WELCOME TO THE

UCL x DeepMind lecture series

In this lecture series, leading research scientists from leading AI research lab, DeepMind, will give 12 lectures on an exciting selection of topics in Deep Learning, ranging from the fundamentals of training neural networks via advanced ideas around memory, attention, and generative modelling to the important topic of responsible innovation.

Please join us for a deep dive lecture series into Deep Learning!

#UCLxDeepMind

General information

Exits:

At the back, the way you came in

Wifi:

UCL guest

TODAY'S SPEAKER

Wojciech Czarnecki

Wojciech Czarnecki is a Research Scientist at DeepMind. He obtained his phd from the Jagiellonian University in Cracow, during which he worked on the intersection of machine learning, information theory and cheminformatics. Since joining DeepMind in 2016, Wojciech has been mainly working on deep reinforcement learning, with a focus on multi-agent systems, such as recent Capture the Flag project or AlphaStar, the first AI to reach the highest league of human players in a widespread professional esport without simplification of the game.

TODAY'S LECTURE

Neural Networks Foundations

Neural networks are the models responsible for the deep learning revolution since 2006, but their foundations go back as far as to the 1960s. In this lecture we will go through the basics of how these models operate, learn and solve problems. We will also set various terminology/naming conventions to prepare attendees for further, more advanced talks. Finally, we will briefly touch upon more research oriented directions of neural network design and development.

DeepMind

Neural Networks Foundations

Wojciech Czarnecki

UCL x DeepMind Lectures

Plan for this Lecture

What is not covered in this lecture

01

"Old school"

- (Restricted) Boltzmann Machines
- Deep Belief Networks
- Hopfield Networks
- Self Organising Maps

02

Biologically plausible

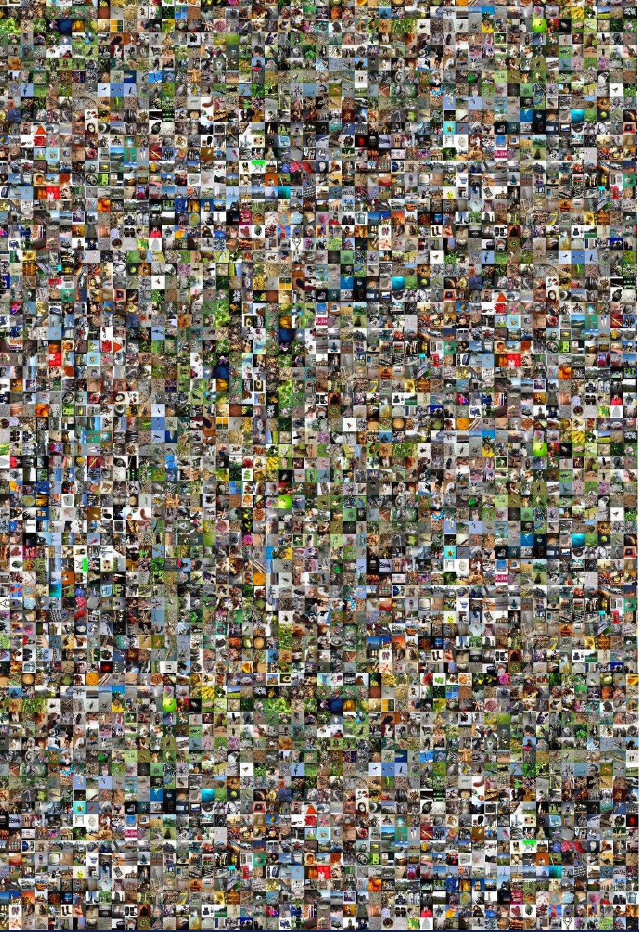
- Spiking networks
- Physical Simulators

03

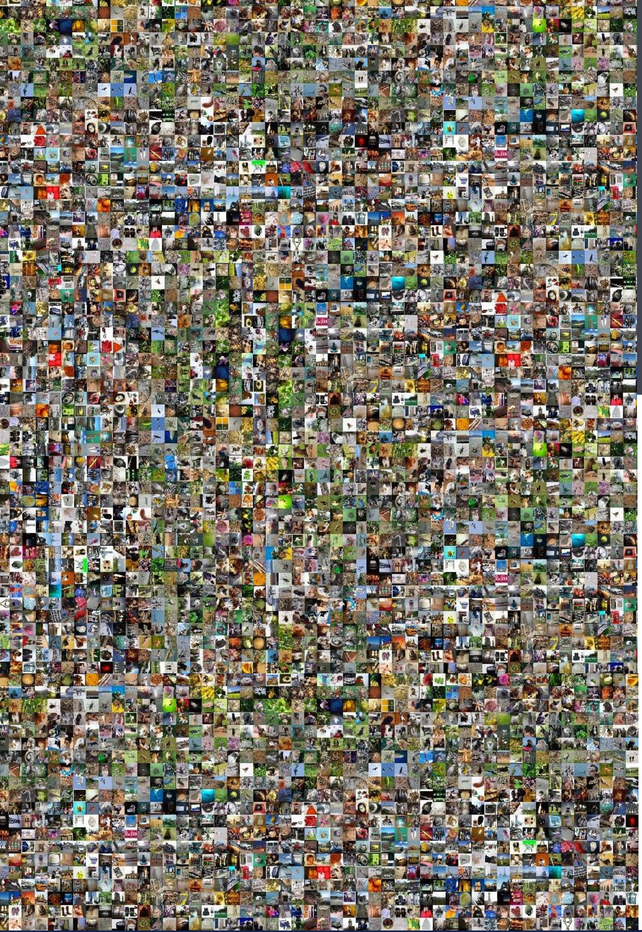
Other

- Capsules
- Graph networks
- Neural Differential Equations
- Convolutional Networks
- Recurrent Neural Networks

1 Overview



Computer Vision



SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

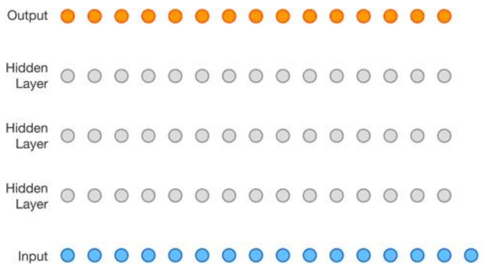
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

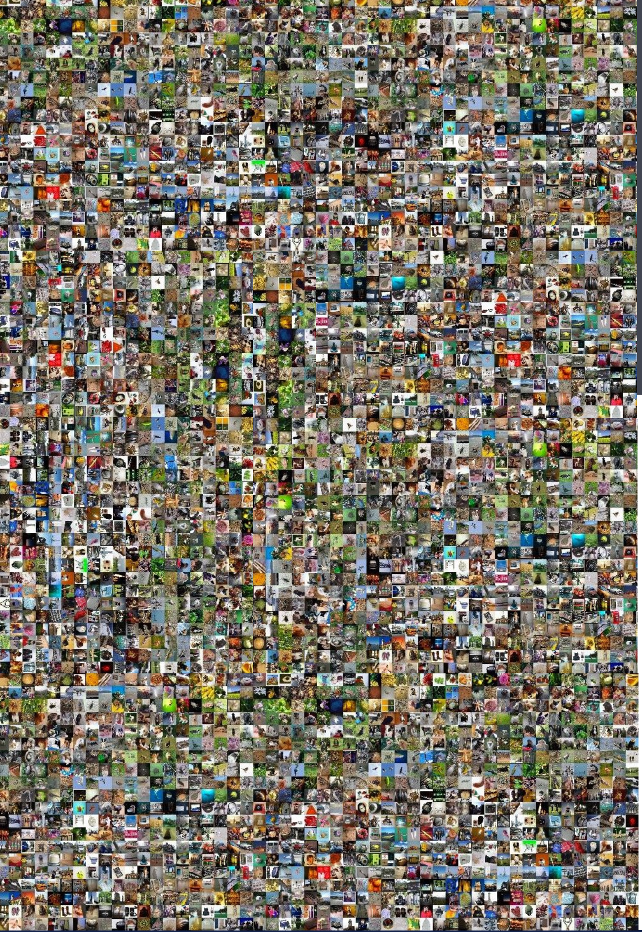
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.



Computer Vision

Text and Speech



SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

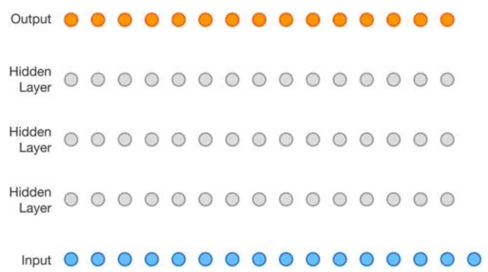
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

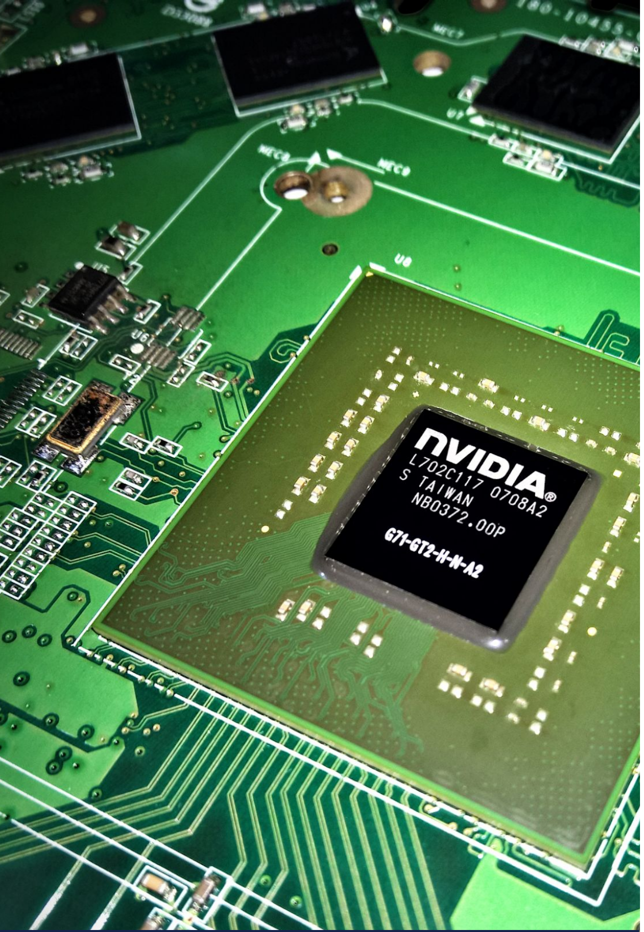
Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.



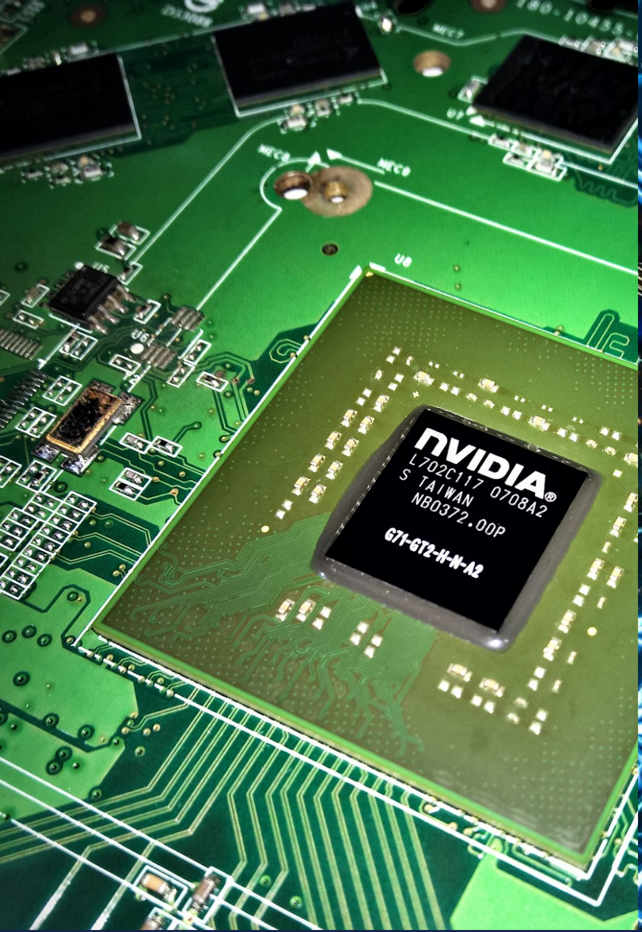
Computer Vision

Text and Speech

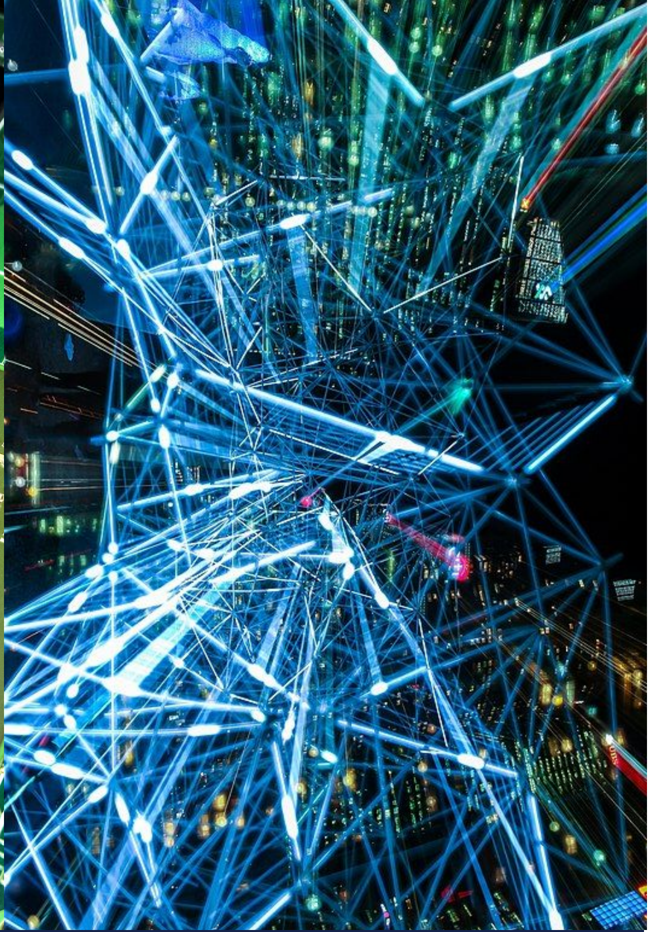
Control



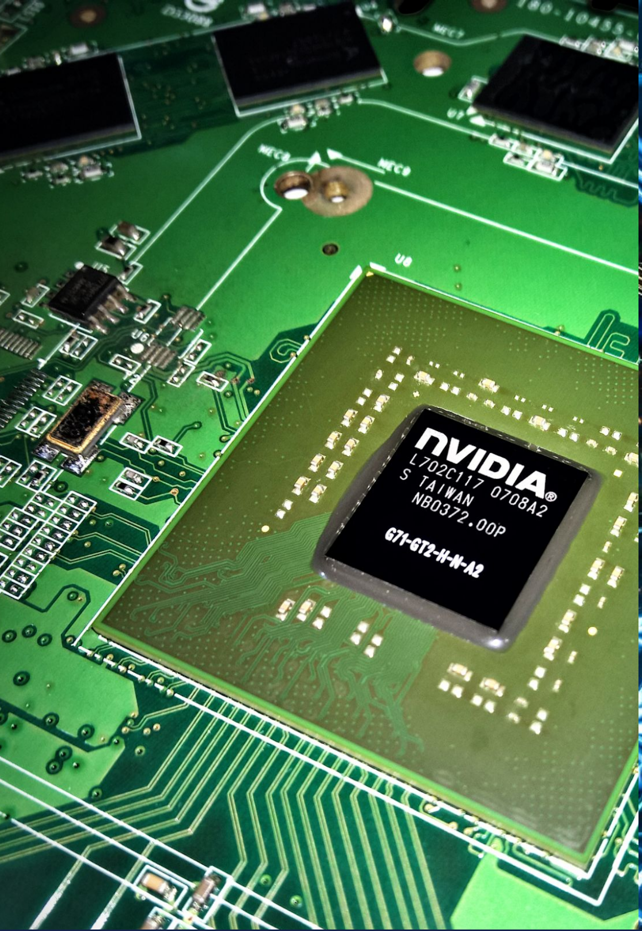
Compute



Compute



Data

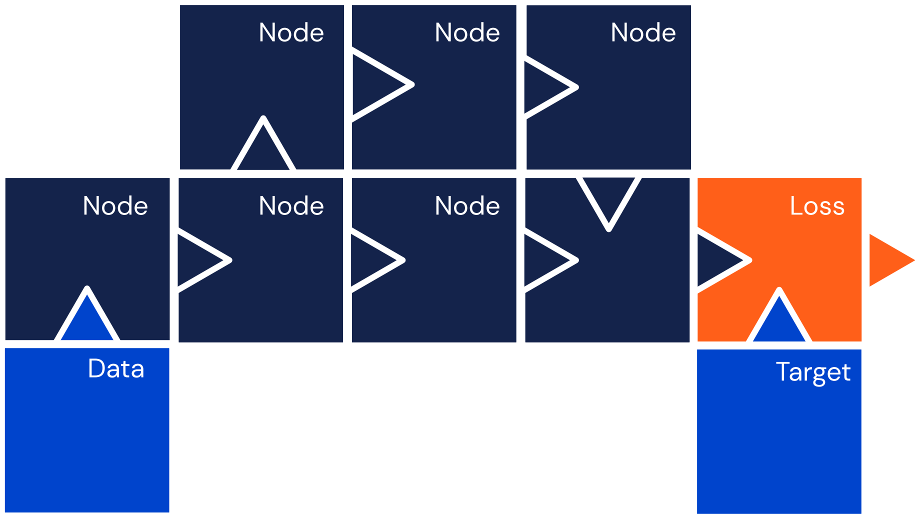


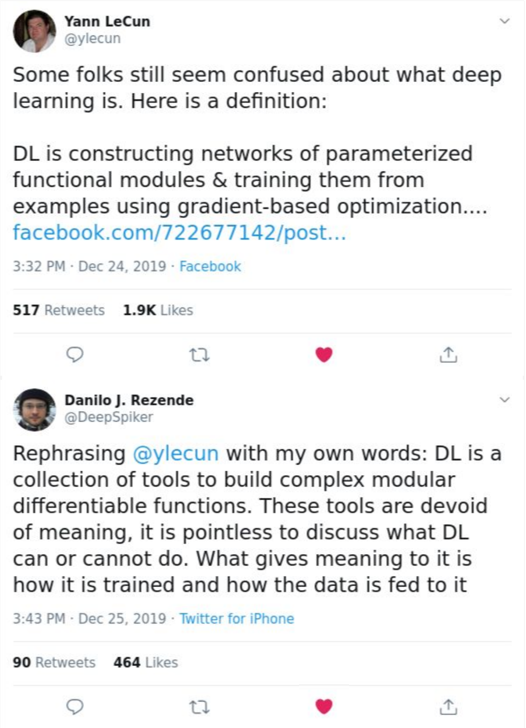
Compute

Data

Modularity

The deep learning puzzle





Yann LeCun
@ylecun

Some folks still seem confused about what deep learning is. Here is a definition:

DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization.... facebook.com/722677142/post...

