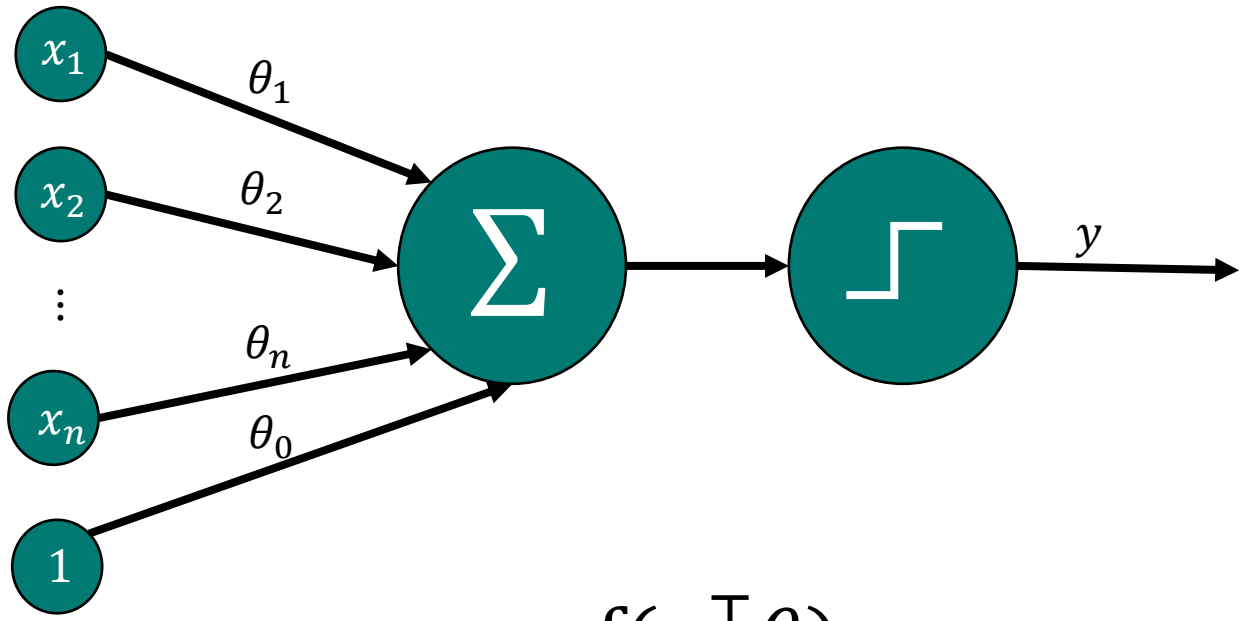
Mathematical Foundations of Deep Learning

Introduction

- Perceptron (Rosenblatt 1957):

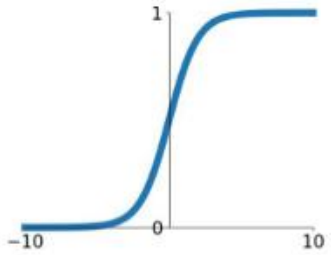


$$y = \mathrm{f}(x^{\top}\theta)$$

Activation Functions

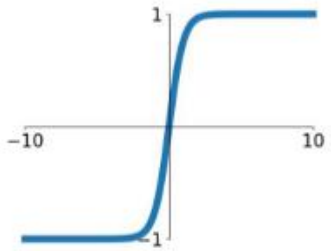
Sigmoid

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



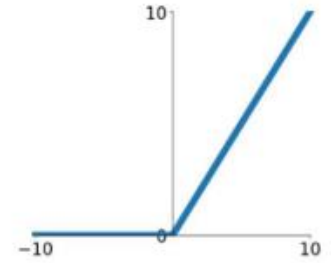
tanh

$$\tanh(x)$$



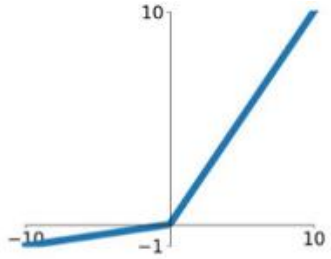
ReLU

$$\max(0, x)$$



Leaky ReLU

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

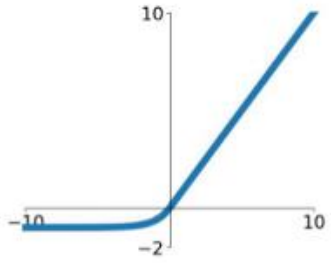


Maxout

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

ELU

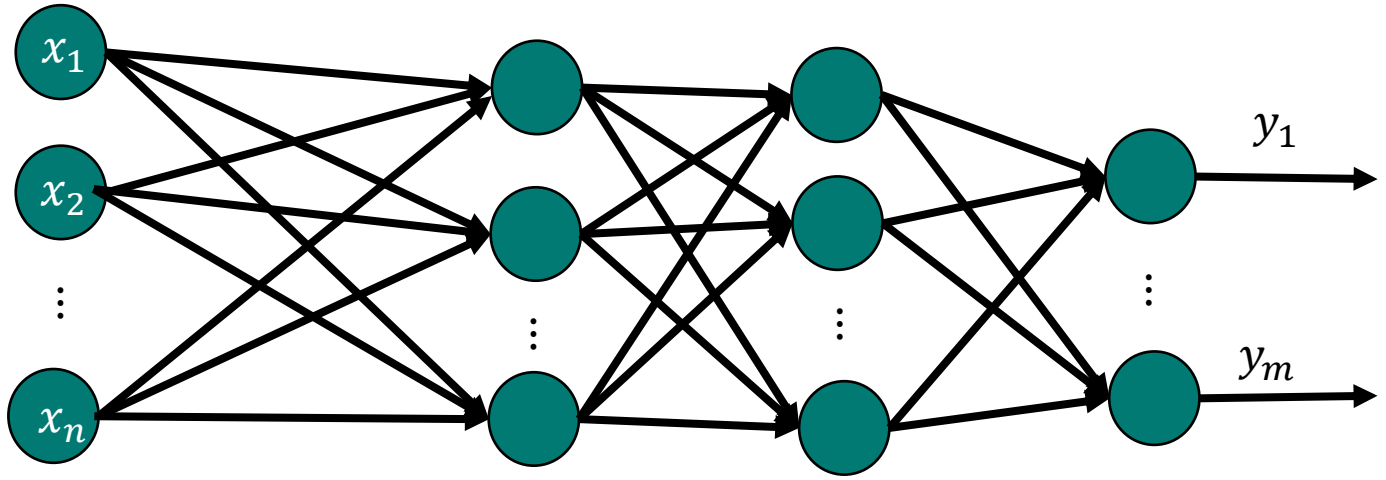
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



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

Introduction

- Networks with hidden layers are obviously more interesting than the perceptrons:

