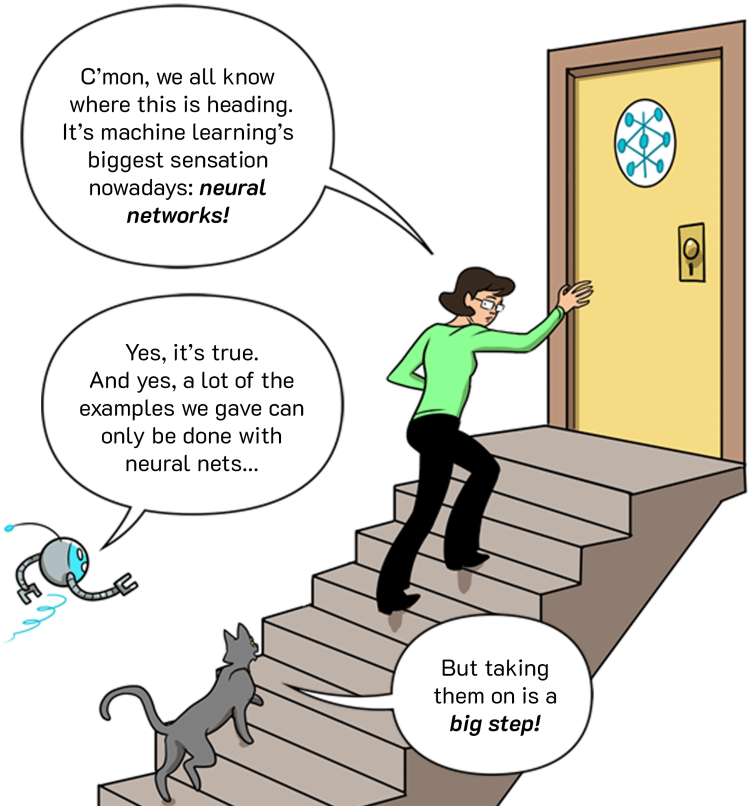
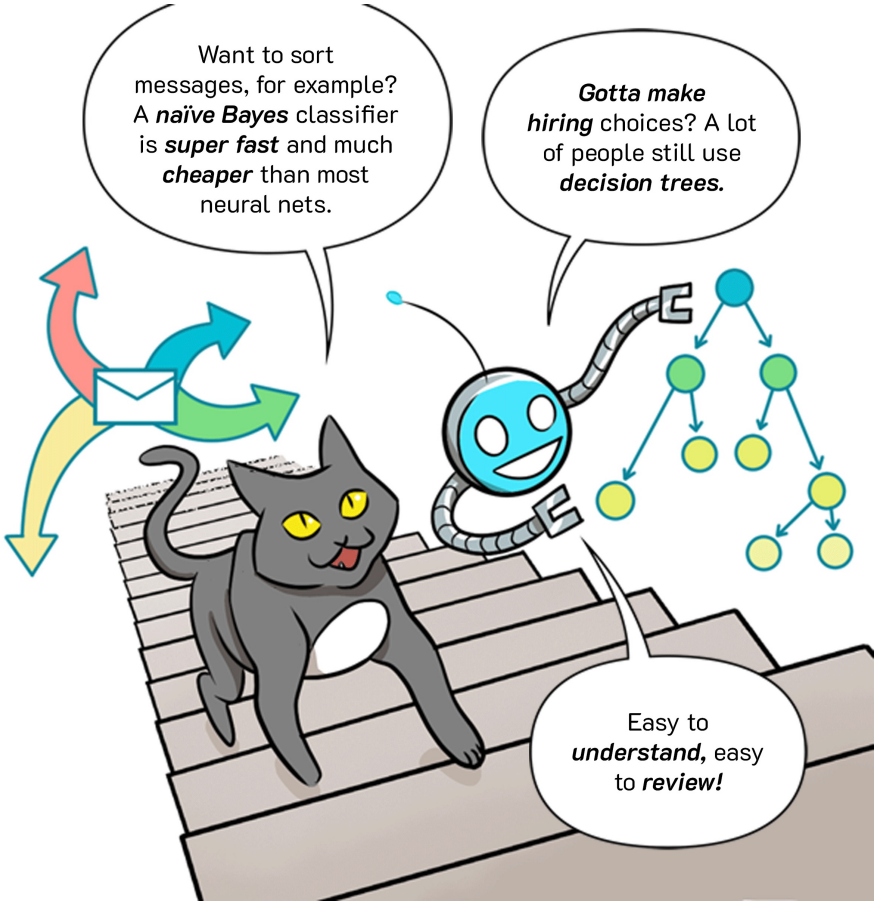
Introduction to
Artificial Neural Networks

Andrea Santamaría García

Laboratory for Applications of Synchrotron Radiation (KIT-LAS)

07/09/2022





Comic source: https://cloud.google.com/products/ai/ml-comic-1

Andrea Santamaría García

Comic source: https://cloud.google.com/products/ai/ml-comic-1

UNIVARIATE LINEAR REGRESSION (one prediction)

Simple (one feature)

We want to fit linearly a set of points $(x_i,\ y_i)$

Hypothesis function: $h_\theta(x_i) = \theta_0 + \theta_1 x_i$

Estimated from data — **weights**

- regression coefficients
- parameters of the model

$h: x \rightarrow y$

Space of input variables = **feature**

independent variable

continuous: regression problem
discrete: classification problem

Space of output variables = **estimated value**

- dependent variable
- scalar response

Multiple (several features)

$$h_\theta(x) = \theta_0 + \sum_{i,\,k=1}^{n,\,p} \theta_k \phi_k(x_i)$$

bias

- $i = 1, \dots, \boldsymbol{n}$ data points
- $k = 1, \dots, \boldsymbol{p}$ features
- $x_{i0} = 1$ pseudo-variable
- $\phi_k(x_i) = x_{ik}$ basis function

