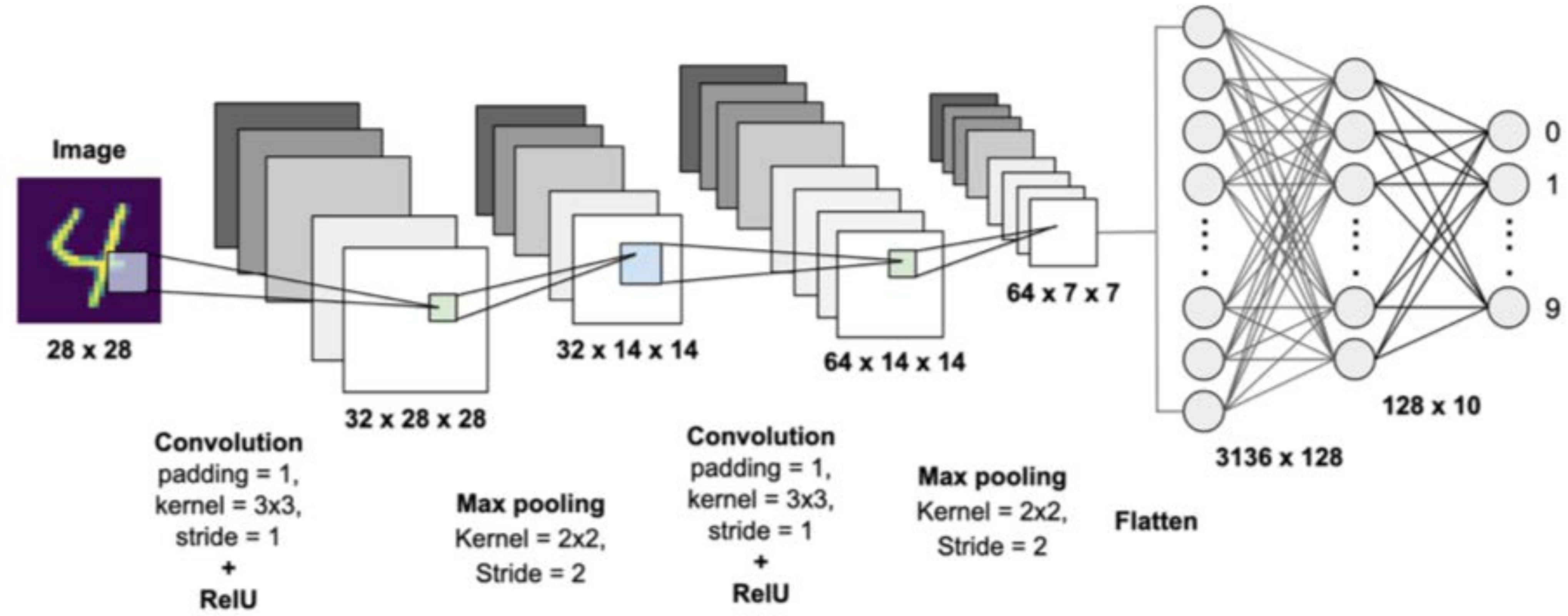
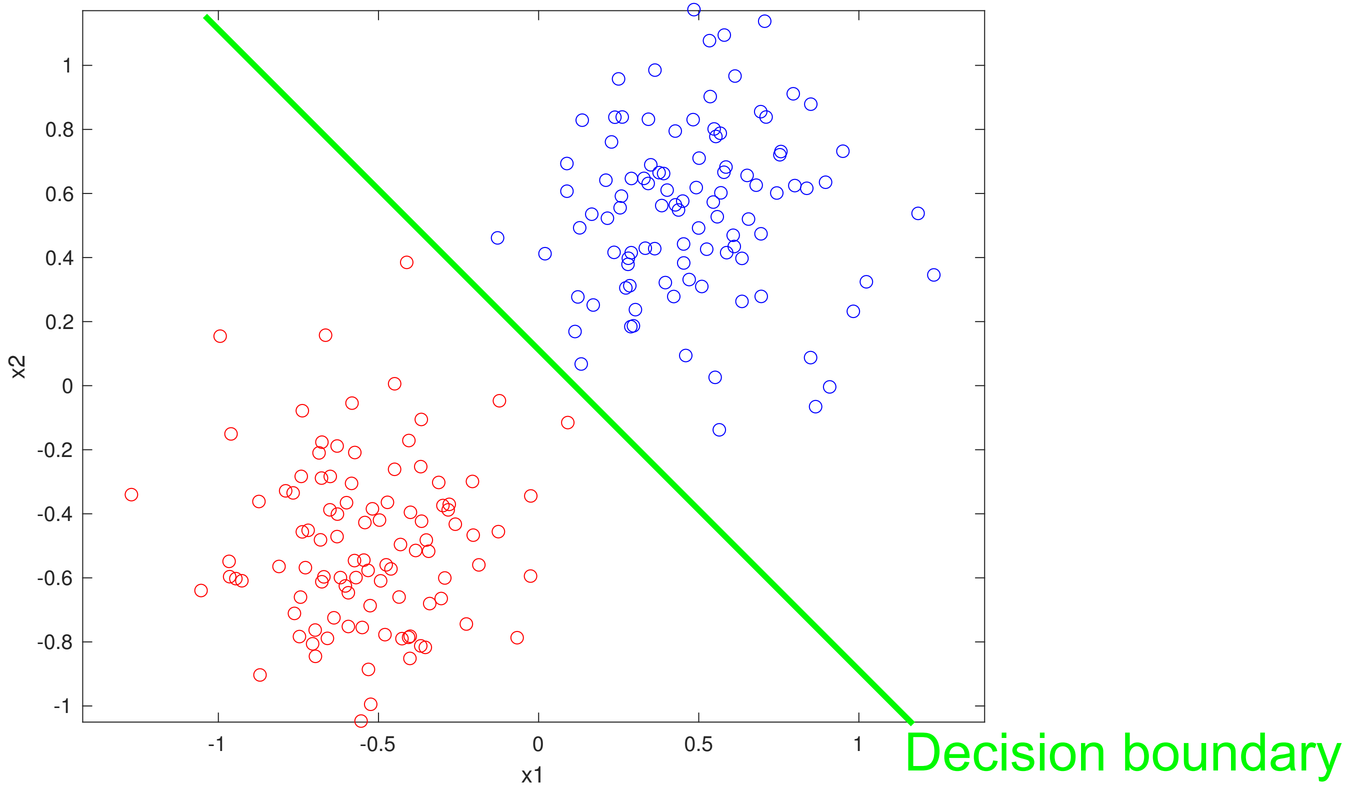


- Single Layer Perceptrons
- Multiple Layer Perceptrons
- Convolutional Neural Nets
- Transformers

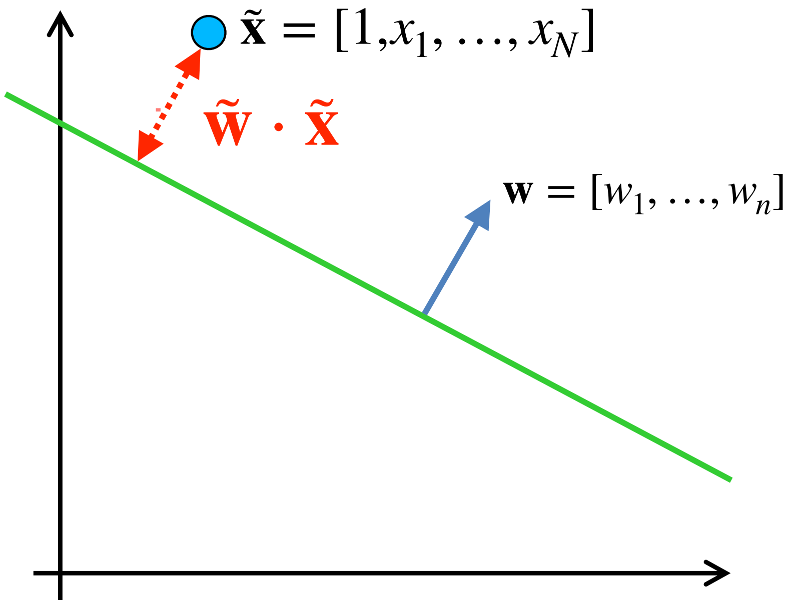
Linear Binary Classification



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.
- Extends naturally to an arbitrary number of dimensions

Binary Classification in N Dimensions



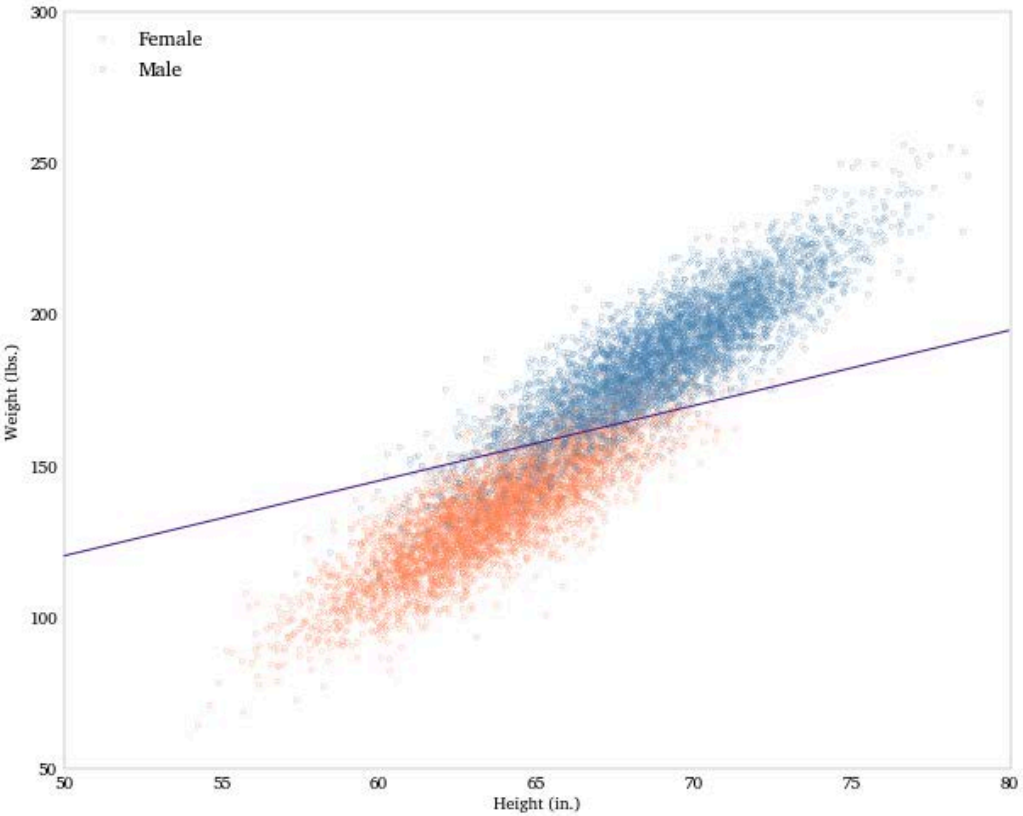
Hyperplane: $\mathbf{x} \in R^N, \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} = 0,$
with $\tilde{\mathbf{x}} = [1 \,|\, \mathbf{x}]$.

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

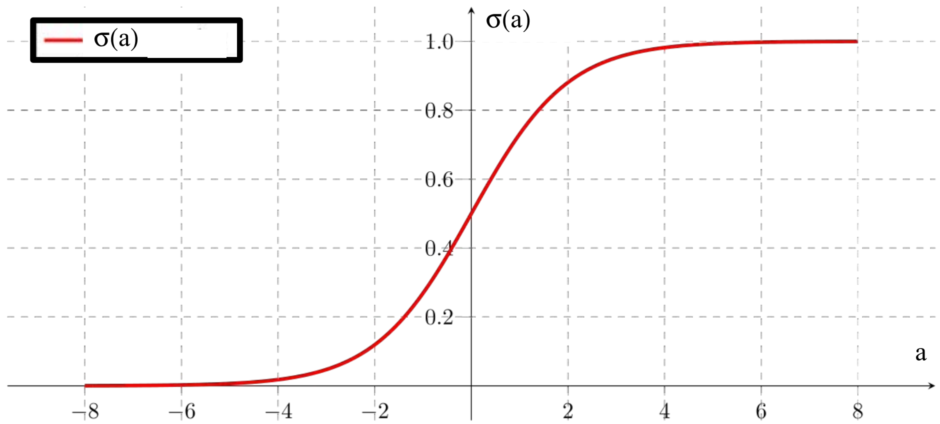
- for all or most positive samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} > 0$,
- for all or most negative samples $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}} < 0$.

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 $\{(\mathbf{x}_n, t_n)_{1 \leq n \leq N}\}$ minimize

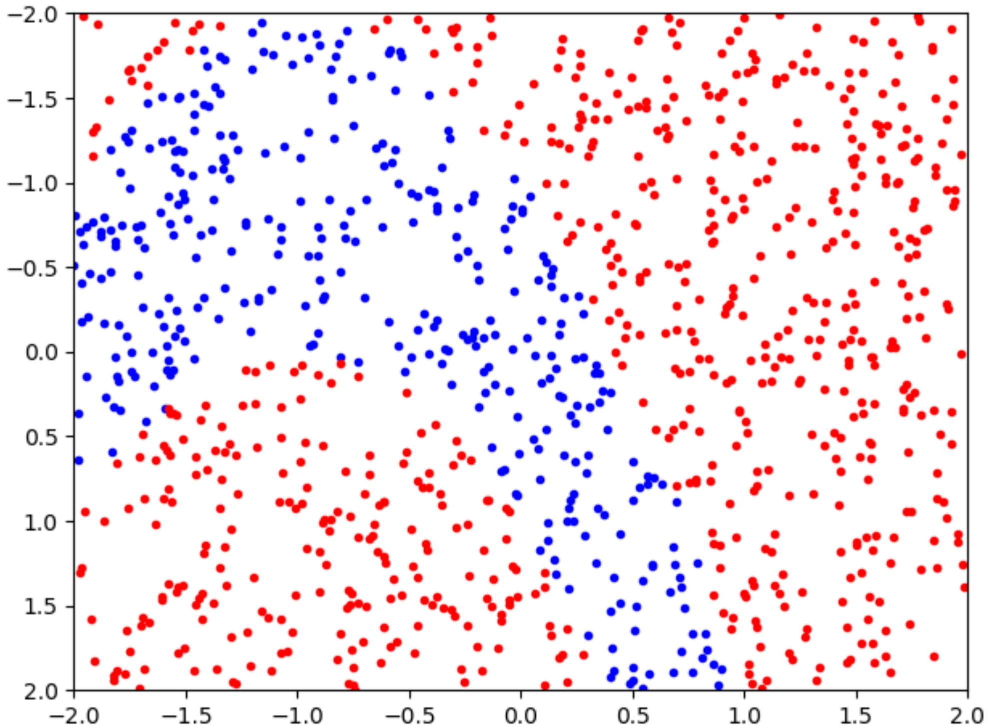
$$-\sum_n (t_n \ln y(\mathbf{x}_n) + (1 - t_n) \ln(1 - y(\mathbf{x}_n))$$

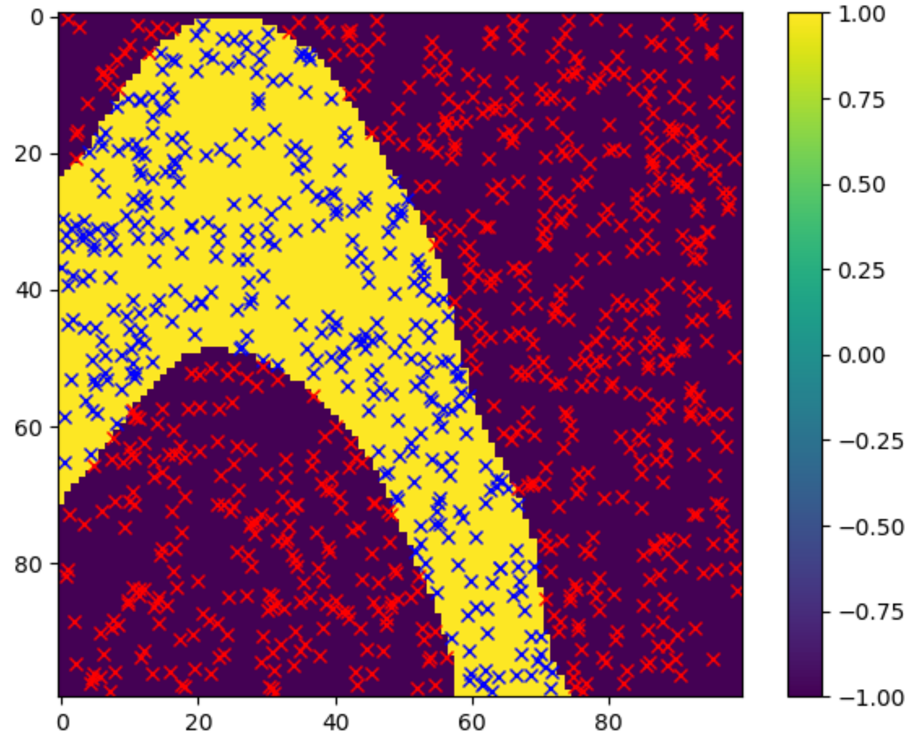
with respect to $\tilde{\mathbf{w}}$.

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

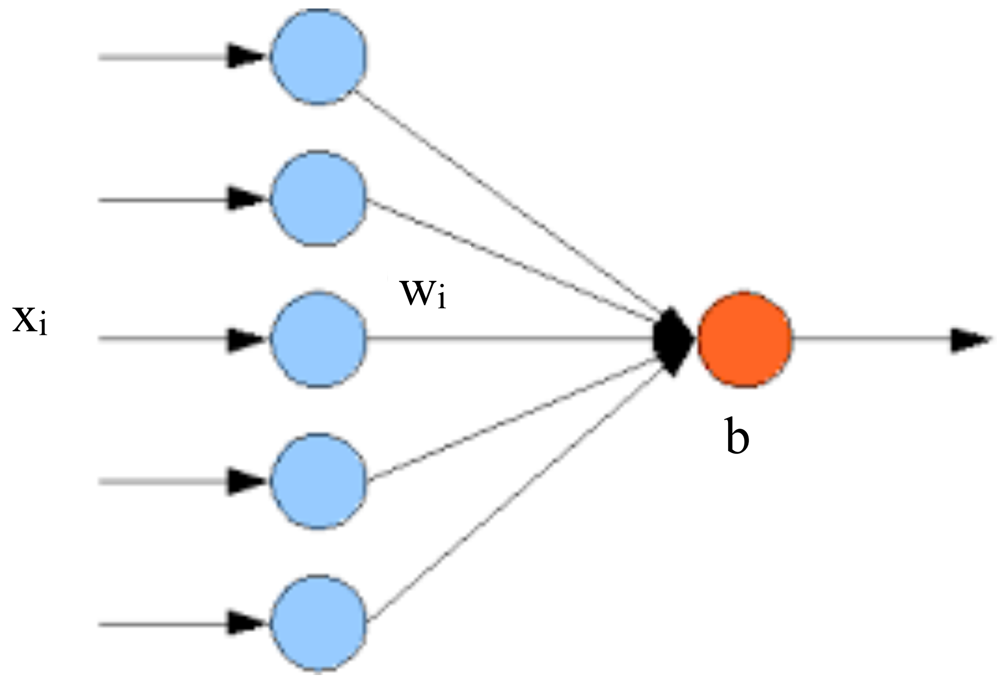
Negative: Otherwise

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

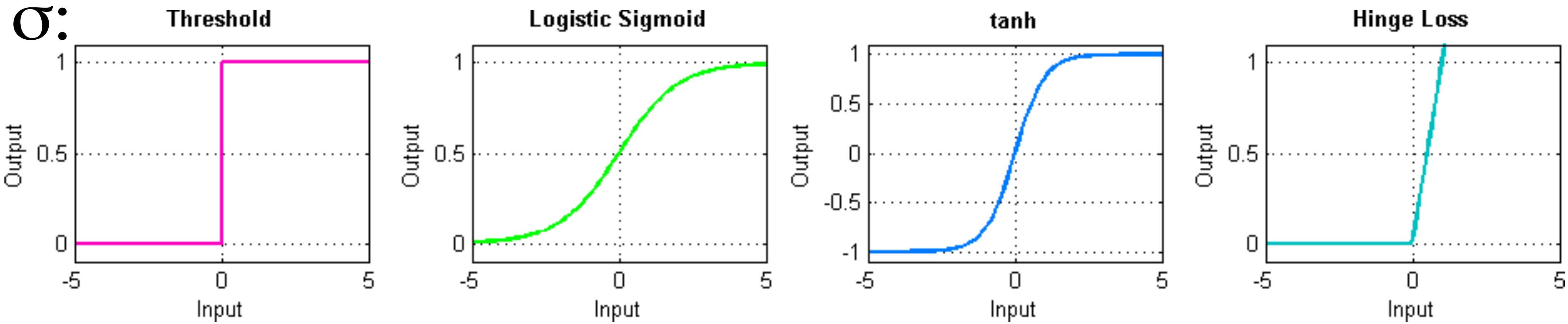


$$y(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

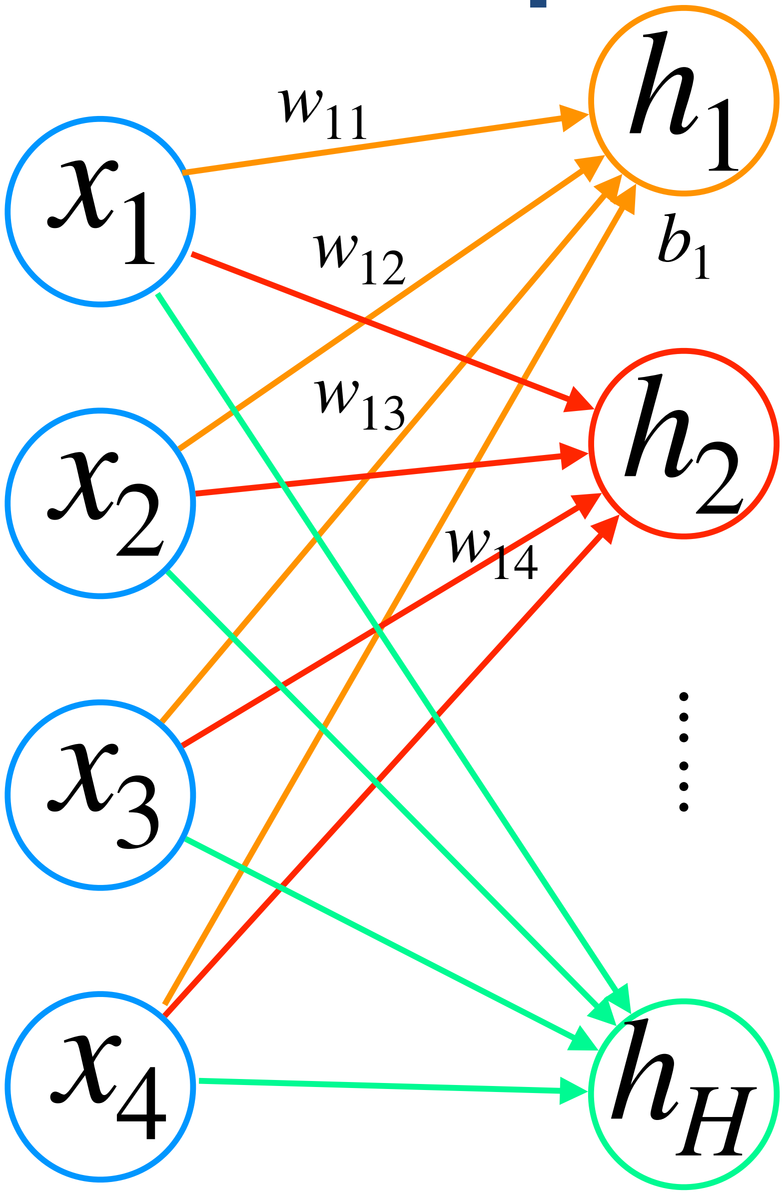
$$\mathbf{x} = \left[x_1, x_2, \ldots, x_n\right]^T$$

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

σ:



Repeating the Process



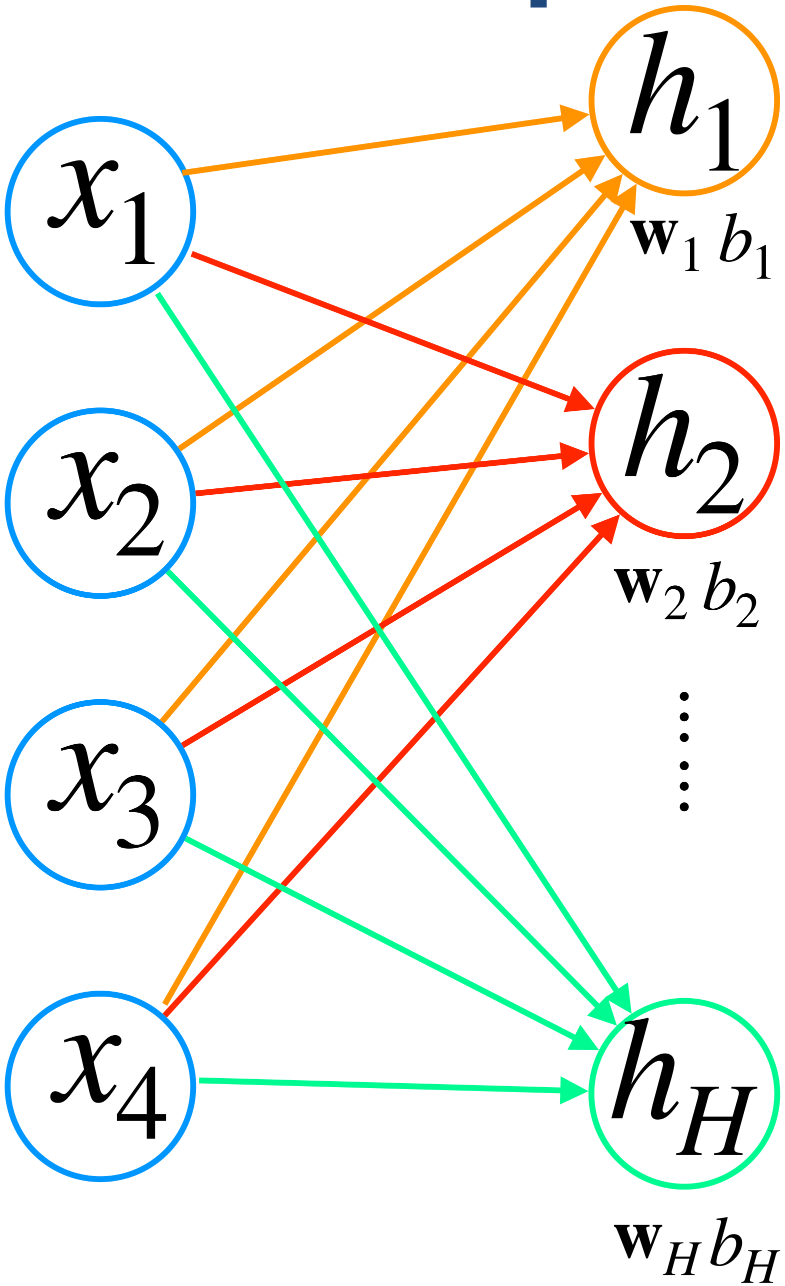
$$h_1 = \sigma(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$$
$$\mathbf{w}_1 = \left[w_{11}, w_{12}, w_{13}, w_{14}\right]^T$$

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

$$\vdots$$

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

Repeating the Process



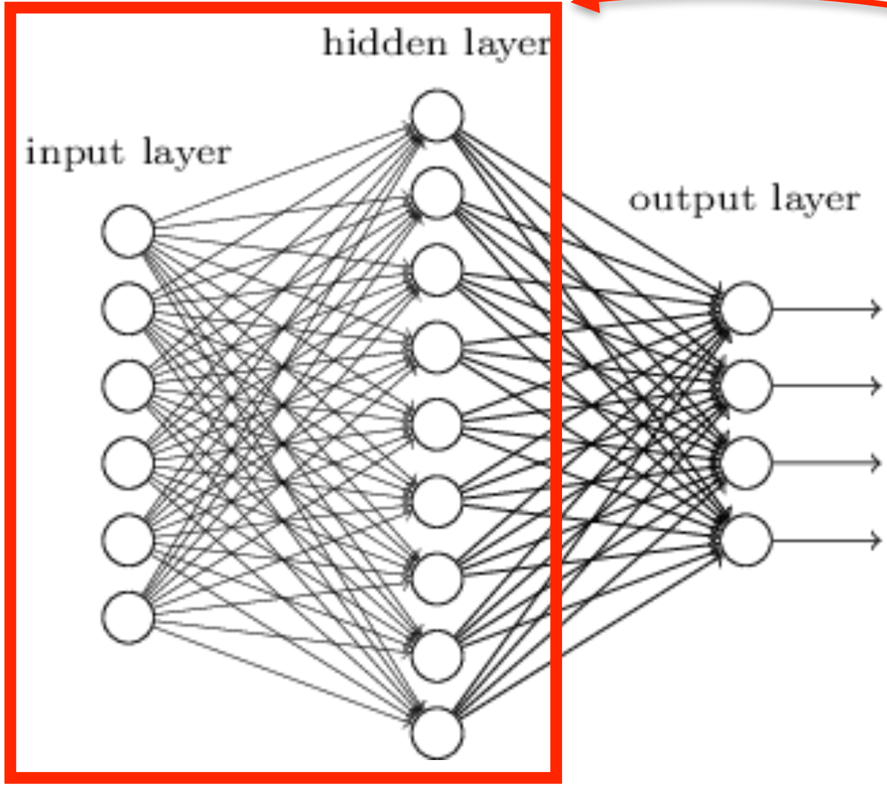
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \;,$$

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

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

EPFL

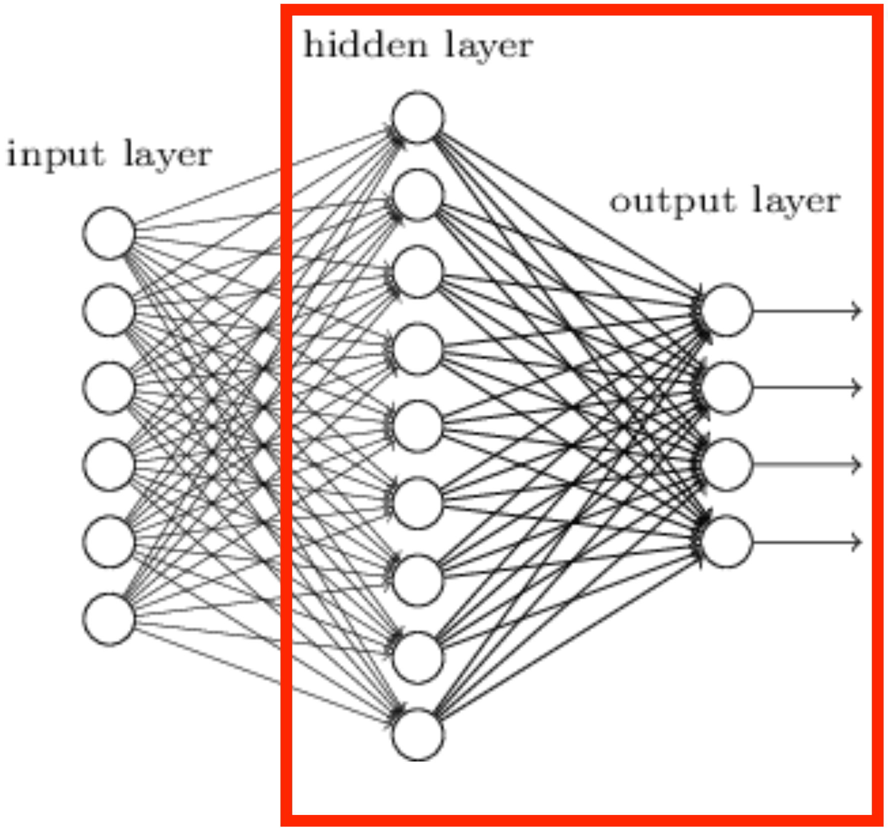
Multi-Layer Perceptron



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

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

Multi-Layer Perceptron

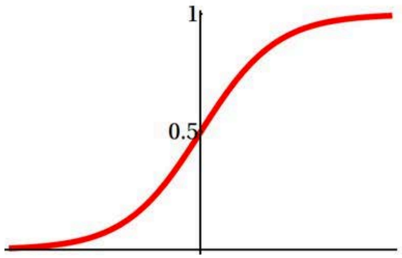


$$\mathbf{h} = \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y} = \sigma_2(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

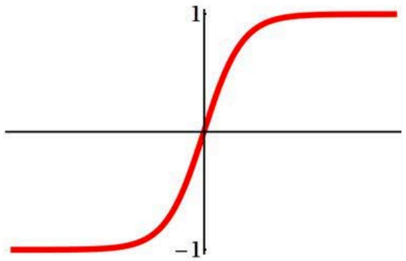
- The process can be repeated several times to create a vector **h**.
- It can then be done again to produce an output **y**.

