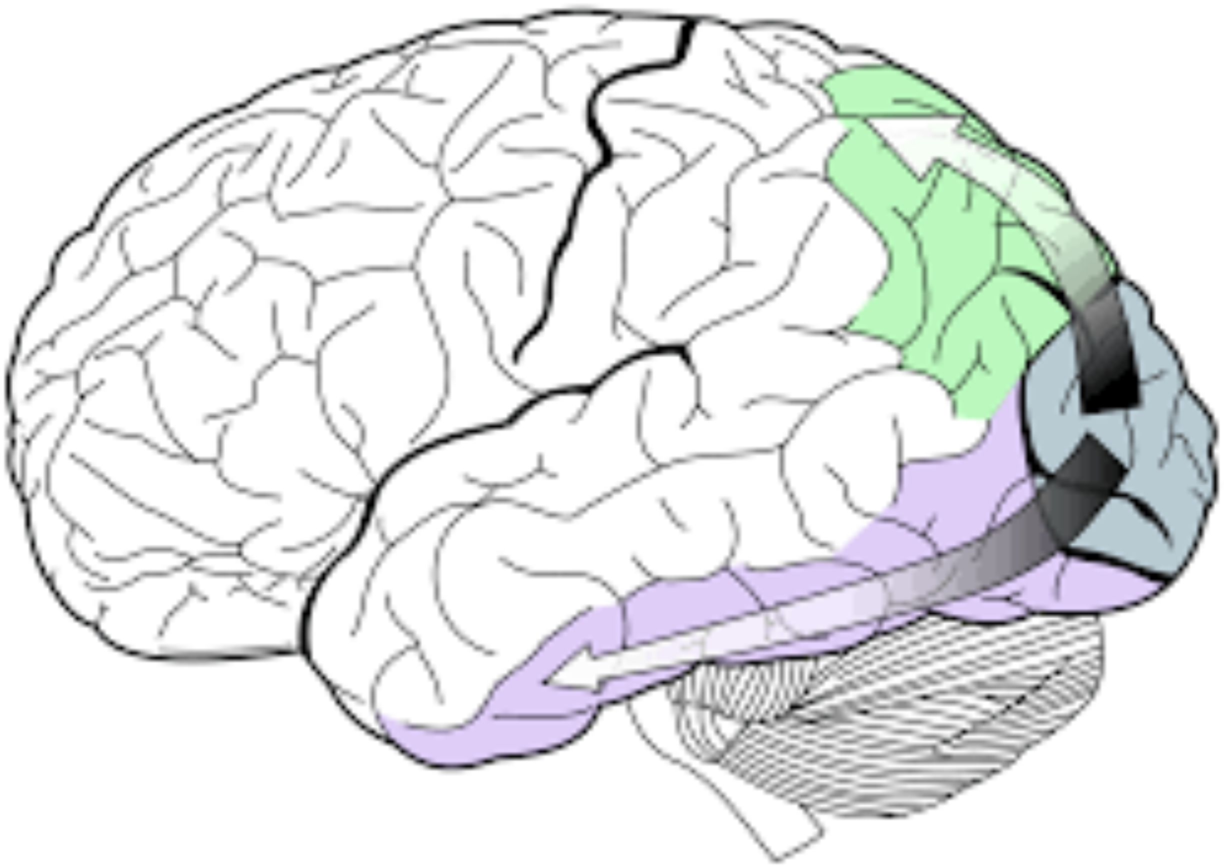
Remelli et al. , ArXiv'19

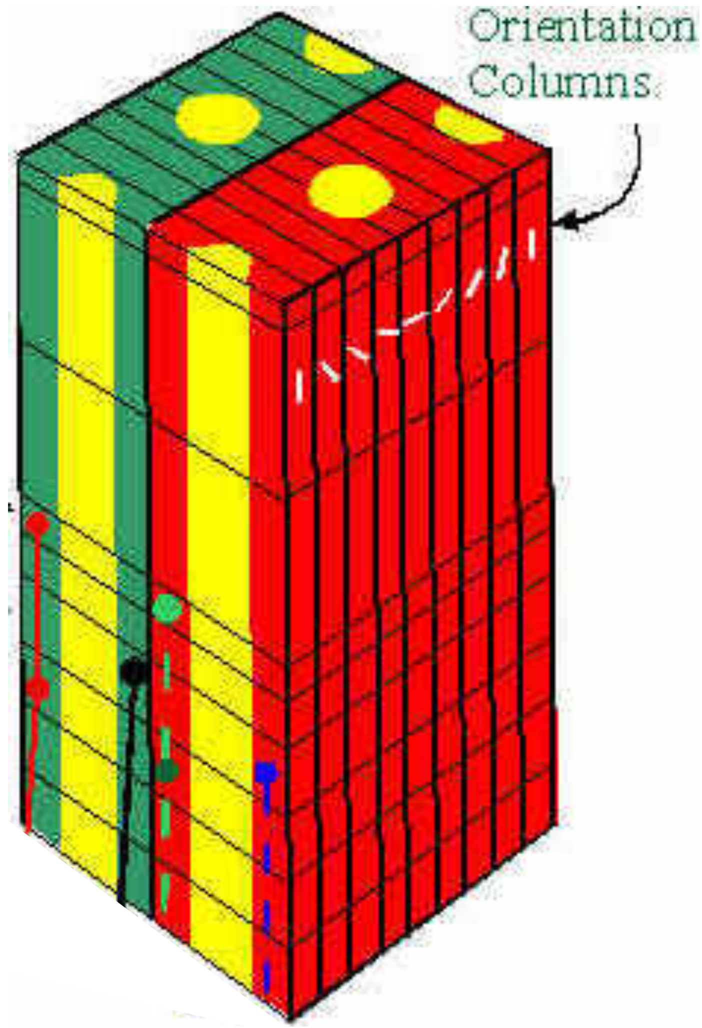
Alpha Go

- Uses Deep Nets to find the most promising locations to focus on.
- Performs Tree based search when possible.
