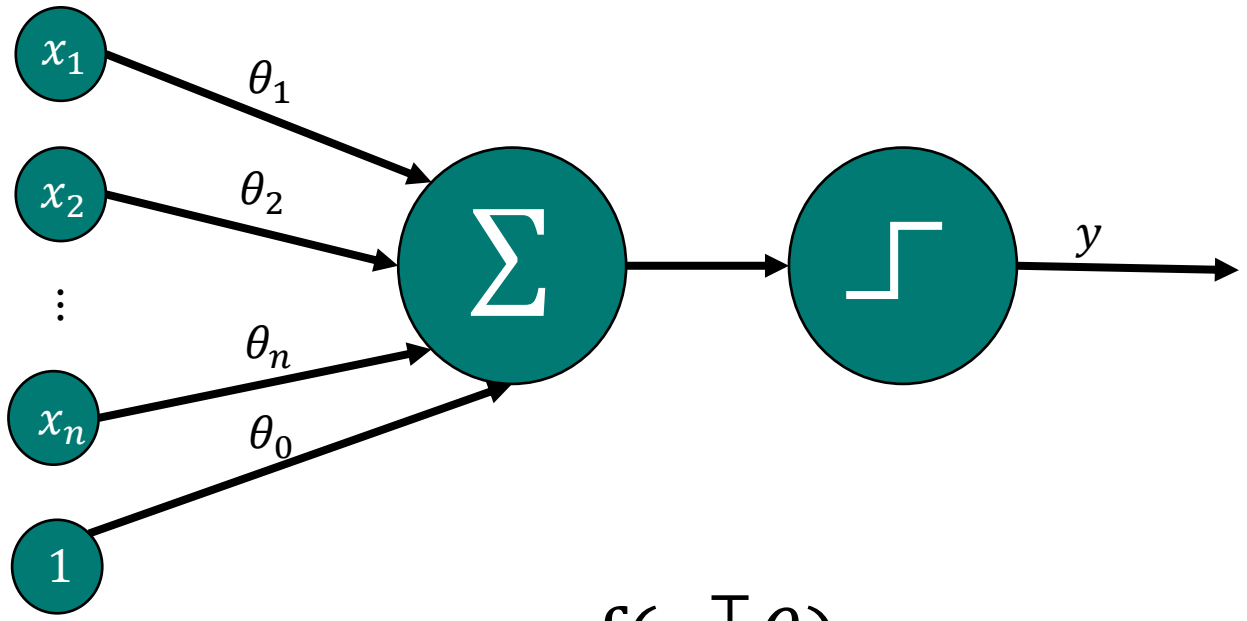
Mathematical Foundations of Deep Learning

Introduction

- Perceptron (McCulloch, Pitts, 1943; Rosenblatt 1957):

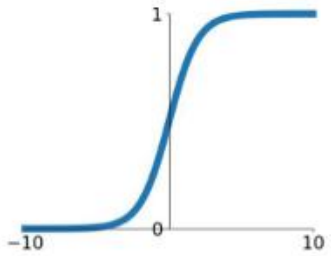


$$y = \mathrm{f}(x^{\top}\theta)$$

Activation Functions

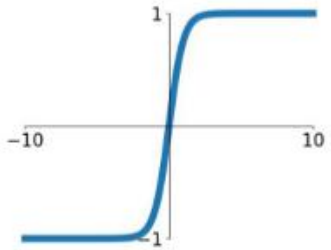
Sigmoid

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



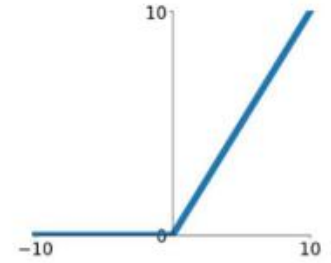
tanh

$$\tanh(x)$$



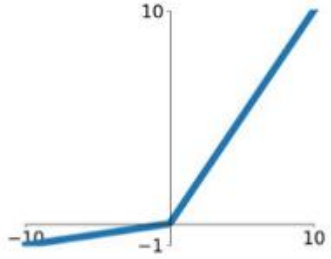
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

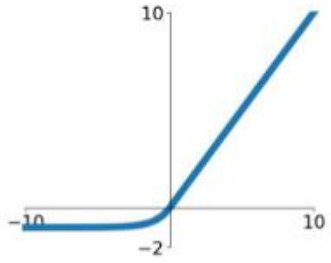


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

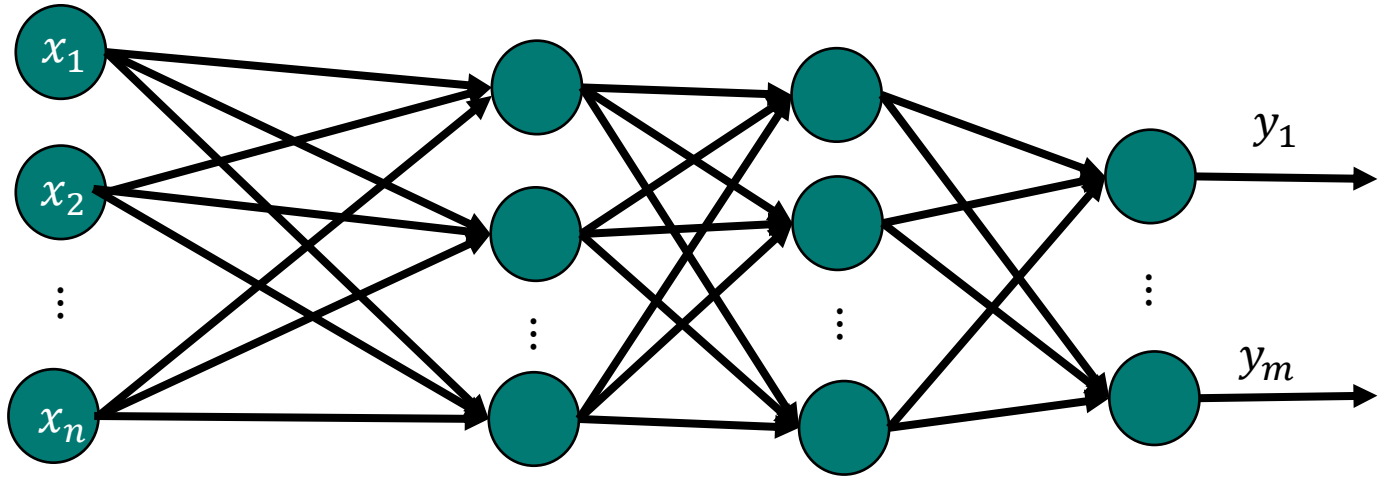
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Source: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Introduction

- Obviously, networks with hidden layers are more interesting than the perceptrons:

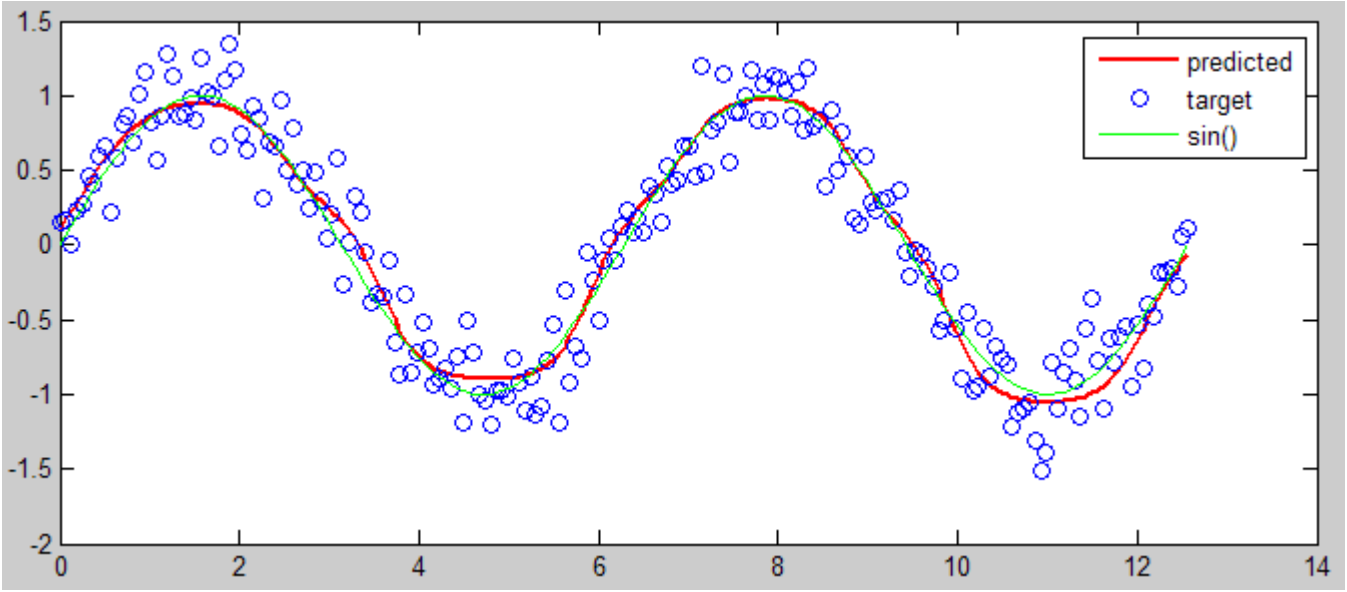


Introduction

- At its core, the problem of the neural network training is simply an approximation task; we are trying to approximate the function $\phi(x)$ by the composition of generalized linear models: $\hat{\phi}(x;\theta) = \mathrm{f}^{(\mathrm{L})}(\ldots f^{(2)}\left(f^{(1)}\left(x,\theta^{(1)}\right),\theta^{(2)}\right),\theta^{(L)})$, where $f^{(i)}$ is an activation function and $\theta^{(\mathrm{i})}$ is a weight vector on i-th layer.