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

Activation Functions



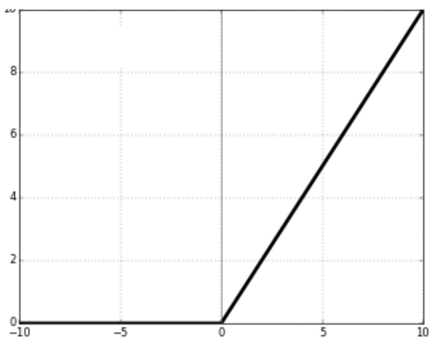
sigm



tanh

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

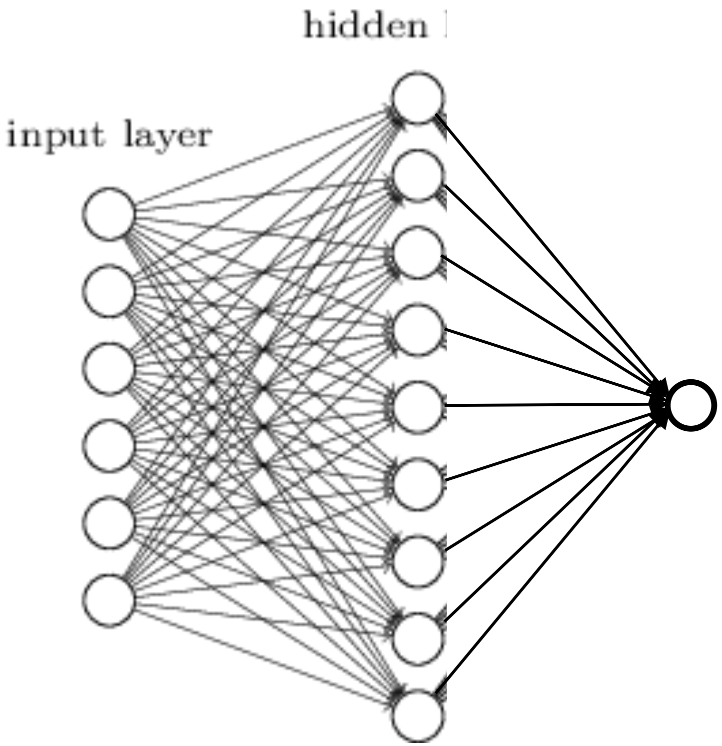
$$\text{tanh:} \quad \sigma(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



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

- One problem with the sigmoid and tanh functions is that when the argument is not close to zero the gradients vanish.
- Empirically, replacing the them by ReLu has significantly boosted performance in many cases.

Binary Case



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

$$y = \sigma(\mathbf{w}_2 \mathbf{h} + b_2)$$

In this case w_2 is vector.

Training

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

- We write

$$y_n = \sigma_2(\mathbf{w}_2(\sigma_1(\mathbf{W}_1 \mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2) \in [0.1]$$

- We want to minimize the binary cross entropy

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

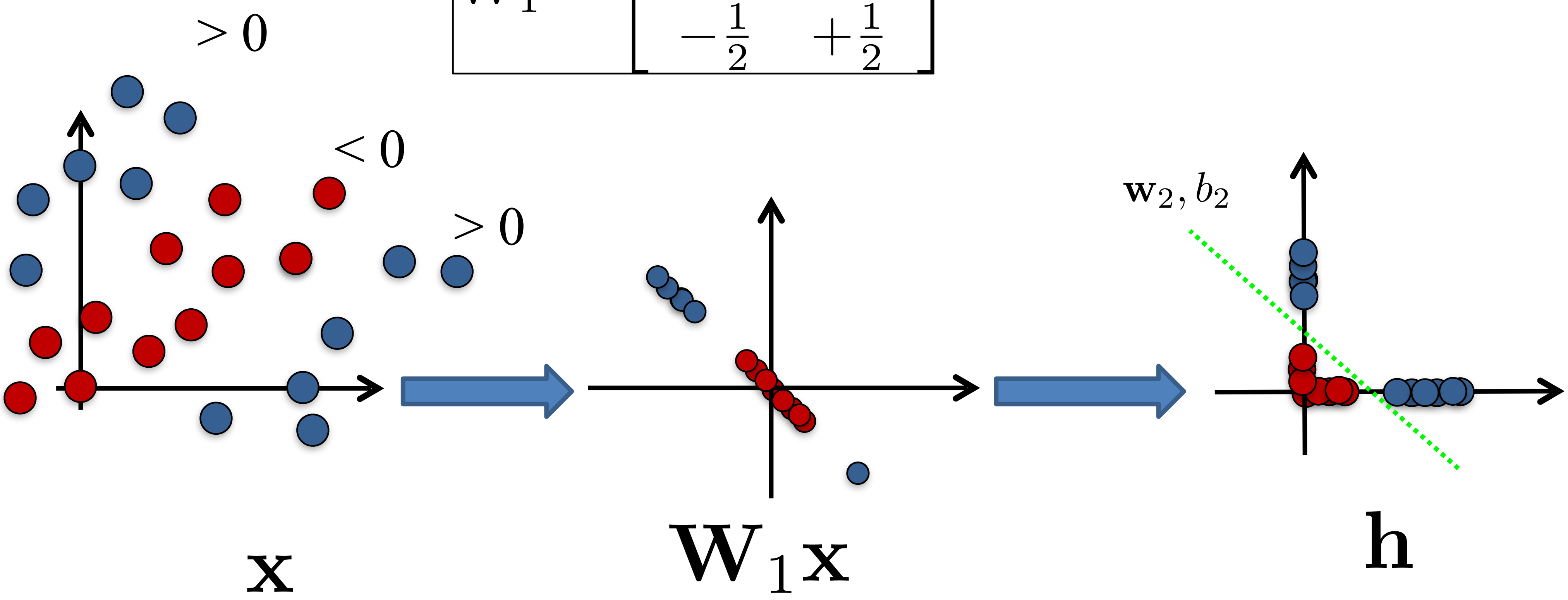
$$E_n(\mathbf{W}_1, \mathbf{w}_2, \mathbf{b}_1, \mathbf{b}_2) = -\left(t_n \ln(y_n) + (1 - t_n)\ln(1 - y_n)\right) \ ,$$

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

- E is a differentiable function and this can be done using a gradient-based technique.

ReLu Behavior

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

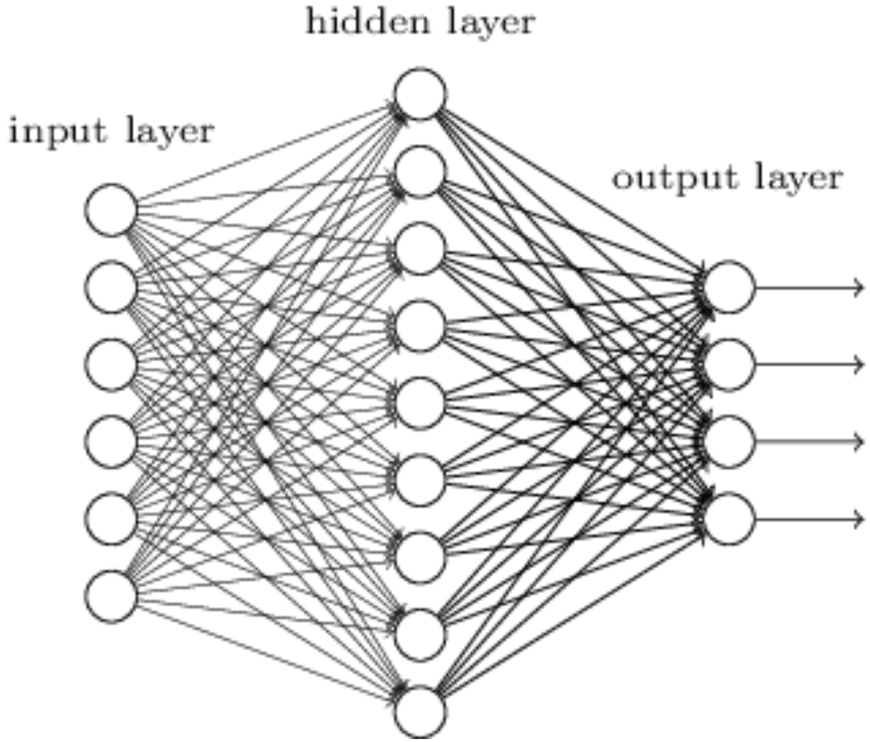


$$\mathbf{h} = ReLu(\mathbf{W}_1\mathbf{x})$$

$$y = \mathbf{w_2} . \mathbf{h} + b_2$$

EPFL

Geometric Interpretation

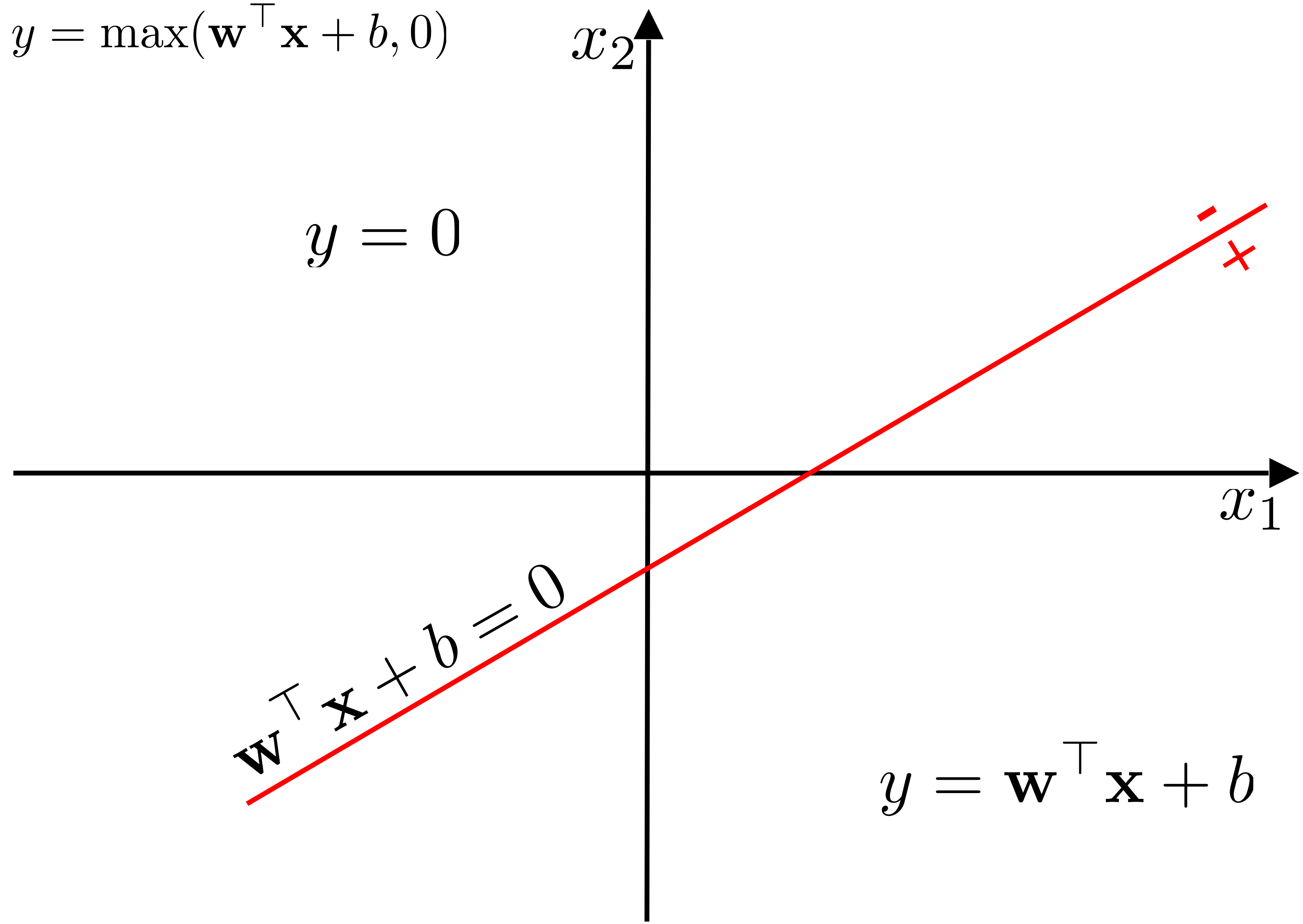


$$\mathbf{h} = \sigma_1(\mathbf{W}_1\mathbf{x}_n + \mathbf{b}_1)$$

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

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

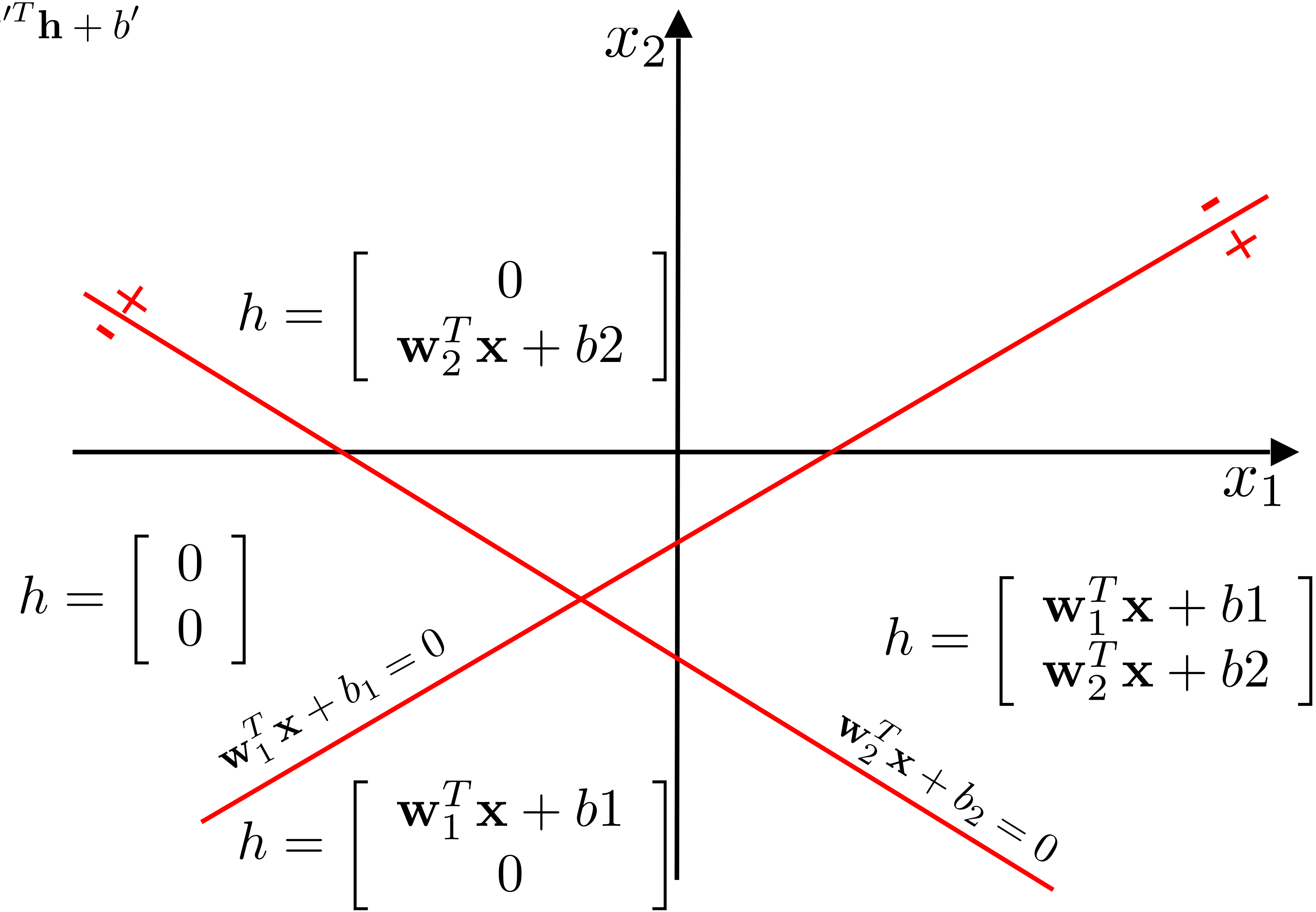
One Single Hyperplane



Two Hyperplanes

$$\mathbf{h} = \max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

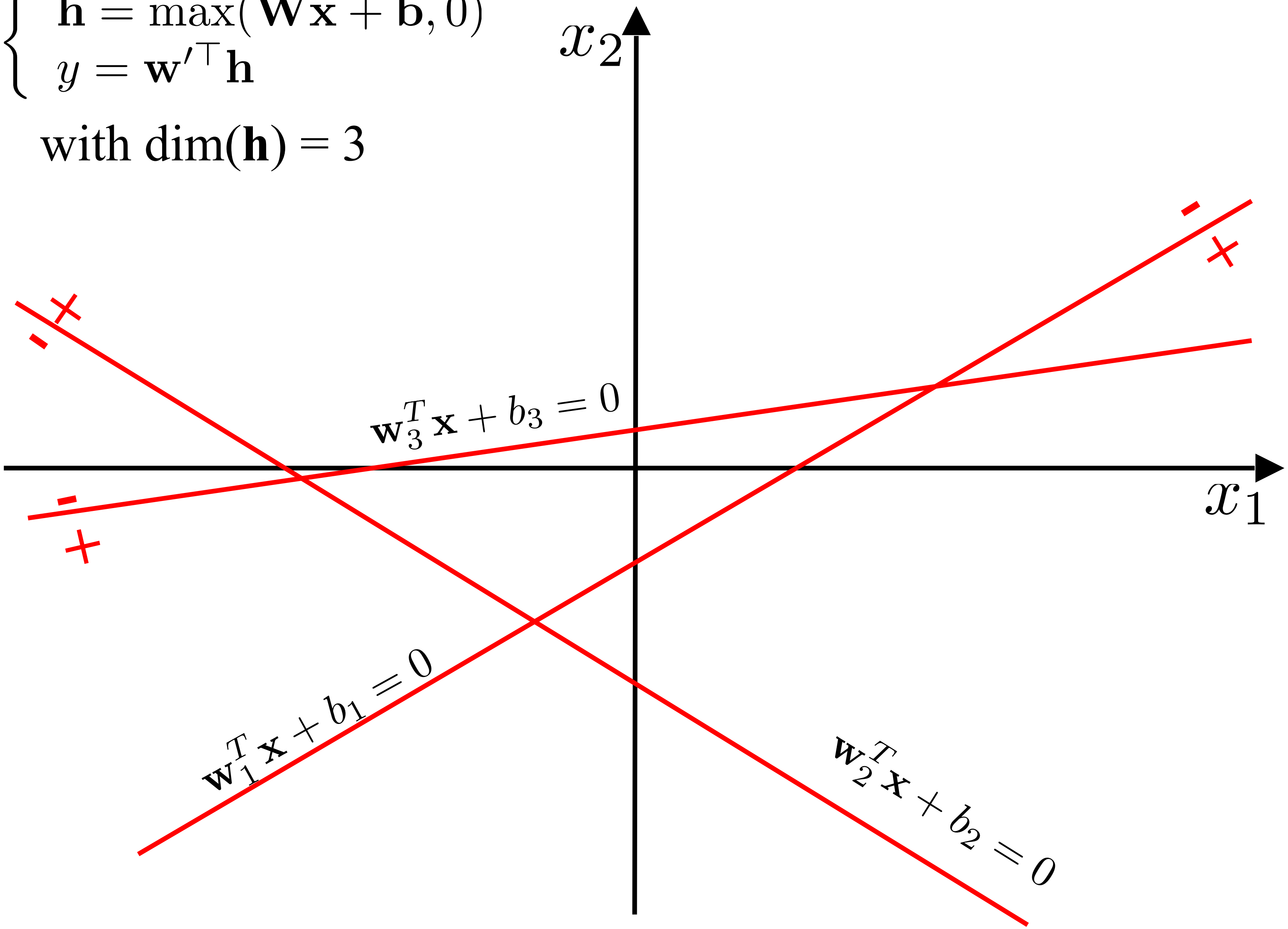
$$y = \mathbf{w}'^T \mathbf{h} + b'$$



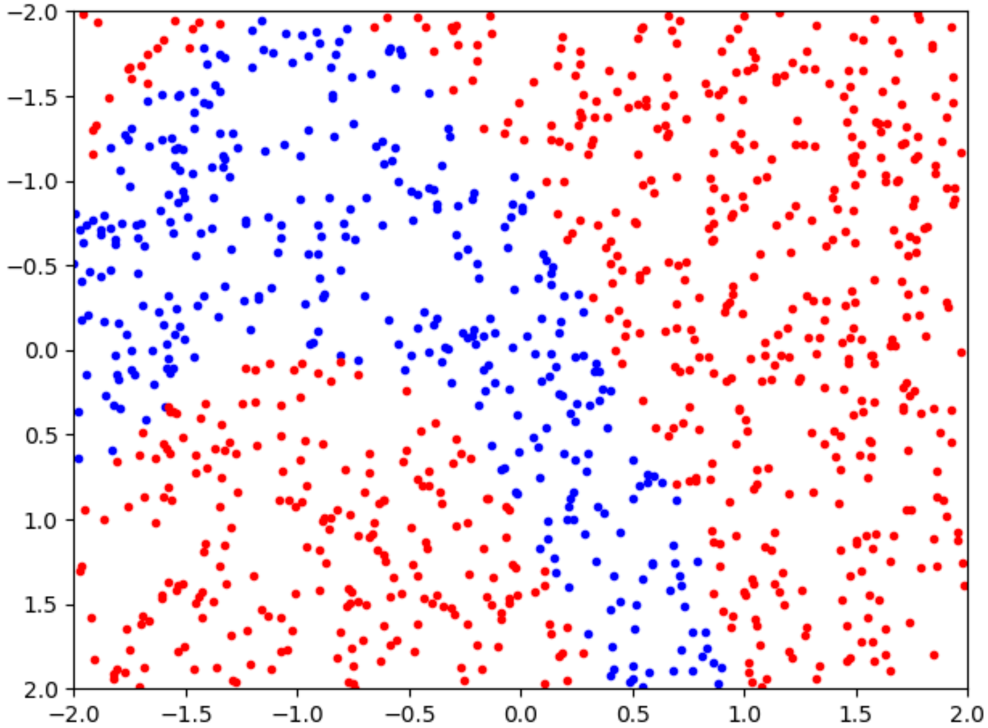
Three Hyperplanes

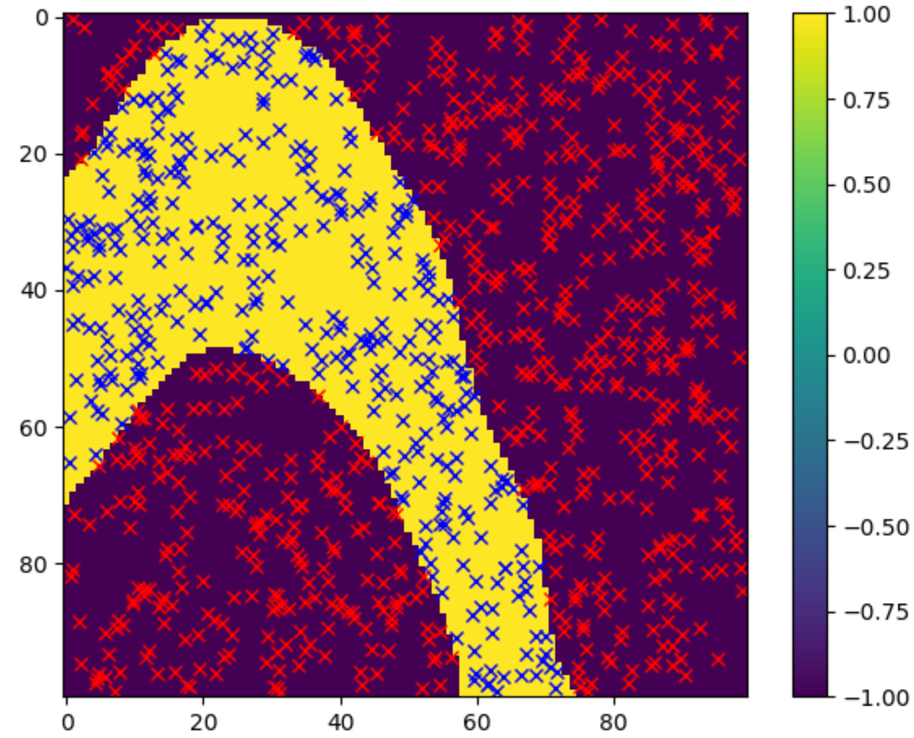
$$\begin{cases} \mathbf{h} = \max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) \\ y = \mathbf{w}'^{\top}\mathbf{h} \end{cases}$$

with $\dim(\mathbf{h}) = 3$



Classification as Regression





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

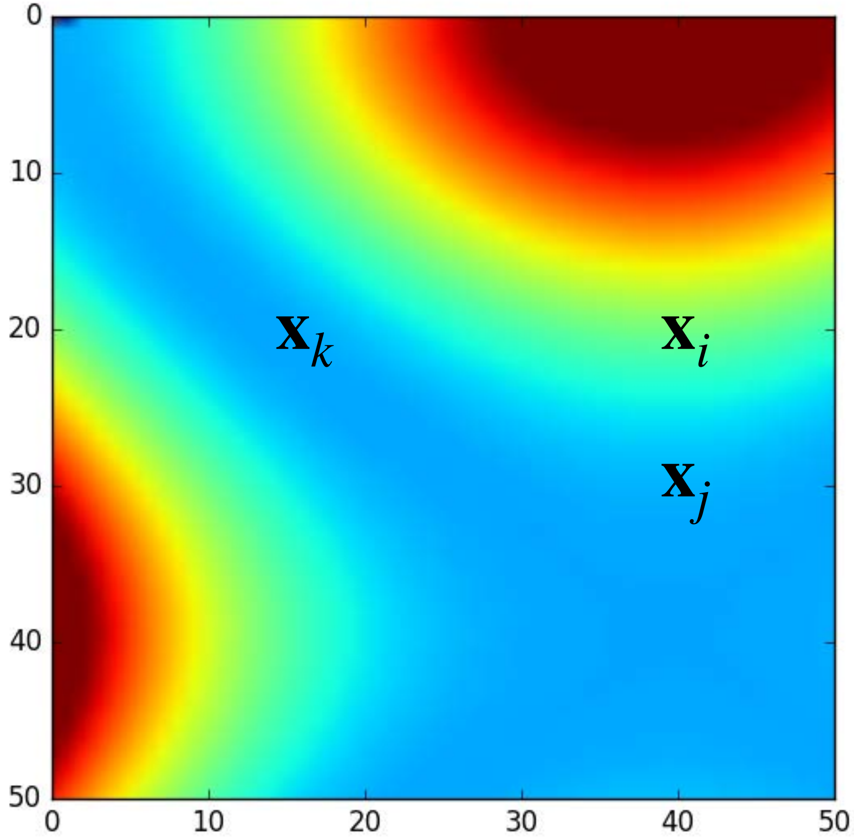
$y(\mathbf{x}; \tilde{\mathbf{w}})$ is now a non-linear function implemented by the network.