- Relies on reinforcement learning and other ML techniques to train.

—> Beat the world champion in 2017.

Visual Cortex





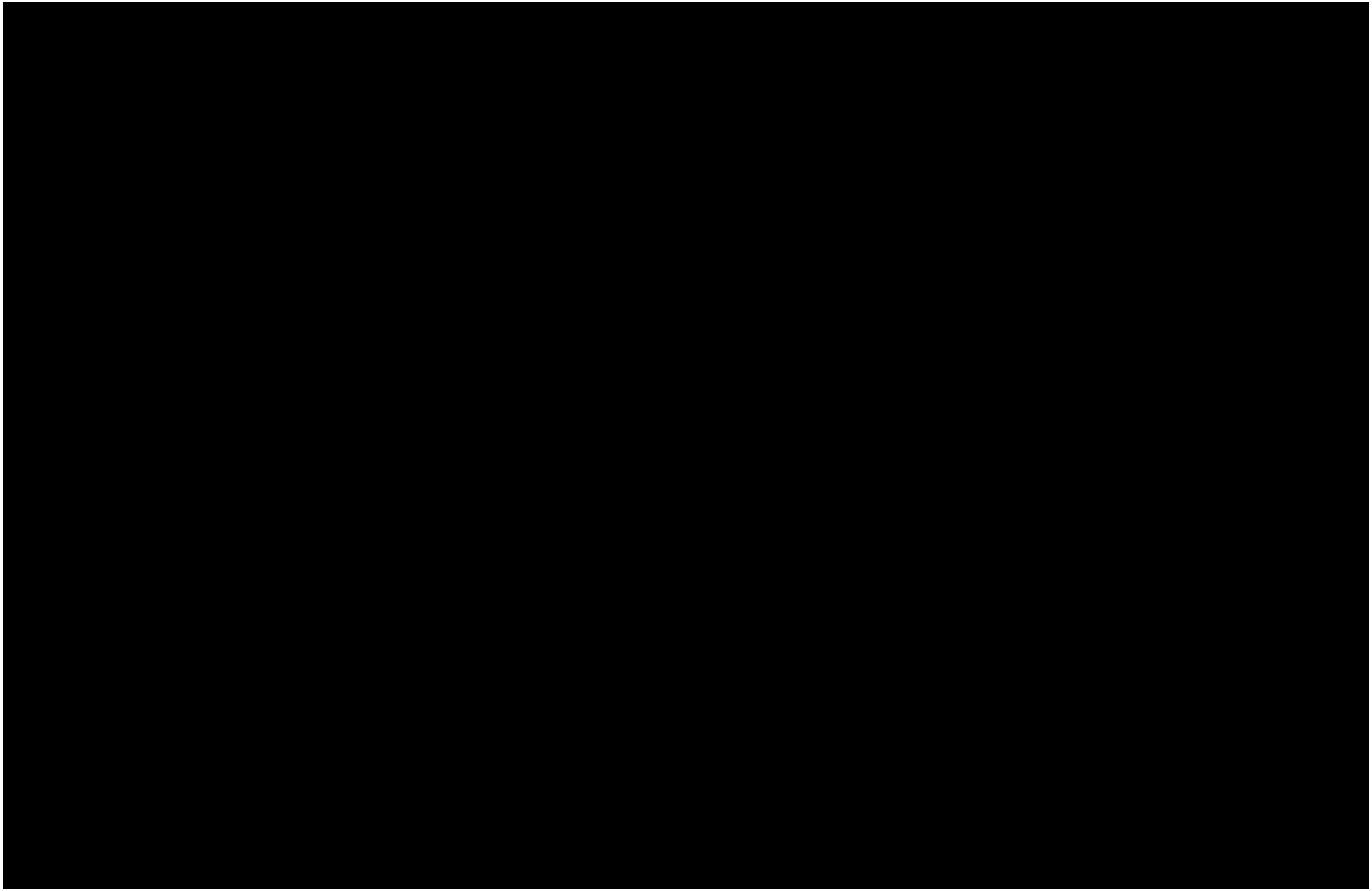
Recognize And Classify: Animal /No Animal

