From Shallow to Deep Neural Networks

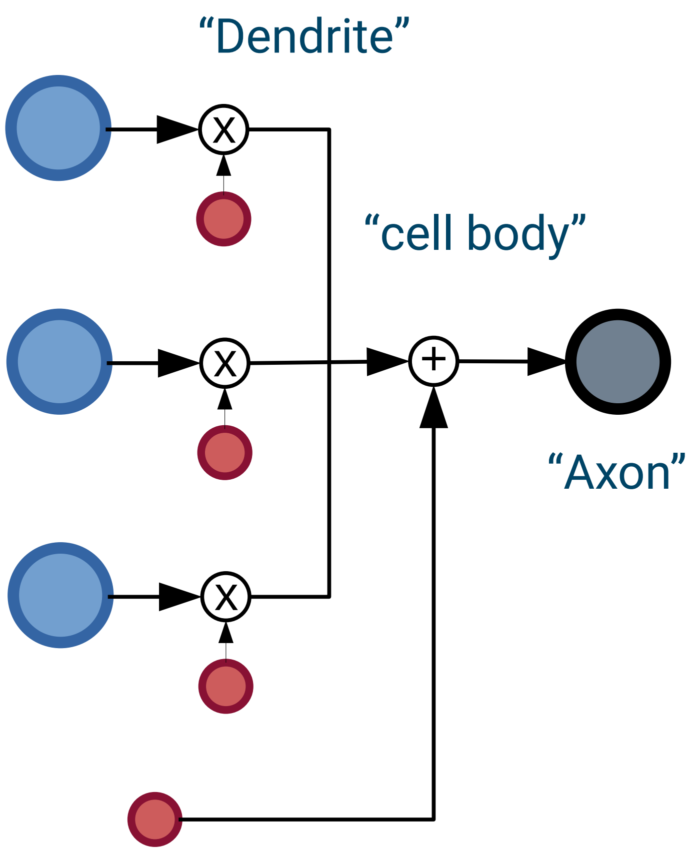
[Building More Complex Models]

José Oramas

University of Antwerp

Previous Session: Artificial Neurons

An artificial counterpart to real neurons



[Rossenblat, 1958]
[Minsky & Papert, 1969]

$$\sum_{i=1}^{d} w_i x_i + b$$

$$\sum_{i=0}^{d} w_i x_i, \quad x_0 := 1$$

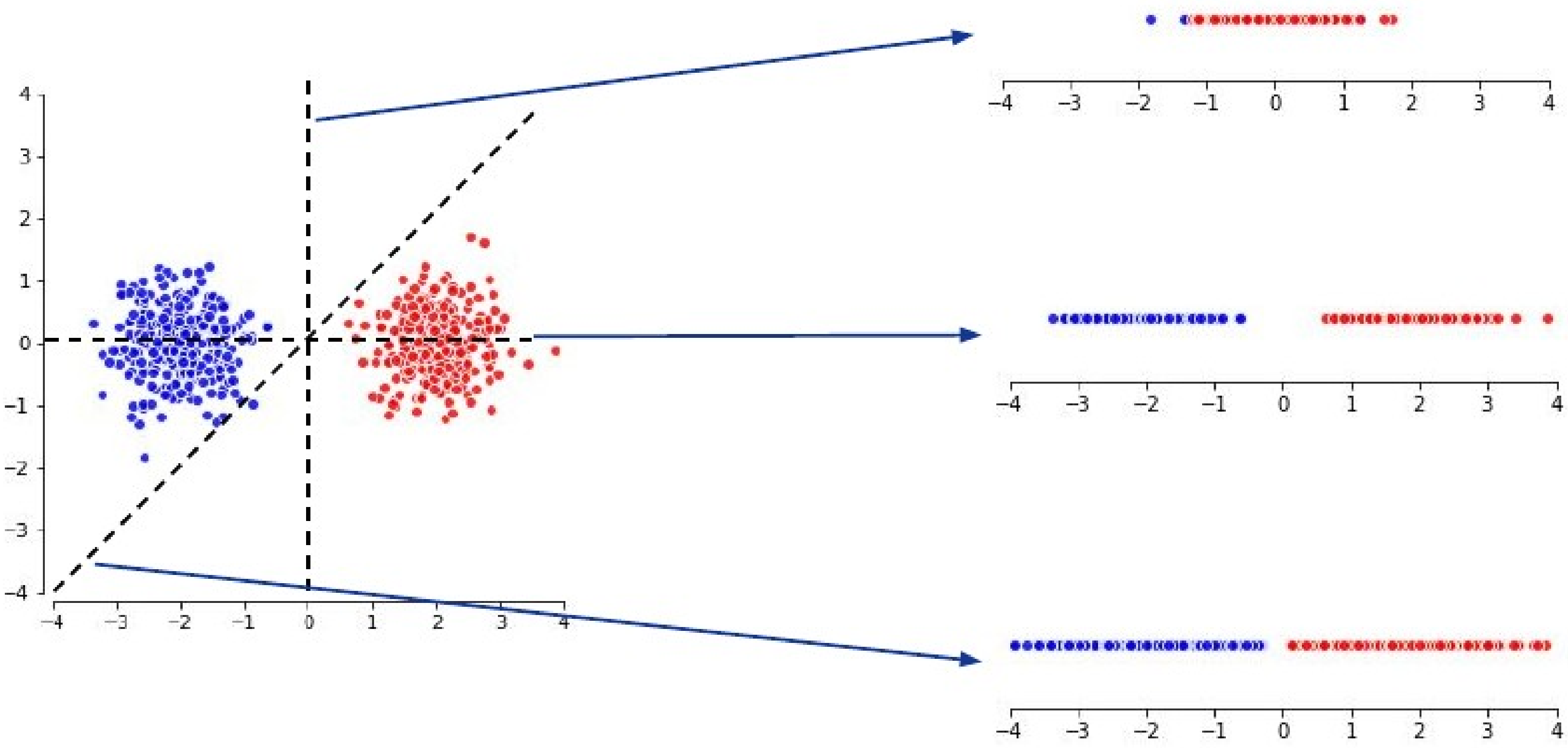
Characteristics:

- Basic computation
- Has inhibition/excitation connections
- Building block
- Time-independent state
- Outputs real values

Previous Session: Artificial Neurons

What they do?

- Define a linear (afine) projection of the data



Oh yes I remember, but ...

How do we use that?

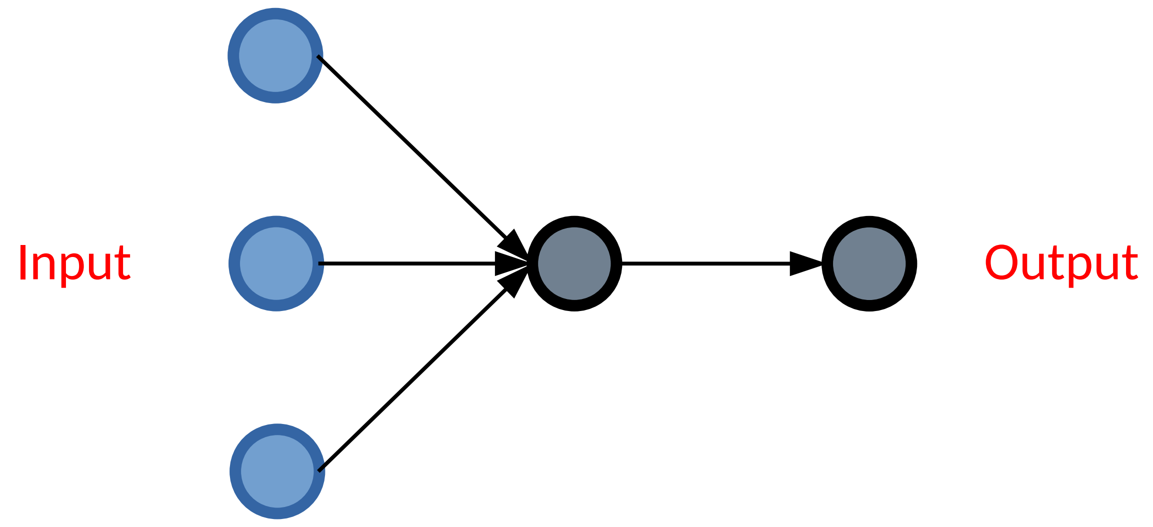
A Shallow Neural Network

[with few layers]

From Neurons to Layers

A Common Composition

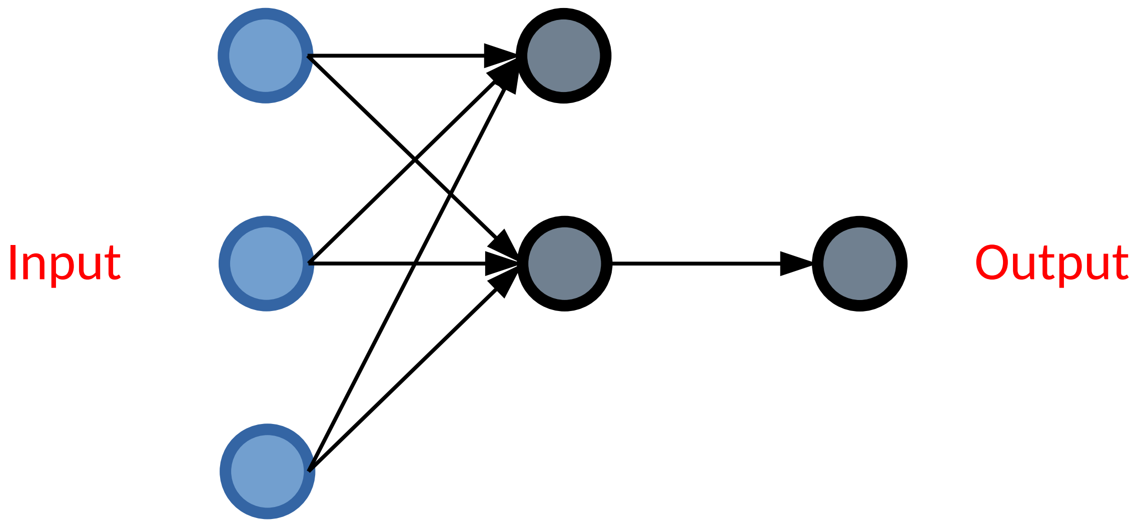
- Add several neurons working on “parallel”.



From Neurons to Layers

A Common Composition

- Add several neurons working on “parallel”.

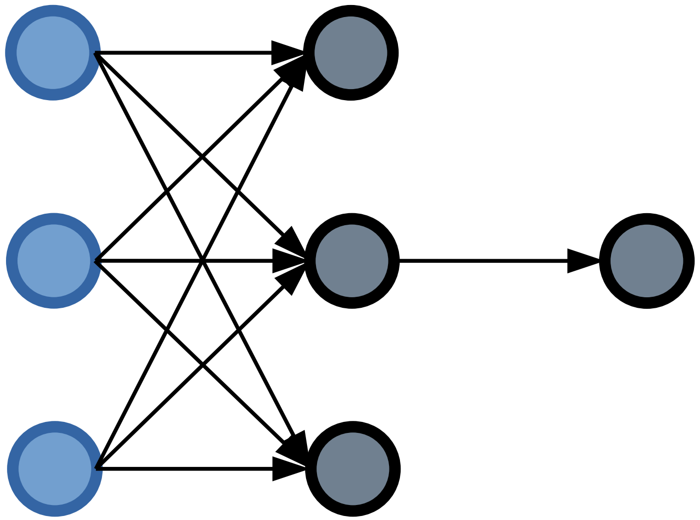


From Neurons to Layers

A Common Composition

- Add several neurons working on “parallel”.

Input

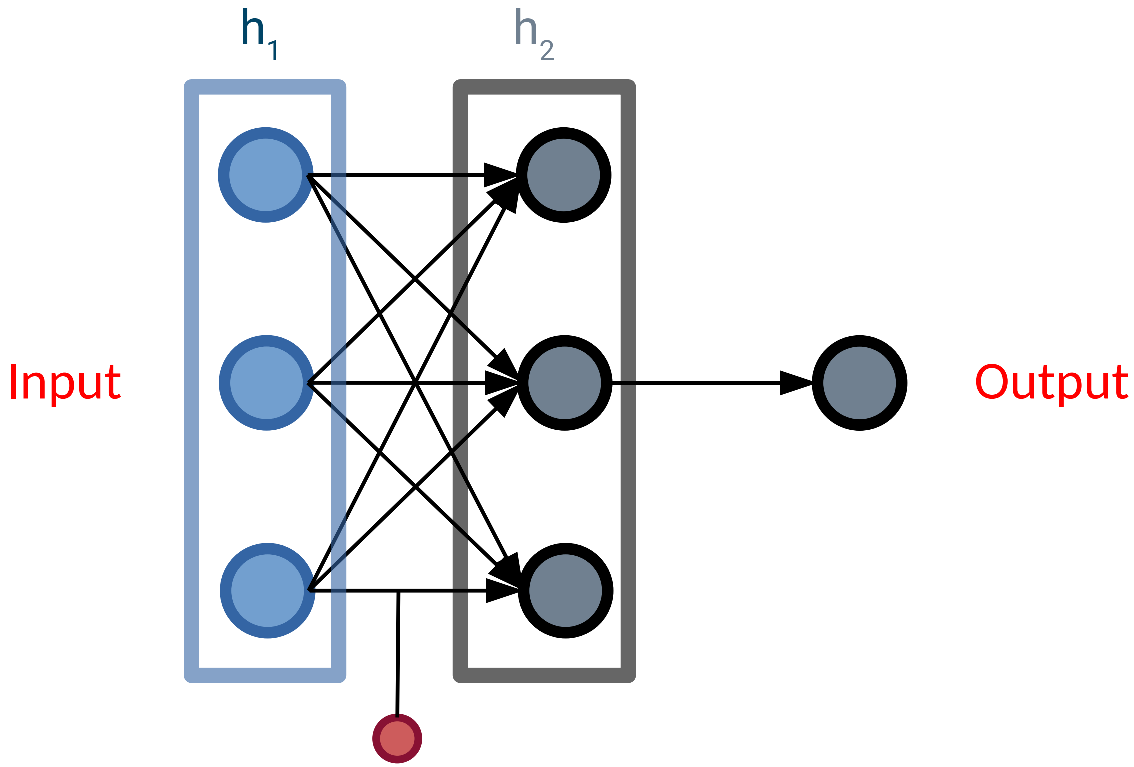


Output

From Neurons to Layers

A Common Composition

- Add several neurons working on “parallel”.

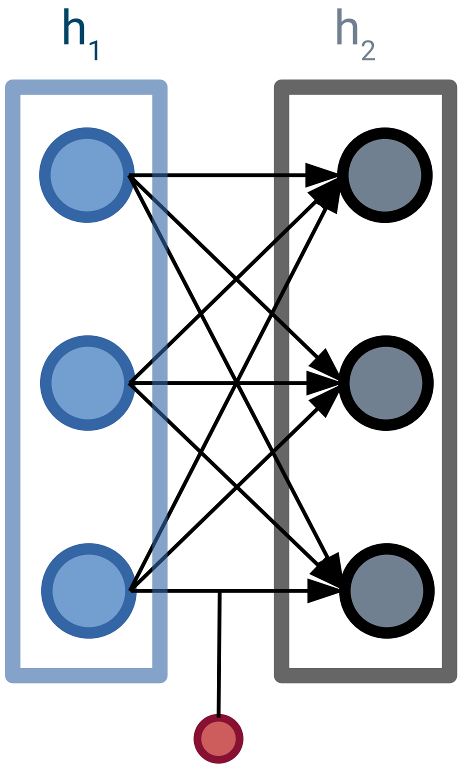


From Neurons to Layers

A Common Composition

- Add several neurons working on “parallel”.

Input

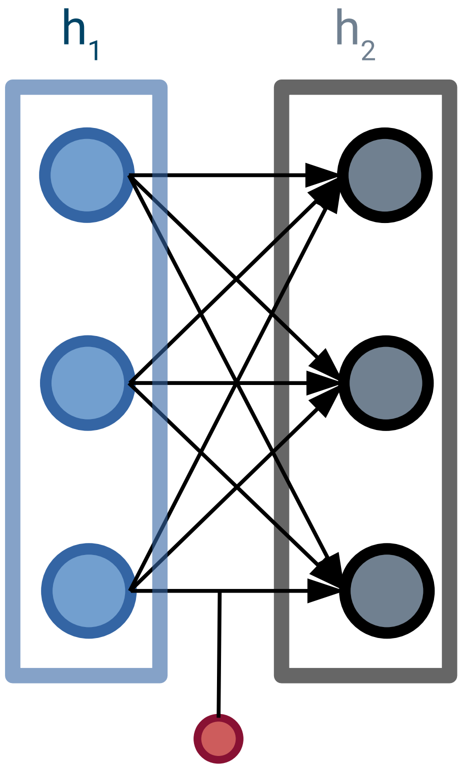


Output

From Neurons to Layers

A Common Composition

- Add several neurons working on "parallel".



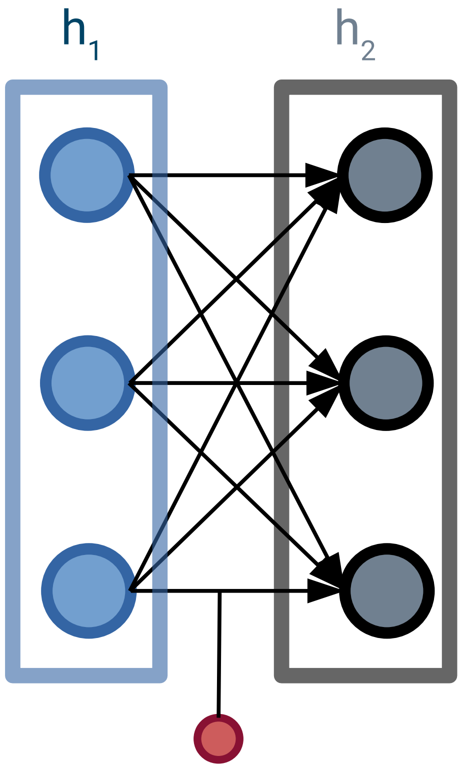
$$h(x, w, b) = \langle w, x \rangle + b$$

$$f_{linear}(x, W, b) = Wx + b$$

From Neurons to Layers

A Common Composition

- Add several neurons working on “parallel”.



$$h(x, w, b) = \langle w, x \rangle + b$$

$$f_{linear}(x, W, b) = Wx + b$$

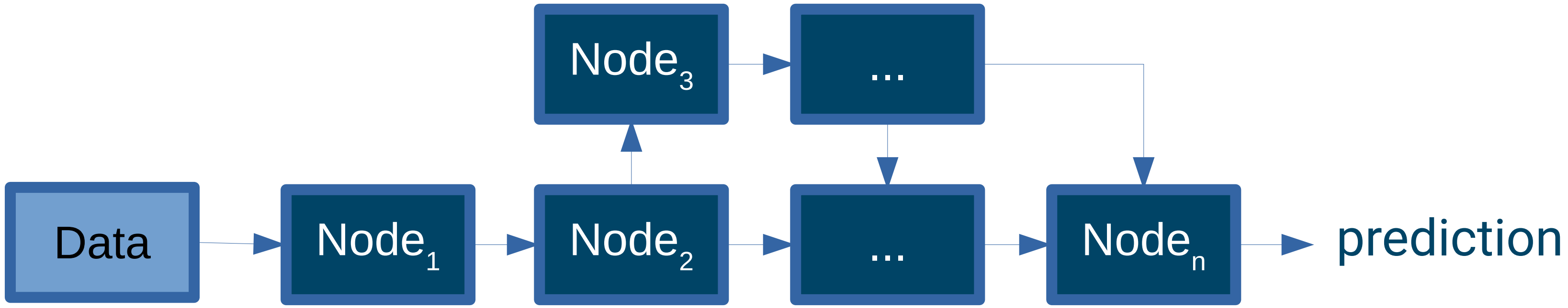
Why?

- Highly optimizable
  – Algorithmically (via smart matrix multiplication)
  – Hardware-wise (via GPUs, TPUs)
- Enable powerful compositions

From Layers to Neurons

Enabling Powerful Composition

- Add several neurons working on “parallel”.