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

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

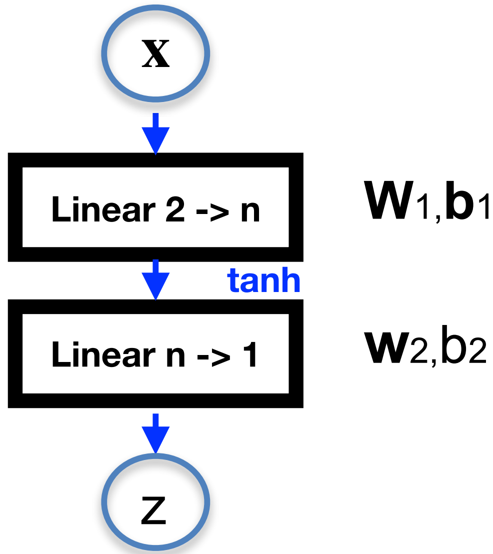
EPFL

Classification as Regression



$$P(\mathbf{x} \text{ in positive class}) = \begin{cases} 1.0 \text{ if } r(\mathbf{x}) < 0.5 \\ 0.0 \text{ otherwise} \end{cases}$$

$$r(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



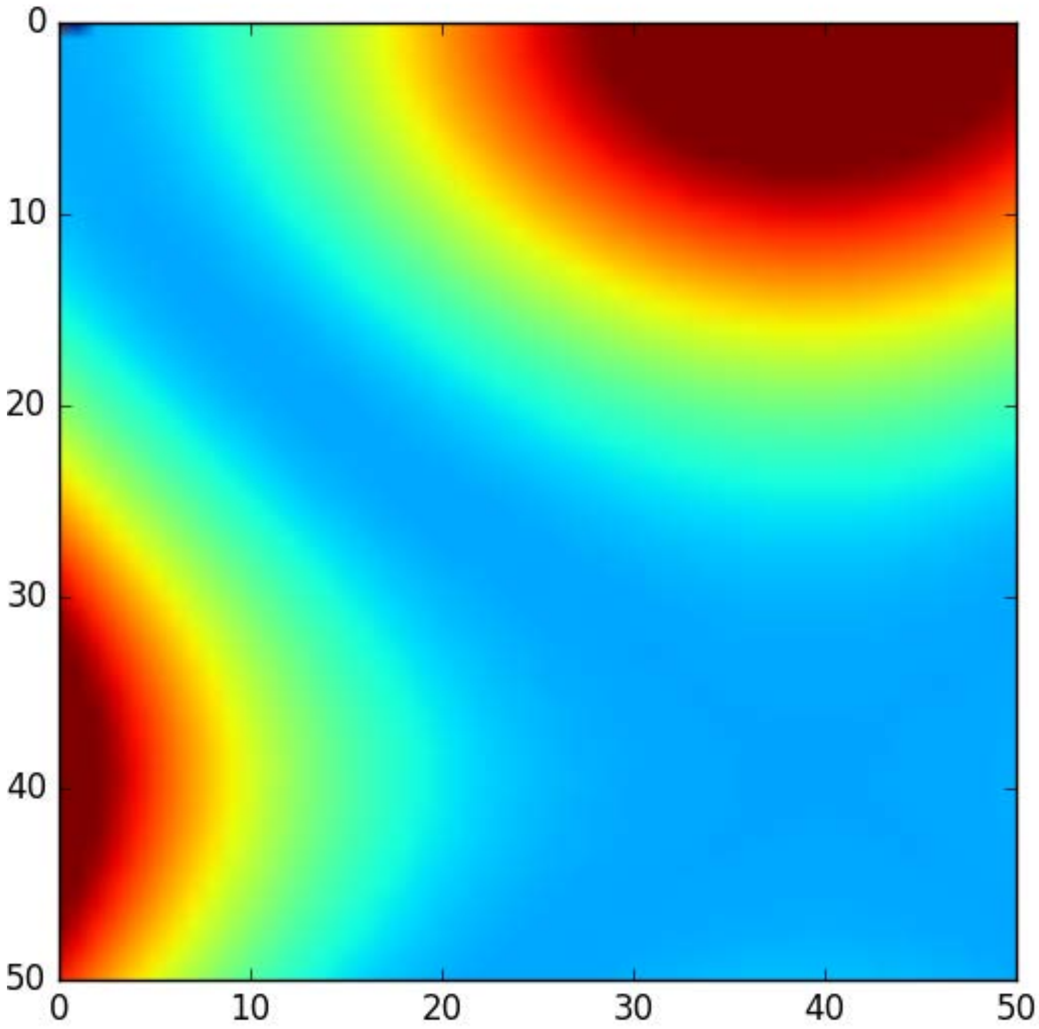
$$y(\mathbf{x}, \tilde{\mathbf{w}}) = \mathbf{w}_2 \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + b_2$$

Problem statement: Given $(\{\mathbf{x}_1, z_1 = r(\mathbf{x_1})\}, \ldots, \{\mathbf{x}_n, z_n = r(\mathbf{x_n})\})$, minimize

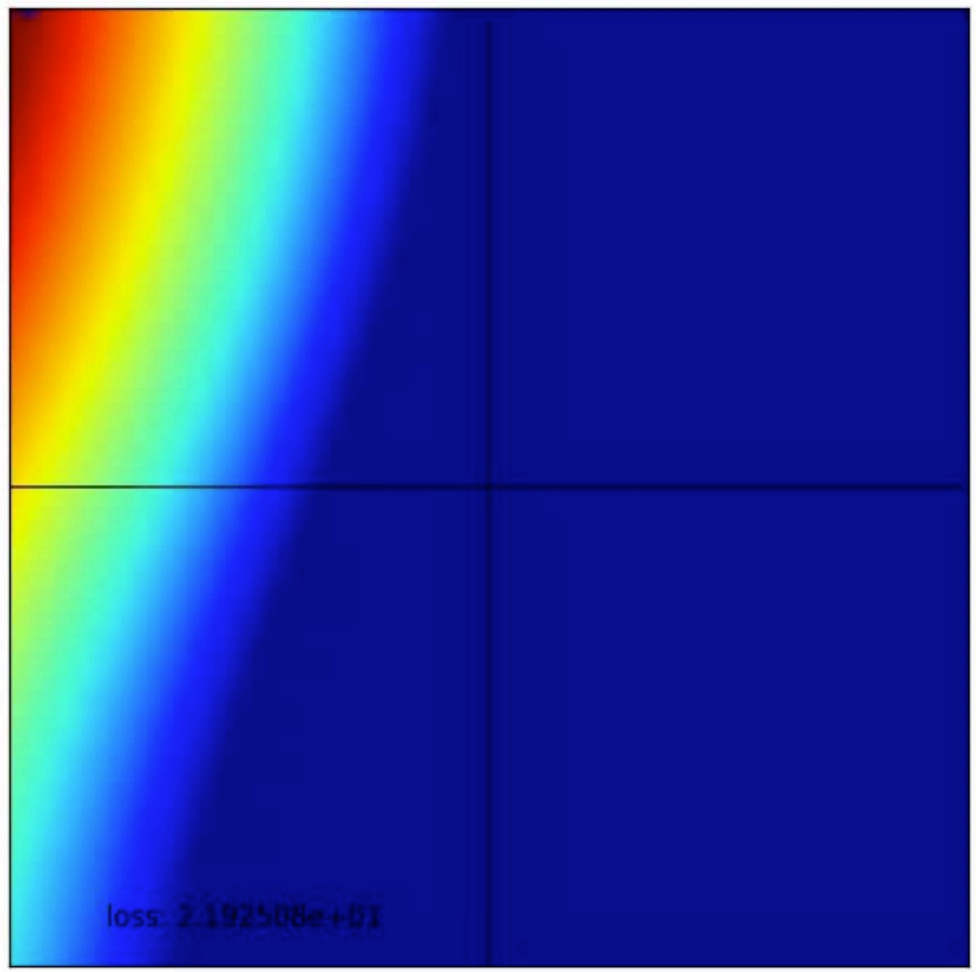
$$\sum_i (z_i - y(\mathbf{x}_i; \tilde{\mathbf{w}})^2)$$

w.r.t. $\tilde{\mathbf{w}}$.

Regressing the Rosenbrock Function



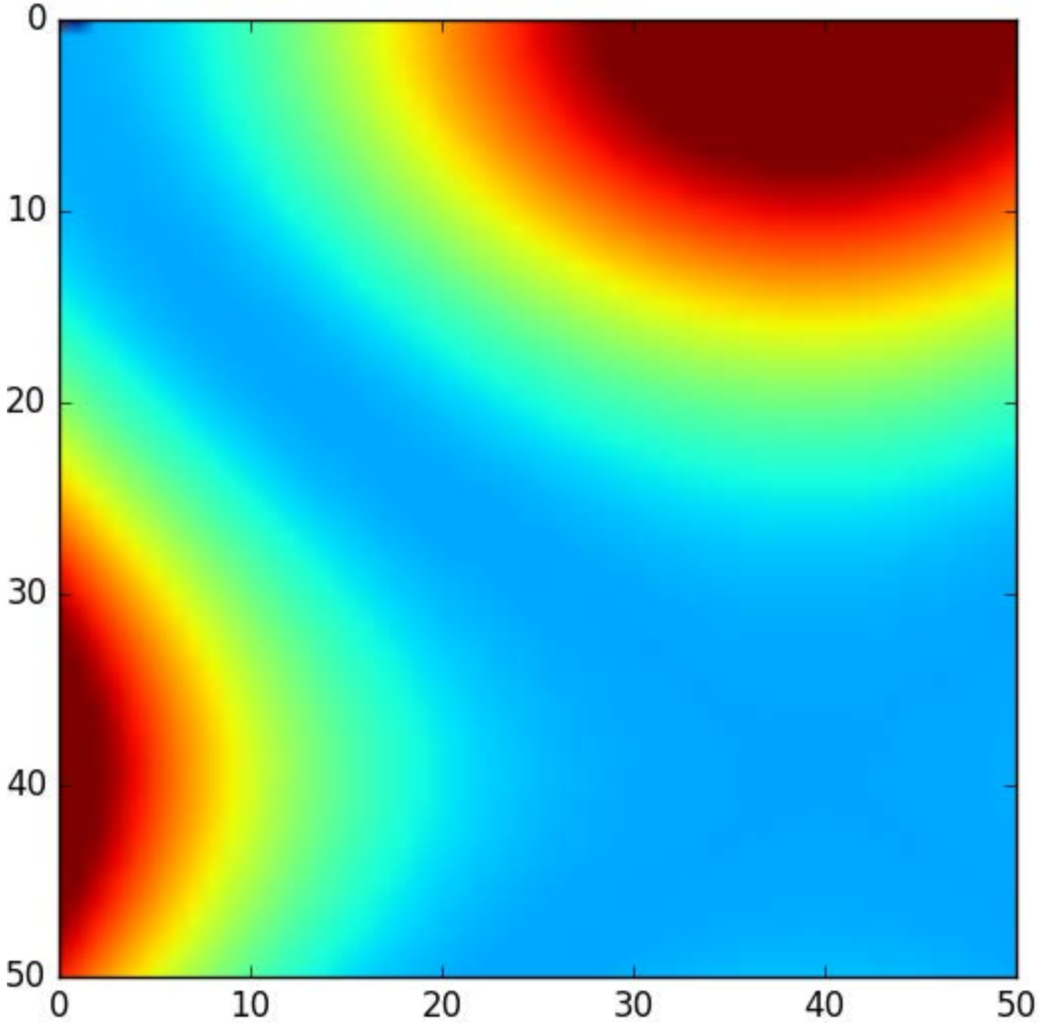
$$z = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



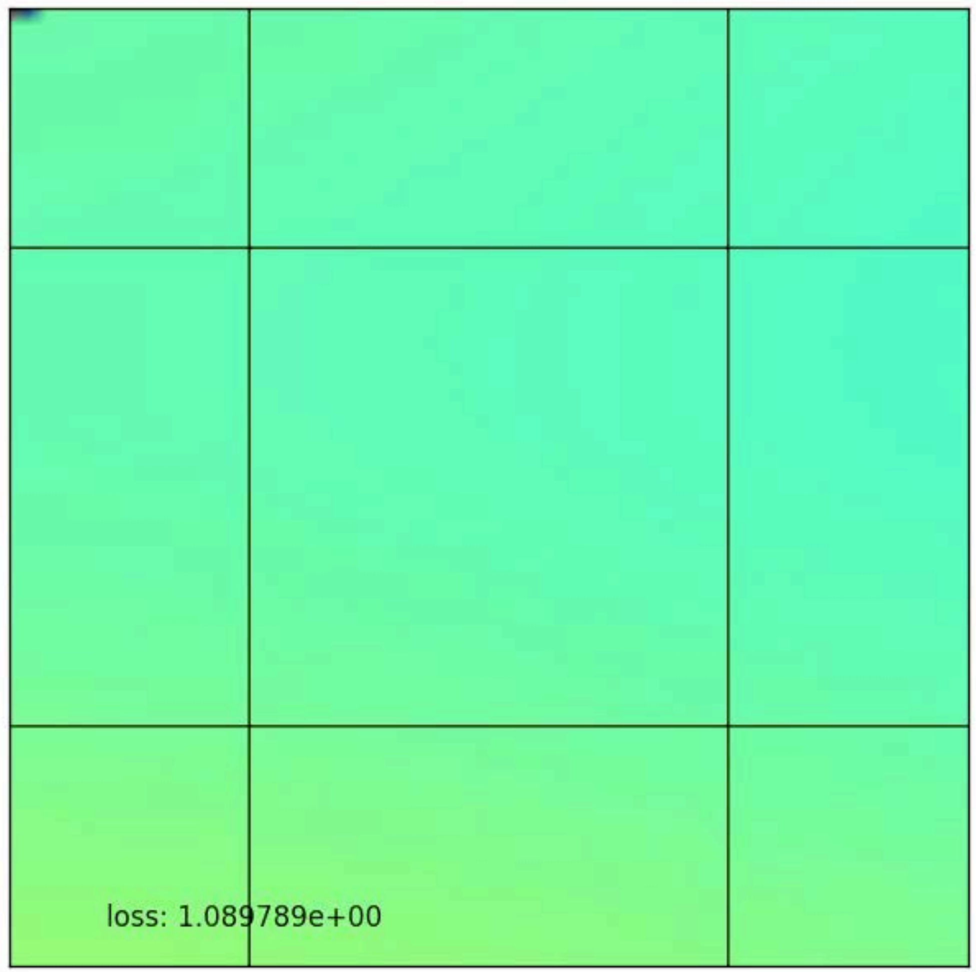
3-node hidden layer

—> 3 nodes is not quite enough.

Regressing the Rosenbrock Function



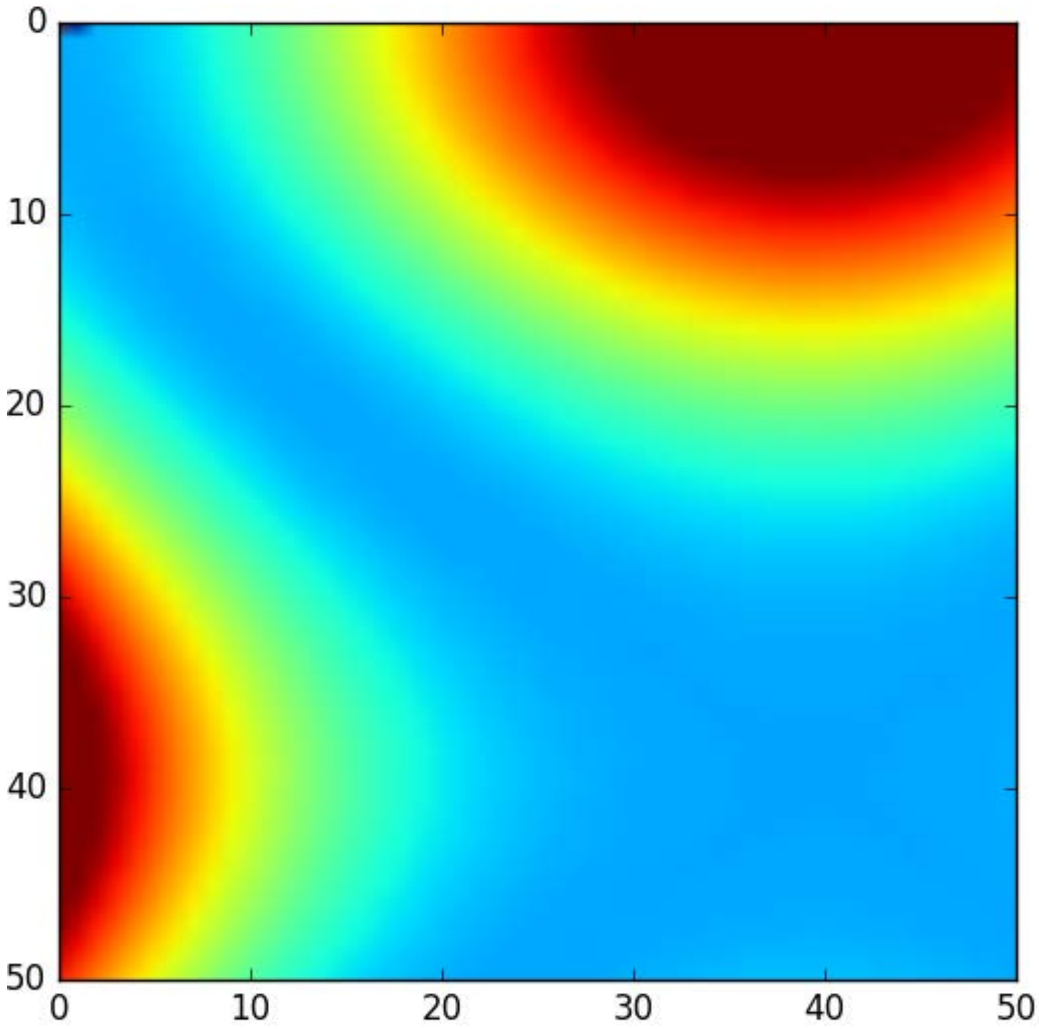
$$z = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



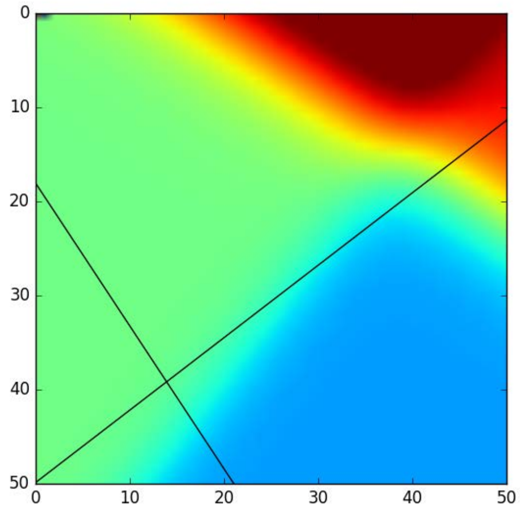
4-node hidden layer

—> 4 nodes is better.

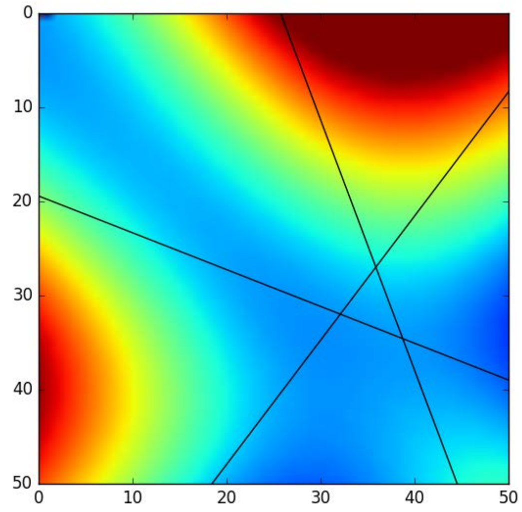
Adding more Nodes



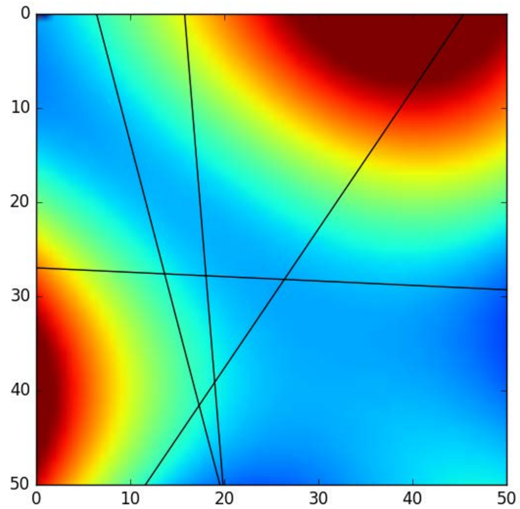
$$z = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



2 nodes -> loss 3.02e-01



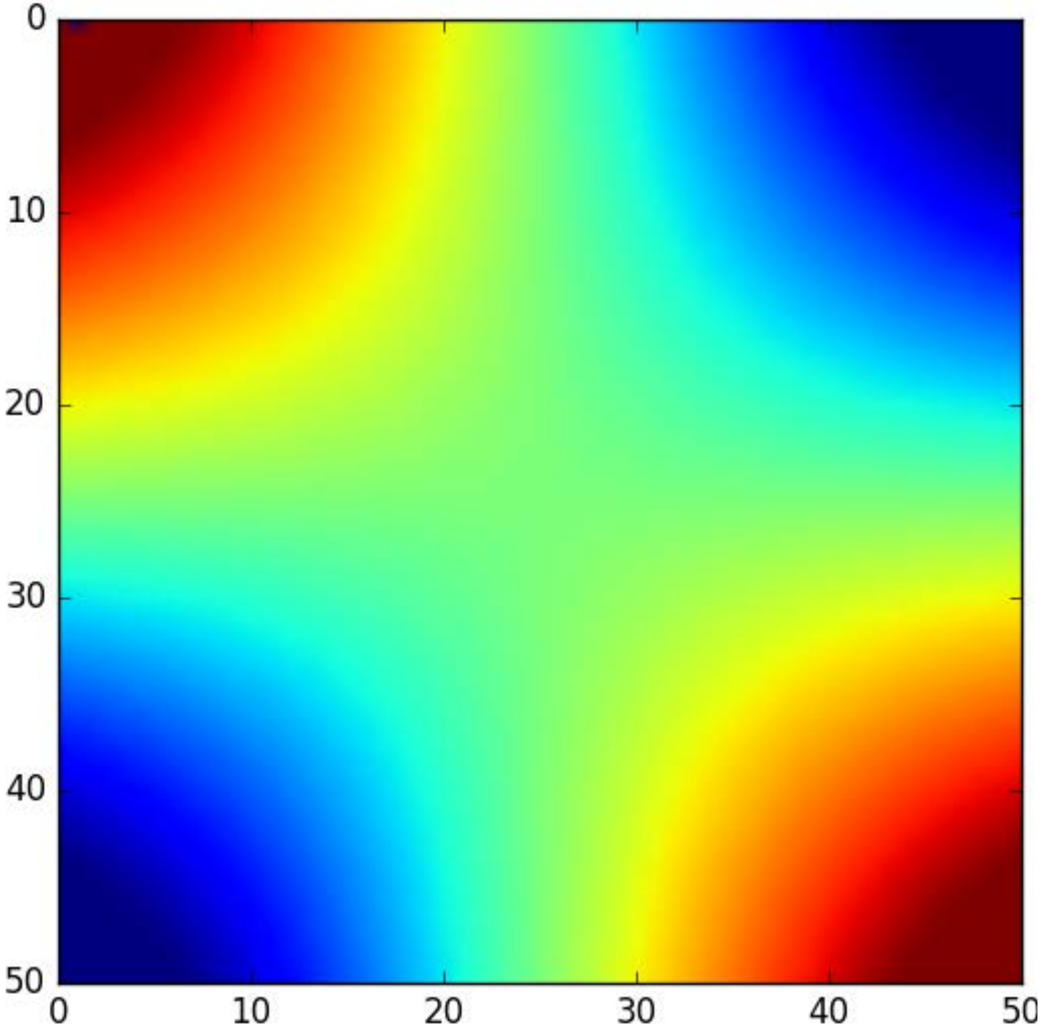
3 nodes -> loss 2.08e-02



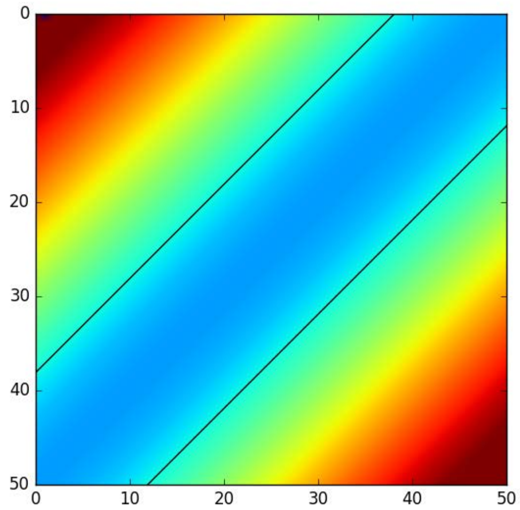
4 nodes -> loss 8.27e-03

EPFL

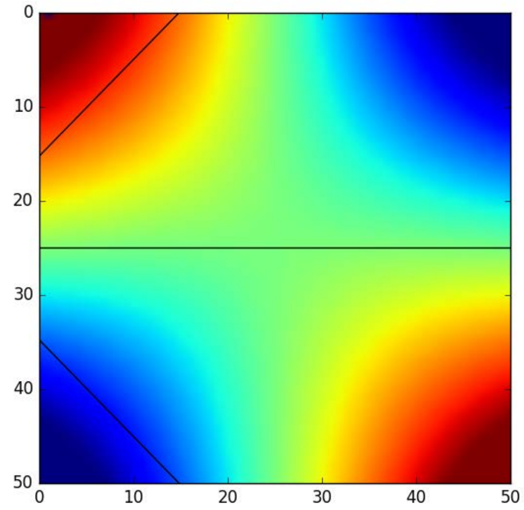
Adding more Nodes



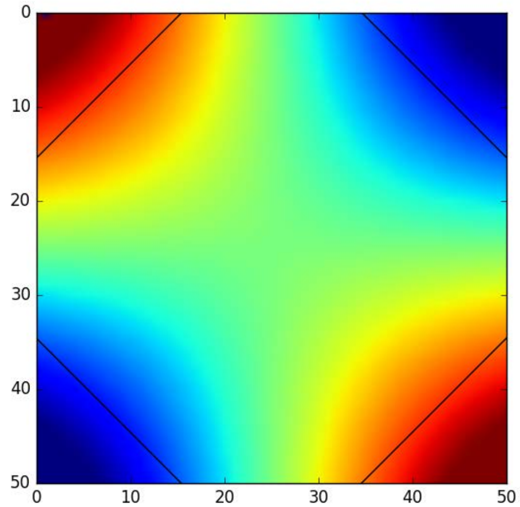
$$z = sin(x)sin(y)$$



2 nodes -> loss 2.61e-01



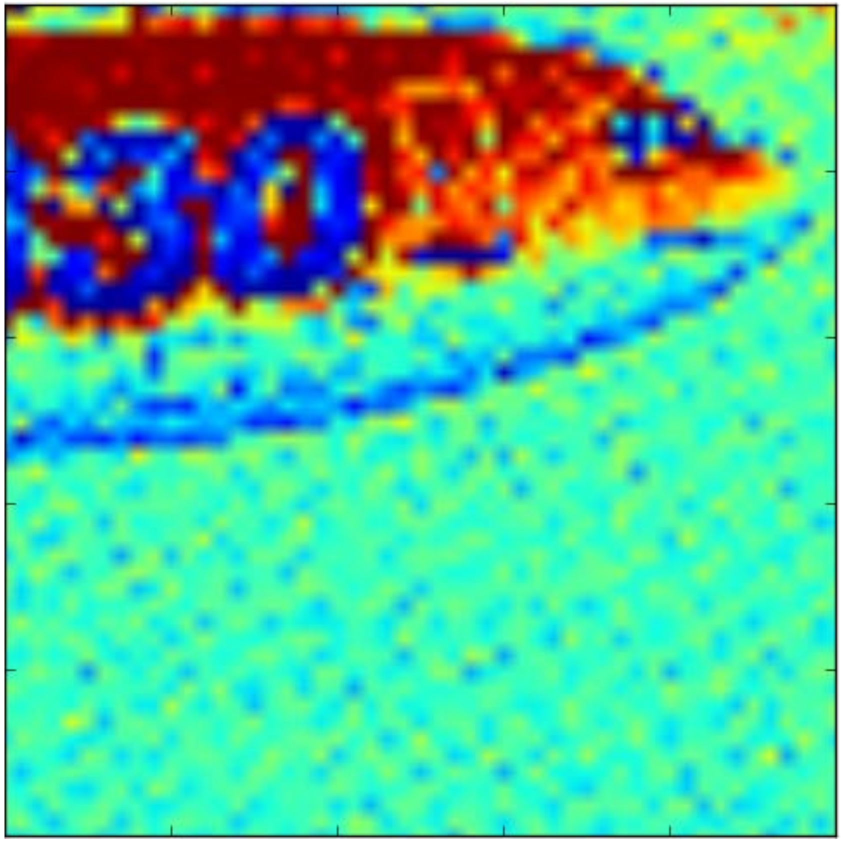
3 nodes -> loss 2.51e-04



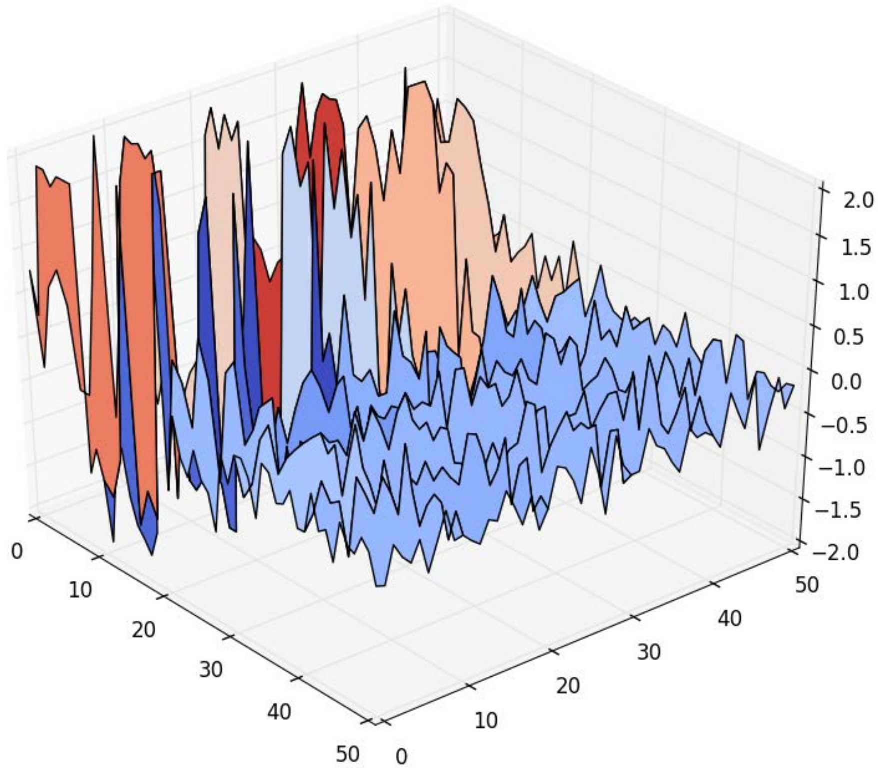
4 nodes -> loss 3.07e-07

EPFL

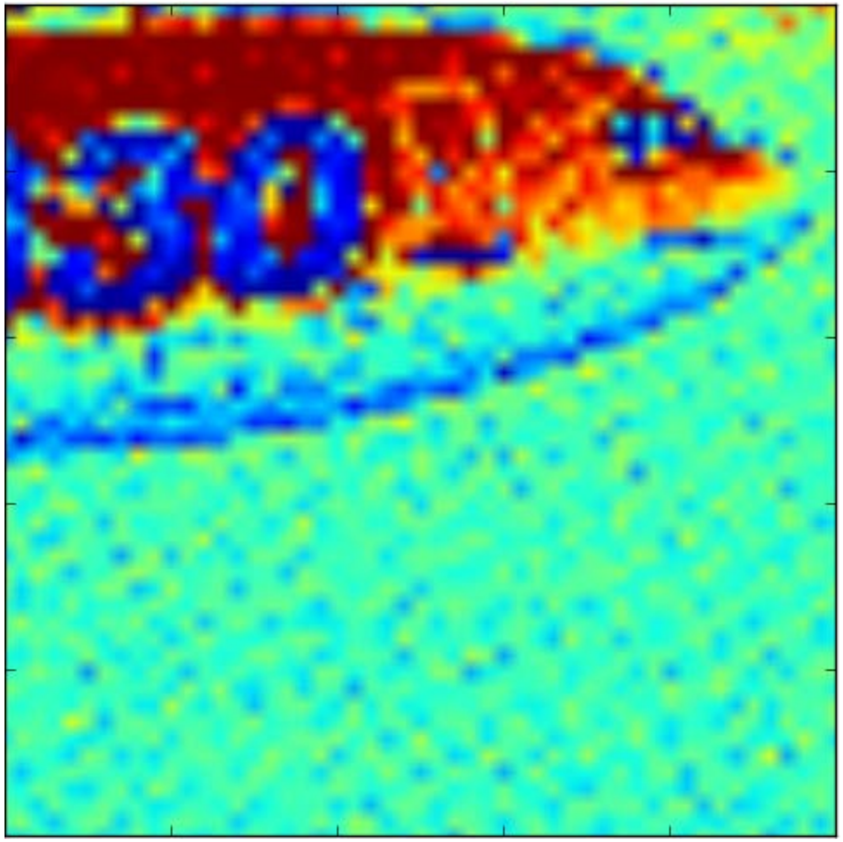
More Complex Surface



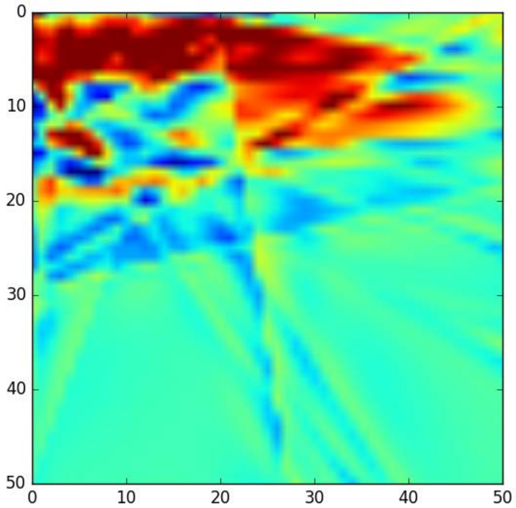
$$I = f(x, y)$$



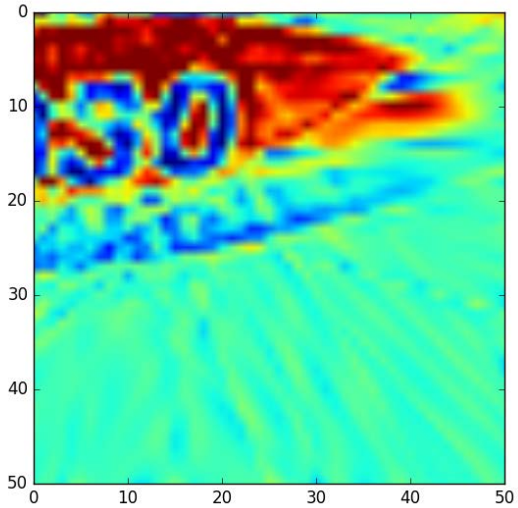
More Complex Surface



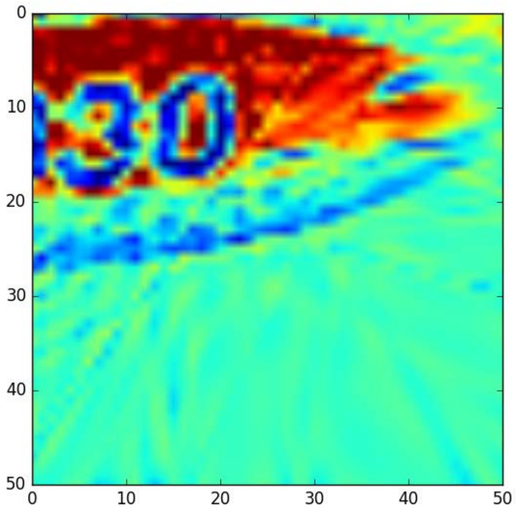
$$I = f(x, y)$$



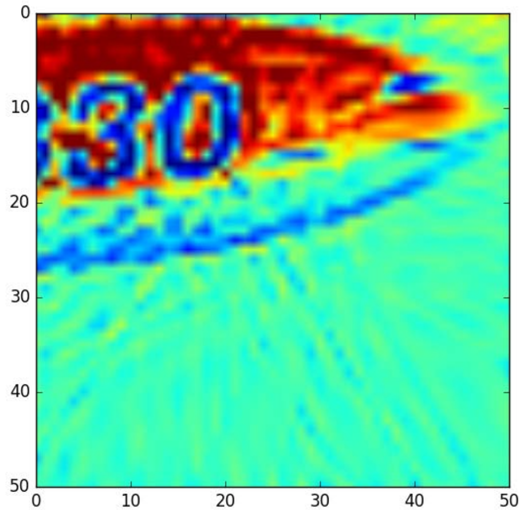
50 nodes -> loss 3.65e-01



100 nodes -> loss 2.50e-01



125 nodes -> loss 2.40e-01



300 nodes -> loss 1.92e-01

Universal Approximation Theorem

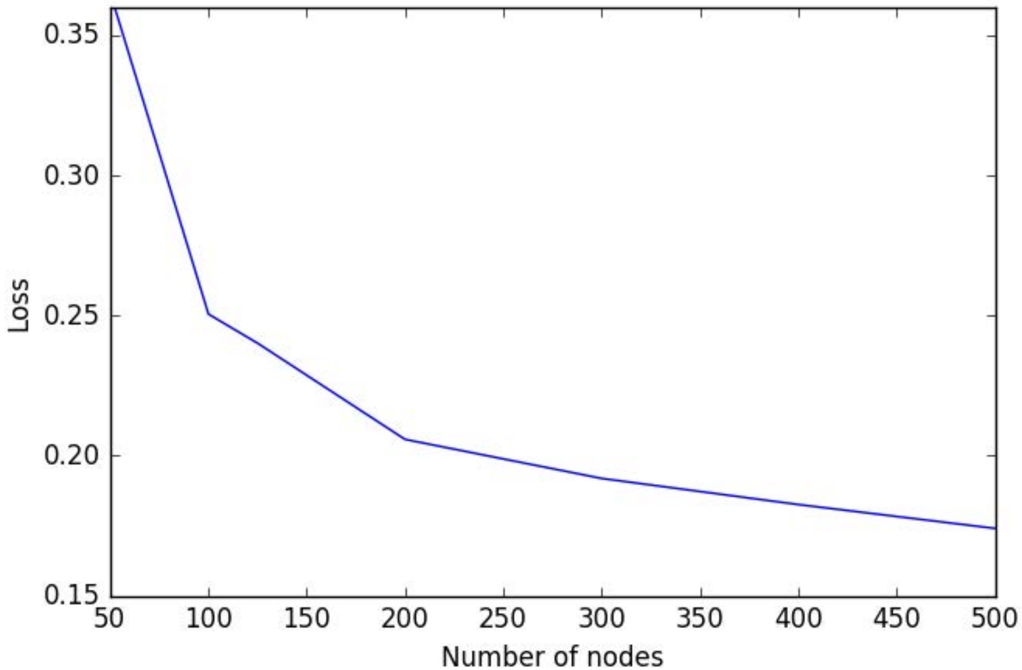
A feedforward network with a linear output layer and at least one hidden layer with any 'squashing' activation function (e.g. logistic sigmoid) can approximate any Borel measurable function (from one finite-dimensional space to another) with any desired nonzero error.