Subjects must raise their hand if they see an animal:

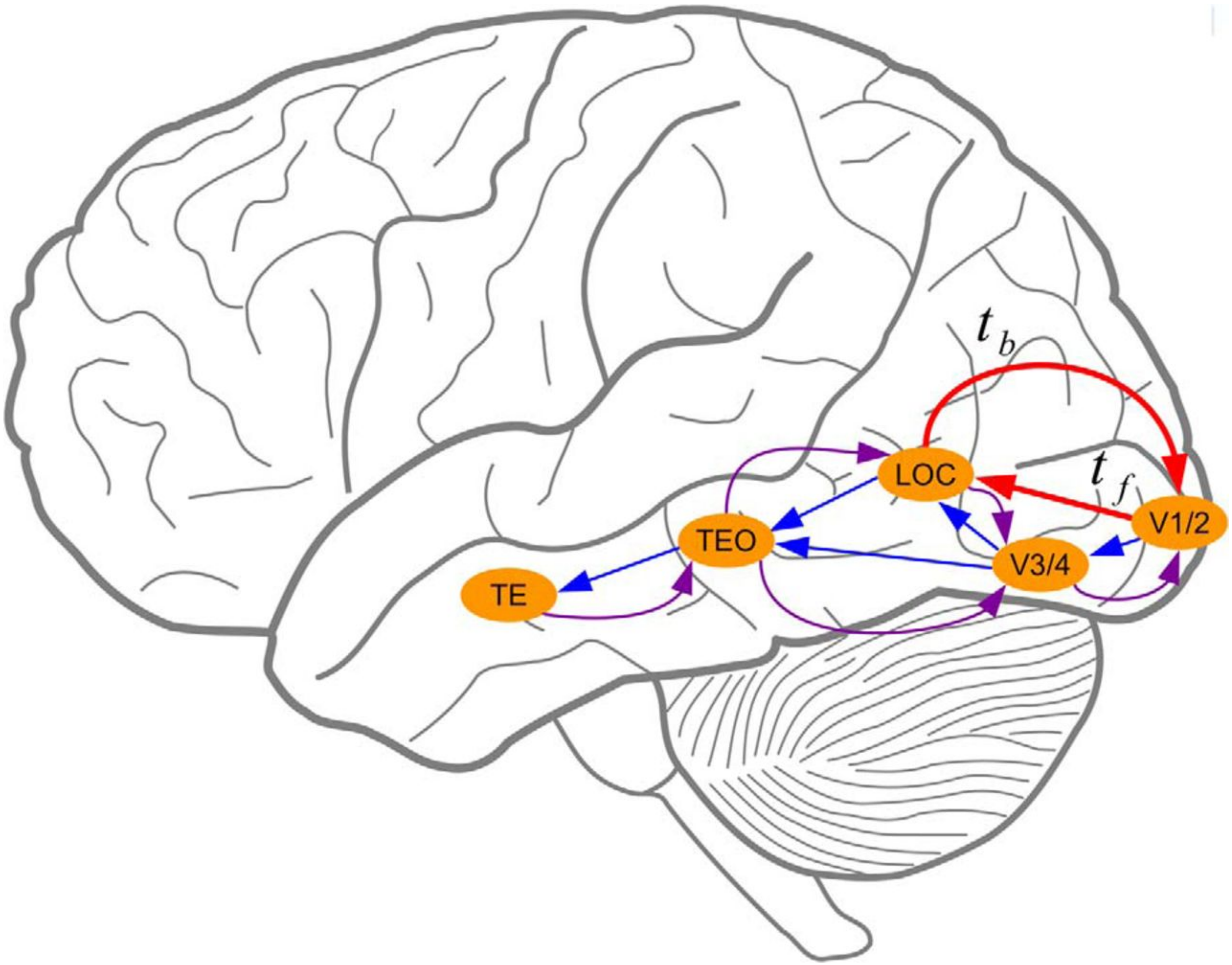
- 60 images
- 1 image per second

→ Measure their reaction time.

Simon Thorpe, *Nature*, 1996



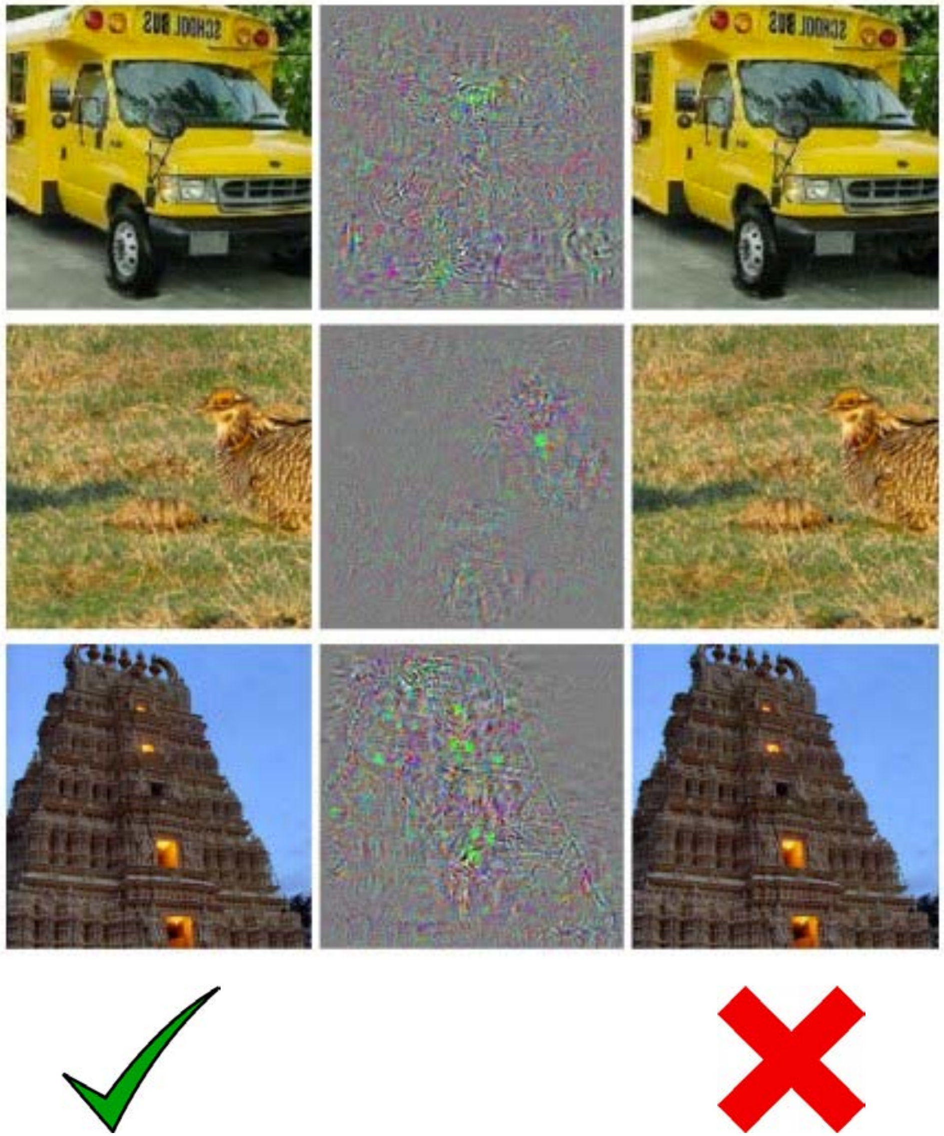
Reminder: Recurrent Pathways



"Shape stimuli are optimally reinforcing each other when separated in time by ~60 ms, suggesting an underlying recurrent circuit with a time constant (feedforward + feedback) of 60 ms."