Polynomial

$\phi(x) = (x, x^2, \cdots, x^p)$

⚠ Fits a nonlinear model to the data, but it's still linear in the parameters θ of the model

Matrix notation $h_\theta(x) = \theta^T x = x^T \theta = \boldsymbol{X\theta}$

$[n \times (p+1)] \times [(p+1) \times 1]$ bias included

UNIVARIATE LINEAR REGRESSION

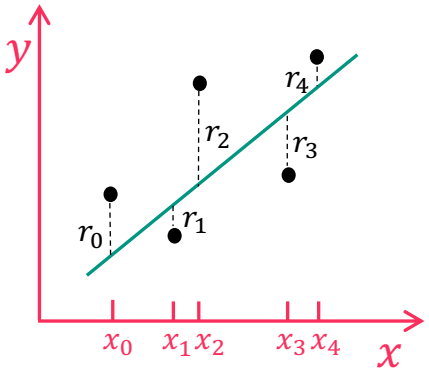
Loss (cost) function

Goal: choose θ_k such that $h_\theta(x_i)$ is as close to y_i as possible $\quad h_\theta(x_i) - y_i = r_i$

as small as possible
= minimization problem

residual

error/disturbance/noise



Assumption:
observations $(x_i,\ y_i)$ = result of random deviations

normal

Least Squares

Reasonable choice that works well for many regression problems (desirable properties)

Find the analytical minimum of:

$$\sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (h_\theta(x_i) - y_i)^2 = |\boldsymbol{y} - \boldsymbol{X\theta}|^2 = J(\theta_k)$$

loss (cost) function

- Takes an average difference of the predictions of the hypothesis
- "Intuitive" number to represent deviation from target
- **Measures of performance of a model**

UNIVARIATE LINEAR REGRESSION

Loss functions - least squares

Maximizes the likelihood of the linear regression model

$$\sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (h_\theta(x_i) - y_i)^2 = |\boldsymbol{y} - \boldsymbol{X\theta}|^2 = J(\theta_k)$$

Ordinary least squares

- Convex problem that has a unique closed-form solution

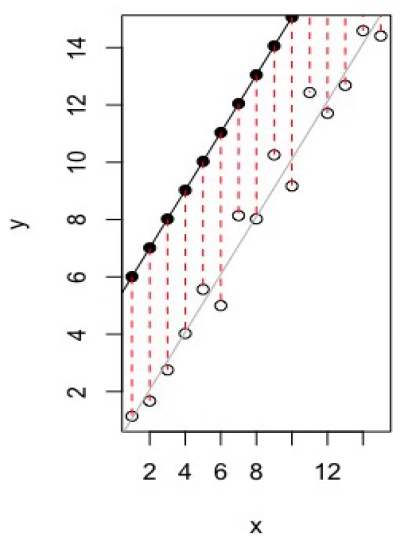
Nonlinear least squares

- Solved iteratively, where each iteration is approximated by a linear one

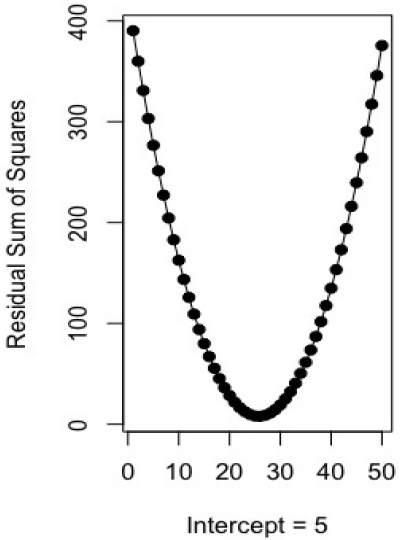
⚠ Use least squares when:

- System is overdetermined
  - Points > features $(n > p + 1)$
- Uncertainties in the data are "controlled"
  - ➢ Otherwise: maximum likelihood estimation, ridge regression, lasso regression, least absolute deviation, bayesian linear regression, etc.

Scan of hypothesis



Calculation of loss



Andrea Santamaría García

Animation from: https://yihui.org/animation/example/least-squares/

OPTIMIZERS
Example: gradient descent

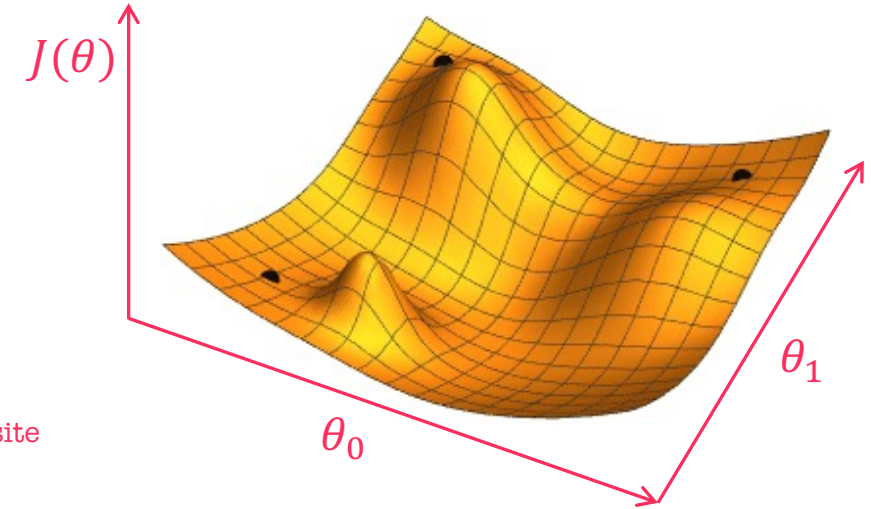
Algorithm to iteratively solve $argmin_\theta J(\theta_k)$

first order

$$\vec{\theta} := \vec{\theta} - \alpha \nabla J(\theta_k)$$

step size
learning rate

take repeated steps in the opposite direction of the gradient



$$J(\theta_k) = \frac{1}{2n}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y_i)$$

$$\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_1} = \frac{1}{n}\sum_{i=1}^{n} x_i(\theta_0 + \theta_1 x_i - y_i)$$

$$\theta_0 := \theta_0 - \alpha \frac{\partial J(\theta_0,\theta_1)}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial J(\theta_0,\theta_1)}{\partial \theta_1}$$

updated simultaneously

- Depending on the initial (θ_0, θ_1) optimization can end up at different points
- If the loss function is not strictly convex and saddle points exist finding the global minimum is not guaranteed
- Works in any number of dimensions

