

Short-course & Hands-on Workshop: Artificial Intelligence Algorithms for Everyone

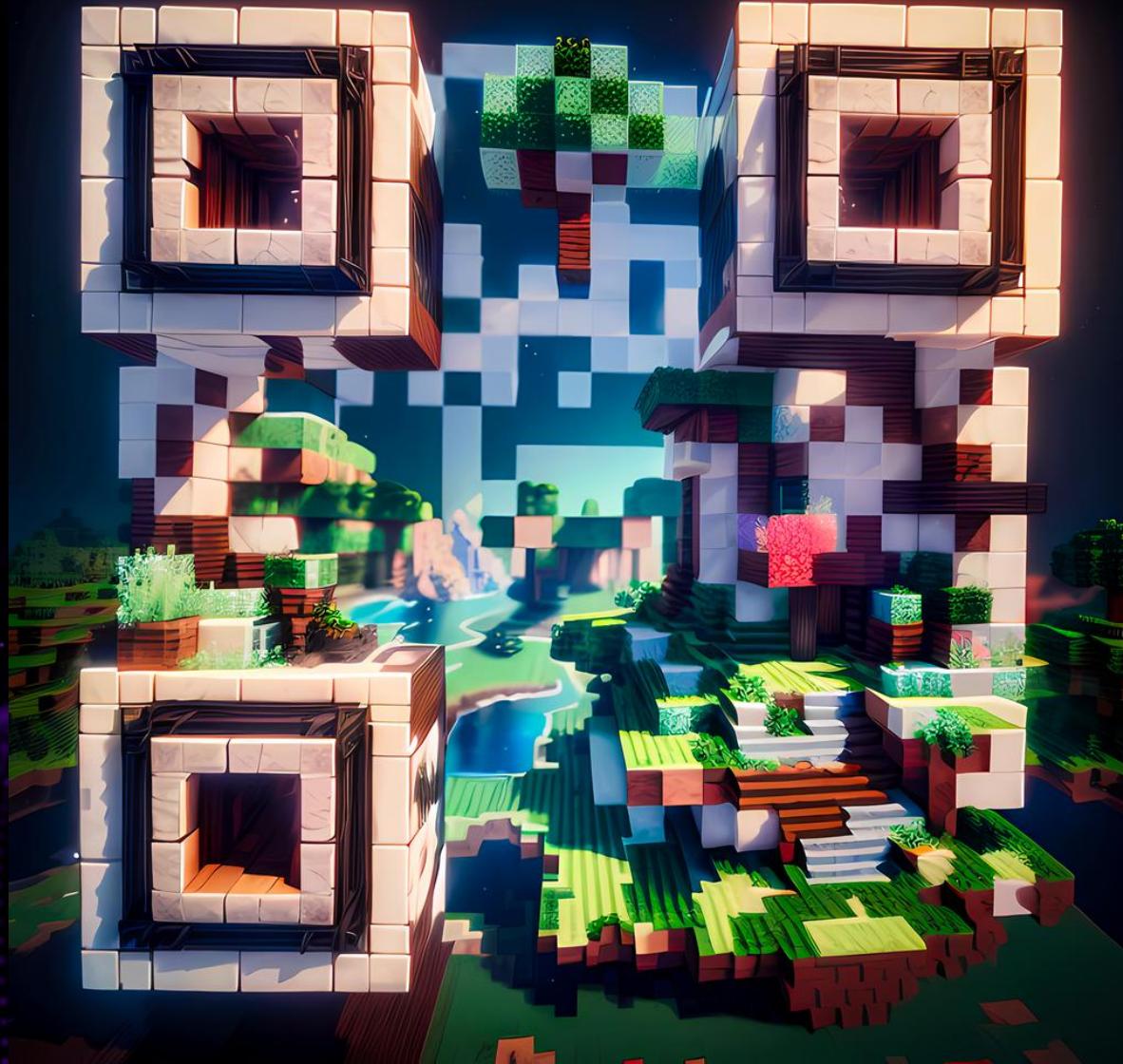
June 23 to June 26, 2025

Dr. R.M. (Rolando) Gonzales Martinez



university of
groningen

GitHub repository



https://github.com/rogon666/AI_workshop

- Presentations
- Databases
- Python scripts
- Additional lectures
(papers, books)

June 23
(Monday)

Introduction to artificial intelligence and Python programming (homogenization)

- The history of the spring and winters in artificial intelligence: from automatons to deep artificial neural networks and up to the singularity of human-level artificial intelligence
- Introduction to Python programming of artificial intelligence algorithms, elastic nets and vife coding

June 24
(Tuesday)

Machine learning algorithms

- Machine learning algorithms for classification and regression: elastic nets, XGBoost, artificial neural networks, support vector machines, random forests
- Time Series Forecasting : modeling and predicting time series data in practice with META's Prophet and long short-term memory models
- Model evaluation, feature engineering, feature selection, and fine-tuning: metrics based on error minimization and the confusion matrix, Bayesian hypertuning

June 25
(Wednesday)

Deep learning algorithms and recent trends in Artificial Intelligence

- Deep learning algorithms and Large Language Models (LLMs): the Transformers' architecture
- Semantic and sentiment analysis of textual data with RoBERTa
- Ethical considerations on the application of AI algorithms and mitigation strategies
- Recent advances in AI: liquid neural networks, how to use DeepSeek without worrying about privacy issues, spatial machine learning, systematic reviews and literature reviews in 5-minutes with GPTs, quantum computing, liquid neural networks

June 26
(Thursday)

Presentations (4 PM):

- Using Mobile Phone Surveys (MPS) to measure child mortality during crisis periods (Kassoum Dianou)
- Ways in which AI can support autonomous navigation in Space Missions (Duna Meya i Mendoza)

Today

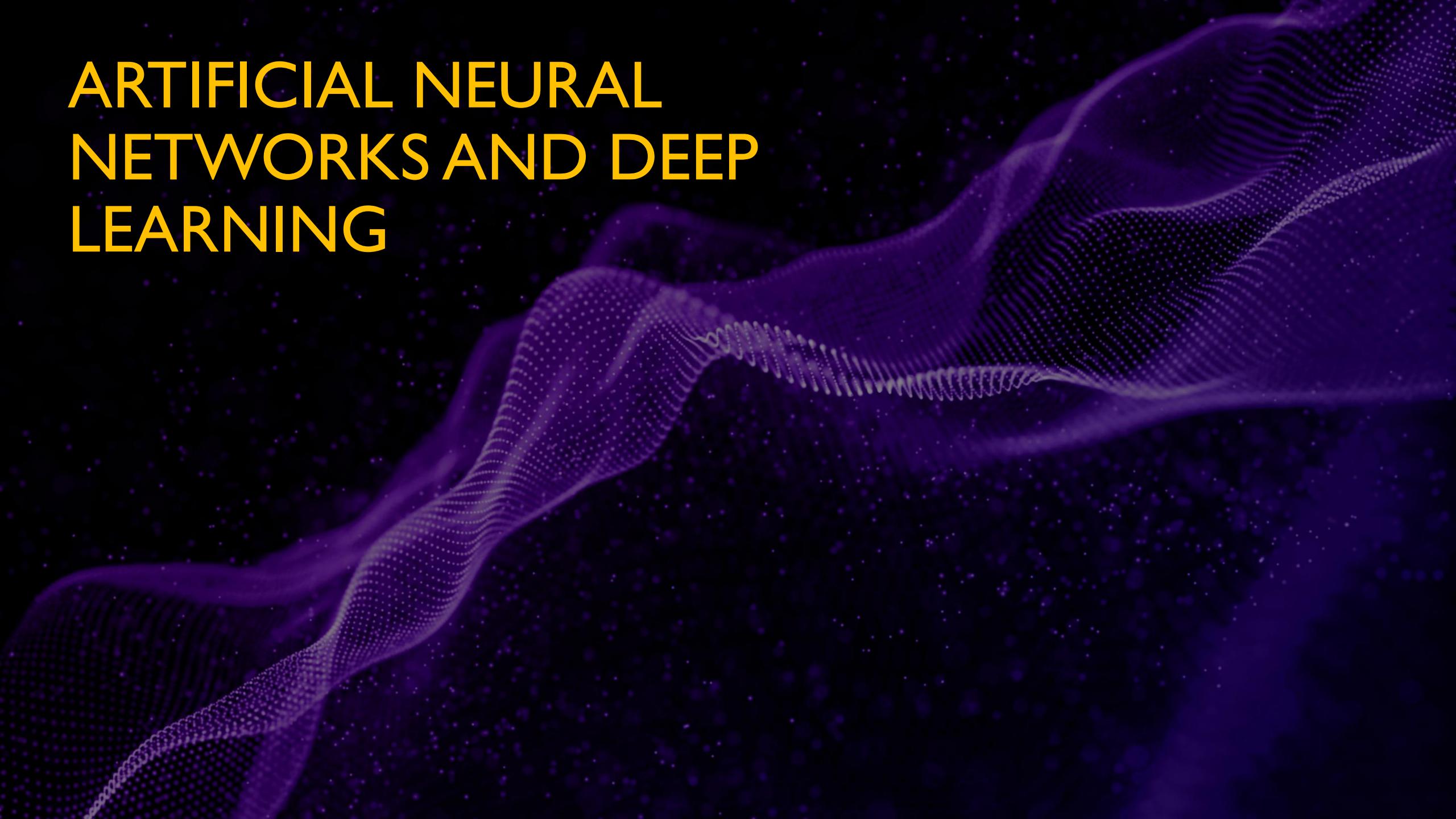
- Artificial neural networks
 - Perceptron and multilayer Perceptron
 - Cybenko-Hornik-Stinchcombe Theorem
 - Backpropagation
 - Non-Linear Activation Functions
 - Regularization
 - Adaptive Moment Estimation (ADAM)
 - Deep learning
 - Recurrent neural networks
 - Neuralprophet
- Large Language Models
 - Running local LLMs
 - Literature reviews with LLMs
 - Sentiment analysis with RoBERTa
- Ethics in Artificial Intelligence
- Recent developments in Artificial Intelligence
 - Spatial machine learning
 - Neuromorphic computing
 - Quantum computing

PRACTICAL APPLICATIONS

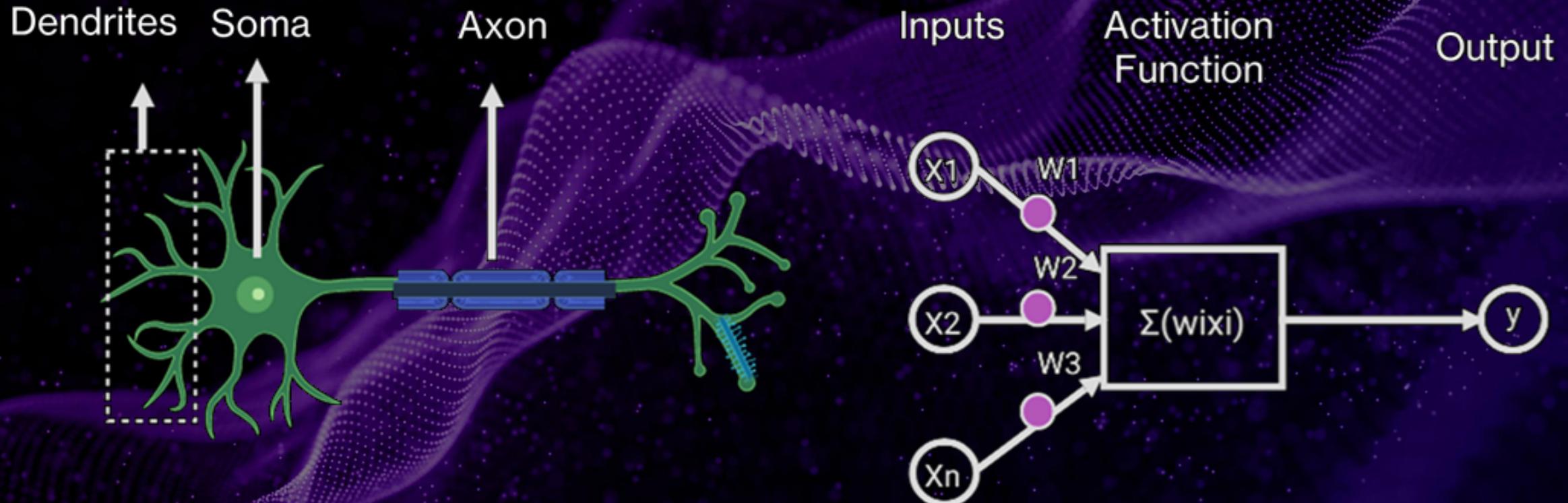
Python scripts for demographic forecasts with AI algorithms:

- `aiw_popfor01.ipynb`: Population forecasts with LSTM
- `aiw_popfor02.ipynb`: Population forecasts with LSTM & GRU
- `aiw_popfor03.ipynb`: Population forecasts with LSTM & GRU and hyper-parameter fine-tuning with grid search