3:32 PM · Dec 24, 2019 · Facebook

517 Retweets **1.9K** Likes

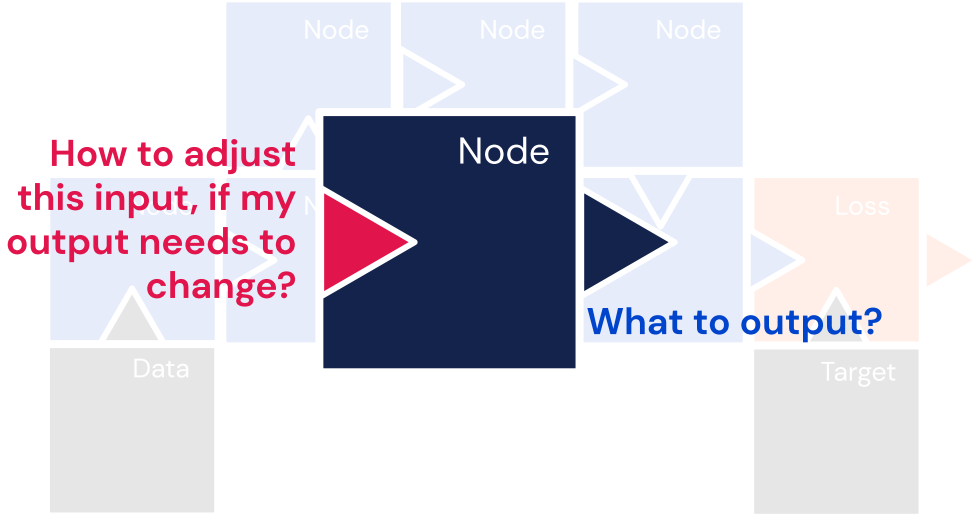
Danilo J. Rezende
@DeepSpiker

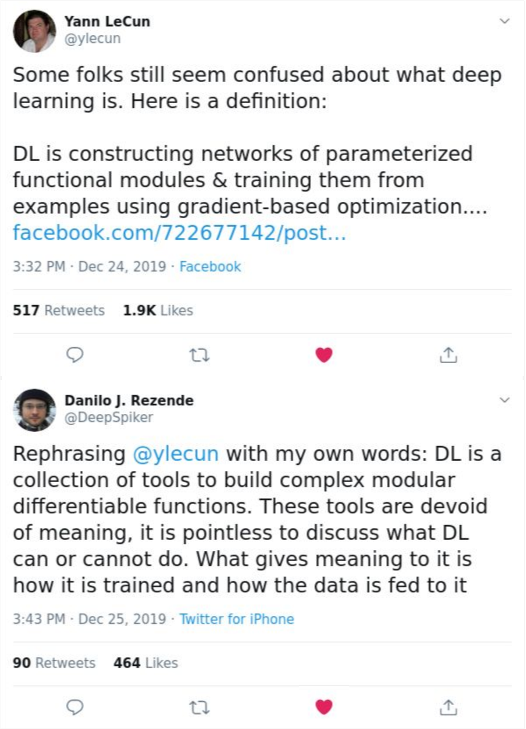
Rephrasing @ylecun with my own words: DL is a collection of tools to build complex modular differentiable functions. These tools are devoid of meaning, it is pointless to discuss what DL can or cannot do. What gives meaning to it is how it is trained and how the data is fed to it

3:43 PM · Dec 25, 2019 · Twitter for iPhone

90 Retweets **464** Likes

The deep learning puzzle





Yann LeCun @ylecun

Some folks still seem confused about what deep learning is. Here is a definition:

DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization.... facebook.com/722677142/post...

3:32 PM · Dec 24, 2019 · Facebook

517 Retweets **1.9K** Likes

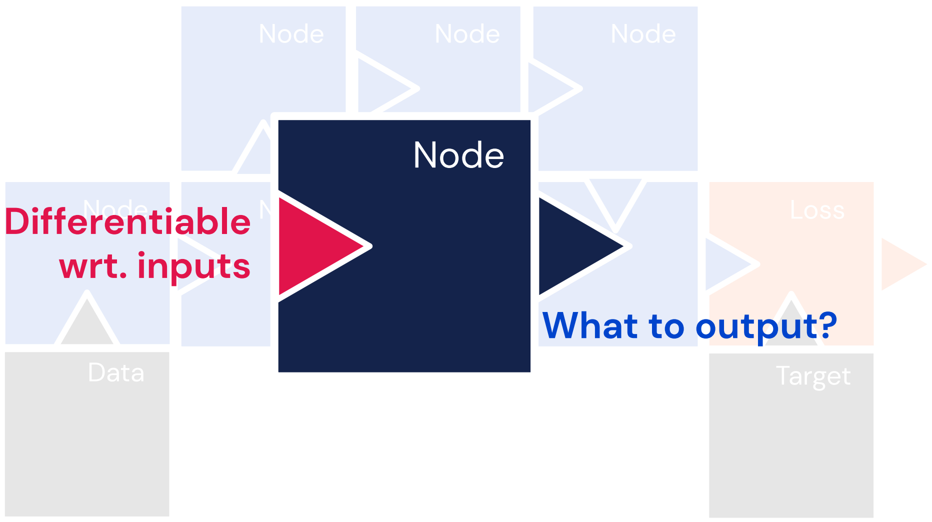
Danilo J. Rezende @DeepSpiker

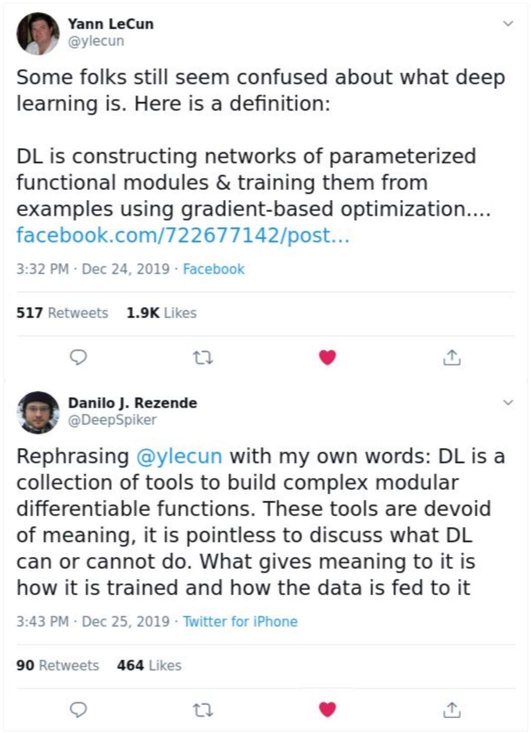
Rephrasing @ylecun with my own words: DL is a collection of tools to build complex modular differentiable functions. These tools are devoid of meaning, it is pointless to discuss what DL can or cannot do. What gives meaning to it is how it is trained and how the data is fed to it

3:43 PM · Dec 25, 2019 · Twitter for iPhone

90 Retweets **464** Likes

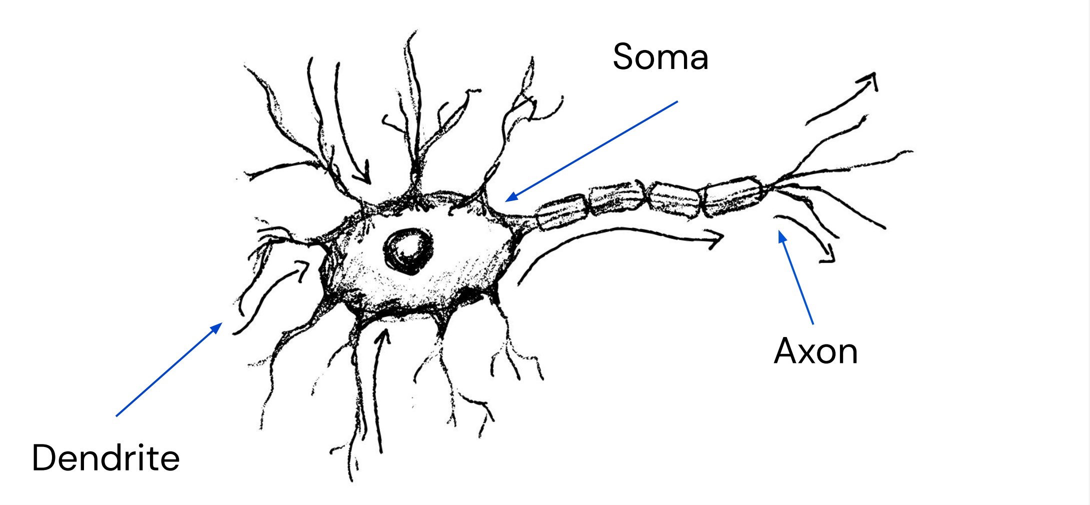
The deep learning puzzle





2 Neural networks

Real neuron



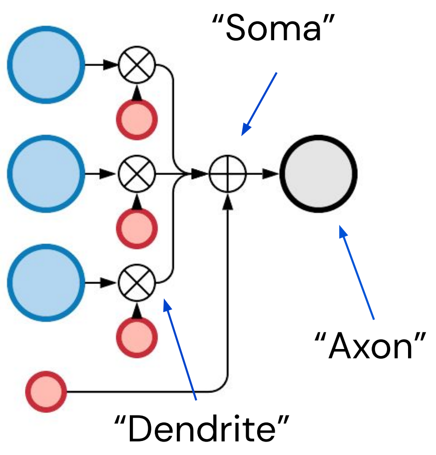
Human brain is estimated to contain around **86,000,000,000** of such neurons.
Each is **connected** to **thousands** of other neurons.

Want to learn more?

Hodgkin AL, Huxley AF **A quantitative description of membrane current and its application to conduction and excitation in nerve**. The Journal of Physiology. 117 (4): 500–44. (**1952**)

- Connected to others
- Represents simple computation
- Has inhibition and excitation connections
- Has a state
- Outputs spikes

Artificial neuron



$$\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i + b$$

$$\sum_{i=0}^{d} \mathbf{w}_i \mathbf{x}_i \qquad \mathbf{x}_0 := 1$$

Want to learn more?

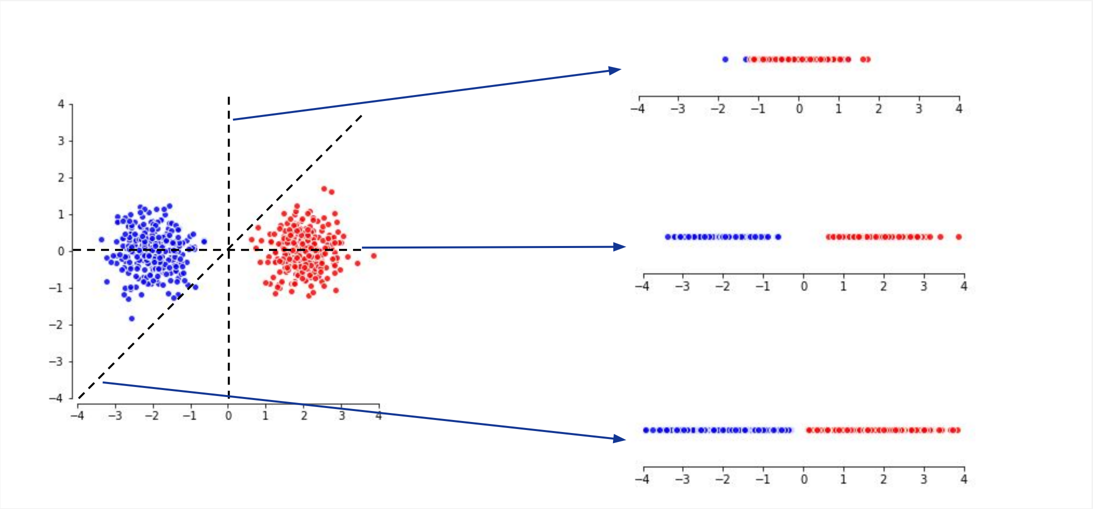
McCulloch, Warren S.; Pitts, Walter
A logical calculus of the ideas immanent in nervous activity
Bulletin of Mathematical Biophysics. 5 (4): 115–133. (**1943**)

- Easy to compose
- Represents simple computation
- Has inhibition and excitation connections

- Is stateless wrt. time
- Outputs real values

The **goal** of simple **artificial neurons** models is to **reflect some** neurophysiological **observations**, **not** to reproduce their **dynamics**.

Artificial neuron



Want to learn more?

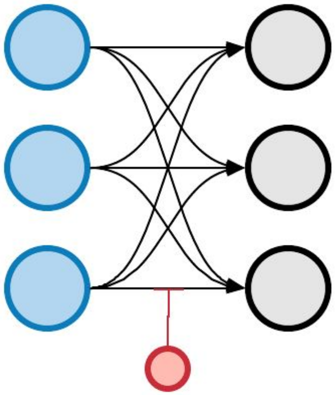
McCulloch, Warren S.; Pitts, Walter
A logical calculus of the ideas immanent in nervous activity
Bulletin of Mathematical Biophysics. 5 (4): 115–133. (**1943**)

- Easy to compose
- Represents simple computation
- Has inhibition and excitation connections

- Is stateless wrt. time
- Outputs real values

The **goal** of simple **artificial neurons** models is to **reflect some** neurophysiological **observations**, **not** to reproduce their **dynamics**.

Linear layer



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

$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

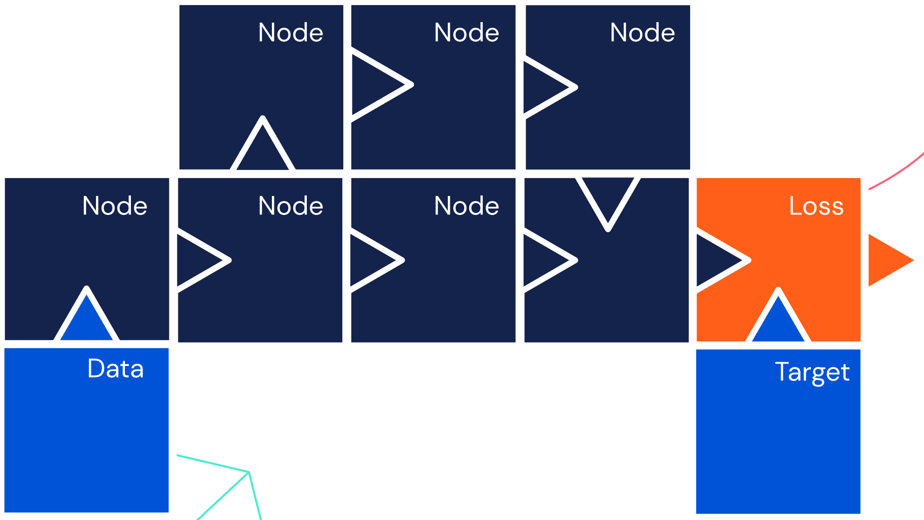
In Machine Learning **linear** really means **affine**.
Neurons in a layer are often called **units**.
Parameters are often called **weights**.

Want to learn more?

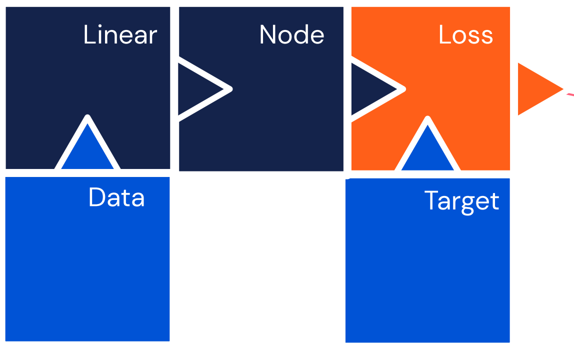
Jouppi, Norman P. et al. ***In-Datacenter Performance Analysis of a Tensor Processing Unit™*** 44th International Symposium on Computer Architecture (ISCA) (**2017**)

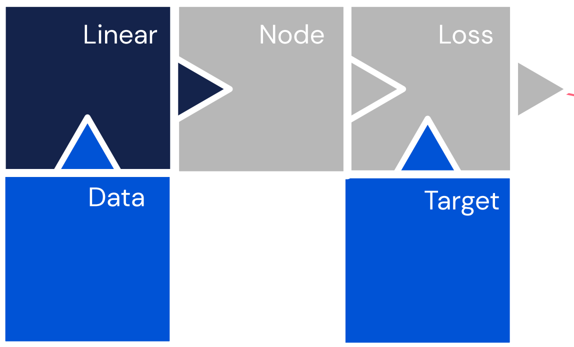
- Easy to compose
- Collection of artificial neurons
- Can be efficiently vectorised
- Fits highly optimised hardware (GPU/TPU)

Isn’t this just linear regression?


Node
Node
Node
Node
Node
Node
Loss
Data
Target

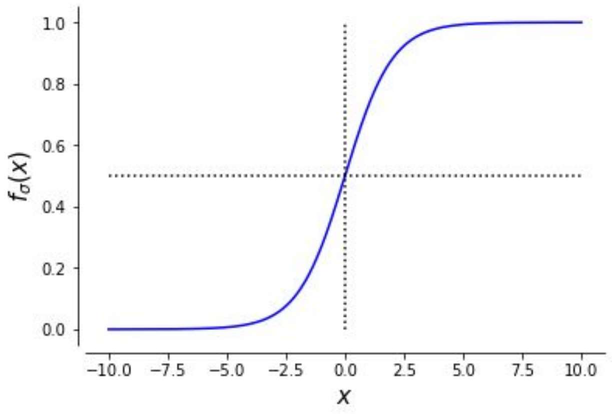
Single layer neural networks


Linear
Node
Loss
Data
Target



Linear
Node
Loss
Data
Target

Sigmoid activation function



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

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

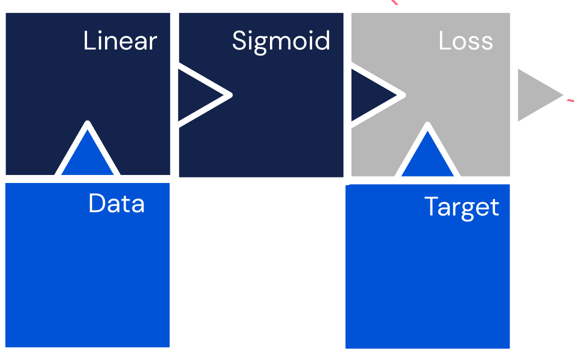
Activation functions are often called **non-linearities**.
Activation functions are applied **point-wise**.

Want to learn more?