- Iteratively (this example): $\mathcal{O}(features \cdot points^2)$
- Analitycally (normal equation): $\mathcal{O}(points^3)$

matrix inversion

Andrea Santamaría García

Animation from Wikimedia commons

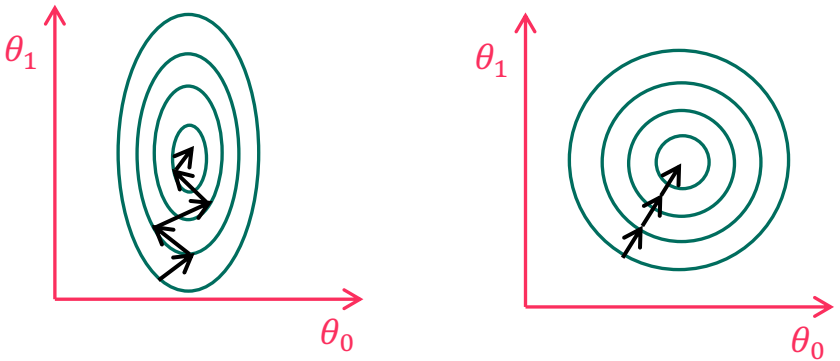
OPTIMIZERS

Feature scaling

- Optimizers can converge faster if the features are on a similar scale
- Becomes very important in polynomial regression:

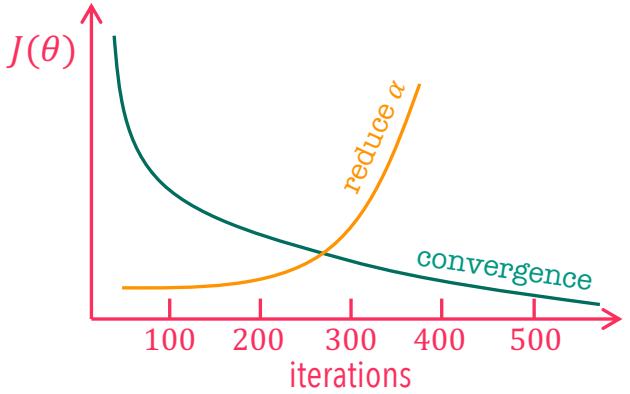
$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

- Normalization: $-1 \leq x_i \leq 1$ or $0 \leq x_i \leq 1$
- Standardization: $\mu = 0,\ \sigma^2 = 1$



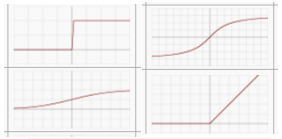
Learning rate

- The optimizer is working correctly if the loss decreases after every iteration
- Usual to **declare convergence tests** (e.g., declare convergence when $J(\theta)$ decreases by less than 10^{-3} in one iteration)
  - α too small: slow convergence
  - α too large: may not converge



NEURAL NETWORKS (NNs)

Perceptron



activation function

- What makes NNs different from linear regression
- Defines the output of the neuron given an input
- "**Mathematical gate**" that filters which data passes to the next layer of neurons (should the neuron fire given this input?)

bias variable that captures the invariant part of the prediction

x_0, x_1, x_2, x_3 with weights θ_0, θ_1, θ_2, θ_3

neuron

computational unit

1. **Weighted sum**

$$z = \theta_0 + \sum_{i,k=1}^{n,p} \theta_k x_{ik} = \boldsymbol{\theta X}$$

2. **Activation function**

$$g(z)$$

$$g(z) = h_\theta(x) = \hat{y}$$ prediction

$$\boldsymbol{X} = \begin{bmatrix} x_{10} & \dots & x_{n0} \\ \vdots & \ddots & \vdots \\ x_{1p} & \dots & x_{np} \end{bmatrix} = \begin{bmatrix} x_0 \\ \vdots \\ x_p \end{bmatrix}$$

n points, p features

we don't know how the data points n will be fed at a time (batches)

Universal approximation theorem

When the activation function is nonlinear, then a two layer NN can be proven to be a universal function approximator

NEURAL NETWORKS (NNs)