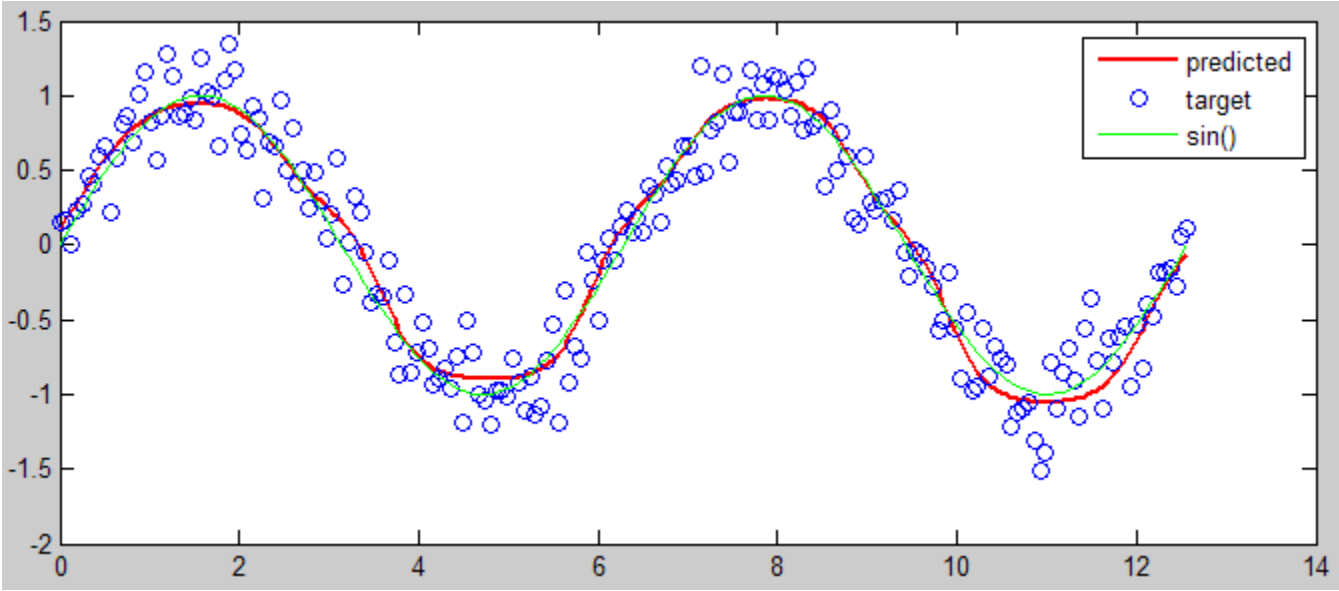


Introduction

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

Introduction

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



Źródło: 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 the approximation $\hat{\phi}$ will be as close as possible to the approximated function ϕ.
- However, we cannot do this directly – we do not know the real form of the function 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.

Optimization

- Cost function $\mathrm{J}(\theta)$ is defined as a 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.

Optimization

- Cost function $\mathrm{J}(\theta)$ is defined as a 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.

Optimization

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

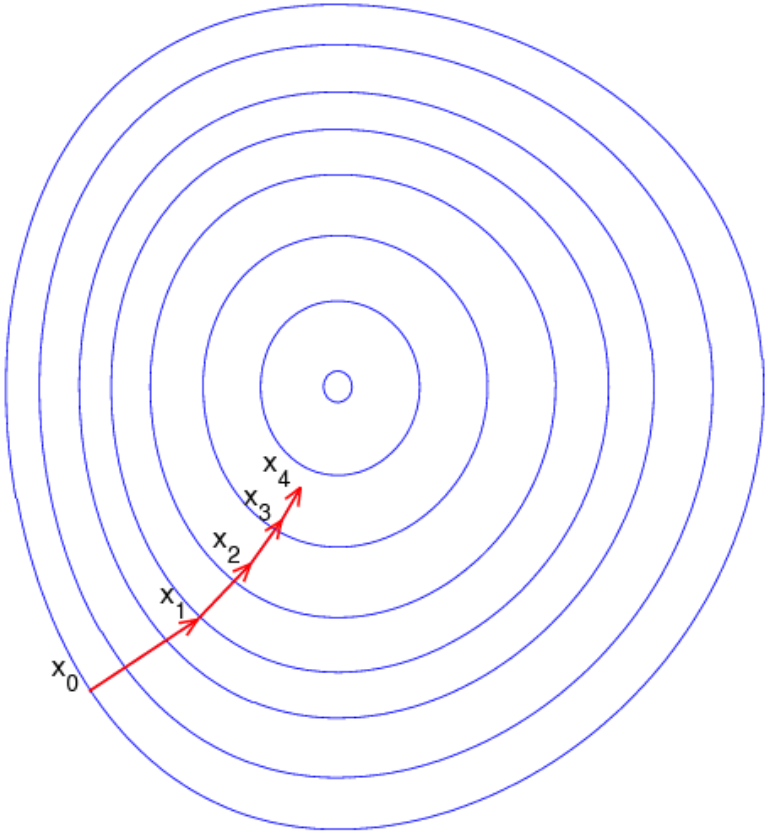
- In most cases we do not want to use a **deterministic** optimization algorithms – they will be slow, inneffective and they will almost surely increase the generalization error.
- **Stochastic algorithms** are much better solution.

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

- In order to train the model efficiently, we must carefully choose a proper values of a **learning rate η**.
- Usually, learning rate will be **decreasing** in the every step of the learning process.

Optimization

- To converge, learning rate must satisfy the following requirements:

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

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

- Commonly, the learning rate is scheduled to decrease linearly with time, up to the iteration $\boldsymbol{\tau}$, after which is constant:

$$\boldsymbol{\eta_i = (1 - \alpha)\eta_0 + \alpha\eta_\tau}$$

where $\boldsymbol{\alpha = \frac{i}{\tau}}$

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 few iterations (epochs).

Hardt's Theorem

- Lets $A(S)$ be an algorithm generating models on a sample $S = (z_1, z_2, \dots, z_n)$ drawn form 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 loss function $\phi(\cdot, y)$ is β-smooth (its gradient $\nabla\phi(\cdot, y)$ is β-Lipschitz function), convex and L-Lipschitz function $\forall\, y$. Suppose that we run stochastic gradient method T times with step sizes $\eta_t \leq 2/\beta$. Then:

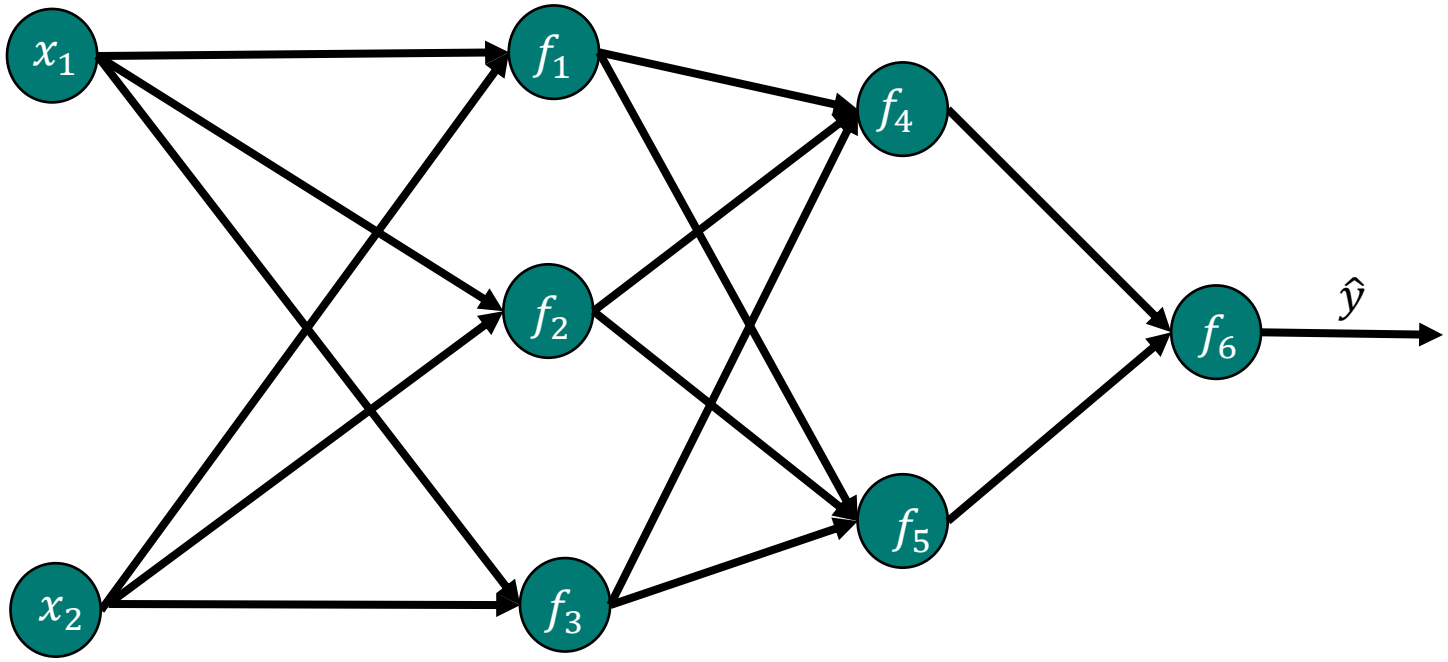
$$\epsilon_s = \frac{2L^2}{n} \sum_{t=1}^{T} \eta_t$$