Idea:

- Every neuron/layer → simple operation
- Using simple operations to build more complex ones
- Obtain a new quality out of the composition

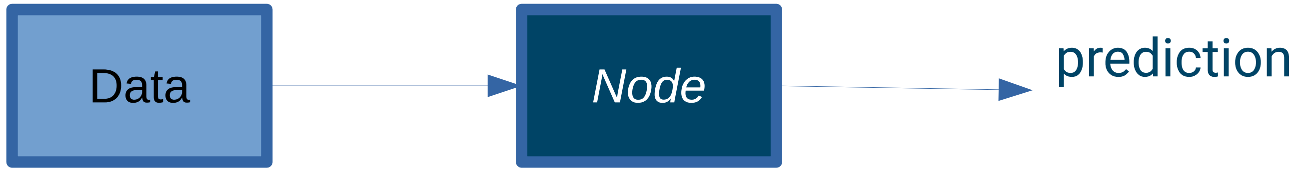
A Single-Layer Neural Network

[with few layers]

A Single Layer Neural Network

Given:

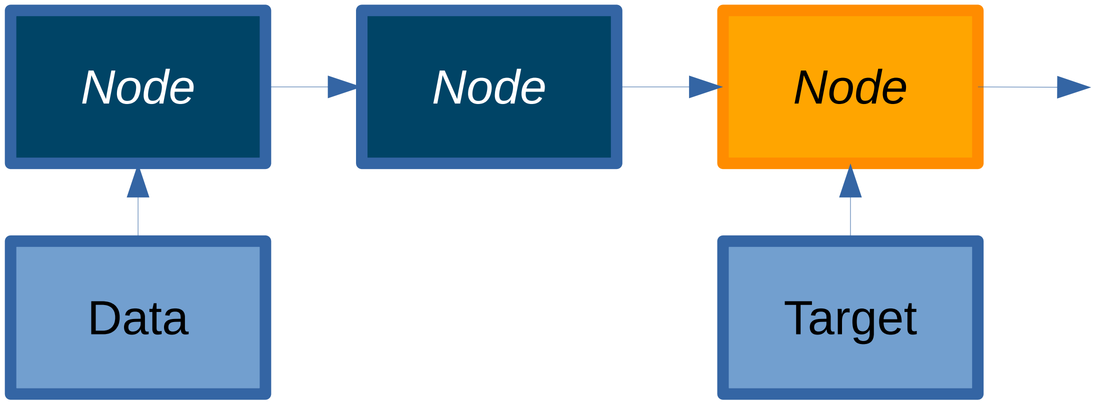
- Let's assume we have the following simple model



A Single Layer Neural Network

Given:

- Let’s assume we have the following simple model



Ok I see where this goes, but ...

How do we train one of those networks?

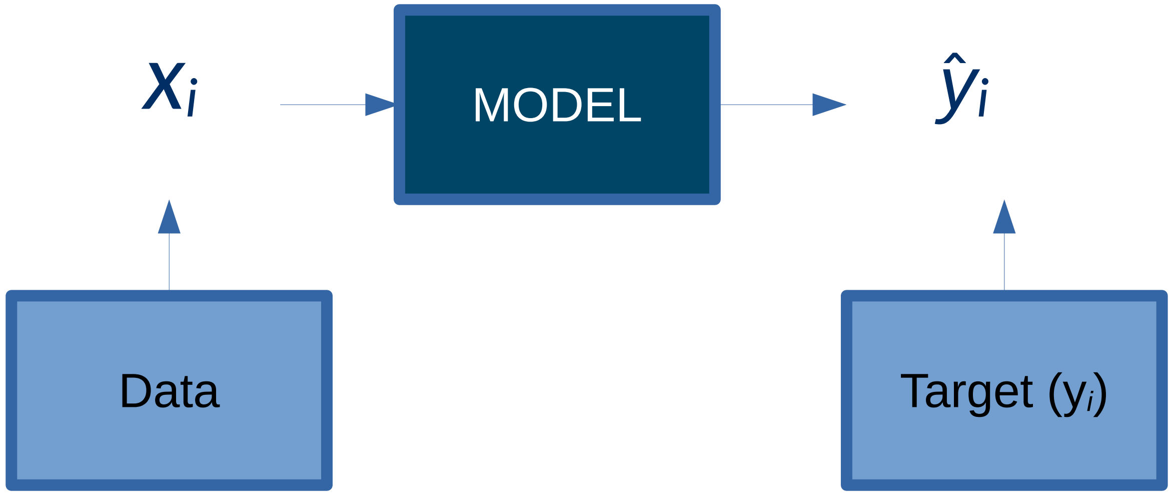
Ok I see where this goes, but …

How do we train a machine learning model in general?

Training a Model

Given:

- *Classification Task* with *k* classes.
- Training Data: inputs (x_i) and labels ($\hat{y}_i$)



A Simple Neural Network

How do we train such a model? → Learn the weights

- Let's see how we did it earlier

Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

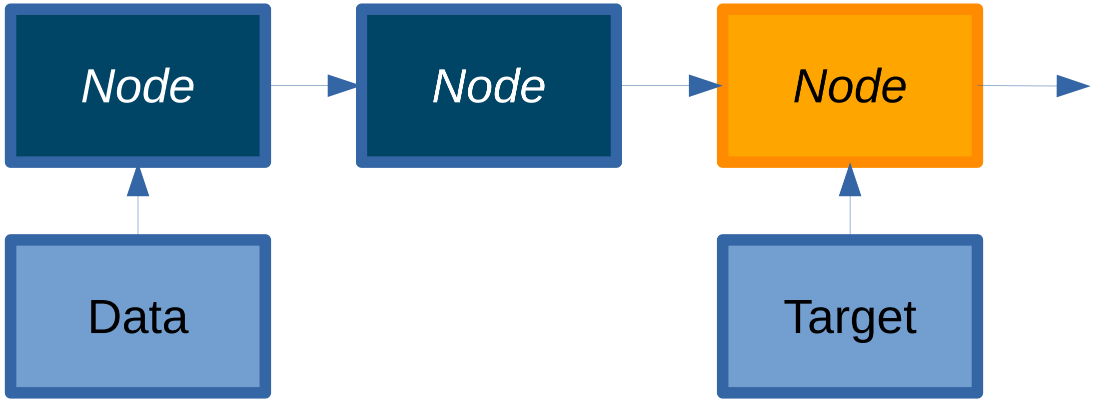
Requirements:

- Examples (with labels)
- A way to evaluate the "goodness" of the model
  (measure performance)
- Stopping criteria

A Single-Layer Neural Network

Given:

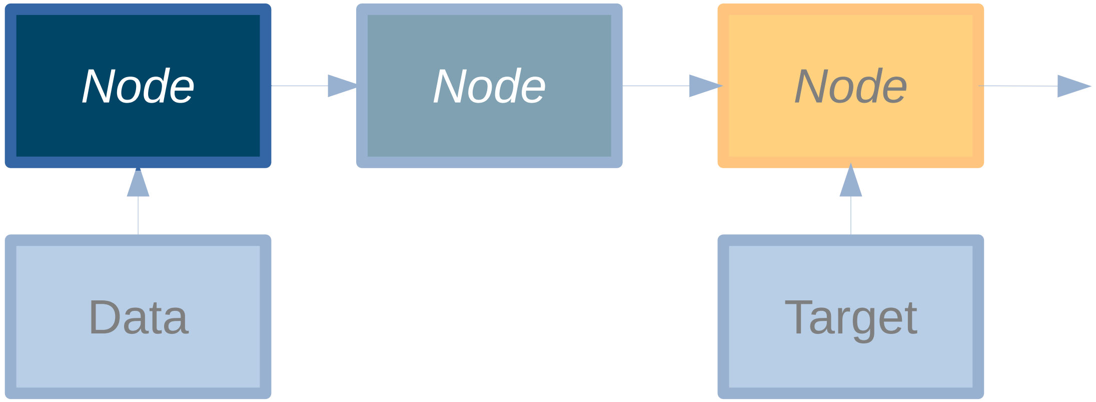
- Let’s assume we have the following simple model



A Single-Layer Neural Network

Given:

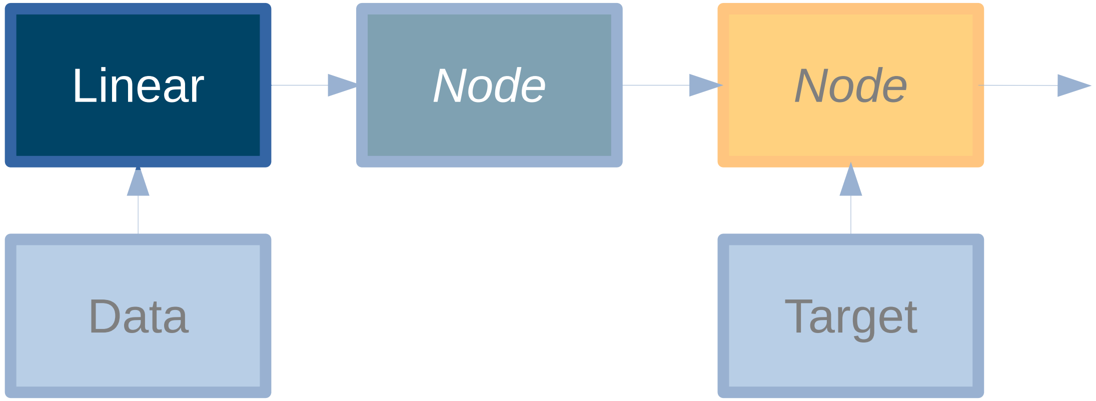
- Let’s assume we have the following simple model



A Single-Layer Neural Network

Given:

- Let’s assume we have the following simple model



$$h(x, w, b) = \langle w, x \rangle + b$$

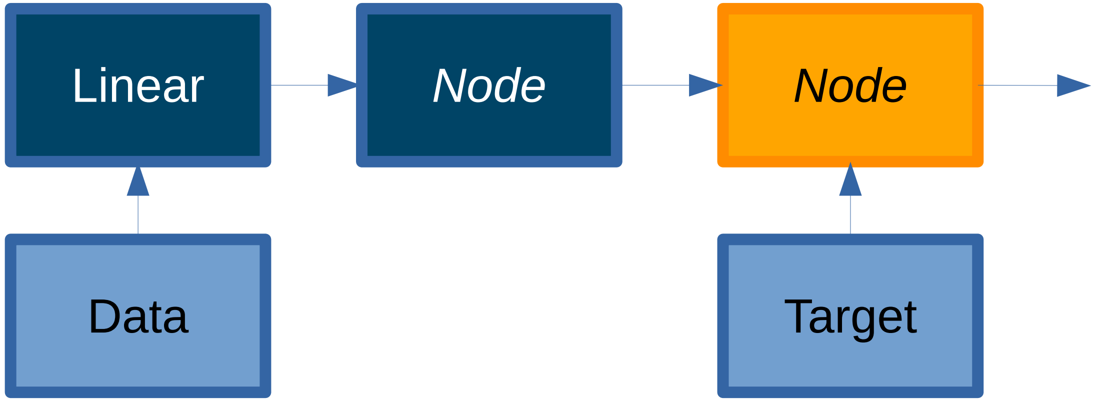
$$f_{linear}(x, W, b) = Wx + b$$

The very same equations of
***layers* of artificial perceptrons**

A Single-Layer Neural Network

Given:

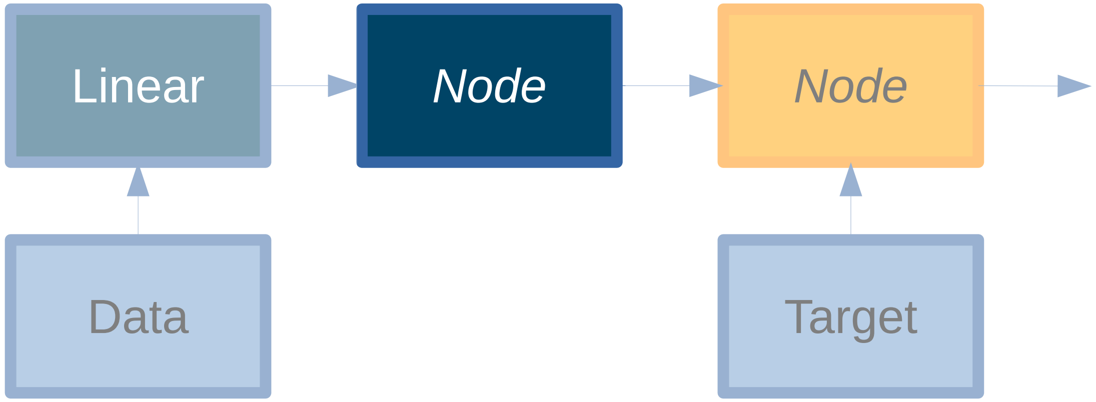
- Let’s assume we have the following simple model



A Single-Layer Neural Network

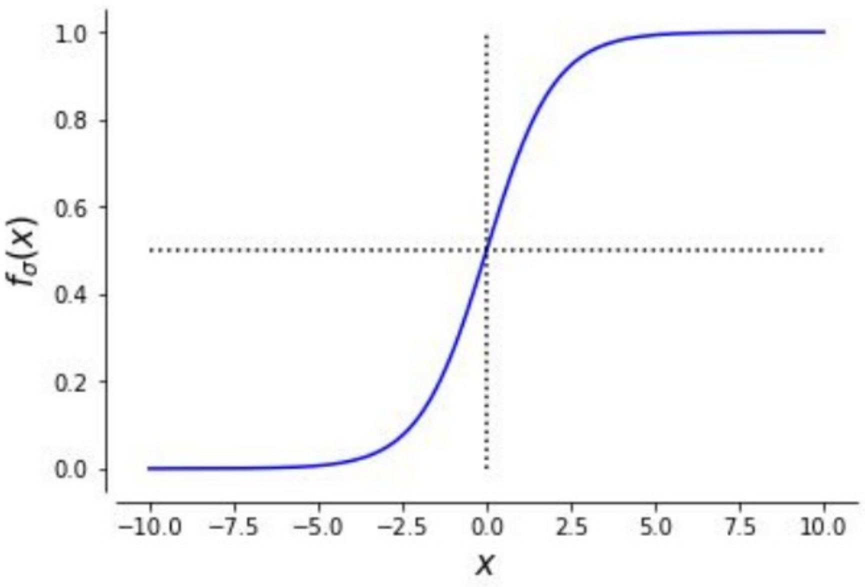
Given:

- Let’s assume we have the following simple model



A Single-Layer Neural Network

Activation Function – Sigmoid

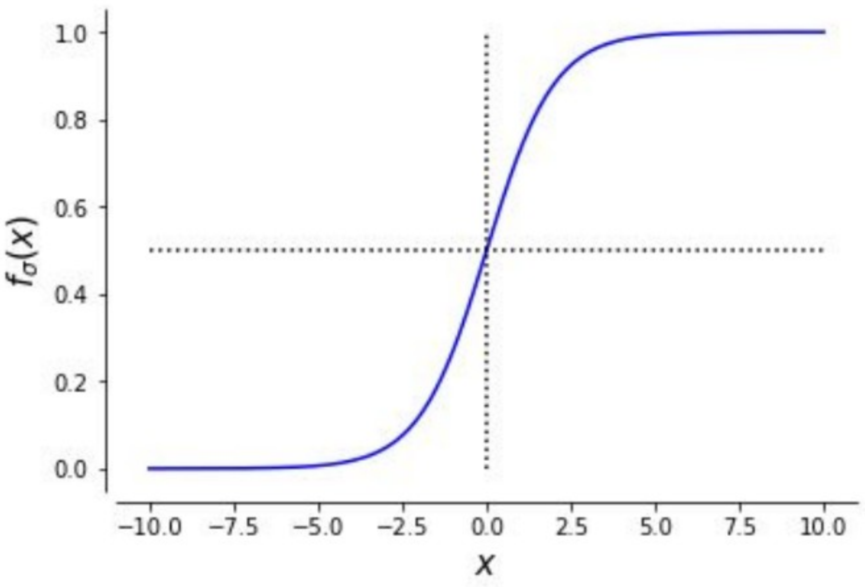


$$f_\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

University of Antwerp

A Single-Layer Neural Network

Activation Function – Sigmoid



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

Characteristics:

- Introduces non-linear behavior
- Scaled output [0-1]
- Simple derivatives

- Saturates
  - Vanishing derivatives

Note:

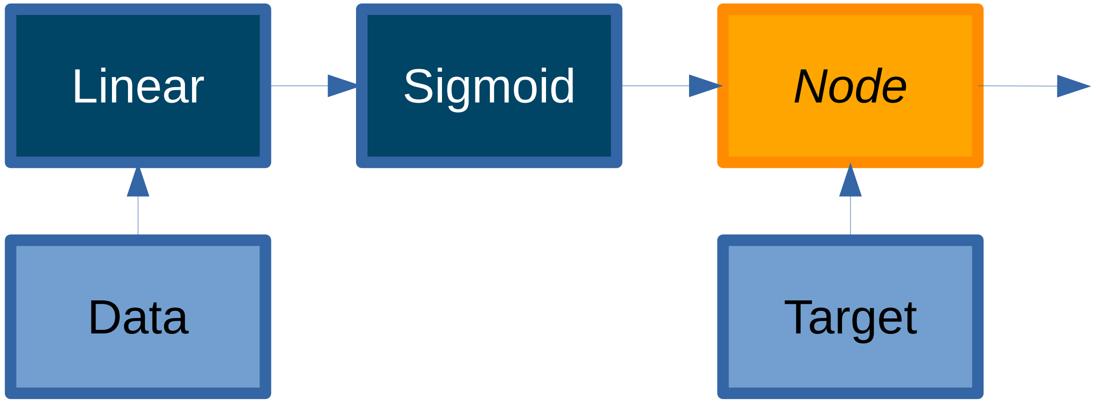
- Often called “non-linearities”
- Applied point-wise

University of Antwerp

A Single-Layer Neural Network

Given:

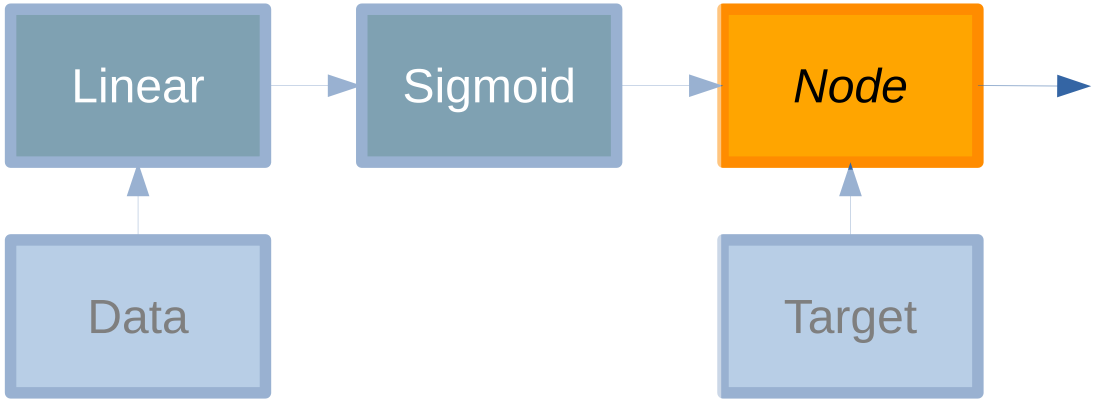
- Let’s assume we have the following simple model



A Single-Layer Neural Network

Given:

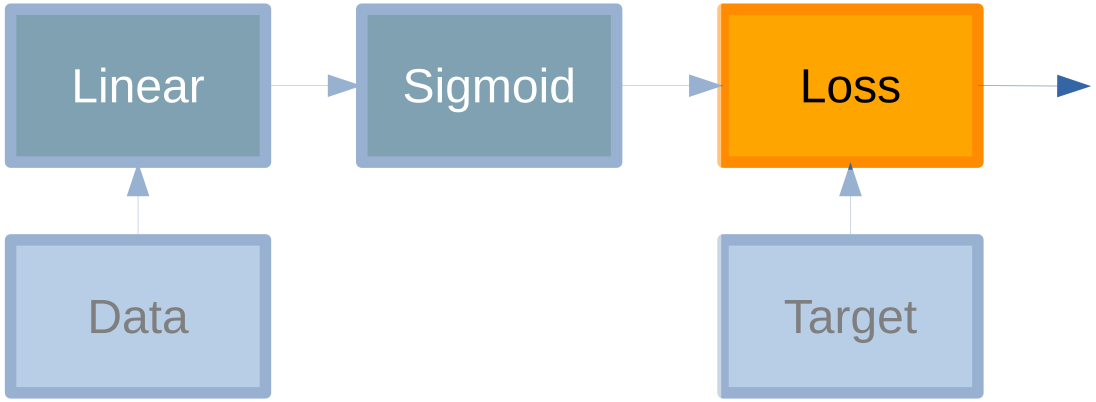
- Let’s assume we have the following simple model