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING



Biological Neural Networks and Artificial Neural Networks



First Developments of Artificial Neural Networks

- **1943: McCulloch-Pitts Neuron.** A simple mathematical model of an artificial neuron based on a step function (non-linear activation function, binary inputs \mathbf{x} , weights \mathbf{w} , and a threshold θ).
- **1958: Frank Rosenblatt's Perceptron.** A single-layer neural network capable of learning to classify linearly separable patterns. It is a generalization of the McCulloch-Pitts model, where weights are updated based on errors made during training, with labels η and a learning rate η .

$$y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i \geq \theta \\ 0 & \text{si } \sum_{i=1}^n w_i x_i < \theta \end{cases}$$

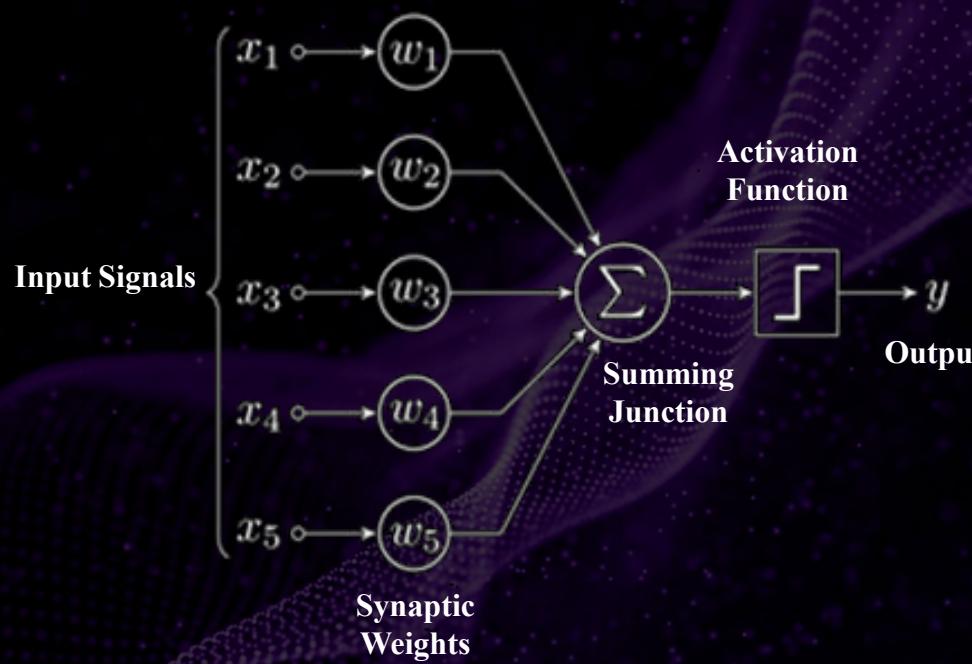
$$y = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i + b > 0 \\ 0 & \text{si } \sum_{i=1}^n w_i x_i + b \leq 0 \end{cases}$$

$$w_i \leftarrow w_i + \eta(d - y)x_i$$

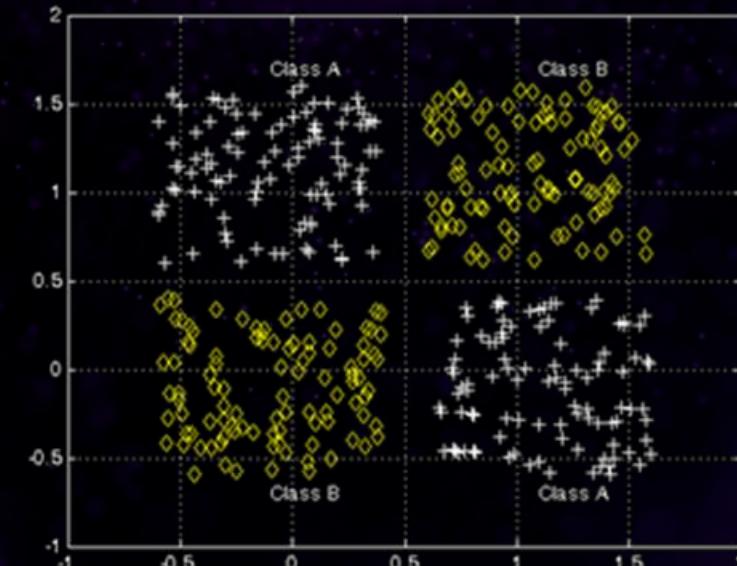
Limitations of the Perceptron

1969: Limitations of the Perceptron (Minsky and Papert)

In their book "Perceptrons", Minsky and Papert demonstrated that the simple perceptron cannot solve nonlinear problems, such as the XOR function, because the perceptron is a linear model and can only separate data that is linearly separable.



$$\text{XOR}(x_1, x_2) = \begin{cases} 1 & \text{si } (x_1 = 1 \text{ y } x_2 = 0) \text{ o } (x_1 = 0 \text{ y } x_2 = 1) \\ 0 & \text{si } x_1 = x_2 \end{cases}$$



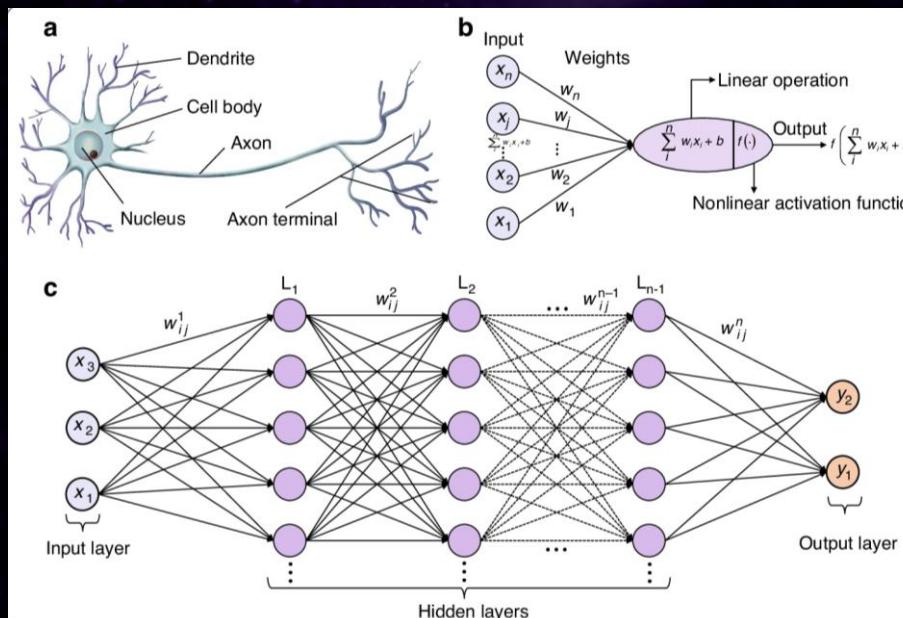
Multilayer Perceptron

Extension of the perceptron to three-layer neural networks (input, hidden, and output), capable of solving non-linearly separable problems.

- The hidden layer is based on non-linear activation functions, such as the sigmoid or hyperbolic tangent functions.
- Moreover, the multilayer perceptron can approximate any continuous function arbitrarily well, thanks to the universal approximation theorem.
- Training these multilayer perceptrons was extremely difficult due to the lack of an efficient method for adjusting the weights across all layers.

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l$$
$$\mathbf{a}_l = \sigma_l(\mathbf{z}_l) \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathbf{a}_L = \sigma_L(\mathbf{W}_L \mathbf{a}_{L-1} + \mathbf{b}_L)$$



Fu, T., Zhang, J., Sun, R., Huang, Y., Xu, W., Yang, S., ... & Chen, H. (2024). Optical neural networks: progress and challenges. *Light: Science & Applications*, 13(1), 263.

Universal Approximation Theorem (Cybenko-Hornik-Stinchcombe)

- Intuitively, there is some analogy with the Weierstrass Approximation Theorem.
- It provides a solid theoretical foundation for the use of neural networks (NNs): they are capable of approximating any continuous function with the desired accuracy, as long as they have enough neurons in the hidden layer and a suitable nonlinear activation function.

$$\begin{aligned} & C([0, 1]^n) \\ & f \in C([0, 1]^n) \\ & F(x) = \sum_{i=1}^h \omega_i \sigma(w_i x + b_i) \\ & \sup_{x \in [0, 1]^n} |f(x) - F(x)| < \epsilon \\ & \epsilon > 0 \end{aligned}$$

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals and Systems, 2(4), 303–314.

Hornik, K., Stinchcombe, M., & White, H. (1989). "Multilayer feedforward networks are universal approximators". Neural Networks, 2(5), 359–366.

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". Neural Networks, 4(2), 251–257.

Backpropagation

1986: Backpropagation: Geoffrey Hinton, David Rumelhart, and Ronald Williams popularized the backpropagation algorithm, which allows for the training of multi-layer neural networks.

- **Step 1: Forward propagation.** The output of the network for a given input x is calculated by propagating the activations from the input layer to the output layer.
- **Step 2: Backward propagation.** The loss function is evaluated, and the gradient of the loss function with respect to the activations of each layer is calculated from the output layer. The error δ for each layer is then backpropagated back to the input layer. The weights and biases are adjusted in the direction opposite to the gradient.

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$$

$$\nabla_w \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

$$\mathbf{a}_L = \sigma_L(\mathbf{W}_L \mathbf{a}_{L-1} + \mathbf{b}_L)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l} = \delta_l \mathbf{a}_{l-1}^T$$

$$\delta_l = (\mathbf{W}_{l+1}^T \delta_{l+1}) \odot \sigma'_l(\mathbf{z}_l)$$

$$\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

$$\mathbf{b}_l \leftarrow \mathbf{b}_l - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_l}$$

Steps 1 and 2 are repeated for several training epochs.

Artificial Neural Networks

Given a function with an input vector \mathbf{x} of dimension k , and θ as the model parameters, which include synaptic weights and biases, an Artificial Neural Network (ANN) is a computational model parameterized based on a succession of composite functions:

In the composite function:

- L is the number of layers ($l = 1, 2, \dots, L$). $f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$
- In the functions of each layer $f(\mathbf{z}, \theta)$, \mathbf{z} is an input vector obtained from the output of the previous layer.
- The functions of each layer $f(\mathbf{z}, \theta)$ are **linear expressions** within a **non-linear** activation function $\sigma(\cdot)$, with weights \mathbf{W} and biases \mathbf{b} for each layer. $f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Artificial Neural Networks

Given a function with an input vector \mathbf{x} of dimension k , and θ as the model parameters, which include synaptic weights and biases, an Artificial Neural Network (ANN) is a computational model parameterized based on a succession of composite functions:

- The neural network is trained by adjusting the parameters θ to minimize a loss function that measures the differences between the observed data and the predicted outcomes.
- The parameter adjustment is performed using the gradient descent ∇ of the loss function, where η is the learning rate.

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Artificial Neural Networks

Given a function with an input vector \mathbf{x} of dimension k , and θ as the model parameters, which include synaptic weights and biases, an Artificial Neural Network (ANN) is a computational model parameterized based on a succession of composite functions:

A typical neural network has three types of layers:

- **Input Layer:** Receives the initial data.
- **Hidden Layers:** Process the information through linear combinations and non-linear functions.
- **Output Layer:** Produces the final result.

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2)\dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Artificial Neural Networks

Given a function with an input vector \mathbf{x} of dimension k , and θ as the model parameters, which include synaptic weights and biases, an Artificial Neural Network (ANN) is a computational model parameterized based on a succession of composite functions:

A typical neural network has three types of layers:

- L tends to have a very large value, specifically in terms of hidden layers.
- Optimization is performed using batch techniques, activation functions like ReLU, regularization methods, and modern variants of stochastic gradient descent, such as ADAM.

$$f(\mathbf{x}; \theta) = f_L(f_{L-1}(\dots f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \dots); \theta_L)$$

$$f_l(\mathbf{z}; \theta_l) = \sigma_l(\mathbf{W}_l \mathbf{z} + \mathbf{b}_l)$$

$$\theta = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$$

$$\mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} \mathcal{L}(f(\mathbf{x}; \theta), \mathbf{y})$$

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Non-Linear Activation Functions

- **Sigmoid Function:** Maps values from 0 to 1. It suffers from the vanishing gradient problem.
- **Hyperbolic Tangent Function:** Maps values from -1 to 1. It also suffers from the vanishing gradient problem.
- **ReLU (Rectified Linear Unit):** Transforms all negative values to zero while keeping positive gradient values. It solves the vanishing gradient problem but may cause dead neurons.
- **Softmax:** Useful for multi-class targets but computationally expensive.
- **Swish:** Works better in deep networks but is computationally more expensive than ReLU.