Perceptron

bias

variable that captures the invariant part of the prediction

x_0

x_1

x_2

x_3

θ_0

θ_1

θ_2

θ_3

weights and biases are randomly initialized

neuron

computational unit

1. Weighted sum

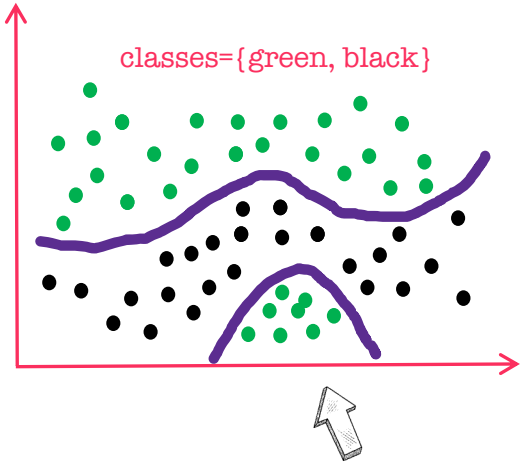
$z = \boldsymbol{\theta X}$

2. Activation function

$g(z)$

$g(z) = \hat{y}$

introduces nonlinearities to approximate arbitrarily complex functions



discrete: classification problem

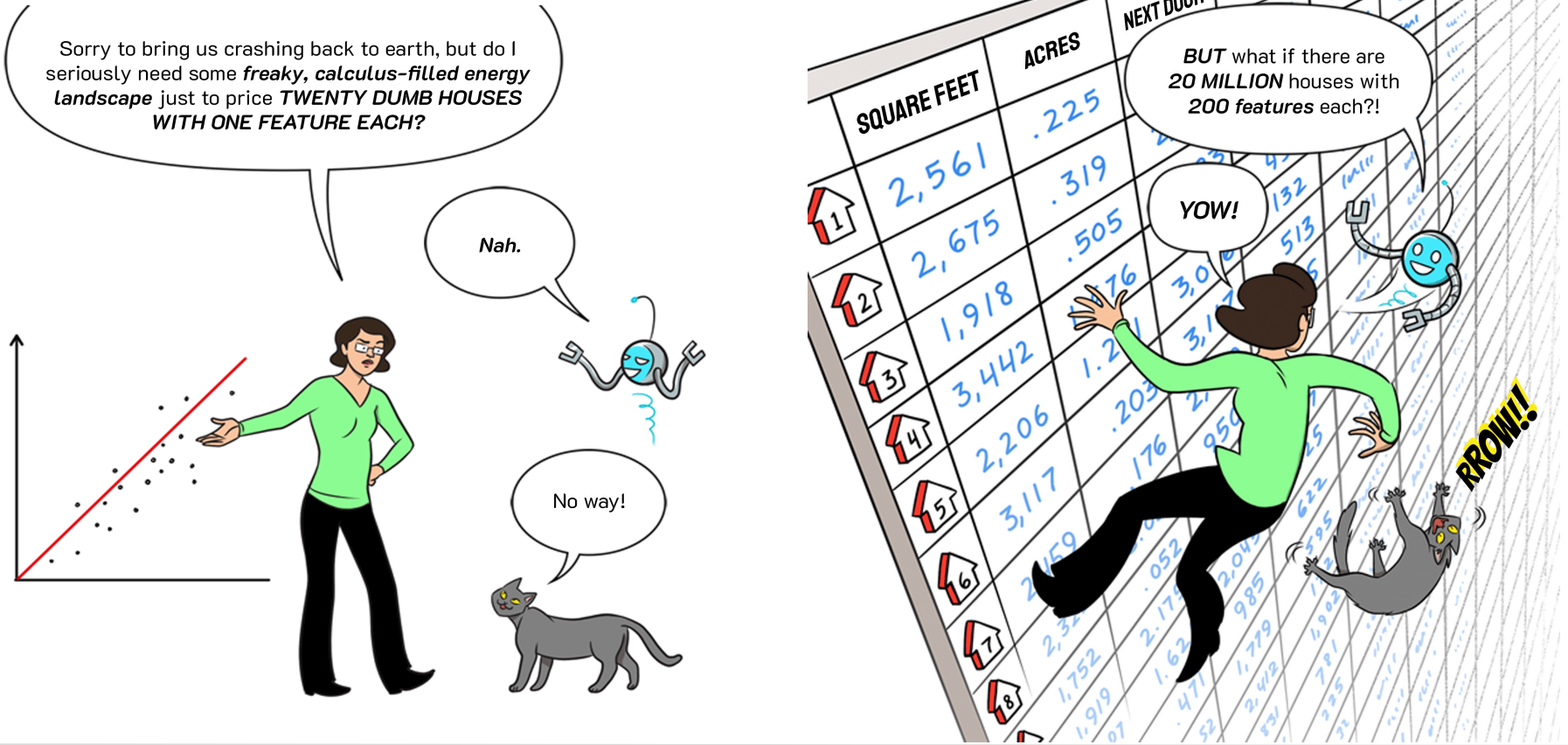
probabilities between 0 and 1 of belonging to a specific class (logistic regression)

continuous: regression problem

$\boldsymbol{\theta}^* = argmin_\theta J(\boldsymbol{\theta})$

iteratively update the weights

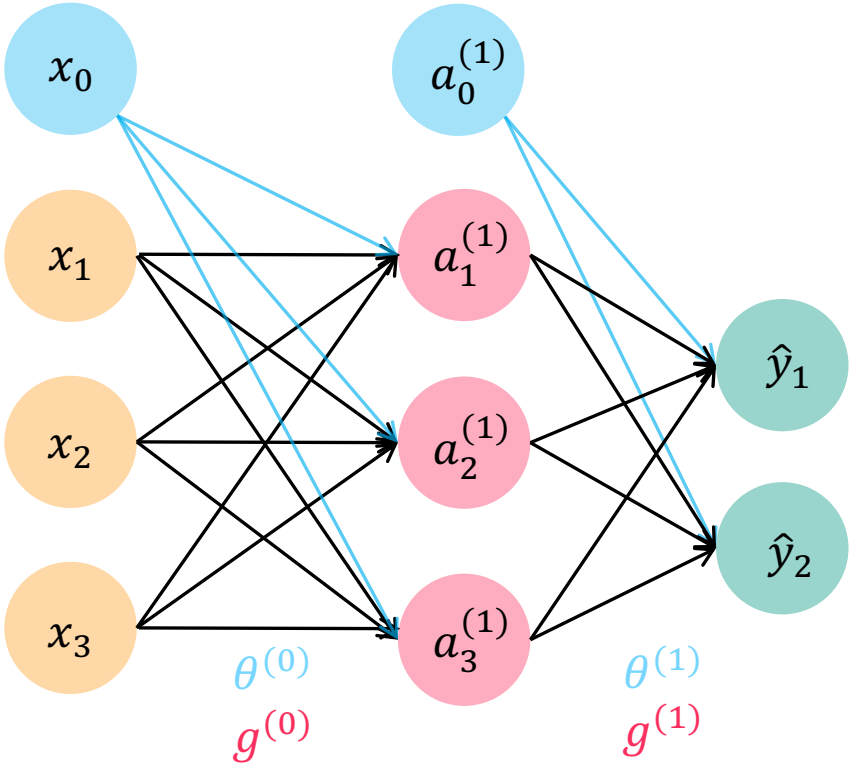
COMPUTATIONALLY EFFICIENT FOR BIG DATA!



