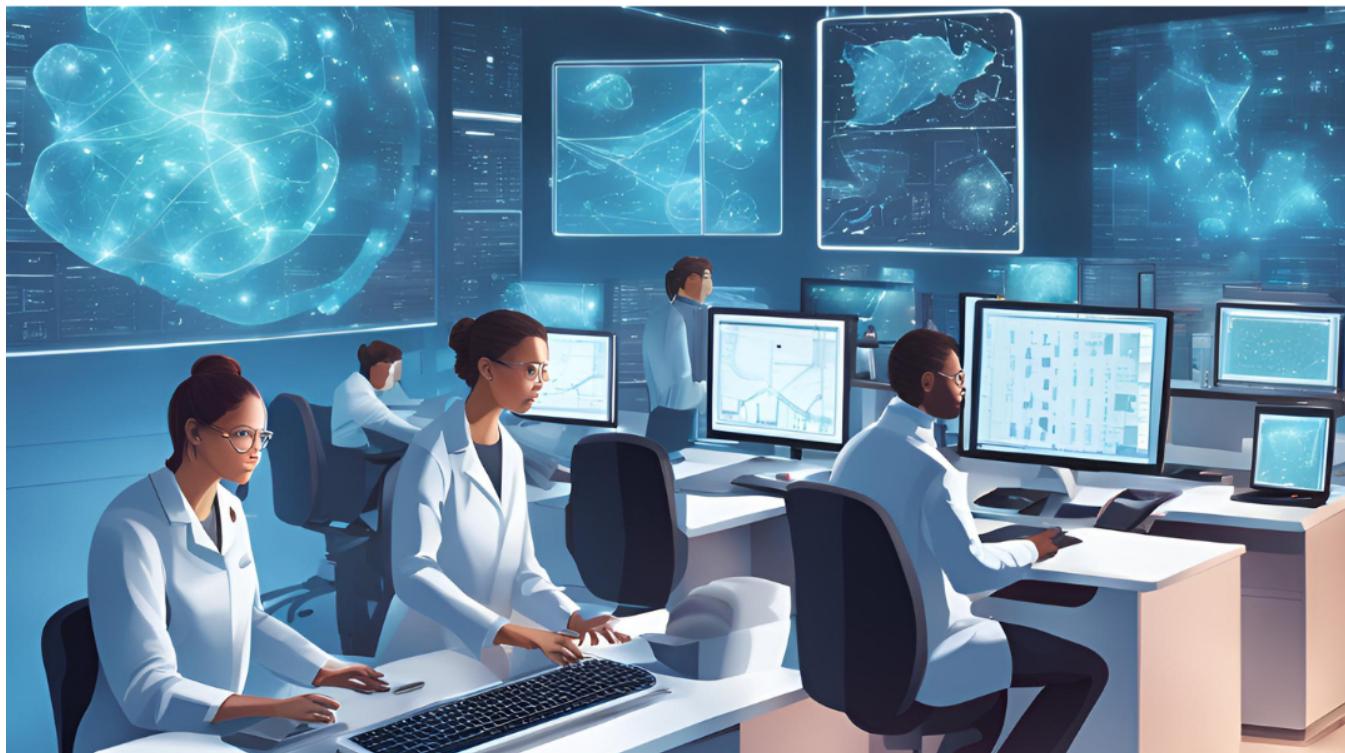


# AI for Genomics: from CNNs and LSTMs to Transformers

Nikolay Oskolkov, Group Leader (PI) at LIOS, Riga, Latvia  
Physalia course, 09.09.2025