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

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

$$\text{ReLU}(x) = \max(0, x)$$

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

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

Non-Linear Activation Functions

- **ELU (Exponential Linear Unit):** Avoids the problem of dead neurons.
- **SELU (Scaled Exponential Linear Unit):** Maintains the normalization of activations, typically with $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$.
- **GELU (Gaussian Error Linear Unit):** Useful in NLP, particularly in transformer architectures.
- **Leaky ReLU:** Adds a hyperparameter α to allow the passage of negative values and prevent dead neurons. PReLU estimates the hyperparameter α during training.

$$\text{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha(\exp(x) - 1) & \text{si } x \leq 0 \end{cases}$$

$$\text{SELU}(x) = \begin{cases} \lambda x & \text{si } x > 0 \\ \lambda\alpha(\exp(x) - 1) & \text{si } x \leq 0 \end{cases}$$

$$\begin{aligned}\text{GELU}(x) &= x \Phi(x) \\ &= x \frac{1}{2} \left[1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right] \\ \operatorname{erf}(x) &= \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt\end{aligned}$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

Non-Linear Activation Functions

- A **dense layer** is a fully connected layer where each neuron is connected to all the neurons in the previous layer.
- The main operation performed by a dense layer is a matrix multiplication followed by an addition with a bias term, and optionally the application of an activation function.
- It is commonly used as **the final layer** in a deep network.

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{a} = f(\mathbf{z}) = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$z_i = \sum_{j=1}^k W_{ij} x_j + b_i$$

$$a_i = f(z_i) = f\left(\sum_{j=1}^k W_{ij} x_j + b_i\right)$$

$$a_i = f(z_i) = \frac{1}{1 + e^{-z_i}}$$

$$\mathbf{x} \in \mathbb{R}^k$$

$$\mathbf{W} \in \mathbb{R}^{n \times k}$$

$$\mathbf{b} \in \mathbb{R}^n$$

$$\mathbf{z} \in \mathbb{R}^n$$

$$\mathbf{a} \in \mathbb{R}^n$$

Adaptive Moment Estimation (ADAM)

ADAM is a widely used optimization algorithm for training deep learning models.

It adapts the learning rates for each parameter using estimates of the first and second moments of the gradients.

Hyperparameters:

- Learning rate: α
- Decay parameters: β_1, β_2
- Term for numerical stability: ϵ
- Iteration counter: $t = 1, 2, \dots, T$

$$\theta_0 \quad m_0 = 0, v_0 = 0 \quad \alpha, \beta_1, \beta_2, \epsilon$$
$$g_t = \nabla_{\theta} L(\theta_{t-1})$$

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \left(h_{\theta}(x^{(i)}) - y^{(i)} \right)^2$$

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^n \left[y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

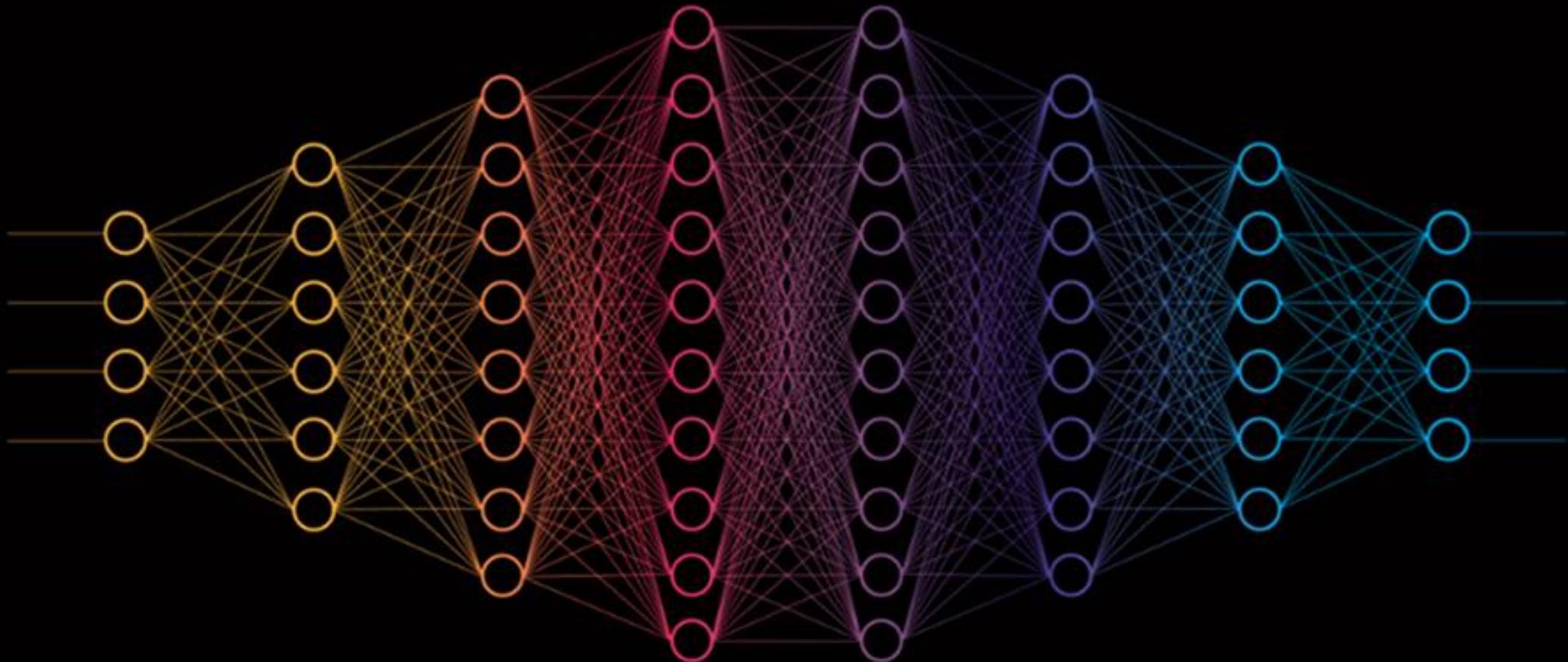
$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Regularization in artificial neural networks

- In artificial neural networks and deep learning, dropout and L1 and L2 regularizations are applied.
- Dropout is a regularization technique that randomly "turns off" a percentage of neurons during training to prevent overfitting:
 - **High dropout:** Can prevent overfitting but may also cause the model to underfit if overused.
 - **Low dropout:** Allows the model to learn more details but increases the risk of overfitting.

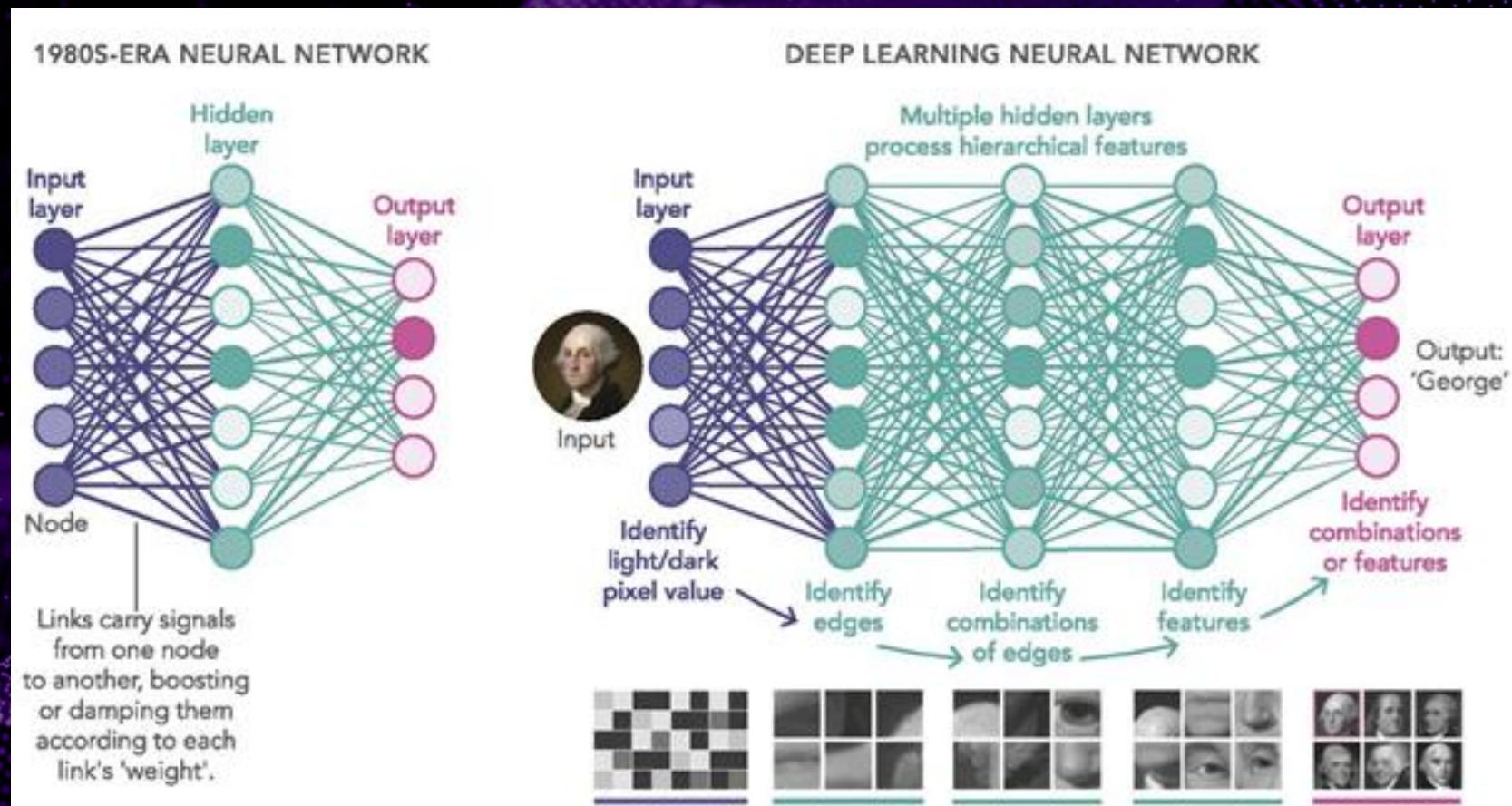
A common value for dropout ranges between 0.2 and 0.5, depending on the problem and the amount of data.

Deep Neural Networks



Deep neural networks

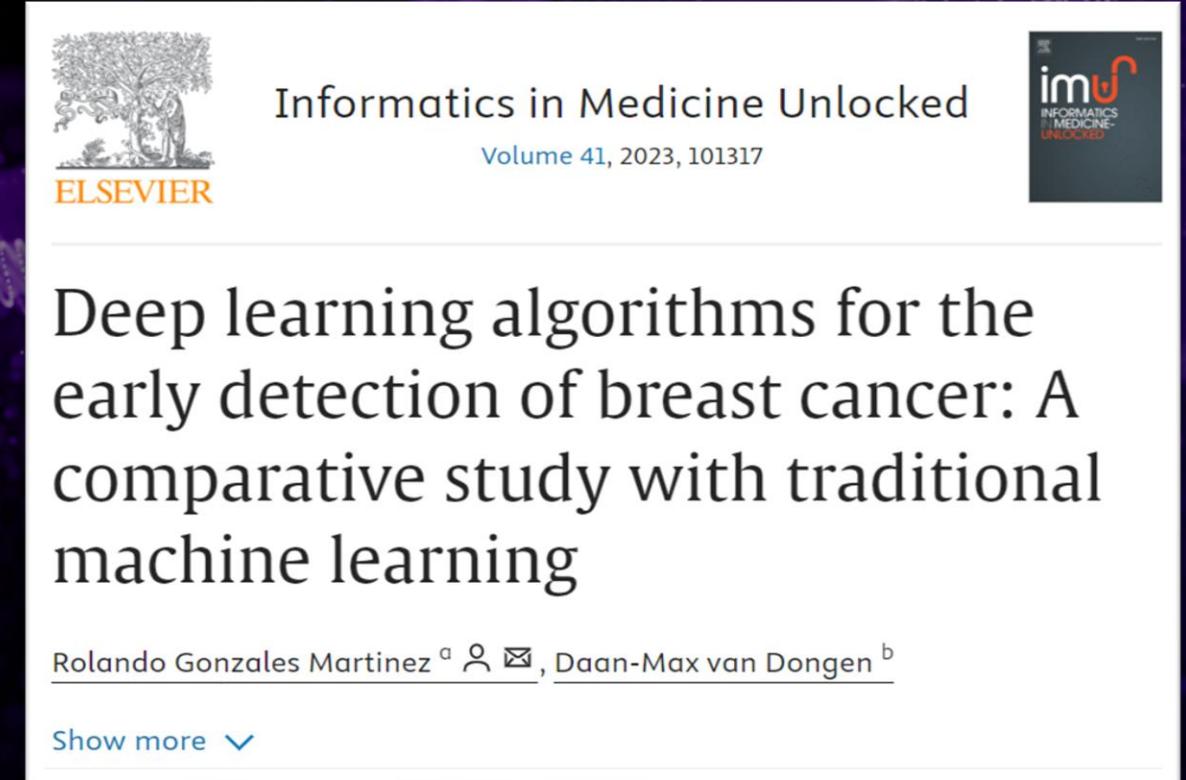
2010s: Deep Learning: Due to increased computational power and access to large volumes of data (big data), deep neural networks began to dominate many areas of artificial intelligence, such as computer vision:



Examples of Deep Learning Applications

ML models/algorithms are compared with deep learning neural networks for breast cancer pre-screening:

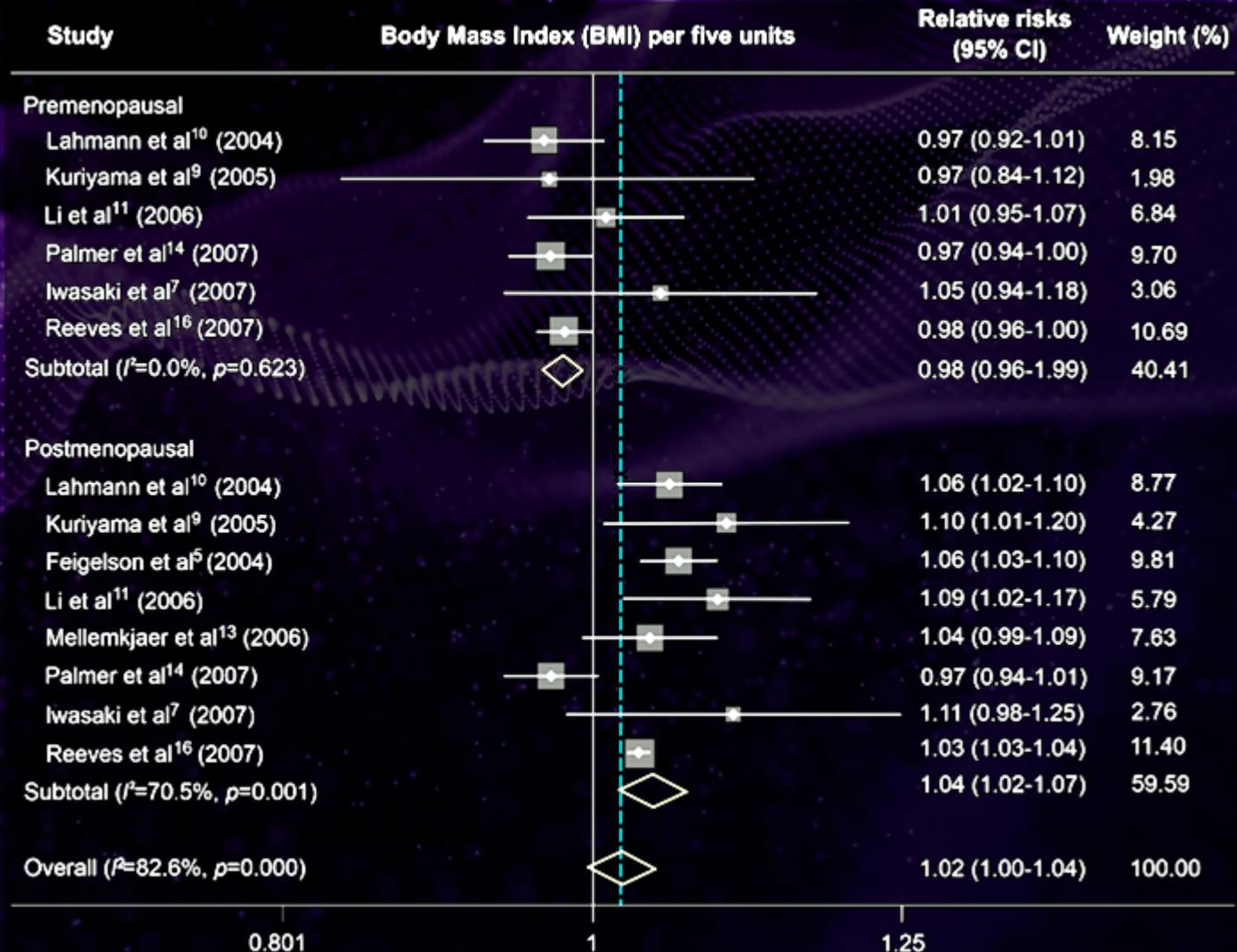
<https://www.sciencedirect.com/science/article/pii/S2352914823001636?via%3Dihub>



The screenshot shows a scientific article page. At the top left is the Elsevier logo, which includes a tree illustration and the word "ELSEVIER". To the right of the logo is the journal title "Informatics in Medicine Unlocked" and the volume information "Volume 41, 2023, 101317". On the far right is the IMU logo, which consists of the letters "im" in white and "u" in red, with the full name "INFORMATICS IN MEDICINE UNLOCKED" written vertically below it. The main title of the article is "Deep learning algorithms for the early detection of breast cancer: A comparative study with traditional machine learning", written in a large, bold, black font. Below the title, the authors are listed as "Rolando Gonzales Martinez ^a   , Daan-Max van Dongen ^b". There is also a "Show more" link at the bottom of the author list.

Examples of Deep Learning Applications

- Breast cancer screening: mammograms, biopsies
- Pre-screening: demographic data, anthropometric data, relative risks (GBD), and data obtained from routine blood samples



Examples of Deep Learning Applications

- Breast cancer screening: mammograms, biopsies
- Pre-screening: demographic data, anthropometric data, relative risks (GBD), and data obtained from routine blood samples

Table 1
Descriptive statistics of the database.

	mean	std.dev.	min	max
Age (years)	57.30	16.11	24	89
BMI	27.58	5.02	18.37	38.58
Glucose (mg/dL)	97.79	22.53	60.00	201.00
Insulin (μ U/mL)	10.01	10.07	2.43	58.46
HOMA	2.69	3.64	0.47	25.05
Leptin (ng/mL)	26.62	19.18	4.31	90.28
Adiponectin (μ g/mL)	10.18	6.84	1.66	38.04
Resistin (ng/mL)	14.73	12.39	3.21	82.10
RRs [19]	1.51	0.48	0.40	2.44
RRs GBD (center)	1.01	0.10	0.89	1.09
RRs GBD (lower)	0.97	0.08	0.87	1.04
RRs GBD (upper)	1.05	0.11	0.91	1.14
High BMI (binary)	0.34	0.47	0	1
Obesity (binary)	0.32	0.47	0	1

Examples of Deep Learning Applications

Feature engineering:

- Feature selection was performed using extreme gradient boosting.
- Interactions terms were created with the features

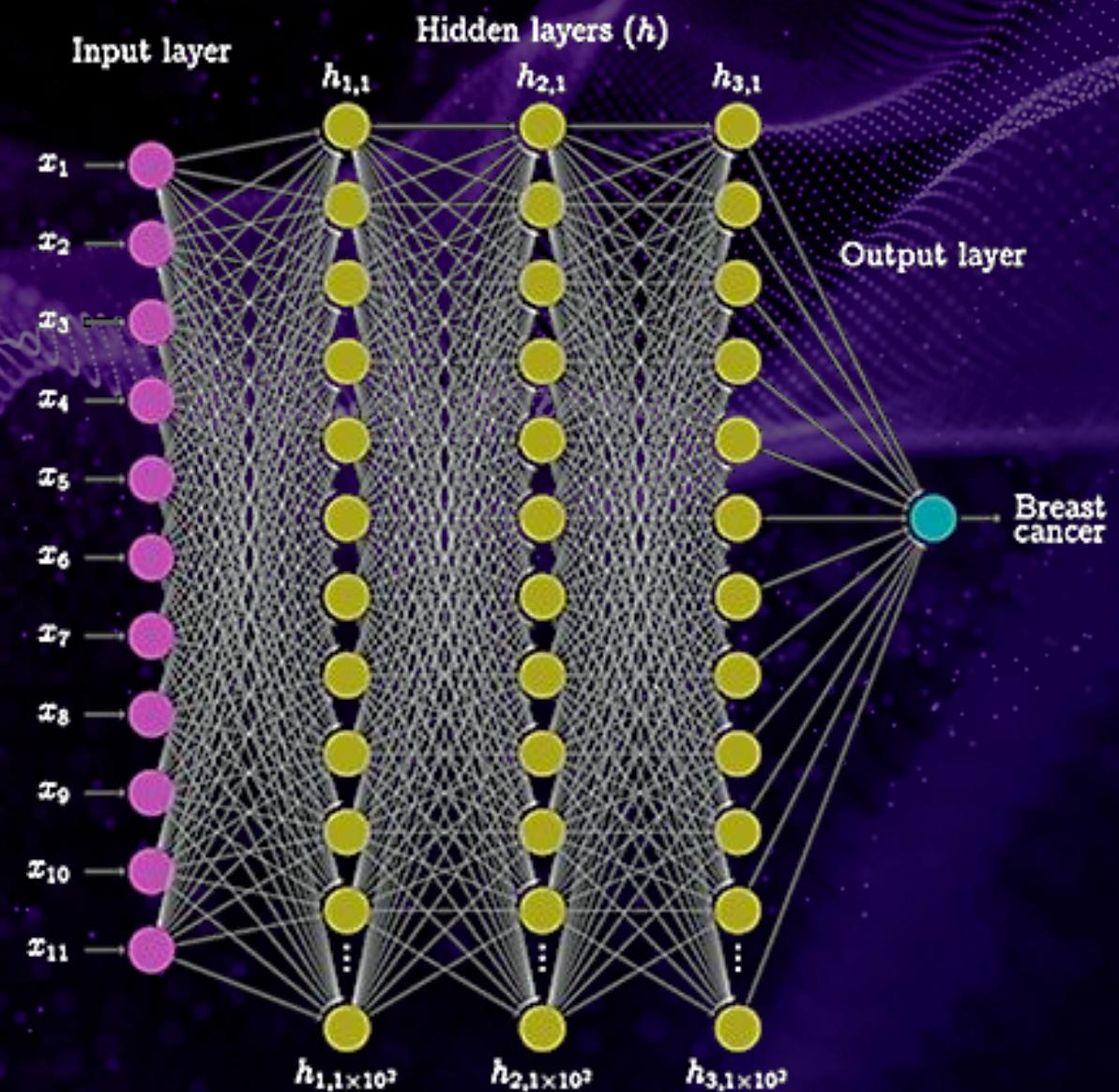
Table 2
Recursive MRMR feature selection with SULOV-gradient boosting.

	description	frequency	min ρ	mean ρ	std.dev. ρ
x ₁	Age (years)	97	0.00	0.49	0.28
x ₂	Resistin (ng/mL)	90	0.00	0.51	0.28
x ₃	RRs GBD (upper)	77	0.05	0.55	0.27
x ₄	Glucose (mg/dL)	76	0.21	0.59	0.22
x ₅	Adiponectin (μ g/mL)	75	0.23	0.60	0.22
x ₆	High BMI (binary)	65	0.27	0.64	0.20
x ₇	MCP-1 (pg/dL)	65	0.26	0.65	0.19
x ₈	Leptin (ng/mL)	64	0.30	0.65	0.19
x ₉	RRs [19]	59	0.37	0.68	0.17
x ₁₀	Obesity (binary)	59	0.11	0.55	0.28
x ₁₁	Insulin (μ U/mL)	33	0.05	0.53	0.26
	BMI	30	0.62	0.81	0.11
	HOMA	29	0.69	0.83	0.09
	RRs GBD (center)	8	0.86	0.90	0.04
	RRs GBD (lower)	8	0.21	0.81	0.25

Examples of Deep Learning Applications

In the architecture of the deep neural network:

- Rectified Linear Activation Functions (ReLU)
- 3 hidden layers
- 102 nodes in each hidden layer, with L1 and L2 regularizers to prevent overfitting

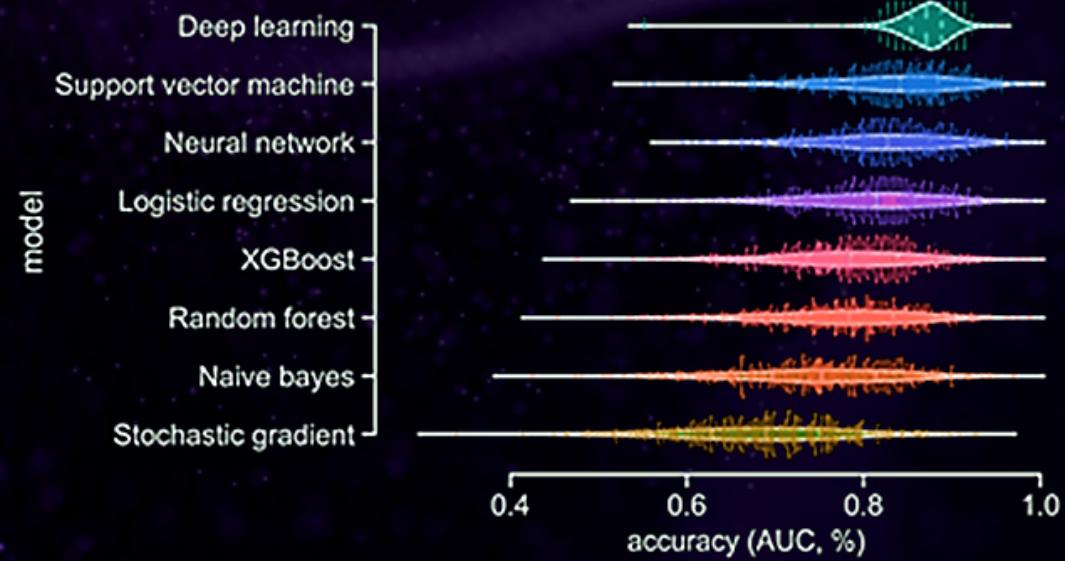


Examples of Deep Learning Applications

Table 3

AUC of the machine learning and deep learning algorithms.