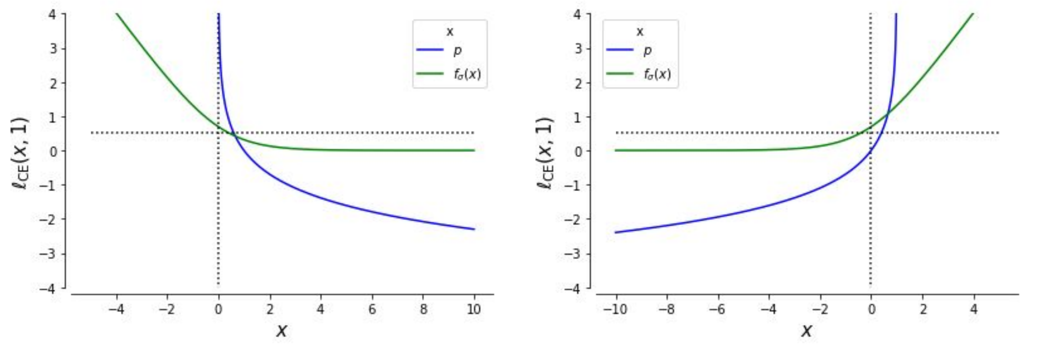
Hinton G. ***Deep belief networks***. Scholarpedia. 4 (5): 5947. (**2009**)

- Introduces non-linear behaviour
- Produces probability estimate
- Has simple derivatives

- Saturates
- Derivatives vanish


Linear
Sigmoid
Loss
Data
Target

Cross entropy



$$\ell_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -[\mathbf{t} \log \mathbf{p} + (1 - \mathbf{t}) \log(1 - \mathbf{p})]$$

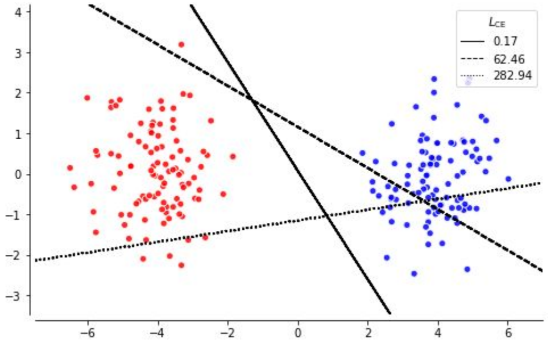
Cross entropy loss is also called
negative log likelihood or **logistic loss.**

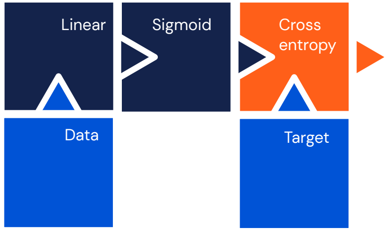
Want to learn more?

Murphy, Kevin **Machine Learning: A Probabilistic Perspective** (2012)

- Encodes negation of logarithm of probability of correct classification
- Composable with sigmoid
- Numerically unstable

The simplest "neural" classifier





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

Cross entropy loss is also called
negative log likelihood or **logistic loss.**
Being **additive** over samples allows for **efficient learning.**

Want to learn more?

Cramer, J. S. **The origins of logistic regression** (Technical report). 119. Tinbergen Institute. pp. 167–178 (**2002**)

- Encodes negation of logarithm of probability of **entirely correct** classification
- Equivalent to logistic regression model
- Numerically unstable

Softmax

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

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

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

Softmax is the most commonly used final activation in **classification**.
It can also be used to have a smooth version of **maximum**.

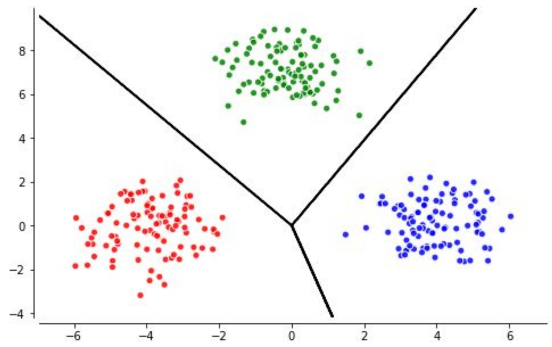
Want to learn more?

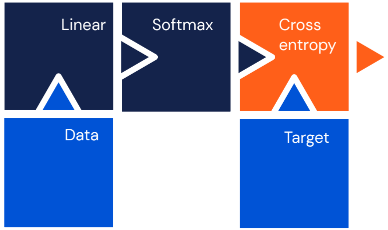
Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron **Softmax Units for Multinoulli Output Distributions**. Deep Learning. MIT Press. pp. 180–184. (**2016**)

- Multi-dimensional generalisation of sigmoid
- Produces probability estimate
- Has simple derivatives

- Saturates
- Derivatives vanish

Softmax + Cross entropy





$$\ell_{\mathrm{CE}}(f_{\mathrm{sm}}(\mathbf{x}), \mathbf{t}) = -\sum_{j=1}^{k} \mathbf{t}_j \log[f_{\mathrm{sm}}(\mathbf{x}_j)] = -\sum_{j=1}^{k} \mathbf{t}_j [\mathbf{x}_j - \log \sum_{l=1}^{k} e^{\mathbf{x}_l}]$$

Widely used not only in **classification** but also in **RL.**
Cannot represent sparse **outputs (sparsemax).**
Does not scale too well with ***k*.**

Want to learn more?

Martins, Andre, and Ramon Astudillo. **From softmax to sparsemax: A sparse model of attention and multi-label classification.** International Conference on Machine Learning. (**2016**)

- Encodes negation of logarithm of probability of entirely correct classification
- Equivalent to multinomial logistic regression model
- Numerically stable combination

Uses

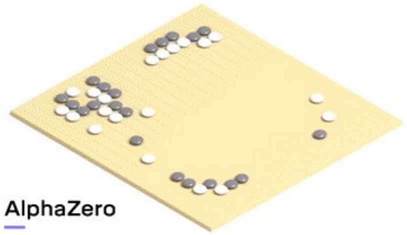
Handwritten digits recognition at **92% level**.

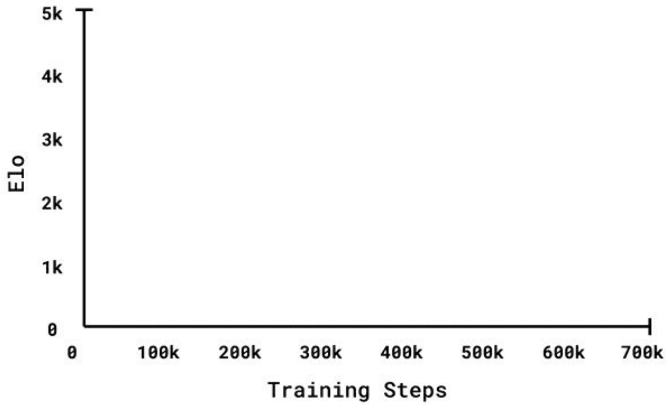
Highly dimensional spaces are surprisingly **easy** to **shatter** with **hyperplanes**.

Widely used in **commercial** applications.

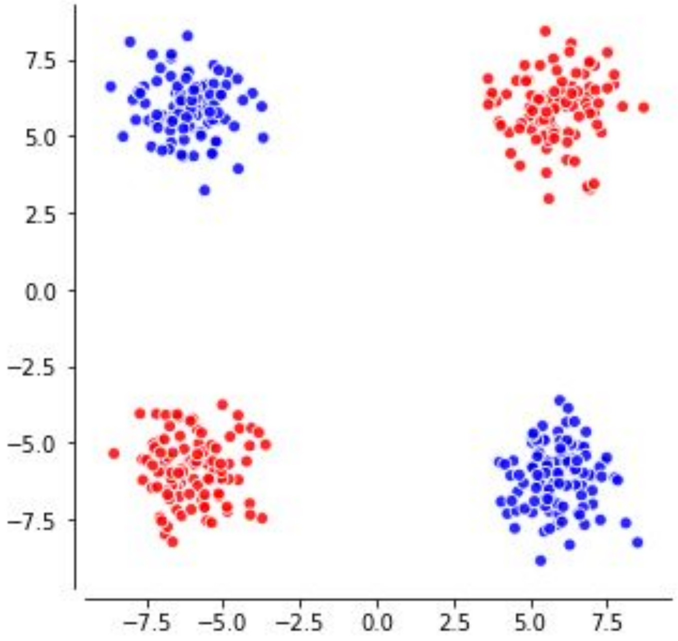
For a long time a crucial model for **Natural Language Processing** under the name of **MaxEnt (Maximum Entropy Classifier)**.

… and limitations

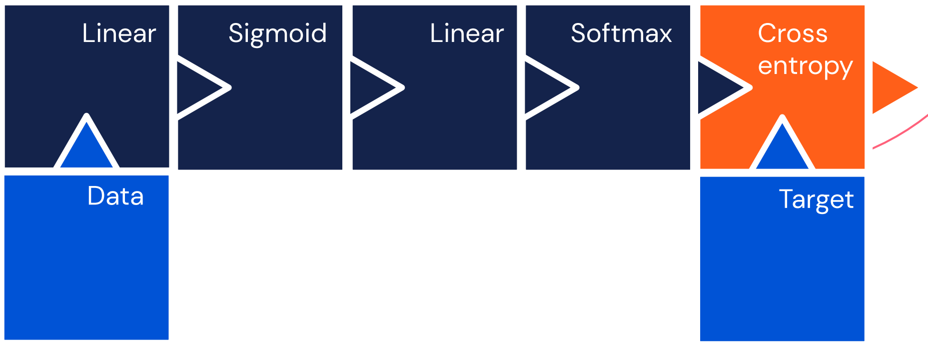


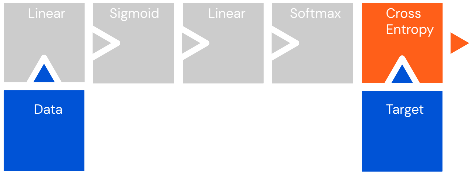


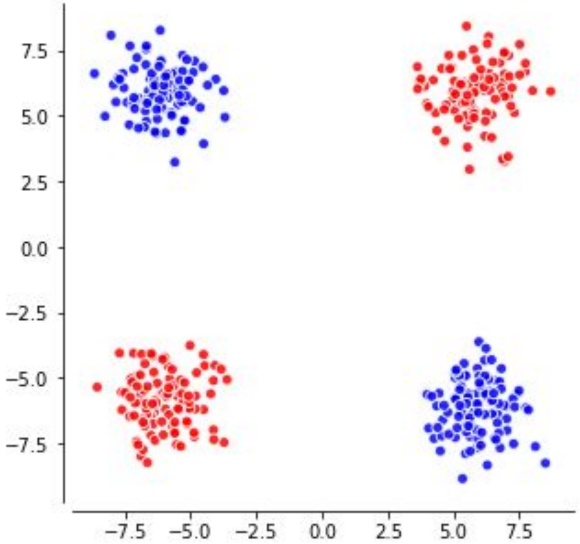
… and limitations

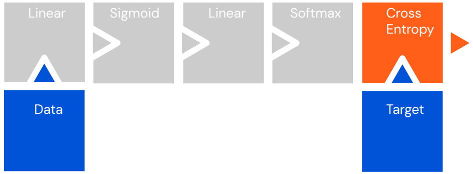


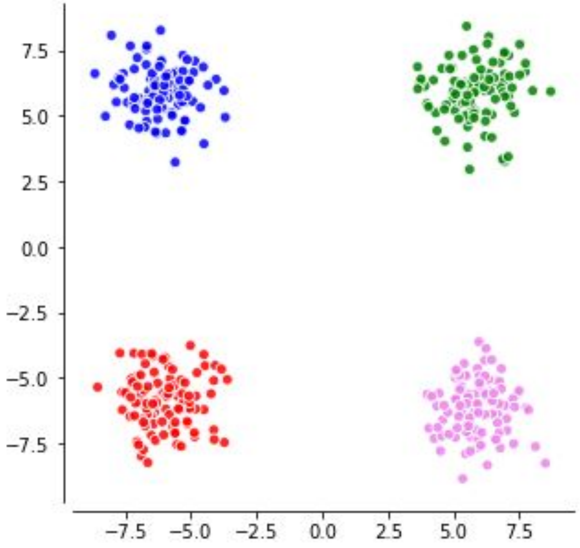
Two layer neural networks

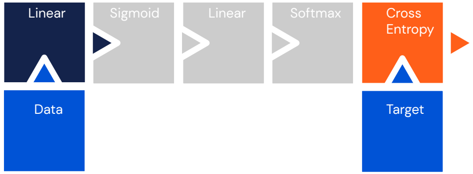

Linear
Sigmoid
Linear
Softmax
Cross entropy
Data
Target

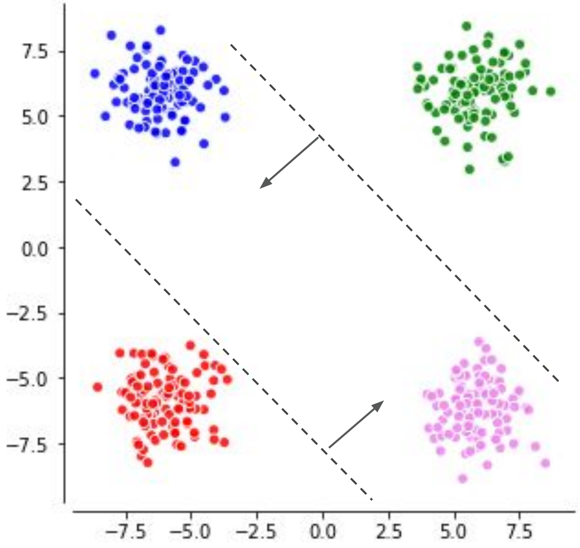

Linear
Sigmoid
Linear
Softmax
Cross Entropy
Data
Target

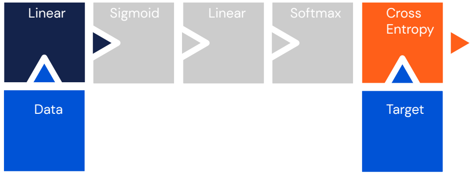

7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

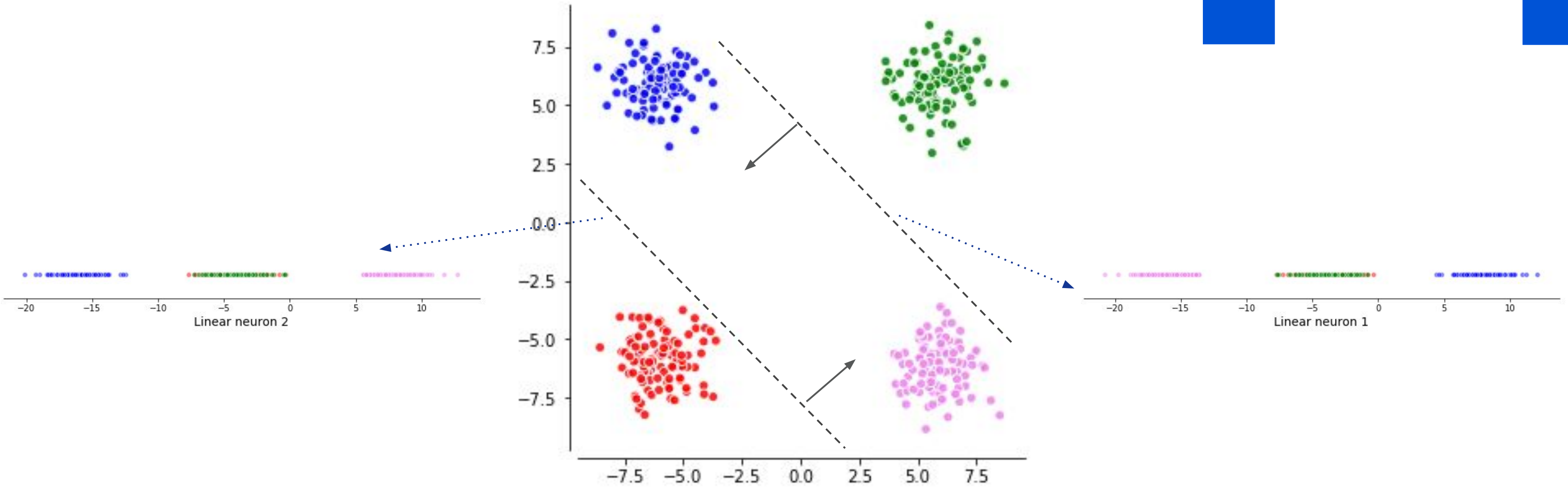

Linear
Sigmoid
Linear
Softmax
Cross Entropy
Data
Target


7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

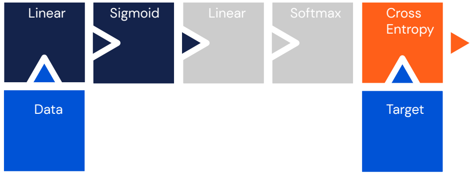

Linear
Sigmoid
Linear
Softmax
Cross Entropy
Data
Target

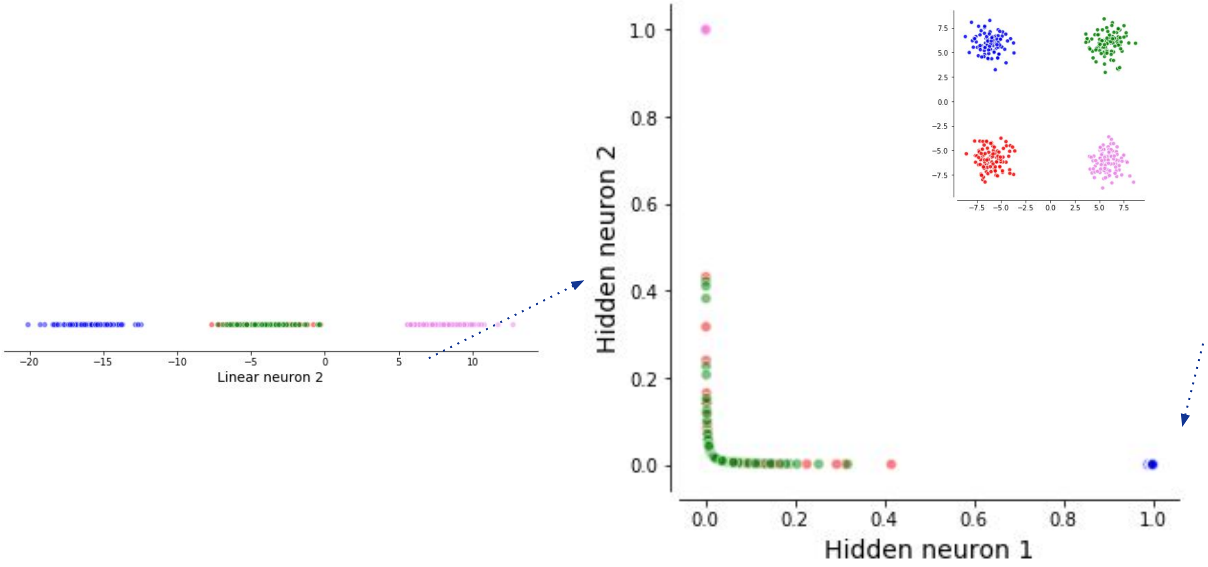

7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5



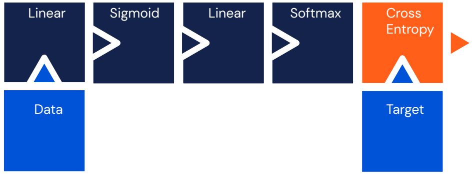


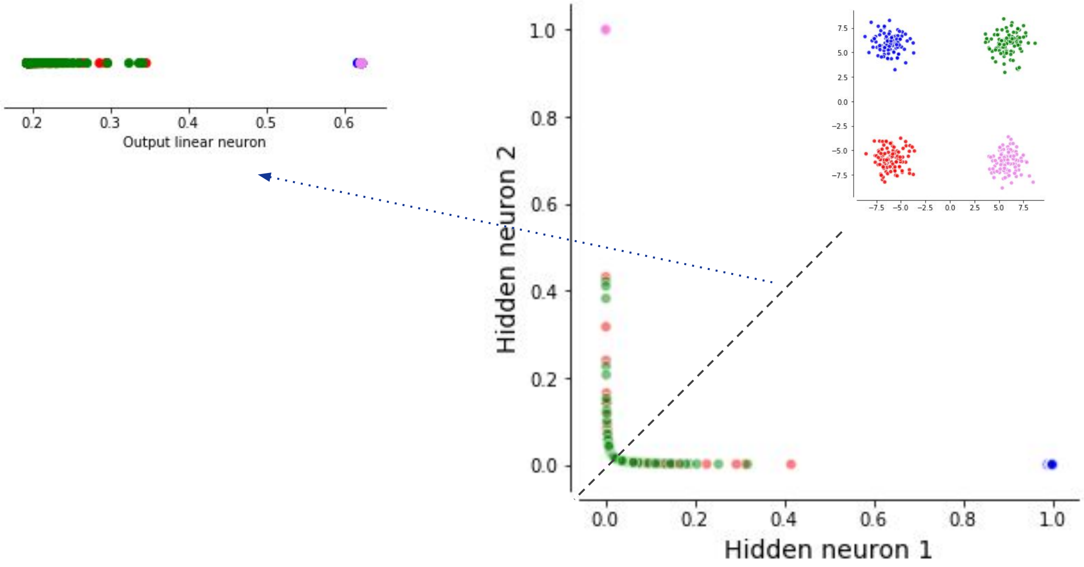
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$