@NikolayOskolkov



@oskolkov.bsky.social



Personal homepage:  
<https://nikolay-oskolkov.com>

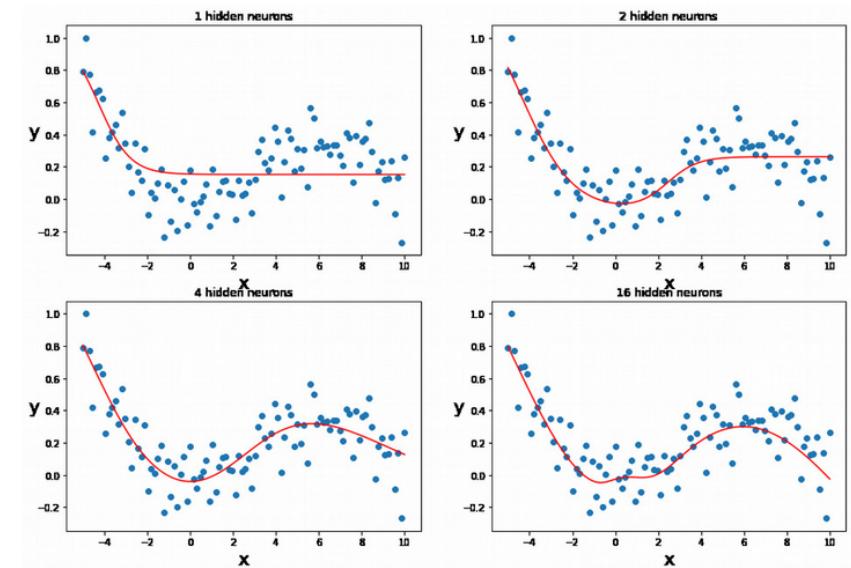
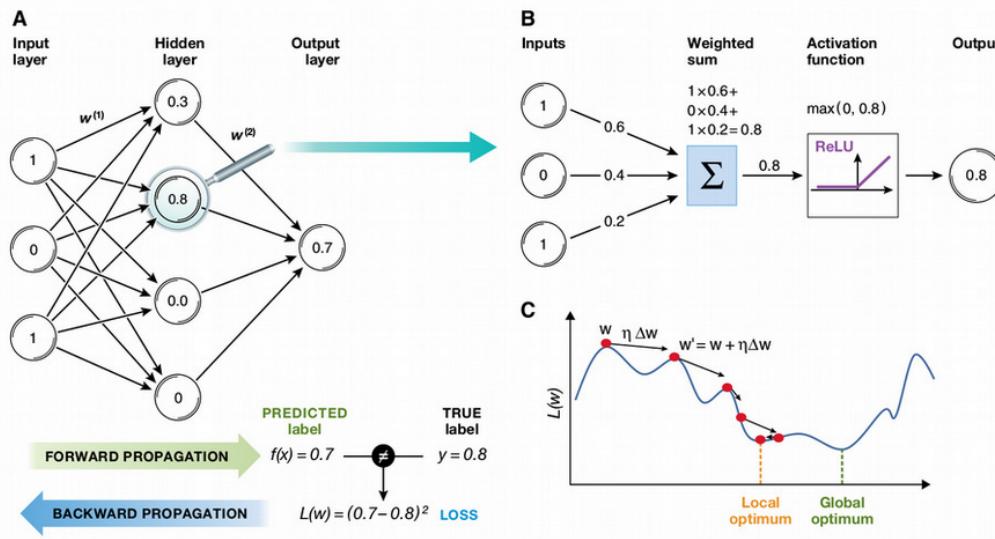
Topics we'll cover in this session:

- 1) Basics of artificial neural networks: gradient descent
- 2) Universal Approximation Theorem (UAT)
- 3) Coding vanilla neural network from scratch in R and Python