A Single-Layer Neural Network

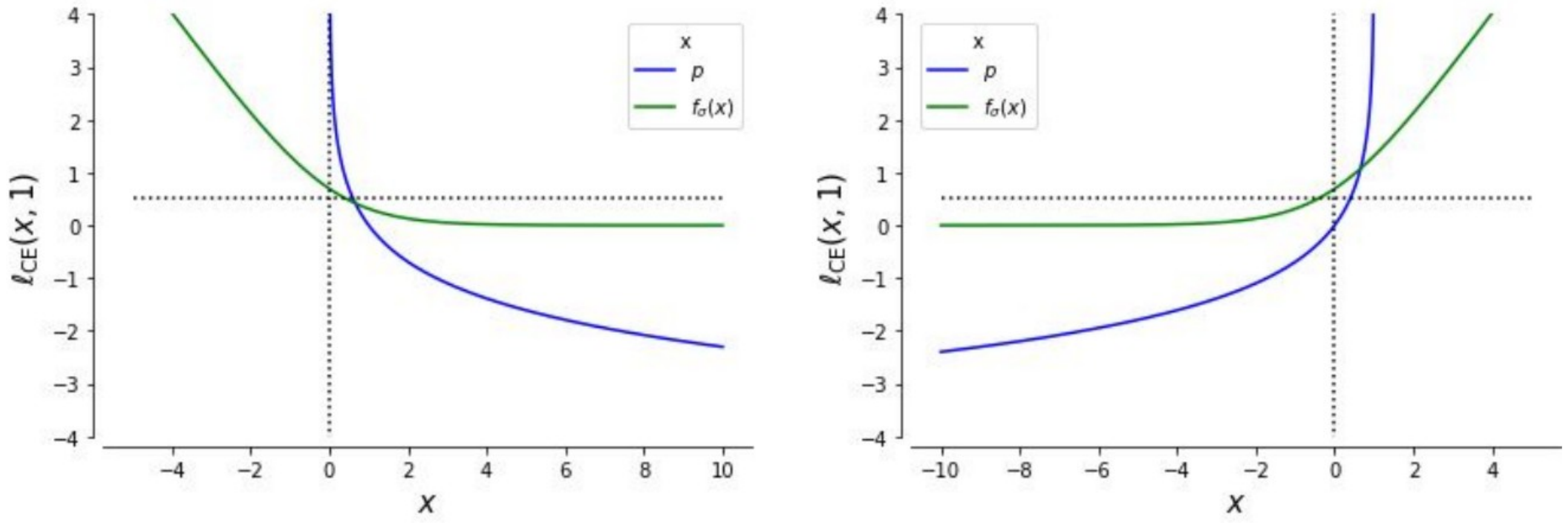
Given:

- Let's assume we have the following simple model



A Single-Layer Neural Network

Loss Function – Cross Entropy



$$l_{CE}(p,t) = -[t \ \log(p) + (1-t) \ \log(1-p)$$

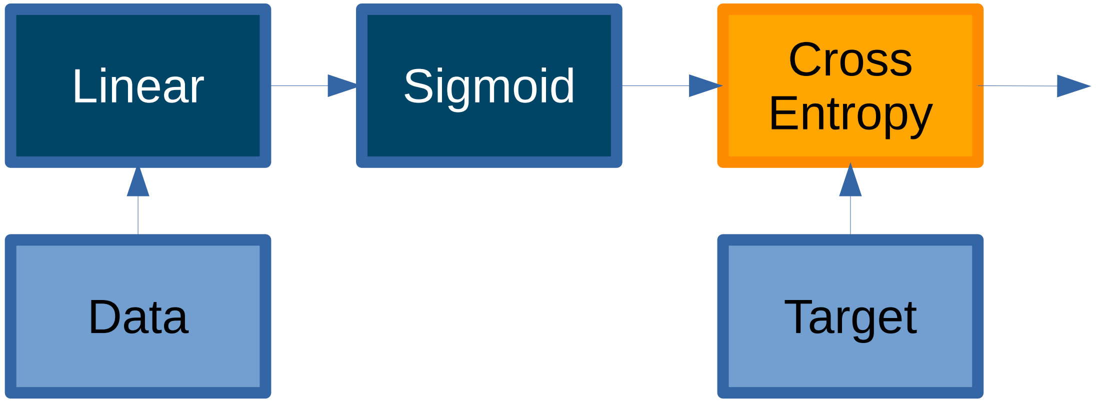
Characteristics:

- Negation of logarithm of probability of correct prediction
- Composable with sigmoid
- Numerically unstable

A Single-Layer Neural Network

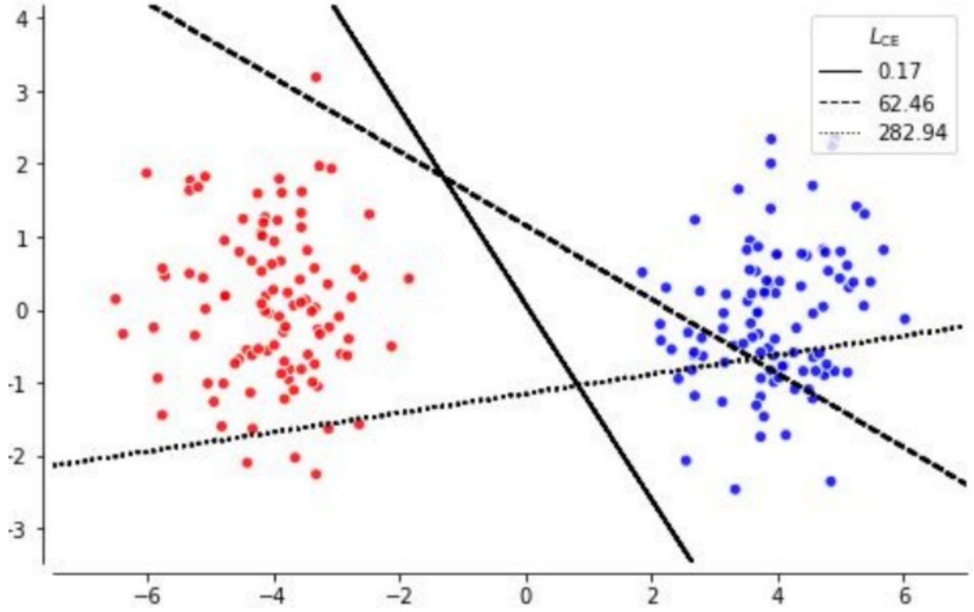
Given:

- Let's assume we have the following simple model



A Single-Layer Neural Network

Loss Function – Cross Entropy



Characteristics:

- **Additive w.r.t. samples**

  (highly desirable)

- Negation of logarithm of probability of correct prediction

  (on the entire dataset)

- Numerically unstable

$$L_{CE}(p,t) = -\sum_{i=1}^{n}[t^{(i)}\ \log(p^{(i)}) + (1-t^{(i)})\ \log(1-p^{(i)})$$

Beyond Binary Classification

[with few layers + multiple classes are possible]

A Single-Layer Neural Network

Beyond Binary Classification – Softmax

$$f_{sm}(x) = \frac{e^{x}}{\sum_{j=1}^{k} e^{x_j}}$$

A Single-Layer Neural Network

Beyond Binary Classification – Softmax

$$f_{sm}(x) = \frac{e^{x}}{\sum_{j=1}^{k} e^{x_j}}$$

Considering,

$$f_{sm}([x, 0]) = \left[\frac{e^{x}}{e^{x}+e^{0}}, \frac{e^{0}}{e^{x}+e^{0}}\right]$$

$$= [f_\sigma(x), 1 - f_\sigma(x)]$$

A Single-Layer Neural Network

Beyond Binary Classification – Softmax

$$f_{sm}(x) = \frac{e^{x}}{\sum_{j=1}^{k} e^{x_j}}$$

Considering,

$$f_{sm}([x, 0]) = \left[\frac{e^{x}}{e^{x}+e^{0}}, \frac{e^{0}}{e^{x}+e^{0}}\right]$$

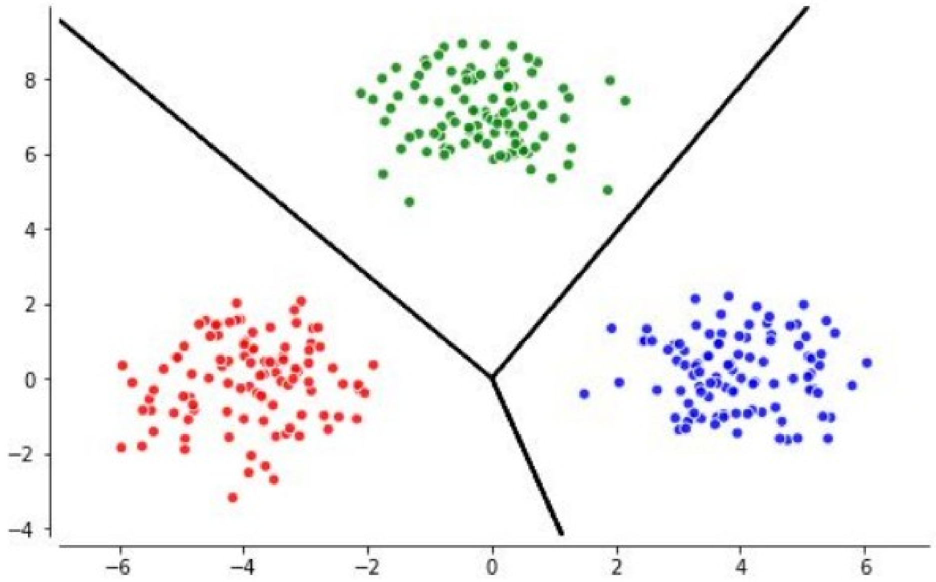
$$= [f_{\sigma}(x), 1 - f_{\sigma}(x)]$$

Characteristics:

- Generalization of the sigmoid
- Does not work properly with sparse outputs
- Does not scale properly w.r.t. the number of classes (k)

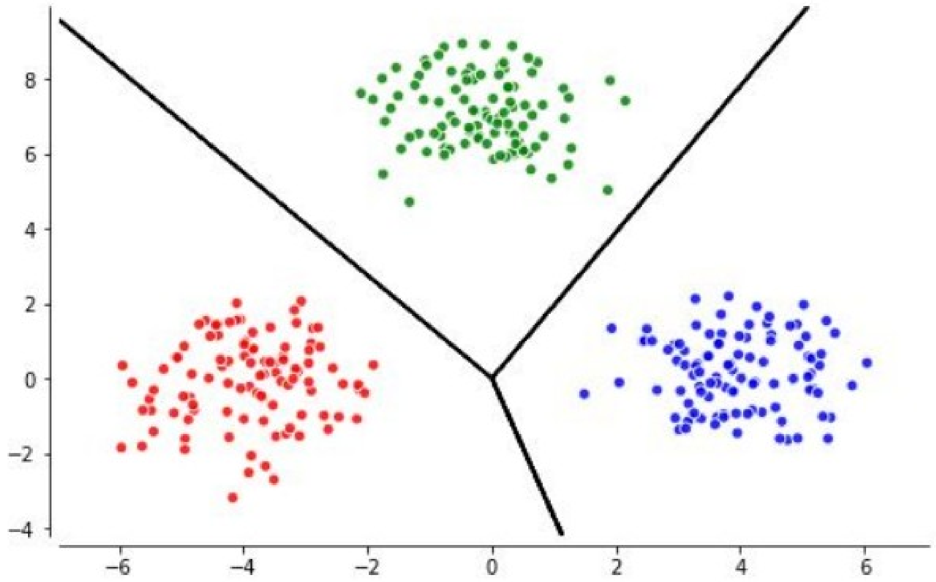
A Single-Layer Neural Network

Beyond Binary Classification – Softmax + Cross Entropy



A Single-Layer Neural Network

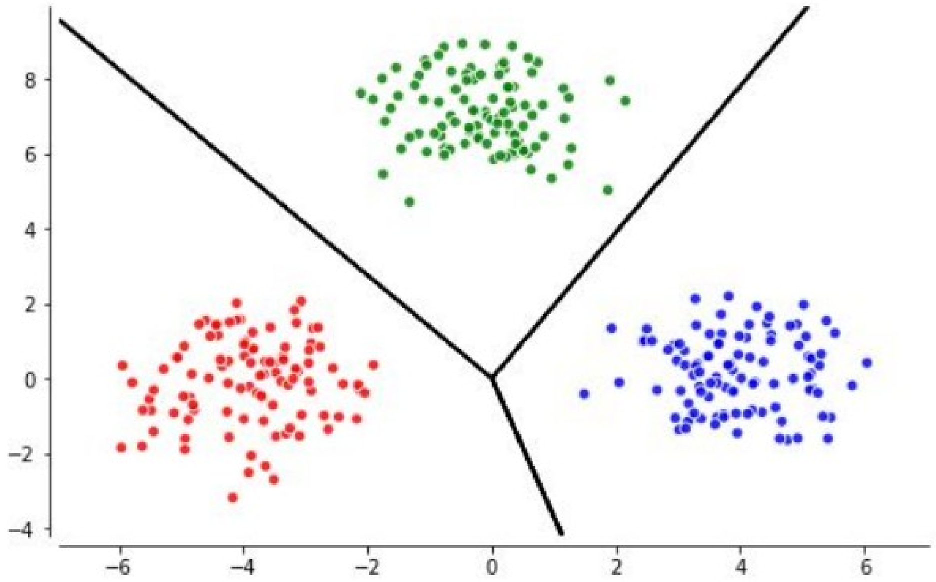
Beyond Binary Classification – Softmax + Cross Entropy



$$l_{CE}(f_{sm}(x), t) = -\sum_{j=1}^{k} t_j \, \log[f_{sm}(x_j)]$$

A Single-Layer Neural Network

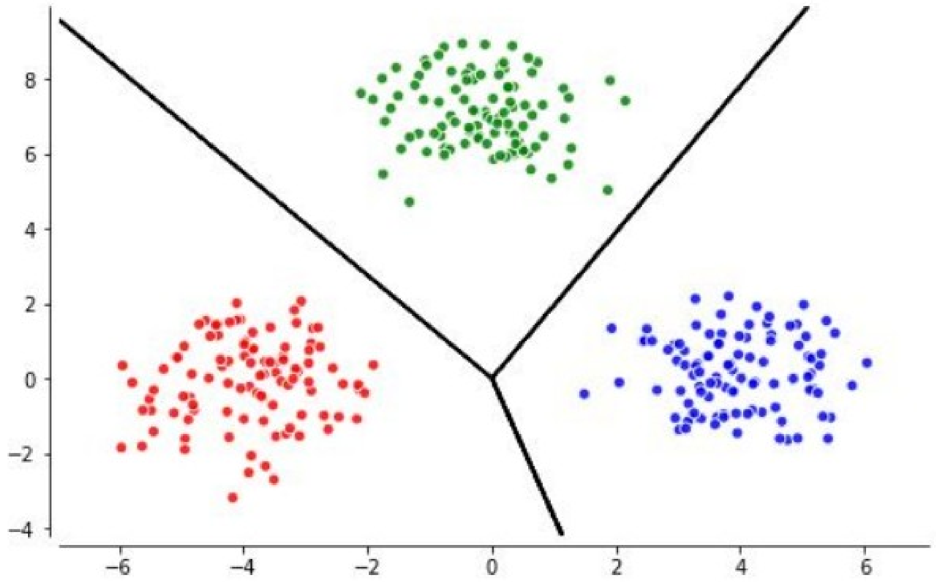
Beyond Binary Classification – Softmax + Cross Entropy



$$l_{CE}(f_{sm}(x), t) = -\sum_{j=1}^{k} t_j \log[f_{sm}(x_j)] = -\sum_{j=1}^{k} t_j [x_j - \log \sum_{l=1}^{k} e^{x_l}]$$

A Single-Layer Neural Network

Beyond Binary Classification – Softmax + Cross Entropy



Characteristics:

- Generalization of the sigmoid
- Becomes numerically stable
- Simple, yet powerful
  (~92% in handwritten digit recognition)

$$l_{CE}(f_{sm}(x), t) = -\sum_{j=1}^{k} t_j \log[f_{sm}(x_j)] = -\sum_{j=1}^{k} t_j [x_j - \log \sum_{l=1}^{k} e^{x_l}]$$

A Single-Layer Neural Network

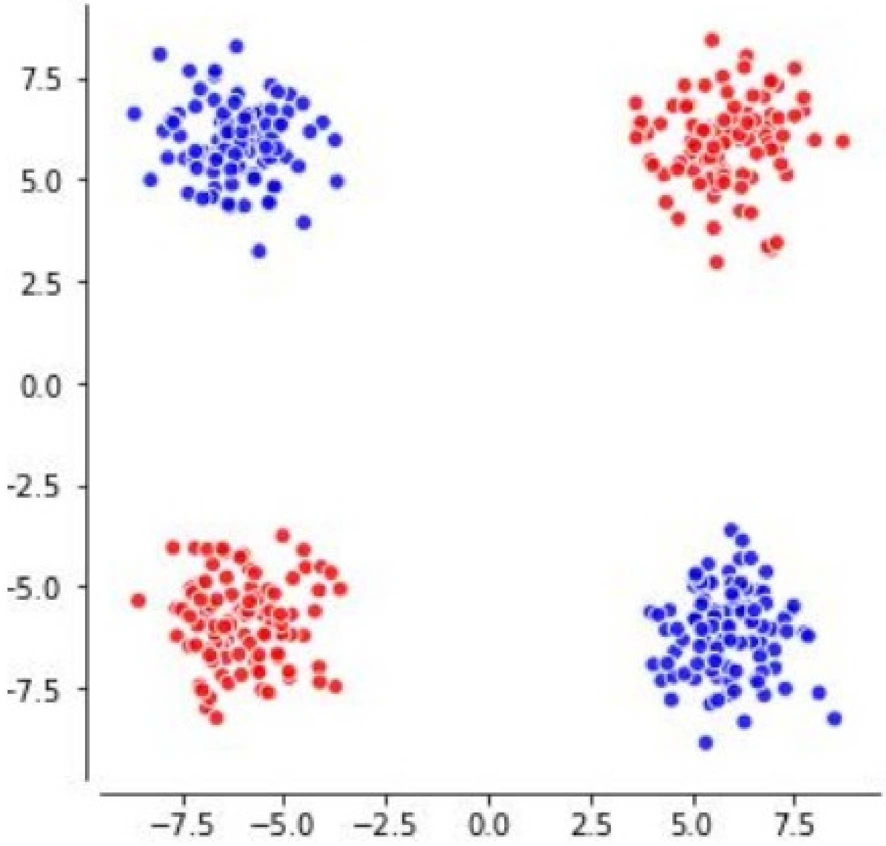
Beyond Binary Classification – Softmax + Cross Entropy

- Simple, yet powerful (~92% in handwritten digit recognition)

MNIST dataset, [Le Cunn et al., 1998a]

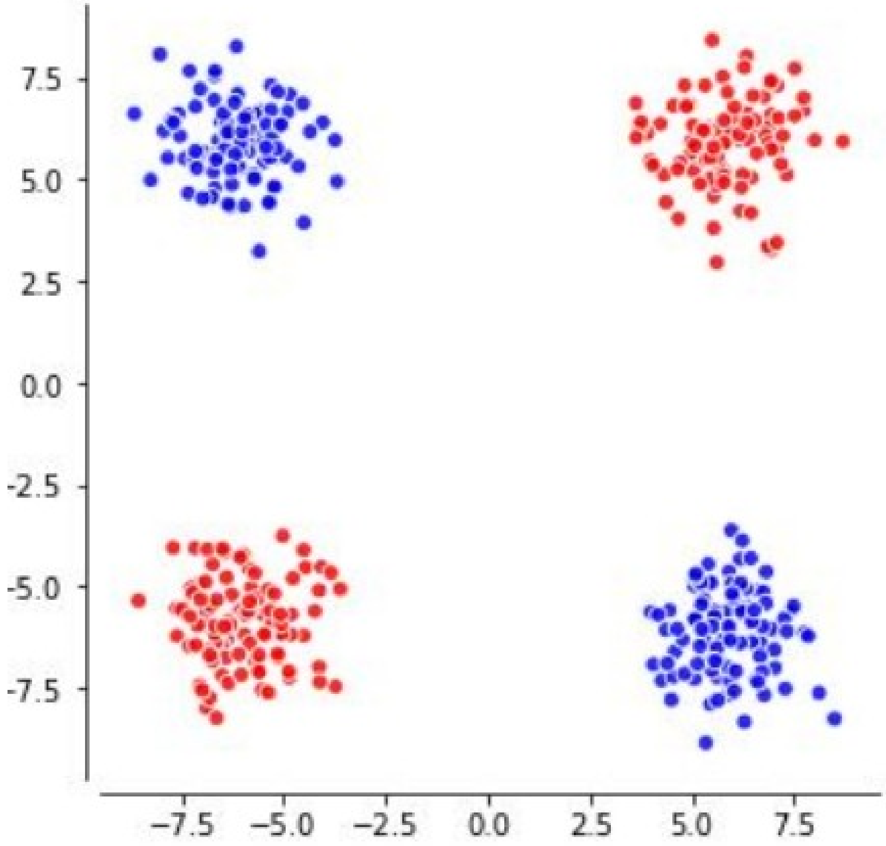
A Single-Layer Neural Network

What if we encounter the following problem?



A Single-Layer Neural Network

What if we encounter the following problem?