Backpropagation

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

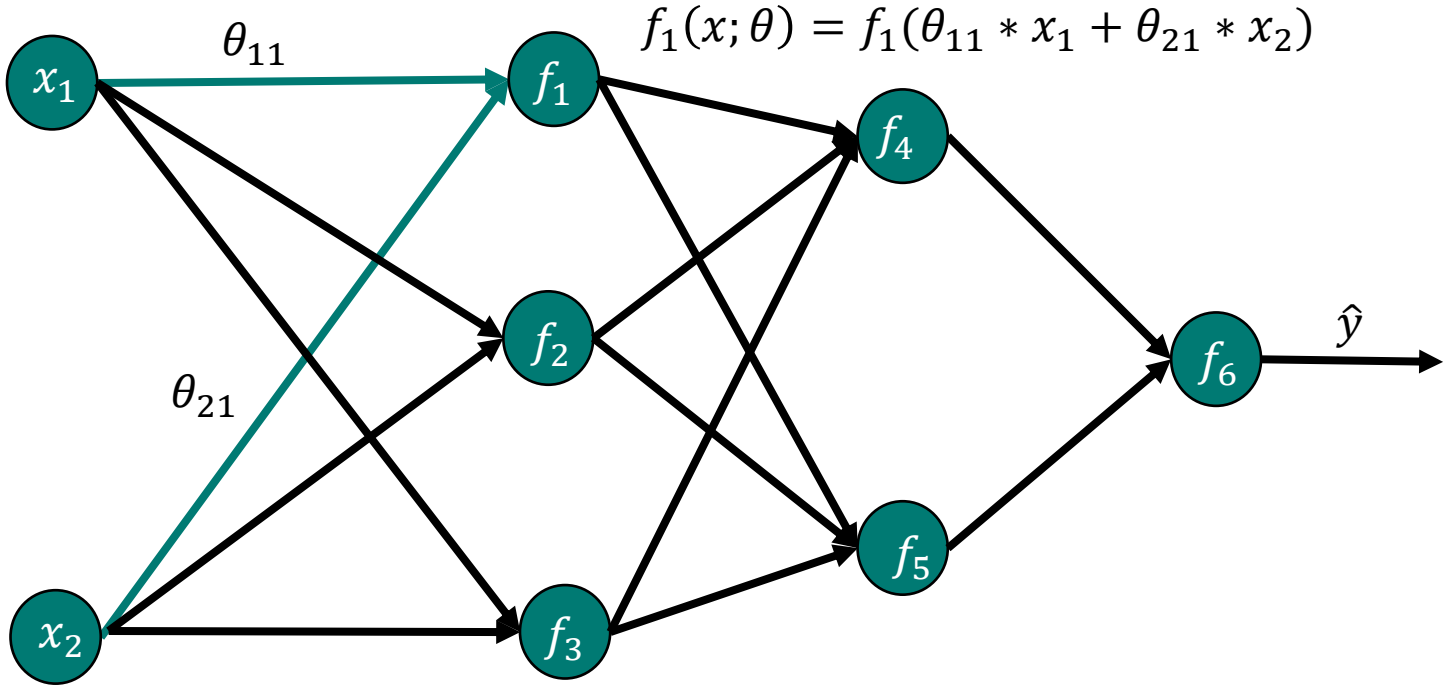
Backpropagation

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



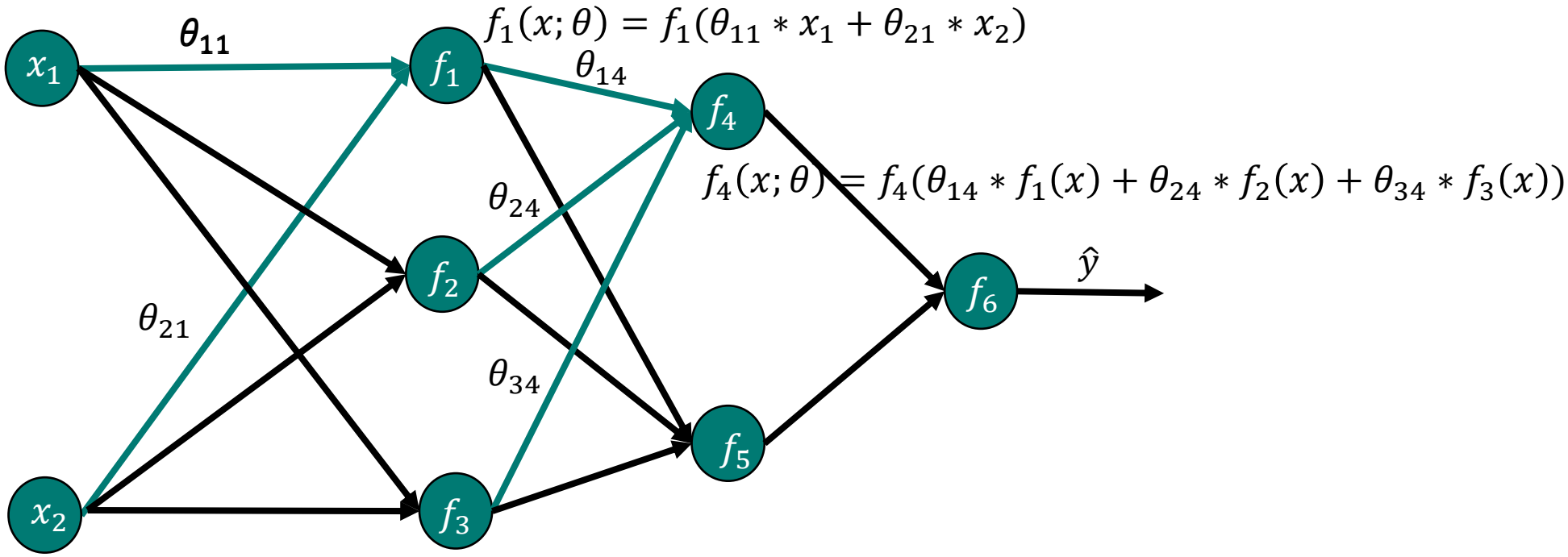
Backpropagation

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