Comic source: https://cloud.google.com/products/ai/ml-comic-1

FORWARD PROPAGATION

Single layer neural network



New notation:

- **Superscript** = layer number
- **Subscript** = neuron number
- a = activation vector or unit
- g = activation function

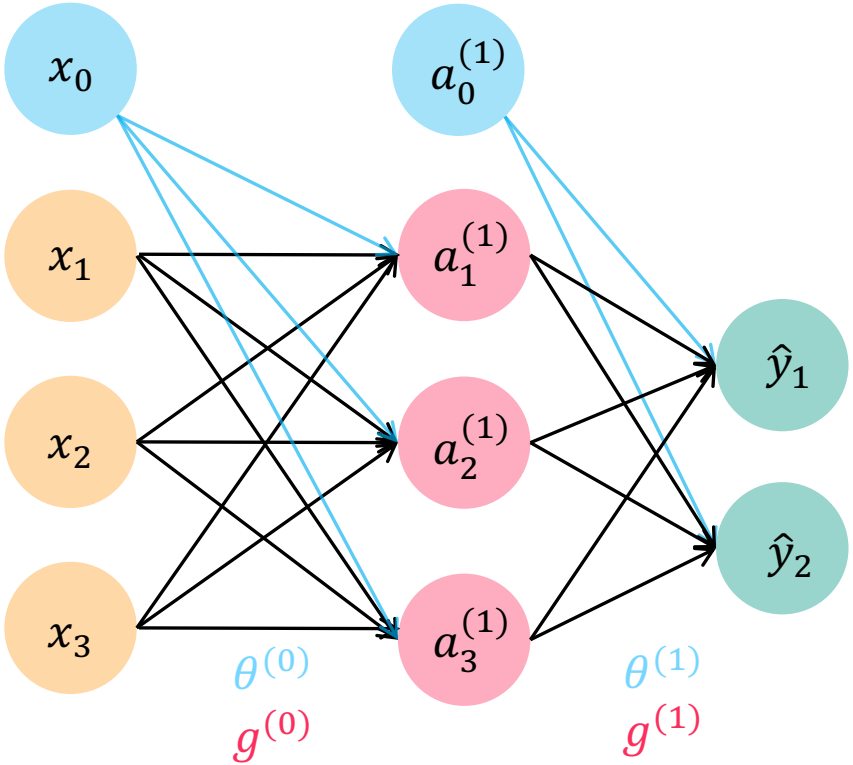
$$\hat{y} = g^{(1)}(\boldsymbol{\theta}^{(1)} g^{(0)}(\boldsymbol{\theta}^{(0)} \boldsymbol{X}))$$

$z^{(2)}$ spans $\boldsymbol{\theta}^{(1)} g^{(0)}(\boldsymbol{\theta}^{(0)} \boldsymbol{X})$; $\boldsymbol{a}^{(1)}$ spans $g^{(0)}(\boldsymbol{\theta}^{(0)} \boldsymbol{X})$; $z^{(1)}$ spans $\boldsymbol{\theta}^{(0)} \boldsymbol{X}$

multilayer network evaluates compositions of functions computed at individual neurons

FORWARD PROPAGATION

Single layer neural network



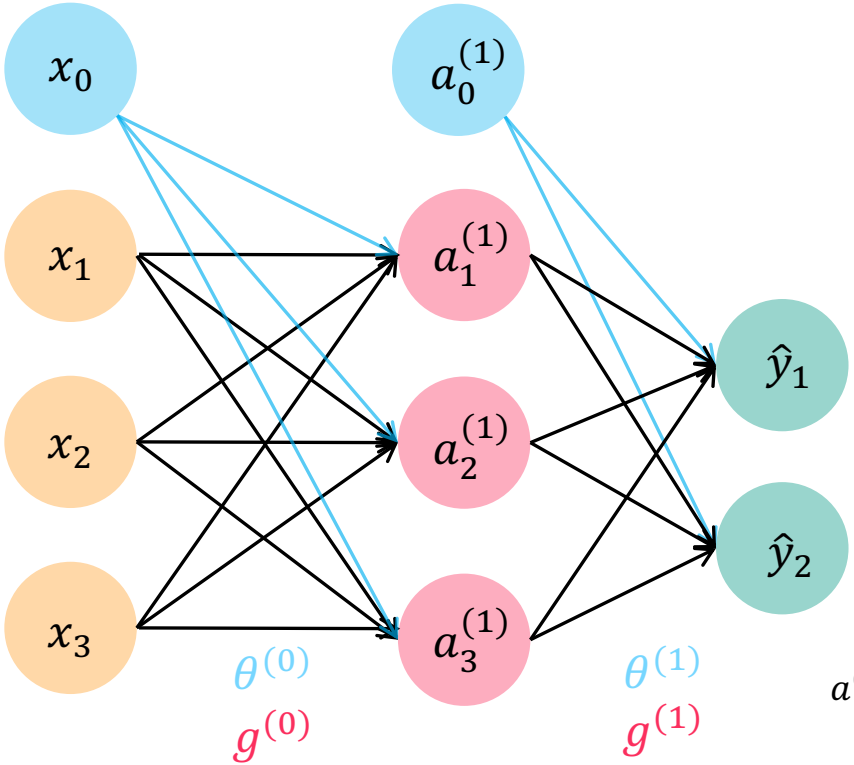
$$z^{(2)} = a^{(1)} = \hat{y} = g^{(1)}(\boldsymbol{\theta}^{(1)} g^{(0)}(\underbrace{\boldsymbol{\theta}^{(0)} \boldsymbol{X}}_{z^{(1)}}))$$