Model	mean	std.dev.	p2.5	p97.5
Deep learning	0.8699	0.0345	0.8190	0.9138
Support vector machines	0.8344	0.0711	0.6768	0.9527
Neural network	0.8232	0.0720	0.6667	0.9476
Logistic regression	0.8078	0.0741	0.6569	0.9334
XGBoost	0.7834	0.0755	0.6263	0.9191
Random forest	0.7763	0.0804	0.6035	0.9192
Naive bayes	0.7504	0.0869	0.5686	0.9001
Stochastic gradient	0.6861	0.0860	0.5120	0.8553



Examples of Deep Learning Applications

Table 4

Performance metrics based on the confusion matrix.

Model	Derivations from confusion matrix*					
	TPR	TNR	FNR	FPR	FDR	FOR
Deep learning	0.9220 (0.0597)	0.8021 (0.1118)	0.0780 (0.0597)	0.1979 (0.1118)	0.1425 (0.0522)	0.0983 (0.0655)
SVM	0.7939 (0.1183)	0.6747 (0.1381)	0.2061 (0.1183)	0.3253 (0.1381)	0.2501 (0.1065)	0.2672 (0.1399)
Neural network	0.7638 (0.1148)	0.7135 (0.1324)	0.2362 (0.1148)	0.2865 (0.1324)	0.2331 (0.1051)	0.2851 (0.1284)
Logistic regression	0.7329 (0.1243)	0.7311 (0.1292)	0.2671 (0.1243)	0.2689 (0.1292)	0.2291 (0.108)	0.3055 (0.1319)
XGBoost	0.7537 (0.1158)	0.6723 (0.1342)	0.2463 (0.1158)	0.3277 (0.1342)	0.2604 (0.1029)	0.3061 (0.1327)
Random forest	0.6892 (0.1267)	0.7038 (0.1365)	0.3108 (0.1267)	0.2962 (0.1365)	0.2575 (0.1144)	0.3477 (0.1263)
Naive bayes	0.5313 (0.1296)	0.8023 (0.1131)	0.4687 (0.1296)	0.1977 (0.1131)	0.2303 (0.126)	0.4147 (0.1083)
Stochastic gradient	0.7126 (0.1394)	0.6518 (0.1587)	0.2874 (0.1394)	0.3482 (0.1587)	0.2792 (0.1106)	0.3412 (0.1409)

(*) TPR: True positive rate, TNR: true negative rate, FNR: false negative rate, FPR: false positive rate, FDR: false detection rate, FOR: false omission rate. Rates are averages of the 1×10^3 k-fold cross validations. Standard deviations in brackets below each average rate. SVM: Support Vector Machines.

Recurrent Neural Networks (RNNs)

- Recurrent Neural Networks (RNNs) have a mathematical structure that allows them to process data sequences such as time series.
- The recurrent connection in RNNs means that the output of a layer at time t can be used as input at time $t+1$.
- They are trained using Backpropagation Through Time (BPTT), an extension of backpropagation, which operates from T down to 1. In this process, the hidden layers at time t are updated with the information from time $t-1$. This can lead to issues like gradient vanishing or gradient exploding.

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = g(W_{hy}h_t + b_y)$$

$$L = \sum_{t=1}^T L_t(y_t, \hat{y}_t)$$

$$\frac{\partial L}{\partial W_{hx}} = \sum_{t=1}^T \frac{\partial L}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hx}}$$

Recurrent Neural Networks: LSTM

- In an LSTM-type RNN, there is a **dynamic memory cell** that depends on a **forget function** and an **input function**.
- A **hyperbolic tangent function** is used to calculate candidates for the new memory cell.
- An **output function** is applied to decide how much of the information stored in the memory cell is used to generate the output h_t , which will be passed as input to the next unit in the sequence.

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

$$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C)$$

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \tanh(C_t)$$

Recurrent Neural Networks: GRU

- It does not have a separate memory cell.
- The update gate (z) controls how much of the previous hidden state should be carried over to the next time step.
- The reset gate decides how much information from the previous hidden state should be forgotten before calculating the new state.
- The intermediate state is a "proposed" version of the new hidden state h_t . It is calculated using the current input and the previous hidden state, modulated by the reset.

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h[r_t h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t)h_{t-1} + z_t \tilde{h}_t$$

The hidden state h is a controlled mixture of past and new information, where z decides how much of the past is retained and how much new information is added.

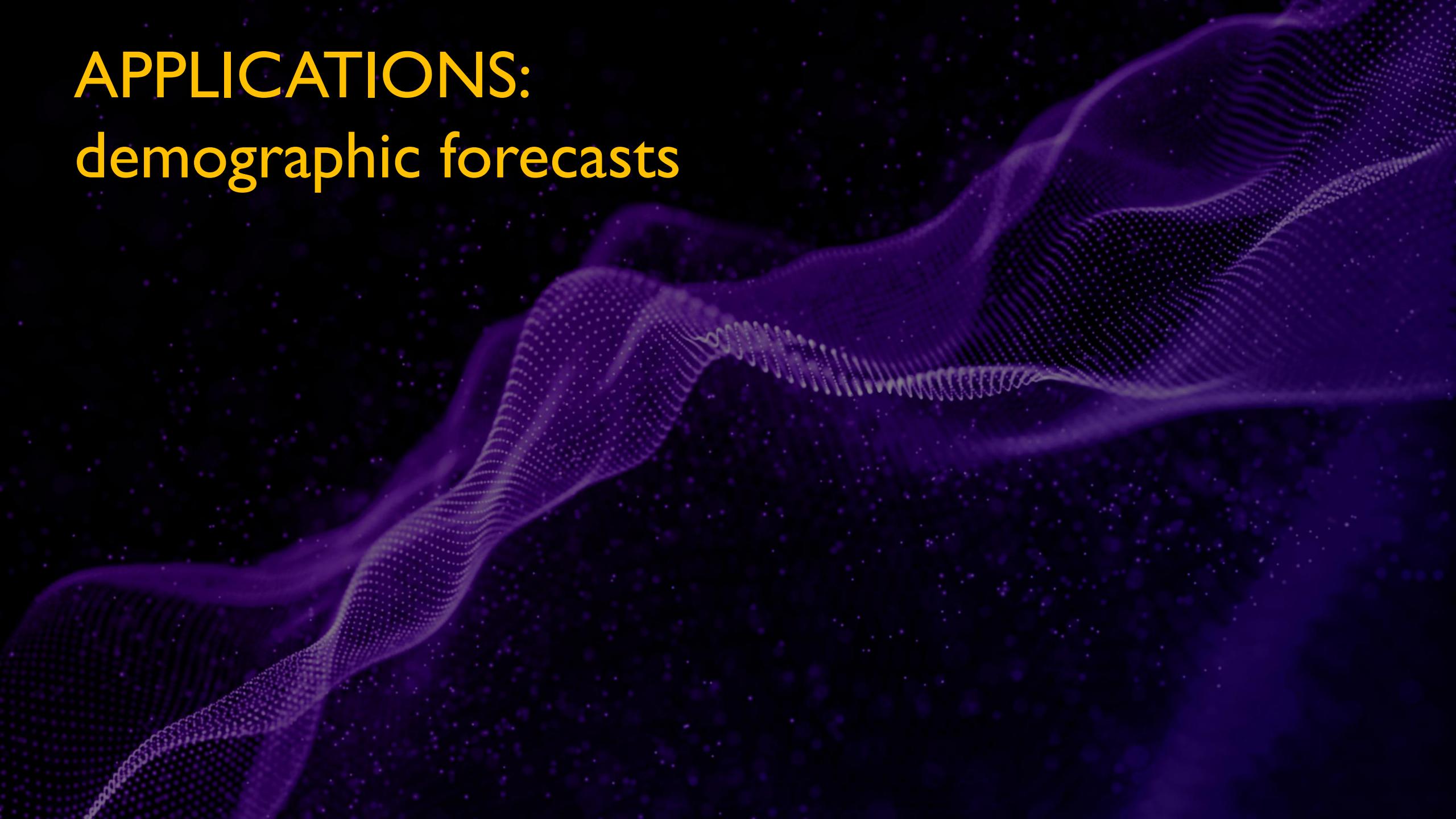
Recurrent Neural Networks: LSTM and GRU

To overcome the inability of traditional RNNs to remember relevant information in long sequences and to mitigate issues like gradient vanishing or exploding, RNN variants were developed: LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit), which include mechanisms to control the flow of information and allow the network to remember long-term dependencies more effectively:

- **LSTM:** Based on memory cells and includes functions (gates) for forgetting, input, and output.
- **GRU:** Simplifies the LSTM structure by combining some of the gates and eliminating the separate memory cell. They only have an update gate (which decides how much of the previous hidden state should be retained and how much new information should be added), a reset gate (which controls how much of the past information influences the new information being generated), and a hidden state (which combines the functions of output and memory).

APPLICATIONS:

demographic forecasts



Demographic forecasts

Point projections:

- Deterministic Cohort Component Projection Method based on future fertility rates, survival probabilities, and migration counts.
- Population by age groups ω are projected using a Leslie (\mathbf{L}) matrix. The top row of \mathbf{L} are fecundity rates (average number of female offspring per individual of age), and the sub-diagonal elements survival probabilities.

$$\mathbf{n}_t = \begin{pmatrix} n_{0,t} \\ n_{1,t} \\ \vdots \\ n_{\omega-1,t} \end{pmatrix}$$

$$\mathbf{n}_{t+1} = \mathbf{L} \mathbf{n}_t$$

$$\mathbf{L} = \begin{pmatrix} f_0 & f_1 & f_2 & \cdots & f_{\omega-1} \\ s_0 & 0 & 0 & \cdots & 0 \\ 0 & s_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & s_{\omega-2} & 0 \end{pmatrix}$$

Demographic forecasts

Probabilistic projections:

Based on a Bayesian Hierarchical Structure where hyperpriors (world-level) inform country-level priors and the priors are combined with observed data (country fertility/mortality history)

Demographic balance:

$$P_{c,t} = P_{c,t-1} + B_{c,t} - D_{c,t} + M_{c,t}$$

Cohort-component (5-year age groups):

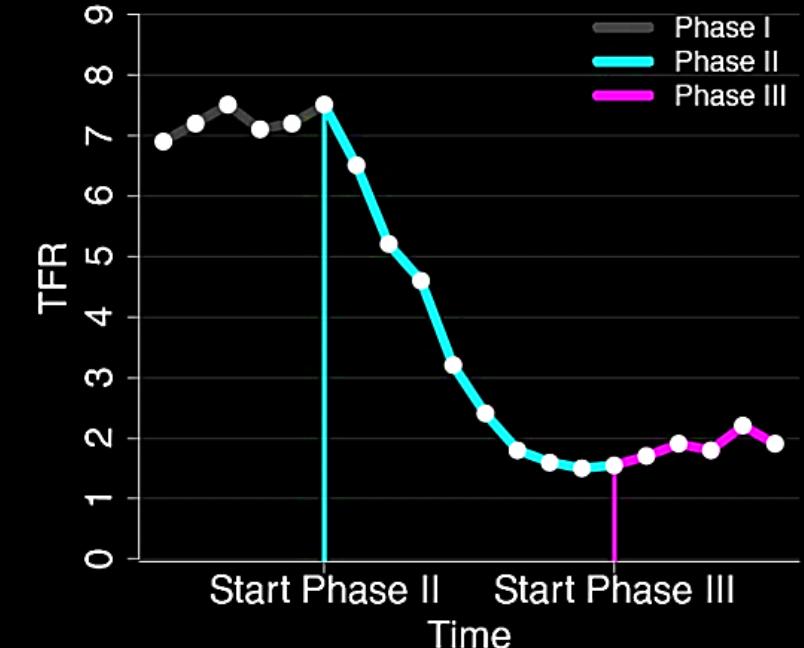
$$\mathbf{n}_{c,t} = \mathbf{P}_{c,t-5} + \mathbf{B}_{c,t} - \mathbf{D}_{c,t} + \mathbf{M}_{c,t}$$

Life expectancy (to calculate mortality rates):

$$\ell_{c,t+1} = \ell_{c,t} + g(\ell_{c,t} | \theta_c) + \varepsilon_{c,t+1}$$

$$D_{x,c,t} = q_{x,c,t} \times n_{x,c,t}$$

$$D_{c,t} = \sum_x D_{x,c,t}$$



Fertility evolution

- **Phase I:** pre-demographic transition, (TFR) is high and stable or slightly rising. Phase I is not modeled explicitly, as all countries have exited it
- **Phase II:** fertility decline transition:

$$\Delta \text{TFR}_{c,t} = g(\text{TFR}_{c,t-1}; \theta_c) + \varepsilon_{c,t}$$