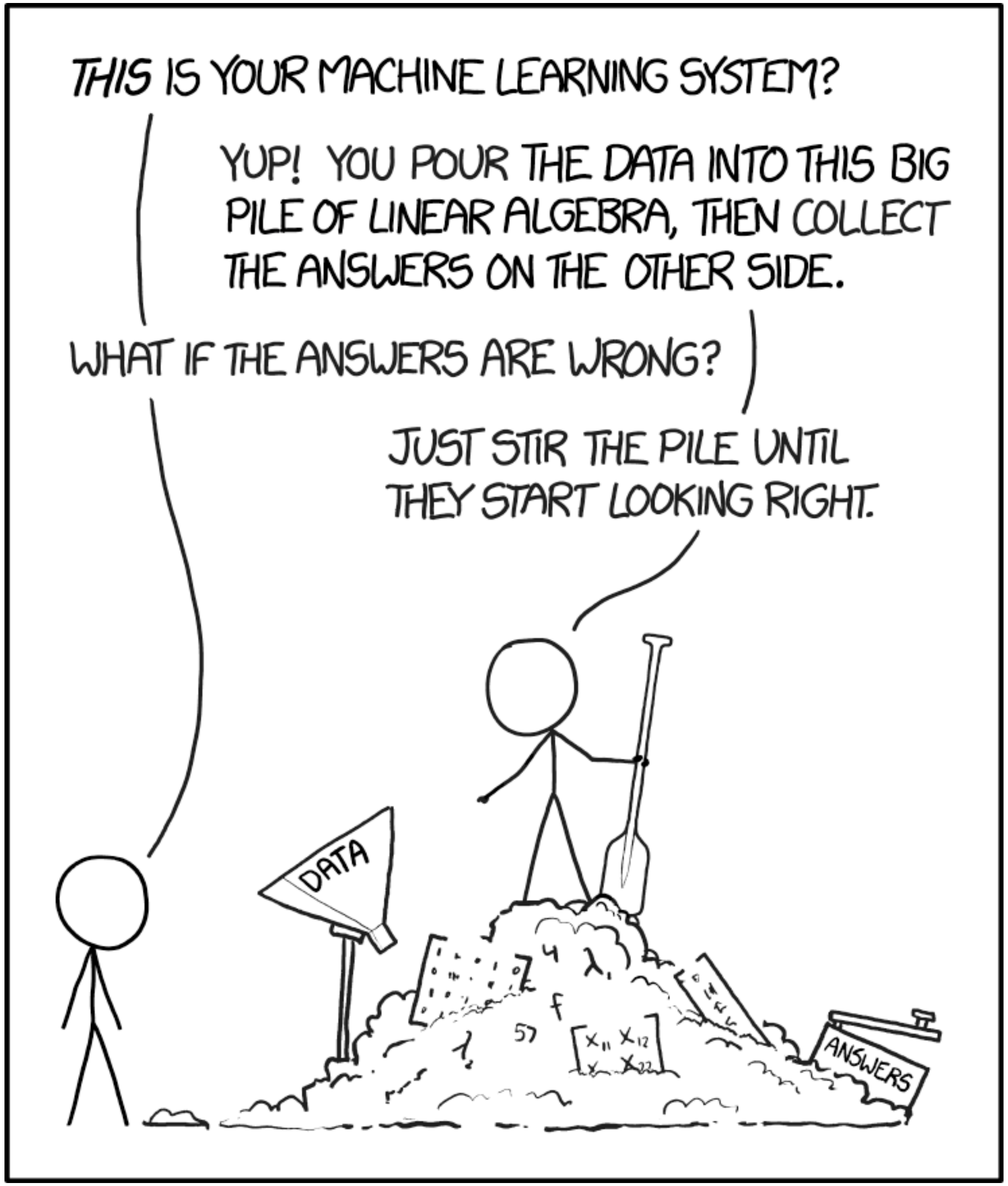
Drewes et al. , *Journal of Neuroscience*, 2016

Adversarial Images



Szegedy et al. 2013

XKCD's View On The Matter



https://xkcd.com/

Deep Nets in Short

- Deep Neural Networks can handle huge training databases.
- When the objective function can be minimized, the results are outstanding.
- There are failure cases and performance is hard to predict.

—> Many questions are still open and there is much theoretical work left to do.