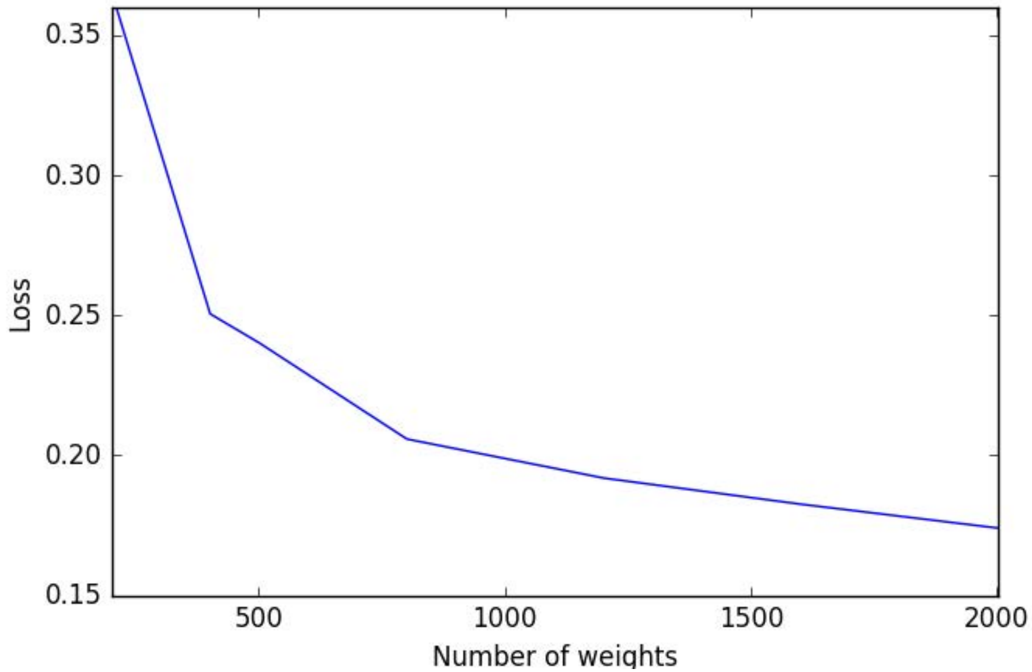
Any continuous function on a closed and bounded set of R^n is Borel-measurable.

—> 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

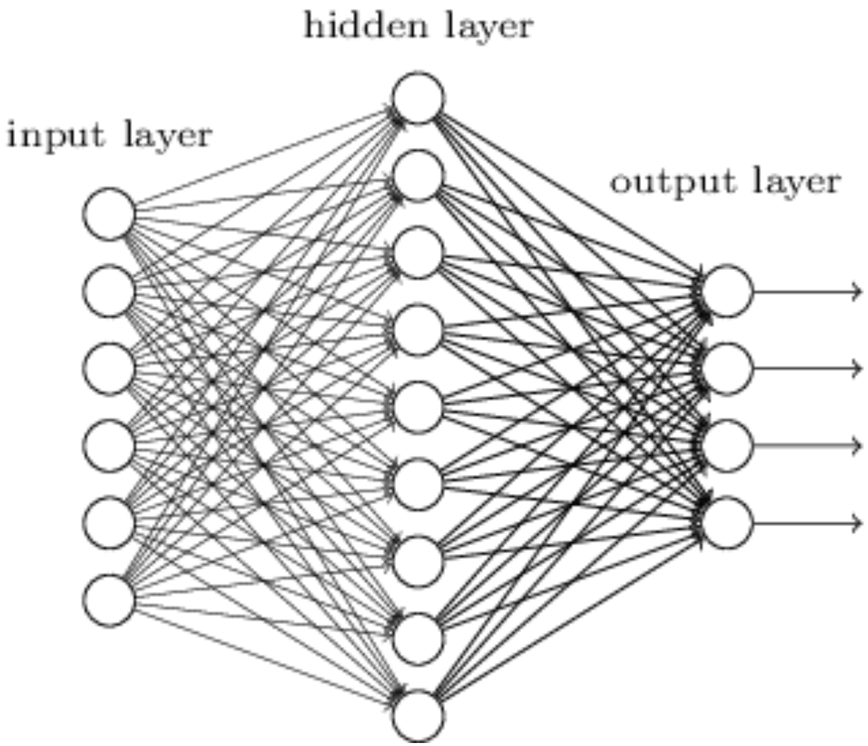




- It may take an exponentially large number of parameters for a good approximation.
- The optimization problem becomes increasingly difficult.

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

Multi-Class Case



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

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

In this case $\mathbf{W}_2$ is a matrix.

Training

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

$$\mathbf{y}_n = \mathbf{W}_2(\sigma_1(\mathbf{W}_1\mathbf{x}_n + \mathbf{b}_1)) + \mathbf{b}_2 \in R^K$$

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

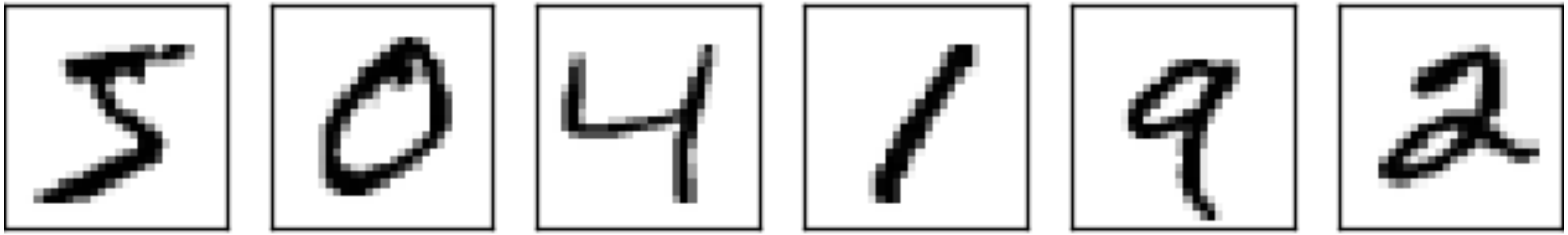
- We want to minimize the cross entropy

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

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

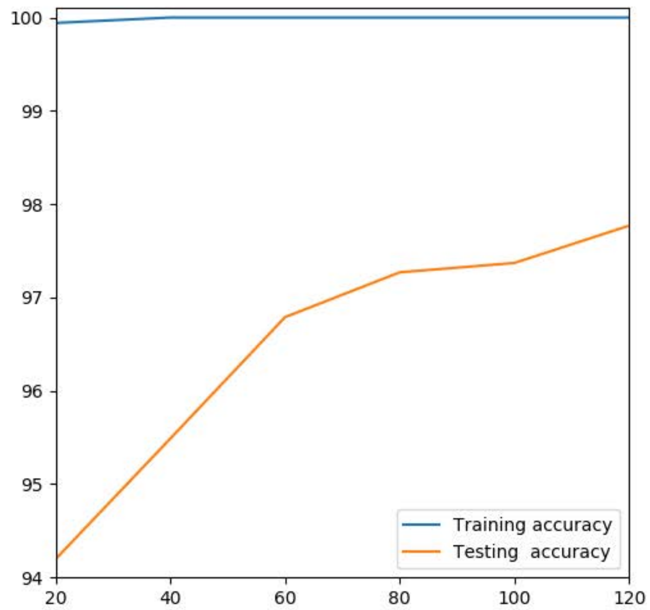
with respect to the coefficients of $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$.

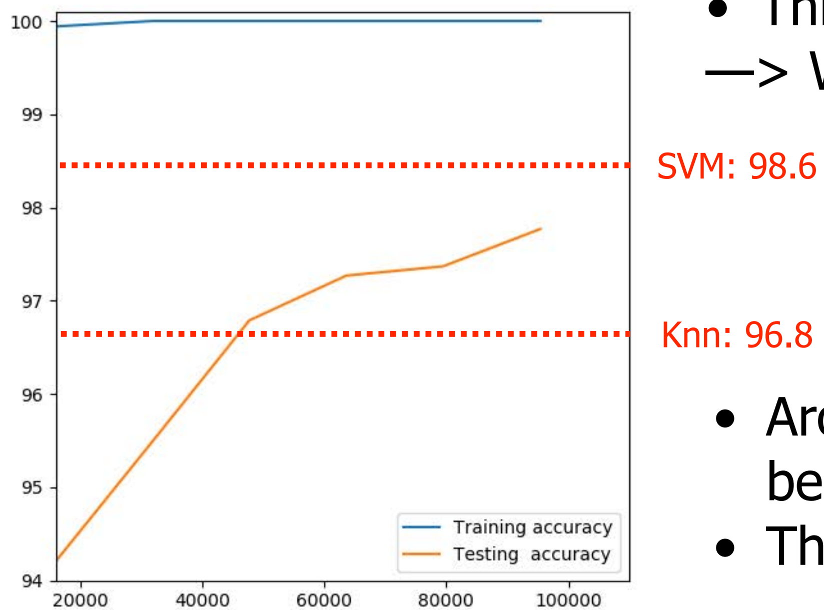
MNIST



- 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

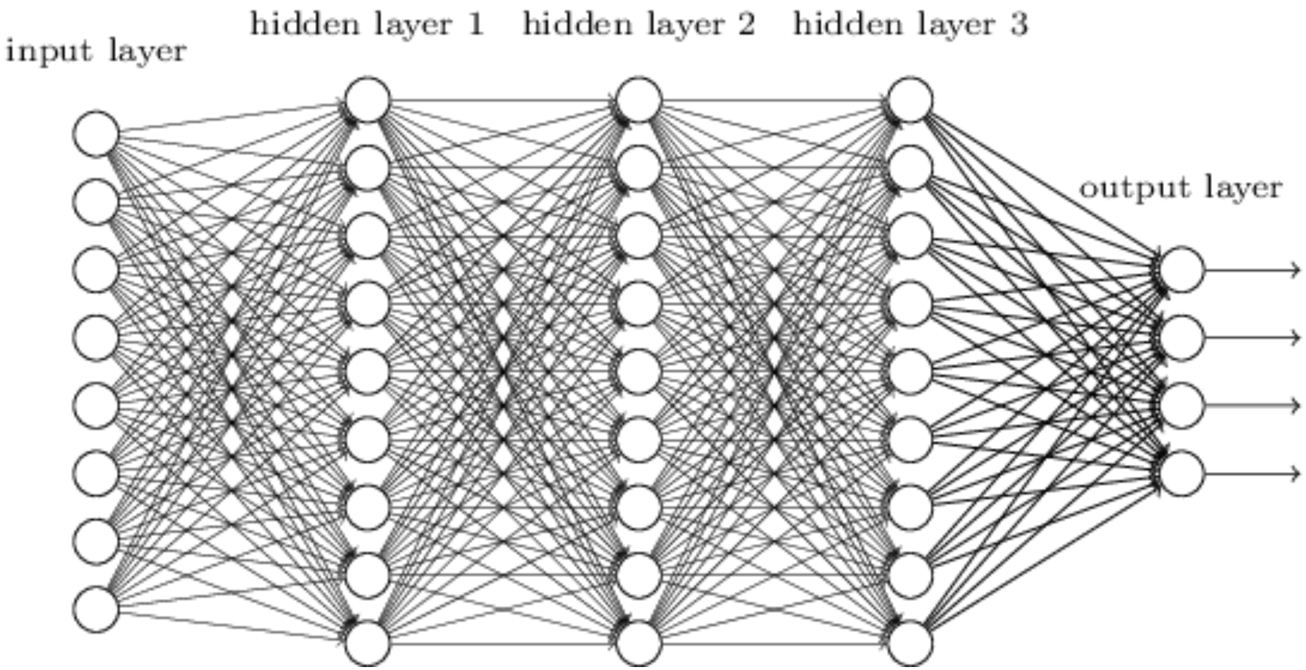
20 < hidden layer size < 120

- 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



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

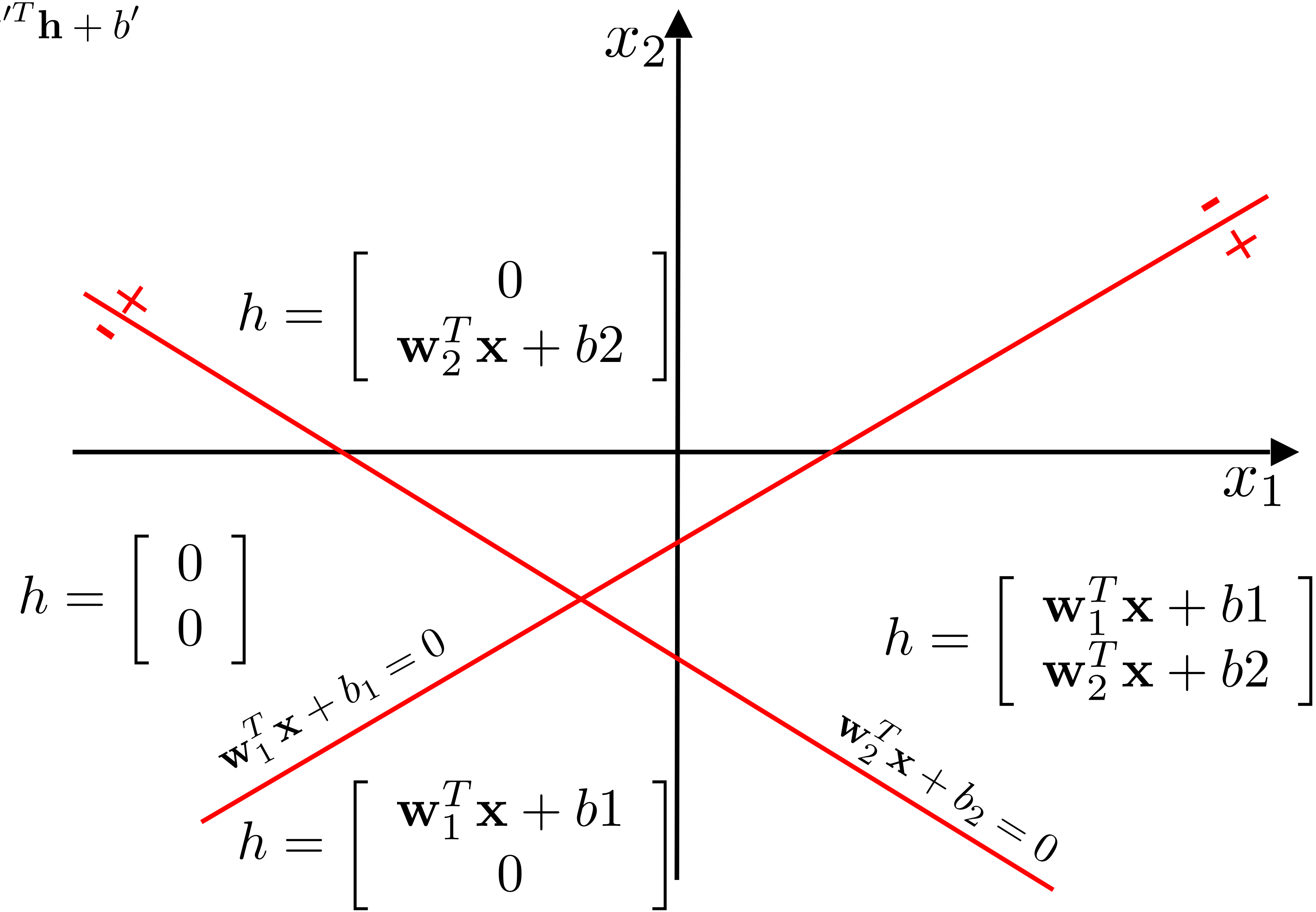
- The descriptive power of the net increases with the number of layers.
- In the case of a 1D signal, it is roughly proportional to $\prod_n W_n$ where w_n is the width of layer n.

Telgarsky, JMLR'16

One Layer: Two Hyperplanes

$$\mathbf{h} = \max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$y = \mathbf{w}'^T \mathbf{h} + b'$$

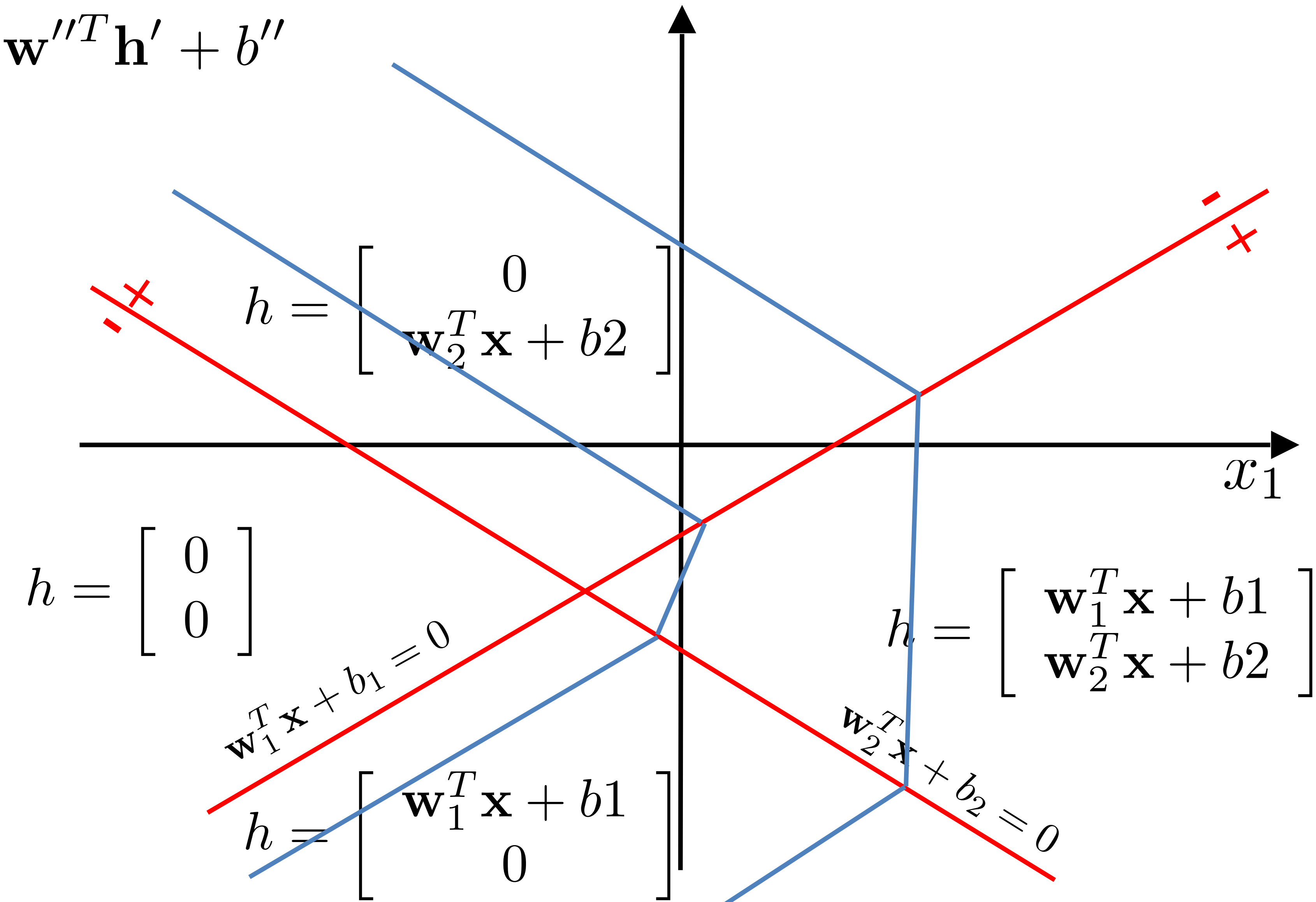


Two Layers: Two Hyperplanes

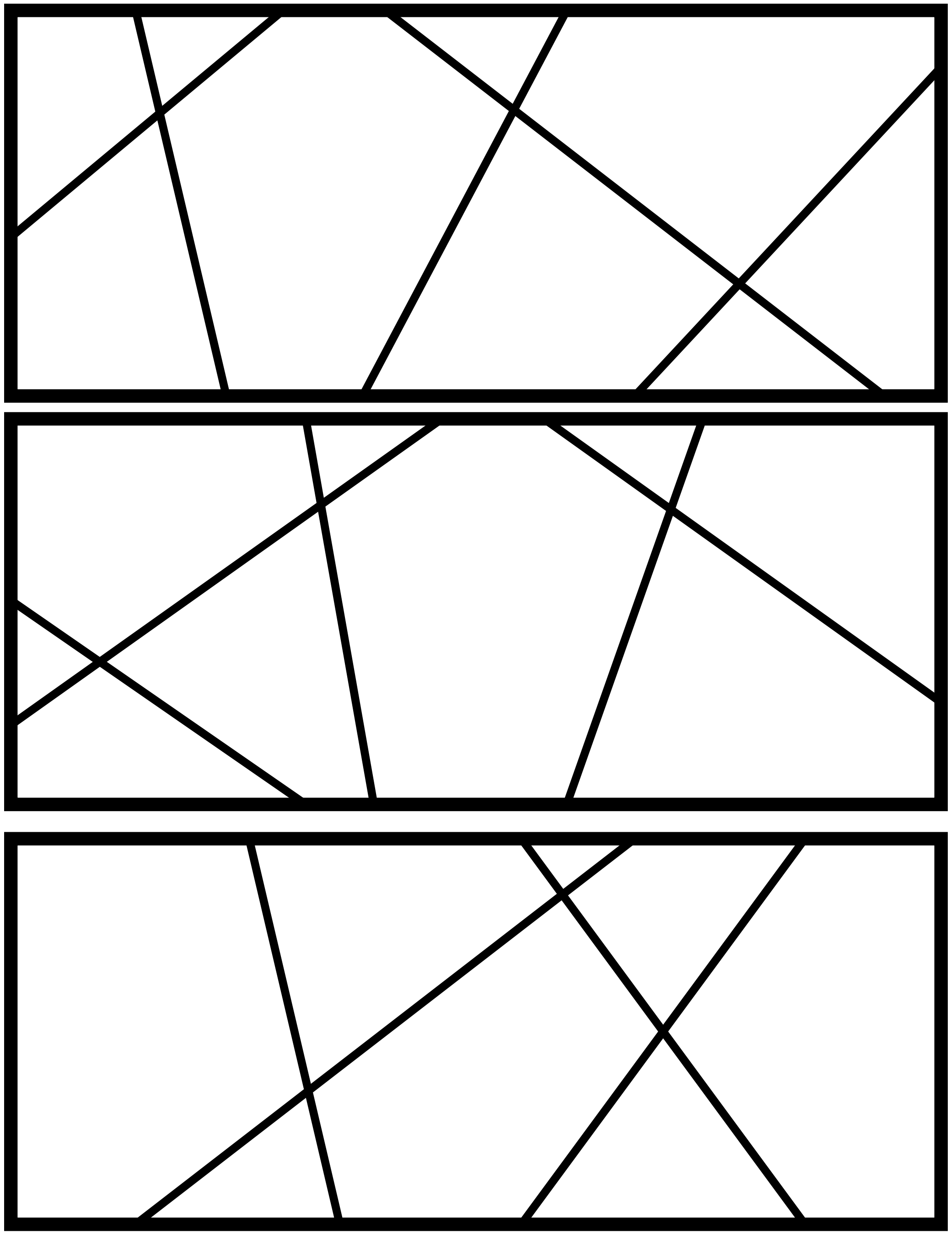
$$\mathbf{h} \ = \max(\mathbf{W}\mathbf{x} + \mathbf{b} \ , 0)$$

$$\mathbf{h}' = \max(\mathbf{W}'\mathbf{h} + \mathbf{b}', 0)$$

$$y = \mathbf{w}''^T \mathbf{h}' + b''$$

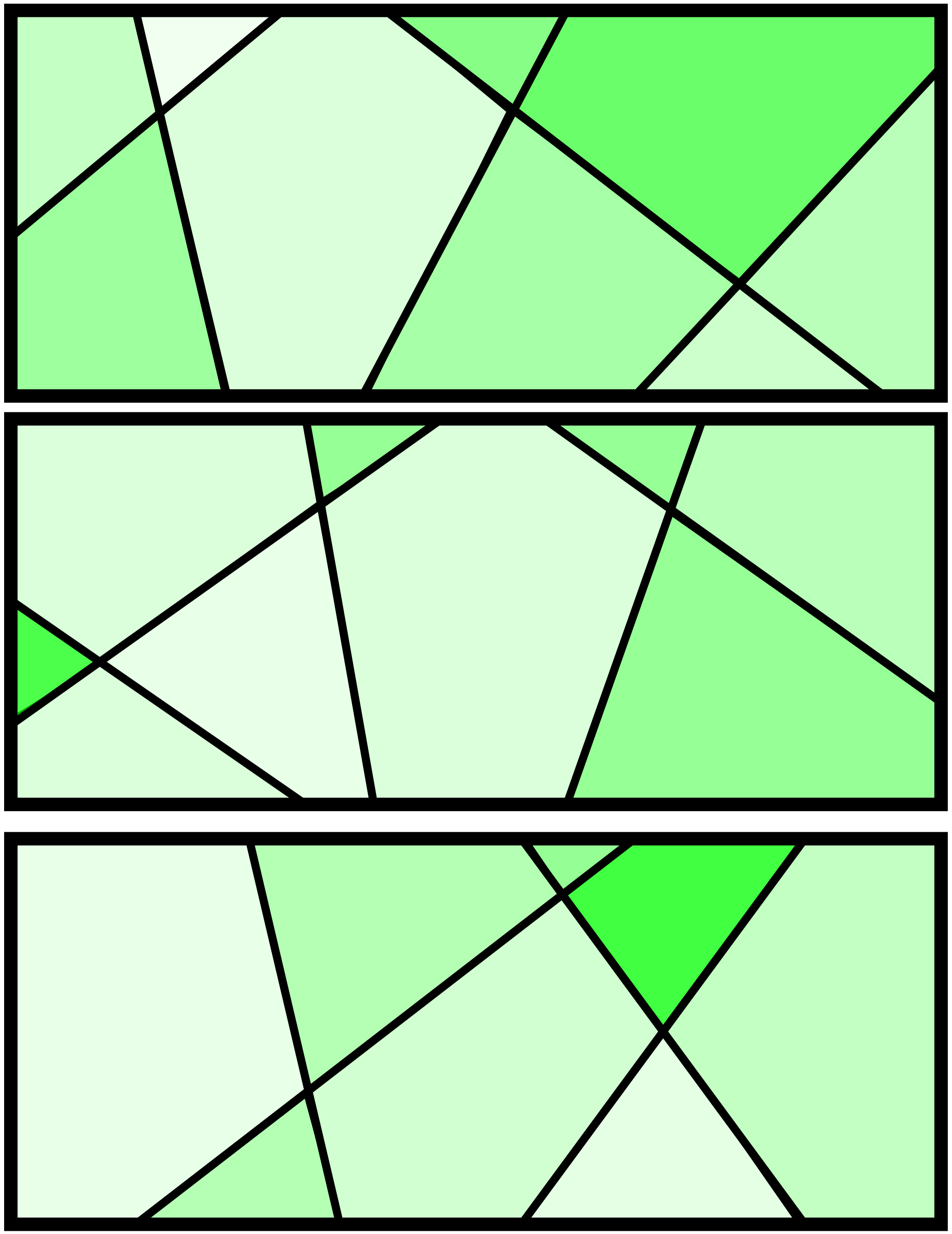


Graphical Interpretation



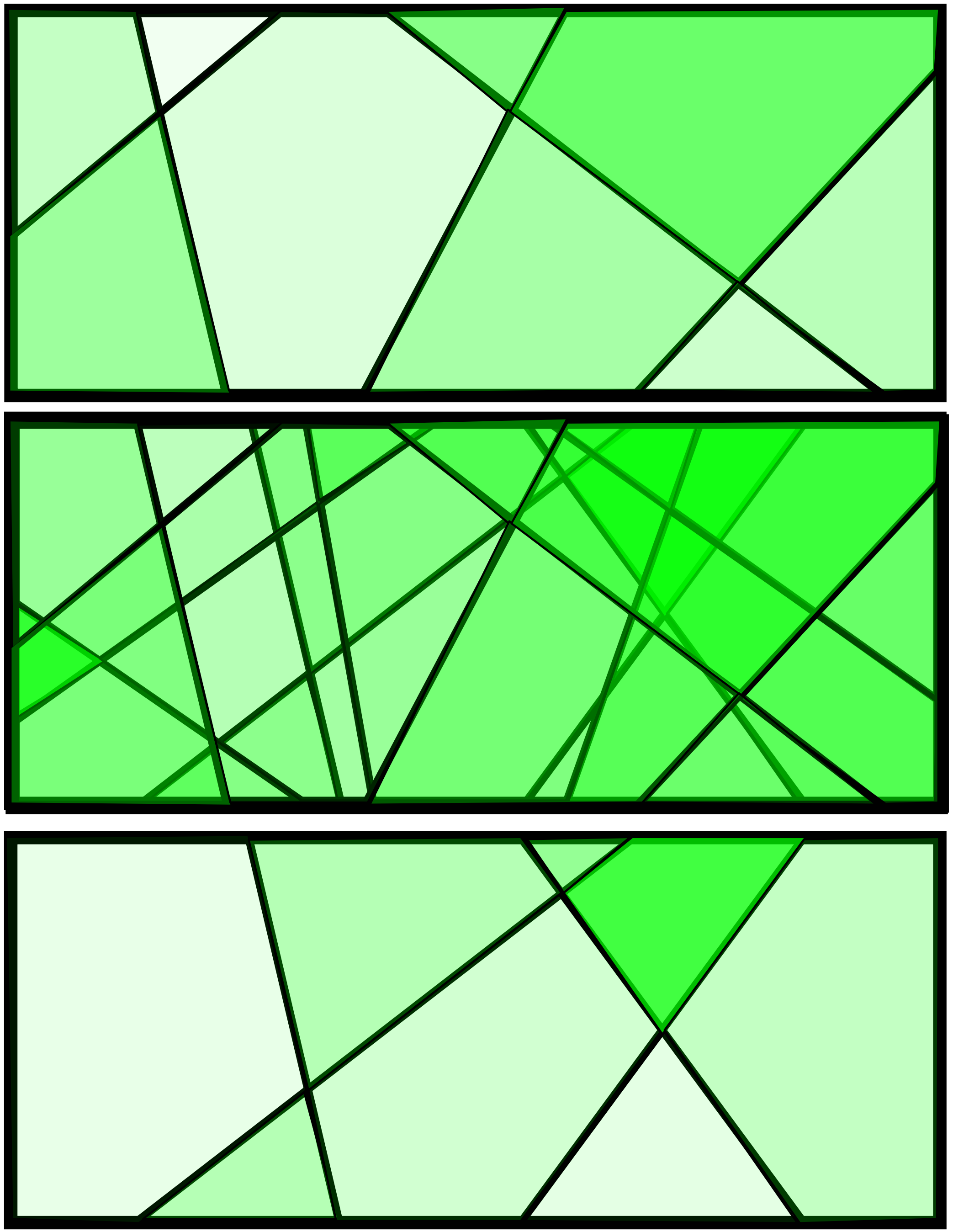
Hyperplanes at every level of the network split the space.