Exclusive Disjunction (XOR)

$$p \oplus q = (p \vee q) \wedge \neg(p \wedge q)$$

A Two-Layer Neural Network

[with few layers]

Break

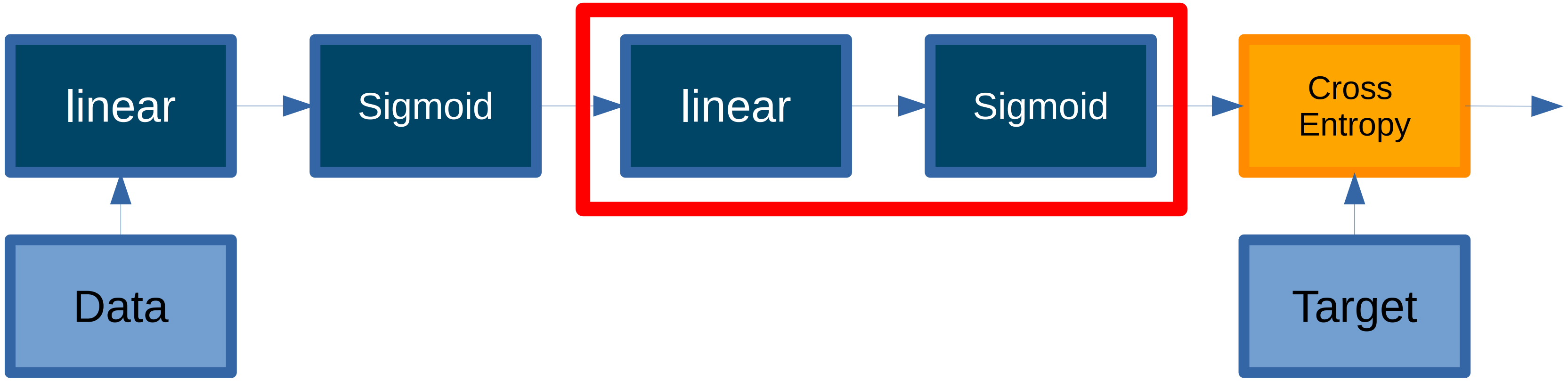
See you in few minutes

A Two-Layer Neural Network

[with few layers]

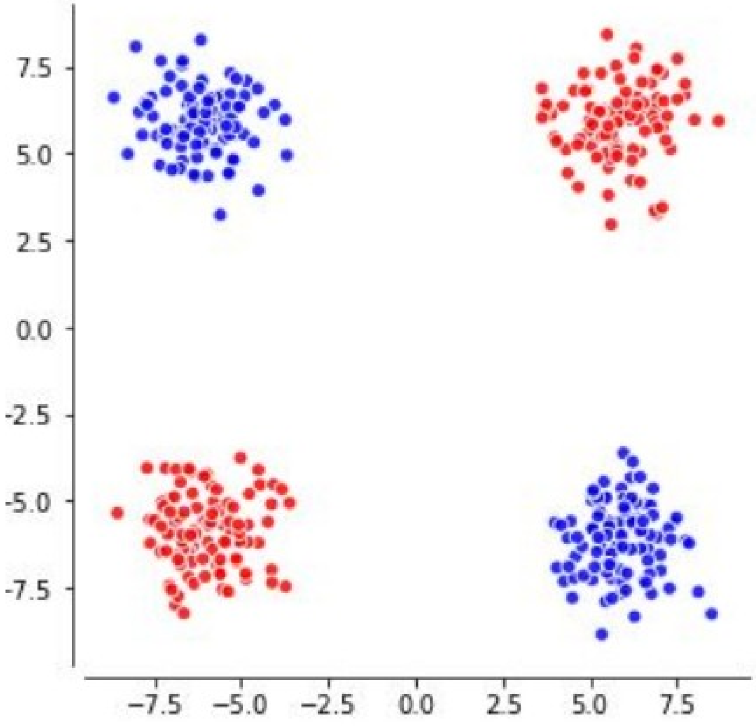
A Two-Layer Neural Network

Extending the previous schematic



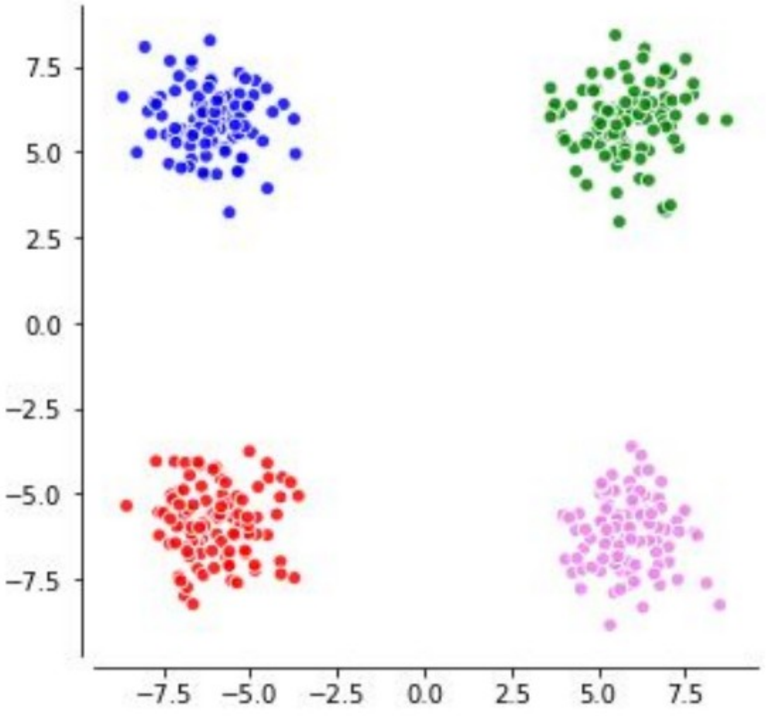
A Two-Layer Neural Network

Going back to the original problem



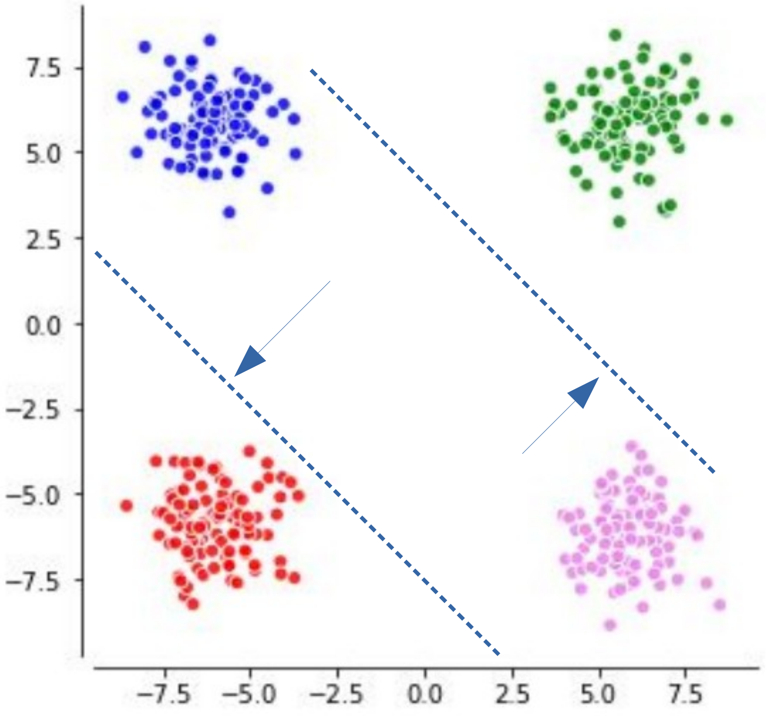
A Two-Layer Neural Network

Going back to the original problem



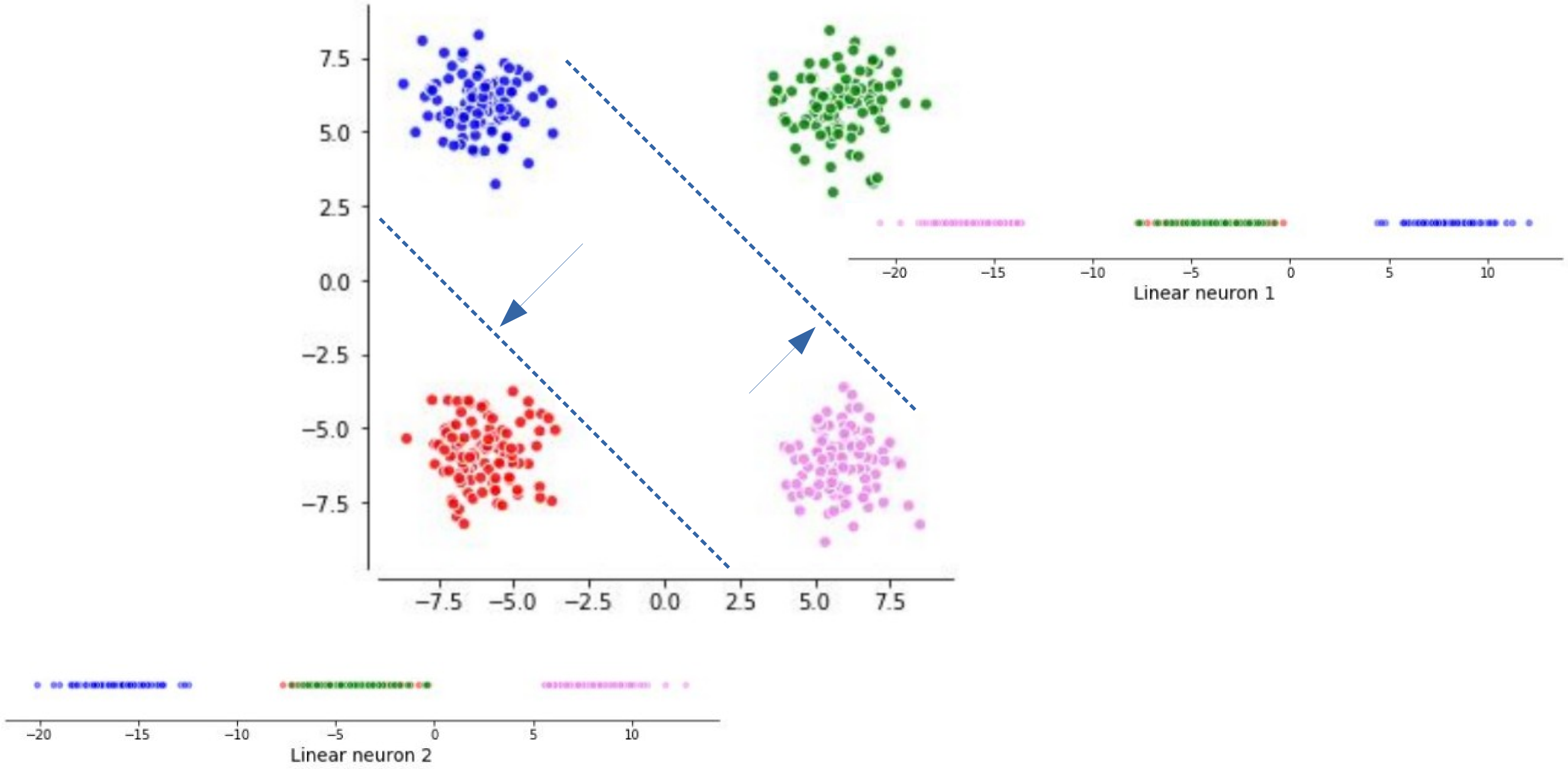
A Two-Layer Neural Network

Let’s assume we have the following hyperplanes



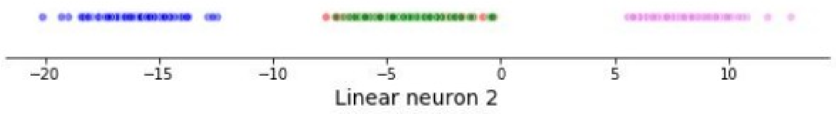
A Two-Layer Neural Network

Projecting our samples on the planes

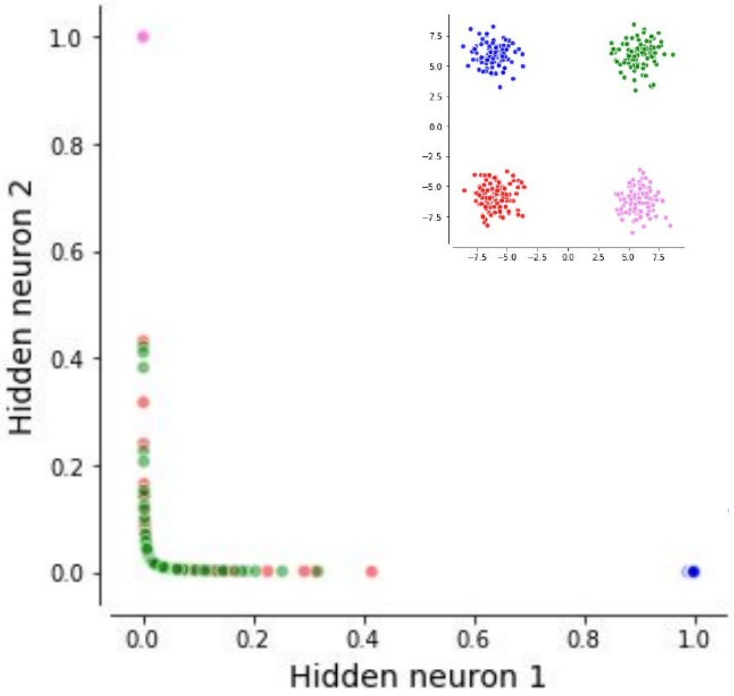


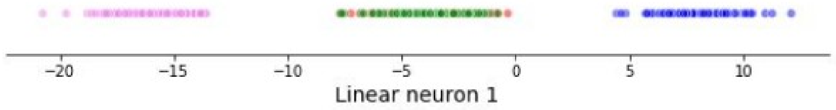
A Two-Layer Neural Network

Going back to the original problem



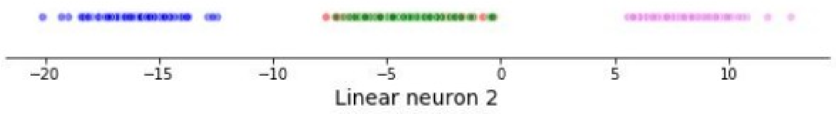
- Squashing our samples to the range [0,1]



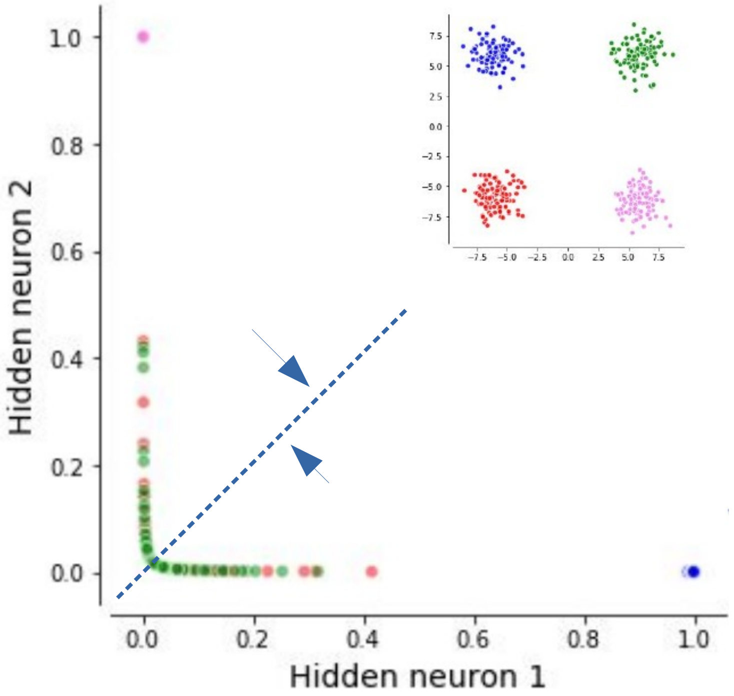


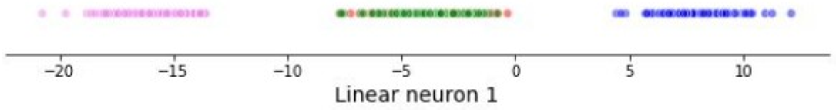
A Two-Layer Neural Network

Going back to the original problem



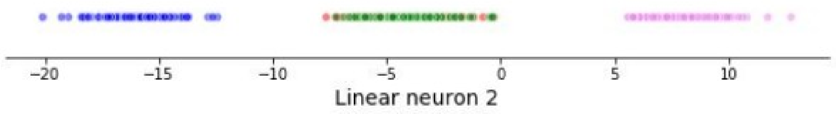
- Squashing our samples to the range [0,1]



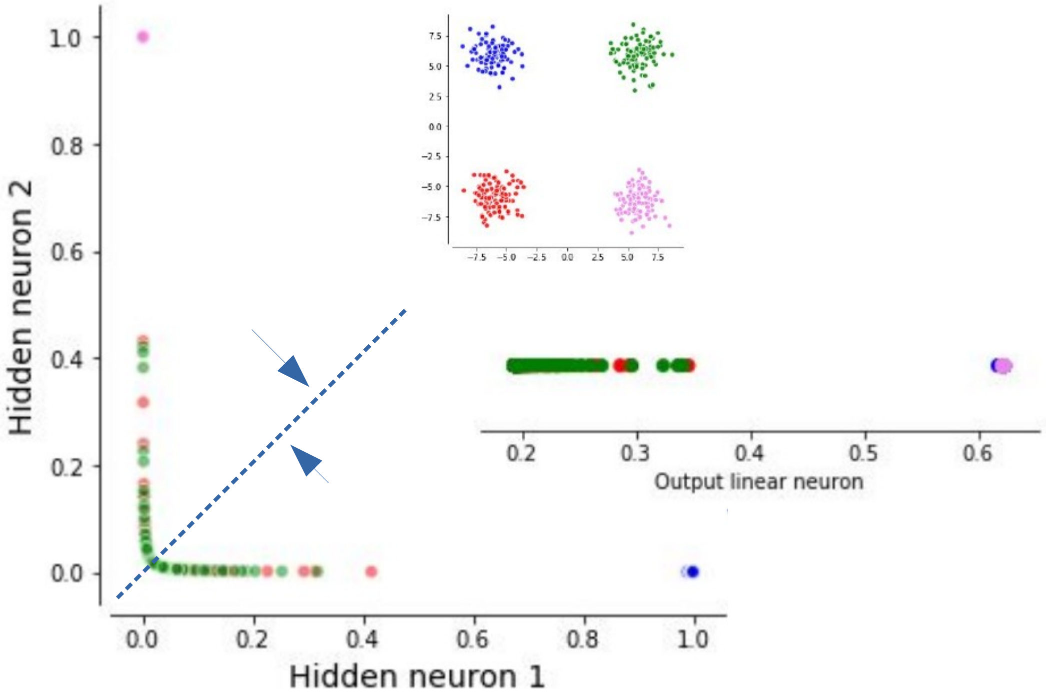


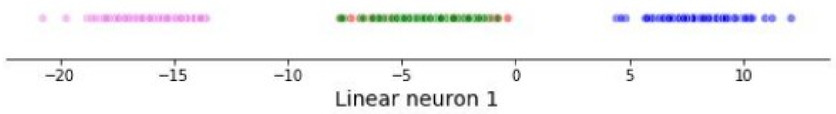
A Two-Layer Neural Network

Going back to the original problem



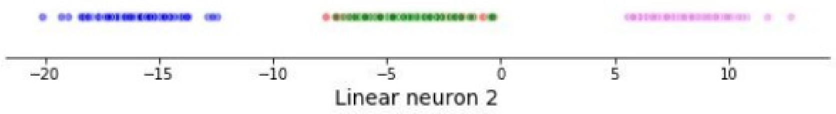
- Squashing our samples to the range [0,1]
- The hidden-layer provides a non-linear input space.



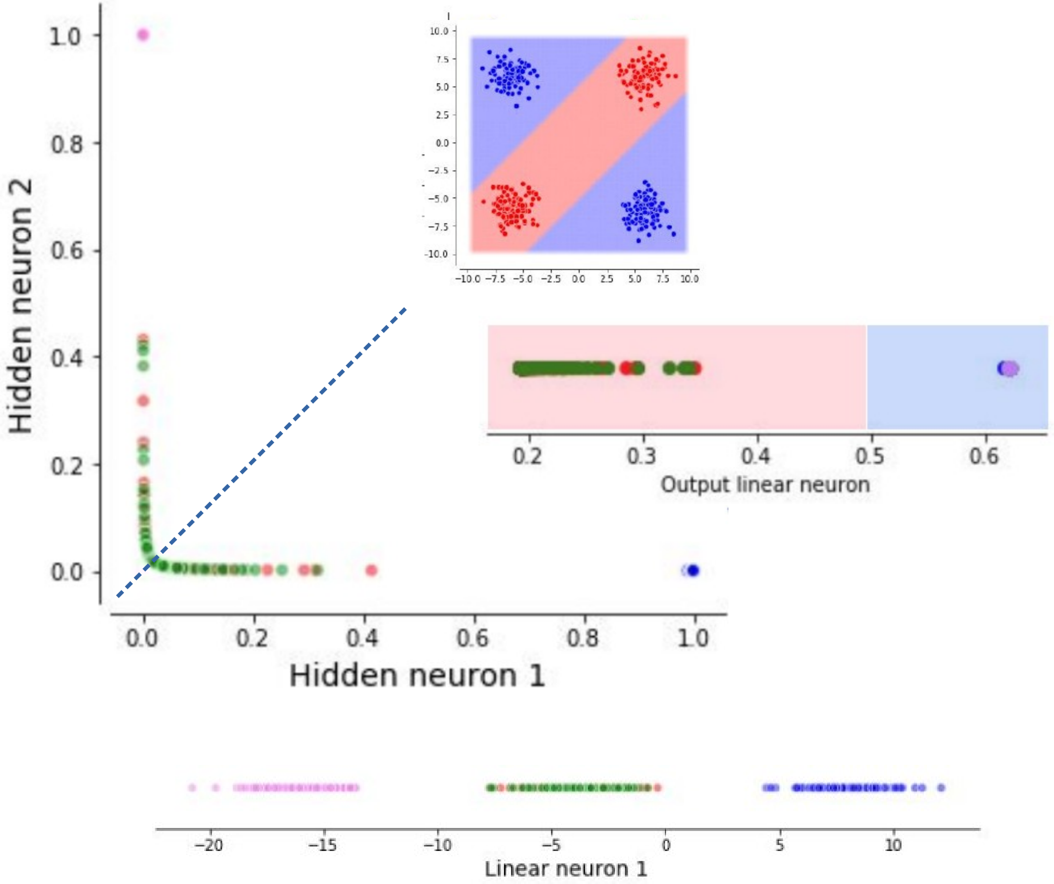


A Two-Layer Neural Network

Going back to the original problem



- Squashing our samples to the range [0,1]
- The hidden-layer provides a non-linear input space.