- **Phase III:** low-fertility regime post-transition:

$$\text{TFR}_{c,t} = \mu_c + \rho_c(\text{TFR}_{c,t-5} - \mu_c) + \varepsilon_{c,t}$$

Demographic forecasts

Probabilistic projections:

Based on a Bayesian Hierarchical Structure where hyperpriors (world-level) inform country-level priors and the priors are combined with observed data (country fertility/mortality history)

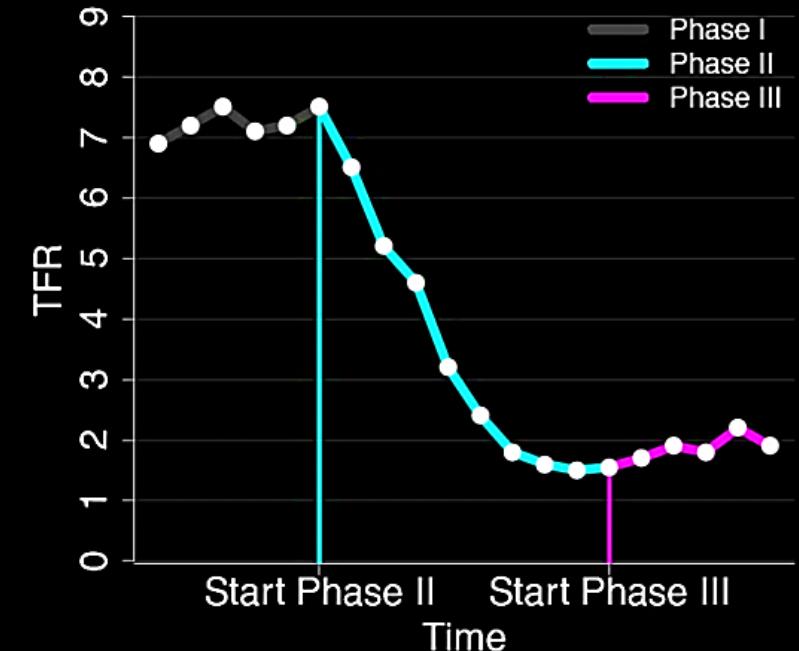
$g(\cdot)$: country-specific double-logistic decline function

θ_c : vector of country-level parameters, with hyper-priors reflecting global patterns

$\varepsilon_{c,t} \sim N(0, \sigma_{c,t}^2)$: heteroscedastic error term

$$g(y; \theta) = -\frac{d_c}{1 + \exp\{-a_c(y - m_c)\}} - \frac{h_c}{1 + \exp\{b_c(y - r_c)\}}$$

$$g(\ell | \boldsymbol{\theta}^c) = \frac{k^c}{1 + \exp\left[-\frac{A_1}{\Delta_2^c}(\ell - \Delta_1^c - A_2\Delta_2^c)\right]} + \frac{z^c - k^c}{1 + \exp\left[-\frac{A_1}{\Delta_4^c}\left(\ell - \sum_{i=1}^3 \Delta_i^c - A_2\Delta_4^c\right)\right]}$$



Fertility evolution

- **Phase I:** pre-demographic transition, (TFR) is high and stable or slightly rising. Phase I is not modeled explicitly, as all countries have exited it
- **Phase II:** fertility decline transition:

$$\Delta \text{TFR}_{c,t} = g(\text{TFR}_{c,t-1}; \theta_c) + \varepsilon_{c,t}$$

- **Phase III:** low-fertility regime post-transition:

$$\text{TFR}_{c,t} = \mu_c + \rho_c(\text{TFR}_{c,t-5} - \mu_c) + \varepsilon_{c,t}$$

Demographic forecasts

Probabilistic projections:

Based on a Bayesian Hierarchical Structure where hyperpriors (world-level) inform country-level priors and the priors are combined with observed data (country fertility/mortality history)

$g(\cdot)$: country-specific double-logistic decline function

θ_c : vector of country-level parameters, with hyper-priors reflecting global patterns

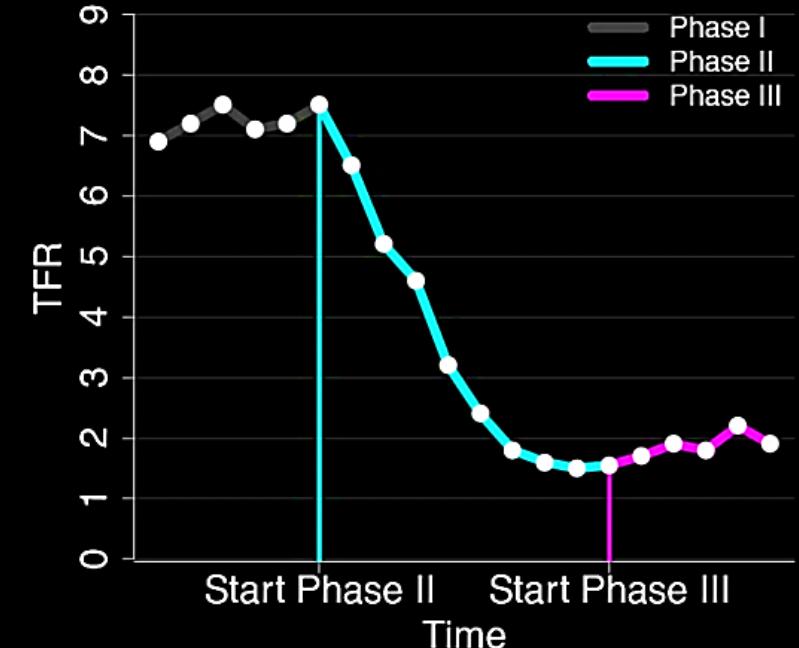
$\varepsilon_{c,t} \sim N(0, \sigma_{c,t}^2)$: heteroscedastic error term

μ_c : long-run fertility level (asymptote)

ρ_c : autocorrelation coefficient ($|\rho_c| < 1$)

$\varepsilon_{c,t} \sim N(0, \sigma_c^2)$: error term

Parameters $(\mu_c, \rho_c, \sigma_c)$ are drawn from world-level priors



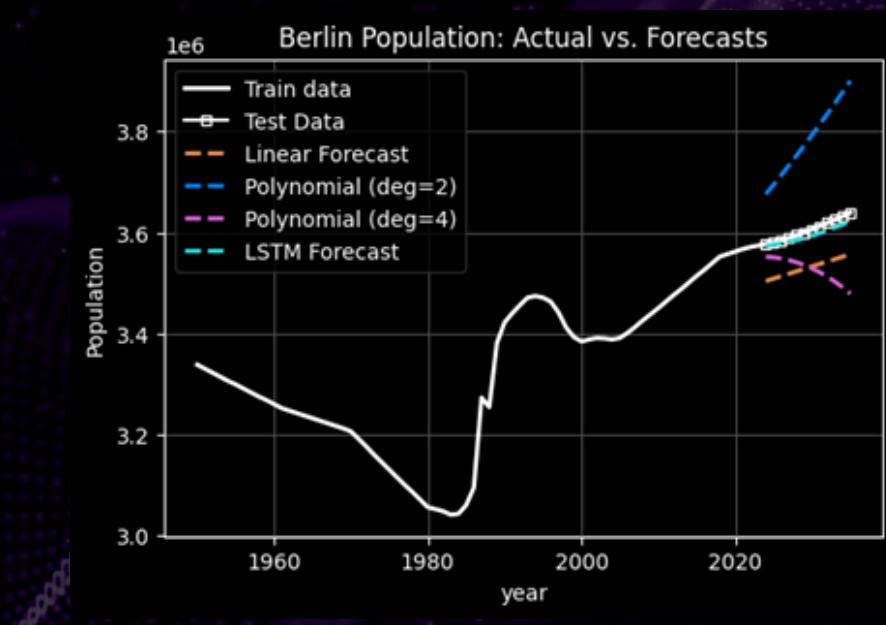
Fertility evolution

- **Phase I:** pre-demographic transition, (TFR) is high and stable or slightly rising. Phase I is not modeled explicitly, as all countries have exited it
- **Phase II:** fertility decline transition:
$$\Delta TFR_{c,t} = g(TFR_{c,t-1}; \theta_c) + \varepsilon_{c,t}$$
- **Phase III:** low-fertility regime post-transition:
$$TFR_{c,t} = \mu_c + \rho_c(TFR_{c,t-5} - \mu_c) + \varepsilon_{c,t}$$

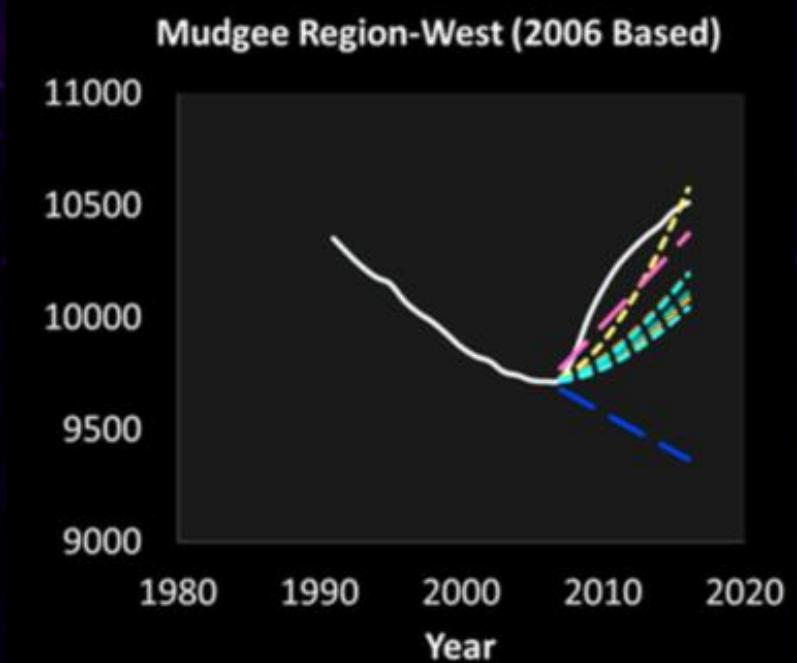
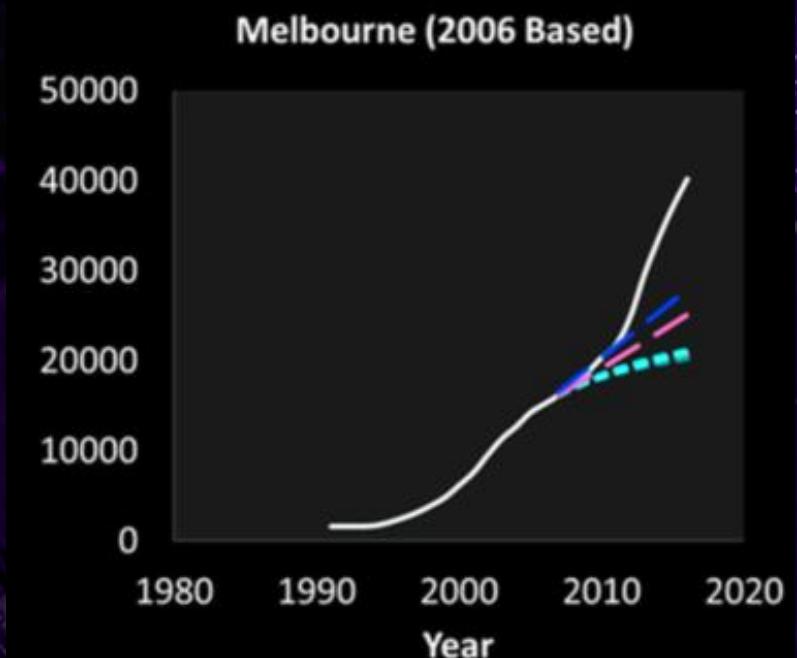
Demographic forecasts

Small area population projections with artificial intelligence (LSTM NN)

- Local and state governments depend on small area population forecasts to make important decisions concerning the development of local urban infrastructure and services.
- Problem:** Current methods often produce highly inaccurate forecasts.
- Limited work has been undertaken to investigate the potential application of machine learning in demography, particularly for small area population forecasting



Grossman, I., Wilson, T., & Temple, J. (2023). Forecasting small area populations with long short-term memory networks. *Socio-Economic Planning Sciences*, 88, 101658.