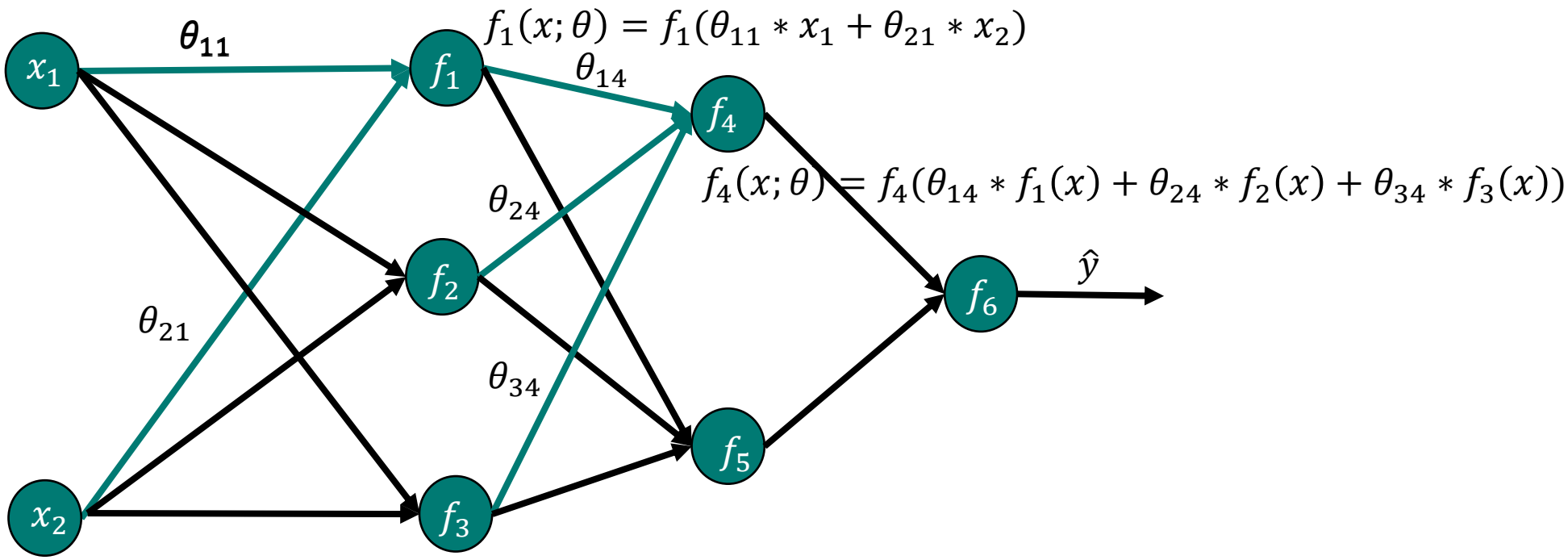
Backpropagation

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



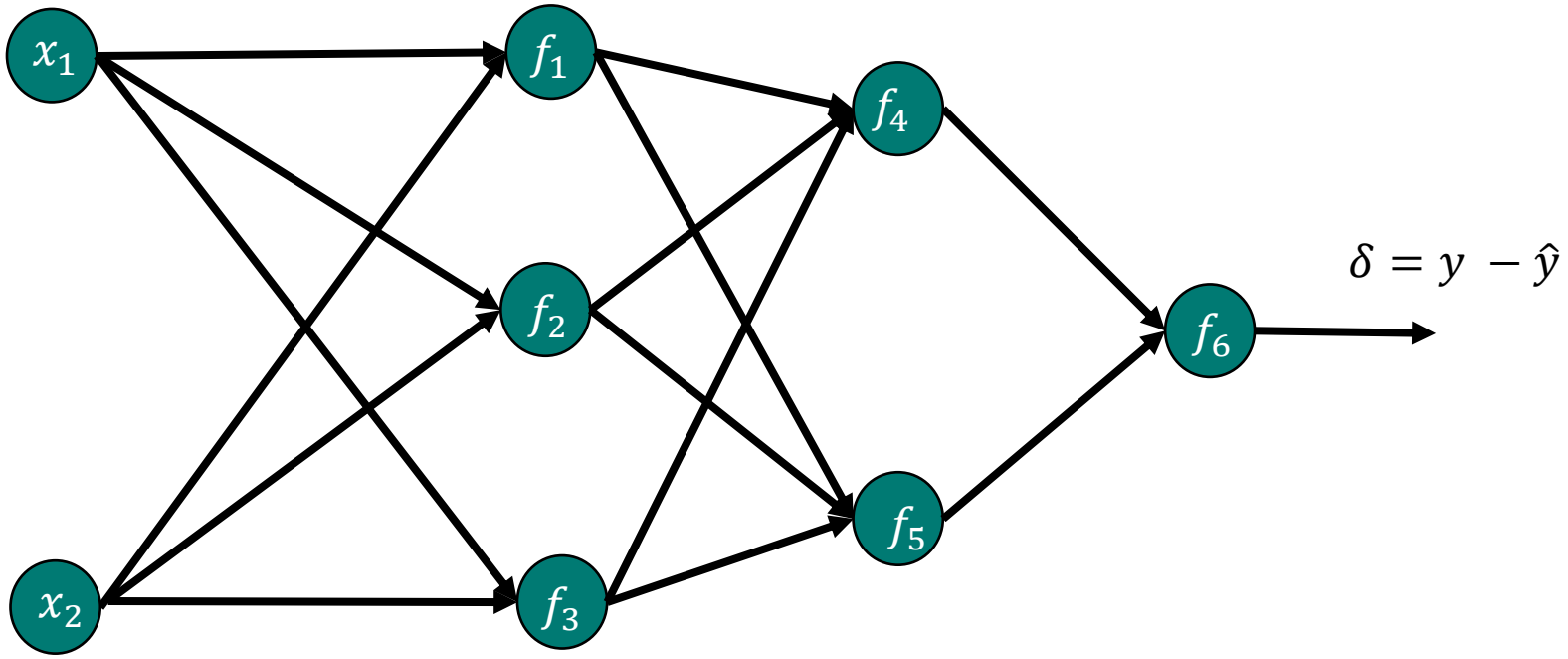
Backpropagation

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



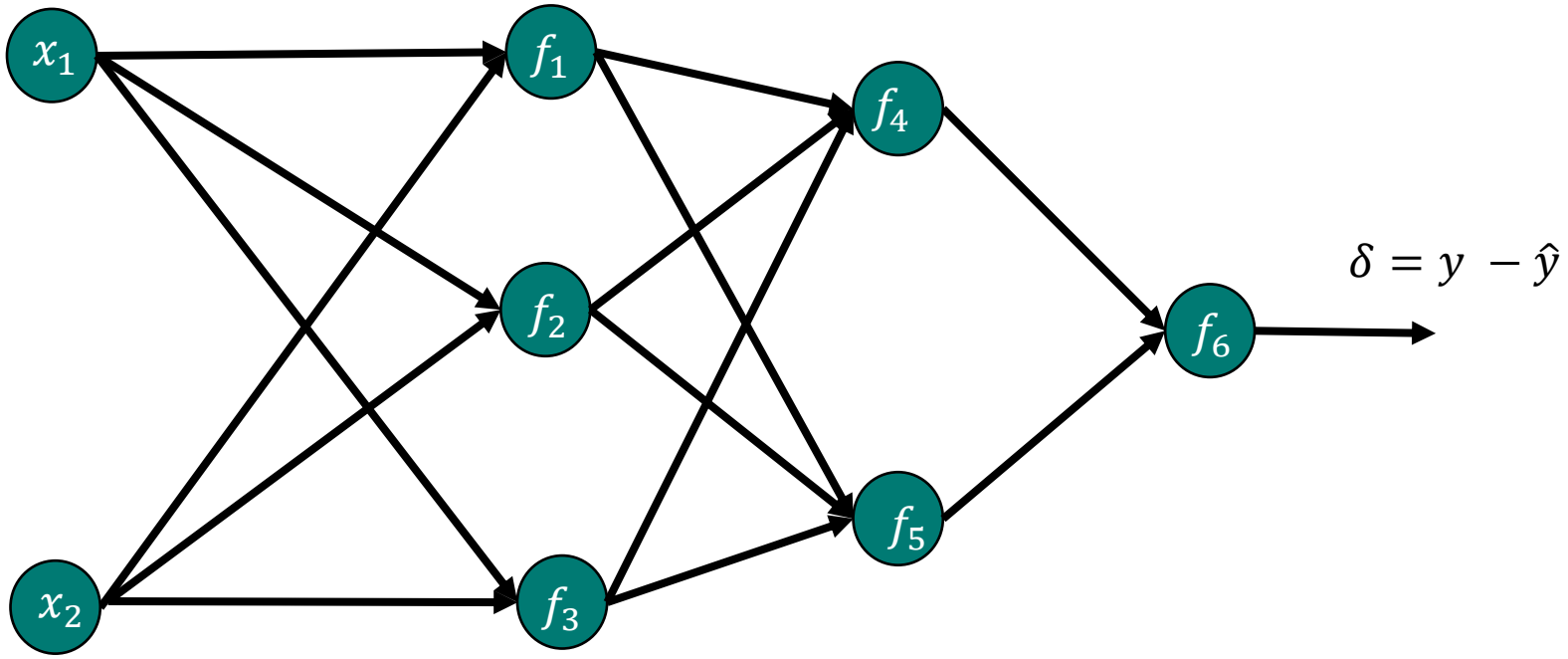
Backpropagation