Introduction

- At its core, the problem of the neural network training is simply an approximation task; we are trying to approximate the function $\phi(x)$ by the composition of generalized linear models: $\hat{\phi}(x;\theta) = \mathrm{f}^{(\mathrm{L})}(\ldots f^{(2)}(f^{(1)}(x,\theta^{(1)}),\theta^{(2)}),\theta^{(L)})$, where $f^{(i)}$ is an activation function and $\theta^{(\mathrm{i})}$ is a weight vector on i-th layer.



Source: https://stackoverflow.com/questions/1565115/approximating-function-with-neural-network

Introduction

- Activation functions on specific layers are chosen beforehand (they are **hyperparameters** of the model).
- Our task is to find the optimal values of weights θ , so that the approximation $\hat{\phi}$ will be as close as possible to the approximated function ϕ.
- Unfortunately, we cannot do this directly – we do not know the real form of ϕ, only some realizations of this function (collected data).
- As a result, we must introduce some performance measure (**cost function**) $\mathrm{J}(\theta)$ and optimize θ indirectly.

Objective

- Cost function $\mathrm{J}(\theta)$ is defined as an expected value of the **loss function** L:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \mathrm{L}(\hat{\phi}(x;\theta), y)$$

  where $\mathcal{D}$ is a distribution of variables x and y.

- Our goal is to **minimize** the value of $\mathrm{J}(\theta)$.
- In other words, we are interested in finding the vector θ with the lowest possible **approximation error**.

Objective

- Cost function $\mathrm{J}(\theta)$ is defined as an expected value of the **loss function** L:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}}\, \mathrm{L}(\hat{\phi}(x;\theta), y)$$

  where $\hat{\mathcal{D}}$ is an empirical distribution of variables x and y.

- Our goal is to **minimize** the value of $\mathrm{J}(\theta)$.
- In other words, we are interested in finding the vector θ with the lowest possible **approximation error** and **estimation** (***generalization***) **error**.

Definition of Machine Learning

Definition of Machine Learning (Vapnik, 1999):

For a given class of functions $\mathcal{F} = \{\alpha \in \Lambda: \hat{f}(x, \alpha)\}$, data generating process $\mathcal{D} = (X, Y)$ and loss function $L(Y, \hat{Y})$ one must solve the following problem:

$$\hat{\alpha} = \underset{\alpha \in \Lambda}{\operatorname{argmin}} \left(\mathbb{E} \left(L\left(\hat{f}(x; \alpha), y \right) \right) \right)$$

given a training set $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ of independent and identically distributed (i.i.d.) observations drawn from $\mathcal{D}$.

Objective

Table 1: List of losses analysed in this paper. $\mathbf{y}$ is true label as one-hot encoding, $\hat{\mathbf{y}}$ is true label as +1/-1 encoding, $\mathbf{o}$ is the output of the last layer of the network, $.^{(j)}$ denotes jth dimension of a given vector, and $\sigma(\cdot)$ denotes probability estimate.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2 + \|\mathbf{y}\|_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

Source: (Janocha, Czarnecki 2017): https://arxiv.org/pdf/1702.05659.pdf

Optimization

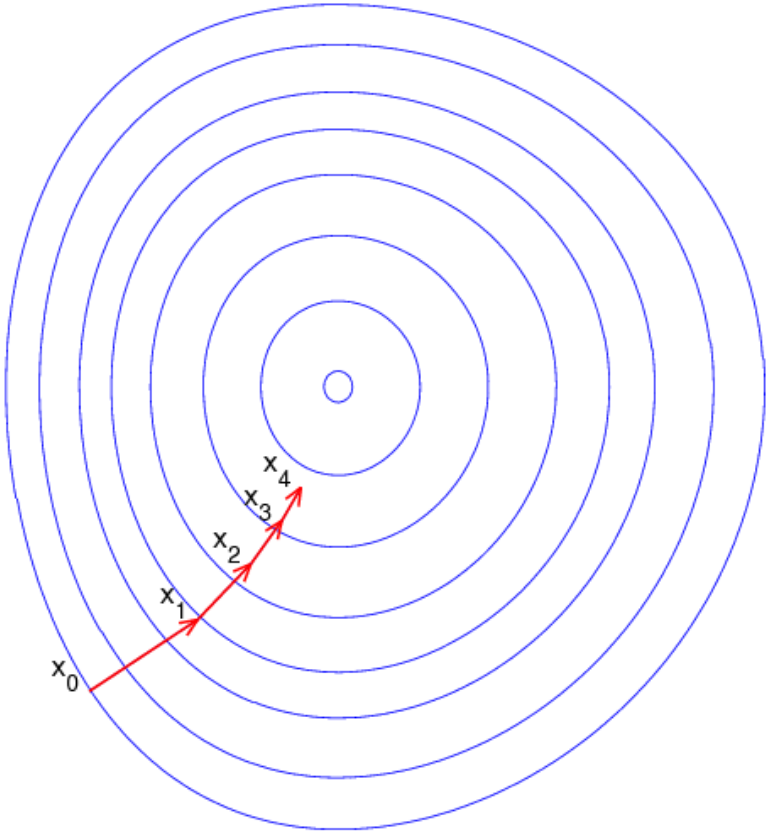
- In most cases we do not want to use **deterministic** optimization algorithms – they will be slow, ineffective and they will almost surely increase the generalization error.
- **Stochastic algorithms** are a much better solution.

Optimization

- In the most cases, we will use a **Stochastic Gradient Descent (SGD)** algorithm:

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while



Optimization

- To train the model efficiently, we must carefully choose the proper values of a **learning rate η**.
- Usually, the learning rate will decrease over time.

Optimization

- To converge, η must satisfy the following requirements:

$$\sum_{i=1}^{k} \eta_i = \infty$$

$$\sum_{i=1}^{k} \eta_i^2 < \infty$$

Generalization Error

- For the loss function:

$$\mathrm{J}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\, \mathrm{L}(\hat{\phi}(x;\theta), y)$$

- And empirical risk:

$$\hat{J}\,(\theta) = \frac{1}{k}\sum_{i=1}^{k} \mathrm{L}(\hat{\phi}(x_i;\theta), y_i)$$

- Generalization error is defined as:

$$\hat{J}\,(\theta)\; - \mathrm{J}(\theta)$$

Hardt's Theorem

Hardt M., Recht B., Singer Y. *Train faster, generalize better: Stability of stochastic gradient descent,* Mathematics of Control, Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, s. 1225-1234, 2016

- Parametric models trained by a stochastic gradient method with a decreasing learning rate have vanishing generalization error after a few iterations (*epochs*).

Hardt's Theorem

- Lets $A(S)$ be an algorithm generating models on a sample $S = (z_1, z_2, \ldots, z_n)$ drawn from the distribution $\mathcal{D}$. Then the upper bound of the generalization error might be represented as:

$$\mathbb{E}_{\mathrm{S,A}}\left[\hat{J}\left(A(S)\right) - \mathrm{J}(A(S))\right] \leq \epsilon_s(A, n)$$

Hardt's Theorem

Theorem:

Assume that the cost function $J(\cdot, y)$ is β-smooth (its gradient $\nabla J(\cdot, y)$ is β-Lipschitz function), convex and L-Lipschitz function $\forall\, y$. Suppose that we run the stochastic gradient method T times with step sizes $\eta_t \leq 2/\beta$. Then:

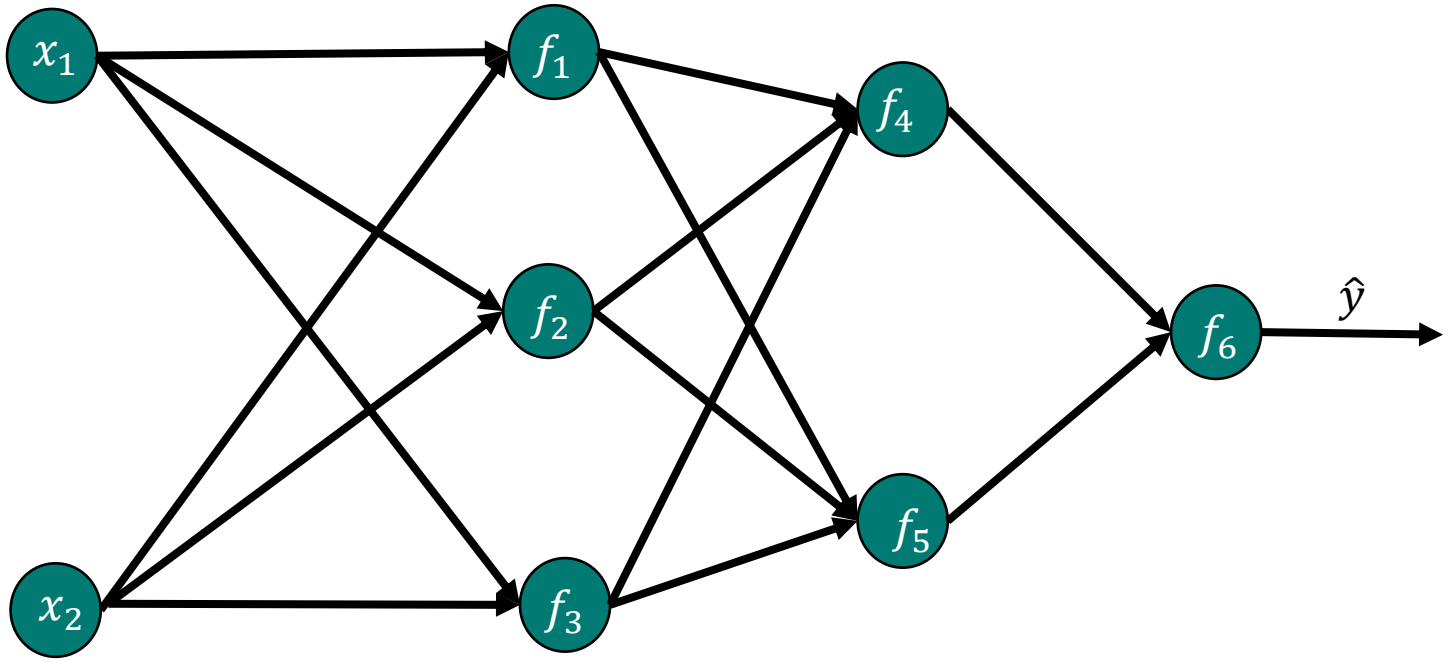
$$\epsilon_s = \frac{2L^2}{n} \sum_{t=1}^{T} \eta_t$$

Backpropagation

- **Backpropagation** is the most widely used algorithm for training feedforward neural networks.