[Blum E.K, Neural Computation, 1989]

Nice, but …

What if we have a more complex problem?

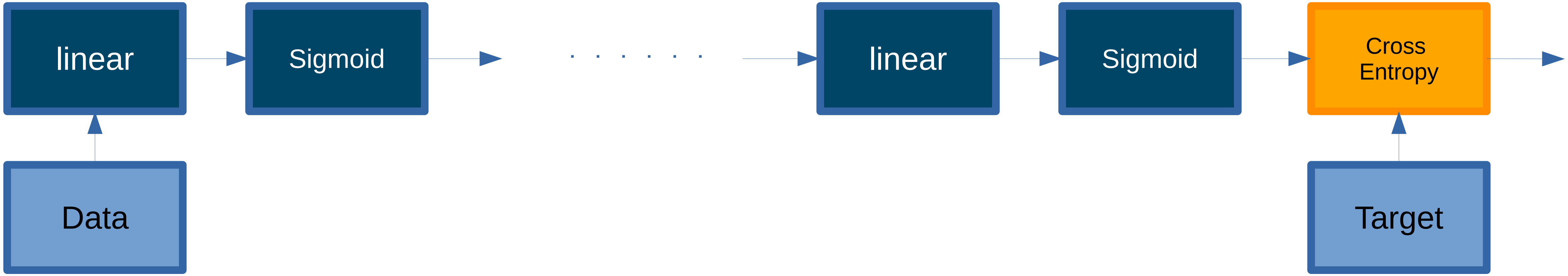

WE NEED TO GO
DEEPER
memegenerat

Deep Neural Networks

[adding more and more layers | something something “deep learning”]

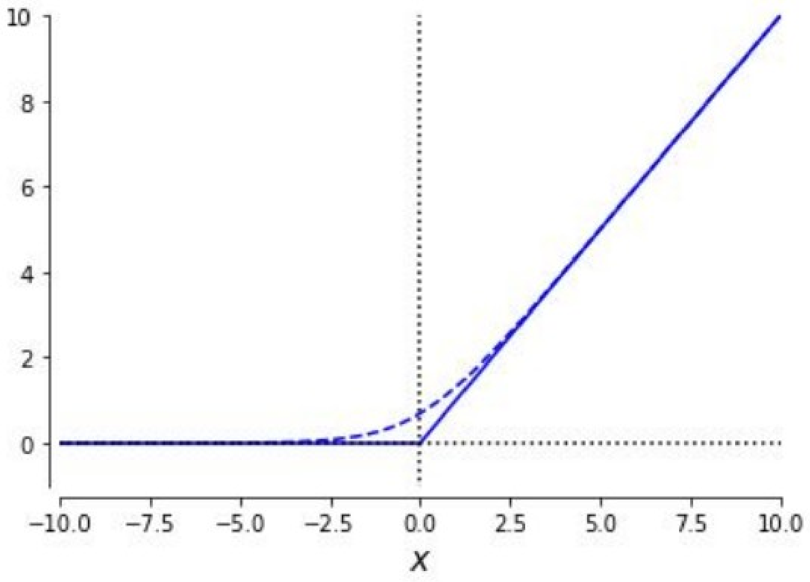
Deep Neural Networks

Further extending the previous schematic



Deep Neural Networks

Activation Function – Rectifier Linear Unit

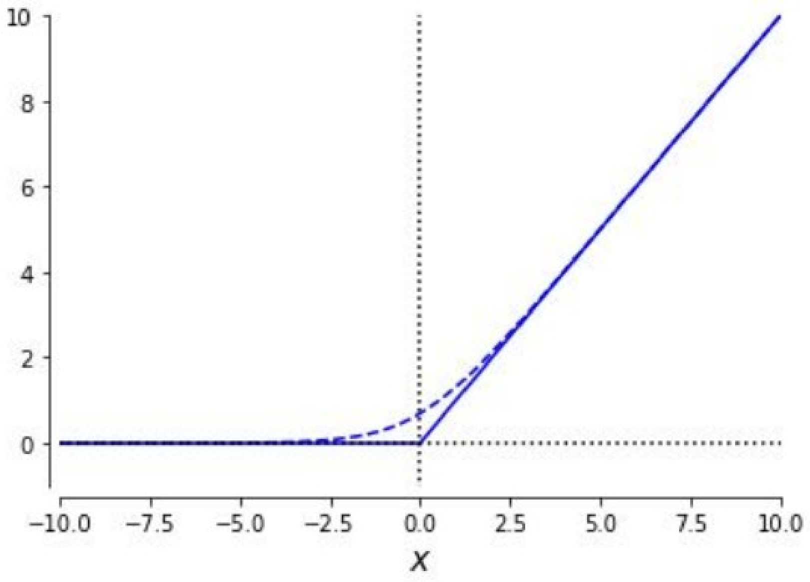


$$f_{relu} = \max(0, x)$$

University of Antwerp

Deep Neural Networks

Activation Function – Rectifier Linear Unit



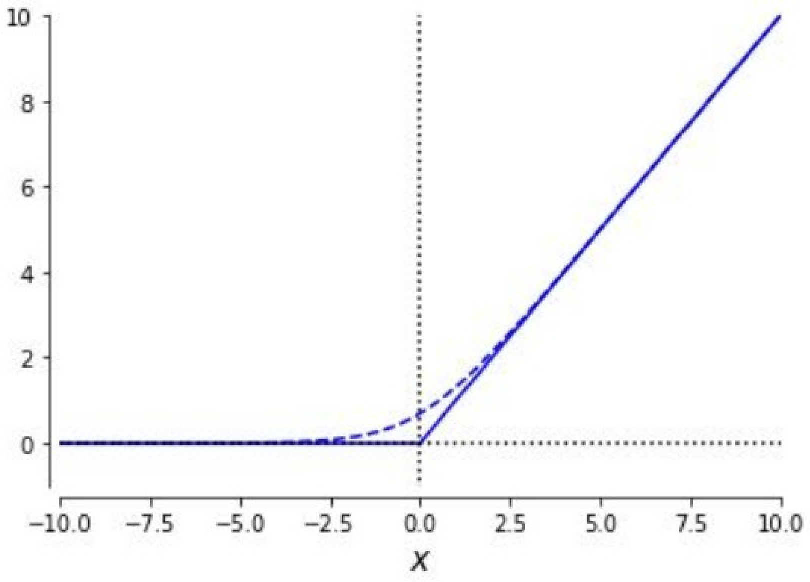
$$f_{relu} = \max(0, x)$$

Characteristics:

- Point-wise operation
- Not linear, but piece-wise linear
- Cut the space into polyhedra

Deep Neural Networks

Activation Function – Rectifier Linear Unit



$$f_{relu} = \max(0, x)$$

Characteristics:

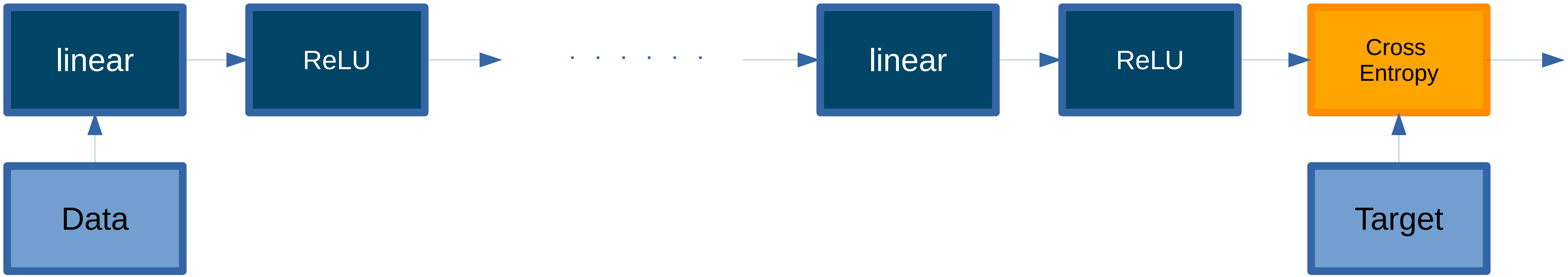
- Point-wise operation
- Not linear, but piece-wise linear
- Cut the space into polyhedra

Note

- Dead neurons can occur
- Not differentiable at 0
- Derivatives do not vanish

Deep Neural Networks

Further extending the previous schematic



Nice, but ...

Does it always work?

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

Given a continuous function from the hypercube to a single real value.

A large network **can approximate** (up to some error epsilon), **not represent,** any smooth function.

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

Given a continuous function from the hypercube to a single real value.

A large network **can approximate** (up to some error epsilon), **not represent,** any smooth function.

Does not provides guarantees over the “learnability” of such network.

Size of the network grows exponentially w.r.t. the input dimensions

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

Given a continuous function from the hypercube to a single real value.

A large network **can approximate** (up to some error epsilon), **not represent,** any smooth function.

Does not provides guarantees over the “learnability” of such network.

Size of the network grows exponentially w.r.t. the input dimensions

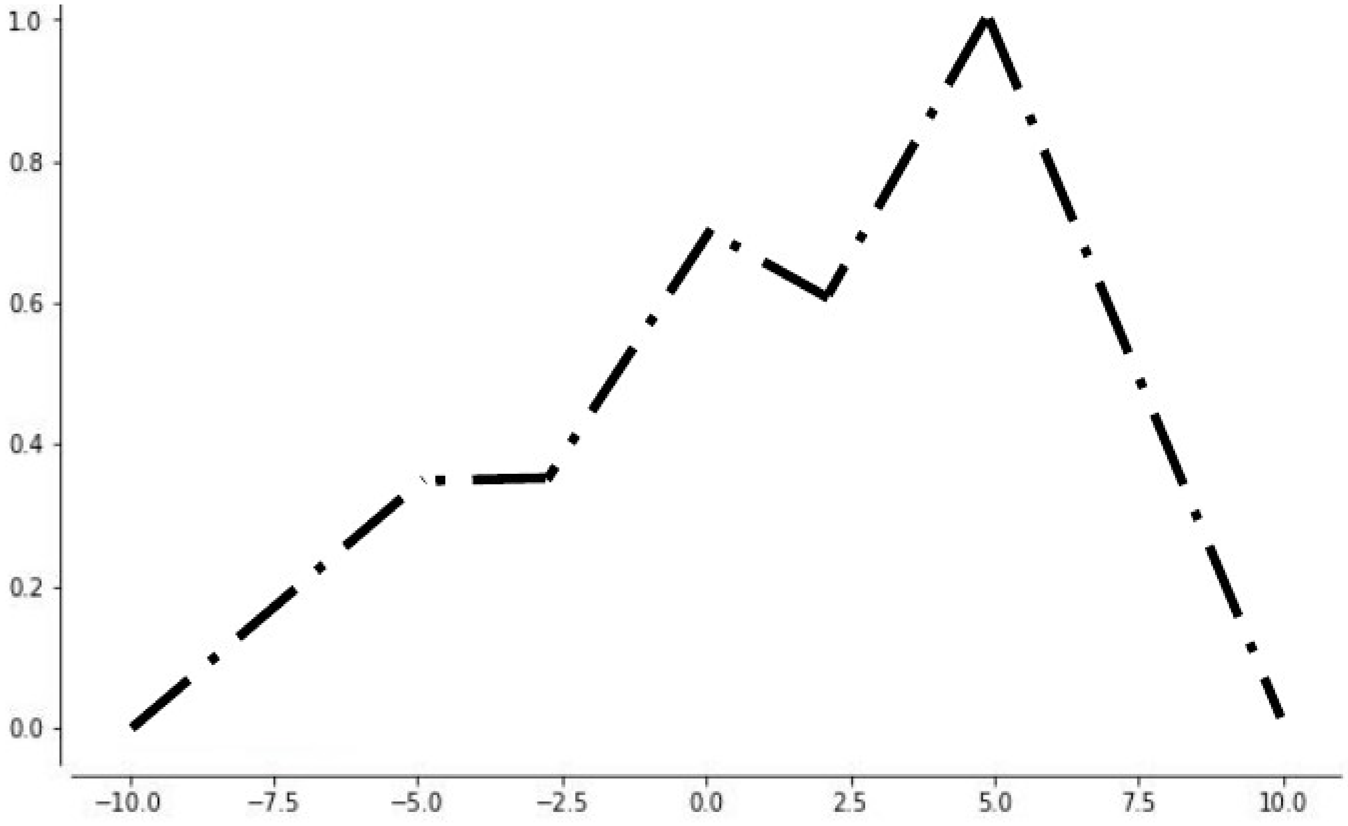
[Hornik, 1991]