$$\boldsymbol{\theta}^{(0)} = \underbrace{\begin{bmatrix} \theta_{10}^{(0)} & \theta_{11}^{(0)} & \theta_{12}^{(0)} & \theta_{13}^{(0)} \\ \theta_{20}^{(0)} & \theta_{21}^{(0)} & \theta_{22}^{(0)} & \theta_{23}^{(0)} \\ \theta_{30}^{(0)} & \theta_{31}^{(0)} & \theta_{32}^{(0)} & \theta_{33}^{(0)} \end{bmatrix}}_{\text{neurons in input layer + bias}} \Big\} \text{neurons in layer 2} \qquad \boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_0^{(1)} \\ g^{(0)}(\theta_{10}^{(0)} x_0 + \theta_{11}^{(0)} x_1 + \theta_{12}^{(0)} x_2 + \theta_{13}^{(0)} x_3) \\ g^{(0)}(\theta_{20}^{(0)} x_0 + \theta_{21}^{(0)} x_1 + \theta_{22}^{(0)} x_2 + \theta_{23}^{(0)} x_3) \\ g^{(0)}(\theta_{30}^{(0)} x_0 + \theta_{31}^{(0)} x_1 + \theta_{32}^{(0)} x_2 + \theta_{33}^{(0)} x_3) \end{bmatrix}$$

FORWARD PROPAGATION

Single layer neural network



$$\hat{y} = g^{(1)}(\boldsymbol{\theta}^{(1)} g^{(0)}(\underbrace{\boldsymbol{\theta}^{(0)}\boldsymbol{X}}_{z^{(1)}})) \quad \overbrace{\quad}^{a^{(1)}} \quad \overbrace{\quad\quad}^{z^{(2)}}$$

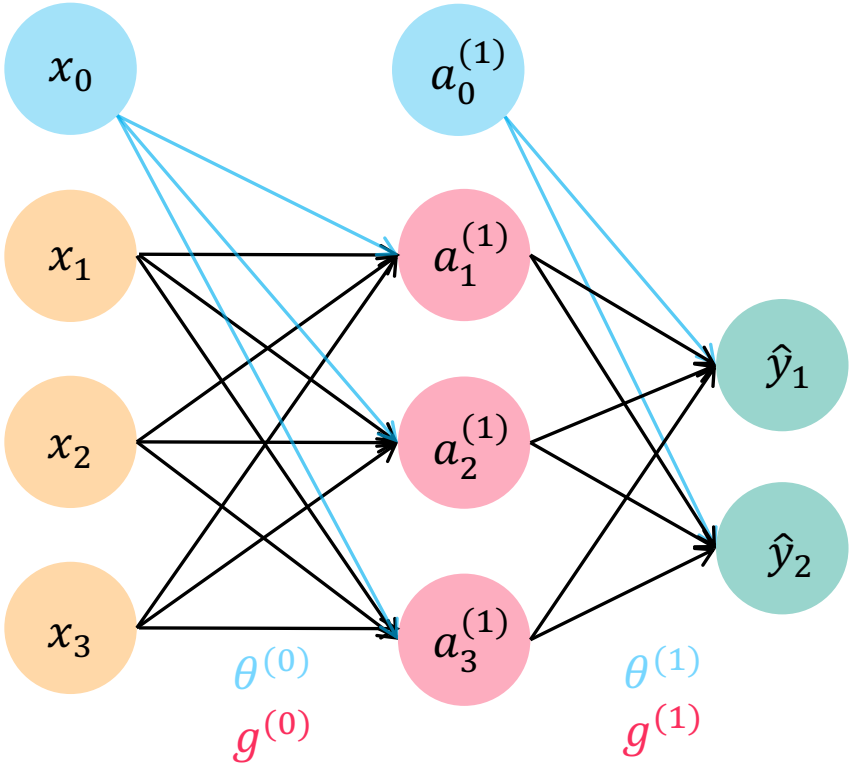
$$\theta^{(1)} = \underbrace{\begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \end{bmatrix}}_{\text{neurons in hidden layer + bias}} \text{ neurons in output layer}$$

$$a^{(1)} = \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix}$$

$$a^{(2)} = \hat{y} = \begin{bmatrix} g^{(1)}(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)}) \\ g^{(1)}(\theta_{20}^{(1)} a_0^{(1)} + \theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)}) \end{bmatrix}$$

BACKPROPAGATION

Single layer neural network



We want to find the network weights that achieve the lowest loss

$\boldsymbol{\theta}^* = argmin_\theta J(\boldsymbol{\theta})$ $\quad \boldsymbol{\theta} = \{\theta^{(1)}, \theta^{(2)}\}$

1. Gradient calculation:

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta^{(1)}} = g'^{(1)}(\theta^{(1)} a^{(1)}) a^{(1)}$$

$$J(\theta) = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(\hat{y}^2 + y^2 + 2\hat{y}y)$$ least squares

$$= \hat{y} - y$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta^{(0)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial \theta^{(0)}} = g'^{(0)}(\theta^{(0)} X) X$$

$$= g'^{(1)}(\theta^{(1)} a^{(1)})\ \theta^{(1)}$$

2. Weight update:

$$\theta^{(0)} := \theta^{(0)} - \alpha \frac{\partial J(\theta^{(0)}, \theta^{(1)})}{\partial \theta^{(0)}}$$

$$\theta^{(1)} := \theta^{(1)} - \alpha \frac{\partial J(\theta^{(0)}, \theta^{(1)})}{\partial \theta^{(1)}}$$

gradient descent

NEURAL NETWORKS
Activation functions

Differentiable, quickly converging wrt the weights

* There are also radial basis functions (RBF) which are efficient as universal function approximators (Gaussian, multiquadratics)