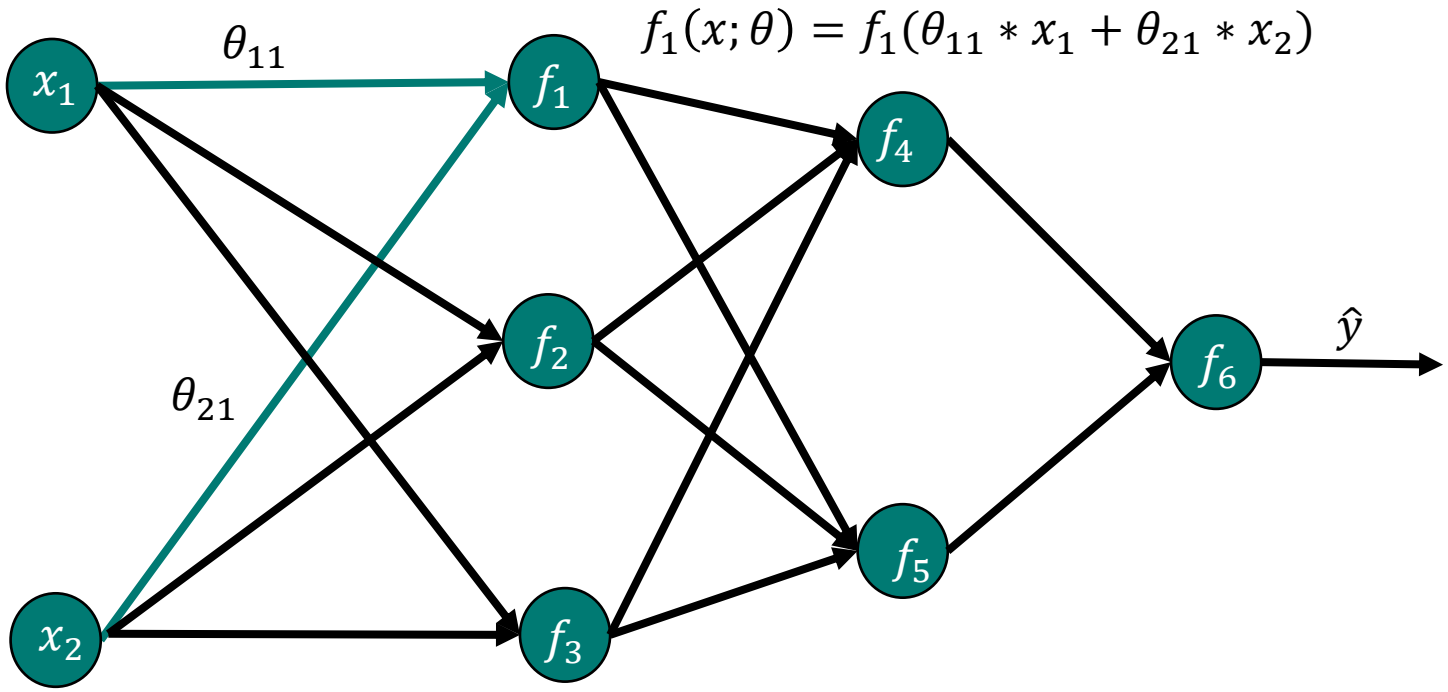
Backpropagation

- Propagation of data through the neural network is pretty intuitive:



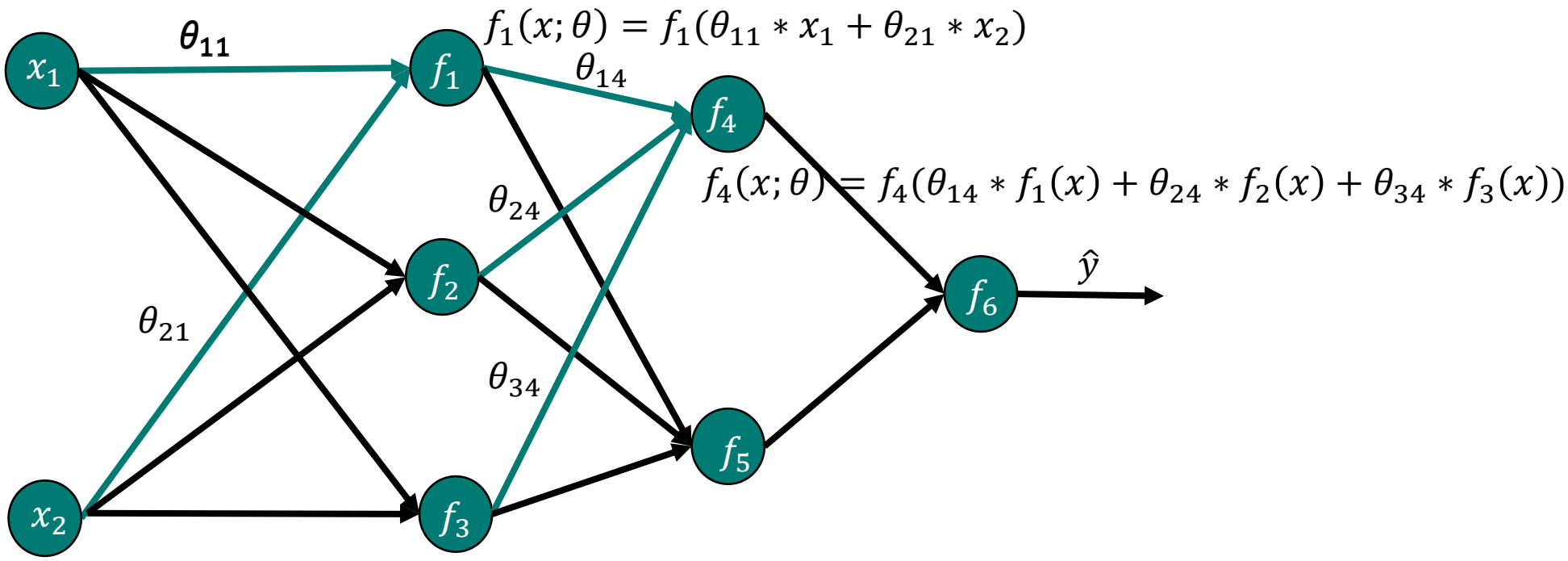
Backpropagation

- The value on each neuron is a linear combination of data from the previous layer with nonlinearity introduced by the activation function:



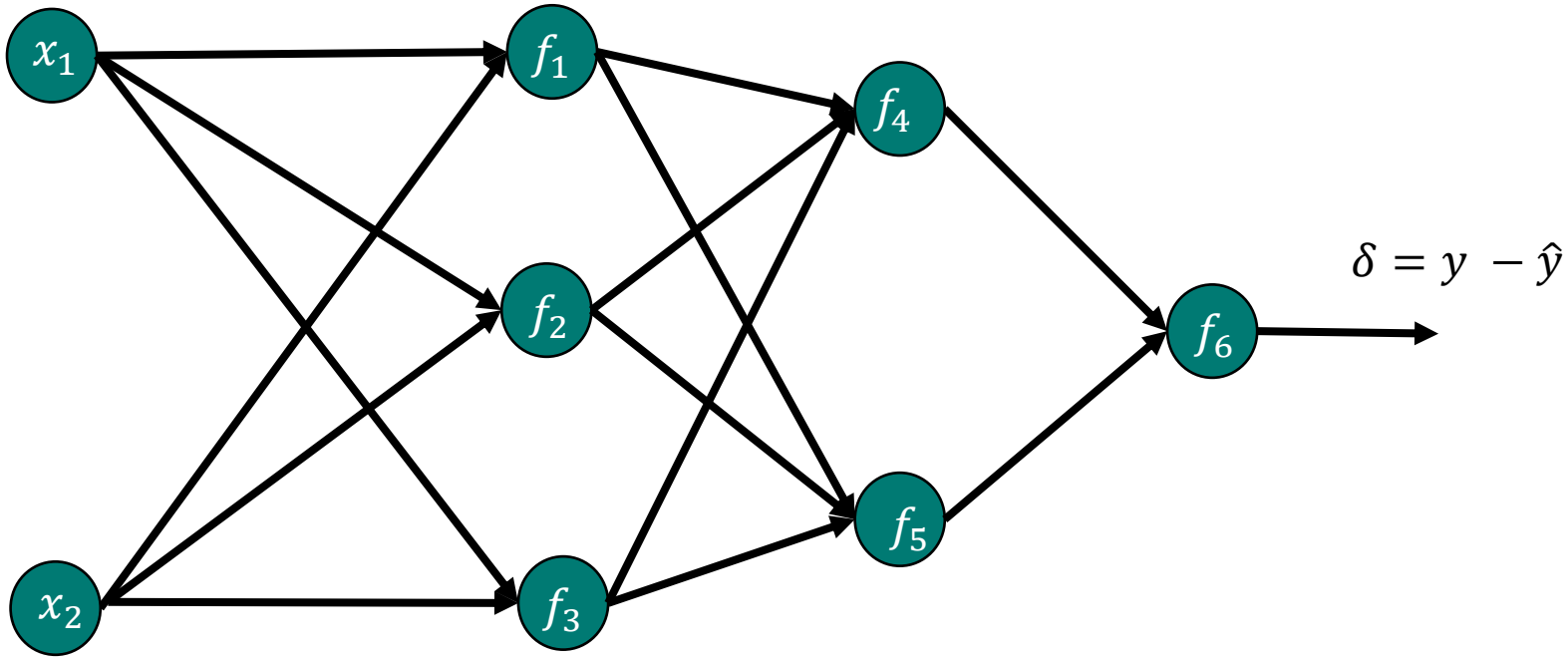
Backpropagation

- The value on each neuron is a linear combination of data from the previous layer with nonlinearity introduced by the activation function:



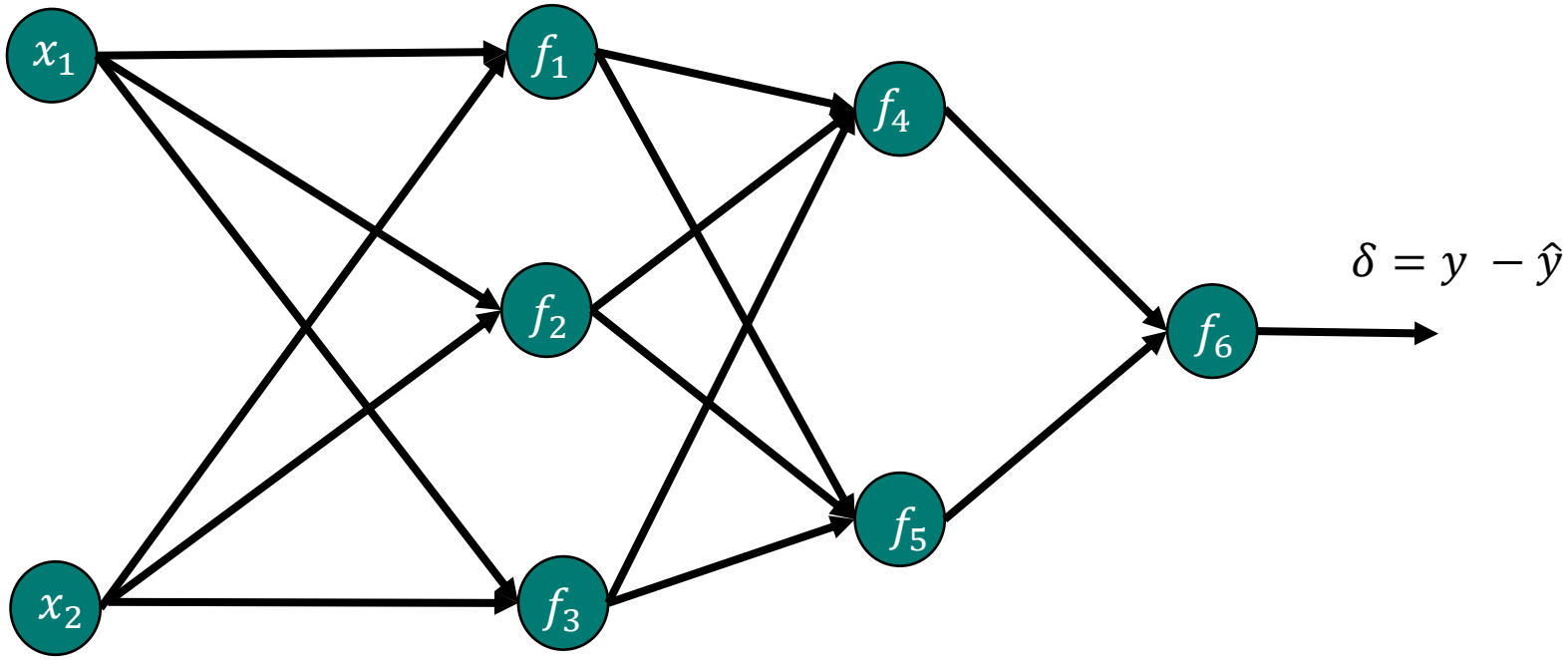
Backpropagation

- But weights updates are non-trivial; we can compute the error on the output:



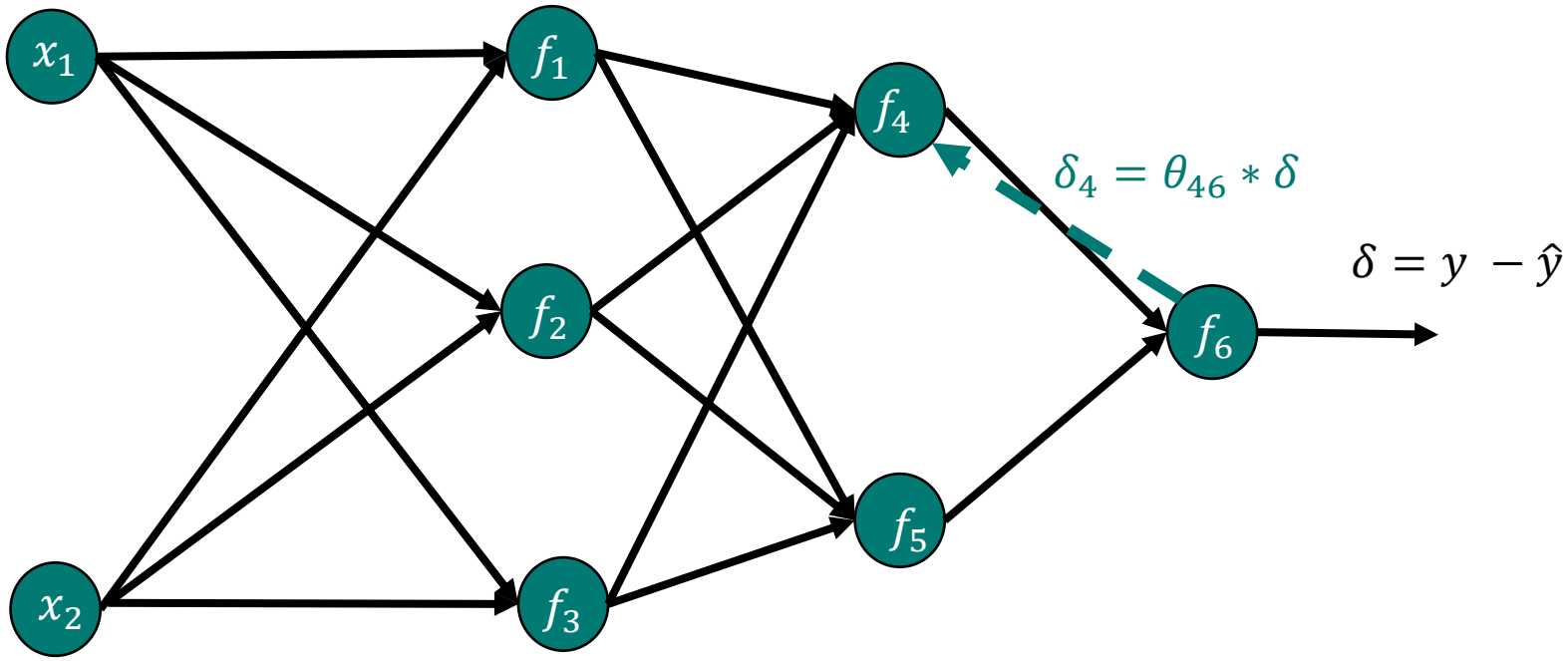
Backpropagation

- To compute the error on the particular neurons we must traverse the graph in the backward direction:



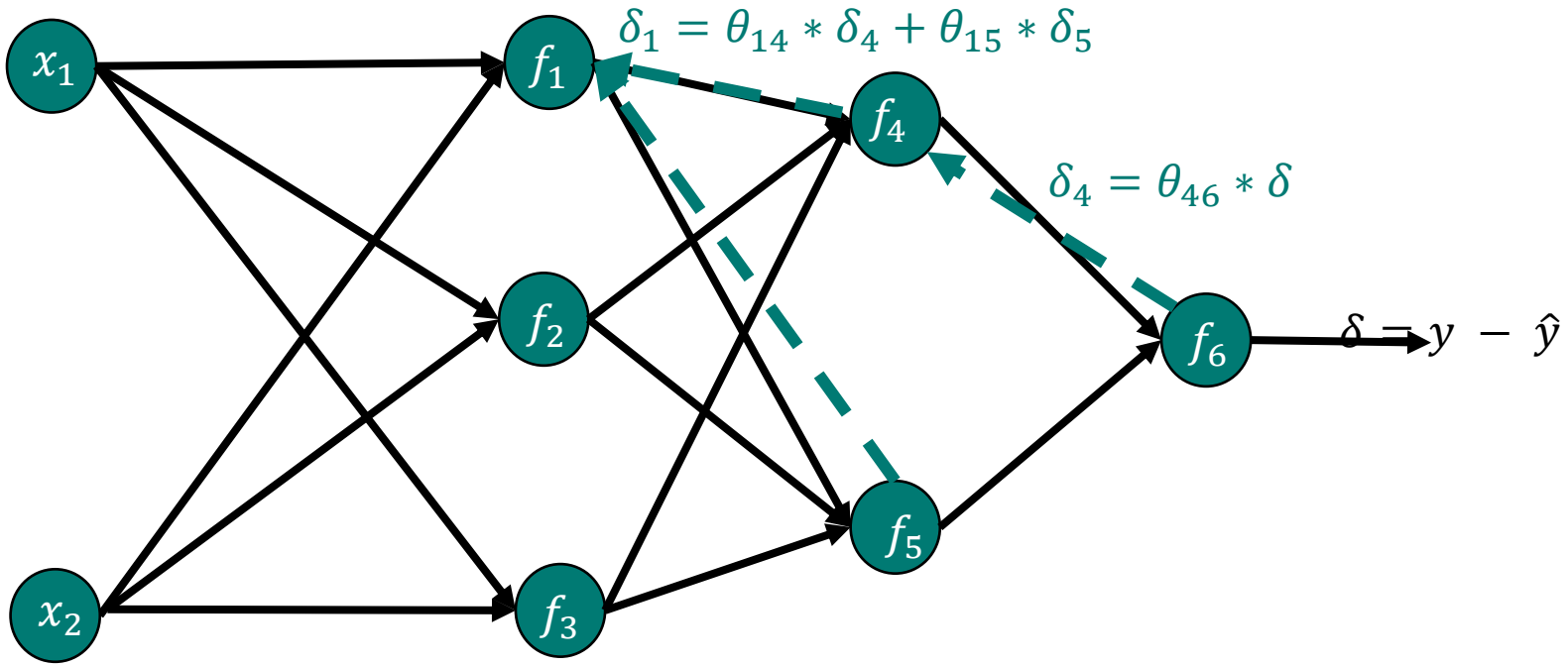
Backpropagation

- To compute the error on the particular neurons we must traverse the graph in the backward direction:



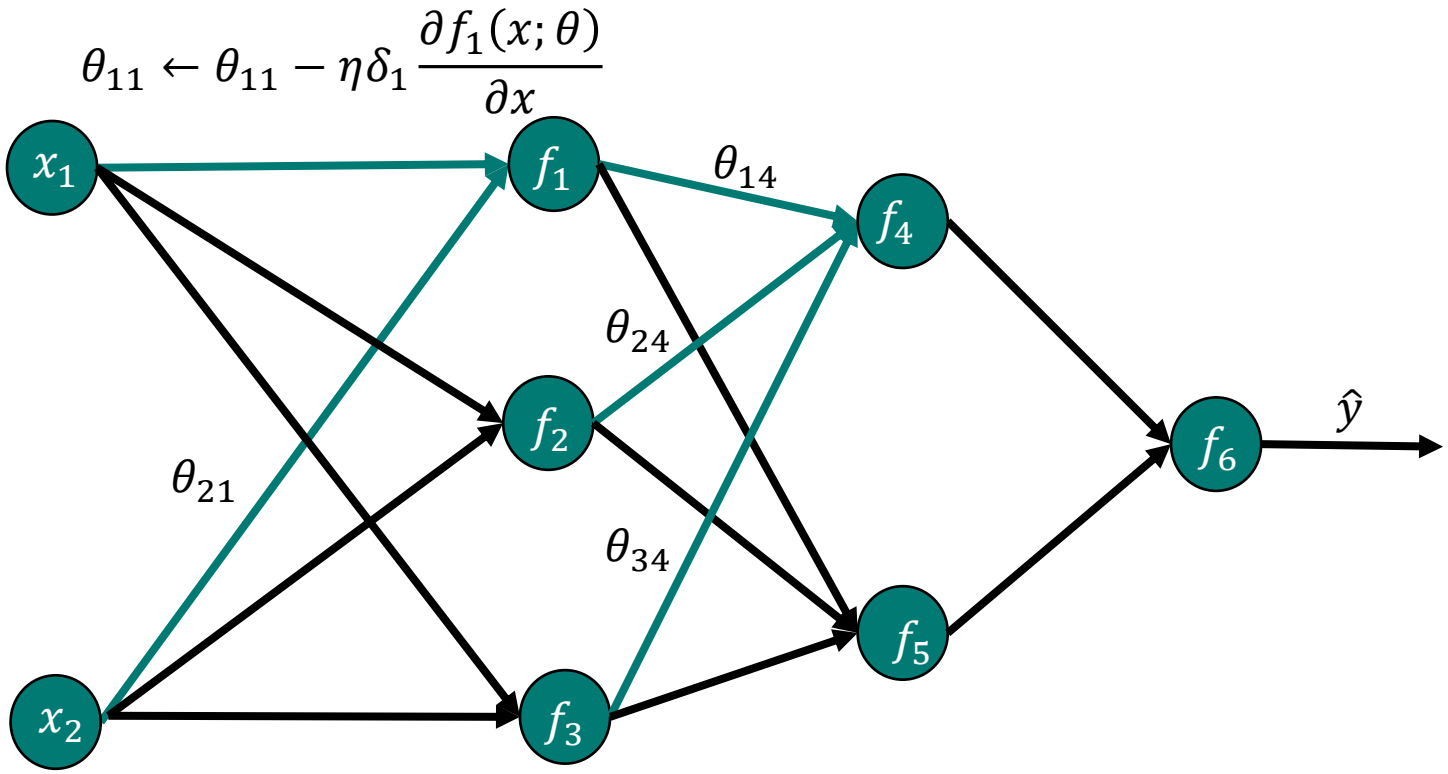
Backpropagation

- To compute the error on the particular neurons we must traverse the graph in the backward direction:



Backpropagation

- Then, the weights are updated by the stochastic gradient method:



Neural Network as an Approximation Tool

- Why the neural nets are so efficient compared to other algorithms?
- It turns out that even the simplest, non-trivial feedforward neural network with one hidden layer and sigmoidal activation function $\sigma(x)$:

$$\sigma(t) \to \begin{cases} 1 \text{ when } x \to +\infty \\ 0 \text{ when } x \to -\infty \end{cases}$$

can approximate any function $f(x)$ with a given precision ϵ.

Cybenko Theorem (Universal Approximation Theorem)

- **Cybenko G., *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals, and Systems, 2, s. 303-314, 1989**
- Every d-dimensional function of real variables x $\in \mathbb{R}^d$ might be approximated with a given precision by a linear combination:

$$G(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i^T x + b_i)$$

  where $w_i \in \mathbb{R}^d$ and $\alpha_i, b_i \in \mathbb{R}$.
- This will hold if the size of the hidden layer n is large enough.

Universal Approximation Theorem

- Define:
  - $\boldsymbol{I_n}$ is a n-dimensional unit hypercube $[0,1]^n$
  - $\mathbf{C}(\boldsymbol{I_n})$ is a space of continous functions on $\boldsymbol{I_n}$
  - $||\boldsymbol{f}||$ is supremum of $\boldsymbol{f} \in \mathbf{C}(\boldsymbol{I_n})$
  - $\boldsymbol{\mathcal{M}}(\boldsymbol{I_n})$ is a space of finite, signed regular Borel measures on $\boldsymbol{I_n}$

Universal Approximation Theorem

- Function σ is **discriminatory** if for a measure $\mu \in \mathcal{M}(I_n)$

$$\int_{I_n} \sigma(w^T \mathrm{X} + b) d\mu(X) = 0 \qquad \forall\, w \in \mathbb{R}^n \; i \; b \in \mathbb{R}$$

  means that $\mu = 0$.

Universal Approximation Theorem

Theorem 1:

Let $\boldsymbol{\sigma}$ be any continuous discriminatory function. Then finite sums of the form:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

is dense in $C(I_n)$.

In other words, given any $f \in C(I_n)$ and $\varepsilon > O$ there is a sum $G(X)$, for which:

$$\|G(X) - f(X)\|_\infty < \varepsilon$$

Universal Approximation Theorem

Theorem 2:

Let $\boldsymbol{\sigma}$ be any continuous sigmoidal function. Then finite sums of the form:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

is dense in $C(I_n)$.

In other words, given any $f \in C(I_n)$ and $\varepsilon > O$ there is a sum $G(X)$, for which:

$$\|G(X) - f(X)\|_\infty < \varepsilon$$

Universal Approximation Theorem

Theorem 3:

Let $\boldsymbol{\sigma}$ be any continuous sigmoidal function. Let $\boldsymbol{f}$ be the decision function for any finite measurable partition of I_n. For any $\epsilon > 0$ there is a finite sum of the form:

$$G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$$

And set $\boldsymbol{D} \subset \boldsymbol{I_m}$ so that Lebesque measure $m(D) \geq 1 - \epsilon$ and:

$$\|G(X) - f(X)\|_\infty < \varepsilon \text{ dla } x \in D$$

Universal Approximation Theorem

Cybenko's theorem resembles one of the most important approximation theorems:

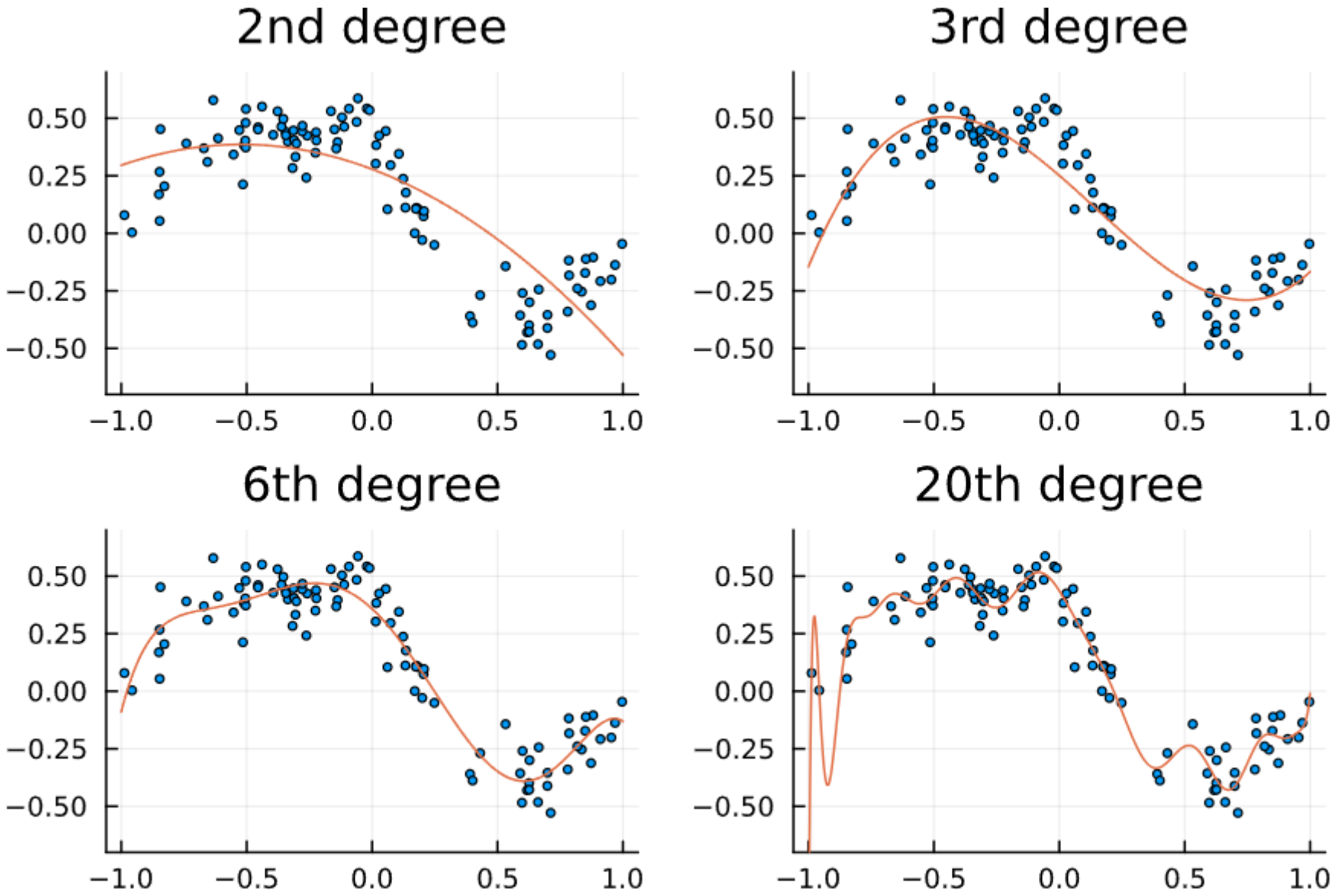
Stone-Weierstrass theorem:

Suppose $f(x)$ is a continuous real-valued function defined on the real interval $[a, b]$. For all $\epsilon > 0$ there exist a n-th degree polynomial $\mathrm{W}_n(x)$ such that:

$$|f(x) - W_n(x)| < \epsilon$$

holds for all x in $[a, b]$.

Universal Approximation Theorem



Neural Network as an Approximation Tool

- Cybenko has proven that neural networks with a sigmoidal activation function can represent a wide variety of functions. But what about the other activation functions?

Hornik's Theorem

- **Hornik K., *Approximation Capabilities of Multilayer Feedforward Networks* , Neural Networks, Vol. 4, pp. 251-257, 1991**
- Hornik extended Cybenko's results for other families of functions. He showed that the approximation capabilities of the neural network do not depend on the choice of the activation functions.

Neural Network as an Approximation Tool

- Both results are strictly theoretical; they show the approximation abilities of neural networks, without implementation details.
- For example, we still do not know how wide the layer of such a network should be.
- But there are other important theorems, which give us boundaries on both, width and depth of the specific types of neural networks.

Barron's Theorem

- **Barron A.E., Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. on Information Theory, 39, s. 930-945, 1993**
- We are still trying to approximate $f(X)$ by a linear combination: $G(X) = \sum_{i=1}^{n} a_i \sigma\left(w_i^T x + b_i\right) + a_0$
- But this time we are interested in estimating the number of neurons n on the hidden layer.

Barron's Theorem

- Let us define **MISE** *(mean integrated squared error)* as:

$$MISE = E\left|\left|G - f\right|\right|_2^2 = E\left[\int \left(G(X) - f(x)\right)^2 dx\right]$$

Barron's Theorem

- We are considering a class of functions $f\colon \mathbb{R}^d \longrightarrow \mathbb{R}$ and $f \in \Gamma_C$ for which there is a Fourier transform representatation of the form:

$$f(x) = \int_{\mathbb{R}^d} e^{i\omega \cdot x} \hat{f}(\omega) d\omega$$

for some complex valued function $\hat{f}(\omega)$ for which $\omega \hat{f}(\omega)$ is integrable. Let us define:

$$C_f = \int_{\mathbb{R}^d} \hat{f}(\omega) |\omega| d\omega$$

where $|\omega| = (\omega \cdot \omega)^{1/2}$ and Γ_C is a set of functions f, so that: $\mathrm{C}_f \leq C$ and $0 < C$.

Barron's Theorem

Theorem:

For every function $f \in \Gamma_c$, every sigmoidal function $\boldsymbol{\sigma}$, arbitrary probability measure μ on the ball $B_r = \{x: |x| \leq r\}$ and $n \geq 1$, there existist a linear combination of n sigmoidal functions of the form
$G(X) = \sum_{i=1}^{n} a_i \sigma\left(w_i^T x + b_i\right) + a_0$, such that:

$$\int \left(G(X) - f(x)\right)^2 \mu(dx) \leq \frac{\left(2rC_f\right)^2}{n}$$

and the coefficients of the linear combination $G(X)$ satisfy two conditions:

1. $\sum_{i=1}^{n} |w_i| \leq 2\mathrm{rC}$
2. $a_0 = f(0)$

Barron's Theorem

- Approximation error is of order $O\left(\frac{1}{n}\right)$ and estimation error of a feedforward neural network with one hidden layer is of order:

$$O\left(\frac{1}{n}\right) + O\left(\frac{nd}{M}\ln M\right)$$

where M is the size of the sample dataset.

  - As we can see, approximation error depends only on the width of the neural net.
  - It means that neural networks avoid the **curse of dimensionality**.

Barron's Theorem

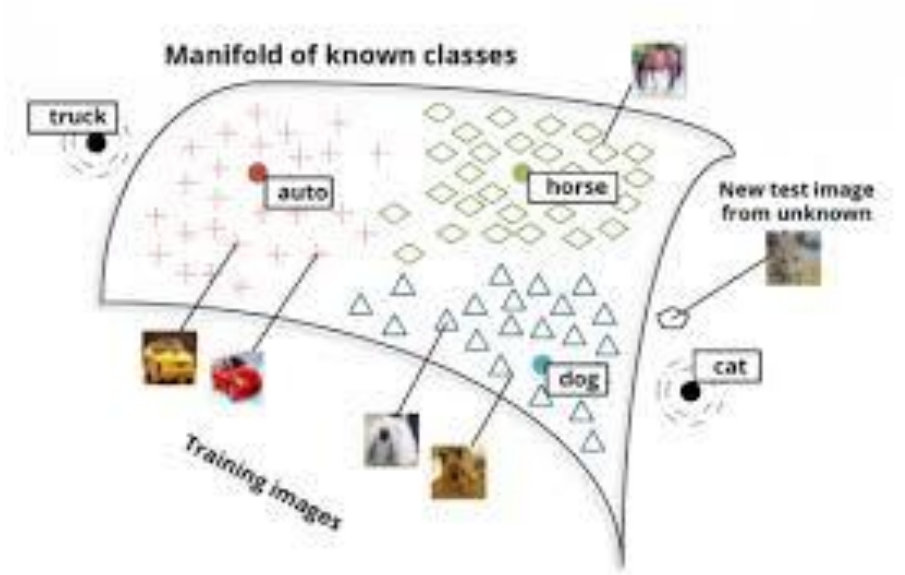
- But still, the width of the hidden layer might be a serious problem.
- In the worst case scenario, the size of the hidden layer n is equal to all possible combinations of the input data 2^d.

Neural Network as an Approximation Tool

- The best method to reduce the complexity of the neural network is to use more than one hidden layer.
- In most cases, additional layers will positively impact the quality of fit.
- The next two theorems will show us how the depth of the network impacts its properties.