The key is that the stacked components are non-constant and bounded

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

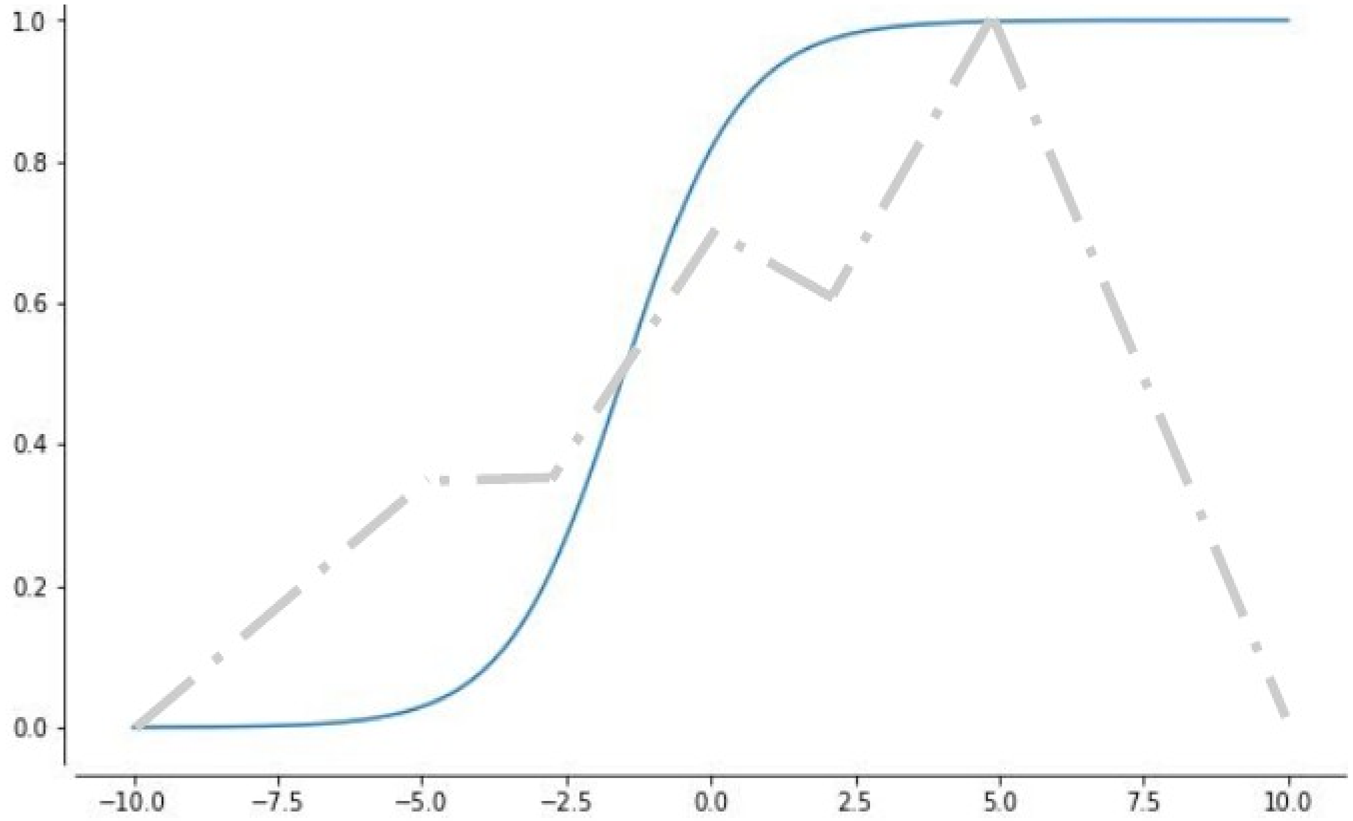
- An intuition on how it works



Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

- An intuition on how it works

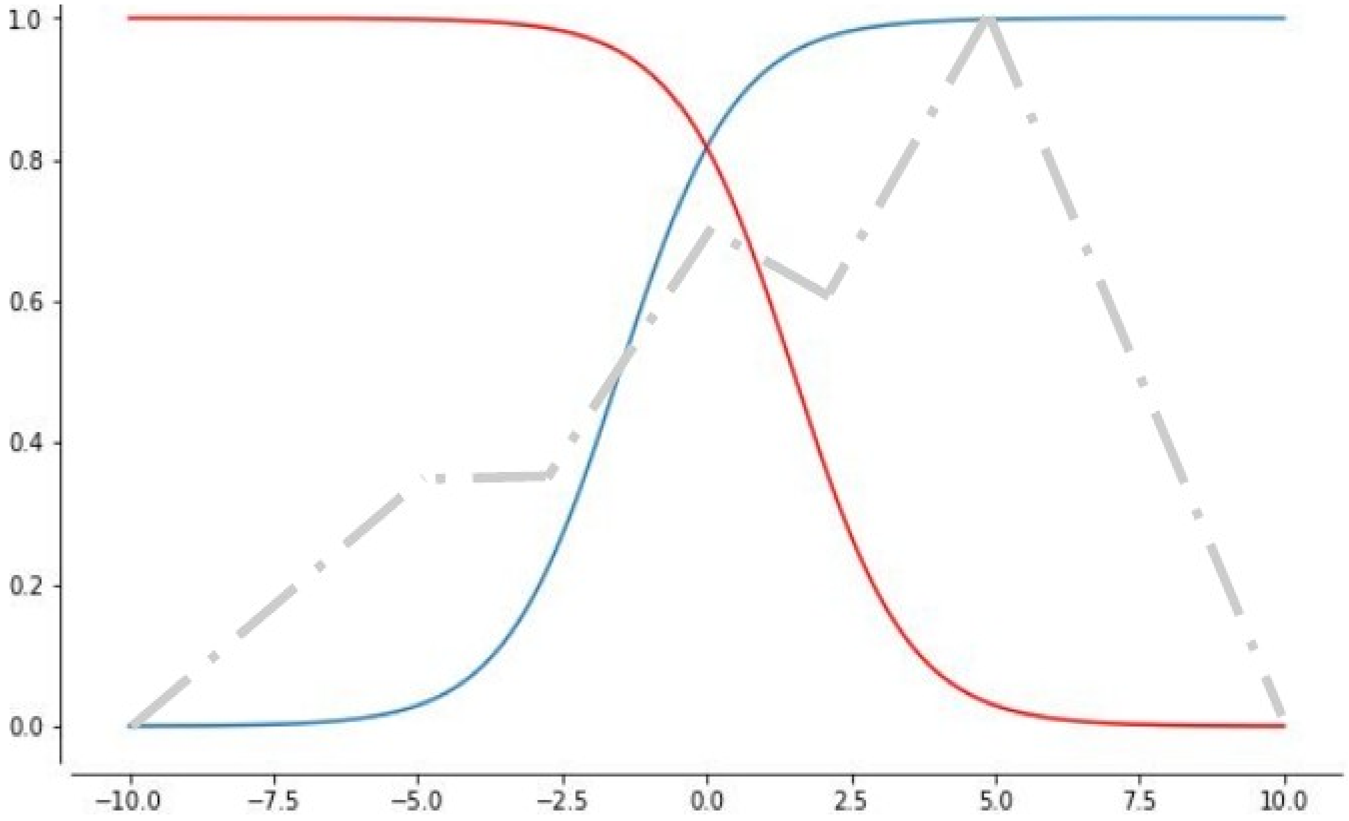


University of Antwerp

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

- **An intuition on how it works**

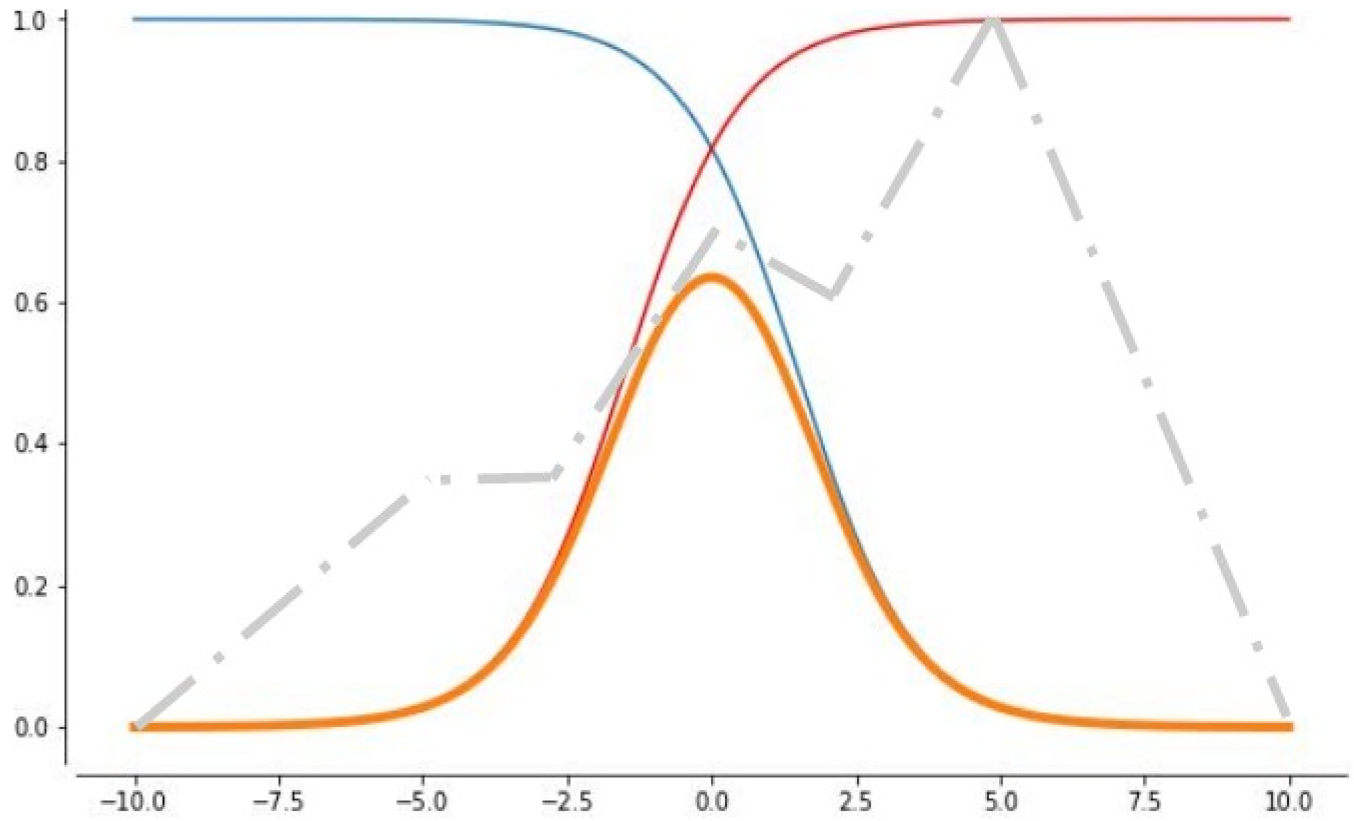


University of Antwerp

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

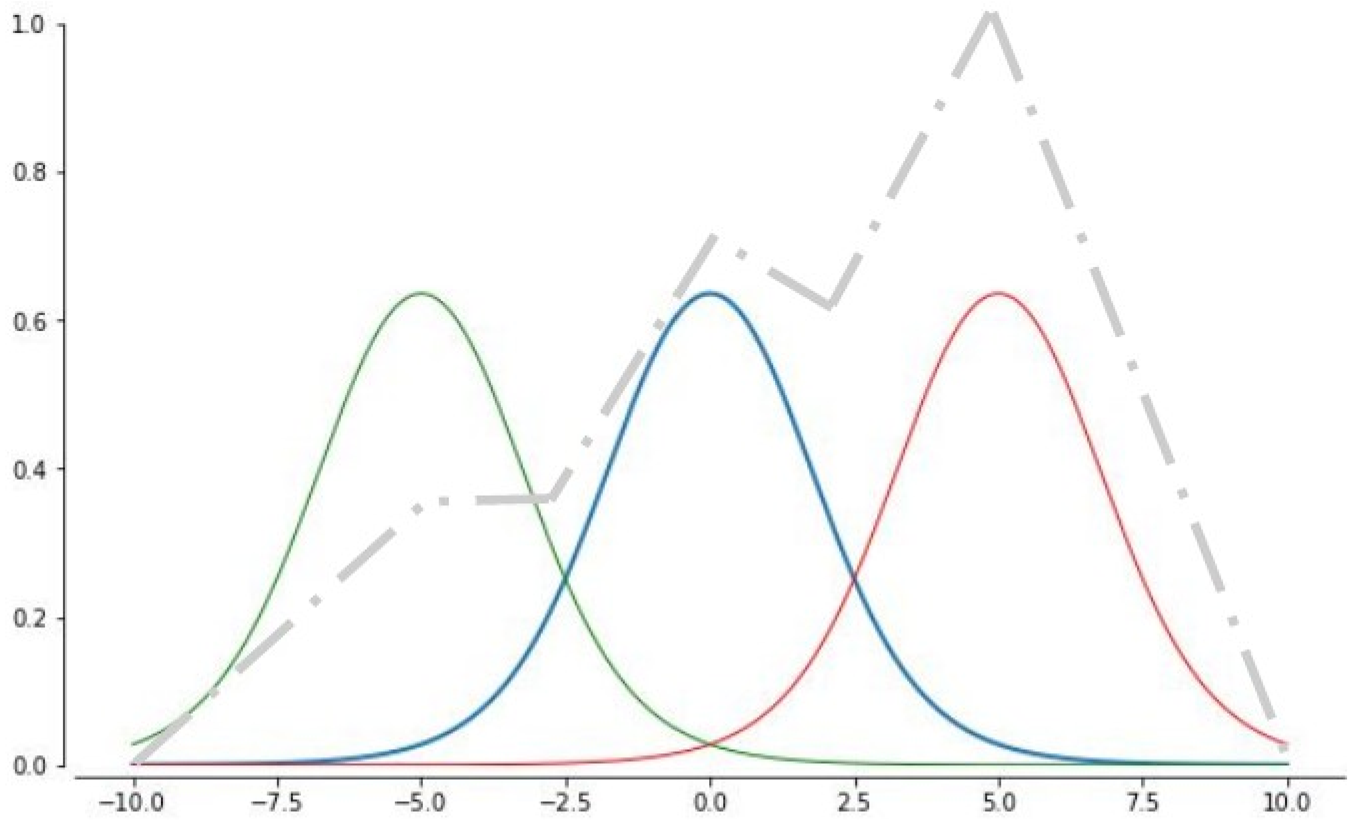
- **An intuition on how it works**



Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

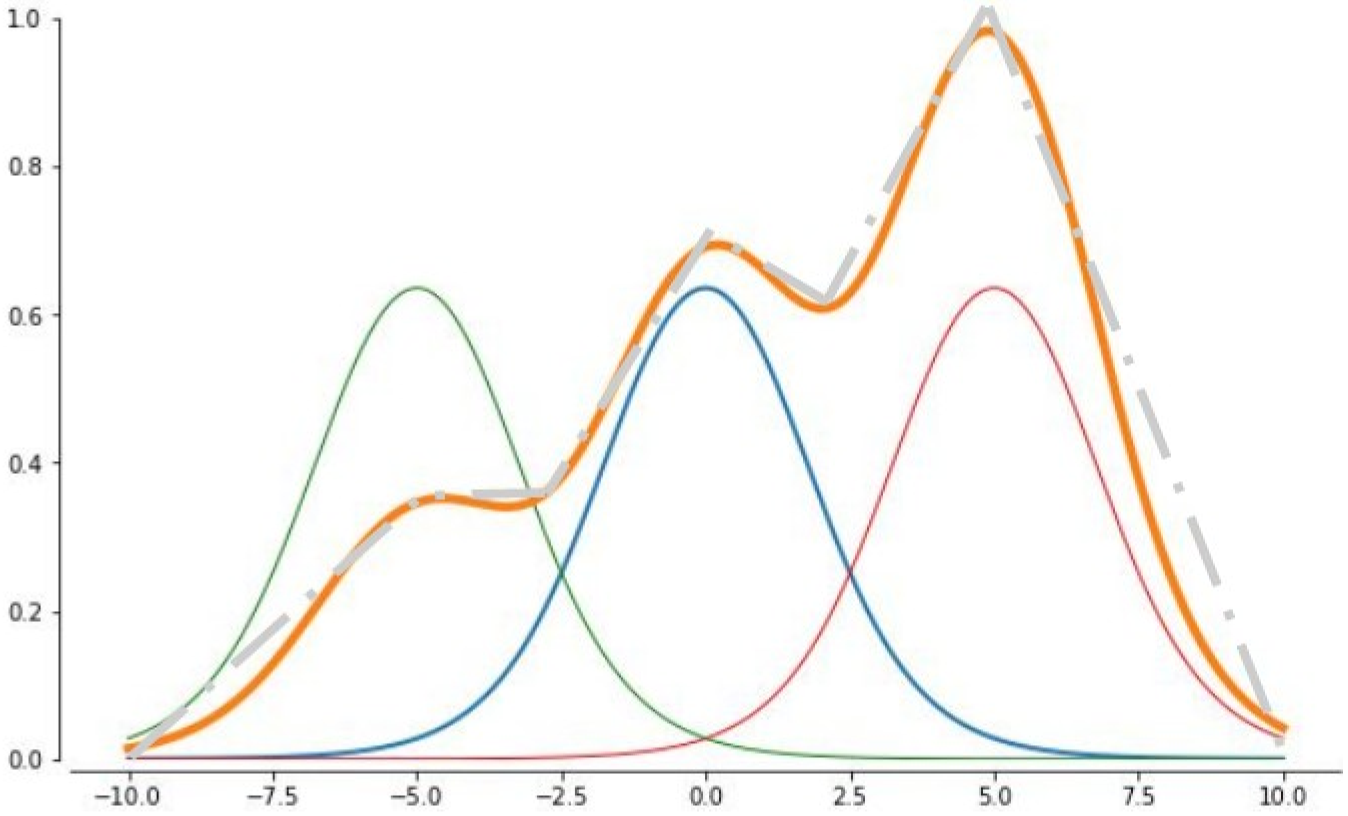
- **An intuition on how it works**



Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

- **An intuition on how it works**



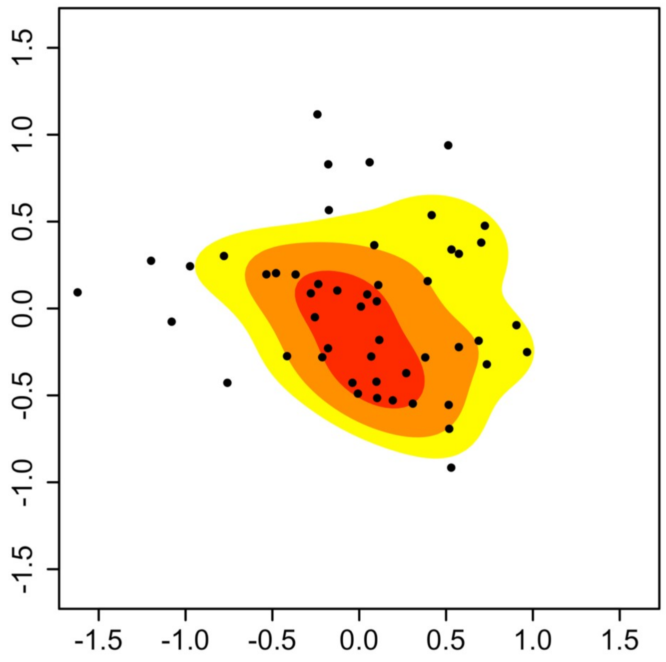
Nice, but …

What happens in high-dimensional spaces?

Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

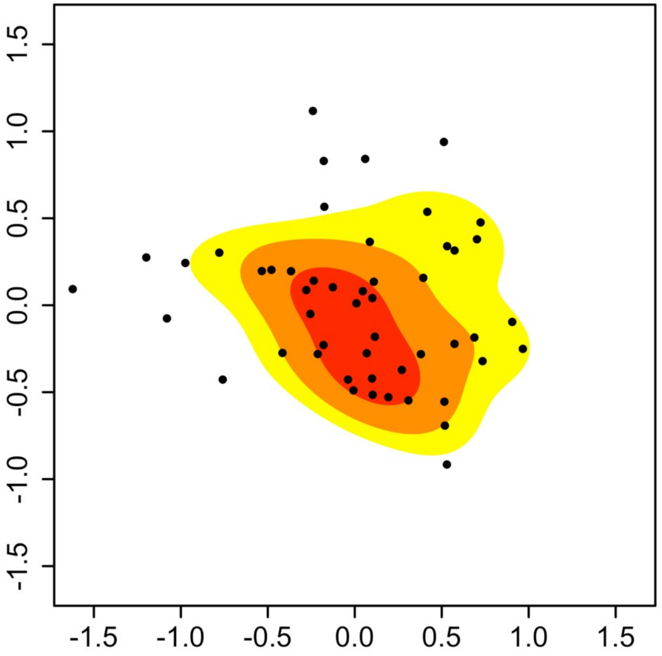
- **An intuition on how it works –** High-Dimensional Spaces

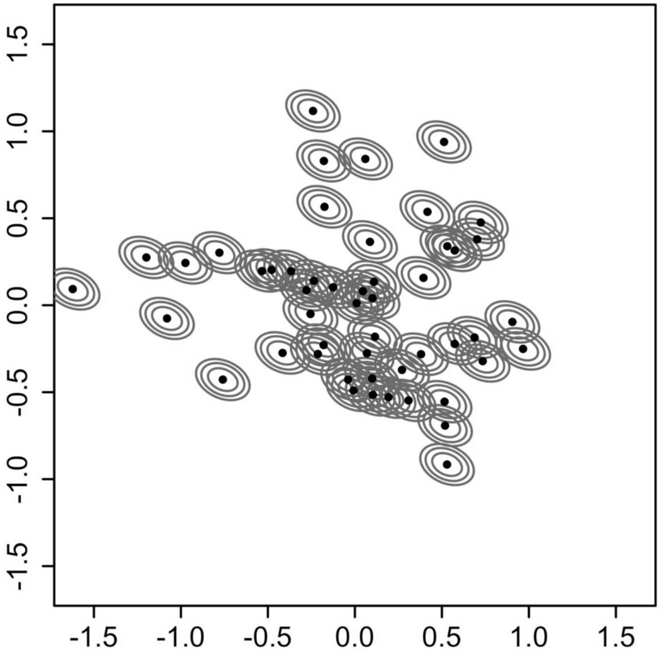


Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

- **An intuition on how it works –** High-Dimensional Spaces

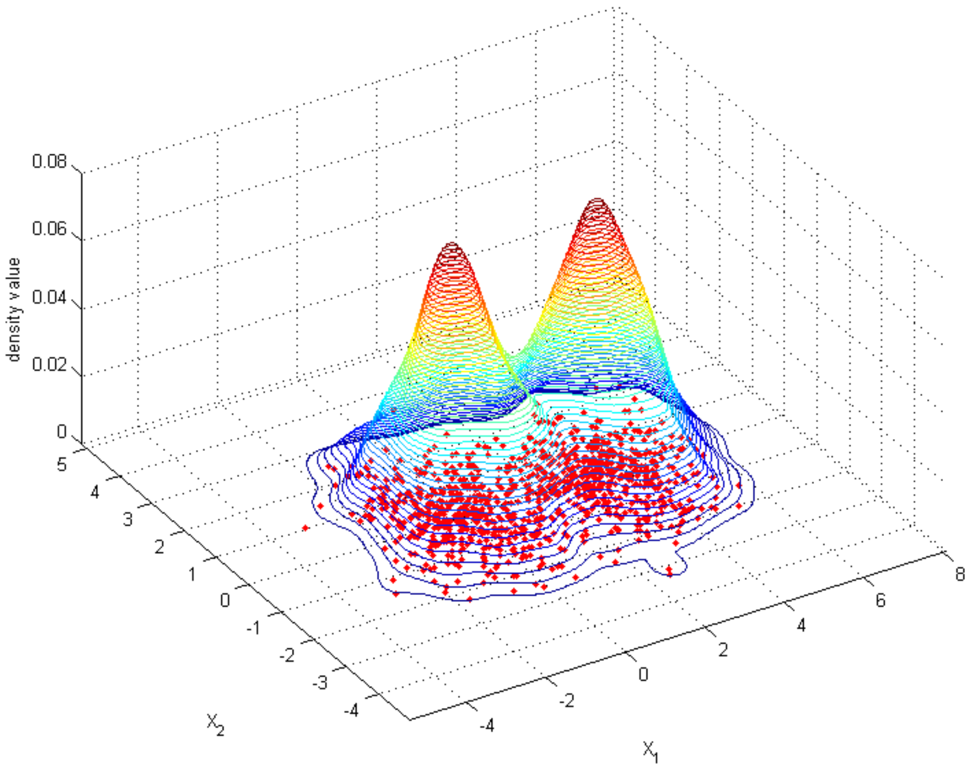




Deep Neural Networks

Universal Approximation Theorem [Cybenko, 1989]

- **An intuition on how it works –** High-Dimensional Spaces



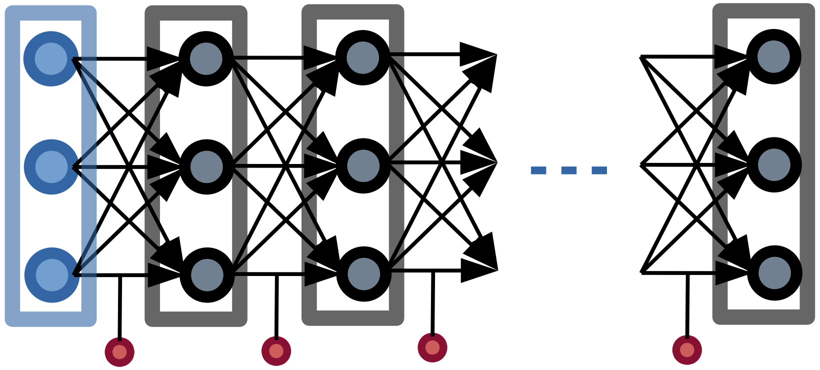
Ok, but …

Why deeper rather than wider?