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



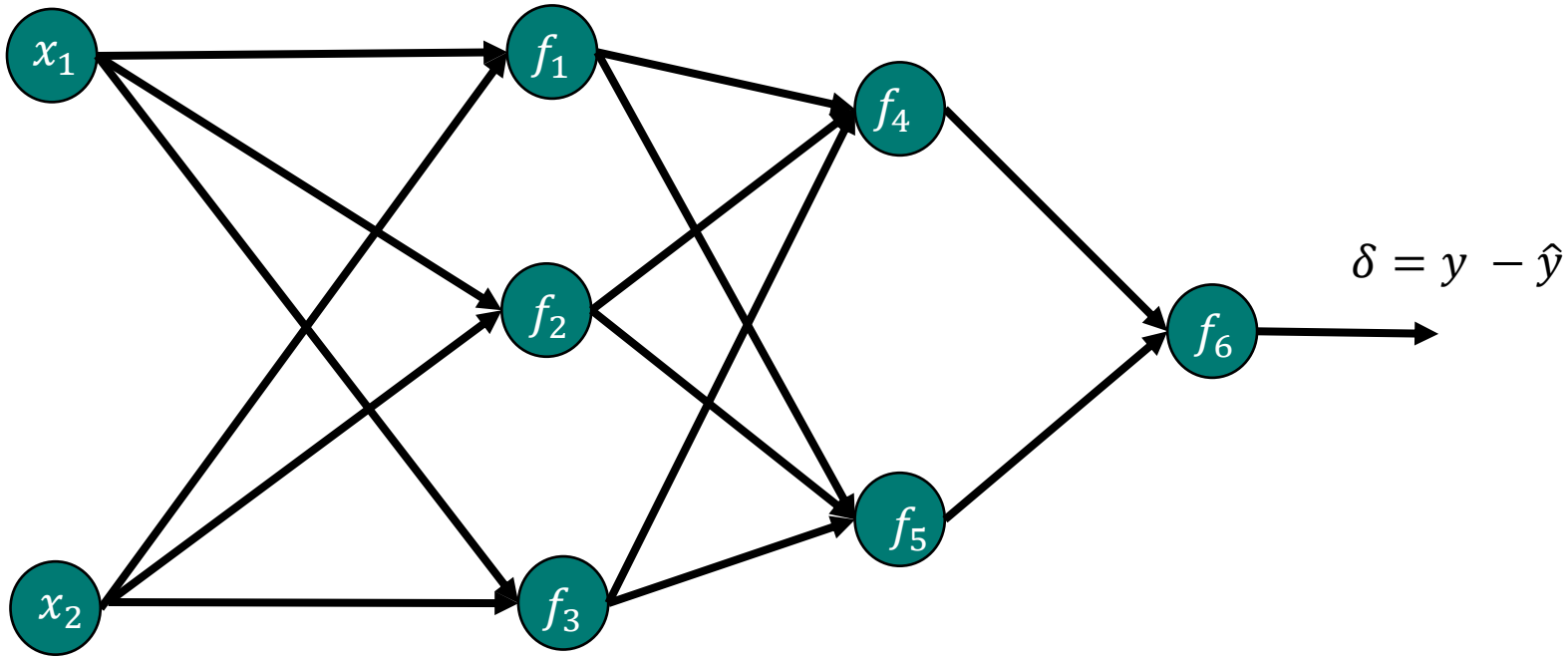
Backpropagation

- But how we could compute the error on the particular neurons?



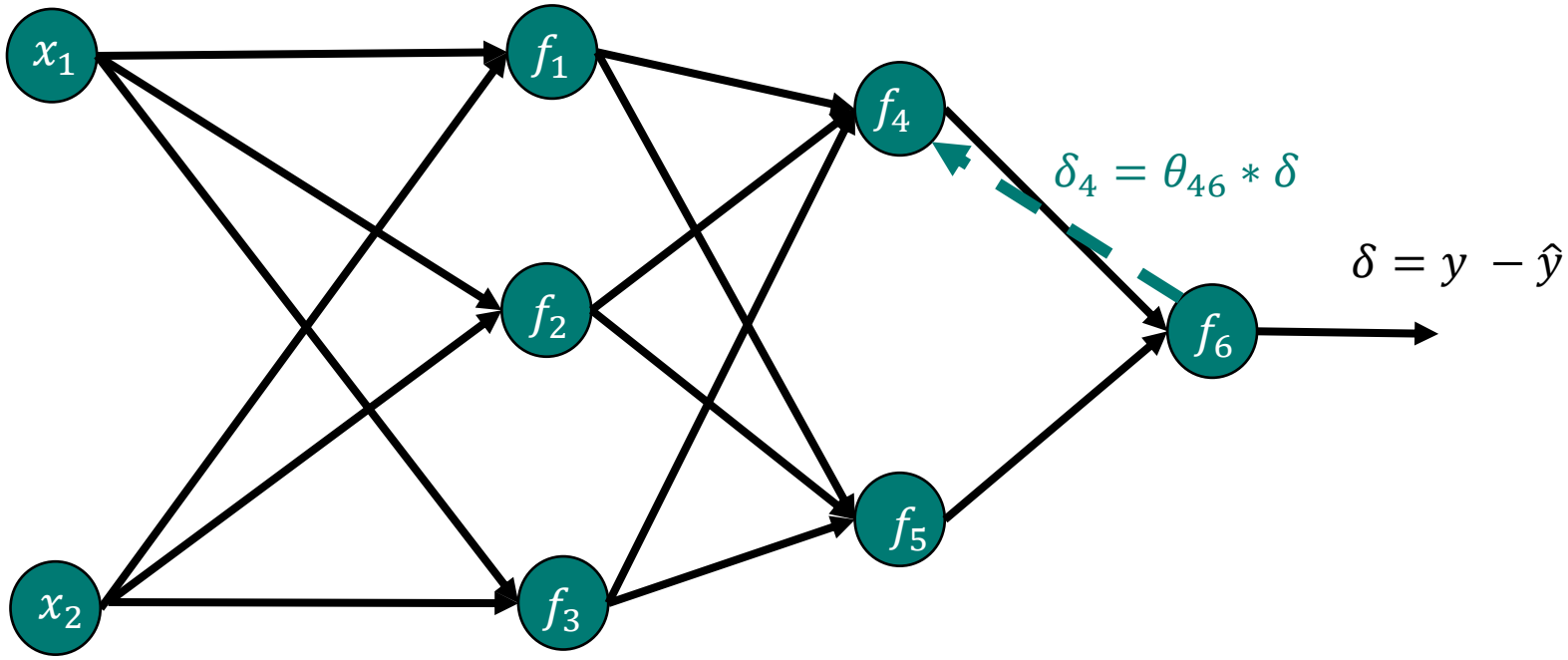
Backpropagation

- In order to do this we must traverse the graph in backward direction.