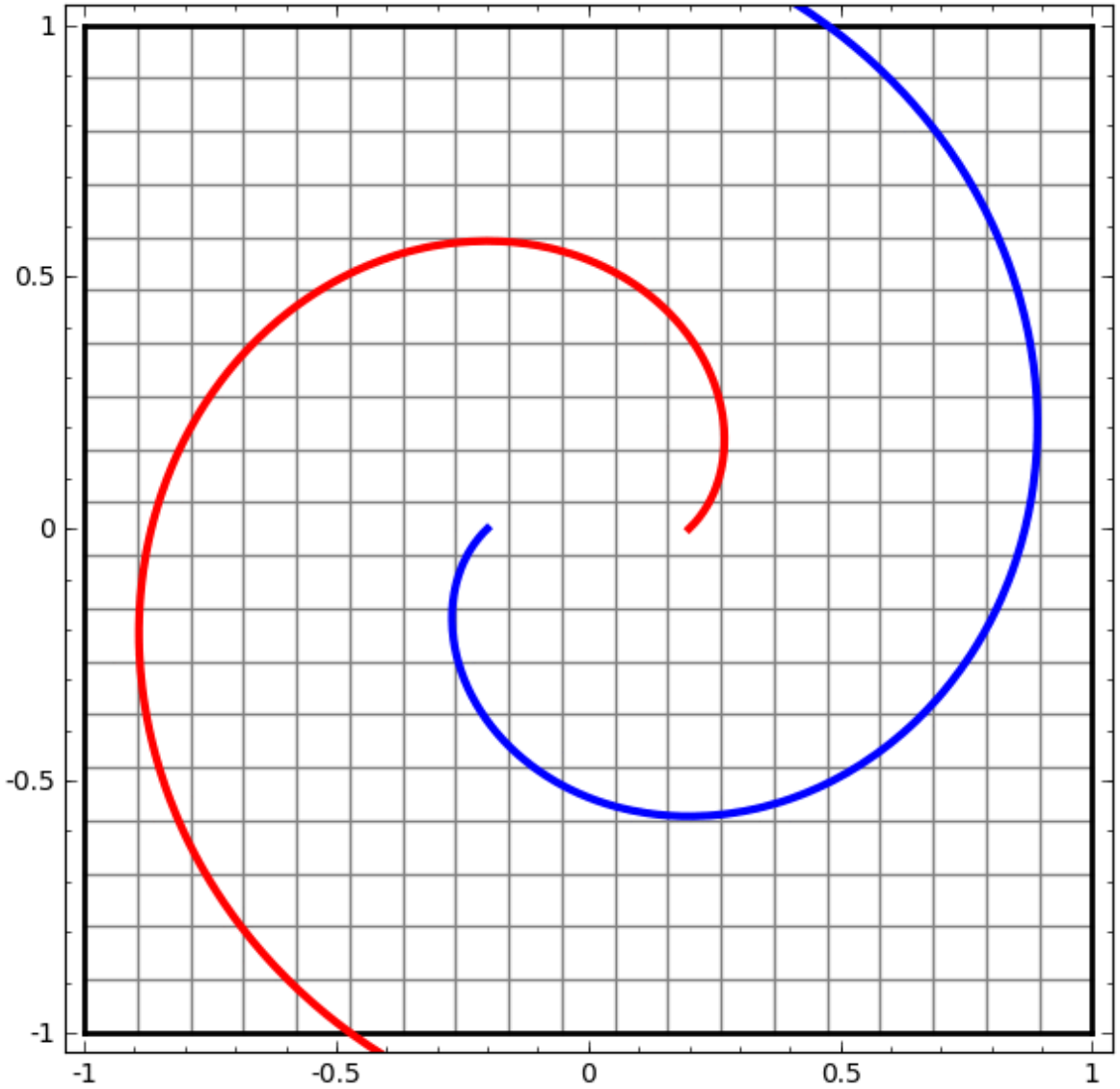
Manifold Hypothesis

- **Manifold Hypothesis** –that the majority of high-dimensional data sets can actually be described by a rather small number of variables, likened to the local coordinate system of the underlying manifold.
- A **manifold** is a topological space that locally resembles Euclidean space near each point.



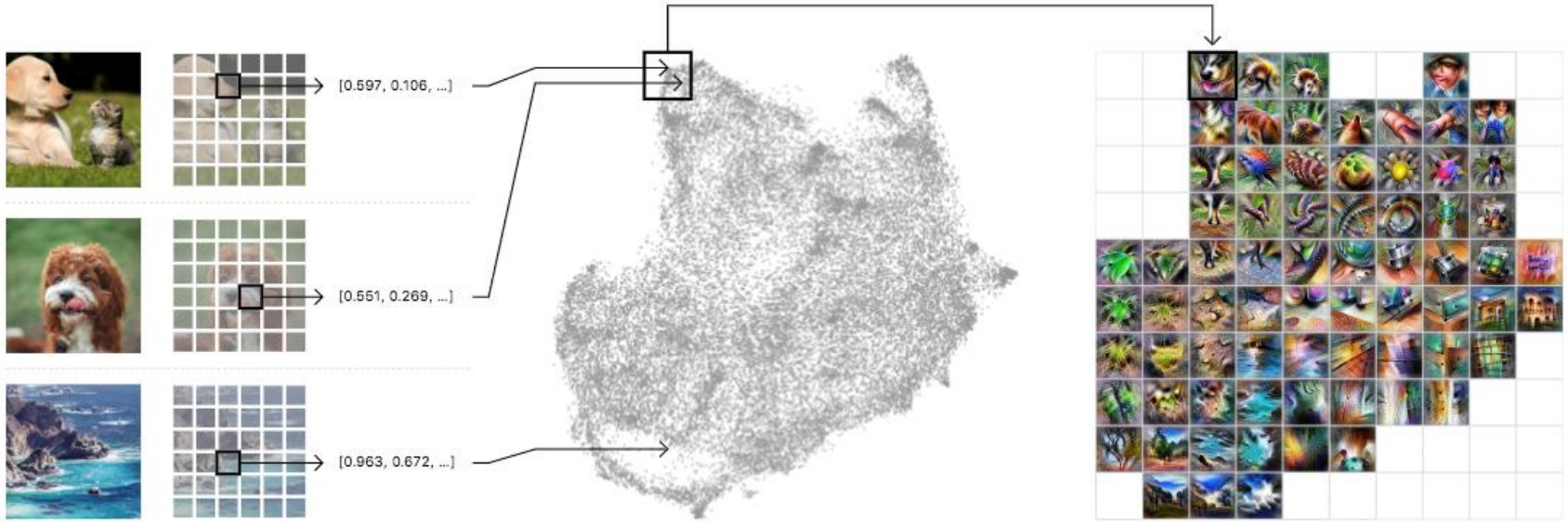
Manifold Hypothesis

https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/



Manifold Hypothesis

- **https://distill.pub/2019/activation-atlas/**



Źródłó: https://research.google/blog/exploring-neural-networks-with-activation-atlases/

Neural Network as an Approximation Tool

- The next two theorems will show us how the depth of the network impacts its properties.

Telgarsky's Theorem

- **Telgarsky M., Representation Benefits of Deep Feedforward Networks**
- Telgarsky uses a simple example to show how much we can benefit from the carefully selected architecture of a deep neural network.

Telgarsky's Theorem

- Function:

$$m(x) = \begin{cases} 2x \text{ for } x \in [0, 0.5] \\ 2(1-x) \text{ for } x \in [0.5, 1] \end{cases}$$

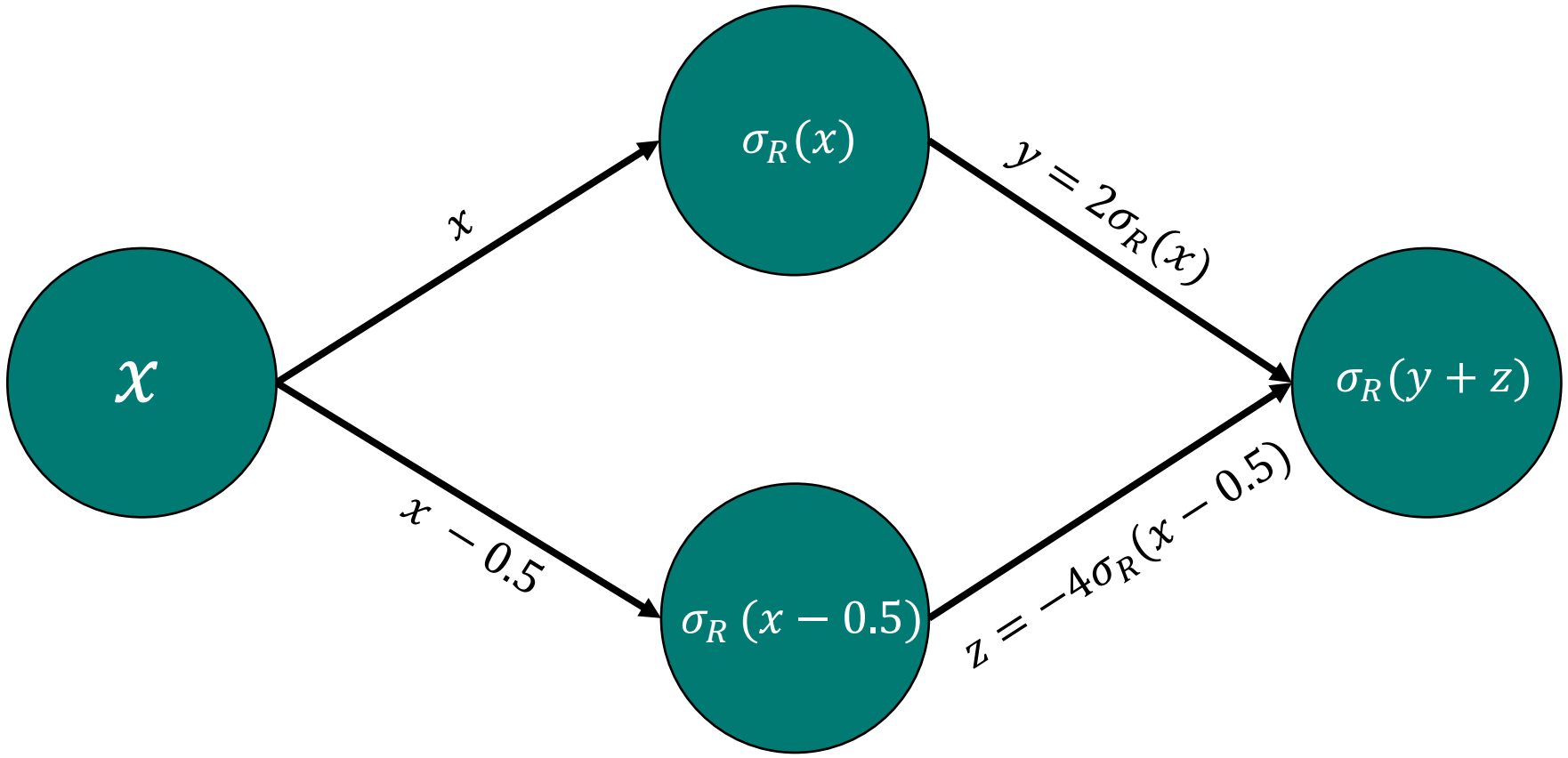
can be approximated with 100% accuracy by neural network of the form:

$$f(x) = \sigma_R(2\sigma_R(x) - 4\sigma_R(x - 0.5))$$

where σ_R is a ReLU function: $\sigma_R(x) = \max(0, x)$

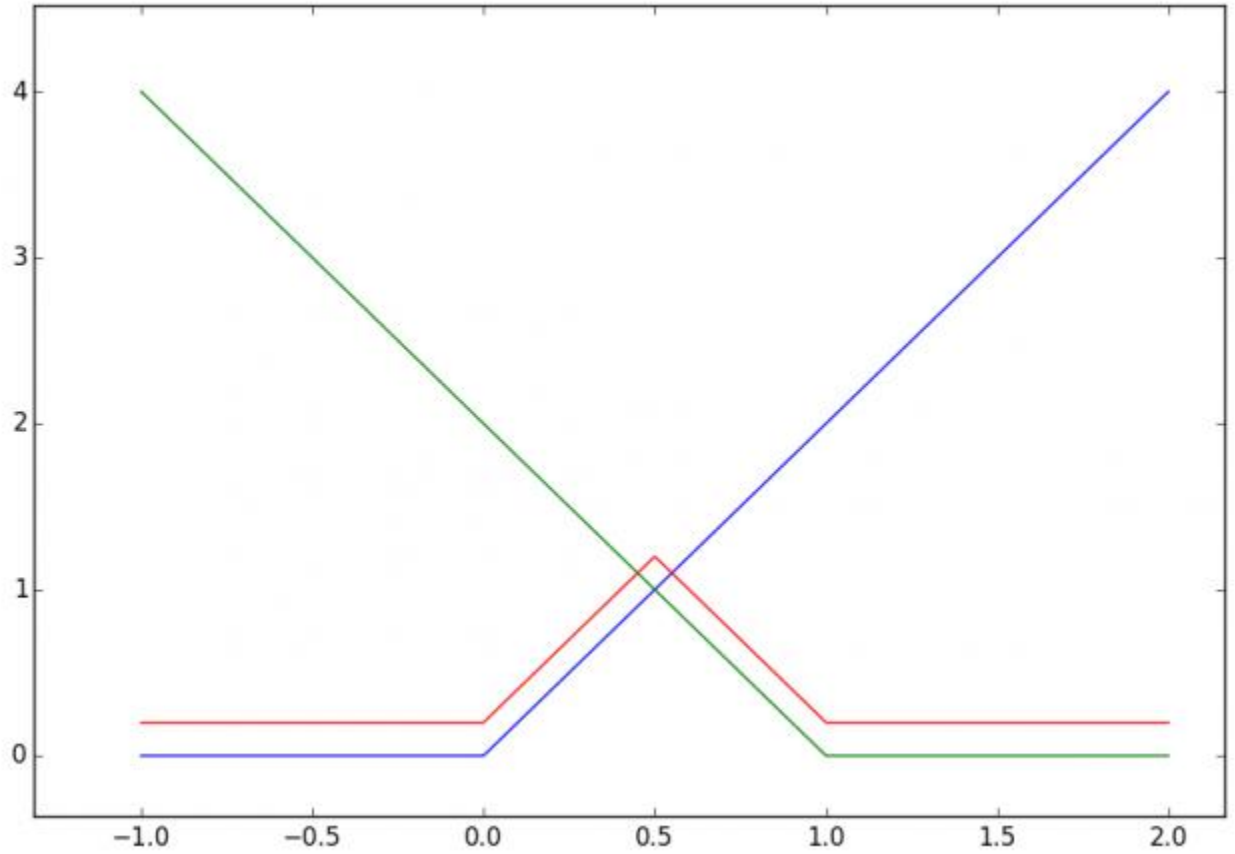
Telgarsky's Theorem

- computational graph:



Telgarsky's Theorem

- and activation functions:



Source: http://elmos.scripts.mit.edu/mathofdeeplearning/2017/04/09/mathematics-of-deep-learning-lecture-2/

Telgarsky's Theorem

- Every composition $m\left(\ldots\left(m(x)\right)\right) = m^{(k)}(x)$ for $k = 1,2,\ldots$ has twice as more „sawtooths" with the same height and half of the previous width.
- Function $m^{(k)}(x)$ has precisely 2^k linear pieces.
- This function can be approximated with 100% precision by a composition of previously presented neural network $f(x)$ with exactly $3k + 1$ neurons.

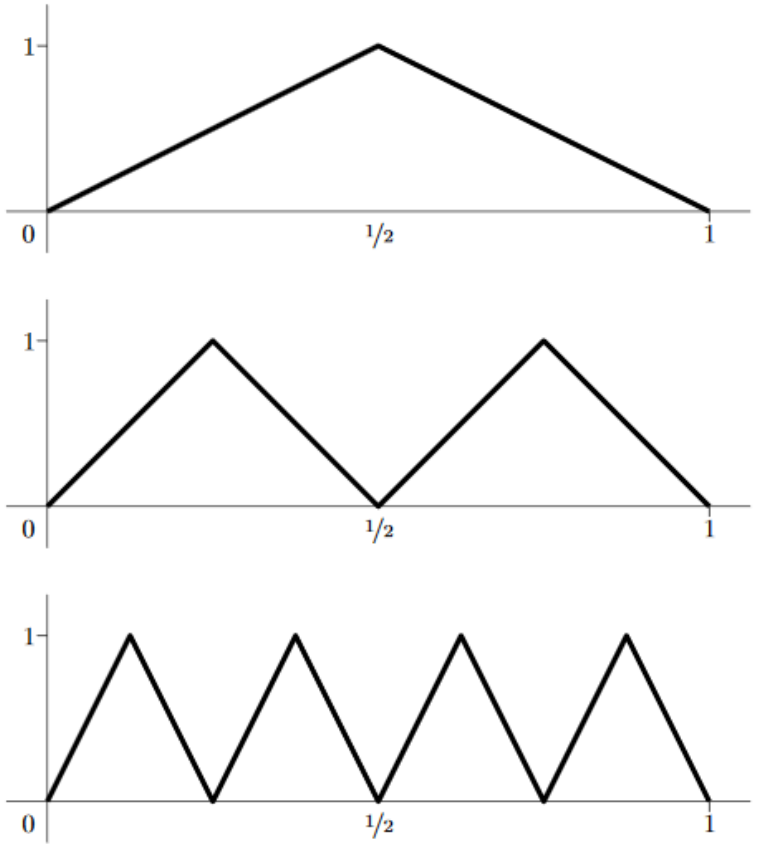


Figure 2: f_{m}, f_{m}^2, and f_{m}^3.

Telgarsky's Theorem

Theorem:

Let $x_i = i/2^k$, $y_i = m^{(k)}(x_i)$ and $y_i \in \{0,1\}$. For a given function F and data set (x_i, y_i) classification error is defined as:

$$R(F) = \frac{1}{n}\sum_i \mathbf{1}\{\tilde{F}(x) \neq y_i\},$$

where $\tilde{F}(x) = \mathbf{1}\{F(x_i) \geq 1/2\}$ is a classifier function. For a given neural network $f(x)$ with 2^k hidden layers of 2 nodes, classification error is always equal to 0:

$$R(f) = 0$$

For every neural network $g(x)$ with l layers of width at most $w < 2^{(n-k)/l - 1}$ a lower bound for error function is equal to:

$$R(g) > \frac{1}{2} - \frac{1}{3 * 2^{k-1}}$$

Montufar's Theorem

- **Montúfar G.F., Pascanu R., Cho K., and Bengio, Y., On the number of linear regions of deep neural networks, in: NIPS'2014, 2014**
- This theorem presents a more general case; a neural network with a piecewise linear activation function (e.g. ReLU) might approximate every function with the precision growing exponentially with the depth of a model ℓ.

Montufar's Theorem

- The reason is straightforward; a deep neural network can identify symmetrical pieces of a function and map them to the same neurons on the next layer.

Definition 1. A map F *identifies* two neighborhoods S and T of its input domain if it maps them to a common subset $F(S) = F(T)$ of its output domain. In this case we also say that S and T are *identified* by F.

Montufar's Theorem

- Intuitively, it might be compared to the folding of a function drawn on a piece of paper by its symmetry axes:

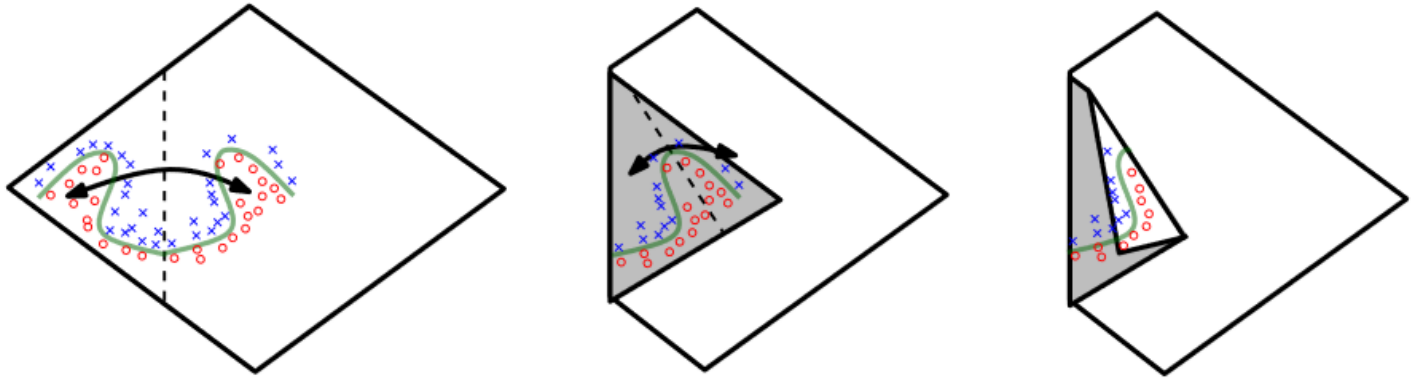


Figure 3: Space folding of 2-D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

Montufar's Theorem

- Deep neural network with the input dimension d, depth ℓ and width n can approximate the following number of linear regions:

$$\mathrm{O}\left(\binom{n}{d}^{d(\ell-1)} n^{d}\right)$$

Montufar's Theorem

- As a result, the quality of the approximation is growing with the depth of the network.
- The model with a single hidden layer utilizes neurons to identify all the pieces of function, even if they are symmetric, thus it requires a significantly larger number of neurons in this layer.

Montufar's Theorem

- Deep neural networks map symmetrical pieces of function into the same neurons on the next layer. As a result, each layer approximates the smaller piece of the function with greater precision.
- However, the benefits from the deep representation depend on the symmetry of the approximated function - the more axes of symmetry (global or local) the function has, the bigger the gain from additional depth will be.

Extra Homework

- Show that the backpropagation algorithm is correctly deriving the error on every neuron in the network (**5 points**).