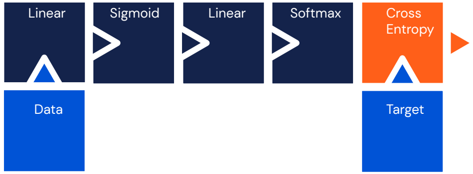


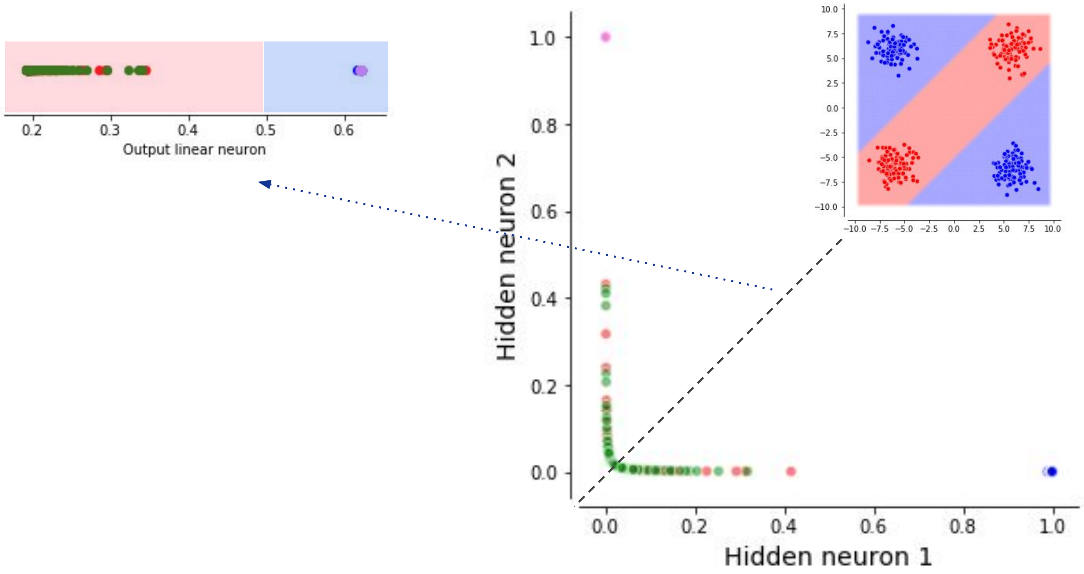
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$





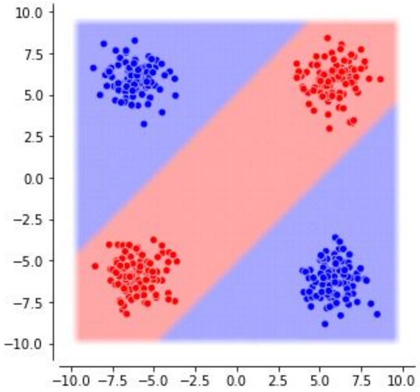
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

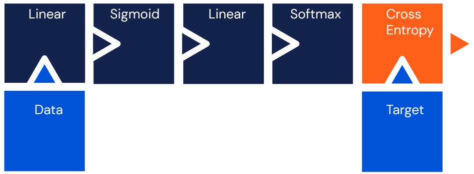




$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

1-hidden layer network vs XOR





$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

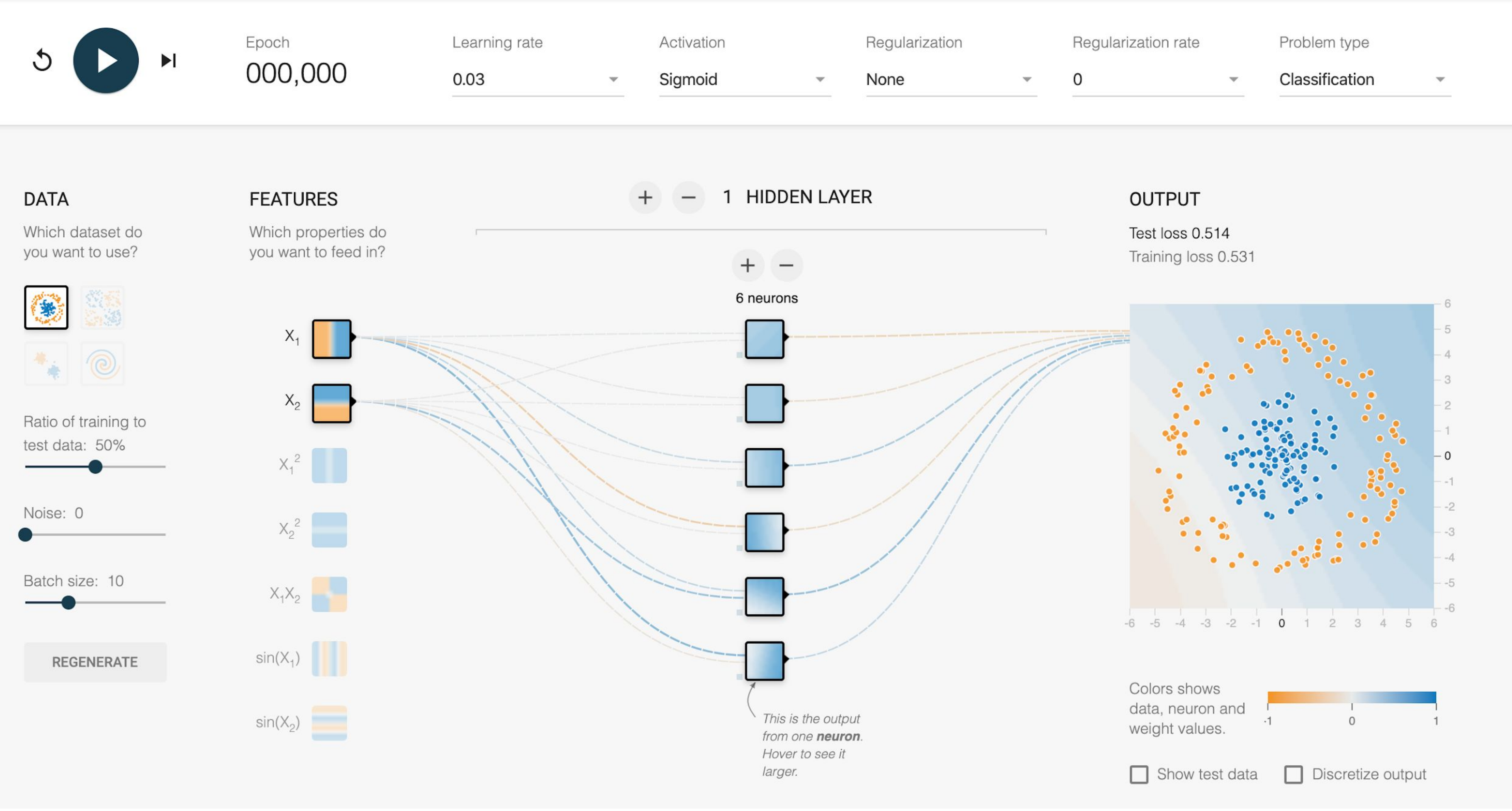
Want to learn more?

Blum, E. K. **Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions** Neural computation 1.4 532-540. (**1989**)

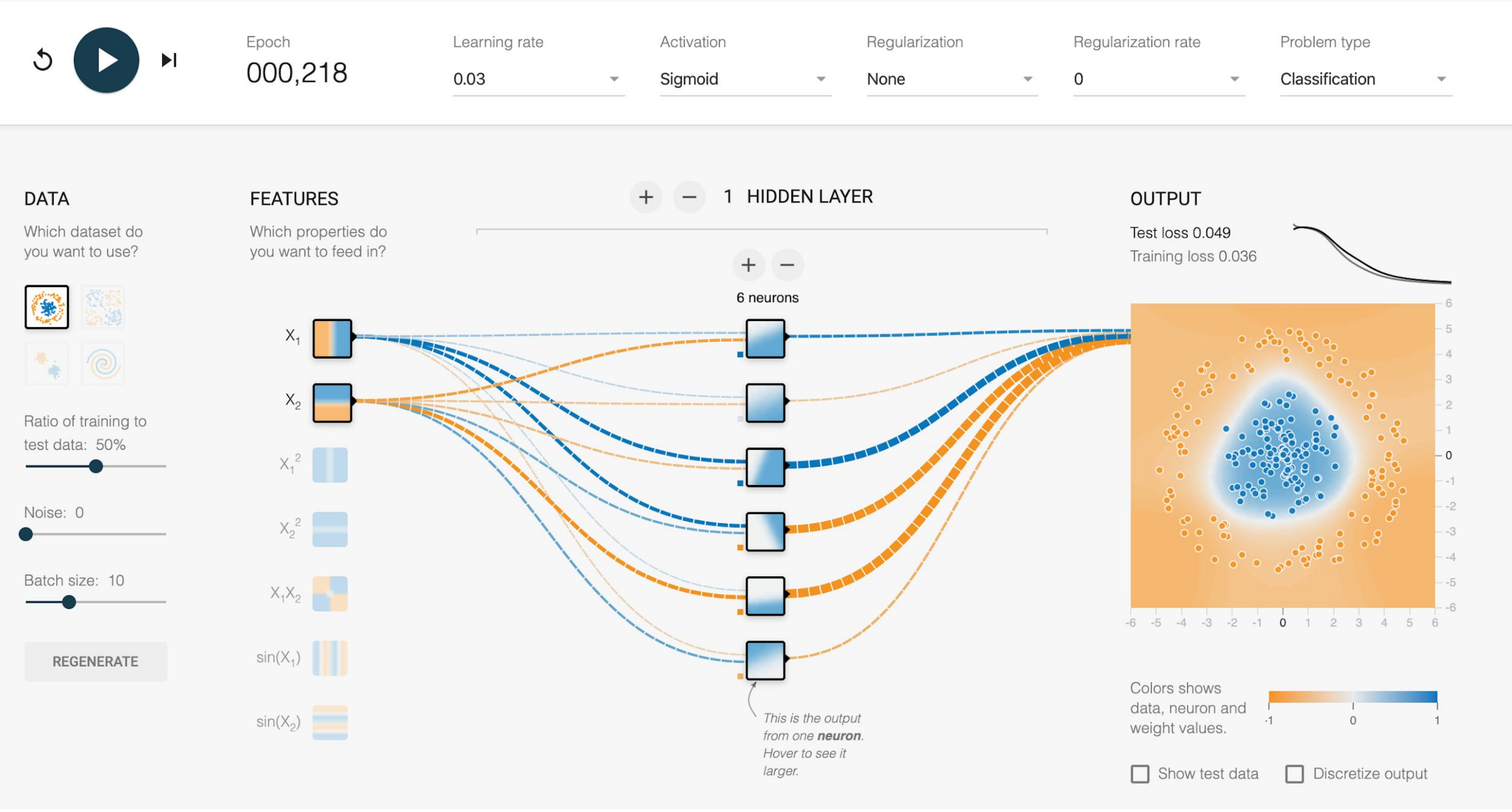
- With just 2 hidden neurons we solve XOR
- Hidden layer allows us to bent and twist input space
- We use linear model on top, to do the classification

Hidden layer provides **non-linear input space transformation** so that final linear layer can classify.

http://playground.tensorflow.org/ by **Daniel Smilkov** and **Shan Carter**



http://playground.tensorflow.org/ by **Daniel Smilkov** and **Shan Carter**



Universal Approximation Theorem

For any continuous function from a hypercube $[0,1]^d$ to real numbers, and every positive epsilon, there exists a **sigmoid** based, 1-hidden layer neural network that obtains at most epsilon error in functional space.

Big enough network can **approximate,** but not **represent** any smooth function. The math trick is to show that **networks** are **dense** in the **space** of **target functions**.

Want to learn more?

Cybenko., G. **Approximations by superpositions of sigmoidal functions**, Mathematics of Control, Signals, and Systems, 2 (4), 303-314 (1989)

- One of the most important theoretical results for Neural Networks
- Shows, that they are extremely expressive
- Tells us nothing about learning
- Size of network grows exponentially

Universal Approximation Theorem

For any continuous function from a hypercube $[0,1]^d$ to real numbers, **non-constant, bounded and continuous activation function f**, and every positive epsilon, there exists a 1-hidden layer neural network using **f** that obtains at most epsilon error in functional space.

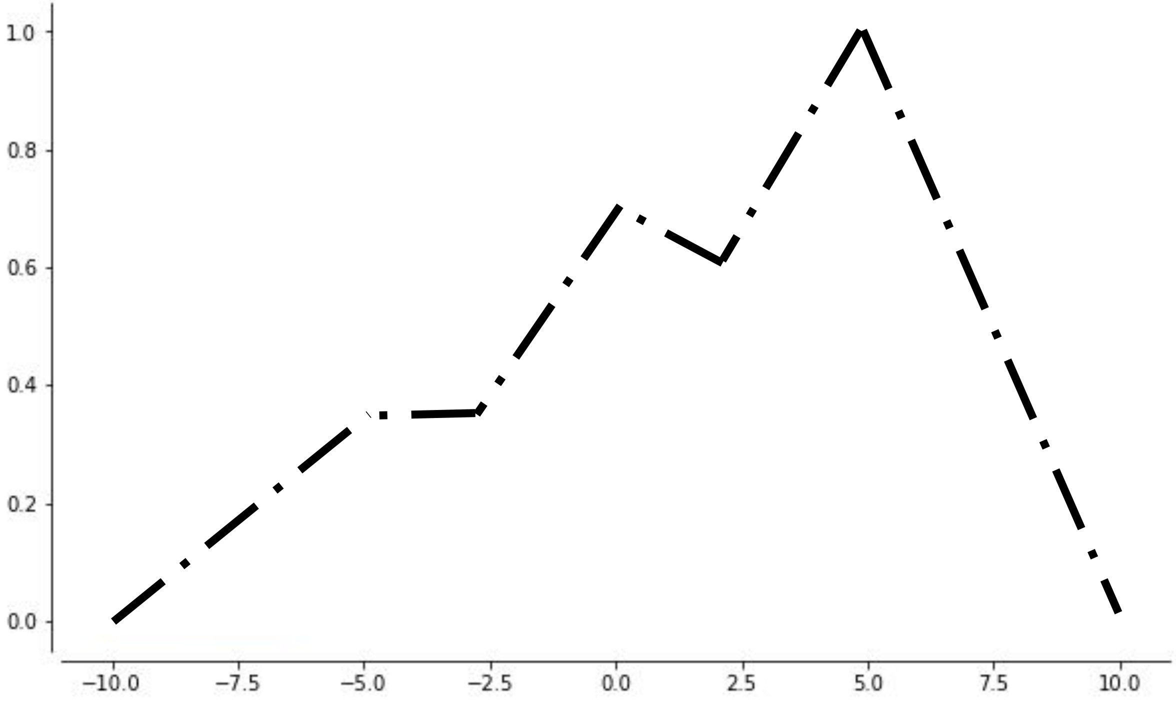
Big enough network can **approximate,** but not **represent** any smooth function. The math trick is to show that **networks** are **dense** in the **space** of **target functions**.

Want to learn more?