Graphical Interpretation



Hyperplanes at every level of the network split the space.

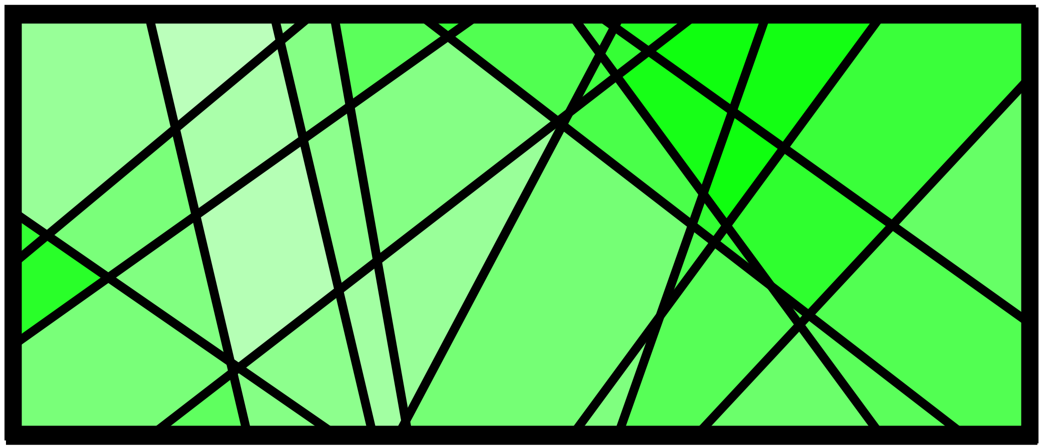
Graphical Interpretation



The splits are combined by the hierarchical nature of the network.

EPFL

Graphical Interpretation



The splits are combined due to the sequential nature of the network.

Multi Layer Perceptrons

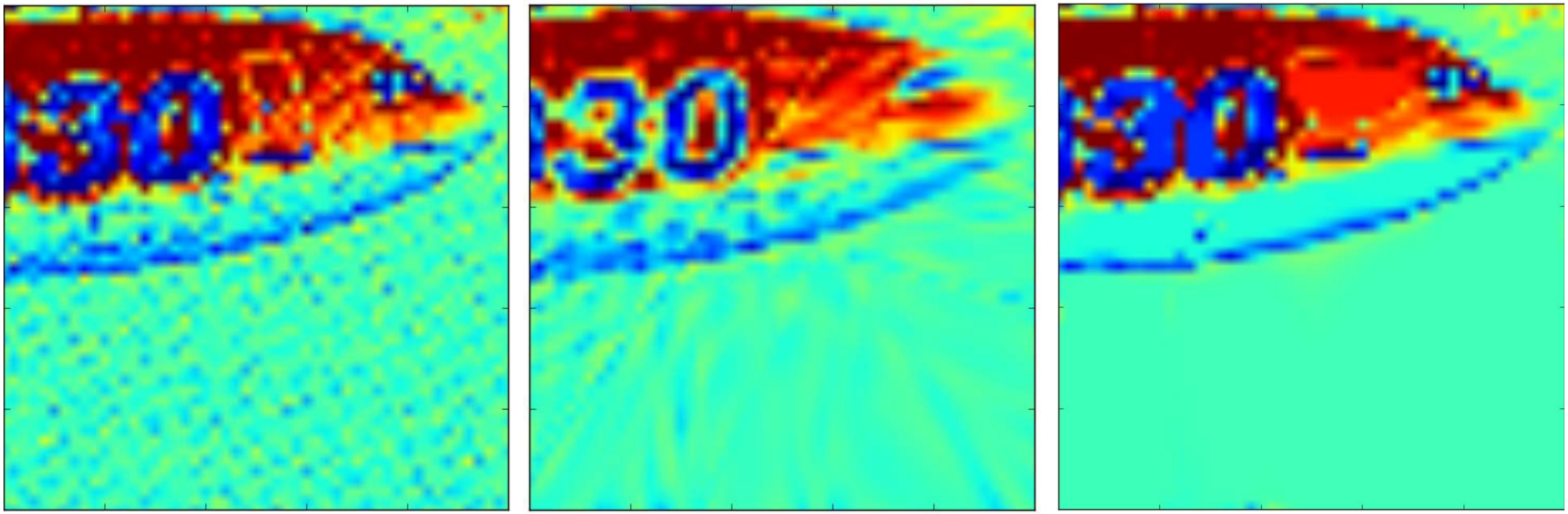
The function learned by an MLP using the ReLU, Sigmoid, or Tanh operators is:

- piecewise affine or smooth;
- continuous because it is a composition of continuous functions.

Each region created by a layer is split into smaller regions:

- Their boundaries are correlated in a complex way.
- Their descriptive power is larger than shallow networks for the same number of parameters.

Second Layer for Approximation



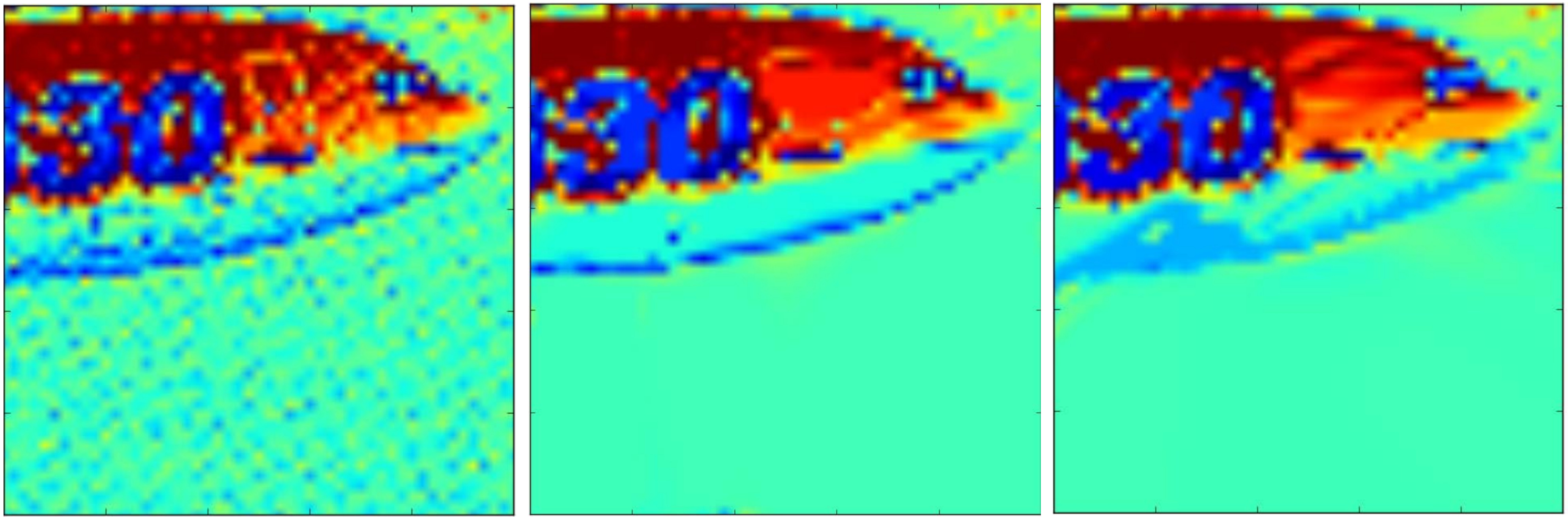
$$I = f(x, y)$$

1 Layer: 125 nodes -> loss 2.40e-01

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

501 weights in both cases

Adding a Third Layer



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

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

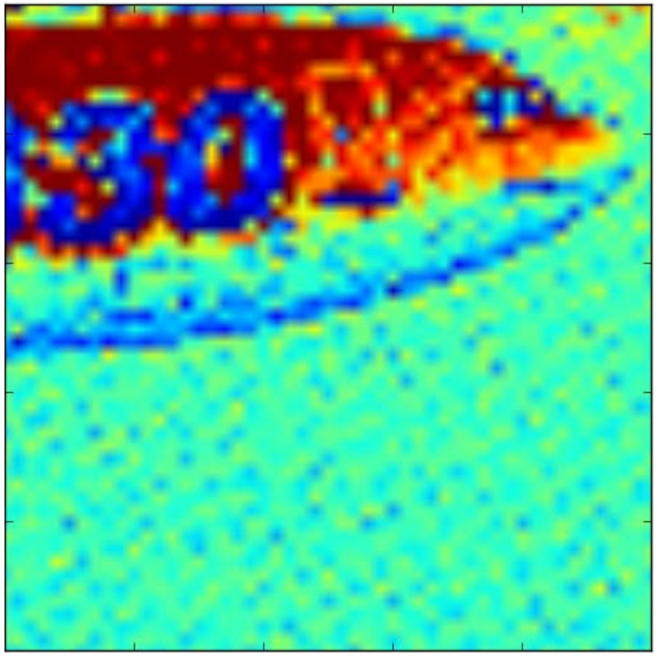
501 weights

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

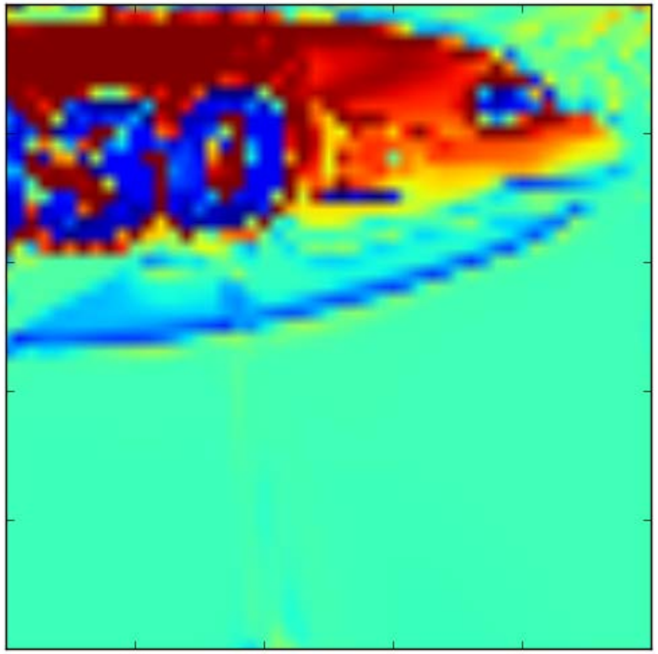
477 weights

EPFL

Adding a Third Layer

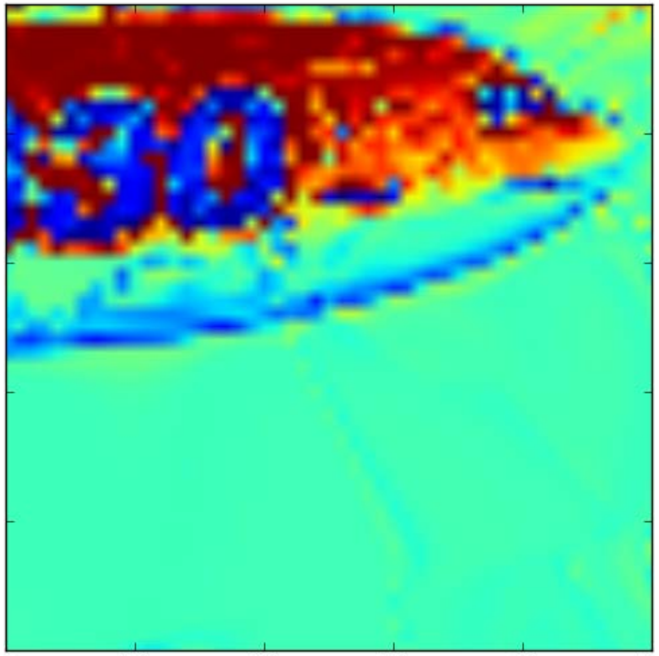


$I = f(x, y)$



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

541 weights

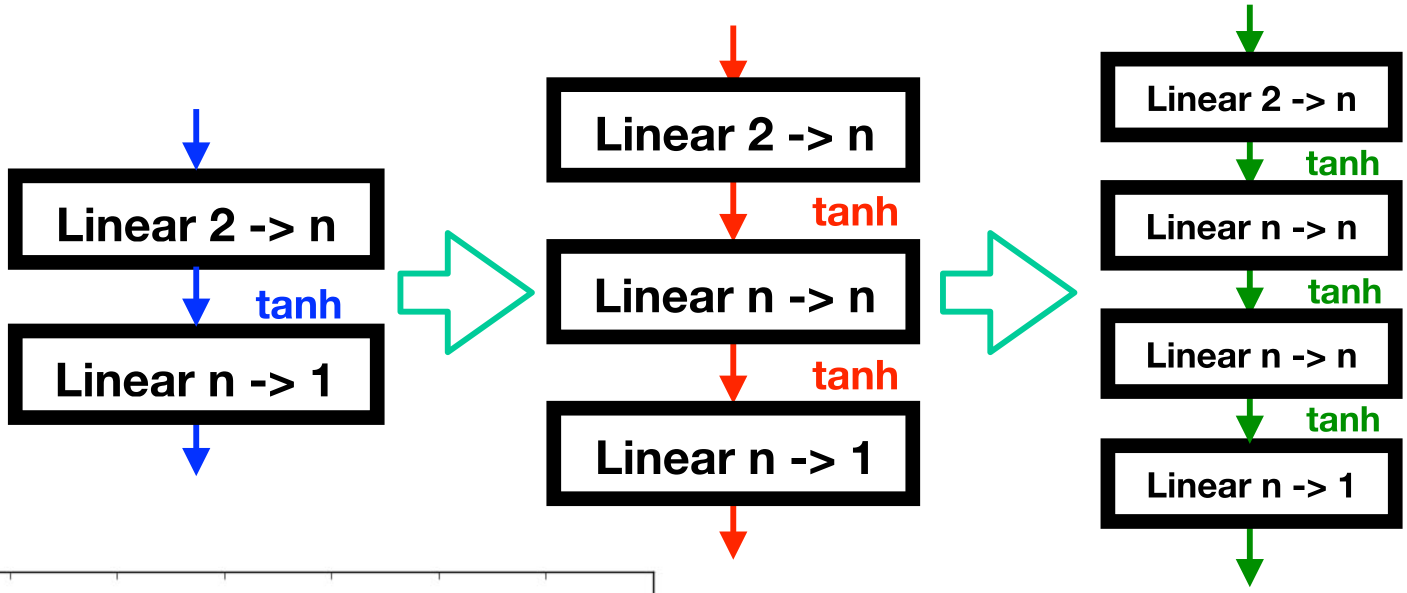


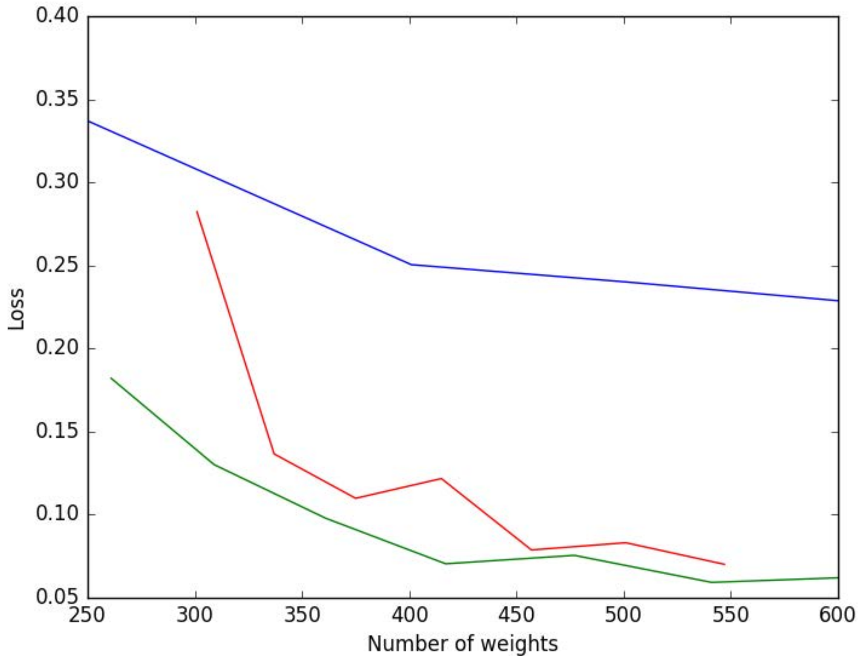
3 Layers: 19 nodes -> loss 4.38e-02

837 weights

EPFL

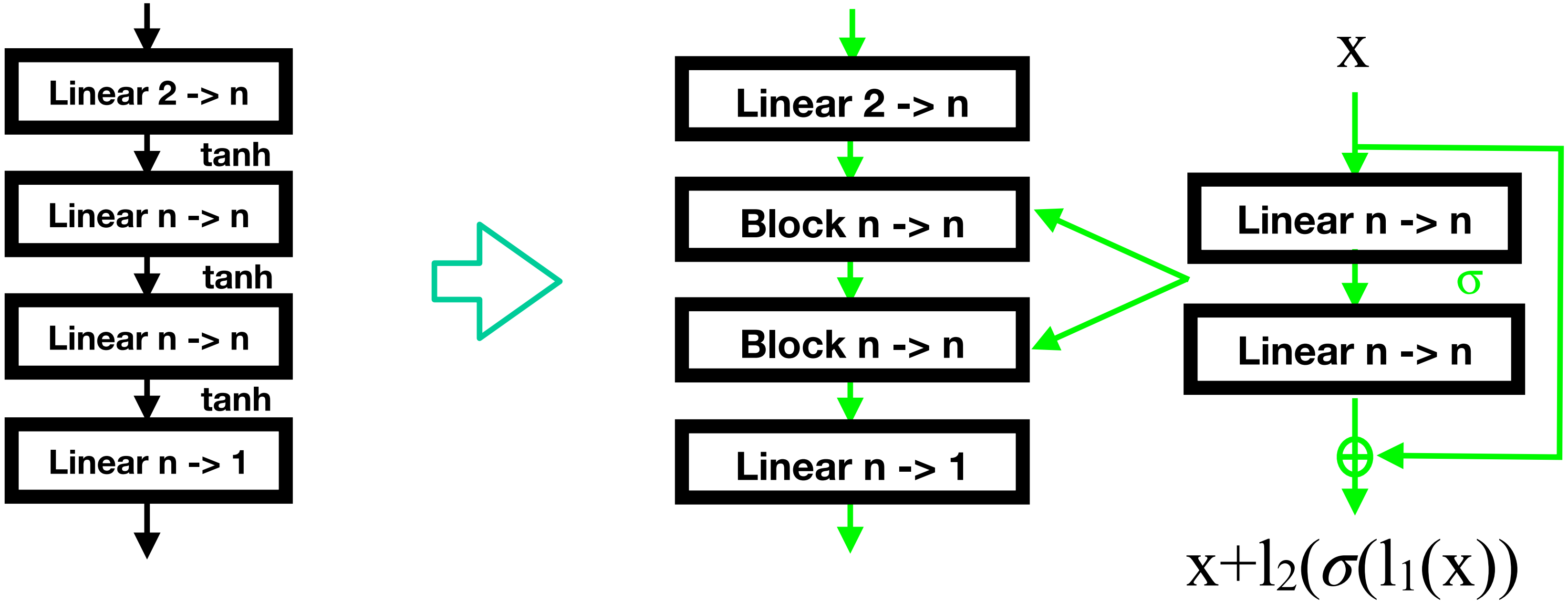
Multi Layer Perceptrons





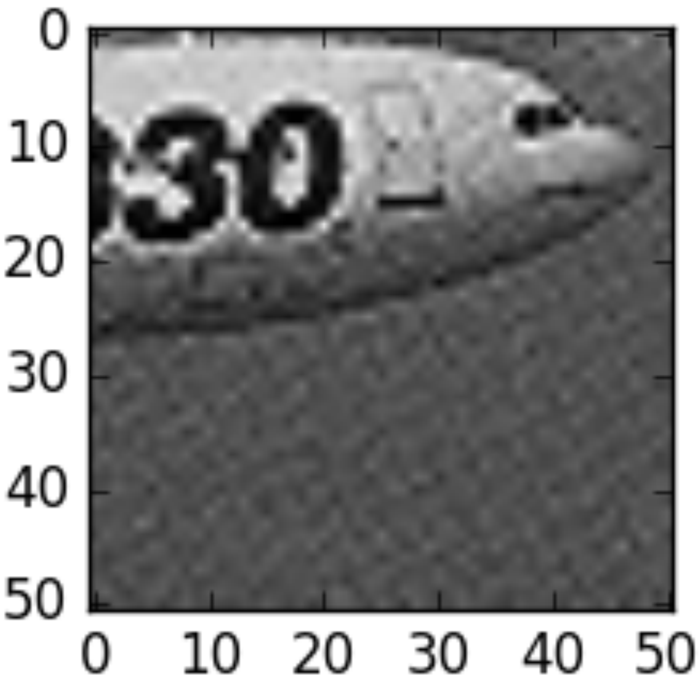
- Adding layers tends to deliver better convergence properties.
- In current practice, deeper is usually better.

MLP to ResNet

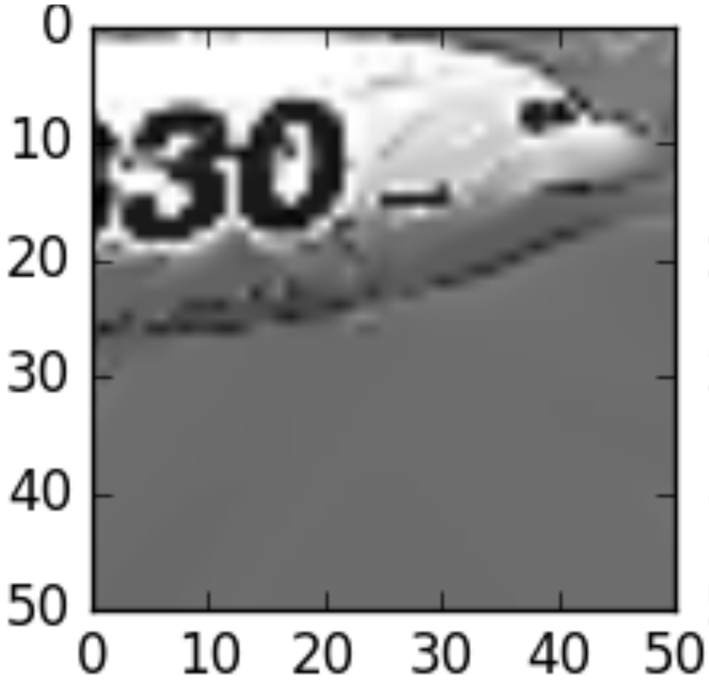


Further improvements in the convergence properties have been obtained by adding a bypass, which allows the final layers to only compute residuals.

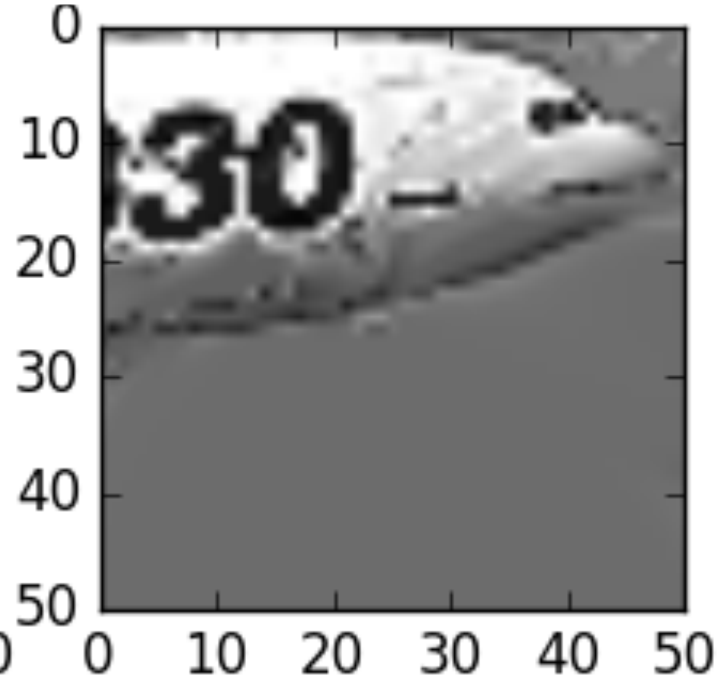
Improving the Network



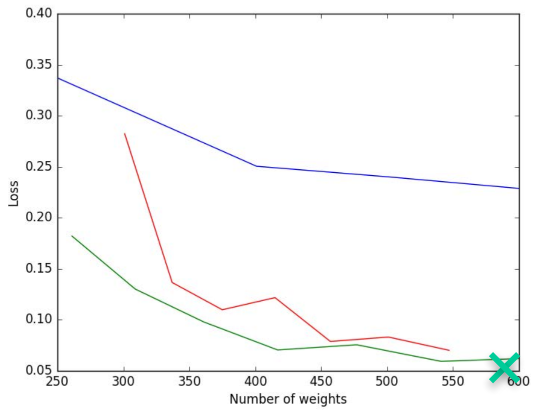
Original 51x51 image:
2601 gray level values.

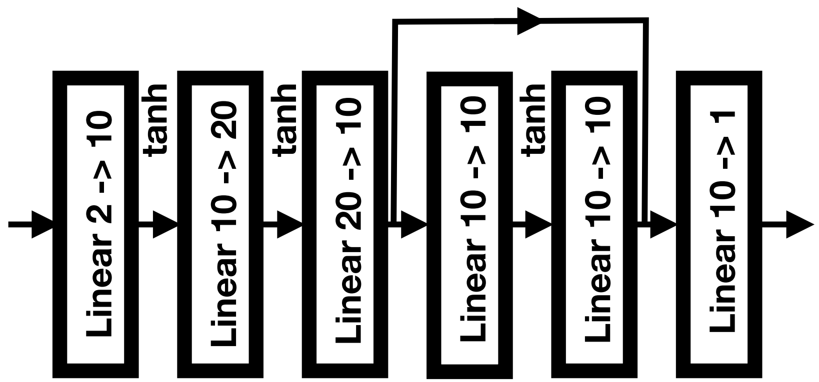


MLP 10/20/10 Interpolation:
471 weights, loss 6.43e-02.



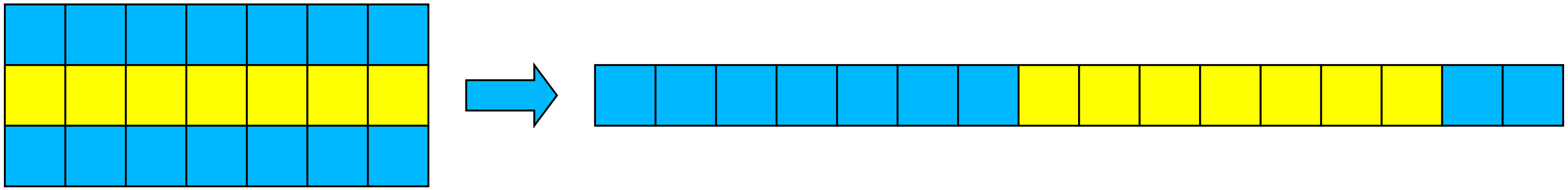
MLP 10/20/10/10 Interpolation:
581 weights, loss 5.30e-2.





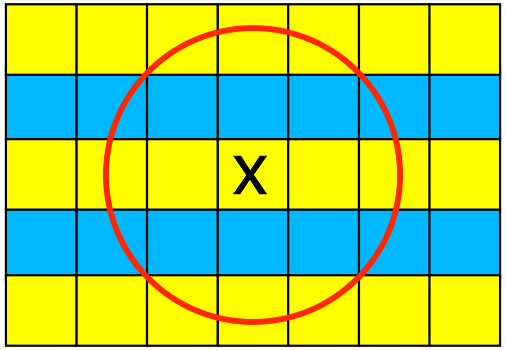
Digital Images

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.