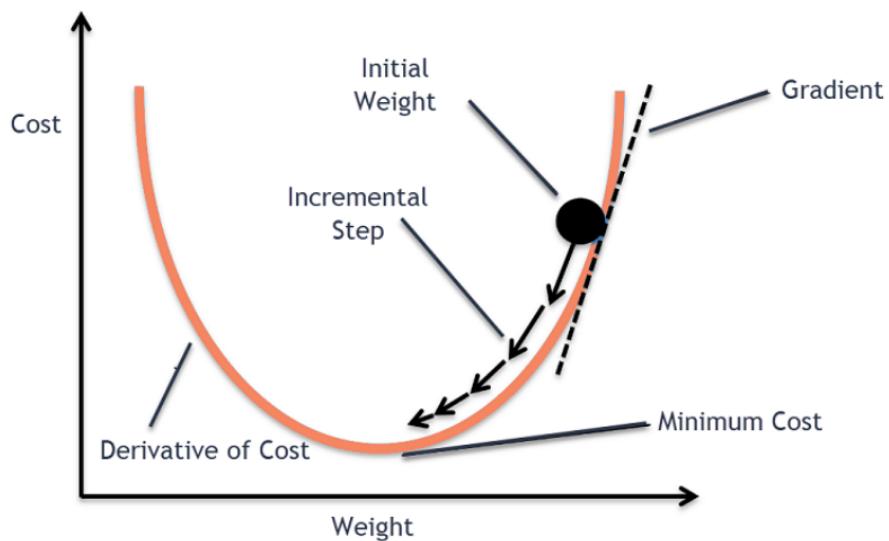
# Basics of artificial neural networks: gradient descent and neural network from scratch

# From linear models to artificial neural networks (ANNs)

- ANN: a mathematical function  $Y = f(X)$  with a special architecture
- Can be non-linear depending on activation function
- Backward propagation (gradient descent) for minimizing error
- Universal Approximation Theorem



# Gradient descent



$$y_i = \alpha + \beta x_i + \epsilon, \quad i = 1 \dots n$$

$$E(\alpha, \beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \alpha - \beta x_i)^2$$

$$\hat{\alpha}, \hat{\beta} = \operatorname{argmin} E(\alpha, \beta)$$

$$\frac{\partial E(\alpha, \beta)}{\partial \alpha} = -\frac{2}{n} \sum_{i=1}^n (y_i - \alpha - \beta x_i)$$

$$\frac{\partial E(\alpha, \beta)}{\partial \beta} = -\frac{2}{n} \sum_{i=1}^n x_i (y_i - \alpha - \beta x_i)$$

Numeric implementation of gradient descent:

$$\alpha_{i+1} = \alpha_i - \eta \frac{\partial E(\alpha, \beta)}{\partial \alpha} \Big|_{\alpha=\alpha_i, \beta=\beta_i}$$

$$\beta_{i+1} = \beta_i - \eta \frac{\partial E(\alpha, \beta)}{\partial \beta} \Big|_{\alpha=\alpha_i, \beta=\beta_i}$$

# Coding gradient descent from scratch in R

```
1 n <- 100 # sample size
2 x <- rnorm(n) # simulated explanatory variable
3 y <- 3 + 2 * x + rnorm(n) # simulated response variable
4 summary(lm(y ~ x))
```

Call:  
lm(formula = y ~ x)

Residuals:

Min	1Q	Median	3Q	Max
-1.9073	-0.6835	-0.0875	0.5806	3.2904

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	2.89720	0.09755	29.70	<2e-16 ***
x	1.94753	0.10688	18.22	<2e-16 ***

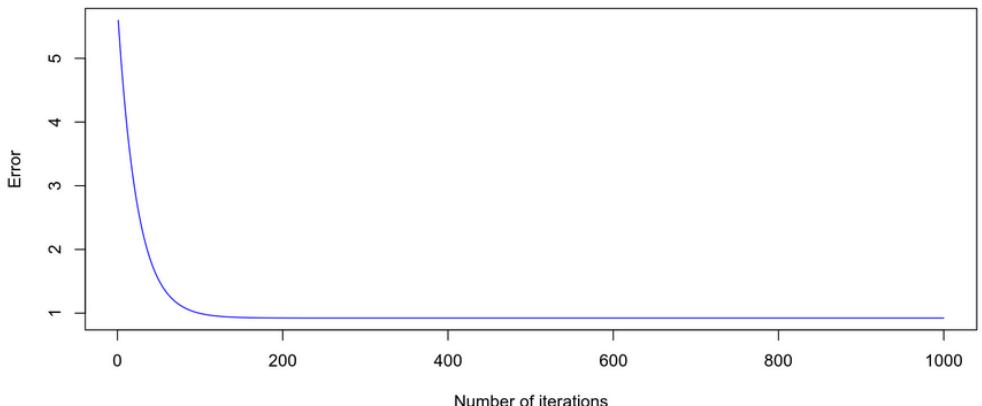
---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9707 on 98 degrees of freedom  
Multiple R-squared: 0.7721, Adjusted R-squared: 0.7698  
F-statistic: 332 on 1 and 98 DF, p-value: < 2.2e-16