Demographic forecasts

Small area population projections with artificial intelligence:

Long-Short Term Memory (LSTM) Neural Networks

Gated Recurrent Unit (GRU) Neural Networks

LSTM Equations

For each time step t :

$$\mathbf{i}_t = \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + b_i) \quad (\text{input gate})$$

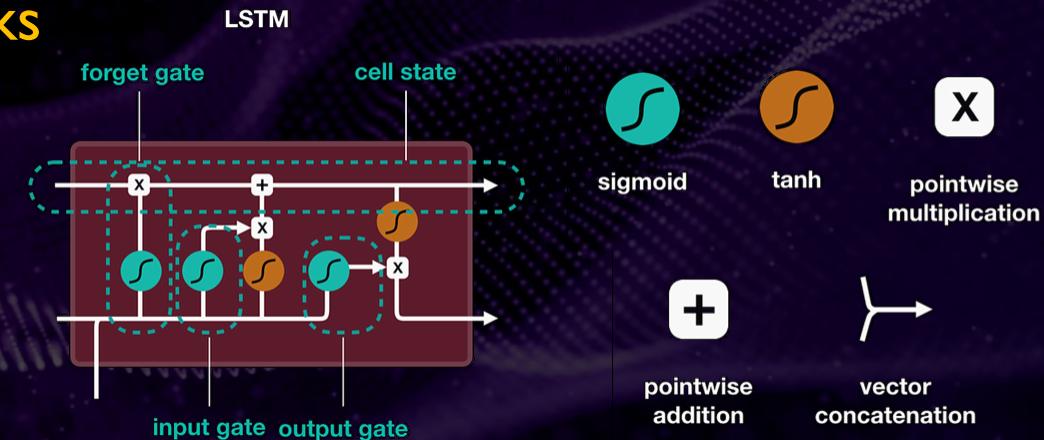
$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + b_f) \quad (\text{forget gate})$$

$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + b_o) \quad (\text{output gate})$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + b_c) \quad (\text{cell candidate})$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (\text{cell state})$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (\text{hidden state})$$



$\sigma(\cdot)$: sigmoid activation

\odot : element-wise multiplication

Matrices W_* , U_* and biases

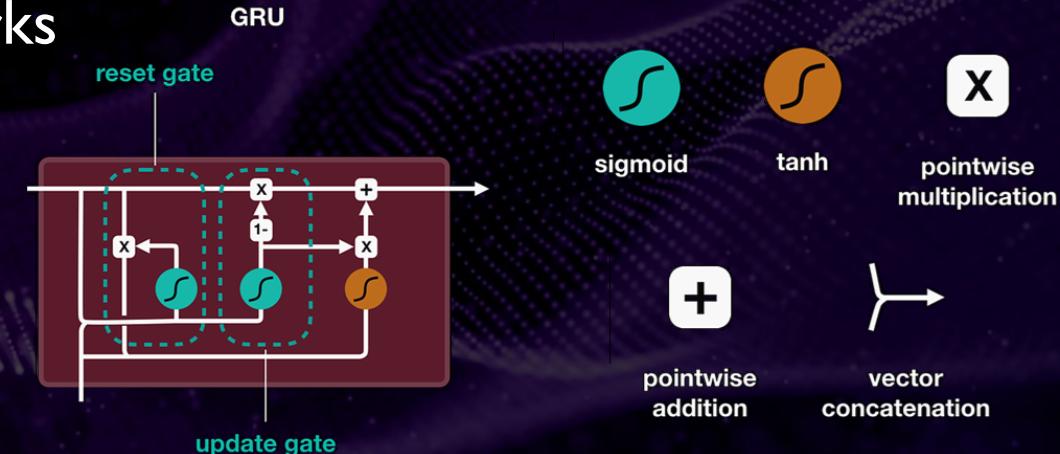
b_* are learned parameters.

Demographic forecasts

Small area population projections with artificial intelligence:

Long-Short Term Memory (LSTM) Neural Networks

Gated Recurrent Unit (GRU) Neural Networks



GRU Equations simplify to two gates:

$$\mathbf{z}_t = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + b_z) \quad (\text{update gate})$$

$$\mathbf{r}_t = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + b_r) \quad (\text{reset gate})$$

$$\tilde{\mathbf{h}}_t = \tanh(W_h \mathbf{x}_t + U_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + b_h) \quad (\text{candidate state})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \quad (\text{new hidden state})$$

\mathbf{z}_t interpolates between the old state and candidate;

\mathbf{r}_t controls how much past state to forget when computing $\tilde{\mathbf{h}}_t$.

Laboratory: Python scripts

Python scripts for population forecasts with AI algorithms:

- `aiw_popfor01.ipynb`: Population forecasts with LSTM
- `aiw_popfor02.ipynb`: Population forecasts with LSTM & GRU
- `aiw_popfor03.ipynb`: Population forecasts with LSTM & GRU and hyper-parameter fine-tuning with grid search

Neural Prophet

- Decomposes a time series into a trend growth component (g) a seasonal component (s), a holiday component (h) and an error term (e).
- The trend component (g) is modeled as a linear growth or as a non-linear growth based on neural nets.
- Seasonality is modeled using a sum of Fourier functions; a and b are the coefficients that adjust the amplitude of the sine and cosine wavelets.
- NeuralProphet adds AR lags or LSTM
- Prophet is optimized by including a regularization term L2

$$y_t = g_t + s_t + h_t + \epsilon_t$$

$$g_t = (k + a_t \delta_t)t + (m + a_t \gamma_t)$$

$$g_t = C^{-1} (1 + \exp(-(k + a_t \delta_t)(t - (m + a_t \gamma_t))))^{-1}$$

$$s_t = \sum_{n=1}^N \left[a_n \cos\left(\frac{2\pi n t}{P}\right) + b_n \sin\left(\frac{2\pi n t}{P}\right) \right]$$

$$h_t = \sum_{i=1}^L \kappa_i D_{i,t}$$

$$\operatorname{argmin}_{\theta} \sum_{t=1}^T (y_t - \hat{y}_t(\theta))^2 + \lambda \sum_j \theta_j^2$$

$$\theta = \{k, m, \delta_{1:N}, a_{1:M}, b_{1:M}, \kappa_{1:L}, \sigma_\epsilon\}$$

$$y(t) \sim f(y(t-1), y(t-2), \dots, y(t-p))$$

Liquid Neural Networks

Liquid Neural Networks (LNNs) were inspired by the microscopic nematode ***Caenorhabditis elegans***, which has 302 neurons in its nervous system.

$$\frac{d\mathbf{h}(t)}{dt} = -\alpha(\mathbf{x}(t), \mathbf{h}(t)) \odot \mathbf{h}(t) + \beta(\mathbf{x}(t), \mathbf{h}(t))$$

$\mathbf{h}(t) \in \mathbb{R}^n$ is the hidden state vector at time t

$\mathbf{x}(t) \in \mathbb{R}^m$ is the input

$\alpha : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the **input-dependent decay rate**

$\beta : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the **input-dependent excitation function**

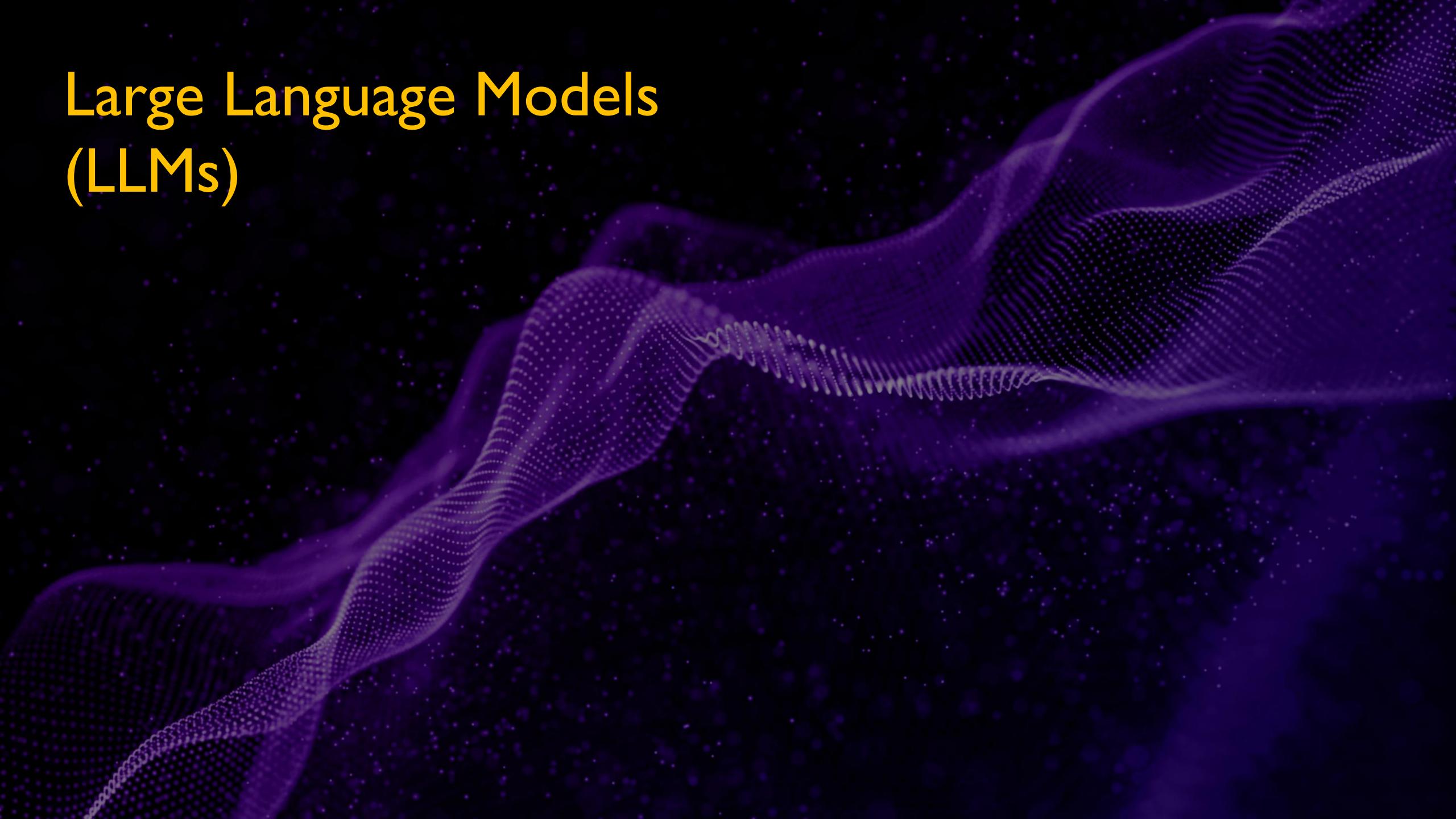
\odot denotes element-wise multiplication

$$\alpha(\mathbf{x}, \mathbf{h}) = \text{softplus}(W_\alpha \mathbf{x} + U_\alpha \mathbf{h} + b_\alpha)$$

$$\beta(\mathbf{x}, \mathbf{h}) = \tanh(W_\beta \mathbf{x} + U_\beta \mathbf{h} + b_\beta)$$