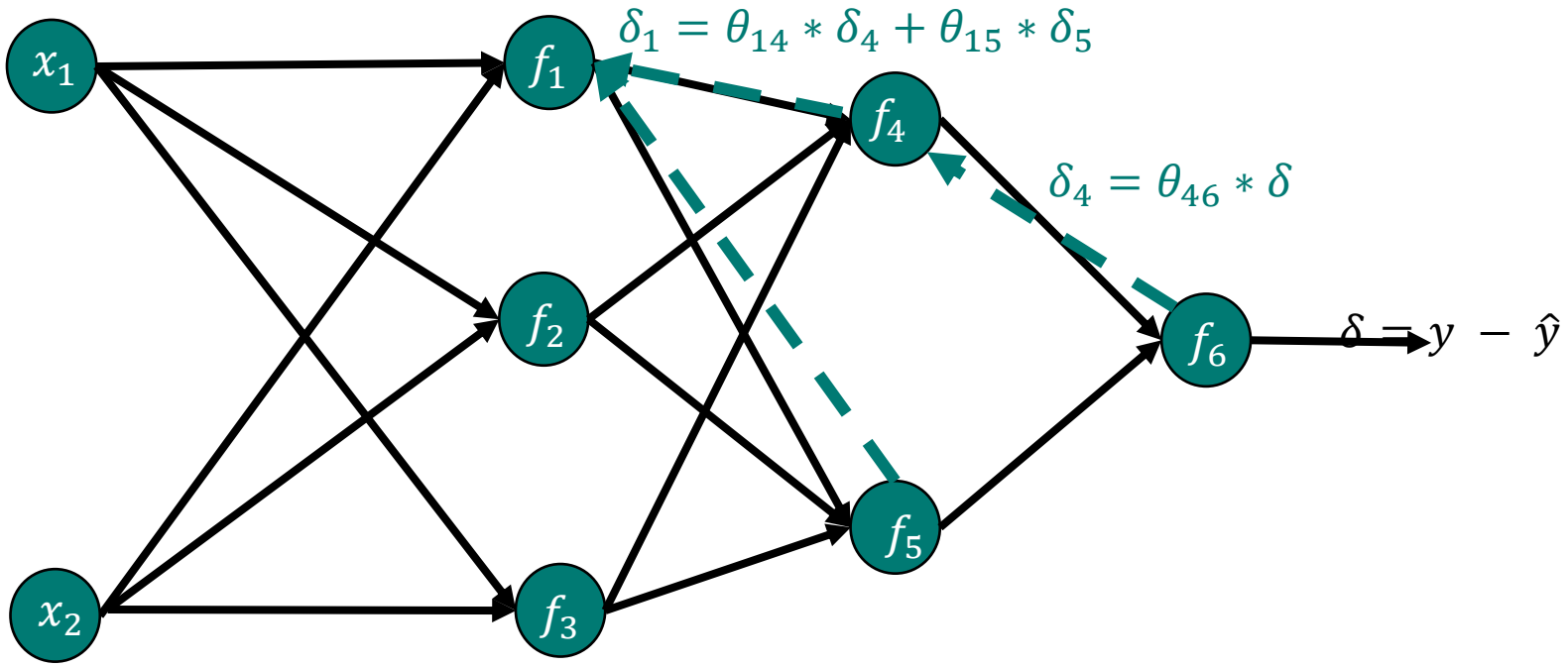
Backpropagation

- In order to do this we must traverse the graph in backward direction.



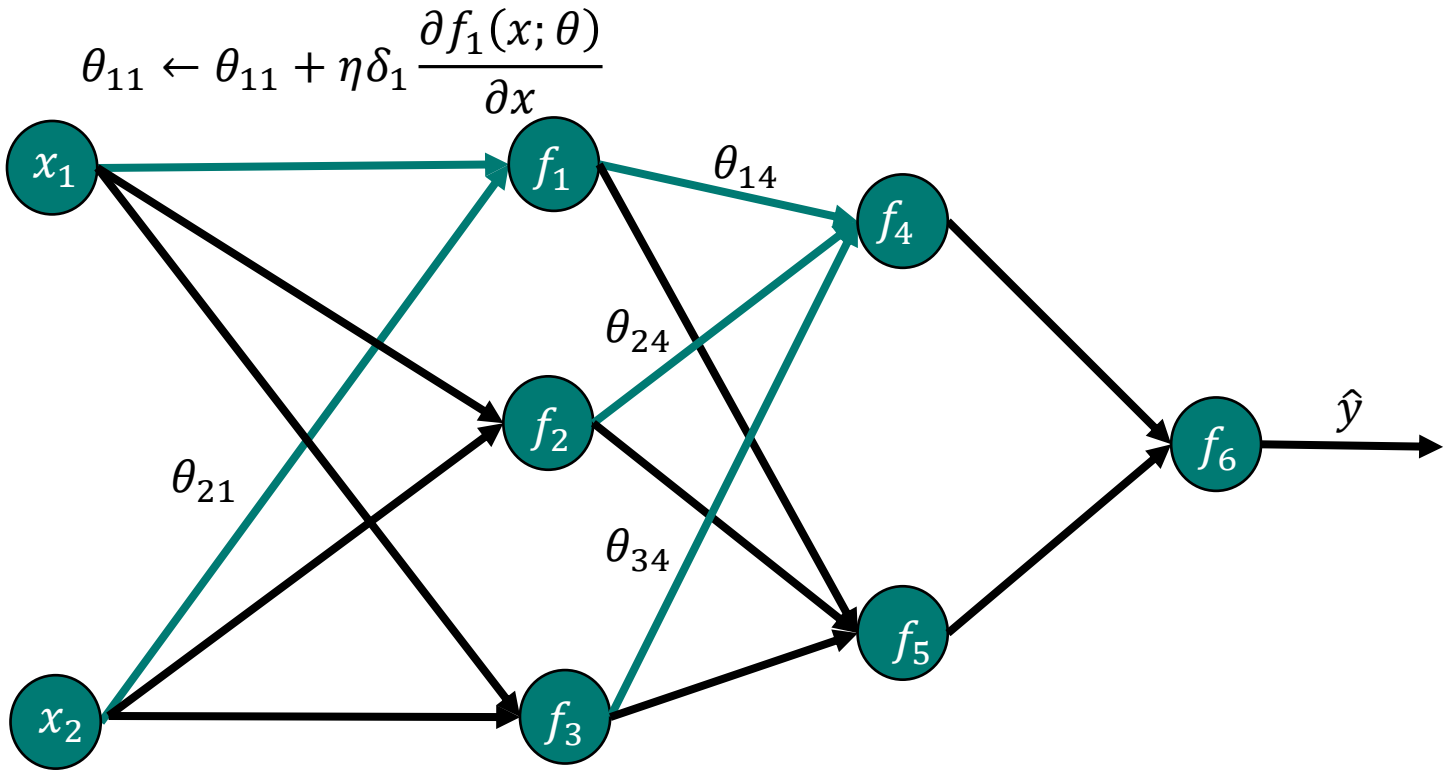
Optymalizacja sieci neuronowej

- In order to do this we must traverse the graph in backward direction.



Backpropagation

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



Neural Network as an Approximation Tool

- Why the neural nets are so efficient comparing to other algorithms?