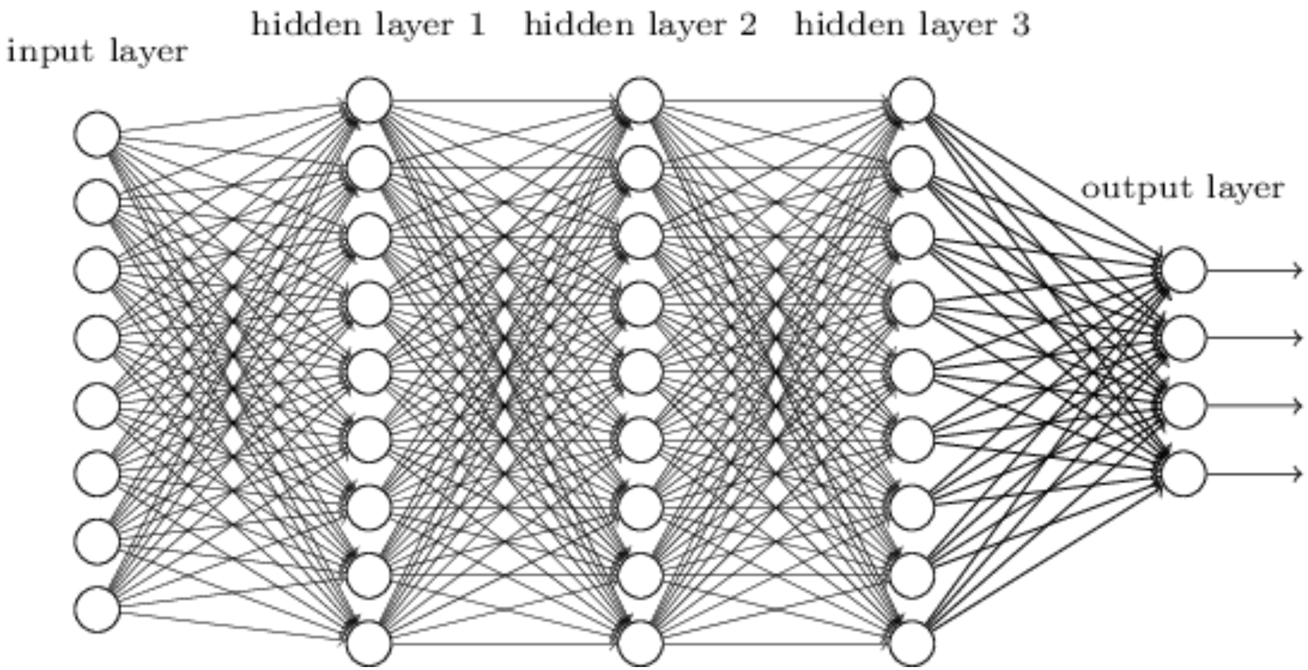
Image Specificities



- 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.

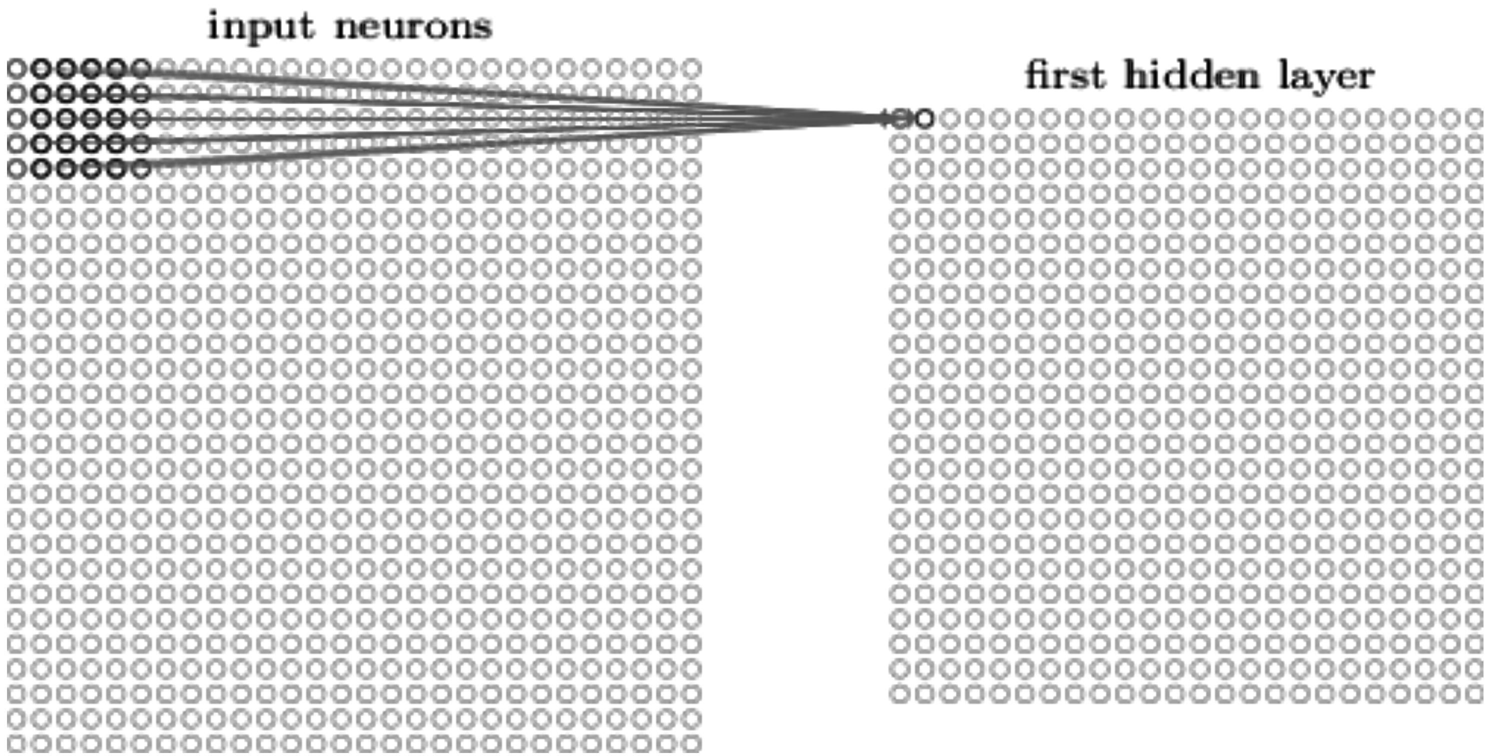
—> These two properties can be exploited to drastically reduce the number of weights required by CNNs using so-called convolutional layers.

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.

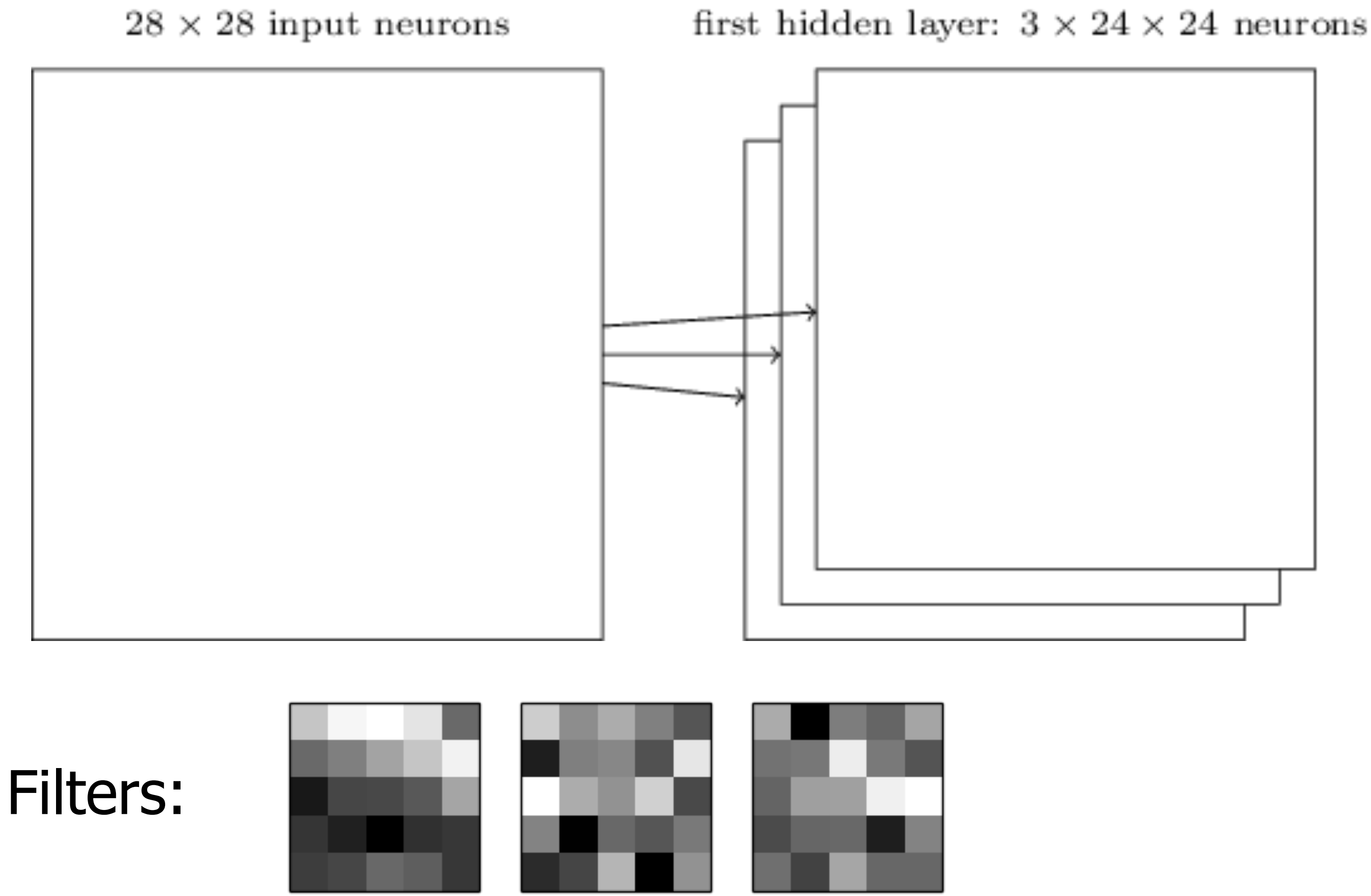
Convolutional Layer



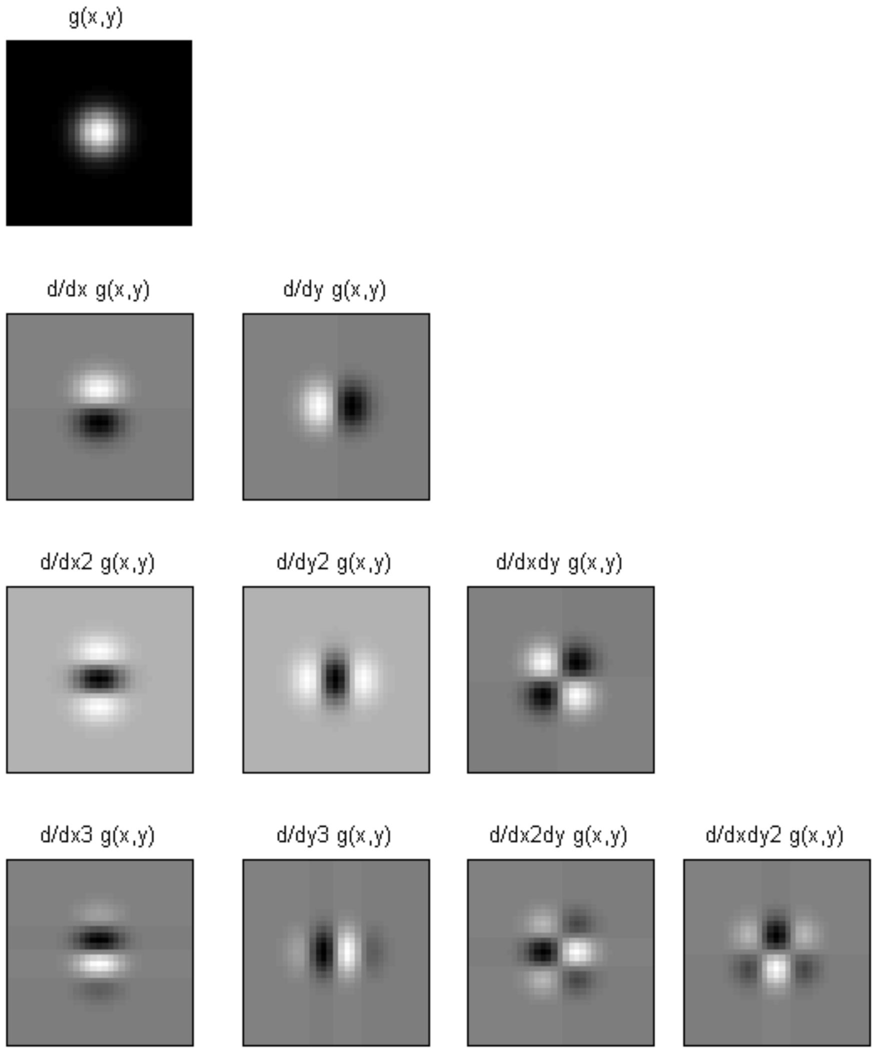
$$\sigma\left(b+\sum_{x=0}^{n_x}\sum_{y=0}^{n_y} w_{x,y}a_{i+x,j+y}\right)$$

EPFL

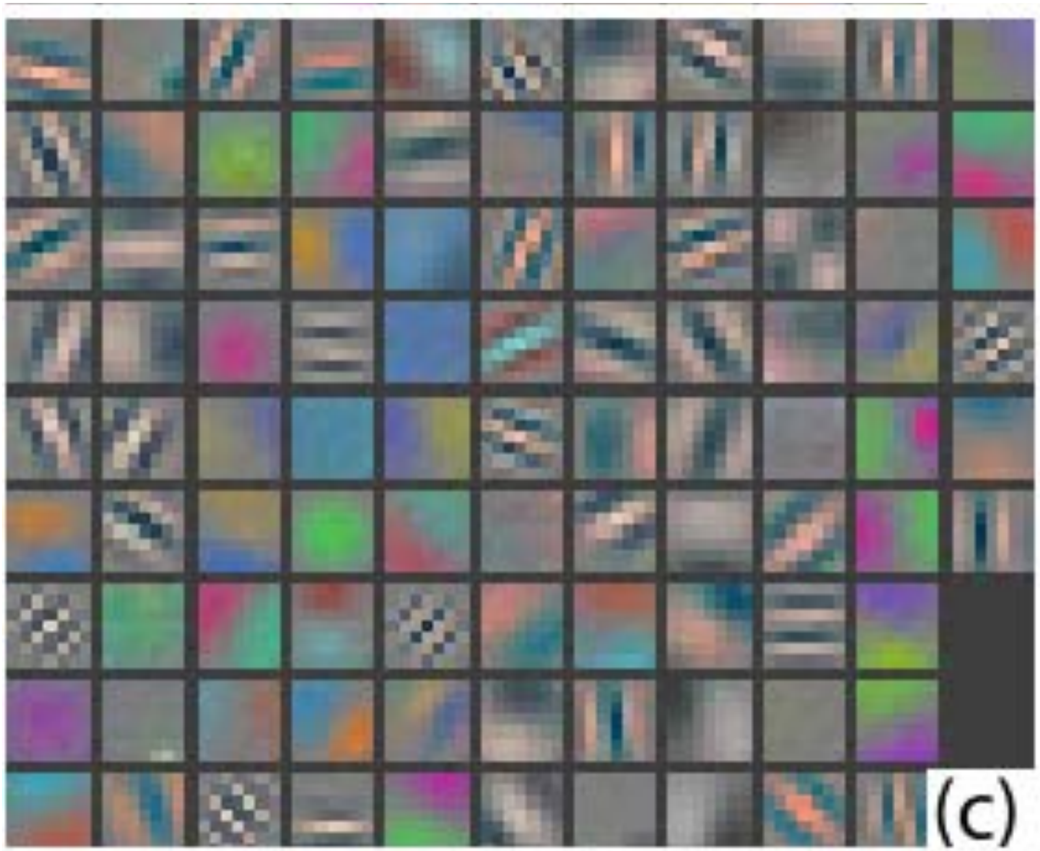
Feature Maps



Filters

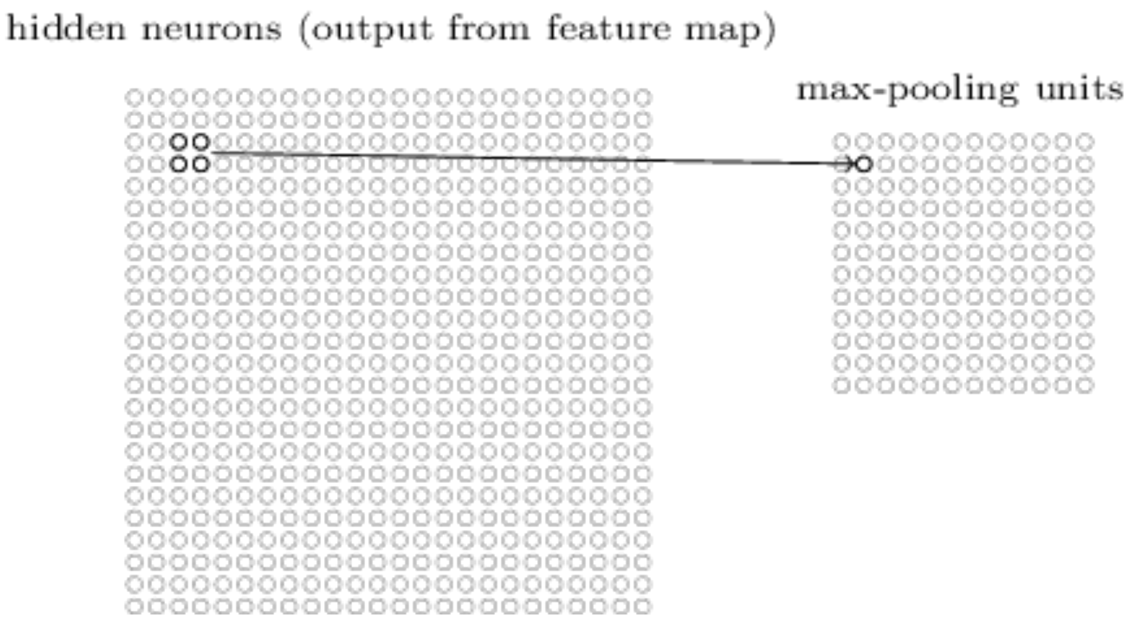


Derivatives



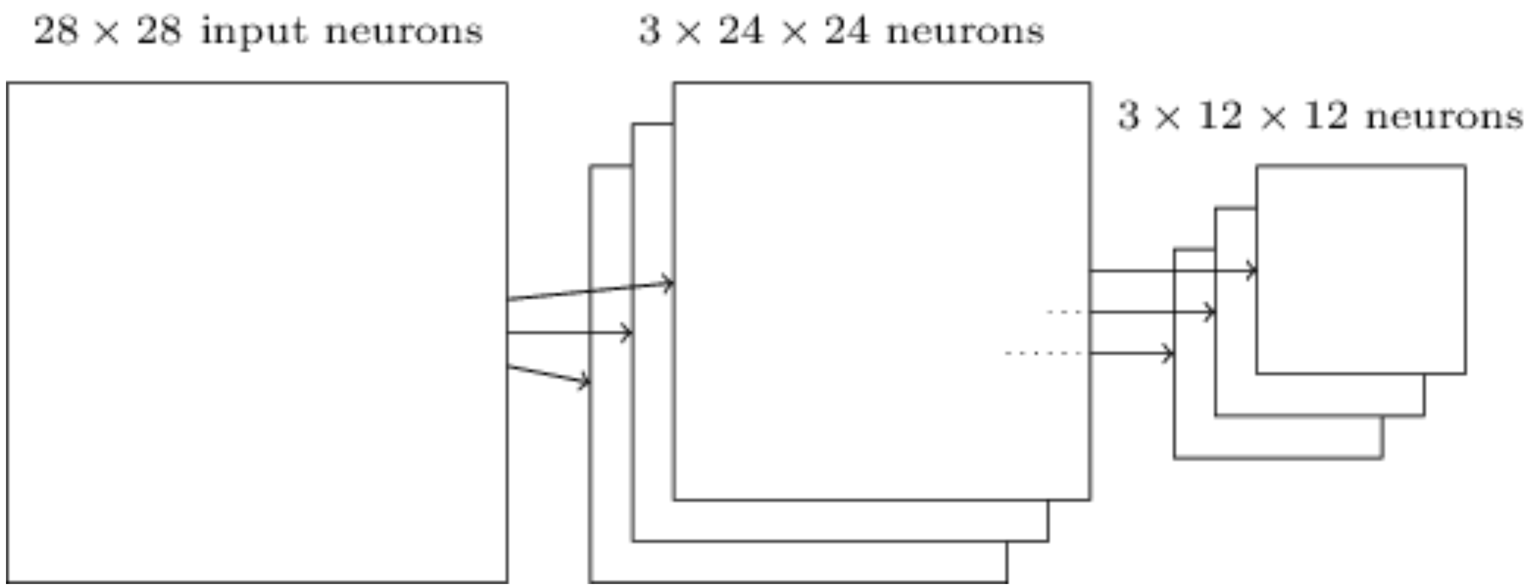
Learned filters

Pooling Layer



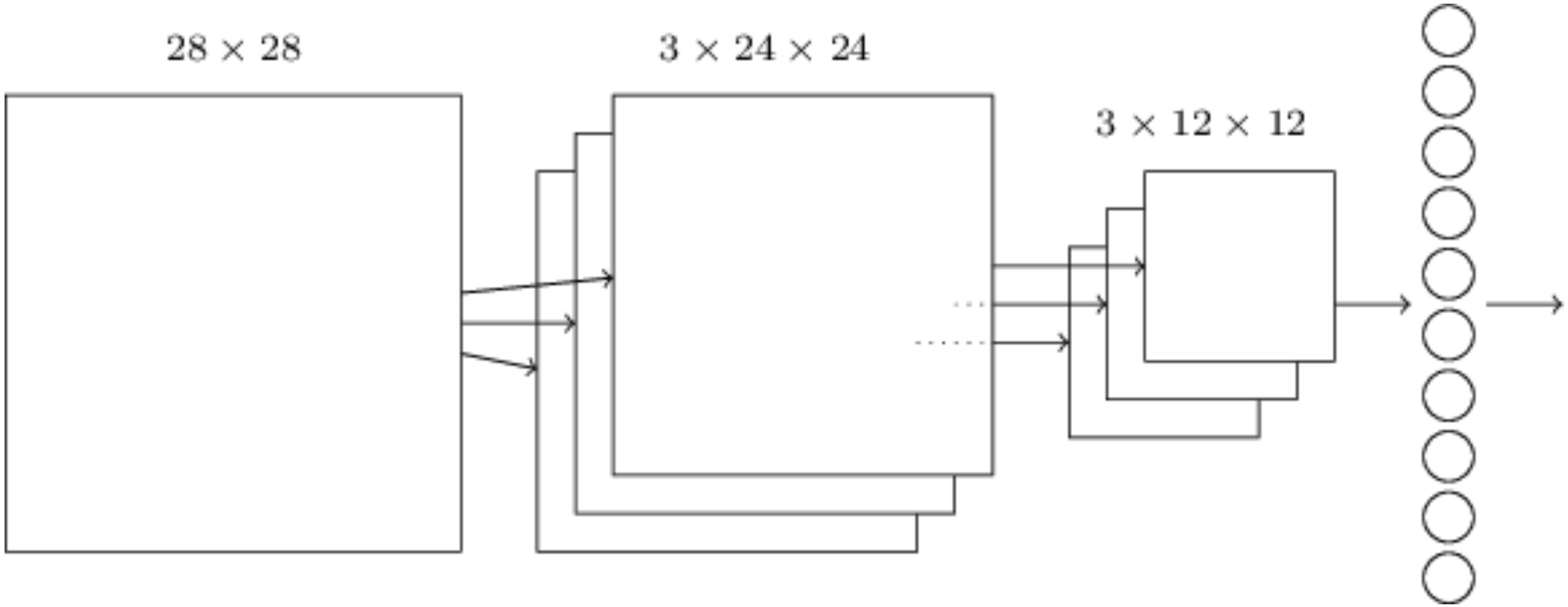
- Reduces the number of inputs by replacing all activations in a neighborhood by a single one.
- Can be thought as asking if a particular feature is present in that neighborhood while ignoring the exact location.

Adding the Pooling Layers



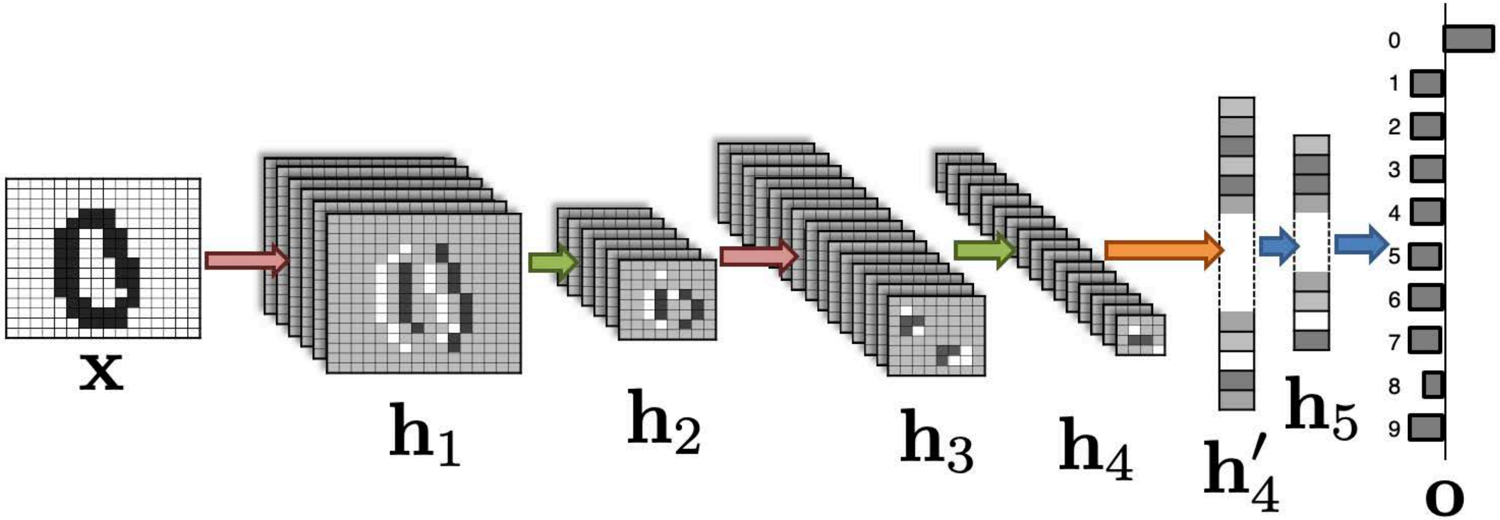
The output size is reduced by the pooling layers.

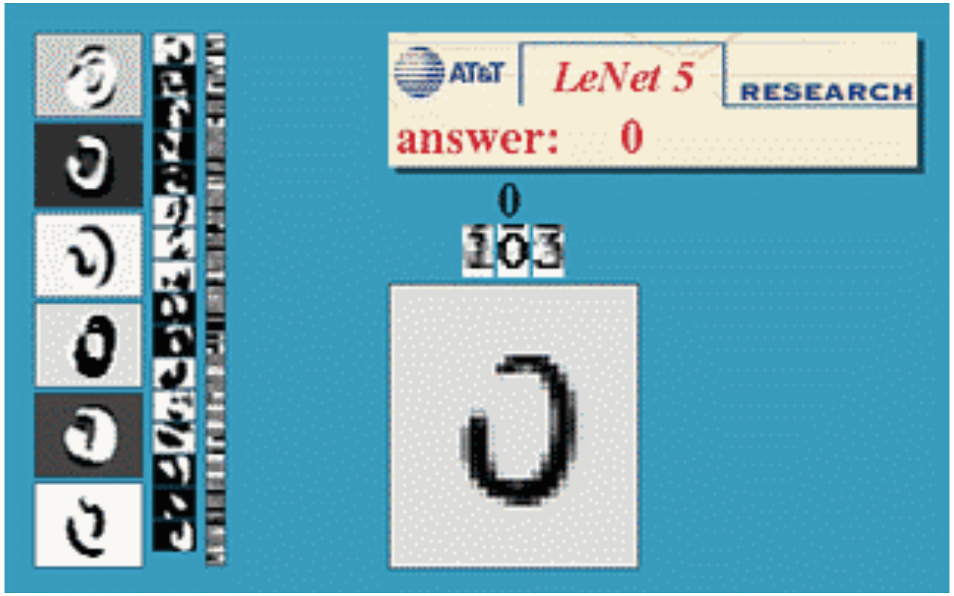
Adding a Fully Connected Layer



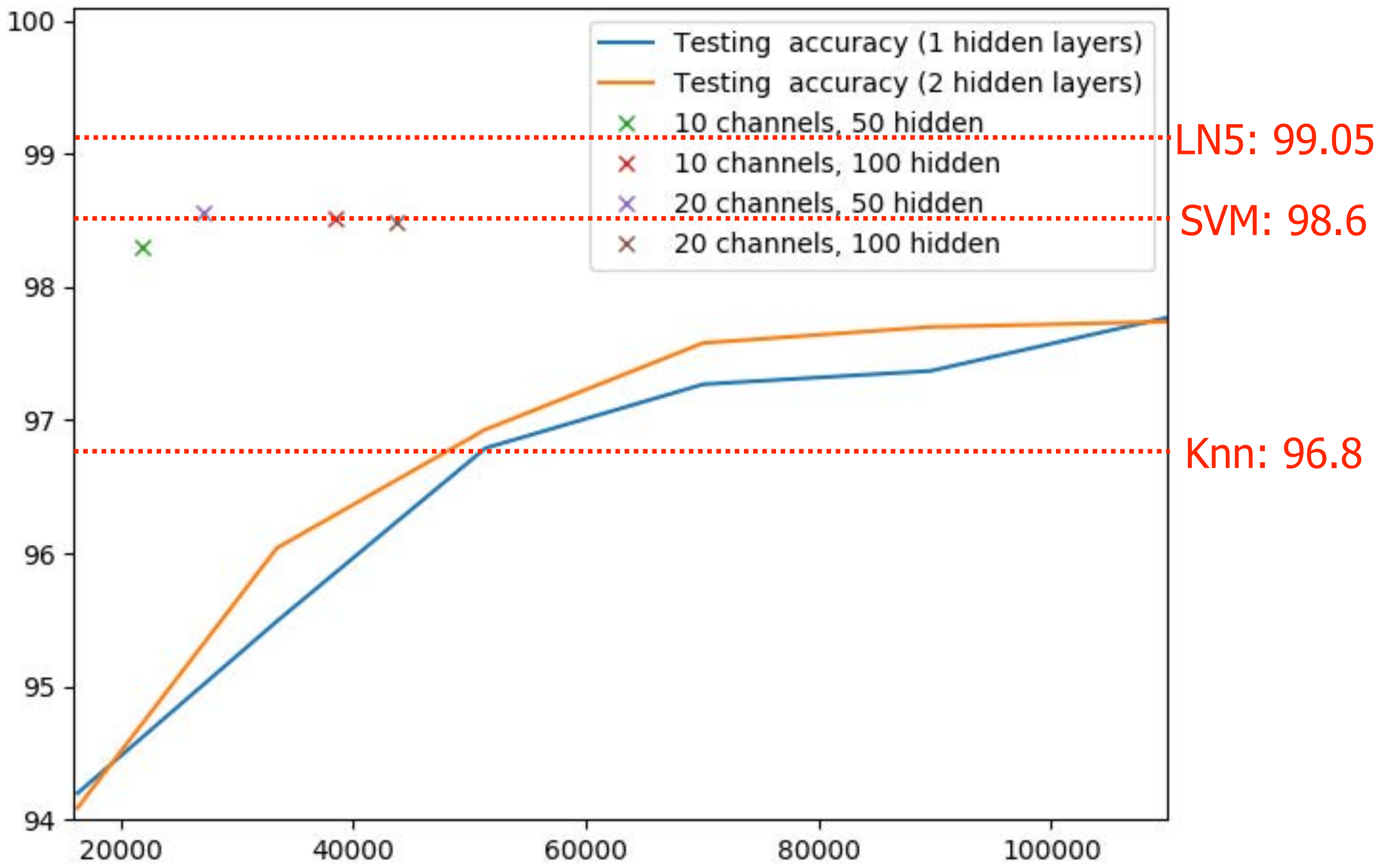
- Each neutron in the final fully connected layer is connected to all neurons in the preceding one.
- Deep architecture with many parameters to learn but still far fewer than an equivalent multilayer perceptron.