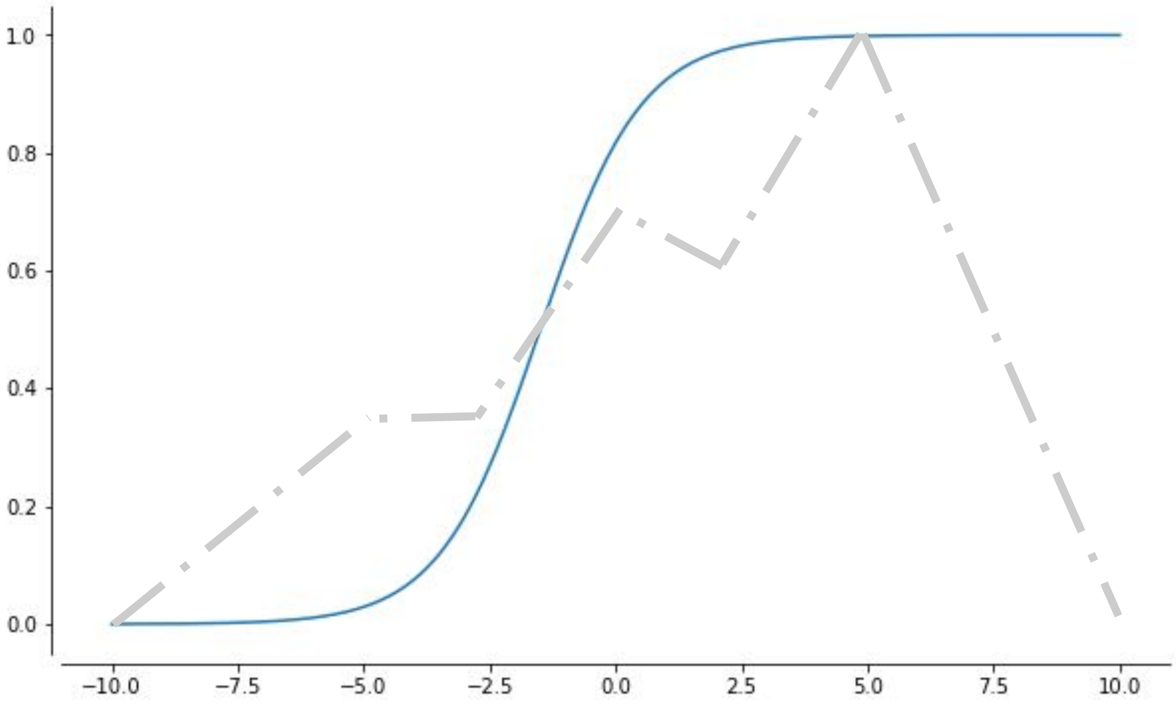
Kurt Hornik **Approximation Capabilities of Multilayer Feedforward Networks**, Neural Networks, 4(2), 251–25 (**1991**)

- One of the most important theoretical results for Neural Networks
- Shows, that they are extremely expressive
- Tells us nothing about learning
- Size of network grows exponentially

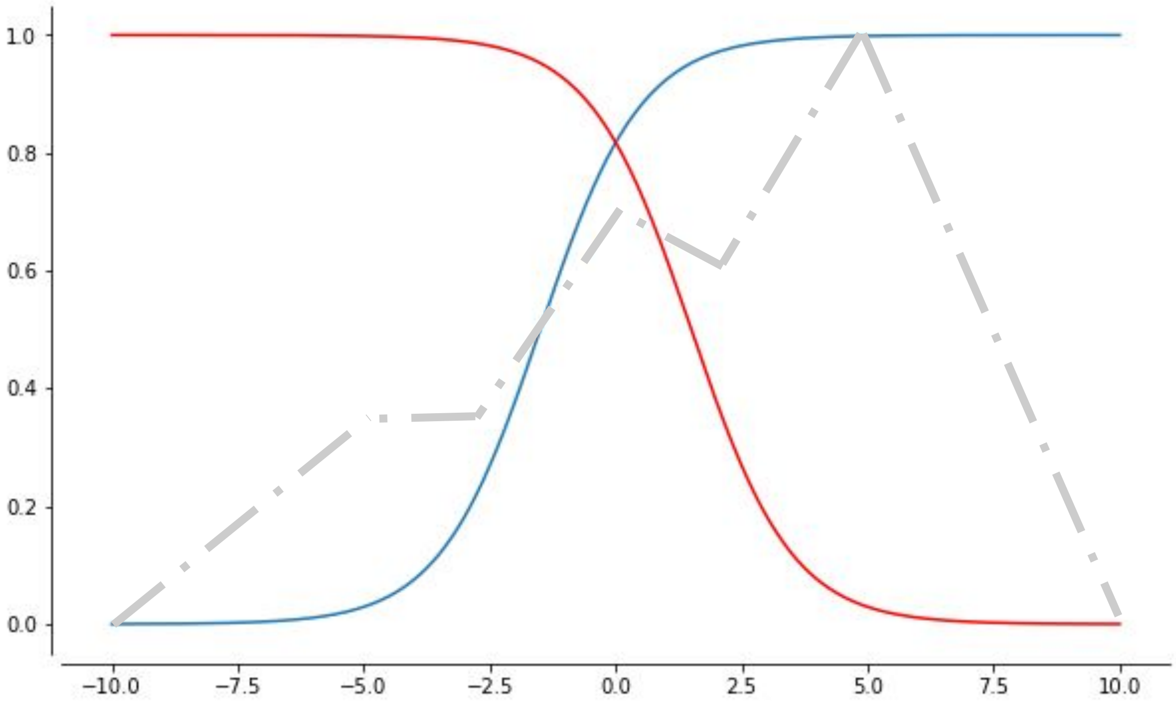
Universal Approximation ~~Theorem~~ Intuition



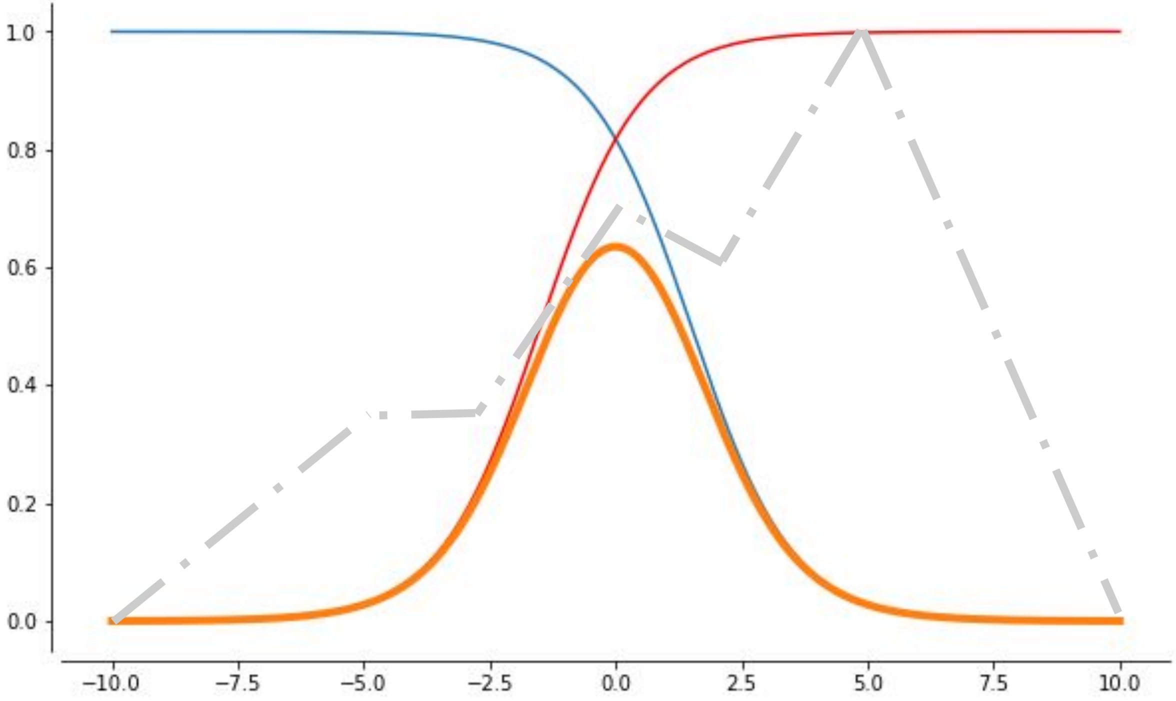
Universal Approximation ~~Theorem~~ Intuition



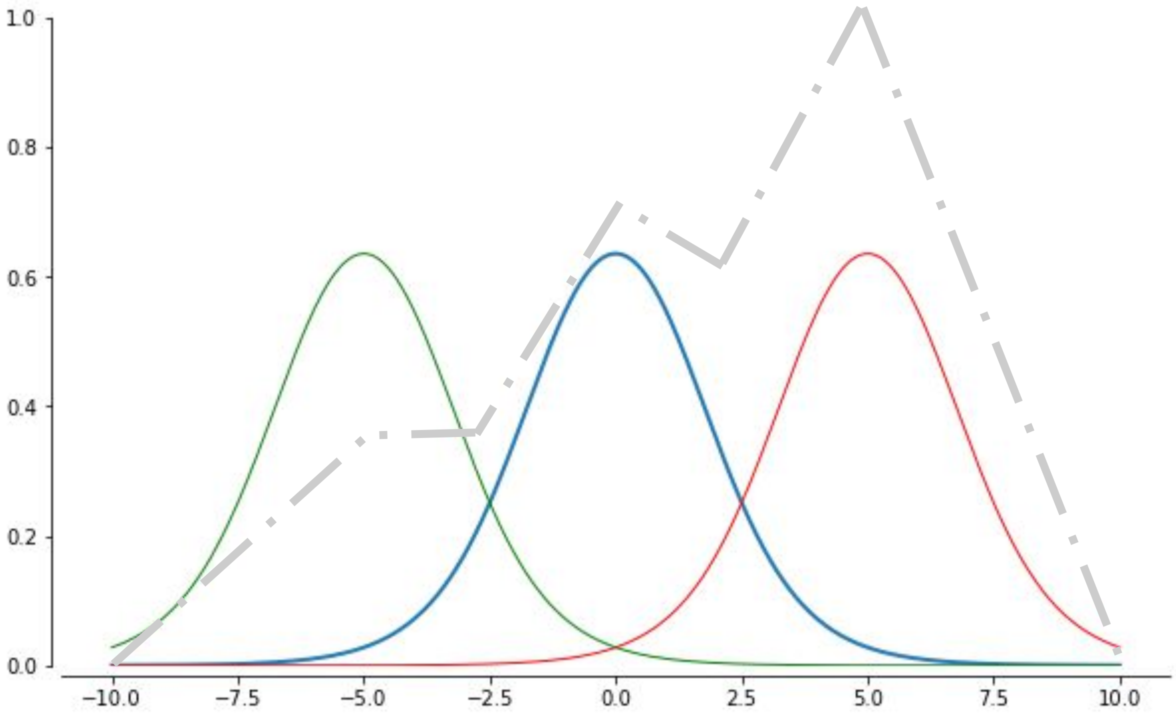
Universal Approximation ~~Theorem~~ Intuition



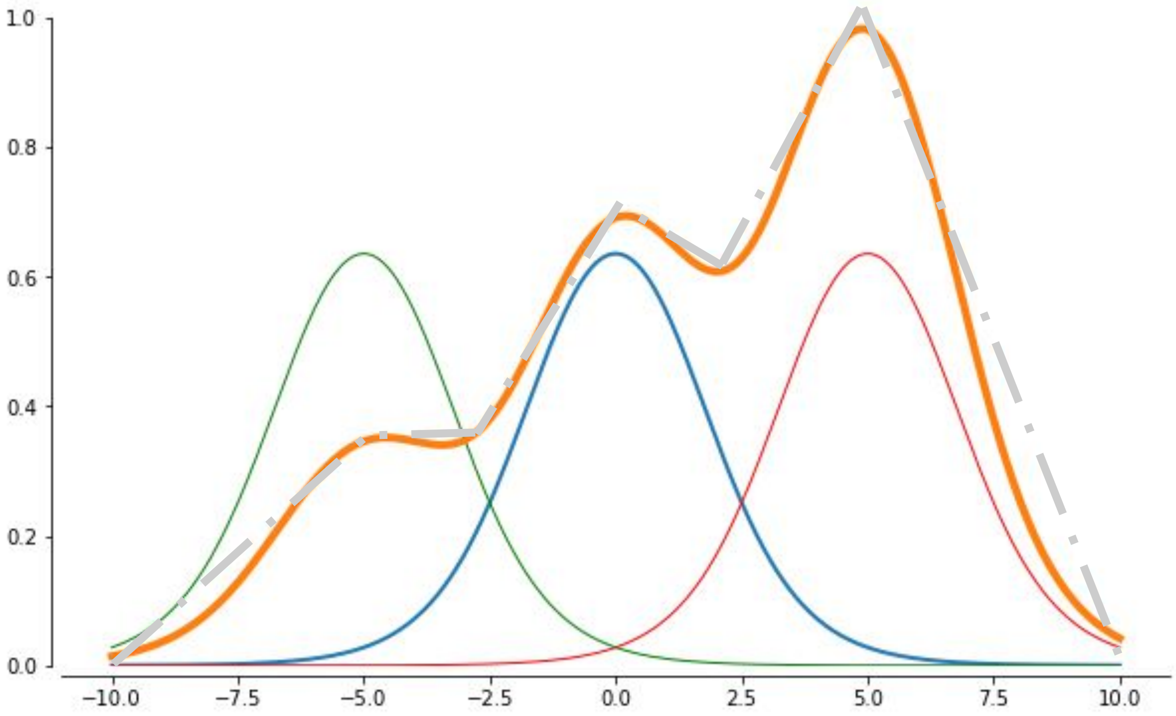
Universal Approximation ~~Theorem~~ Intuition



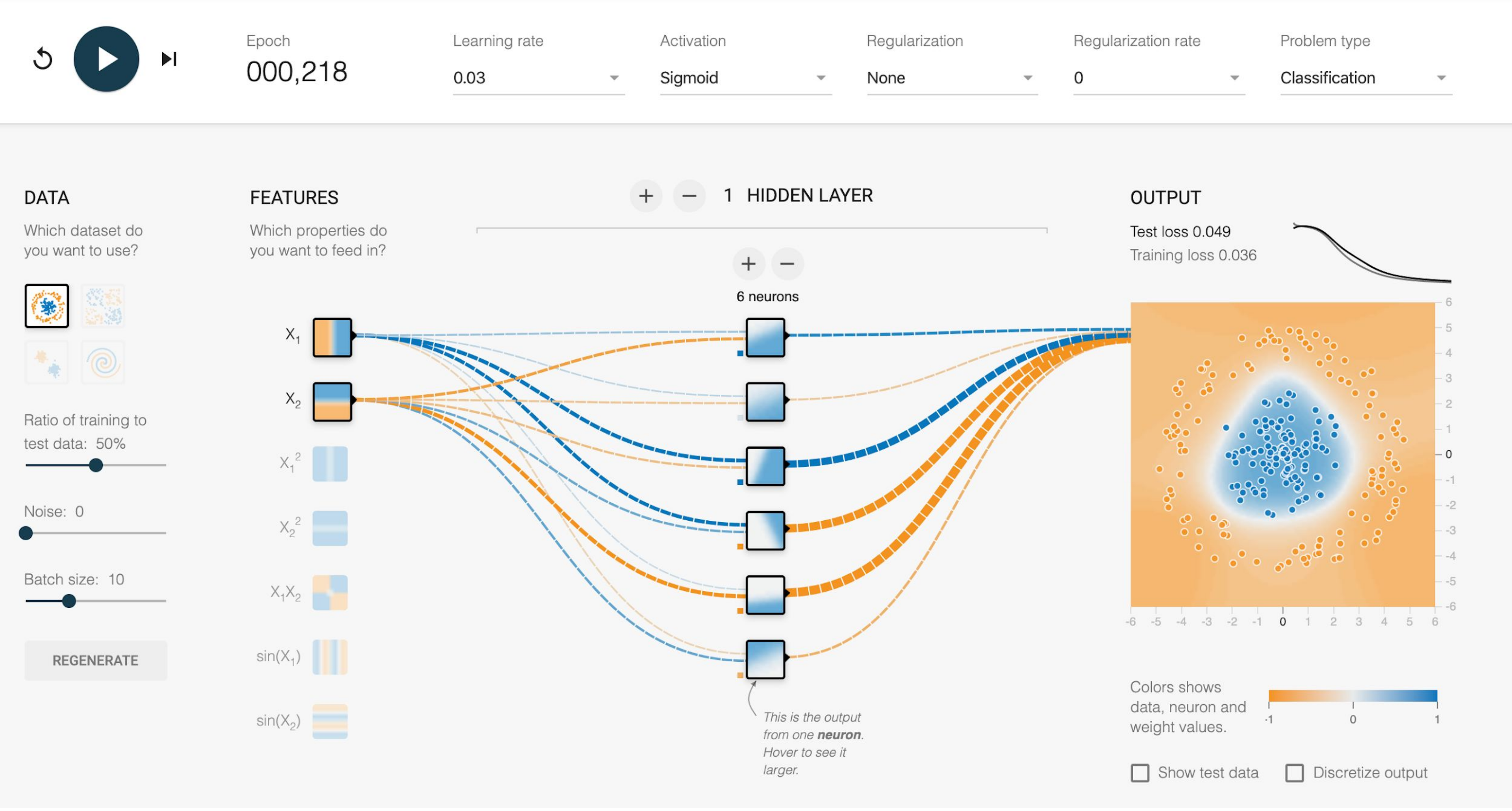
Universal Approximation ~~Theorem~~ Intuition



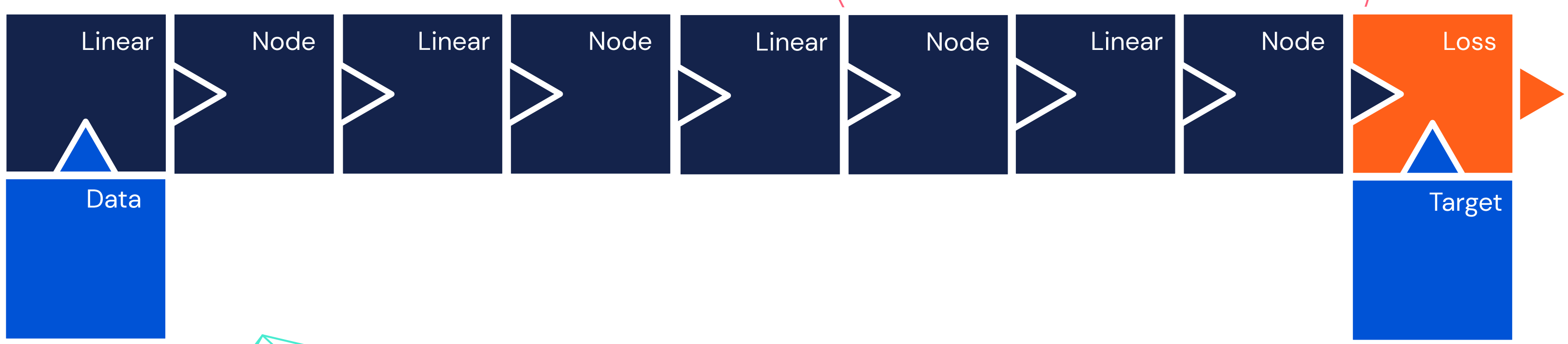
Universal Approximation ~~Theorem~~ Intuition



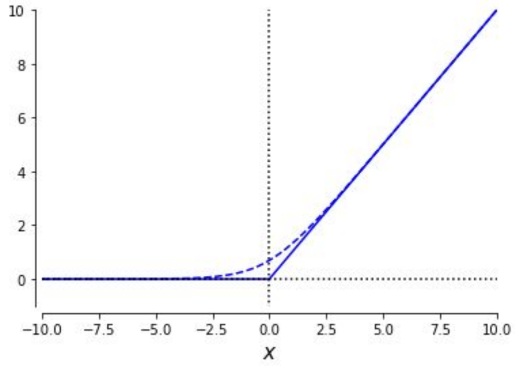
http://playground.tensorflow.org/ by **Daniel Smilkov** and **Shan Carter**



Deep neural networks


Linear
Node
Linear
Node
Linear
Node
Linear
Node
Loss
Data
Target

Rectified Linear Unit (ReLU)



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

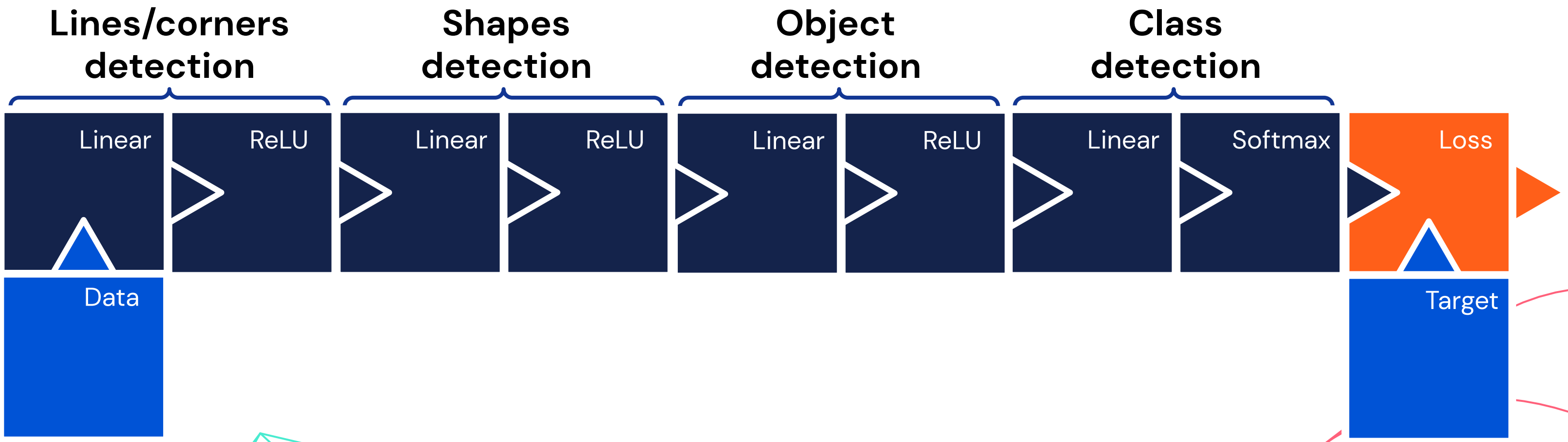
$$f_{\mathrm{sp}}(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$$

Want to learn more?