Deep Neural Networks

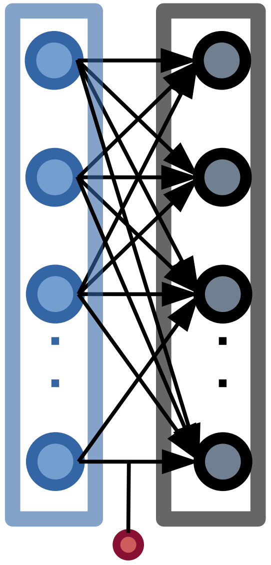
Deeper VS. Wider Architectures

Deeper



VS

Wider



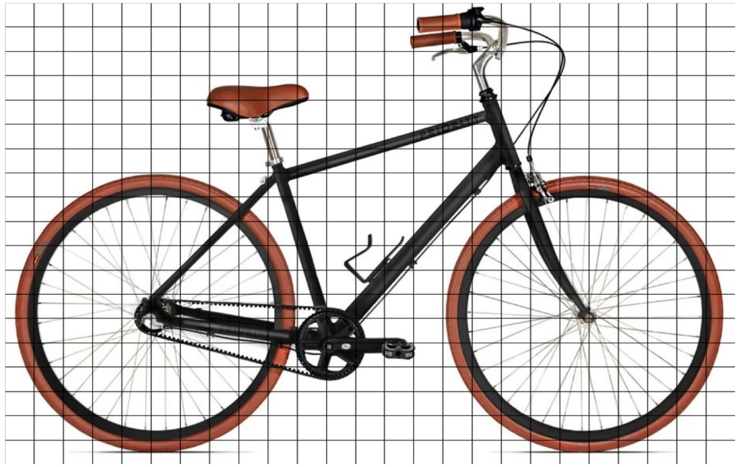
Growth of the partitioning space

- Exponential by depth
- Polynomial by width

More insights in [Motúfar et al, NeurIPS 2014]

Deep Neural Networks

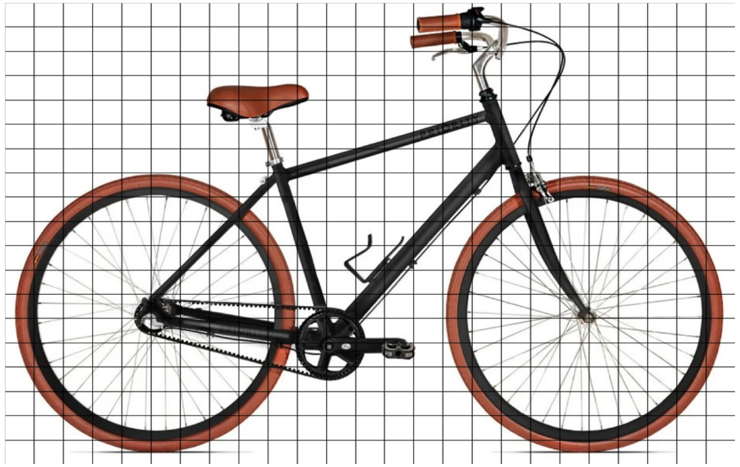
Deeper VS. Wider Architectures



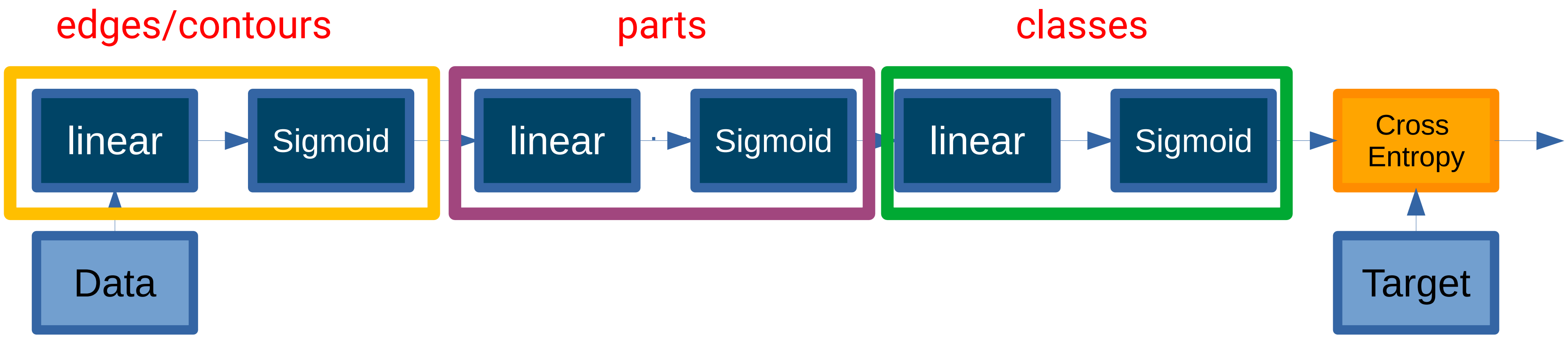
← Learning how to recognize a bicycle

Deep Neural Networks

Deeper VS. Wider Architectures



← Learning how to recognize a bicycle

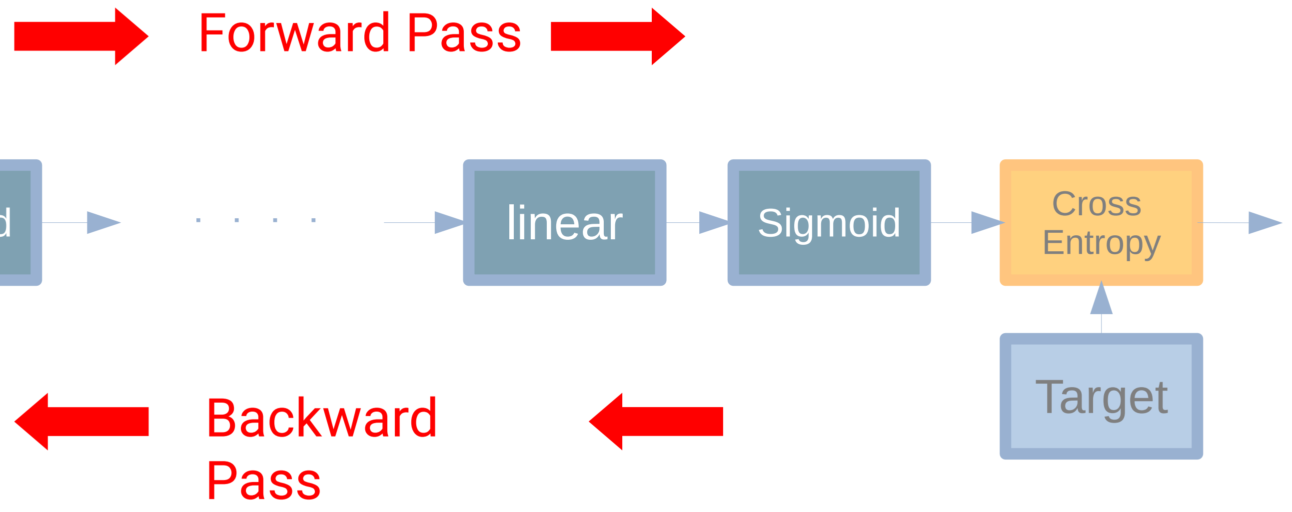


Learning

[with few layers]

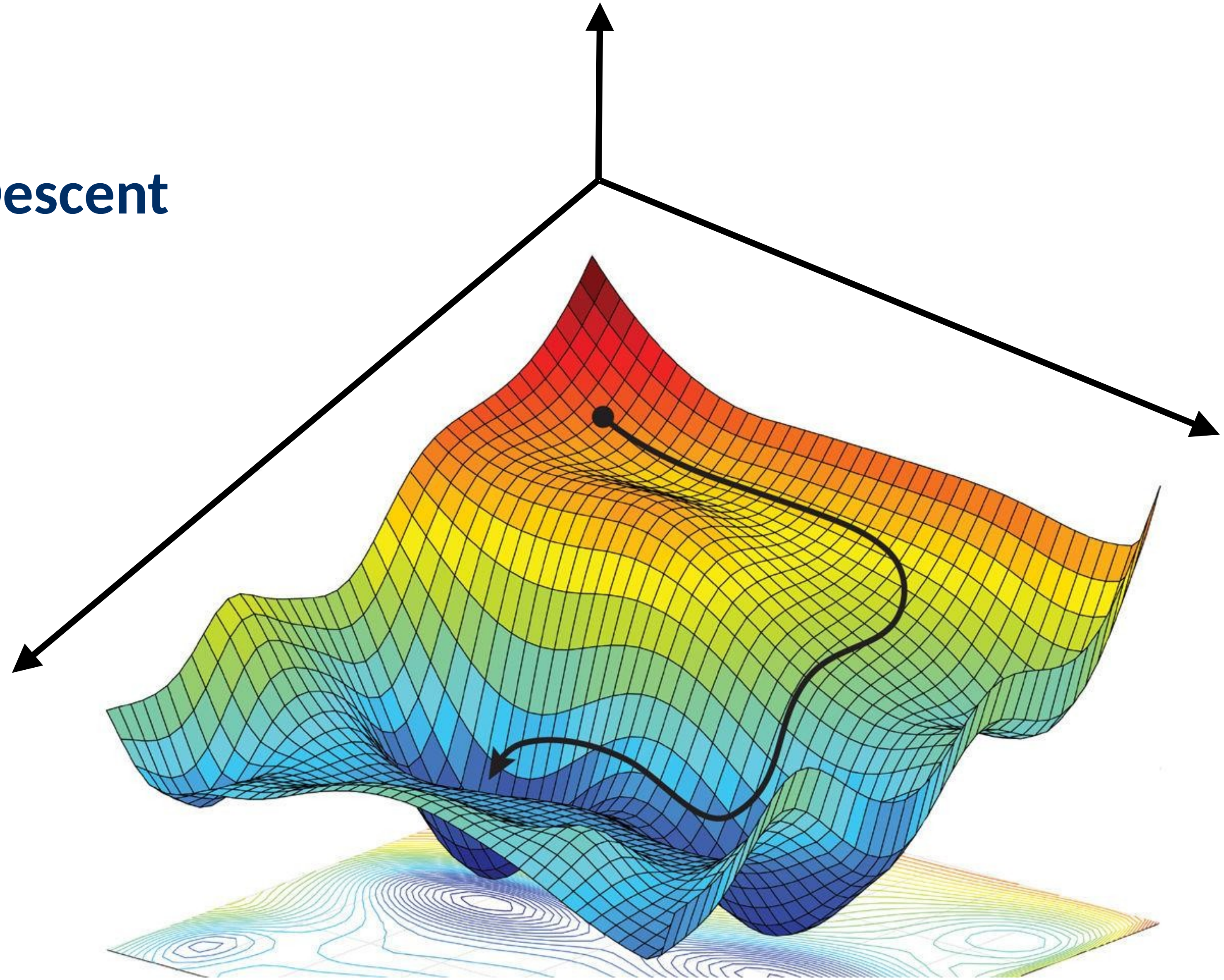
Learning

Process Overview



Learning

Gradient Descent



[doi:10.1126/science.aau0577]

Learning

Gradient Descent – Algebraic Foundations

$$y = f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$$

$$\frac{\partial y}{\partial x} = \nabla_x f(x) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right]$$

Gradient

Learning

Gradient Descent – Algebraic Foundations

$$y = f(x) : \mathbb{R}^d \rightarrow \mathbb{R}$$

$$\frac{\partial y}{\partial x} = \nabla_x f(x) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right]$$

Gradient

$$\frac{\partial y}{\partial x} = \mathbf{J}_x f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial x_1} & \cdots & \frac{\partial f_k}{\partial x_d} \end{bmatrix}$$

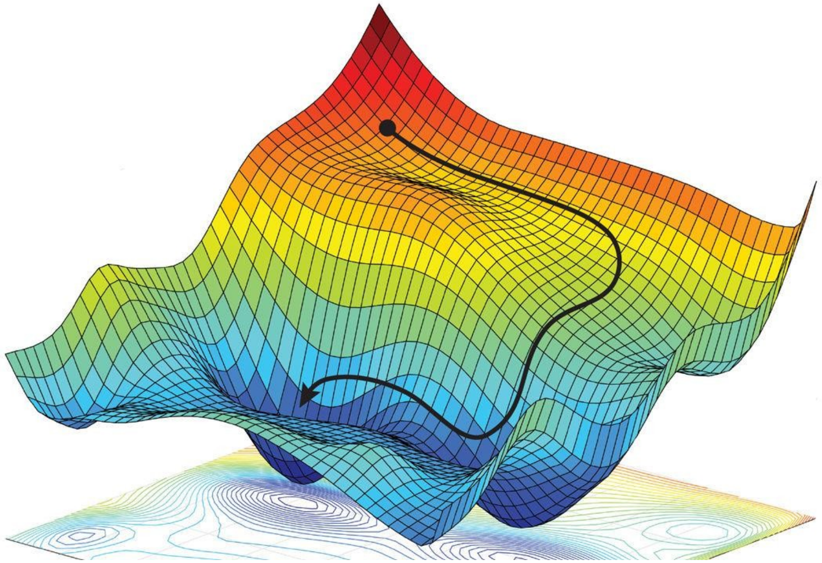
Jacobian

multi-dimensional generalization

Learning

Gradient Descent

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta L(\theta_t)$$



[doi:10.1126/science.aau0577]

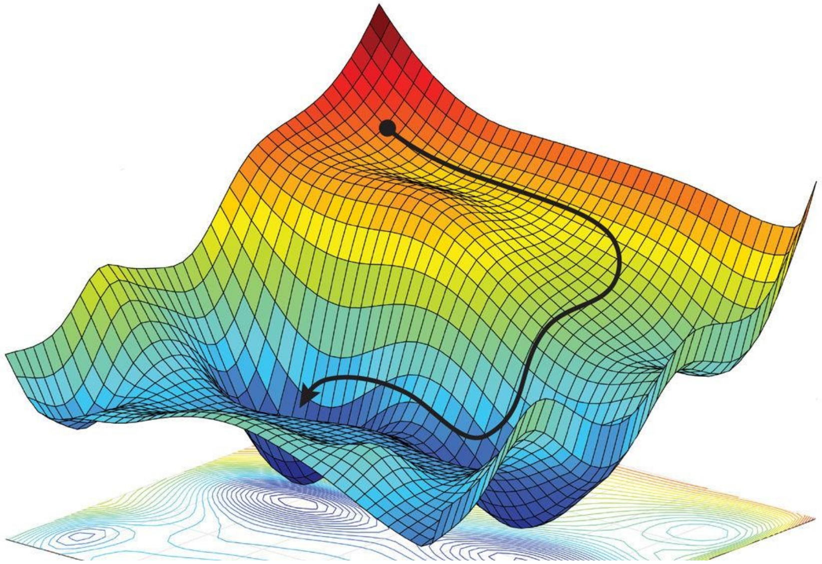
Learning

Gradient Descent

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta L(\theta_t)$$

$$\nabla_\theta L(\theta_t) = \nabla_\theta \sum_i l(f(x^{(i)}, \theta_t), y^{(i)})$$

$$= \sum_i \nabla_\theta l(f(x^{(i)}, \theta_t), y^{(i)})$$



[doi:10.1126/science.aau0577]

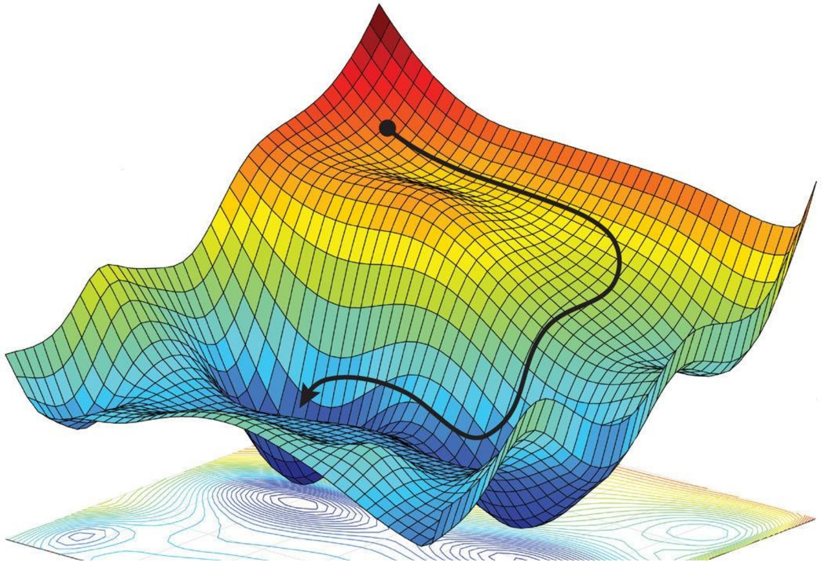
Learning

Gradient Descent

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta L(\theta_t)$$

$$\nabla_\theta L(\theta_t) = \nabla_\theta \sum_i l(f(x^{(i)}, \theta_t), y^{(i)})$$

$$= \sum_i \nabla_\theta l(f(x^{(i)}, \theta_t), y^{(i)})$$



[doi:10.1126/science.aau0577]

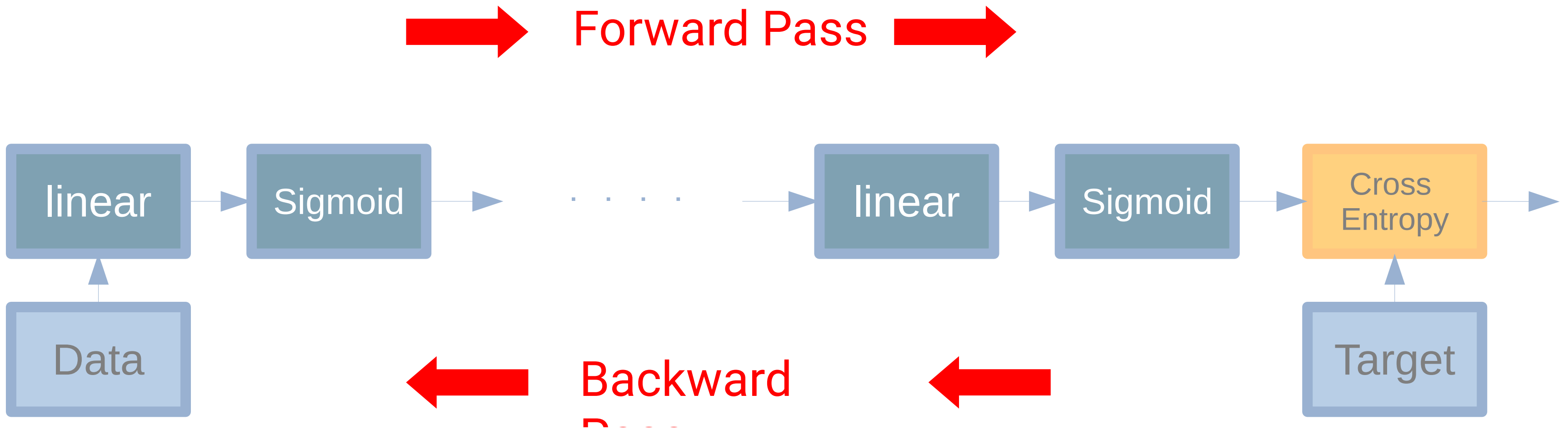
Characteristics:

- Works for any smooth function
- Less guarantees for some non-smooth targets
- Converges to local optimum
- Critical effect of the *Learning rate*