LeNet (1989-1999)

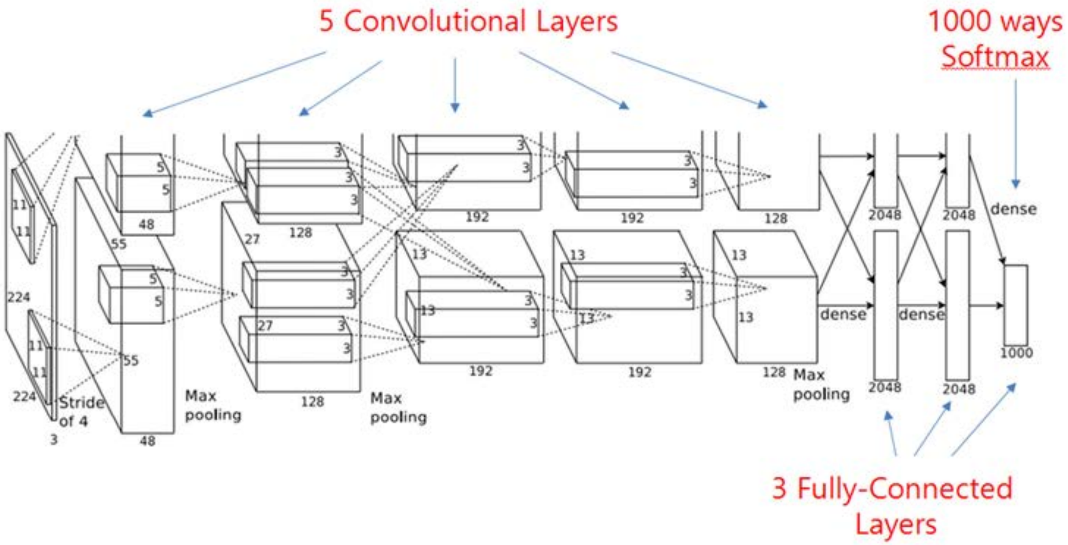




Lenet Results



AlexNet (2012)



Task: Image classification

Training images: Large Scale Visual Recognition Challenge 2010

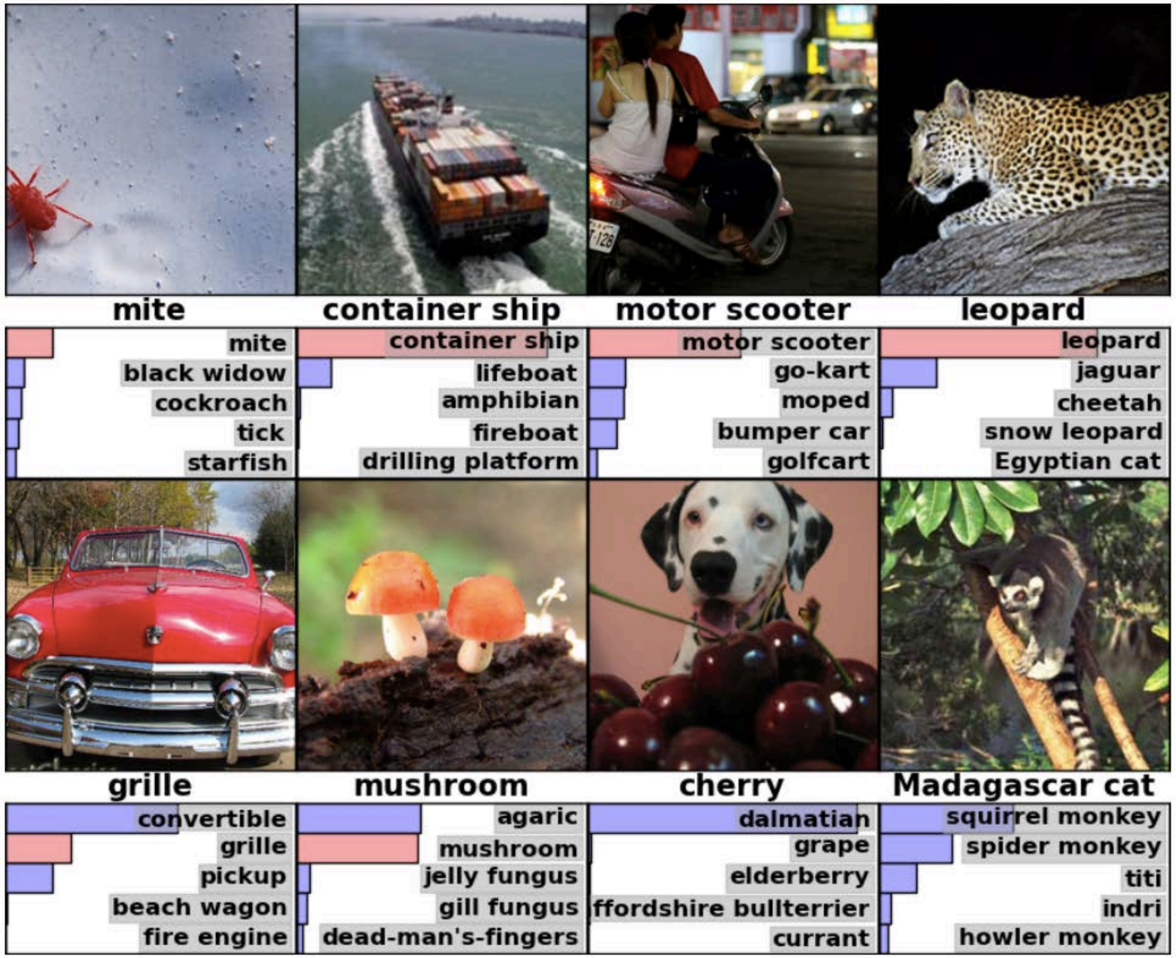
Training time: 2 weeks on 2 GPUs

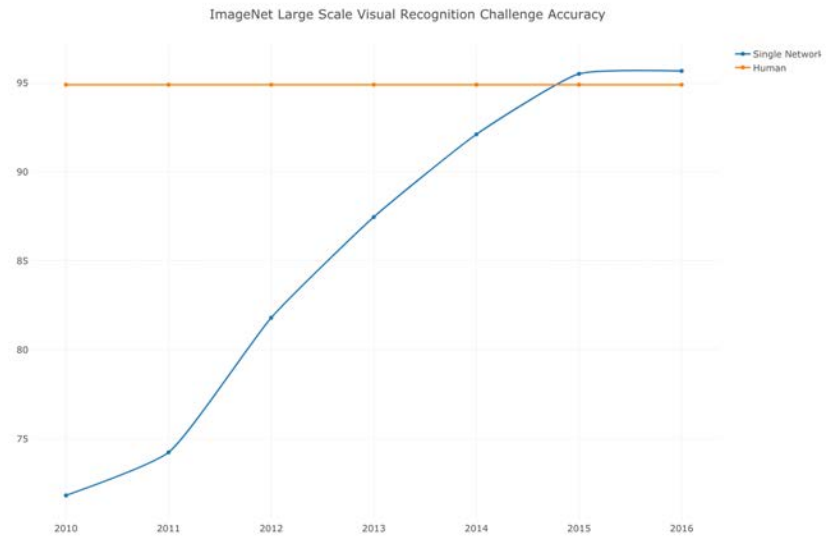
Major Breakthrough: Training large networks has now been shown to be practical!!

EPFL

Krizhevsky, NIPS'12

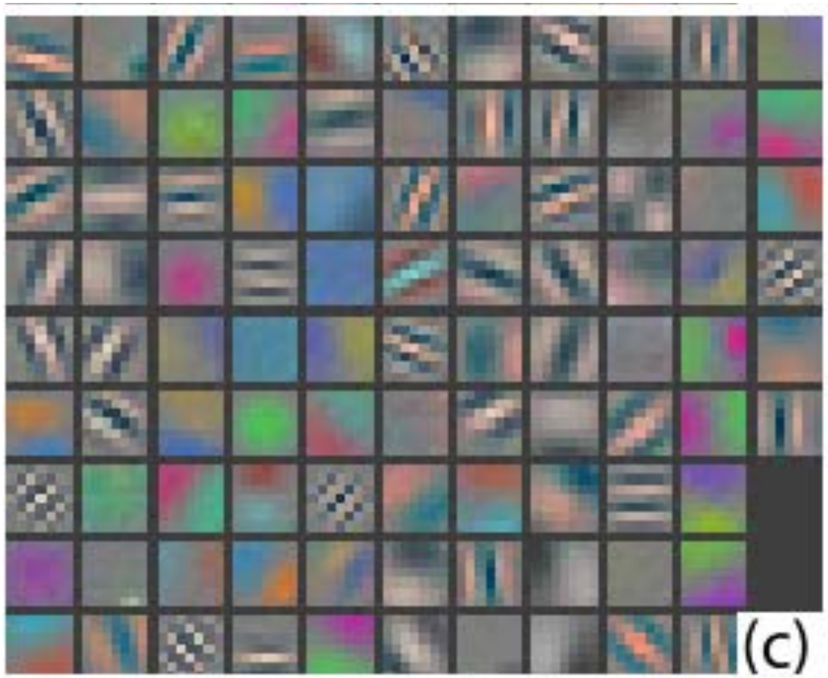
AlexNet Results



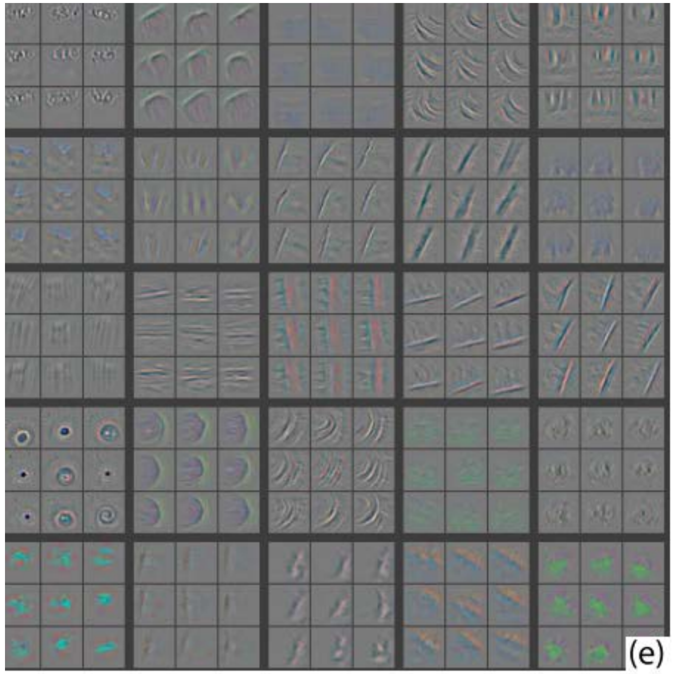


- At the 2012 ImageNet Large Scale Visual Recognition Challenge, AlexNet achieved a top-5 error of 15.3%, more than 10.8% lower than the runner up.
- Since 2015, networks outperform humans on this task.

Feature Maps



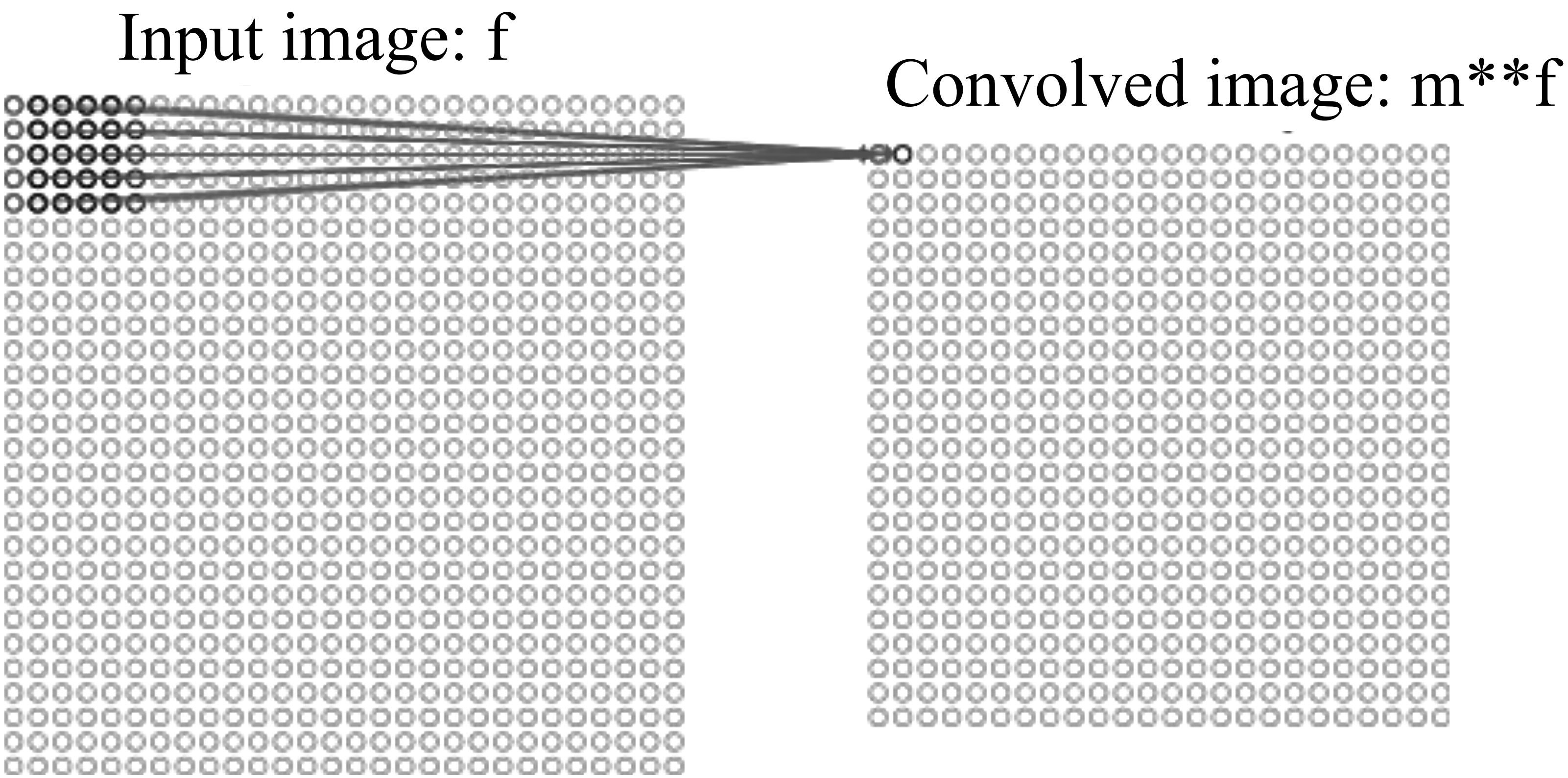
First convolutional layer



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.

Reminder: Discrete 2D Convolution



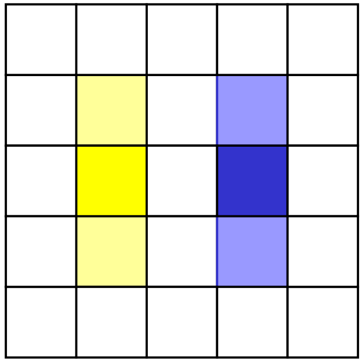
Convolution mask m, also known as a *kernel*.

$$\begin{bmatrix} m_{11} & \dots & m_{1w} \\ \dots & \dots & \dots \\ m_{w1} & \dots & m_{ww} \end{bmatrix}$$

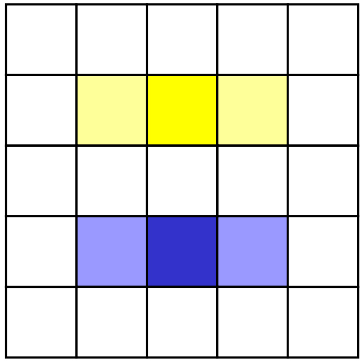
$$m**f(x,y) = \sum_{i=0}^{w} \sum_{j=0}^{w} m(i,j) f(x-i, y-j)$$

Reminder: 3X3 Masks

x derivative



y derivative



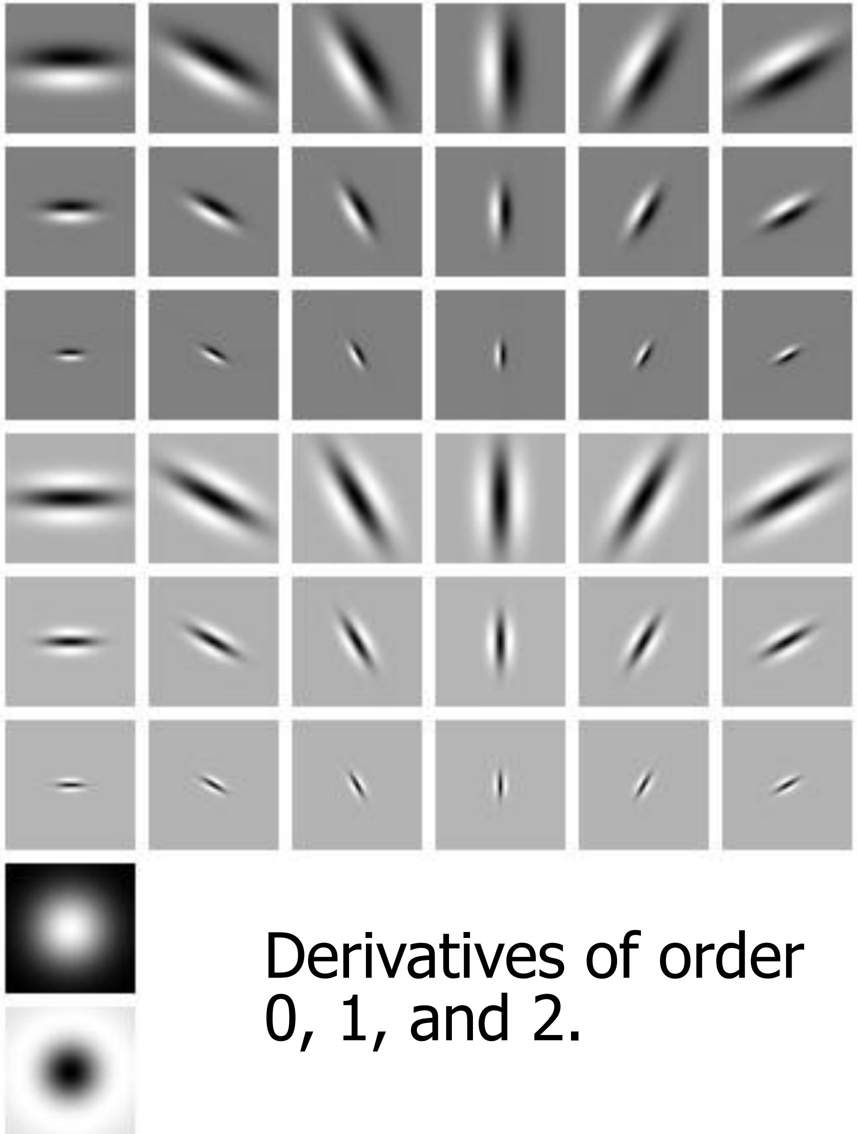
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Prewitt operator

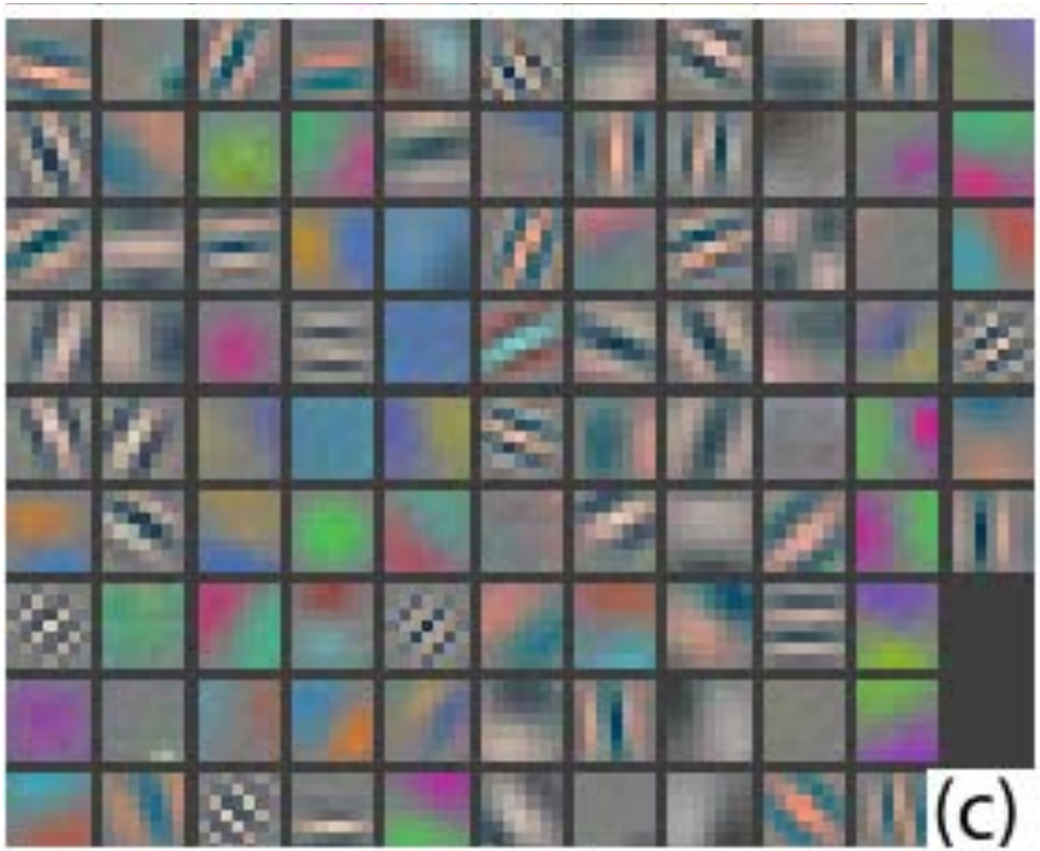
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel operator

Filter Banks

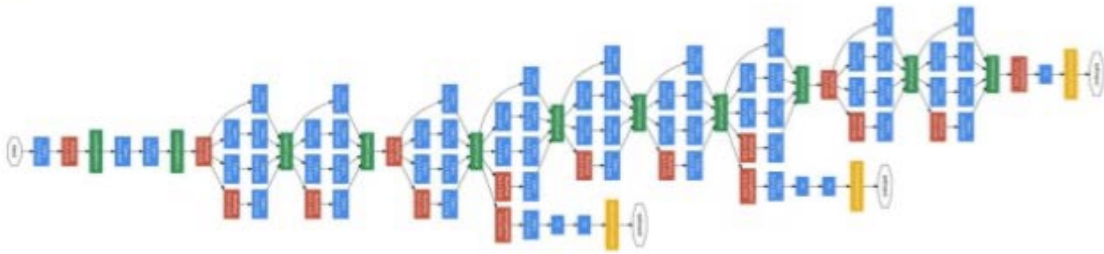


Derivatives of order 0, 1, and 2.



Learned

Bigger and Deeper



"hibiscus"

"dahlia"

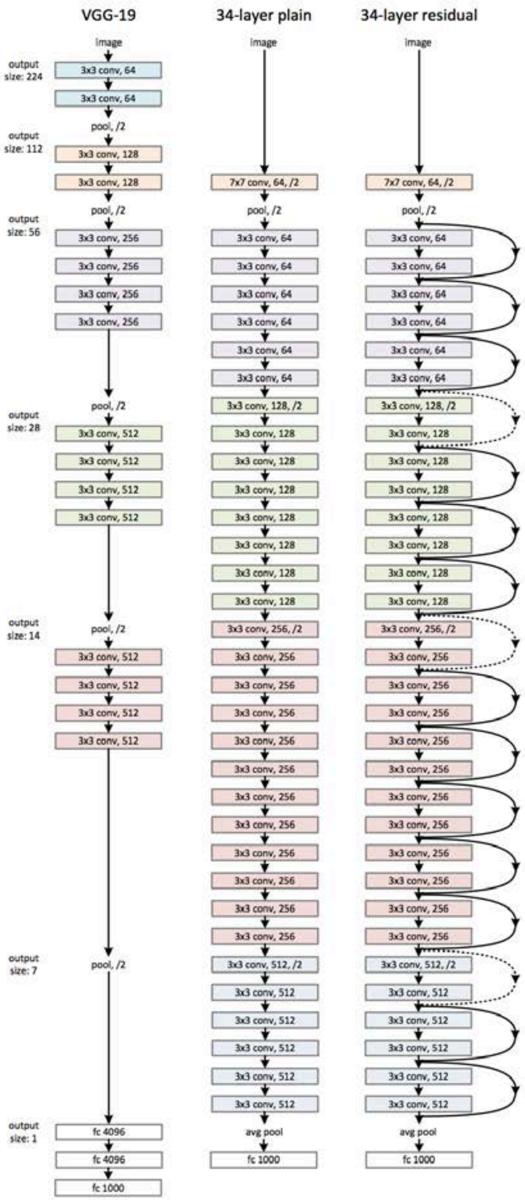
VGG19, 3 weeks of training.

GoogleLeNet.

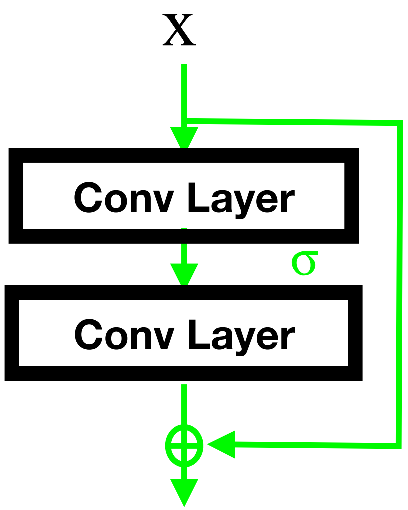
"It was demonstrated that the representation depth is beneficial for the classification accuracy, and that state-of-the-art performance on the ImageNet challenge dataset can be achieved using a conventional ConvNet architecture."

Simonyan & Zisserman, ICLR'15

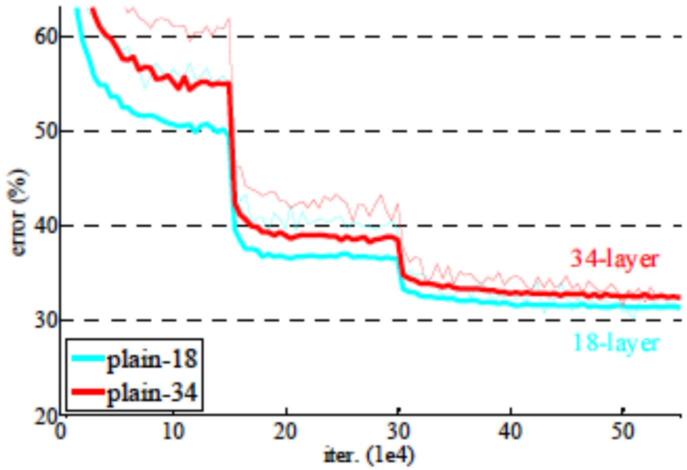
Deeper and Deeper

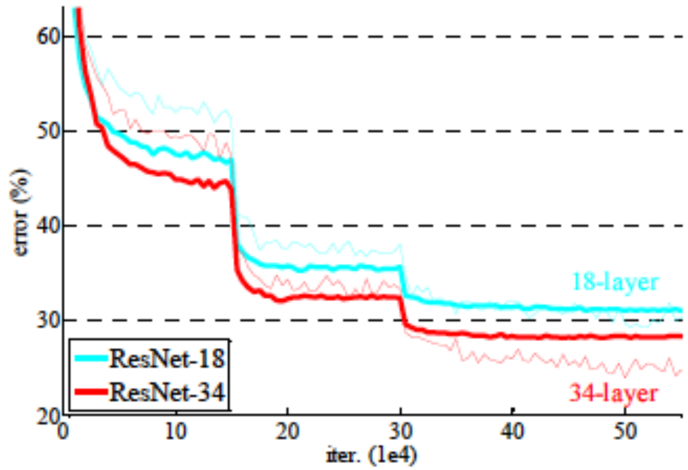


Resnet



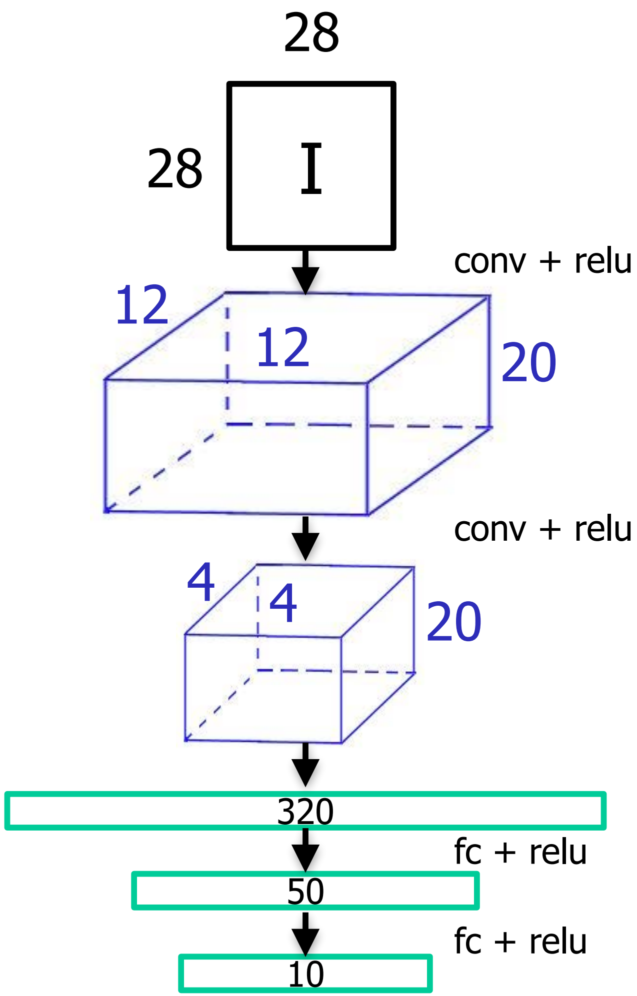
$$x+l_2(\sigma(l_1(x)))$$





He et al. , CVPR’16

Without Max Pooling

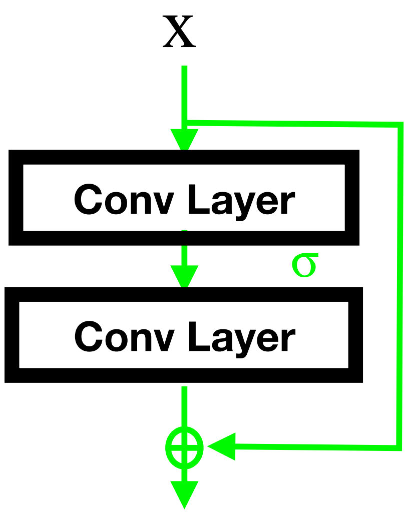


Accuracy	Train	Test
Conv 5x5, stride 1 Max pool 2x3	99.58	98.77
Conv 5x5, stride 2	99.42	98.31
Conv 5x5, stride 1 Conv 3x3, stride 2	99.38	98.57

Max pooling can be replaced by Gaussian convolutions with stride > 1 .