Hahnloser, R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; Seung, H. S. **Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit**. Nature. 405: 947–951 (**2000**)

- Introduces non-linear behaviour
- Creates piecewise linear functions
- Derivatives do not vanish
- Dead neurons can occur
- Technically not differentiable at 0

One of the **most commonly used activation functions.**
Made **math analysis** of networks **much simpler.**


Lines/corners detection
Shapes detection
Object detection
Class detection
Linear
ReLU
Linear
ReLU
Linear
ReLU
Linear
Softmax
Loss
Data
Target

Depth

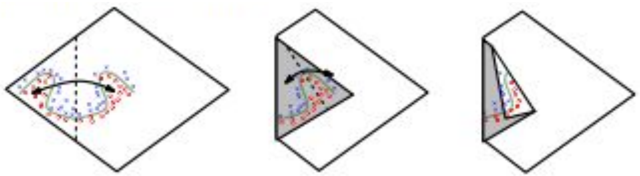


Figure 3: Space folding of 2-D space in a non-trivial way. Note how the folding can potentially identify symmetries in the boundary that it needs to learn.

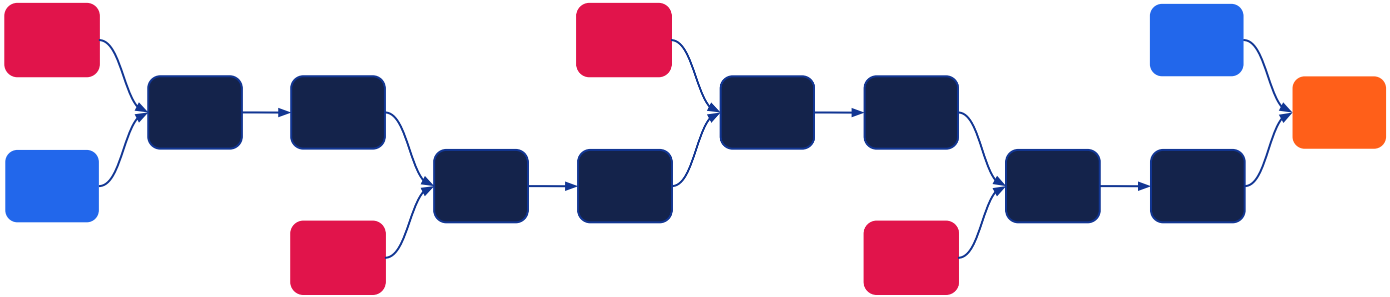
Number of **linear regions** grows **exponentially** with **depth**, and **polynomially** with **width**.

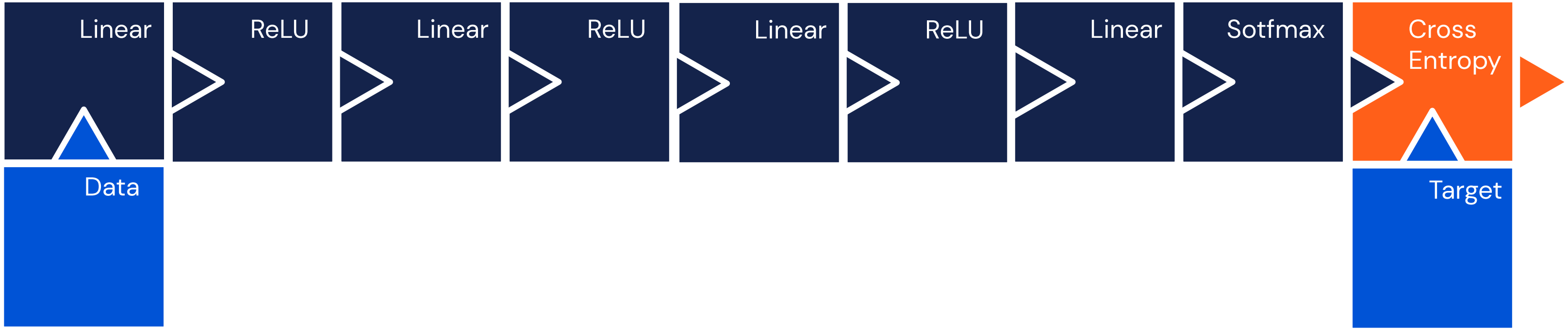
Want to learn more?

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, Yoshua Bengio. **On the Number of Linear Regions of Deep Neural Networks** Arxiv (2014)

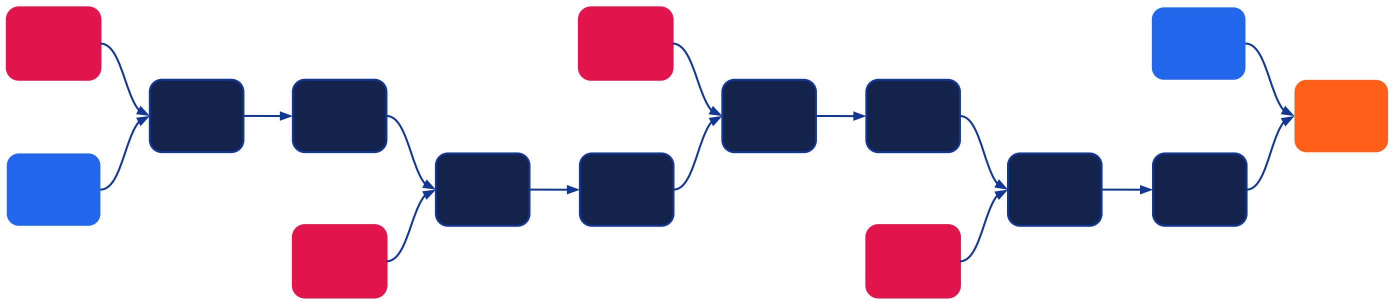
- Expressing symmetries and regularities is much easier with deep model than wide one.
- Deep model means many non-linear composition and thus harder learning

Neural networks as computational graphs

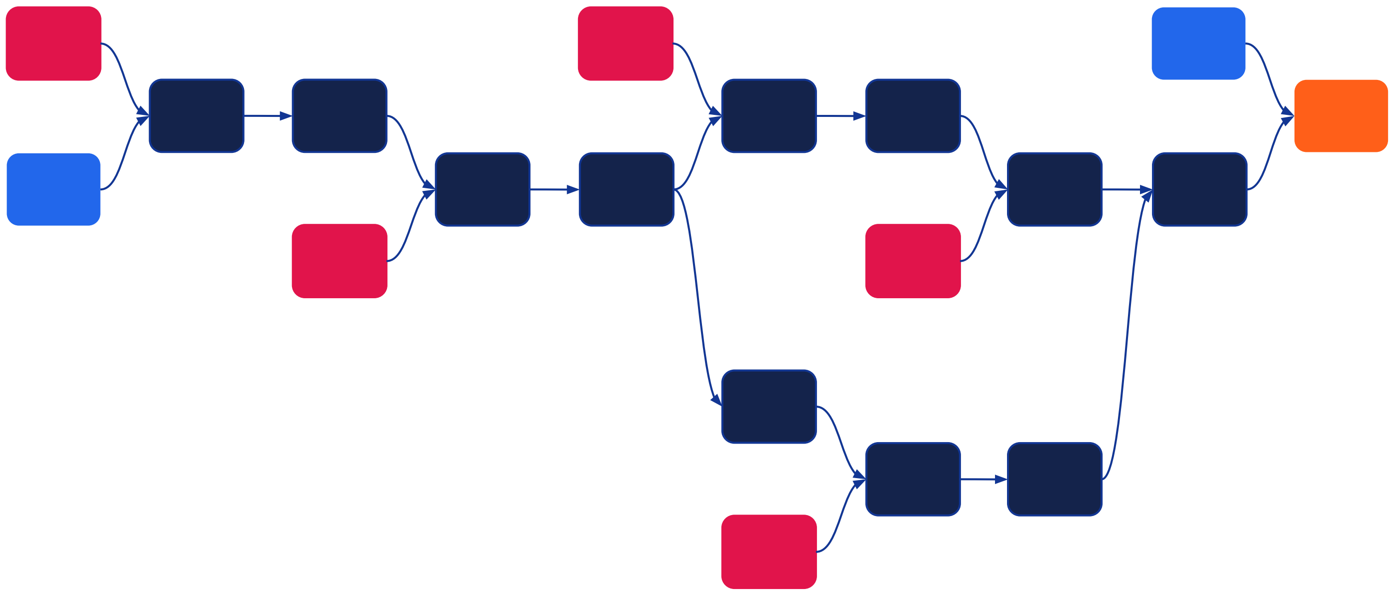




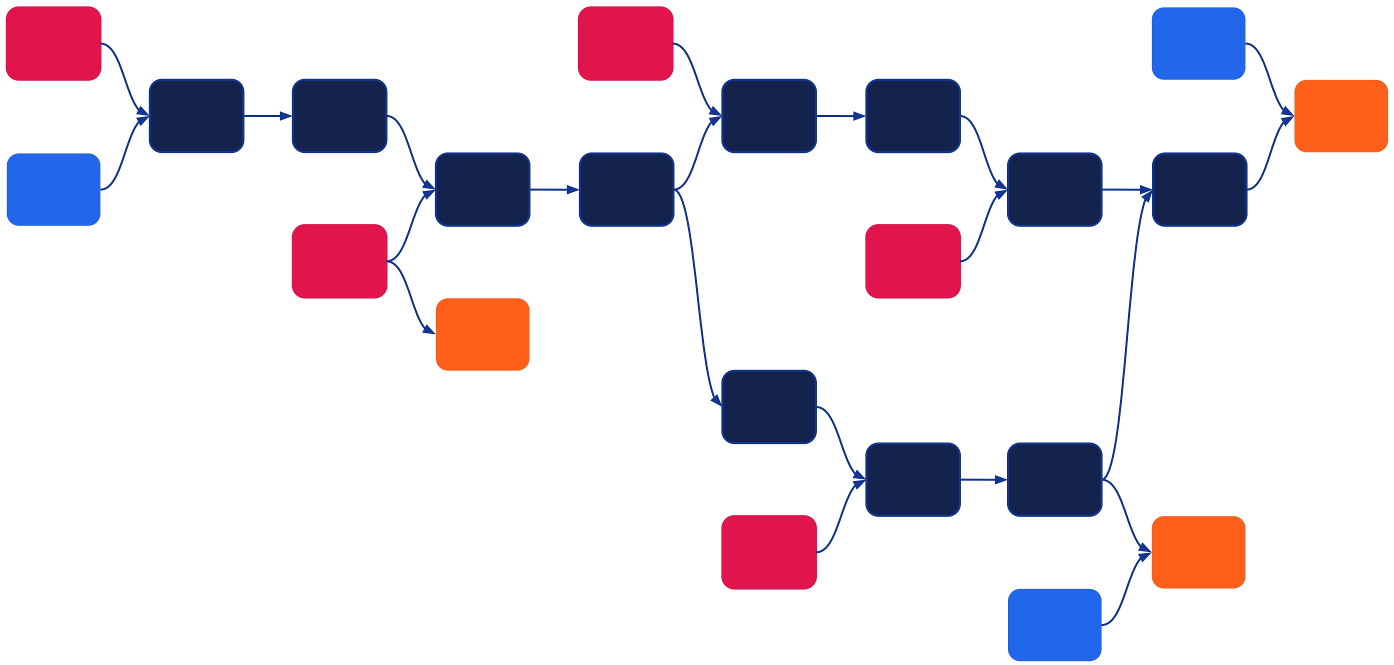
Neural networks as computational graphs



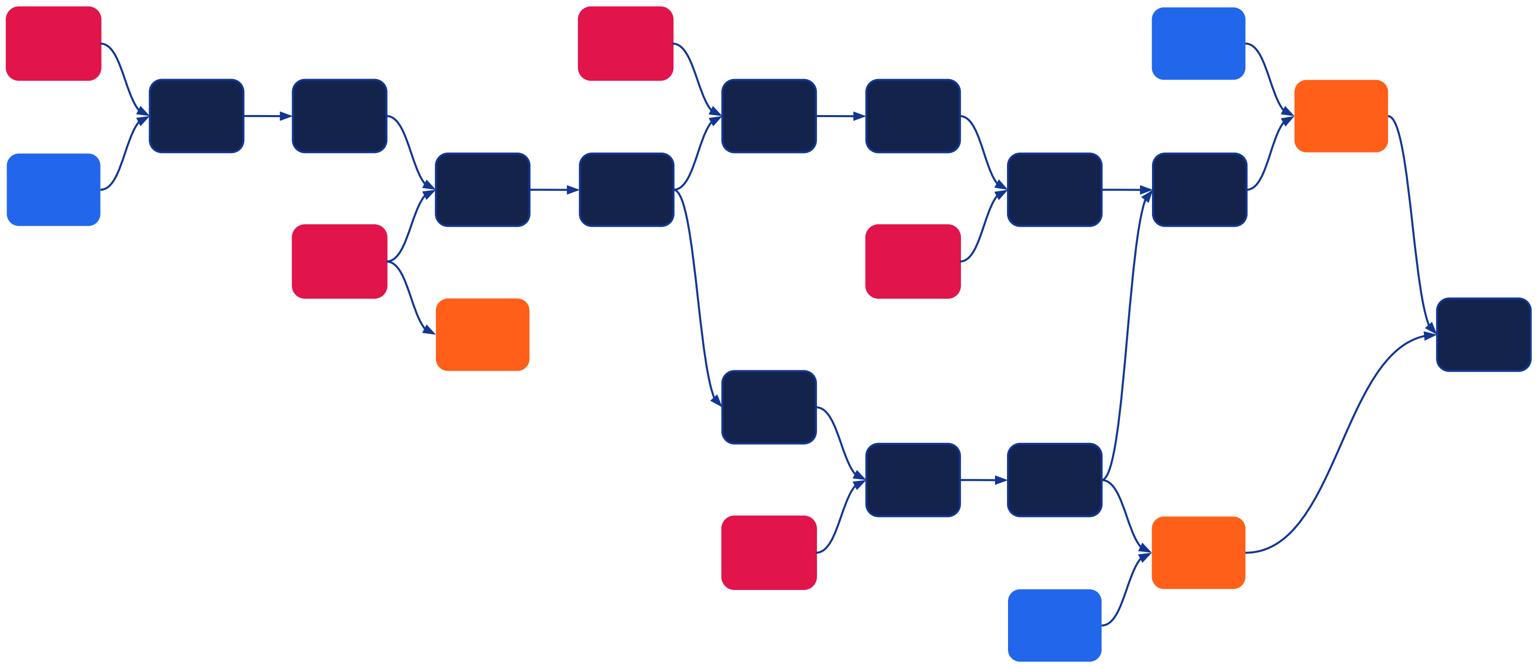
Neural networks as computational graphs



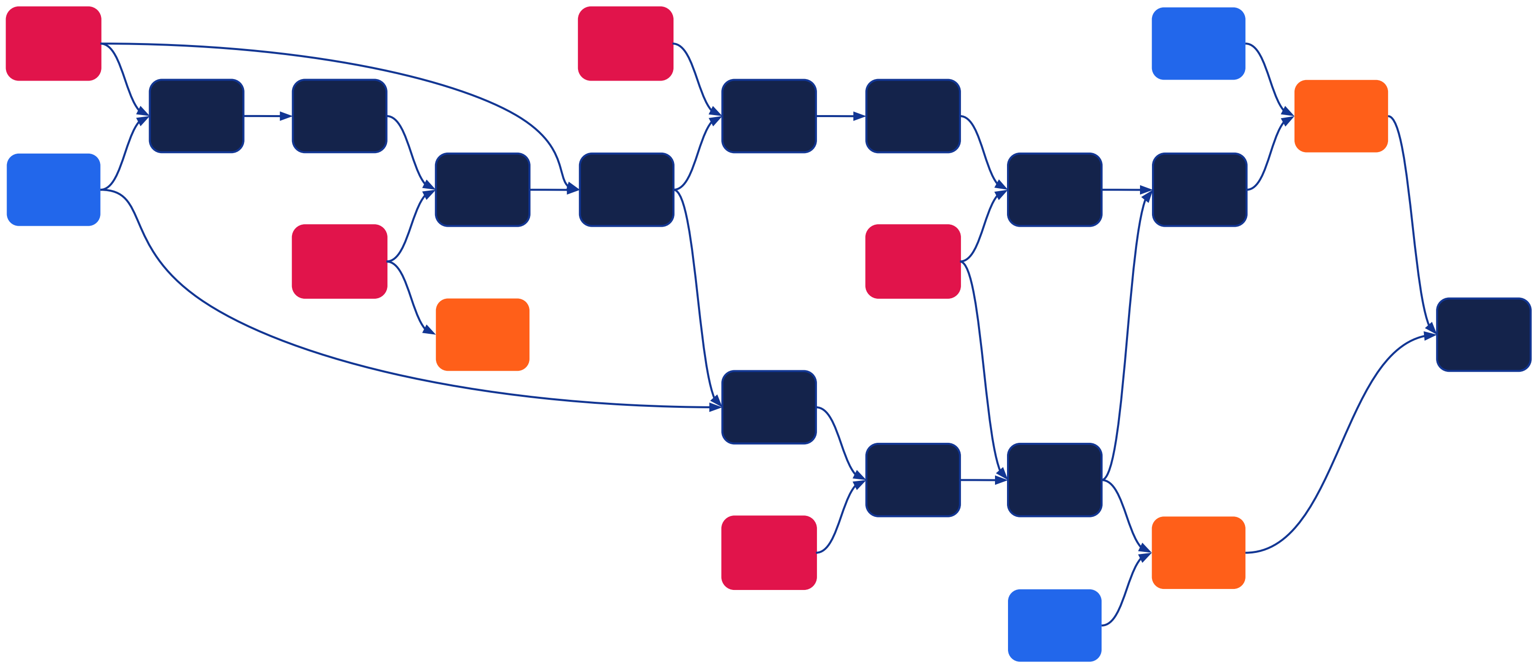
Neural networks as computational graphs



Neural networks as computational graphs



Neural networks as computational graphs



3 Learning

Linear algebra recap

Gradient

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

$$\frac{\partial y}{\partial \mathbf{x}} = \nabla_{\mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial f}{\partial \mathbf{x}_1}, \ldots, \frac{\partial f}{\partial \mathbf{x}_d}\right]$$

Jacobian

$$\mathbf{y} = f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^k$$

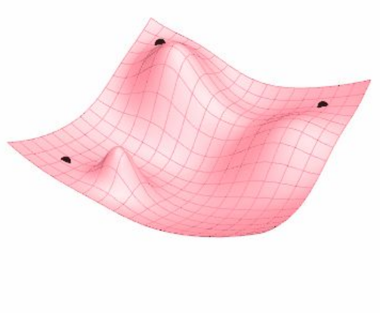
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \mathbf{J}_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_1}{\partial \mathbf{x}_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f_k}{\partial \mathbf{x}_d} \end{bmatrix}$$

Gradient descent recap

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

$$\nabla_\theta L(\theta_t) = \nabla_\theta \sum_i \ell(g(\mathbf{x}^{(i)}, \theta_t), \mathbf{t}^{(i)})$$

$$= \sum_i \nabla_\theta \ell(g(\mathbf{x}^{(i)}, \theta_t), \mathbf{t}^{(i)})$$



Choice of learning rate is **critical**.
Main learning algorithm behind deep learning.
Many modifications: **Adam, RMSProp, ...**

Want to learn more?