Neural Network as an Approximation Tool

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

$$\sigma(t) \rightarrow \begin{cases} 1\ gdy\ \ x \rightarrow\ +\infty \\ 0\ gdy\ x \rightarrow -\infty \end{cases}$$

  is able to 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 $\alpha_i, b_i \in \mathbb{R}$ are constant.
- This will hold if the size of the hidden layer n is large enough.

Universal Approximation Theorem

- Let's 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 $\boldsymbol{f} \in \mathbf{C}(\boldsymbol{I_n})$
  - $\boldsymbol{M}(\boldsymbol{I_n})$ is a space of finite, signed regular Borel measures on $\boldsymbol{I_n}$

Universal Approximation Theorem

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

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

  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:

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

Neural Network as an Approximation Tool

- Cybenko have proven that the neural nets with a sigmoidal 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 extend the results of Cybenko for wider families of functions, proving that the architecture of the neural network, not the activation functions, is crucial for the approximation capabilities.

Neural Network as an Approximation Tool

- Both results are theoretical; they show the capacity of neural networks, they ability to represent a given function, but without elaborating about the details.
- For example, we still do not now how wide the layer of such 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 $\boldsymbol{f(X)}$ by a linear combination: $\mathbf{G}(\boldsymbol{X}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{a_i \sigma(w_i^T x + b_i)}$
- But this time we are interested in estimating the number of neurons n on the hidden layer.

Barron's Theorem

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

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

- We are considering a class of functions f: $\mathbb{R}^d \longrightarrow \mathbb{R}$ i $f \in \Gamma_C$ which Fourier transform $\hat{f}(\omega)$ satisfies:

$$\int \hat{f}(\omega)|\omega| d\omega \leq C$$

  - 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 $\boldsymbol{\sigma}$, every probability measure μ i $n \geq 1$, there existist linear combination of n sigmoidal functions $G(X) = \sum_{i=1}^{n} a_i \sigma(w_i^T x + b_i)$, such that:

$$\int \big(G(X) - f(x)\big)^2 \, \mu(dx) \leq \frac{(2C)^2}{n}$$

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

1. $\sum_{i=1}^{n} |w_i| \leq 2C$
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 a 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 are able to 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, size of the hidden layer n is equal to the all possible combinations of the input data 2^d.

Neural Network as an Approximation Tool

- The best method to avoid such problems is to use more than one hidden layer.
- In the most cases it allows us to reduce the number of neccessary neurons in the net, reduce the approximation error or both at the same time.
- Next two theorems will show us how it works.

Neural Network as an Approximation Tool

- Deep neural networks effectively exploit the properties of approximated function; because of that deep neural network might be less complex and retain or even exceed the quality of fit.

Telgarsky's Theorem

- **Telgarsky M., Representation Benefits of Deep Feedforward Networks**
- Telgarsky shows on a very simple example how much we can benefit from the properly suited deep representation of function.

Telgarsky's Theorem

- Function:

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

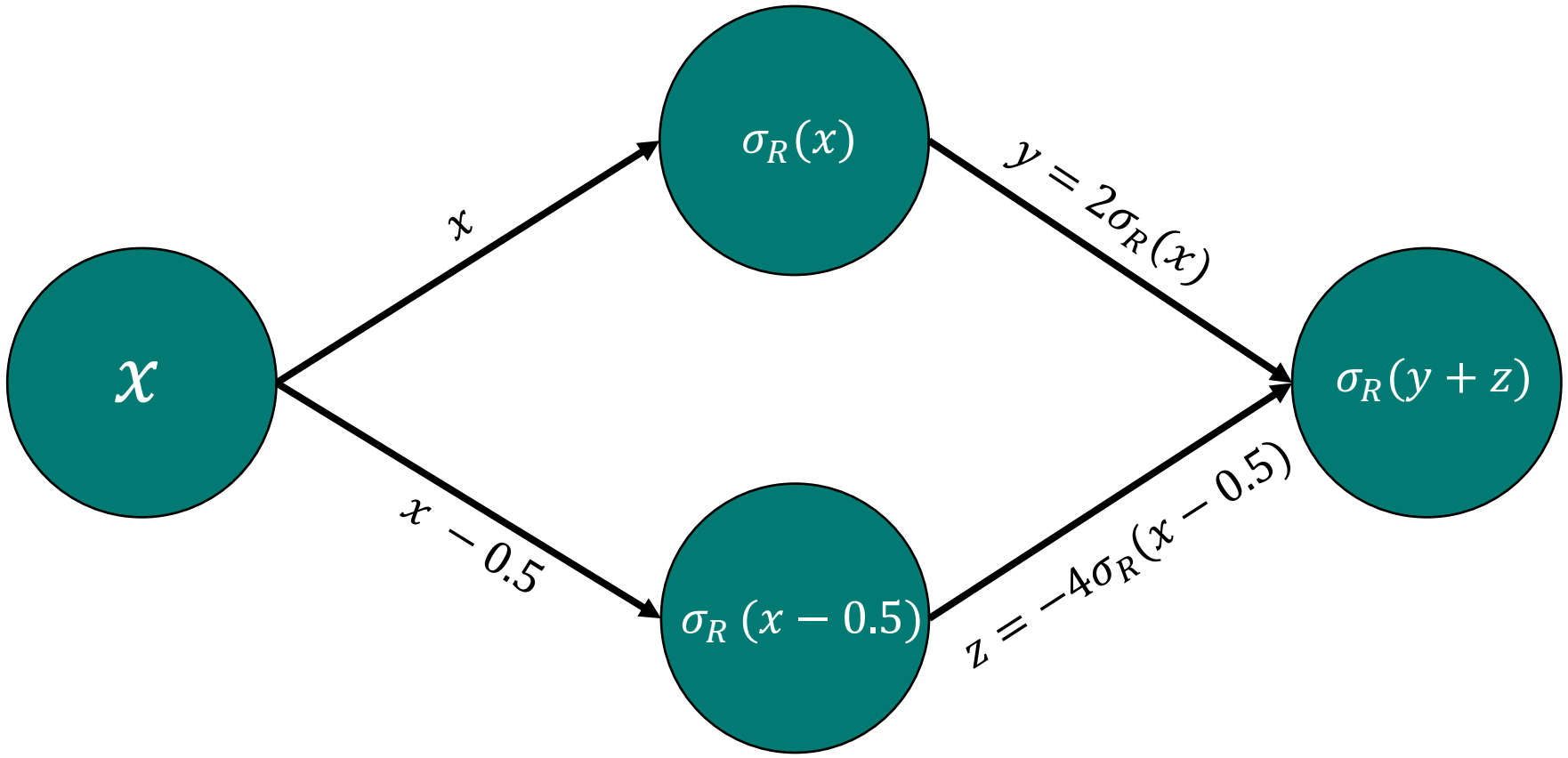
- Could be approximated with 100% accuracy by a neural network in a form:

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

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

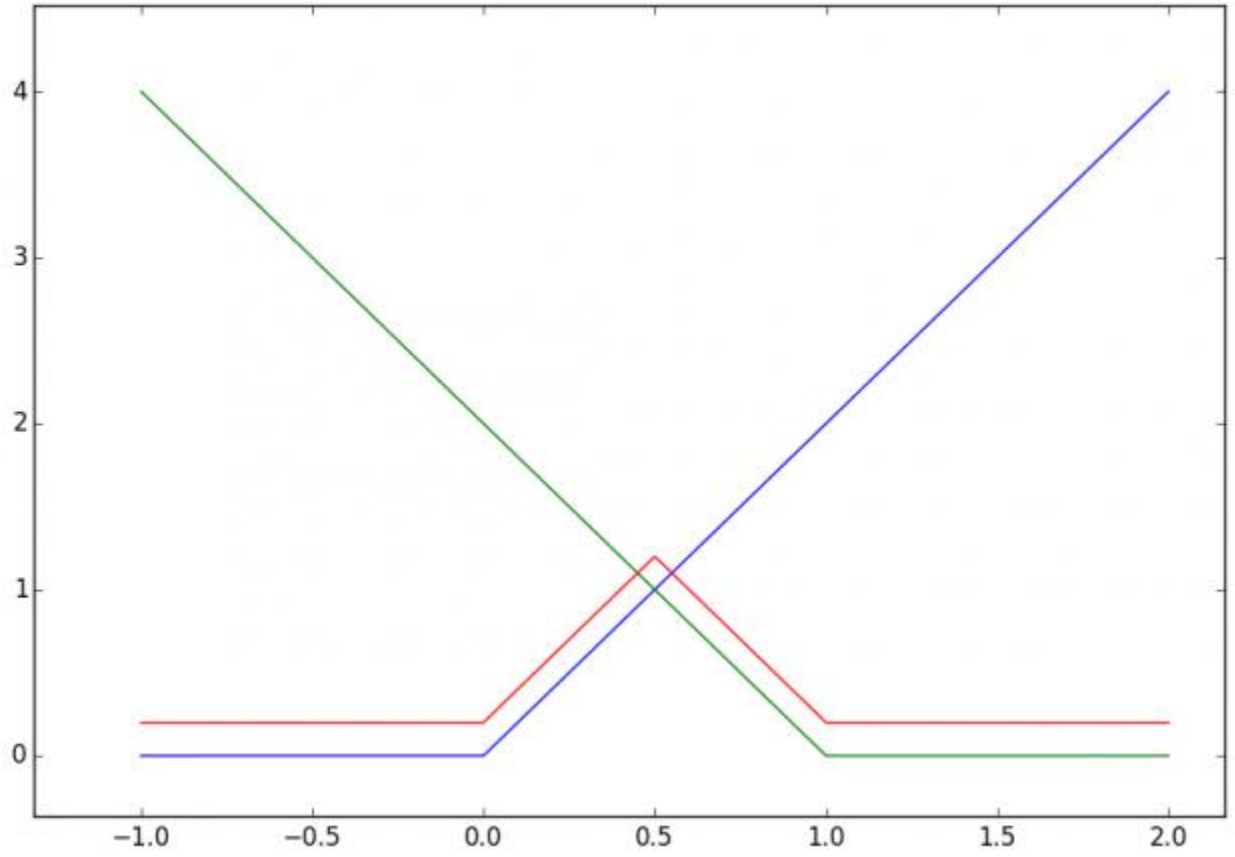
Telgarsky's Theorem

- The neural network has a form:



Telgarsky's Theorem

- And the activation functions looks like that:



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

Telgarsky's Theorem

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

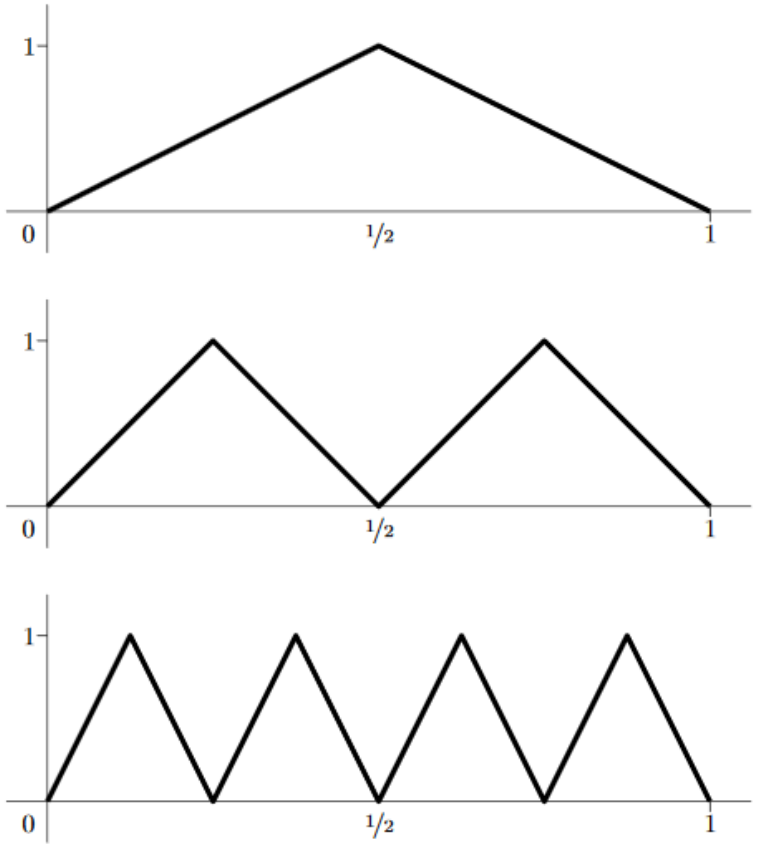


Figure 2: f_{m}, f_{m}^2, and f_{m}^3.

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 $w < 2^{(n-k)/l - 1}$ lower bound of 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 shows how the depth of neural network influence the quality of approximation.
- It turns out that by using the piecewise linear activation function (e.g. ReLU) neural network might approximate every function with the precision growing exponetially with a depth of a model.

Montufar's Theorem

- The reason is simple; deep neural network is able to identify the parts of a function with the same behawior and map it 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:

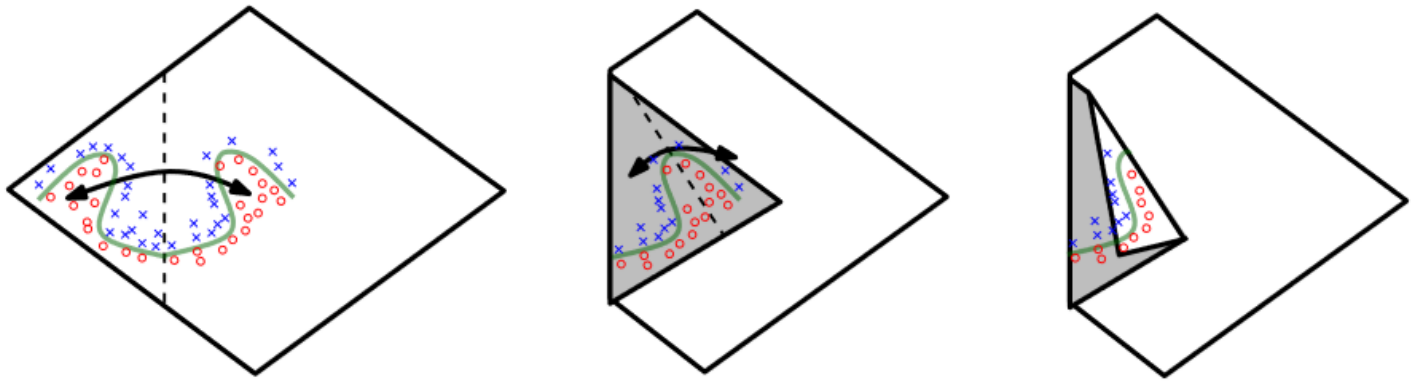


Figure 3: Space folding of 2-D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

Montufar's Theorem

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

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

Montufar's Theorem

- As a result, the quality of approximation is growing with a depth on a network.
- Model with a single, wide hidden layer utilizes neurons to identify all regions of function, even if they are symmetric, thus it requires significantly larger numer of neurons in the layer.

Montufar's Theorem

- Deep networks are mapping similar regions into the same neurons on the output of the layer.
- Because of that, neurons on the next layer could be used to approximate the smaller piece of a function more precisely.
- However, the benefits from the deep representation are strongly depending on the symmetry of the approximated function.
- The more axes of symmetry (global or local) function has, the bigger benefit from deep representation will be.

Extra Homework

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