Learning

Process Overview

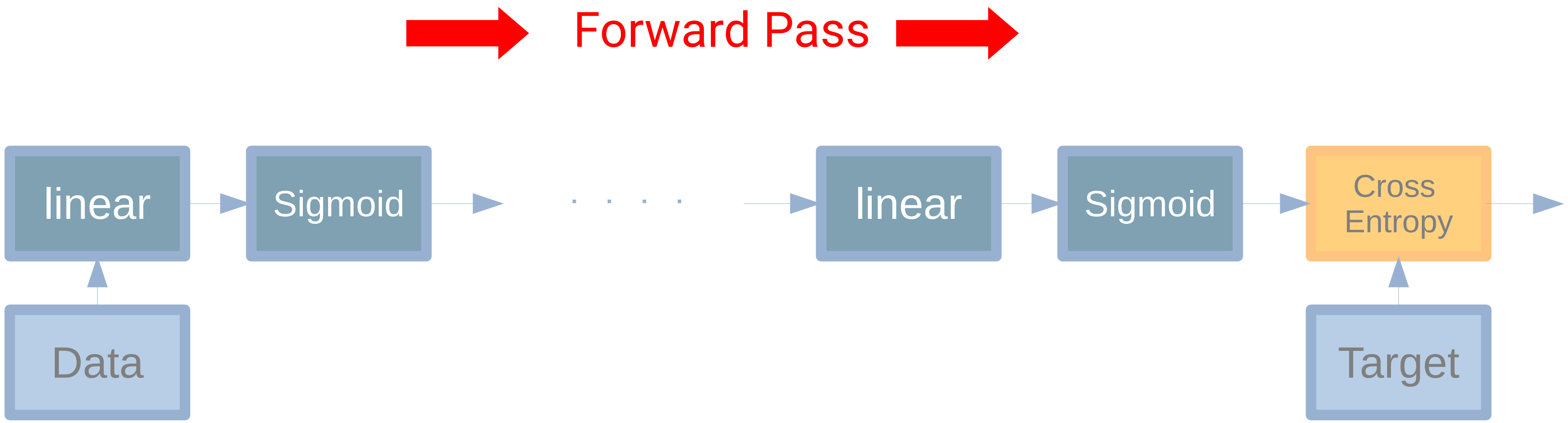
$$f(x)$$



$$\mathbf{J}_x f(x)$$

Learning

Back-Propagation Algorithm

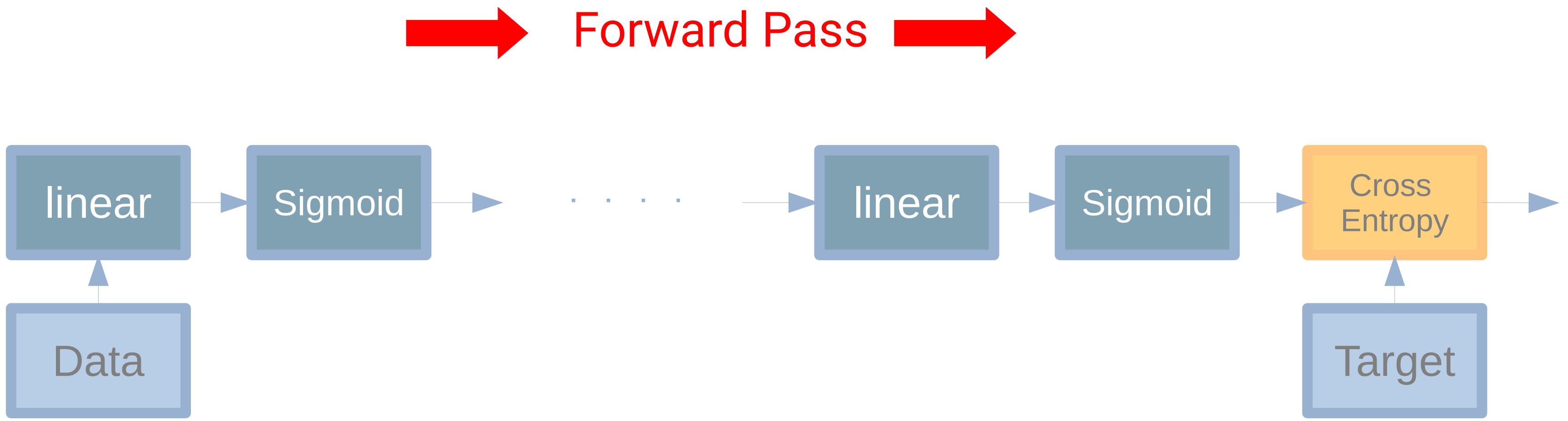


$$f_1(x)$$

Looking the forward pass as a composition

Learning

Back-Propagation Algorithm

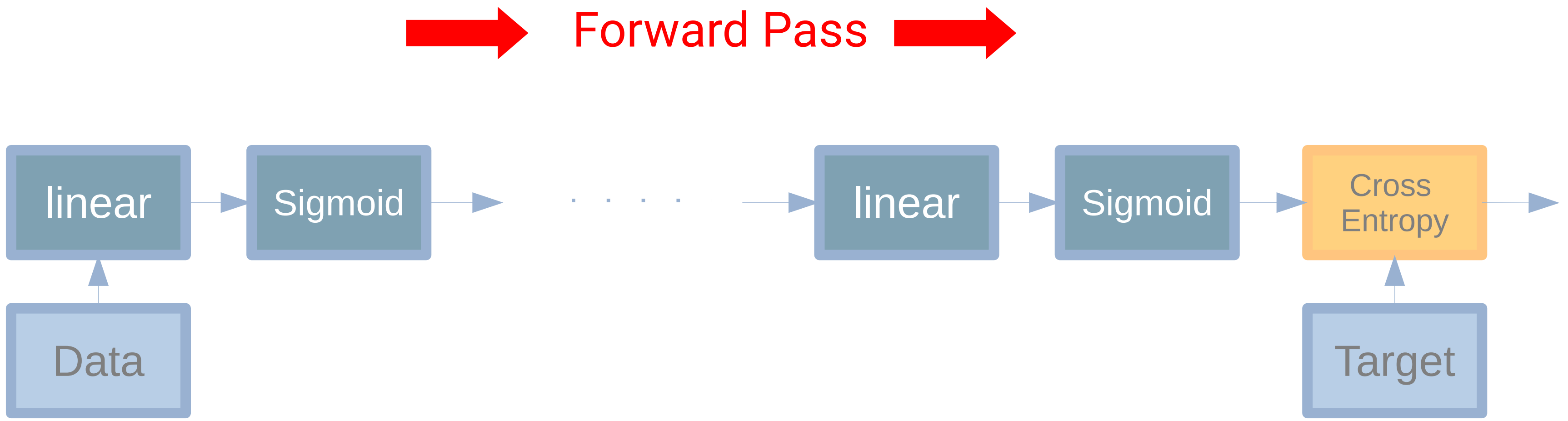


$$f_2(\ f_1(x)\ )$$

Looking the forward pass as a composition

Learning

Back-Propagation Algorithm

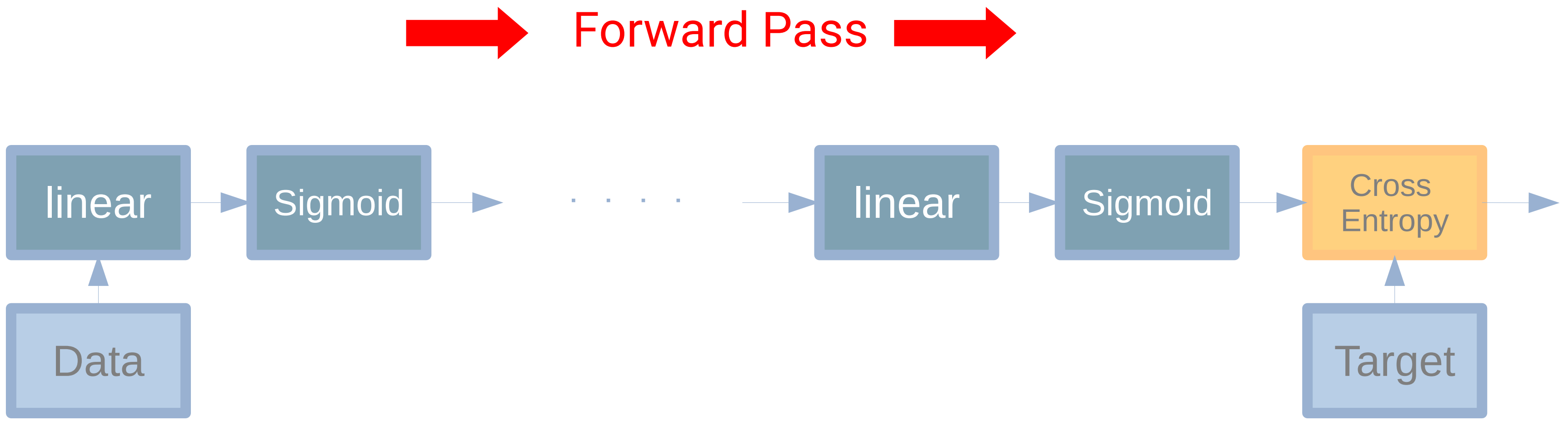


$$f_{k-1}(\ f_{k-2}(\ldots f_2(\ f_1(x)\ )\ )\ )$$

Looking the forward pass as a composition

Learning

Back-Propagation Algorithm

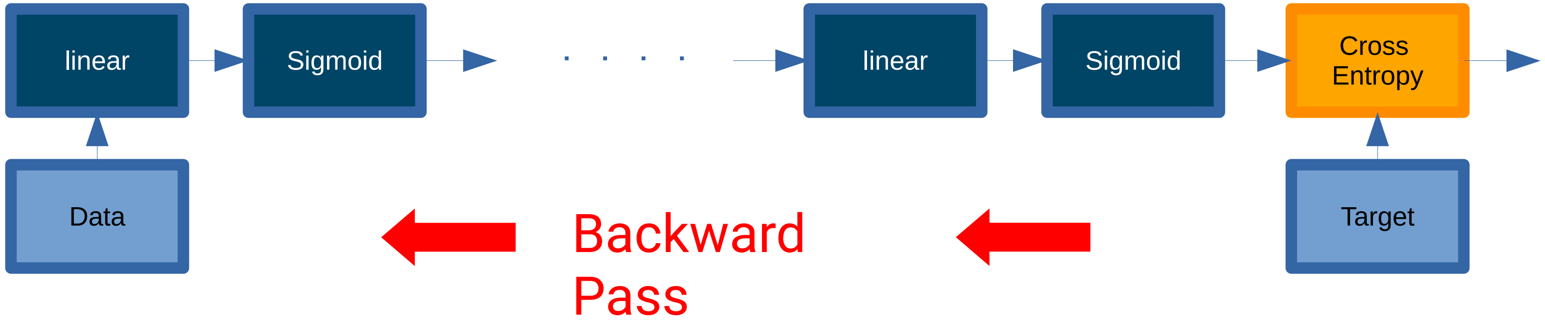


$$y = f_k(\ f_{k-1}(\ f_{k-2}(\ldots f_2(\ f_1(x)\ )\ )\ )\ )$$

Looking the forward pass as a composition

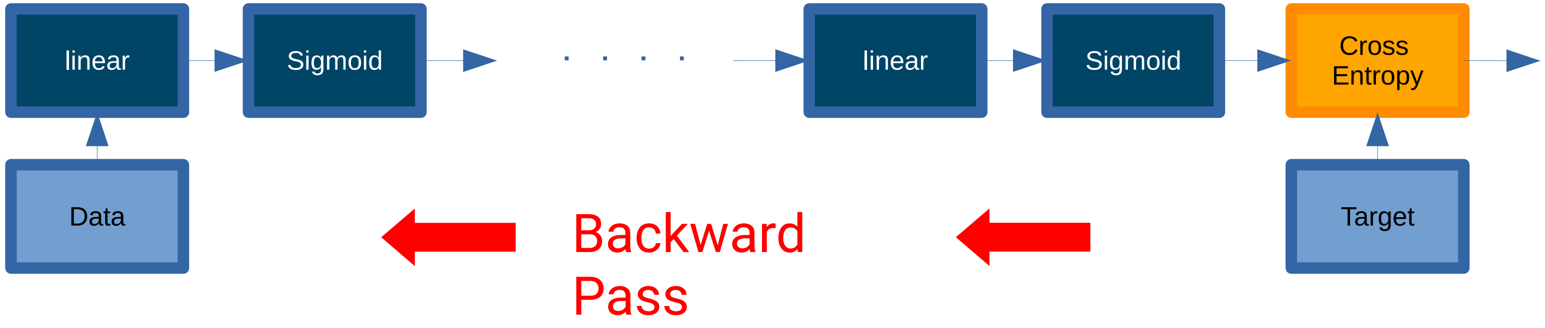
Learning

Back-Propagation Algorithm



Learning

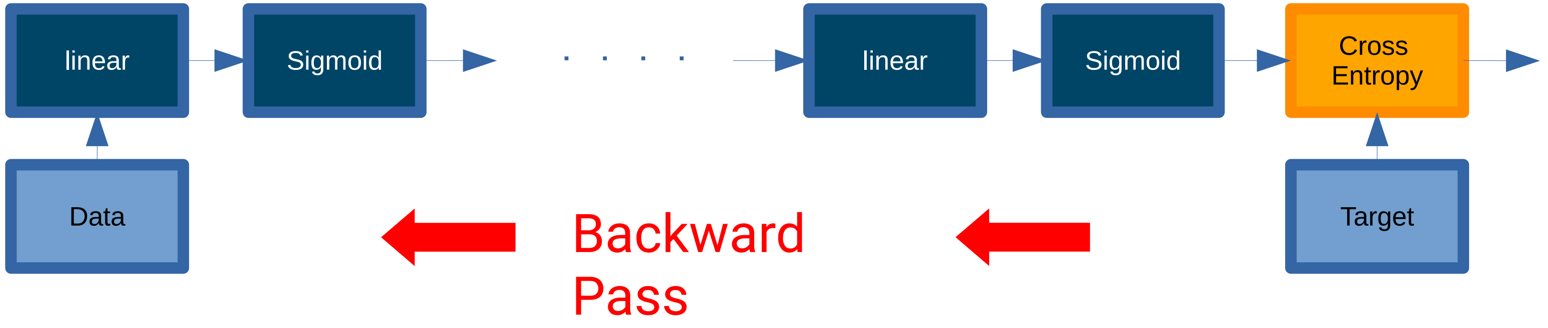
Back-Propagation Algorithm



$$y = f(g(x))\,\frac{\partial y}{\partial x}$$

Learning

Back-Propagation Algorithm

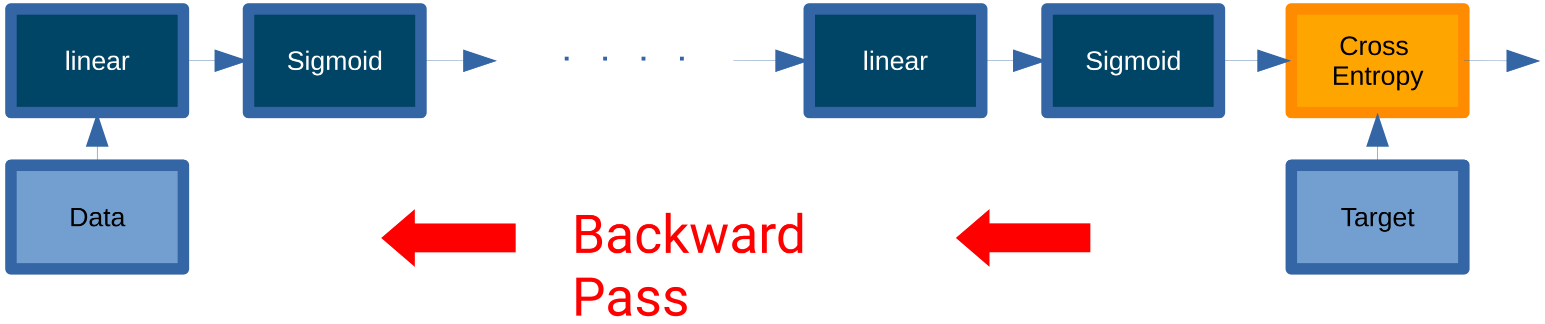


$$y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial g}\frac{\partial g}{\partial x}$$

$$\boxed{\frac{dL}{dw} = \frac{dL}{dy} \cdot \frac{dy}{dz} \cdot \frac{dz}{dw}}$$

Learning

Back-Propagation Algorithm



$$y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial g}\frac{\partial g}{\partial x}$$

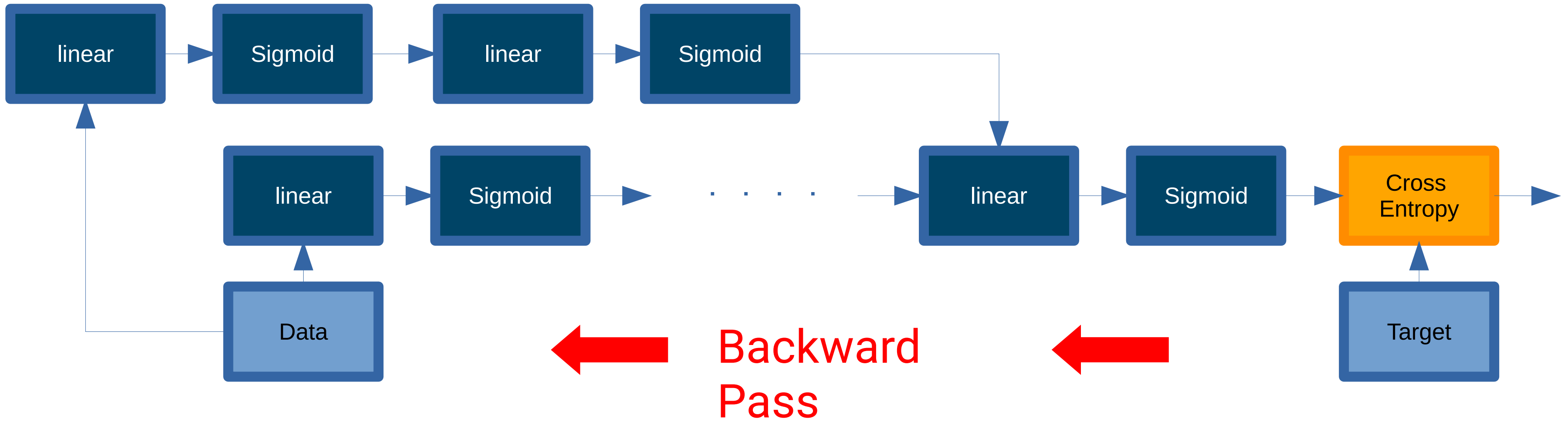
$$\boxed{\frac{dL}{dw} = \frac{dL}{dy} \cdot \frac{dy}{dz} \cdot \frac{dz}{dw}}$$

Characteristics:

- Chain rule for derivative computation
- Computations can be re-used (makes is linear [faster], otherwise quadratic)

Learning

Back-Propagation Algorithm



$$y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial g} \frac{\partial g}{\partial x}$$

$$y = f(g(X)) \quad \frac{\partial y}{\partial X} = \sum_{i=1}^{m} \frac{\partial y}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial X}$$

Characteristics:

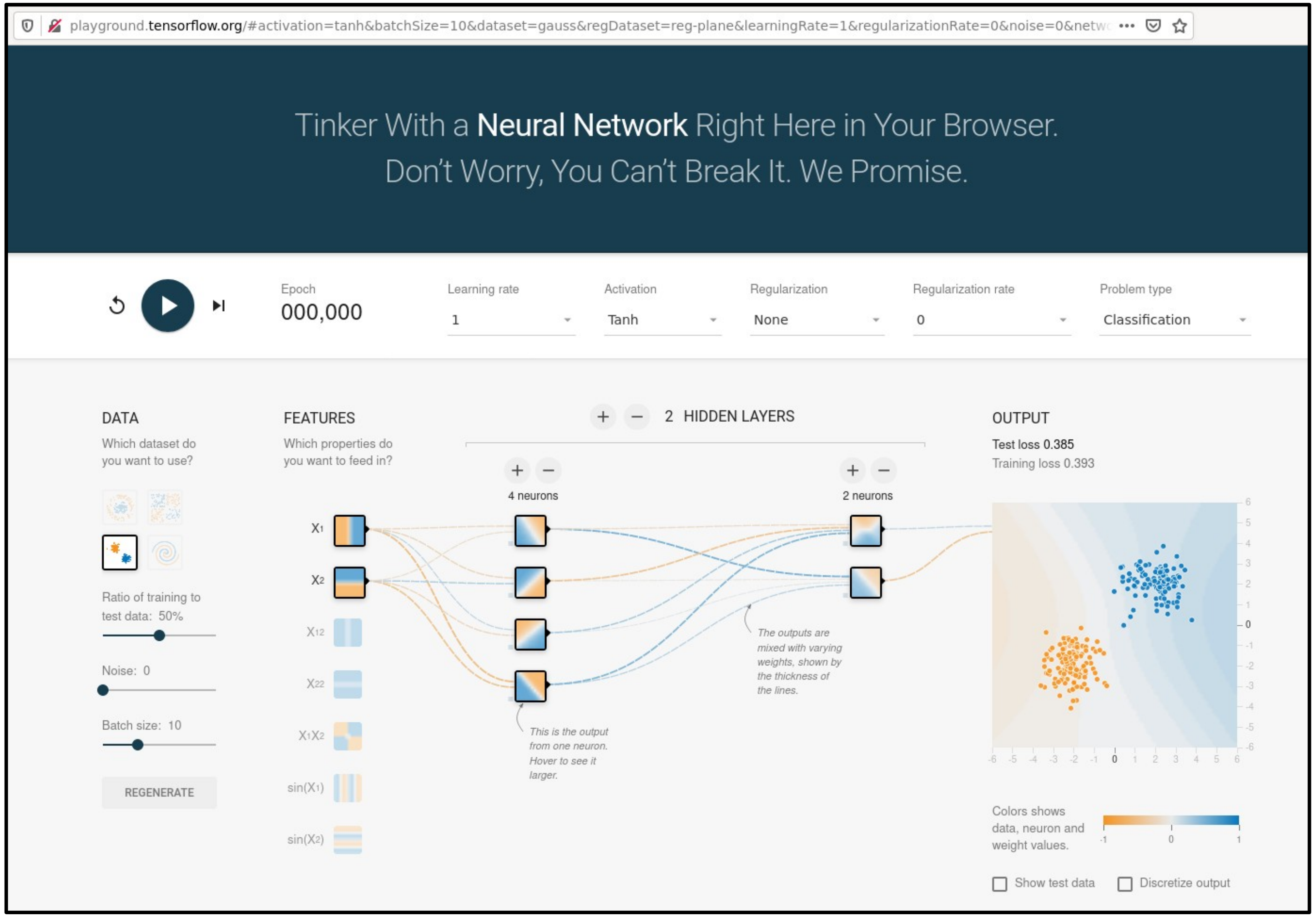
- Chain rule for derivative computation
- Computations can be re-used (makes is linear [faster], otherwise quadratic)

wait, …

That’s it?

From Shallow to Deep Neural Networks

Some Extra Practice – http://playground.tensorflow.org



Summarizing

[Finally :D]

Summarizing

- **A From Neurons to Networks**
  - Neurons → Layers → Networks

Summarizing

- **A From Neurons to Networks**
  - Neurons → Layers → Networks

- **Power through Composition**
  - It is not about a single unit but their combination
  - Capable of approximating any function producing a single real value as output
  - Better deeper (exponential) than wider (polynomial) architectures

Summarizing

- **A From Neurons to Networks**
  - Neurons → Layers → Networks

- **Power through Composition**
  - It is not about a single unit but their combination
  - Capable of approximating any function producing a single real value as output
  - Better deeper (exponential) than wider (polynomial) architectures

- **Some Enablers**
  - Efficient algorithmic computations
  - Use of dedicated hardware

Pay Attention...

[one last tip for today]

Pay attention to…

- Everything

References

Universal Approximation Capabilities of Deep Neural Networks

- G. Cybenko, Approximation by superpositions of a sigmoidal function,Math. Control Signals Systems,2 (1989), 303–314.
  https://link.springer.com/article/10.1007/BF02134016
- K. Hornik Approximation Capabilities of Muitilayer Feedforward Networks
  https://web.njit.edu/~usman/courses/cs675_spring20/hornik-nn-1991.pdf

Deep VS. Wide architectures

- Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, Yoshua Bengio, On the Number of Linear Regions of Deep Neural Networks. NeurIPS 2014
  https://papers.nips.cc/paper/2014/file/109d2dd3608f669ca17920c511c2a41e-Paper.pdf

ReLU

- R H Hahnloser 1 , R Sarpeshkar, M A Mahowald, R J Douglas, H S Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 2000
  https://pubmed.ncbi.nlm.nih.gov/10879535/

Questions?

From Shallow to Deep Neural Networks

[Building More Complex Models]

José Oramas