Kingma, Diederik P., and Jimmy Ba. **Adam: A method for stochastic optimization** arXiv preprint arXiv:1412.6980 (**2014**).

- Works for any "smooth enough" function
- Can be used on non-smooth targets but with less guarantees
- Converges to local optimum

Neural networks as computational graphs - API

$f(\mathbf{x})$

Forward pass

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

Backward pass

Neural networks as computational graphs - API

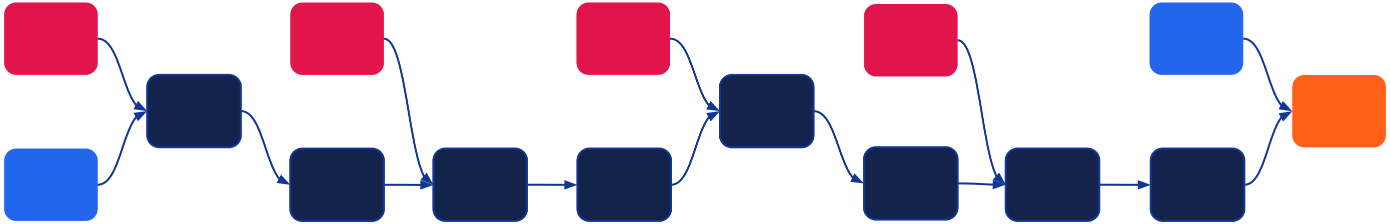
$$f(\mathbf{x})$$

Forward pass

$$\frac{\partial L}{\partial \mathbf{y}} \mathbf{J}_{\mathbf{x}} f(\mathbf{x})$$

Backward pass

Gradient descent and computational graph



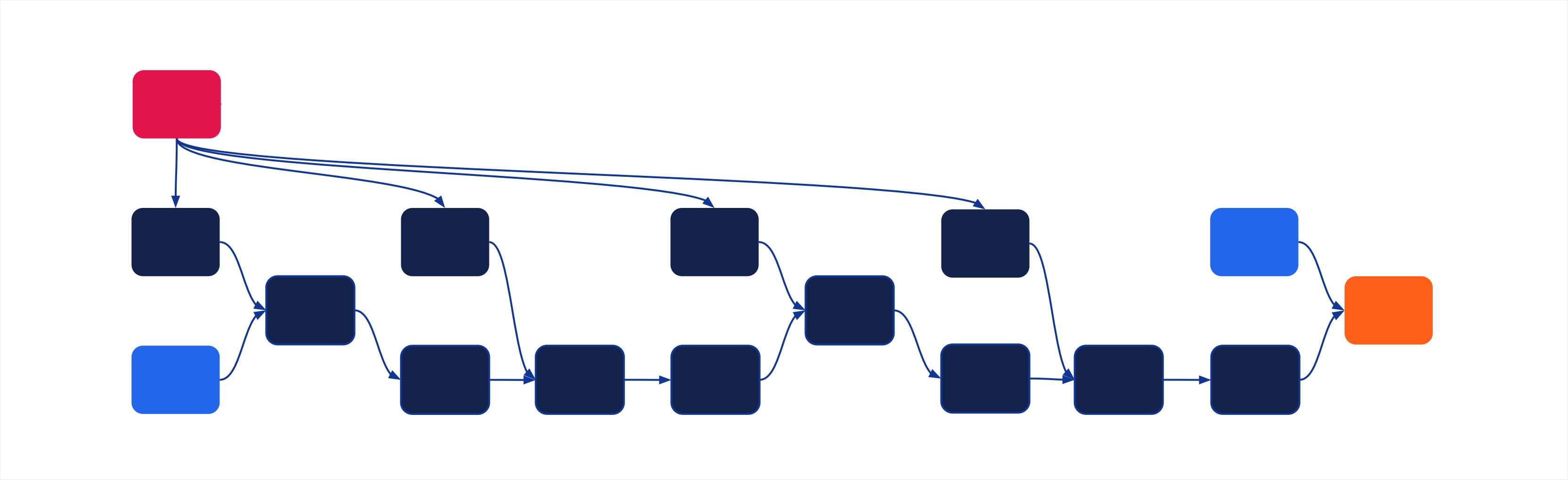
$$\theta_{t+1} := \theta_t - \alpha_t \nabla_\theta L(\theta_t)$$

$$\nabla_\theta L(\theta_t) = \nabla_\theta \sum_i \ell(g(\mathbf{x}^{(i)}, \theta_t), \mathbf{t}^{(i)}) = \sum_i \nabla_\theta \ell(g(\mathbf{x}^{(i)}, \theta_t), \mathbf{t}^{(i)})$$

Want to learn more?

Abadi, Martín, et al. **Tensorflow: A system for large-scale machine learning.** 12th Symposium on Operating Systems Design and Implementation (2016)

Gradient descent and computational graph



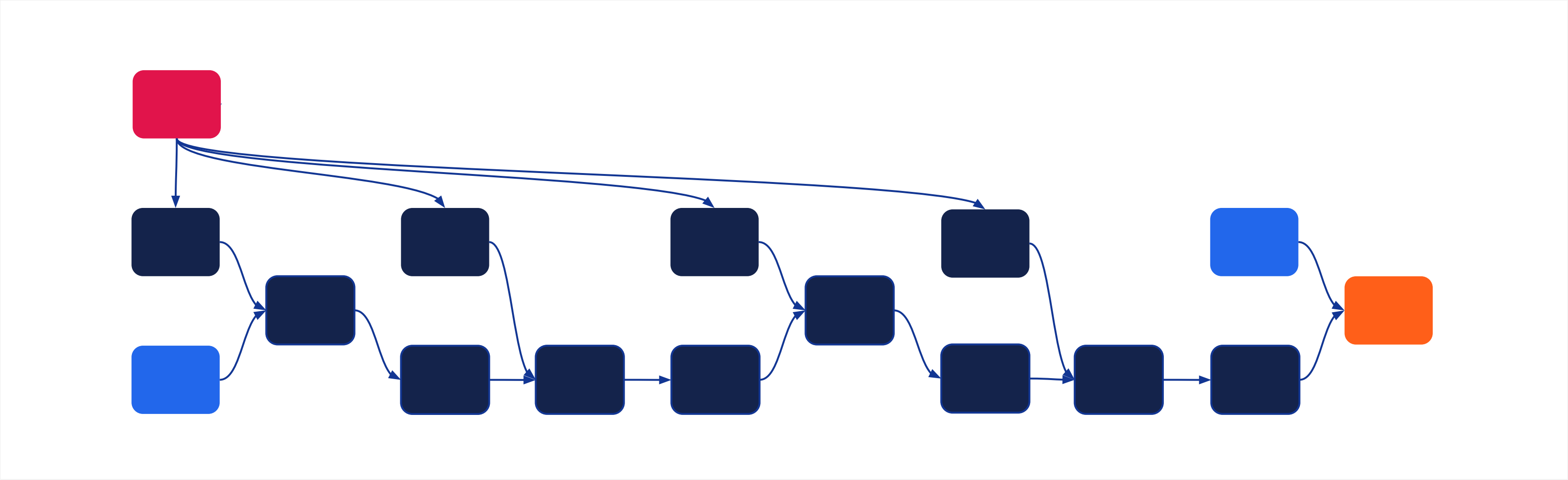
$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \alpha_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t)$$

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}} \sum_i \ell(g(\mathbf{x}^{(i)}, \boldsymbol{\theta}_t), \mathbf{t}^{(i)}) = \sum_i \nabla_{\boldsymbol{\theta}} \ell(g(\mathbf{x}^{(i)}, \boldsymbol{\theta}_t), \mathbf{t}^{(i)})$$

Want to learn more?

Abadi, Martín, et al. **Tensorflow: A system for large-scale machine learning.** 12th Symposium on Operating Systems Design and Implementation **(2016)**

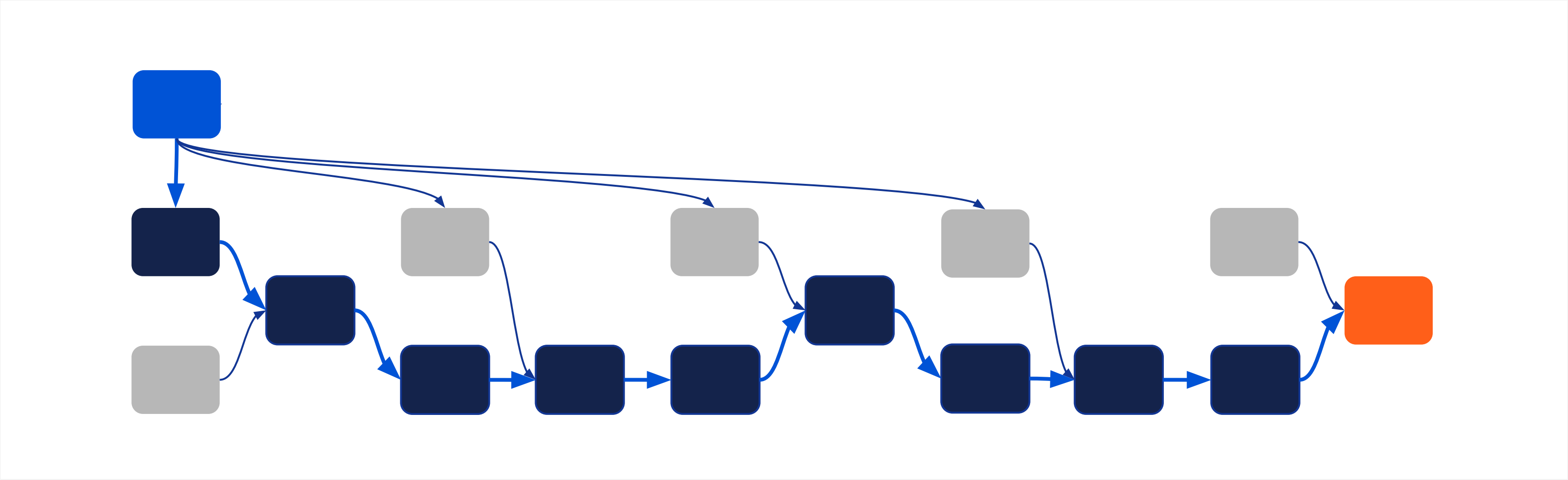
Chain rule, backprop and automatic differentiation



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

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

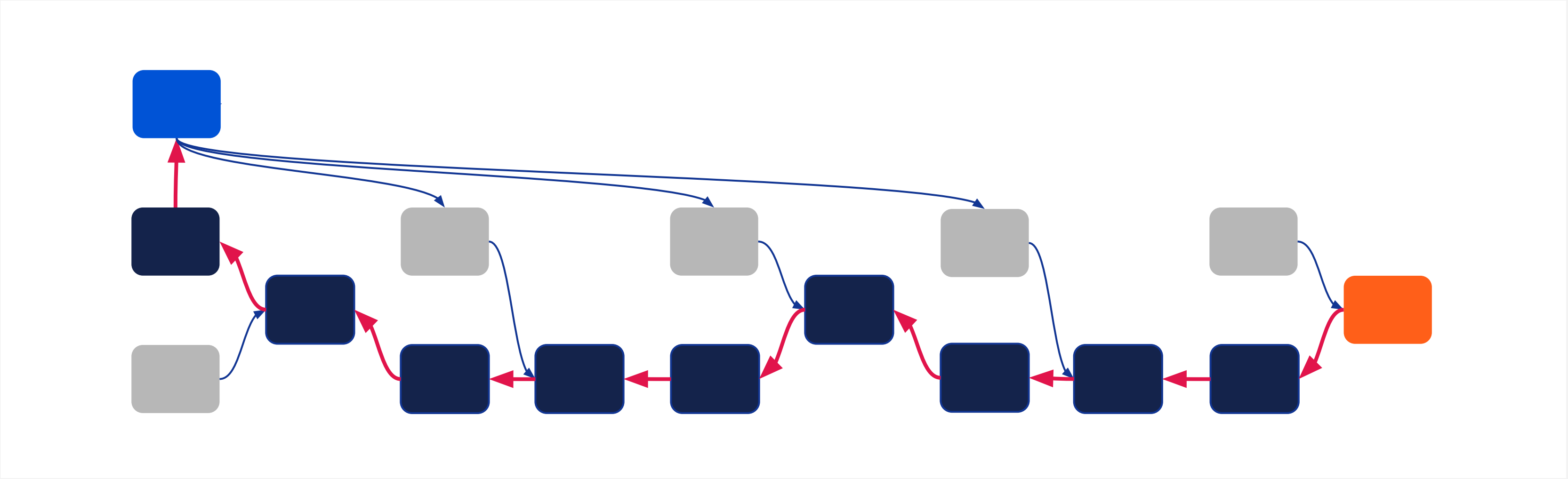
Chain rule, backprop and automatic differentiation



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

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

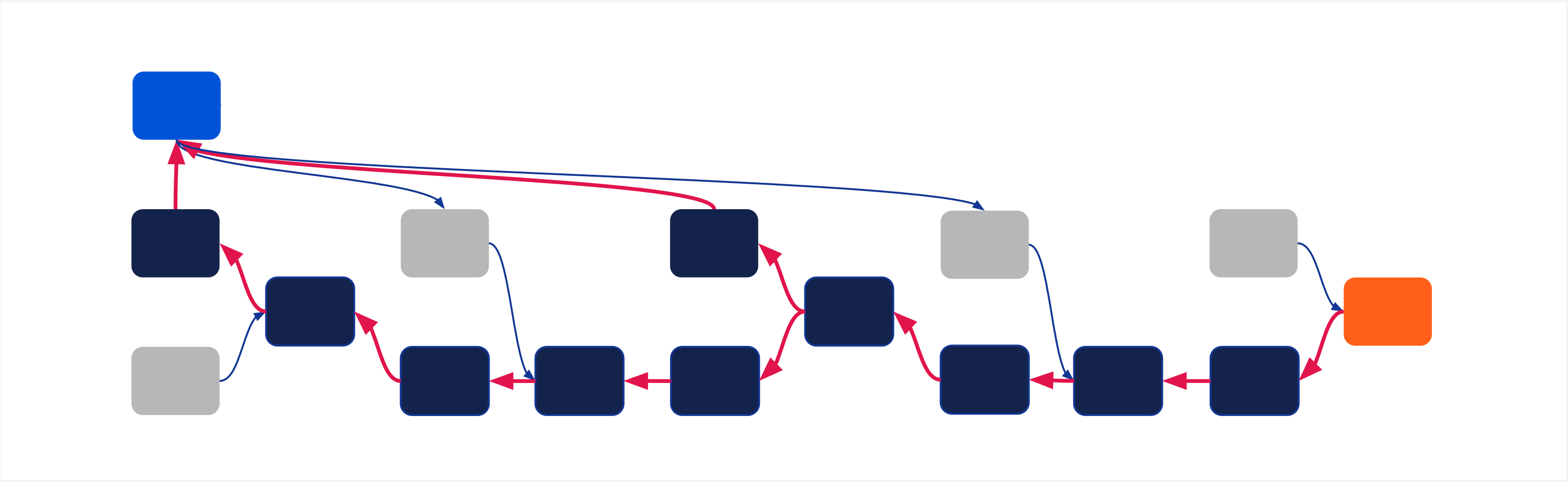
Chain rule, backprop and automatic differentiation



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

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

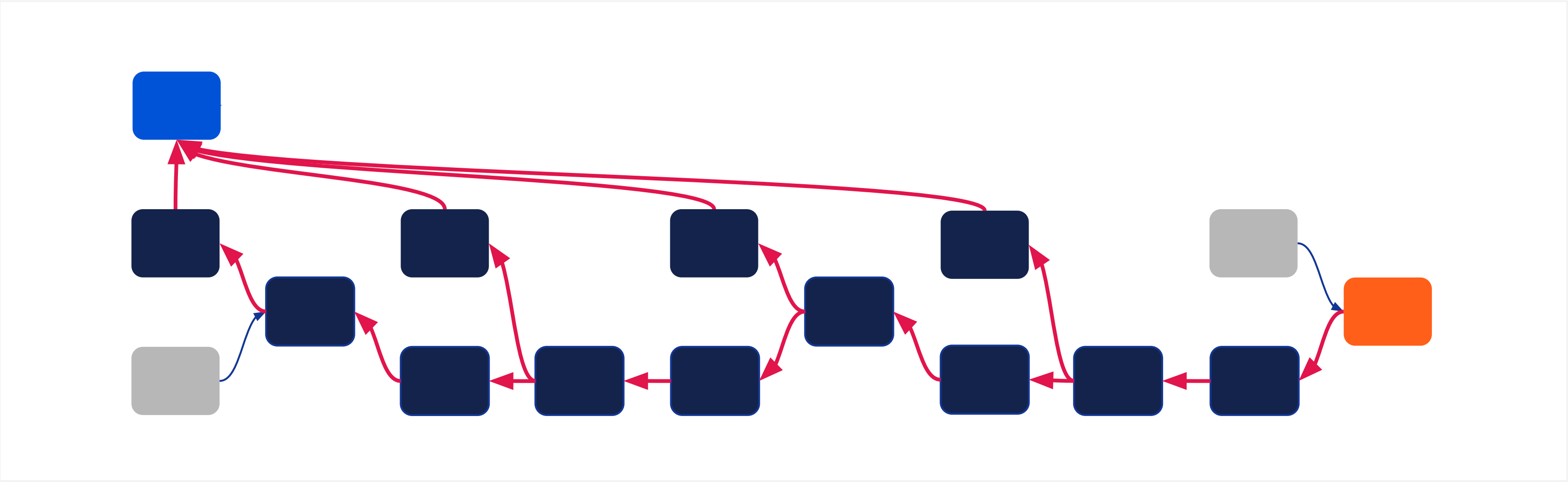
Chain rule, backprop and automatic differentiation



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

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

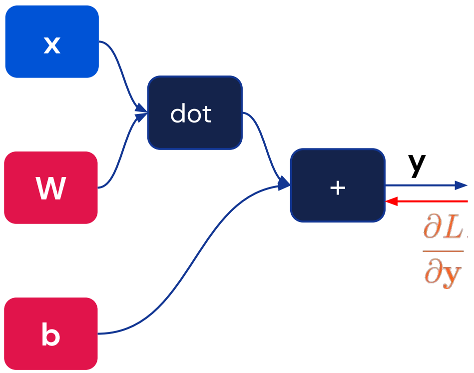
Chain rule, backprop and automatic differentiation



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

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

Linear layer as a computational graph



$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{W}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \mathbf{y}}\right)^{\mathrm{T}} \mathbf{x}^{\mathrm{T}}$$

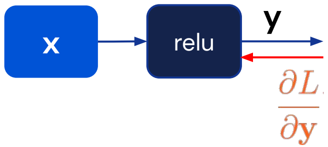
Symmetry between weights and inputs

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}}$$

Biases are adjusted proportional to error

Note that **backward pass** is a **computational graph** itself.