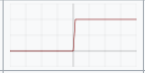
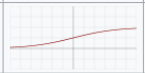
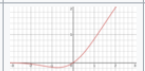
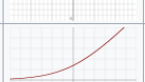
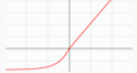
Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$
Identity		x	1
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$	$g(x)(1-g(x))$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1-g(x)^2$
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0,x) = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$
Gaussian Error Linear Unit (GELU)[5]		$\frac{1}{2}x\left(1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$
Softplus[9]		$\ln(1+e^{x})$	$\frac{1}{1+e^{-x}}$
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha\left(e^{x}-1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^{x} & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$
Scaled exponential linear unit (SELU)[11]		$\lambda\begin{cases} \alpha(e^{x}-1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^{x} & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[12]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$

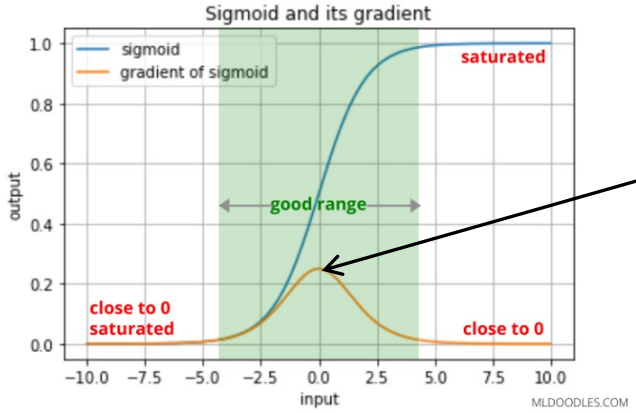
Image from wikipedia

NEURAL NETWORKS

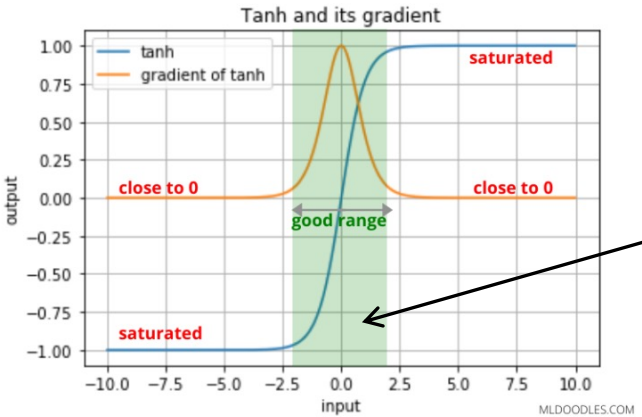
Vanishing gradients

appear in backpropagation using gradient-based methods in deep networks

not goodin hidden layers



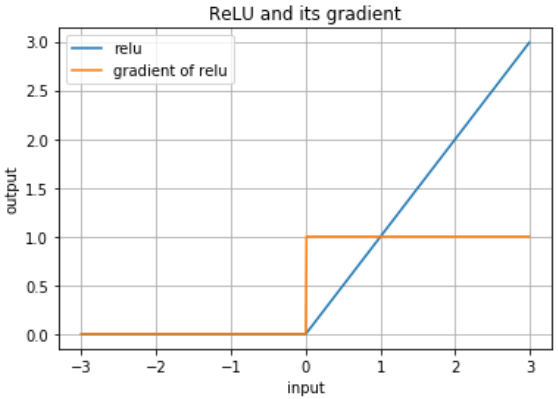
Maximum of gradient 0.25
With chain rule the gradient product can become very small



When the partial derivative vanishes the weights are not updated anymore

Very narrow range, small values

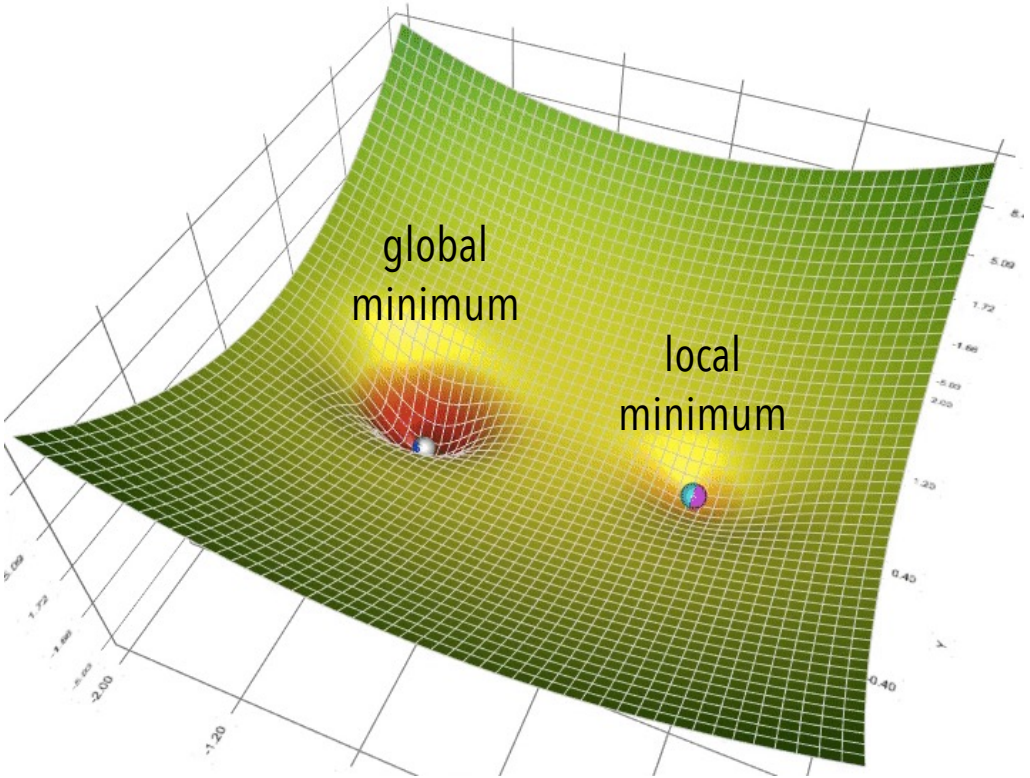
good for hidden layers



Andrea Santamaría García

Images from mldoodles

NOTE ON OPTIMIZERS IN NNs



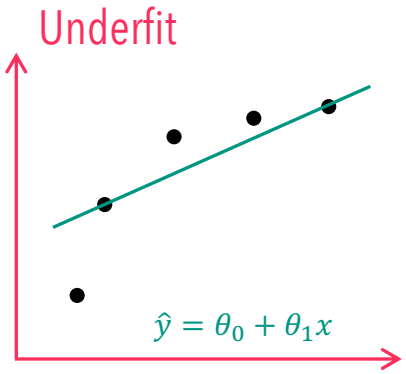
Gradient based methods (first order):