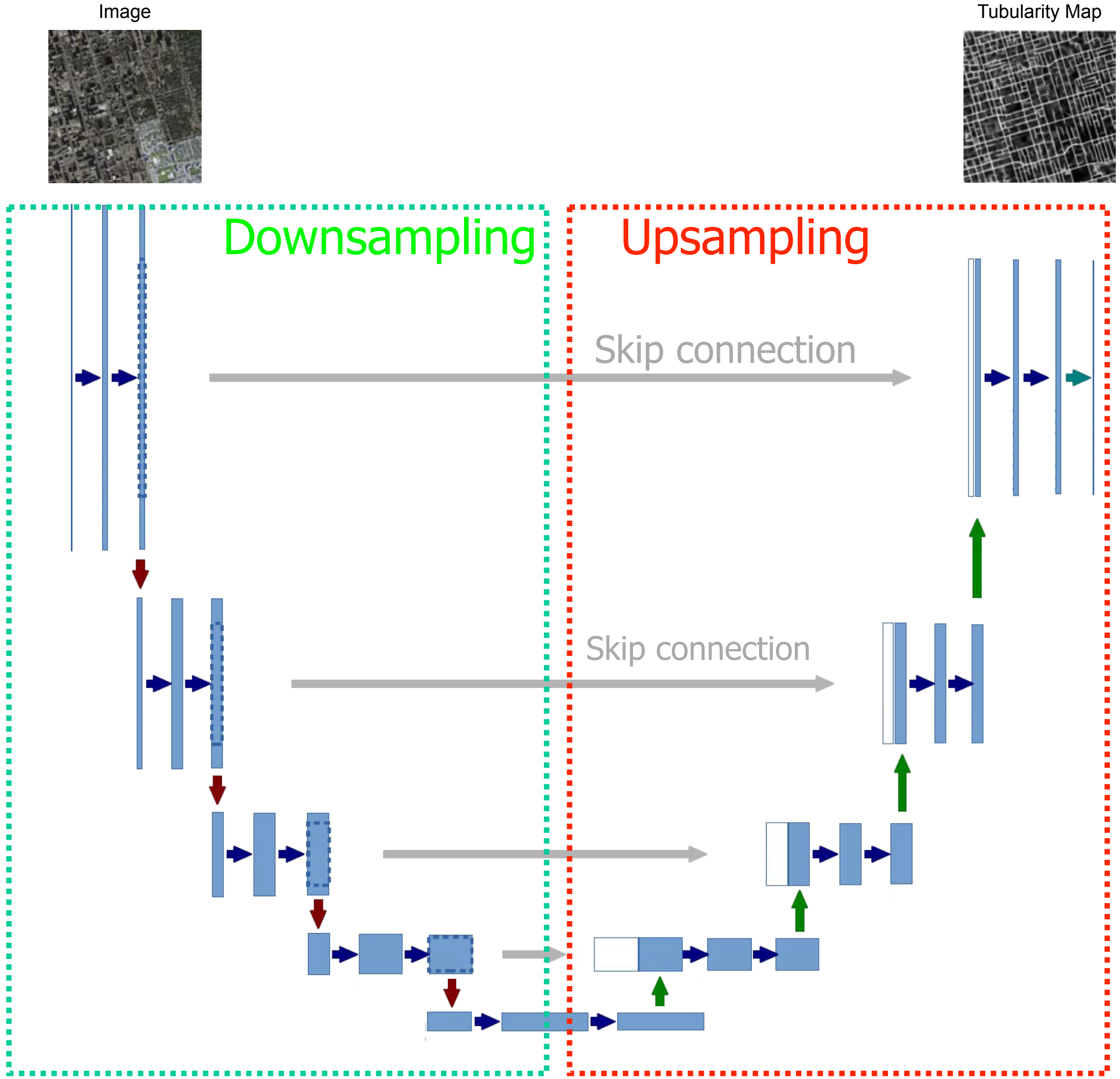
Springenberg et al., ICLR'15

ResNet to U-Net



$x + l_2(\sigma(l_1(x)))$

ResNet block



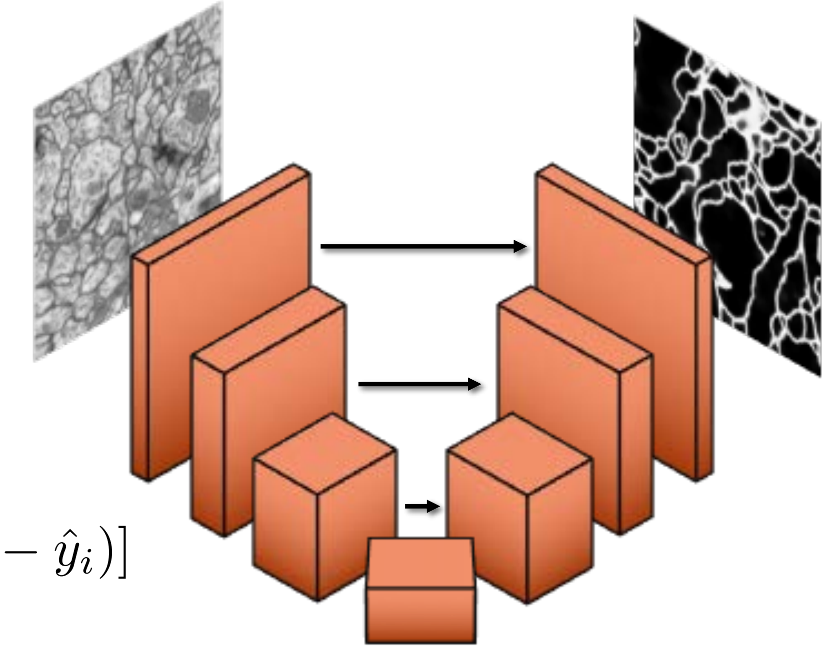
U-Net

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

Ronneberger, MICCAI'15

Training a U-Net

Train Encoder-decoder U-Net architecture using binary cross-entropy



Minimize

$$L_{bce}(\mathbf{x}, \mathbf{y}; \mathbf{w}) = -\frac{1}{i} \sum_{1}^{P} [y_{\,\mathrm{i}} \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

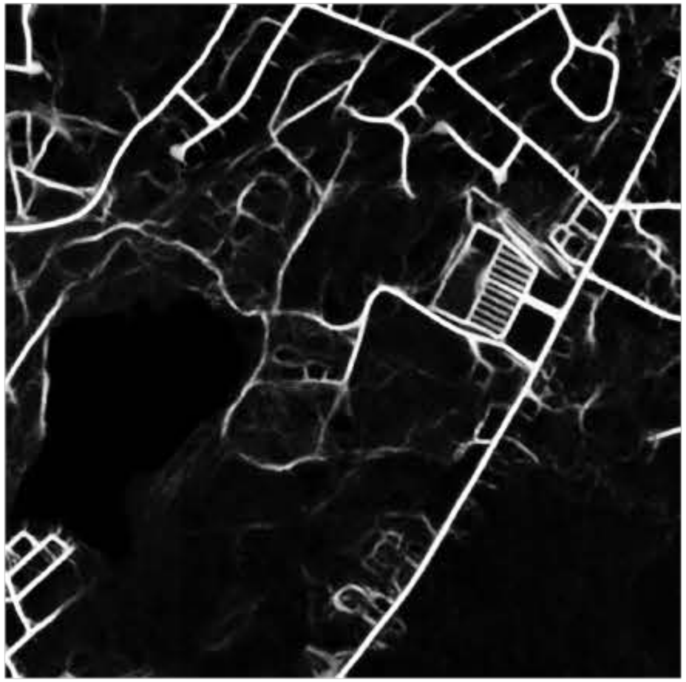
where

- $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$,
- $\mathbf{x}$ in an input image,
- $\mathbf{y}$ the corresponding ground truth.

Mosinska et al, CVPR'18.

Network Output

Image



BCE Loss

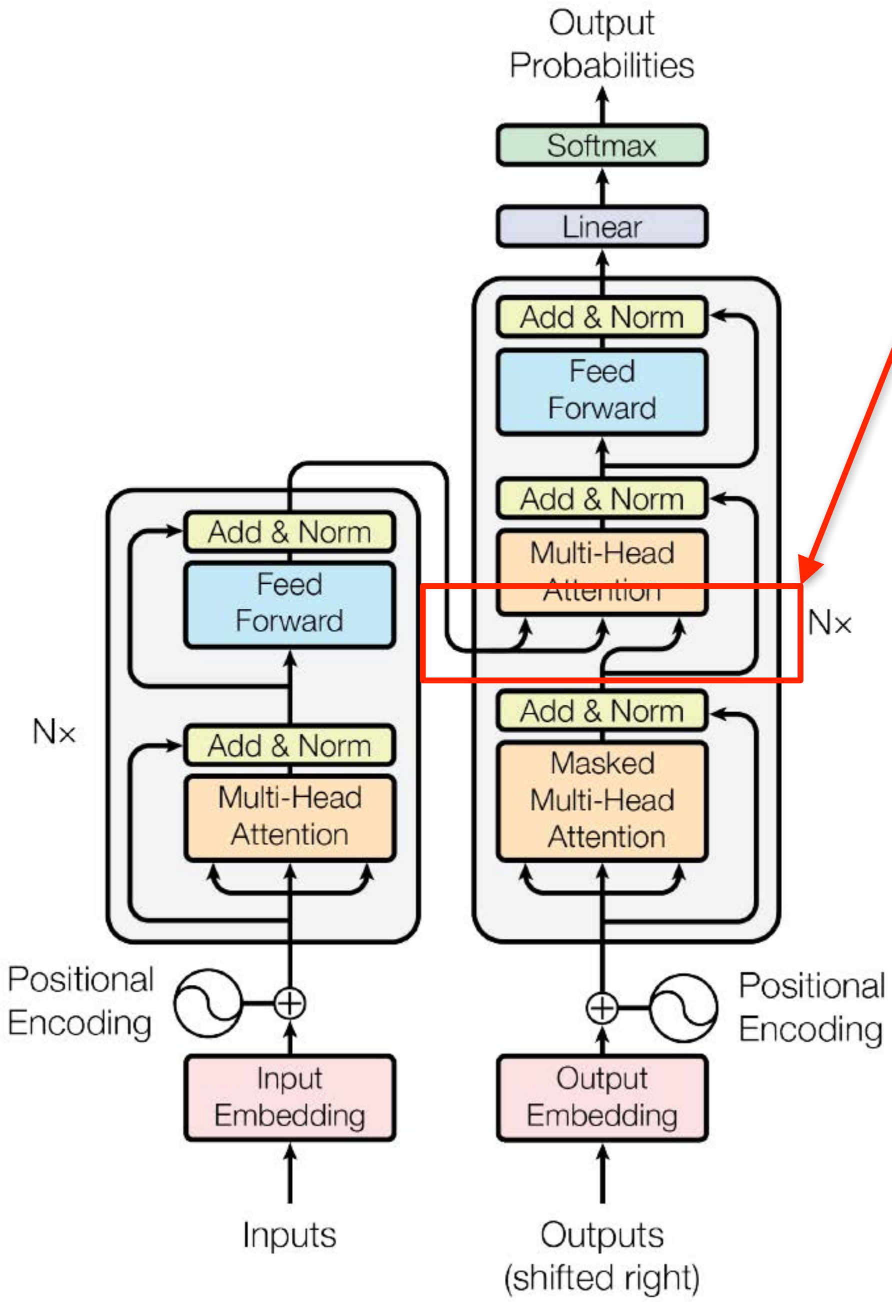
Ground truth

- Good start but not the end of the story.
- We will discuss this again during the delineation lecture.

EPFL

Streets Of Toronto

Language Transformers



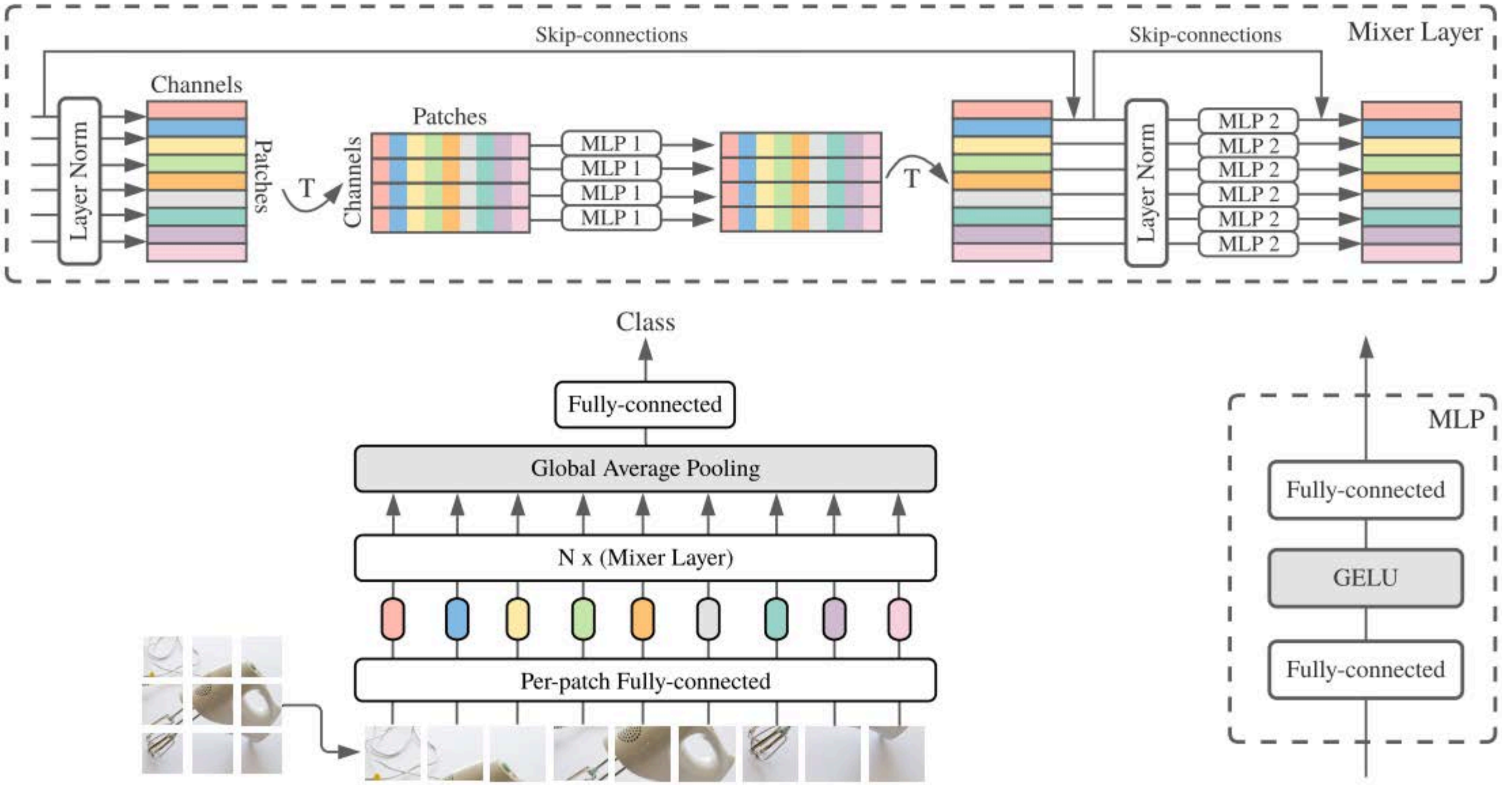
Vaswani'17

- At this point, the transformer layer is fed both the prompt and the already generated text.
- It uses this information to guess the next word.
- The process is then iterated.

Keys to LLM successes:

- The network looks as far back as needed.
- It uses a huge corpus.
- Human guided training.
- DeepSeek has shown that the last may not be needed.

Vision Transformers

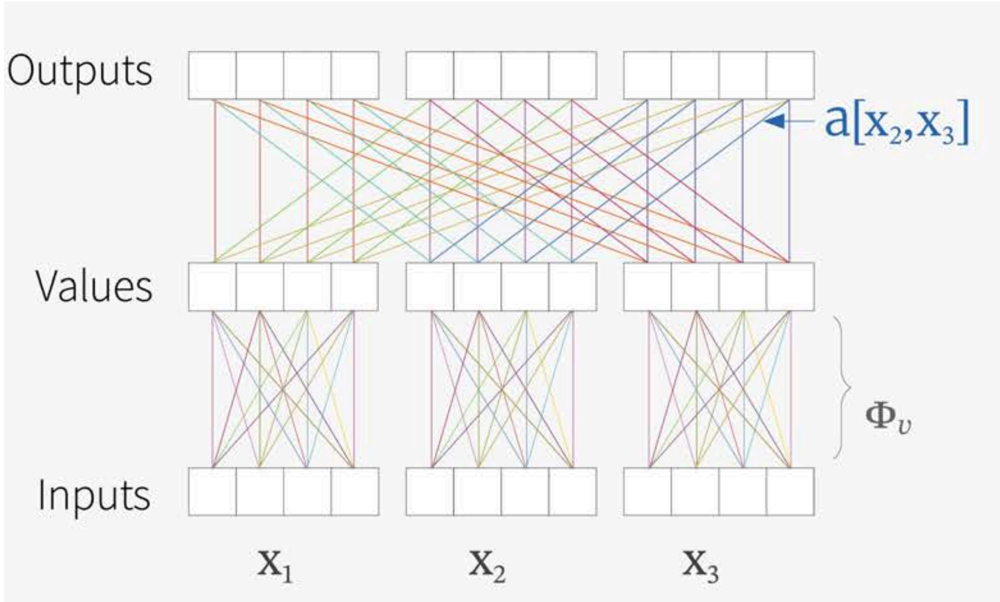


- Break up the images into square patches.
- Transform each path into a feature vector.
- Feed to a transformer architecture.

EPFL

Tolstikhin et al. , ArXiv'21

Self Attention

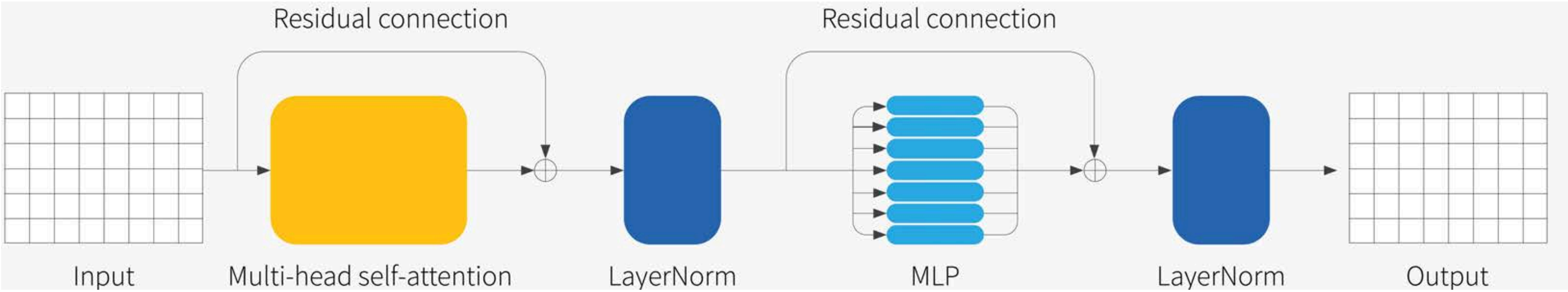


Given $\mathbf{X} = [\mathbf{x_1}, \ldots, \mathbf{x_I}]$:

- $a[\mathbf{x_i}, \mathbf{x_j}]$ is the attention that $\mathbf{x_i}$ gives to $\mathbf{x_j}$. It measures the influence of one on the other.
- It can be computed for all I and j using far fewer weights that in a fully connected layer.

—> Provides context.

Transformer Layer



$$\mathbf{X} \leftarrow \mathbf{X} + Sa(\mathbf{X})$$

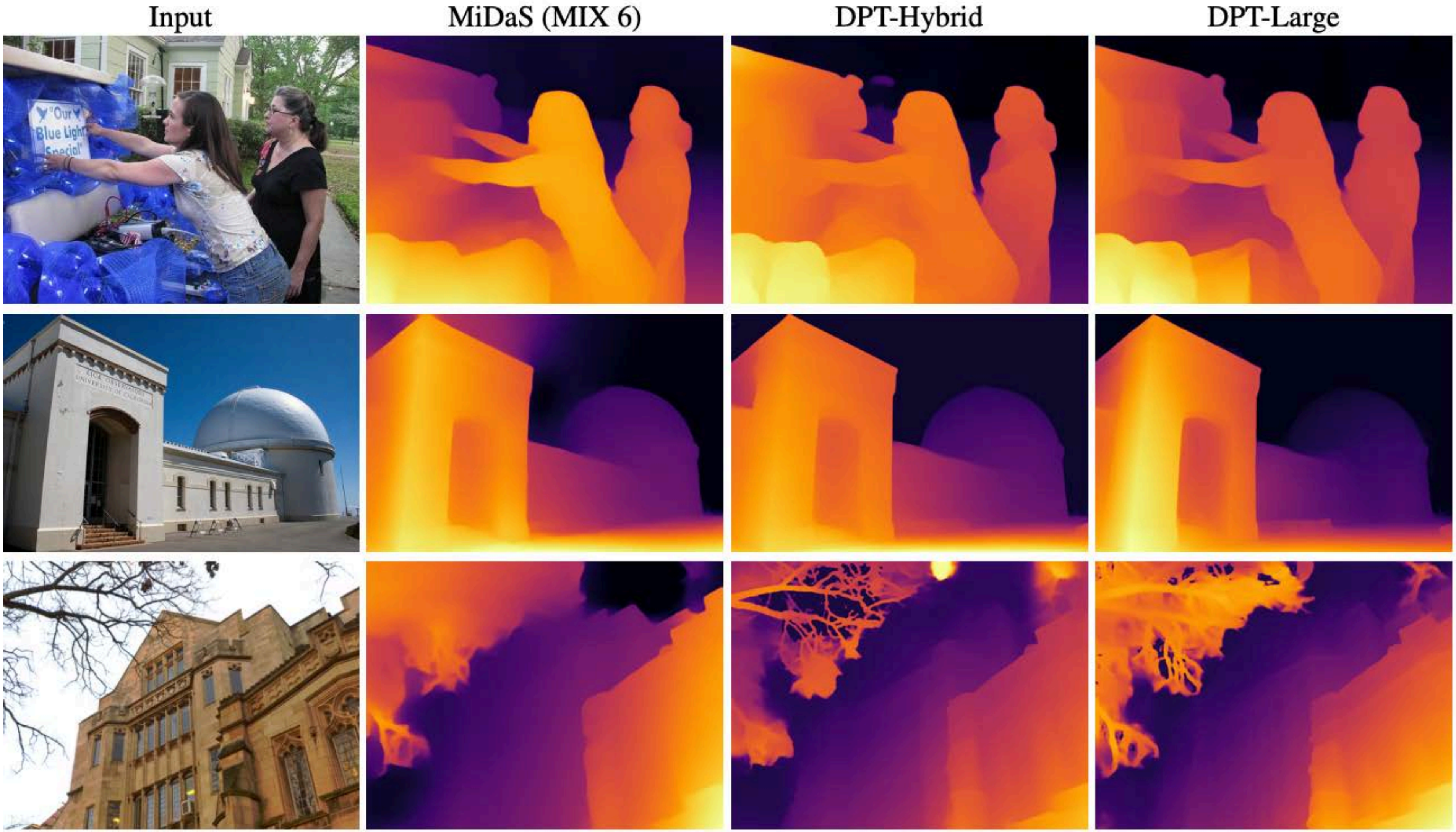
$$\mathbf{X} \leftarrow LayerNorm(\mathbf{X})$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + mlp[\mathbf{x}_i] \quad \forall i$$

$$\mathbf{X} \leftarrow LayerNorm(\mathbf{X})$$

EPFL

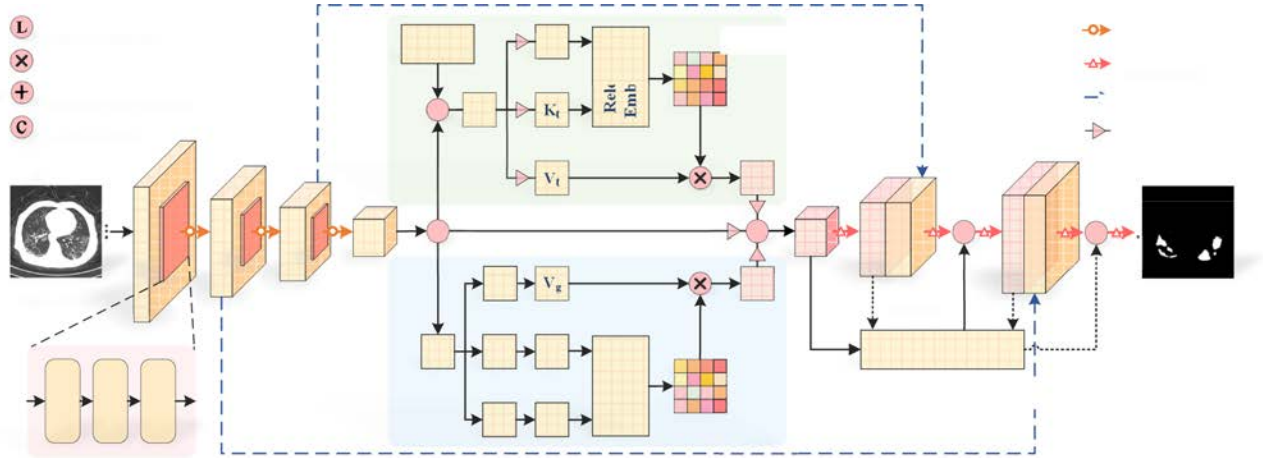
Depth from Single Images

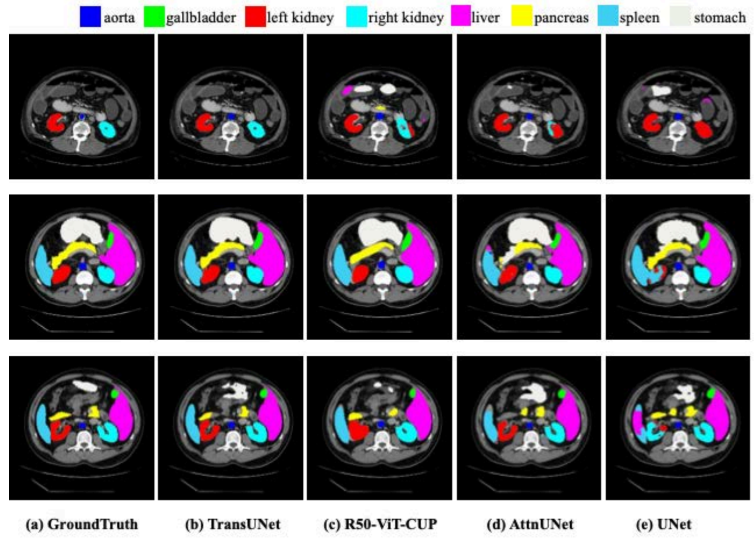


- Pros: Good at modeling long range relationships.
- Cons: Flattening the patches looses some amount of information.

Ranftl et al. , CVPR'21

U-NET + Transformers





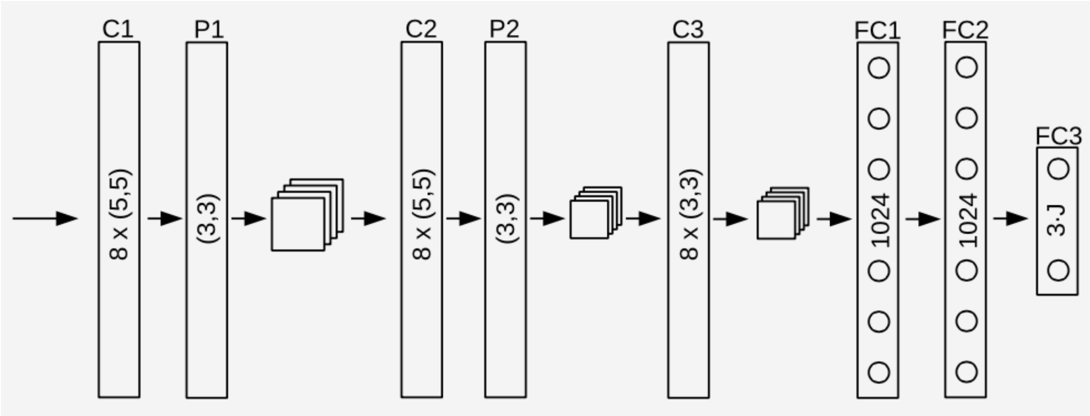
- A CNN produces a low-resolution feature vector.
- A transformer operates on that feature vector.
- The upsampling is similar to that of U-Net

—> Best of both worlds?

Chen et al., TETCI’23

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,
- ....

Body Pose Estimation

Pavlakos, CVPR'19

People and their Clothes

- Regress the body pose.
- Drape the garments on the body.
- Account for garment motion.
- Enforce consistency.

Li et al., SIGGRAPH'25

Crowd Counting

EPFL at lunchtime: The colors denote crowd density.

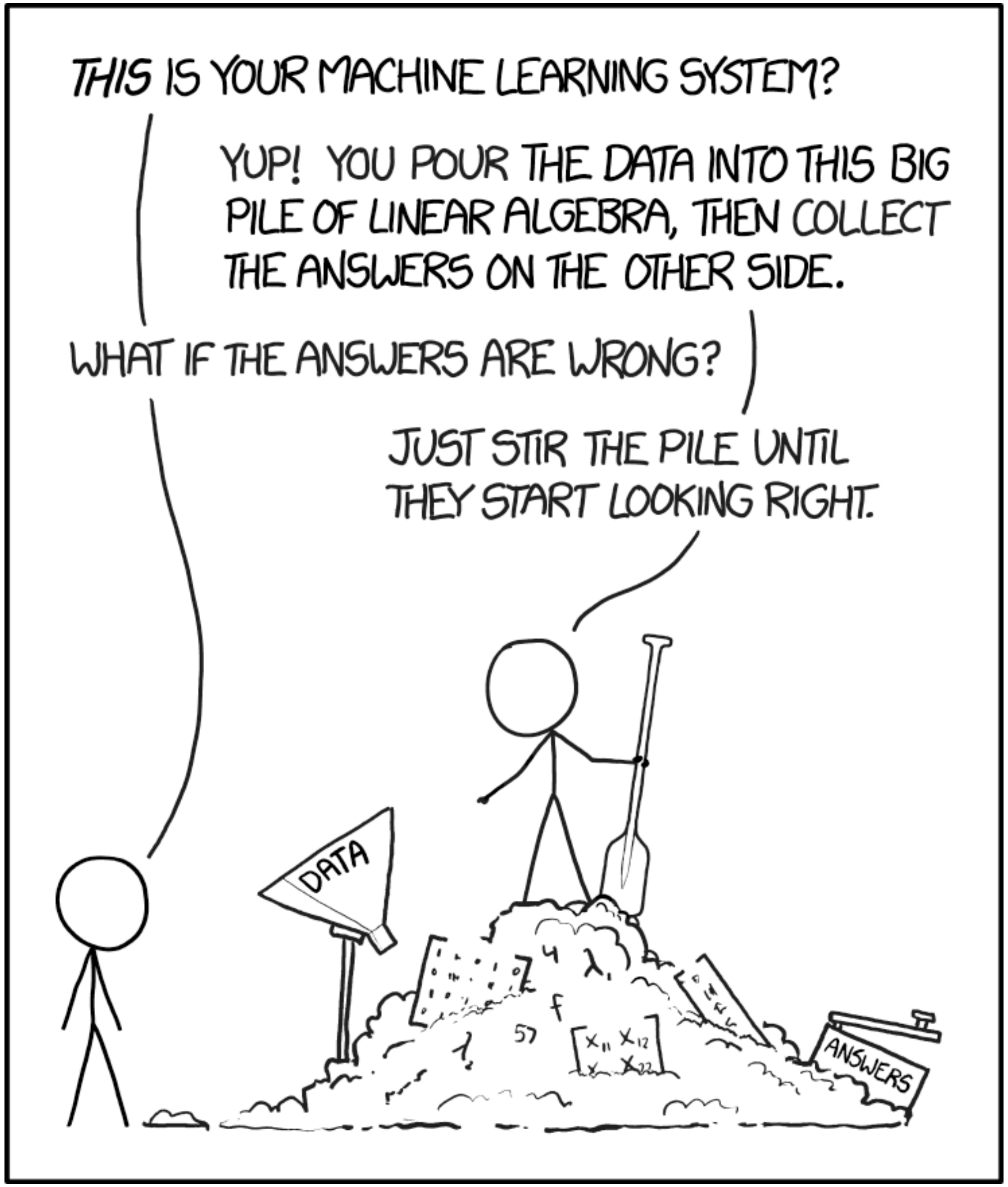
Liu et al. , CVPR’22

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.

XKCD's View On The Matter



https://xkcd.com/

EPFL

Deep Nets in Short

- Deep Neural Networks can handle huge training databases, which yields impressive performance.
- When there is not so much training data, domain knowledge must be used.
- There are failure cases and performance is hard to predict.

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

What Does it Mean for Vision?

Two distinct approaches to increasing performance:

- Use ever larger training databases.
  - How do build them?
  - How do we tame the computational explosion?

- Use existing knowledge to reduce the need for training data.
  - Physics-based knowledge.
  - Geometrical knowledge.

—> Self-supervised methods.