ReLU as a computational graph



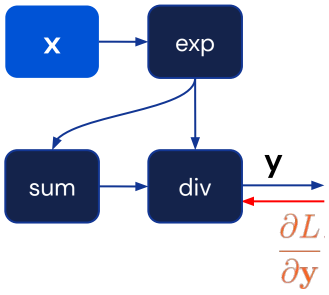
$$f_{\text{relu}}(\mathbf{x}) = \max(0, \mathbf{x})$$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{1}_{\mathbf{y}>0}$$

Can be seen as gating the incoming gradients. The ones going through neurons that were active are passed through, and the rest zeroed.

We usually put “gradient” at zero to be equal to zero.

Softmax as a computational graph



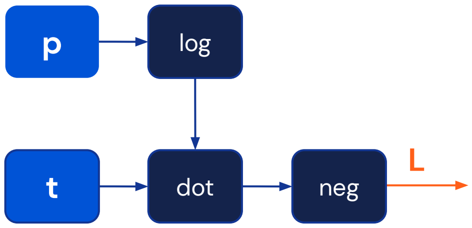
Since exponents of big numbers will cause overflow, it is rarely explicitly written like this.

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

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{x}_j} &= \sum_{i=1}^{m} \frac{\partial L}{\partial \mathbf{y}_i} \mathbf{y}_i(\delta_{ij} - \mathbf{y}_j) \\ &= \frac{\partial L}{\partial \mathbf{y}} - \mathbf{y} \sum_{i=1}^{m} \frac{\partial L}{\partial \mathbf{y}_i}\end{aligned}$$

Backwards pass is essentially
a difference between incoming gradient
and our output.

Cross entropy as a computational graph



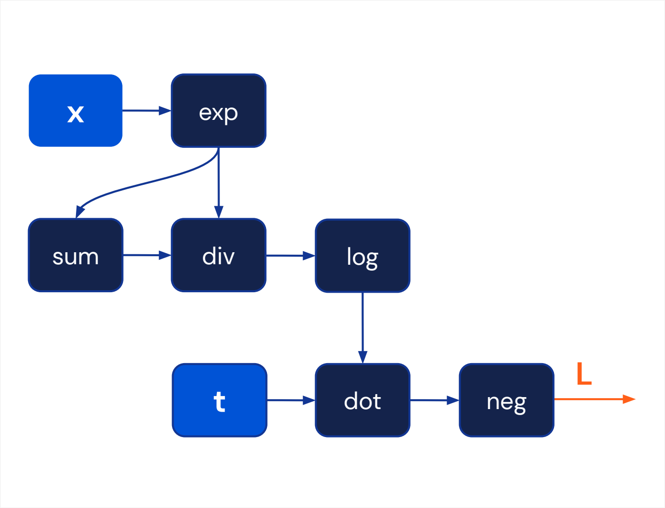
Even though it is a loss, we could still multiply its backwards by another incoming errors.

$$\ell_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -\mathbf{t}^{\mathrm{T}} \log \mathbf{p}$$

$$\frac{\partial L}{\partial \mathbf{p}} = -\mathbf{t} \oslash \mathbf{p} \longleftarrow$$ Dividing by **p** can be numerically unstable

$$\frac{\partial L}{\partial \mathbf{t}} = -\log \mathbf{p} \longleftarrow$$ We can also backprop into labels themselves

Cross entropy with logits as a computational graph



$$\ell_{\mathrm{CE}}(f_{\mathrm{sm}}(\mathbf{x}), \mathbf{t}) = -\sum_{j=1}^{k} \mathbf{t} \log f_{\mathrm{sm}}(\mathbf{x})$$

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{t} - \mathbf{x} \longleftarrow$$

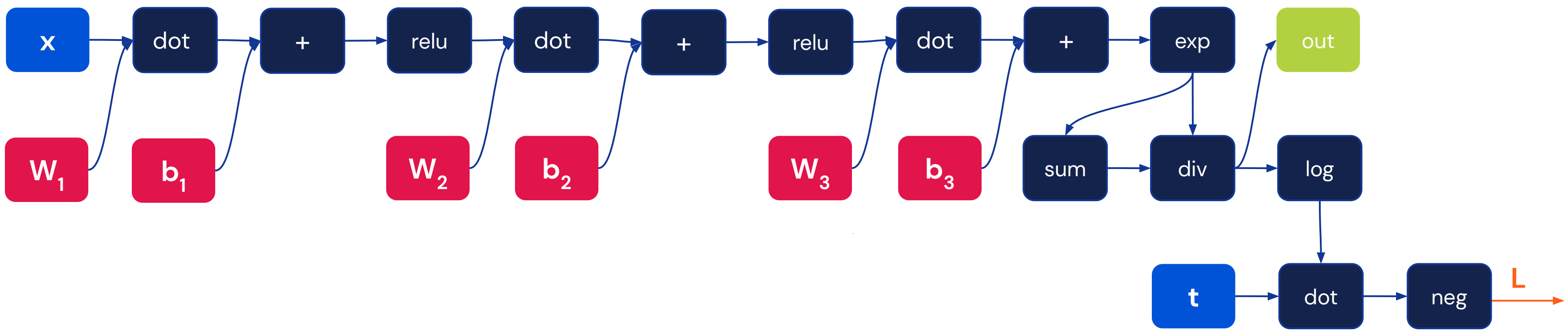
Simplifies extremely!

$$\frac{\partial L}{\partial \mathbf{t}} = -\log f_{\mathrm{sm}}(\mathbf{x})$$

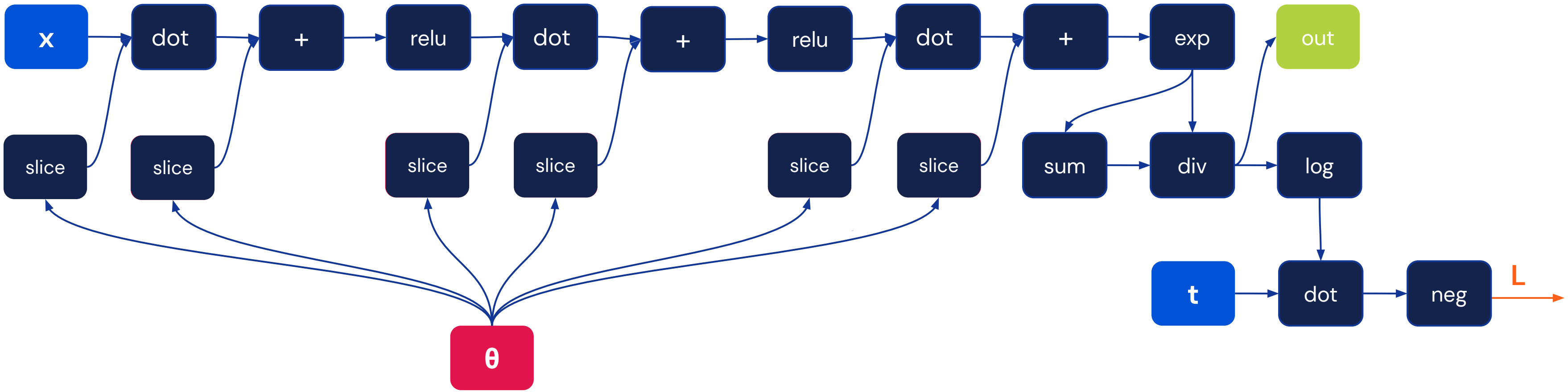
We can also backprop into labels themselves

For **numerical stability** it is usually a **single operation** in a computational graph.

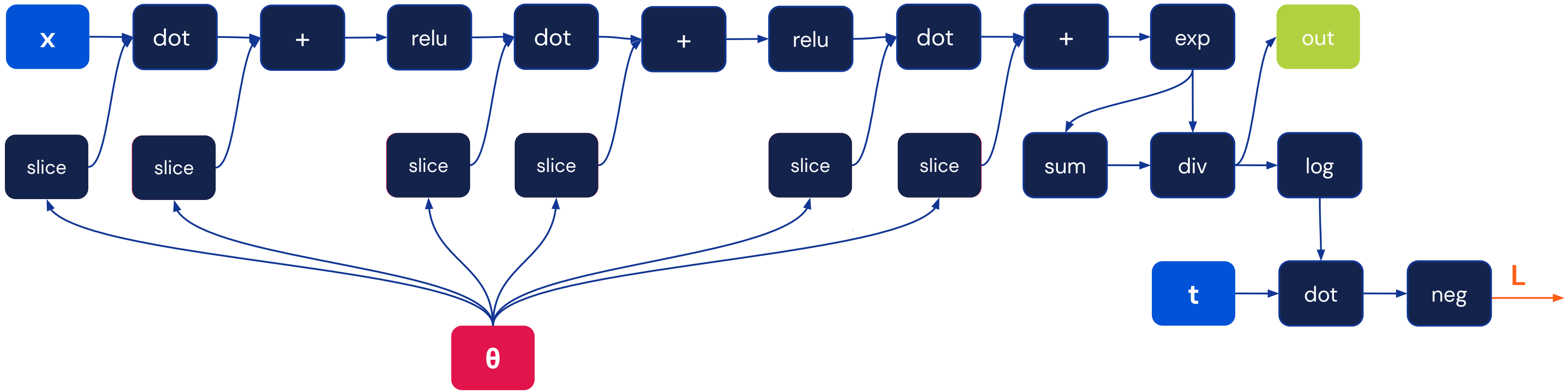
Example - 3 layer MLP with ReLU activations



Example - 3 layer MLP with ReLU activations



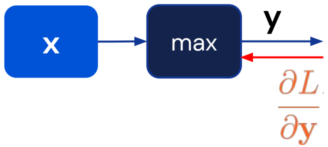
Example - 3 layer MLP with ReLU activations



$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t - \alpha_t \sum_i \nabla_{\boldsymbol{\theta}} \ell(g(\mathbf{x}^{(i)}, \boldsymbol{\theta}_t), \mathbf{t}^{(i)})$$

4 Pieces of the puzzle

Max as a computational graph



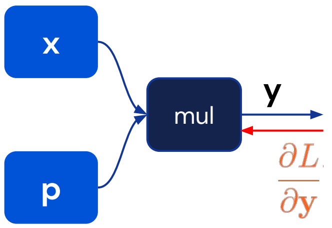
$$f_{\max}(\mathbf{x}) = \max_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{1}_{\mathbf{x} = f_{\max}(\mathbf{x})}$$

Gradients only flow through the selected element. Consequently we are not learning how to select.

Used in max pooling.

Conditional execution as a computational graph



$$f_{\mathrm{cond}}(\mathbf{x}, \mathbf{p}) = \mathbf{x} \odot \mathbf{p}$$

Let's assume **p** is probability distribution (e.g. one hot).

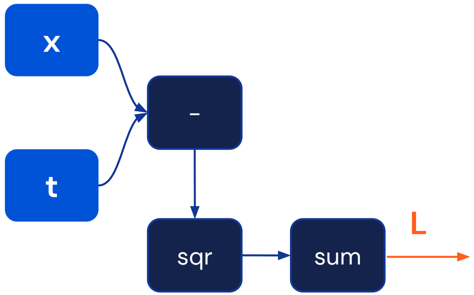
$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{p}^{\mathrm{T}}$$

Backwards pass is gated in the same way forward one is

$$\frac{\partial L}{\partial \mathbf{p}} = \frac{\partial L}{\partial \mathbf{y}} \odot \mathbf{x}^{\mathrm{T}}$$

We can learn conditionals themselves too, just use softmax.

Quadratic loss as a computational graph



$$\ell_2(\mathbf{x}, \mathbf{t}) = \|\mathbf{t} - \mathbf{x}\|^2$$

$$\frac{\partial L}{\partial \mathbf{x}} = 2(\mathbf{x} - \mathbf{t})^T \longleftarrow$$

Backwards pass is just a difference in predictions

$$\frac{\partial L}{\partial \mathbf{t}} = 2(\mathbf{t} - \mathbf{x})^T \longleftarrow$$

Learning targets is analogous

Typical loss for all **regression** problems (e.g. Value function fitting)

5 Practical issues

Overfitting and regularisation

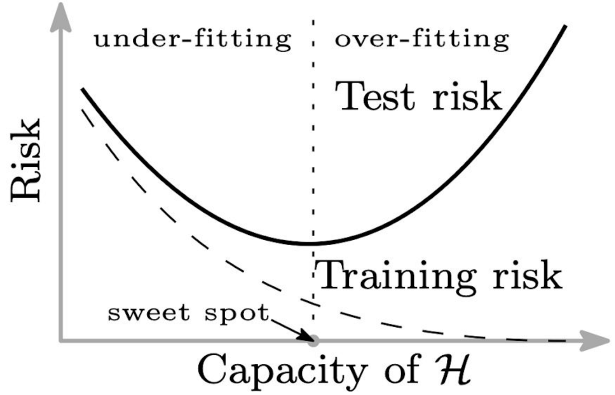


Figure from Belkin et al. (2019)

- L_p regularisation
- Dropout
- Noising data
- Early stopping
- Batch/Layer norm

Want to learn more?

Vapnik, Vladimir. **The nature of statistical learning theory**. Springer science & business media, (**2013**)

- As your model gets more powerful, it can create extremely complex hypotheses, even if they are not needed
- Keeping things simple guarantees that if the training error is small, so will the test be.

Classical results from statistics and **Statistical Learning Theory** which analyses the **worst case scenario**.

Overfitting and regularisation

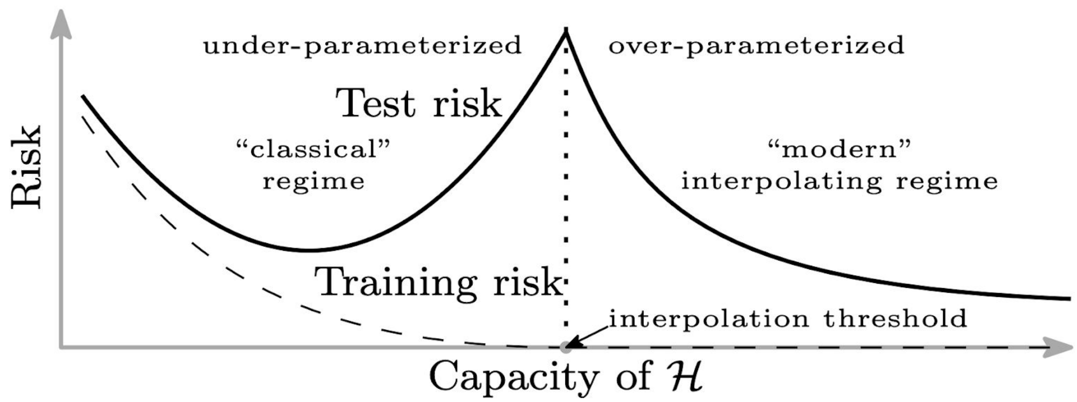


Figure from Belkin et al. (2019)

New results, that take into consideration learning effects.

Want to learn more?

Belkin, Mikhail, et al. **Reconciling modern machine-learning practice and the classical bias–variance trade-off.** Proceedings of the National Academy of Sciences 116.32 (**2019**)

- As models grow, their learning dynamics changes, and they become less prone to overfitting.
- New, exciting theoretical results, also mapping these huge networks onto Gaussian Processes.

Overfitting and regularisation

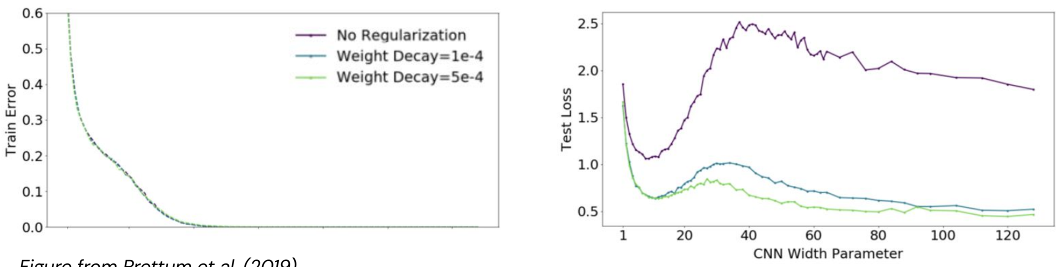


Figure from Prettum et al. (2019)

Model **complexity is not** as simple as **number of parameters.**

Want to learn more?

Nakkiran, Preetum, et al. **Deep double descent: Where bigger models and more data hurt.** arXiv preprint arXiv:1912.02292 (**2019**)

- Even big models still need (can benefit from) regularisation techniques.
- We need new notions of effective complexity of our hypotheses classes.

Diagnosing and debugging

- **Initialisation** matters
- **Overfit** small sample
- **Monitor** training **loss**
- **Monitor** weights **norms** and **NaNs**
- Add **shape asserts**
- Start with **Adam**
- **Change one thing** at the time

Want to learn more?

Karpathy A. **A Recipe for Training Neural Networks** http://karpathy.github.io/2019/04/25/recipe/ (2019)

- It is always worth spending time on verifying correctness.
- Be suspicious of good results more than bad ones.
- Experience is key, just keep trying!

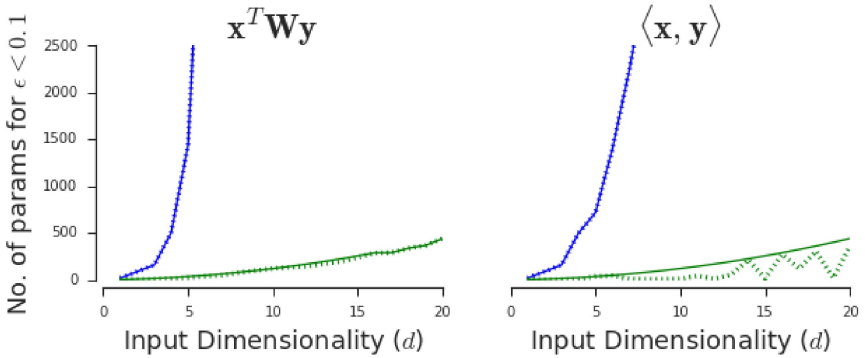
6 Bonus: Multiplicative interactions

What MLPs cannot do?

What MLPs cannot do?

$$f(\mathbf{x},\mathbf{z}) = \langle \mathbf{x},\mathbf{z} \rangle$$

Multiplicative interactions



$$f(\mathbf{x}, \mathbf{z}, \mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}) = \mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{z} + \mathbf{z}^{\mathrm{T}}\mathbf{U} + \mathbf{V}\mathbf{x} + \mathbf{b}$$

Being able to **approximate** something is not the same as **represent it**.

Want to learn more?

Siddhant M. Jayakumar et al. **Multiplicative Interactions and Where to Find Them** Proceedings of International Conference on Learning Representations (**2019**)

- Multiplicative units unify attention, metric learning and many others
- They enrich the hypothesis space of regular neural networks in a meaningful way

If you want to do research in fundamental building blocks of Neural Networks, **do not seek** to marginally improve the way they behave by finding **new activation function.**

Ask yourself what current modules cannot represent or guarantee right now, and propose a module that can.

Thank you

Questions