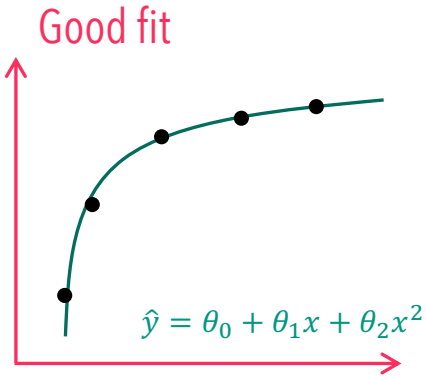
- **Gradient descent**
- **momentum**
- **AdaGrad (white)**
- **RMSProp**
- **Adam**

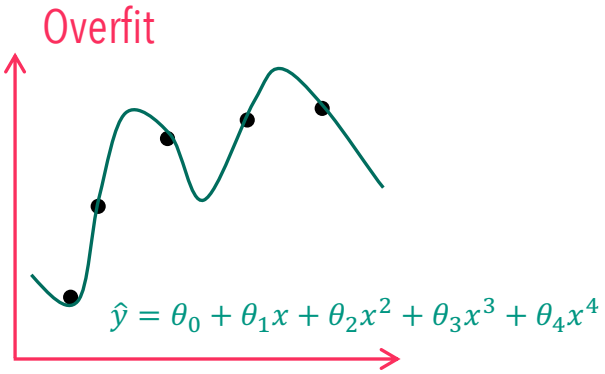
} dynamic adjustment of algorithm parameters

Animation from Lili Jiang

NOTE ON OVERFITTING







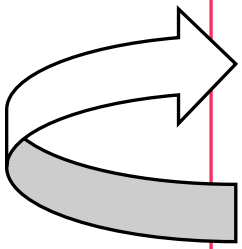
Test for overfitting with independent test dataset
→ Difference in loss = generalization error

Regularization

Improves generalization on useen data by constraining the optimization problem to discourage complex models

- Regularization term in **loss function** (penalty or shrinkage term, L1, L2)
- **Dropout**: not relying on certain nodes that only learn certain patterns (e.g., set 50% of activations to 0 during training)
- **Early stopping**: stop training before overfitting
- **Batch normalization**: normalizing each activation value using the batch (μ, σ)

SO WHAT'S THE RECIPE?



1. Select data features + perform data scaling
2. Choose network architecture
3. Choose activation function for each layer
4. Choose loss function & convergence criteria
5. Choose an optimizer & set its hyperparameters
6. Choose number of epochs to train
7. Decide on regularization techniques
8. Perform forward propagation
9. Compute gradients with backwards propagation
10. Update weights & keep track of loss

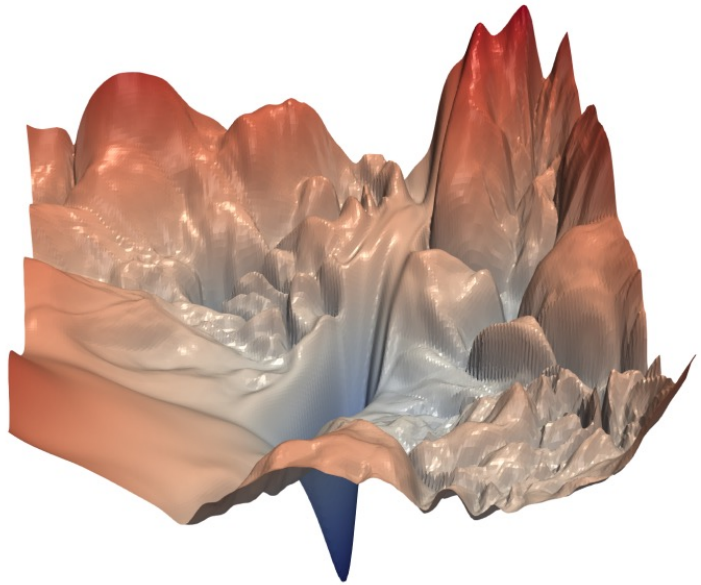
Let's try it!! → https://playground.tensorflow.org

Thank you for your attention!

What questions do you have for me?

https://ml-cheatsheet.readthedocs.io/

https://buildmedia.readthedocs.org/media/pdf/ml-cheatsheet/latest/ml-cheatsheet.pdf



Highly non-convex loss of ResNet-56 without skip connections [arxiv:1712.09913]

Let's connect! andrea.santamaria@kit.edu / @ansantam

All icons from this talk from TheNounProject

NOTE ON LOSS FUNCTION FOR CLASSIFICATION

Maximum likelihood estimation (MLE)

- $\vec{x}$ = random sample from unknown joint probability distribution $p(\vec{y}|\vec{x};\vec{\theta})$ ——— probability of y given x and θ (conditional probability)
- $\vec{\theta}$ = parametrization of p
- Optimal $\vec{\theta}$ value can be estimated by maximizing a likelihood function:

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p\,(y_i|x_i;\theta_i) \quad \rightarrow \quad argmax_\theta\, \mathcal{L}(\theta)$$

for independent observations

Most NNs use the negative log-likelihood as loss function: $J(\theta) = -\ln(\mathcal{L}(\theta))$ cross-entropy between the training data and model distribution

Thanks to machine-learning algorithms,
the robot apocalypse was short-lived.

https://www.smbc-comics.com/comic/rise-of-the-machines