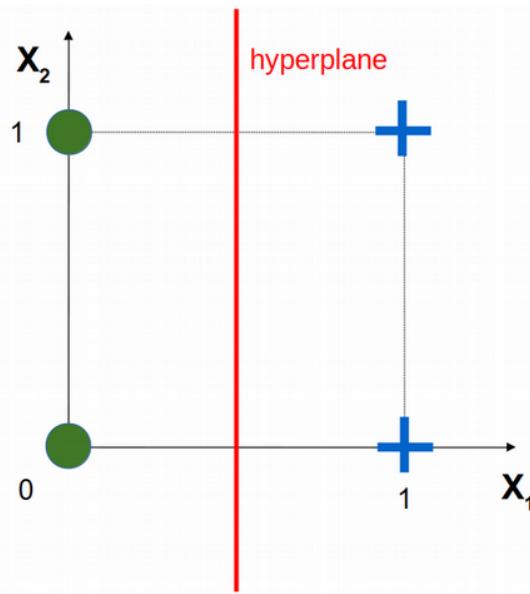
```
1 alpha <- vector(); beta <- vector()
2 E <- vector(); dEdalpha <- vector(); dEdbeta <- vector()
3 eta <- 0.01; alpha[1] <- 1; beta[1] <- 1 # initialize alpha and beta
4 for(i in 1:1000)
5 {
6   E[i] <- (1/n) * sum((y - alpha[i] - beta[i] * x)^2)
7   dEdalpha[i] <- -sum(2 * (y - alpha[i] - beta[i] * x)) / n
8   dEdbeta[i] <- -sum(2 * x * (y - alpha[i] - beta[i] * x)) / n
9
10  alpha[i+1] <- alpha[i] - eta * dEdalpha[i]
11  beta[i+1] <- beta[i] - eta * dEdbeta[i]
12 }
13 print(paste0("alpha = ", tail(alpha, 1), ", beta = ", tail(beta, 1)))
```

```
[1] "alpha = 2.89719694937354, beta = 1.94752837381973"
```



Let us now reconstruct the intercept and slope from gradient descent

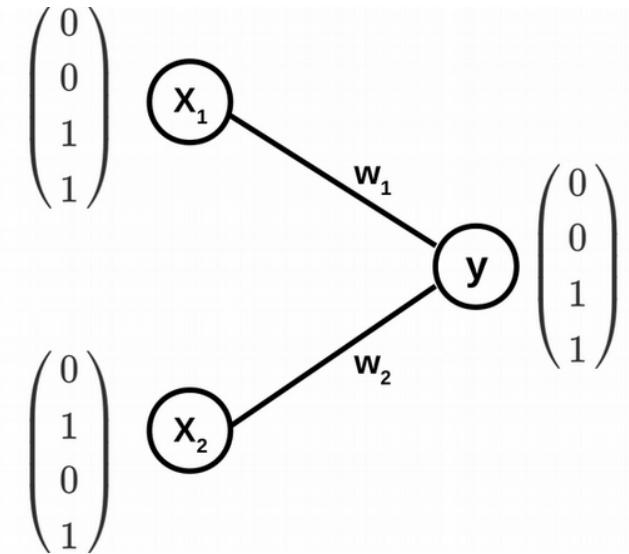
# ANN from scratch in R: problem formulation



$x_1$	$x_2$	d (true) y (pred)
0	0	0 - circle
0	1	0 - circle
1	0	1 - cross
1	1	1 - cross

```
1 d <- c(0, 0, 1, 1) # true labels
2 x1 <- c(0, 0, 1, 1) # input variable x1
3 x2 <- c(0, 1, 0, 1) # input variable x2
4
5 data.frame(x1 = x1, x2 = x2, d = d)
```

x1	x2	d
0	0	0
0	1	0
1	0	1
1	1	1



$$y(w_1, w_2) = \phi(w_1 x_1 + w_2 x_2)$$

$$\phi(s) = \frac{1}{1 + e^{-s}} - \text{sigmoid}$$

$$\phi'(s) = \phi(s)(1 - \phi(s))$$

# ANN from scratch in R: implementation in code

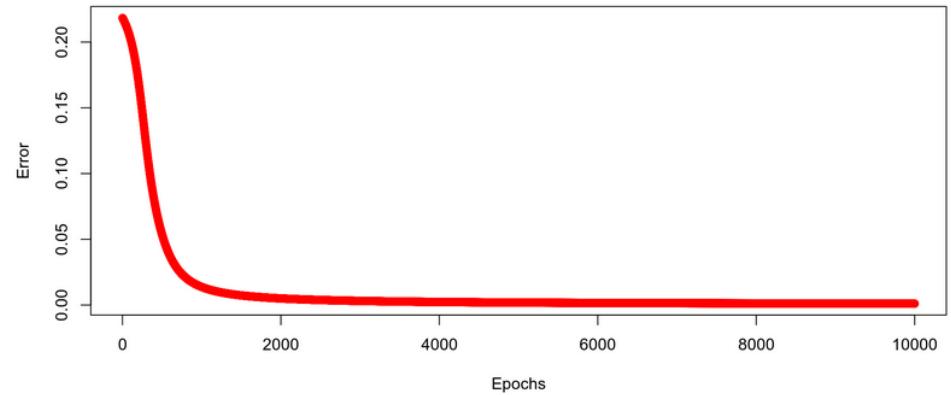
$$E(w_1, w_2) = \frac{1}{2N} \sum_{i=1}^N (d_i - y_i(w_1, w_2))^2$$

$$w_{1,2} = w_{1,2} - \mu \frac{\partial E(w_1, w_2)}{\partial w_{1,2}}$$

$$\frac{\partial E}{\partial w_1} = -\frac{1}{N} \sum_{i=1}^N (d_i - y_i) * y_i * (1 - y_i) * x_{1i}$$

$$\frac{\partial E}{\partial w_2} = -\frac{1}{N} \sum_{i=1}^N (d_i - y_i) * y_i * (1 - y_i) * x_{2i}$$

```
1 phi <- function(x){return(1/(1 + exp(-x)))} # activation function
2
3 mu <- 0.1; N_epochs <- 10000
4 w1 <- 0.1; w2 <- 0.5; E <- vector()
5 for(epochs in 1:N_epochs)
6 {
7     #Forward propagation
8     y <- phi(w1 * x1 + w2 * x2 - 3) # we use a fixed bias -3
9
10    #Backward propagation
11    E[epochs] <- -(1 / (2 * length(d))) * sum((d - y)^2)
12    dE_dw1 <- -(1 / length(d)) * sum((d - y) * y * (1 - y) * x1)
13    dE_dw2 <- -(1 / length(d)) * sum((d - y) * y * (1 - y) * x2)
14    w1 <- w1 - mu * dE_dw1
15    w2 <- w2 - mu * dE_dw2
16 }
17 plot(E ~ seq(1:N_epochs), xlab="Epochs", ylab="Error", col="red")
```



```
1 y
```

```
[1] 0.04742587 0.05752359 0.95730271 0.96489475
```

We nearly reconstruct true labels  $d = (0, 0, 1, 1)$

Take home messages of the session:

- 1) Artificial neural networks are based on computing derivatives
- 2) Gradient descent (backward propagation) is the engine of ANNs
- 3) Universal Approximation Theorem allows fitting any function



# National Bioinformatics Infrastructure Sweden (NBIS)



*Knut och Alice  
Wallenbergs  
Stiftelse*



**LUNDS  
UNIVERSITET**

# STOP