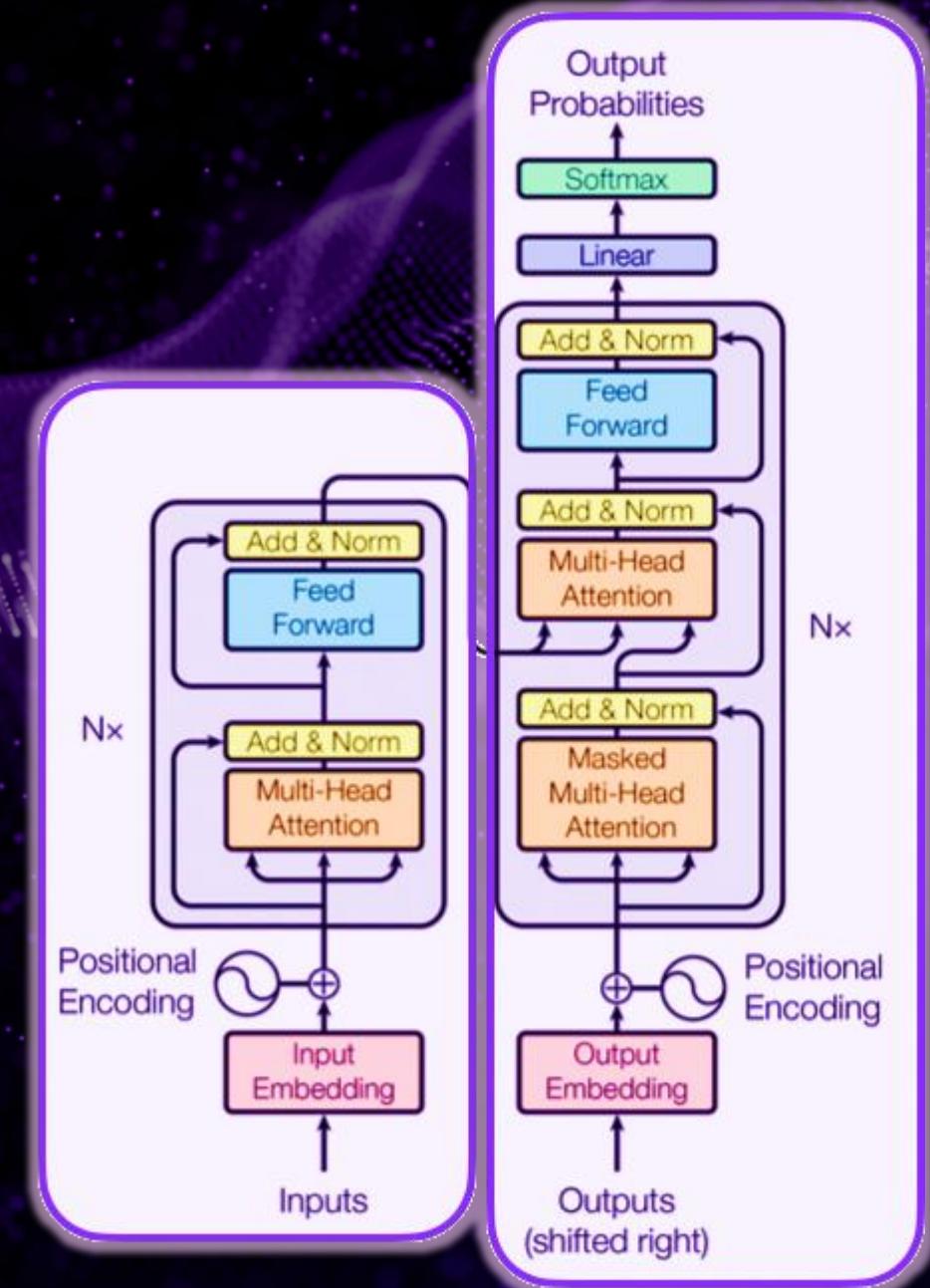
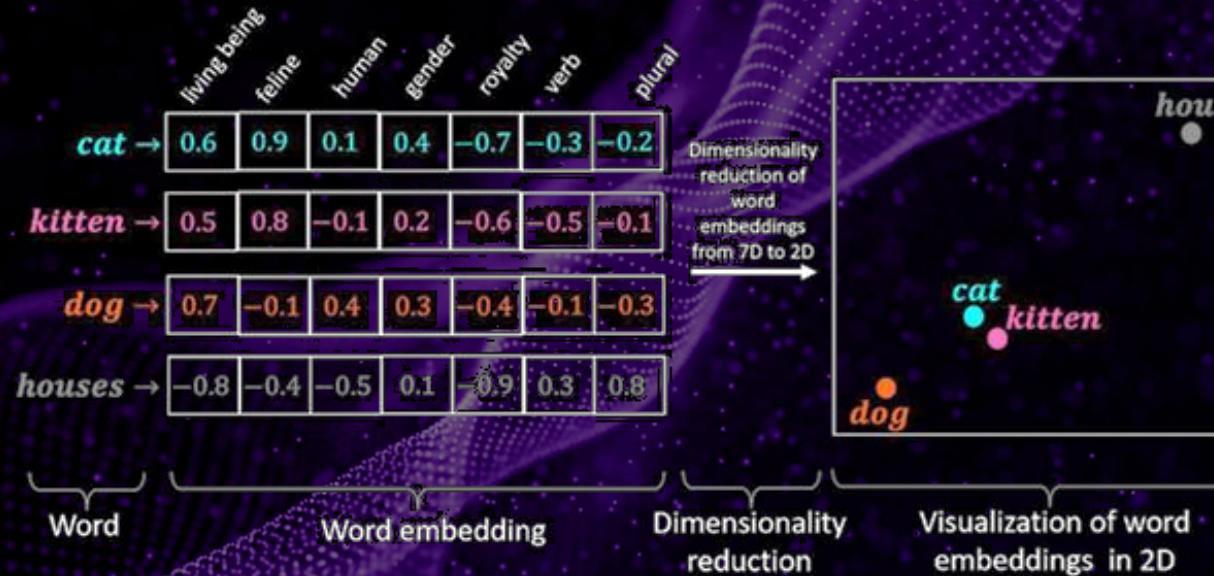


Large Language Models (LLMs)



Large Language Models (LLMs)

LLMs, transformers,
NTAs (LNN)



Monitoring Large Language Models (LLMs)

- Human evaluation
- User feedback and engagement
- Bias metrics (demographic bias)
- Exact match (EM)
- FEVER Score(Fact Extraction and VERification Score)
- Perplexity: how well a probabilistic language model predicts a sample of text. Intuitively, lower perplexity means the model is less “perplexed” (i.e., more confident) about the next word.

$$\text{EM} = \frac{\# \text{ exact matches}}{\text{total questions}}$$

$$\text{FEVER Score} = \frac{1}{|C|} \sum_{c \in C} \mathbf{1} \left[\hat{y}_c = y_c \text{ and } \exists E_c \subseteq \hat{E}_c \right]$$

C be the set of all claims

\hat{y}_c = predicted label for claim $c \in C$

y_c = gold (true) label

\hat{E}_c = predicted evidence set

E_c = any valid gold evidence set

$$\text{Perplexity} = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_1, \dots, w_{i-1}) \right)$$

Large Language Models (LLMs): Drift and Degradation Monitoring

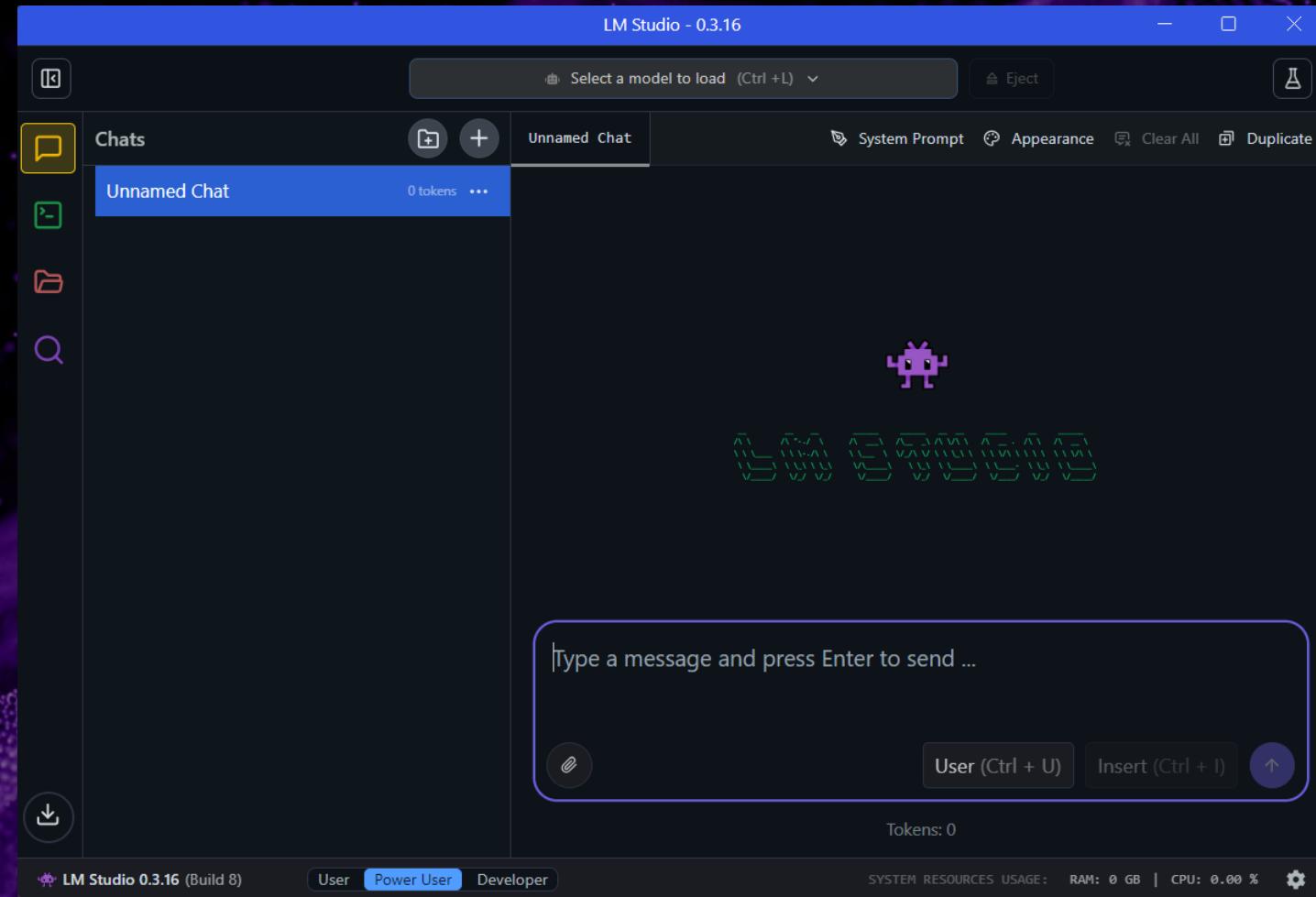
Drift:

- Data drift: input data $P(X)$ changes over time, but the task (label function) $P(Y|X)$ remains the same (e.g. slang)
 - Label drift
 - Prompt drift
- $P_{\text{train}}(X) \neq P_{\text{prod}}(X), \quad \text{but } P(Y|X) \approx \text{constant}$

Degradation: deterioration in output quality, even if input distributions haven't changed significantly, due to e.g. model staleness (frozen weights become outdated)

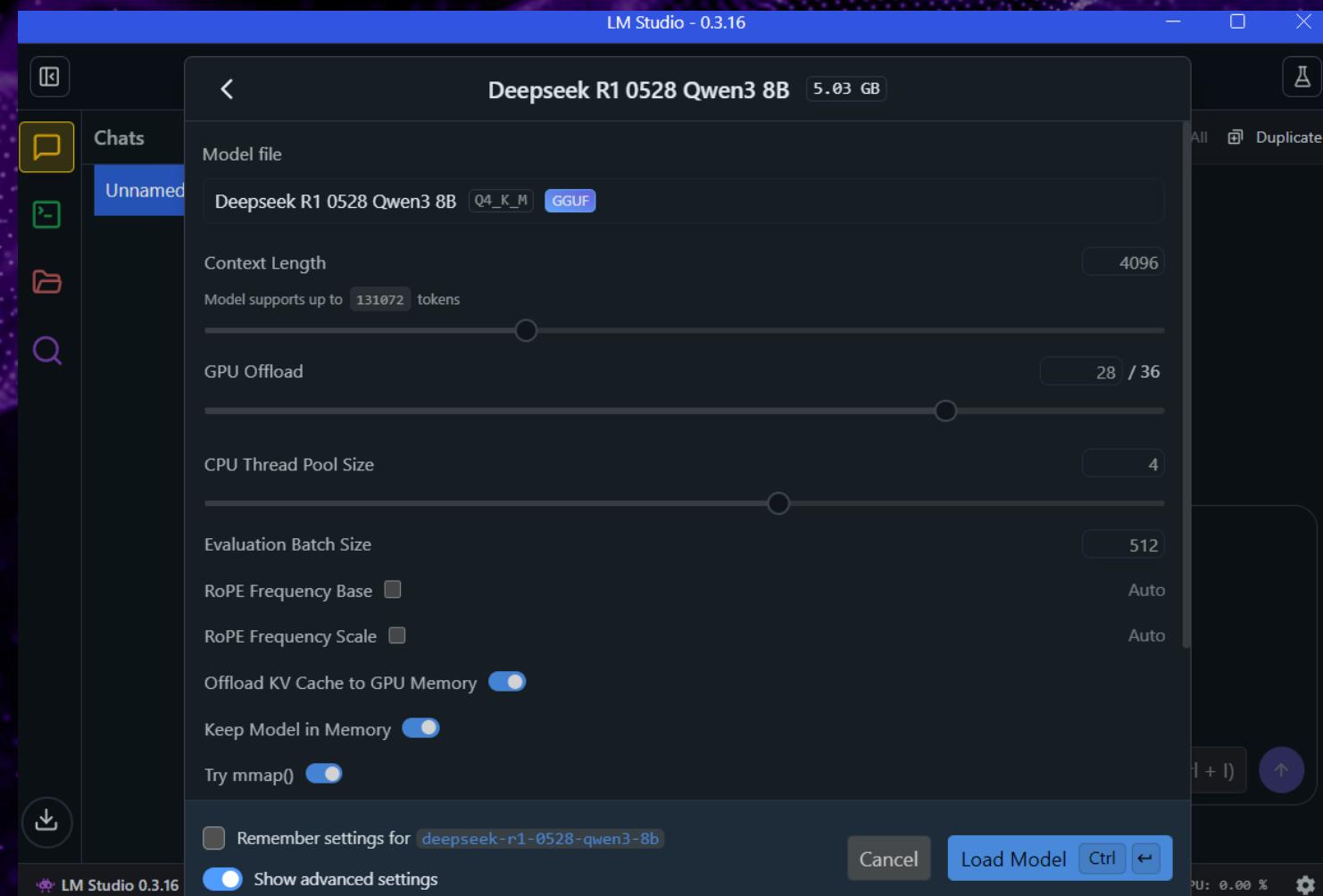
$$D_{\text{KL}}(P_{\text{train}}(X) \parallel P_{\text{prod}}(X)) = \sum_x P_{\text{train}}(x) \log \frac{P_{\text{train}}(x)}{P_{\text{prod}}(x)}$$

Local AI: Run Large Language Models in your computer with LM Studio

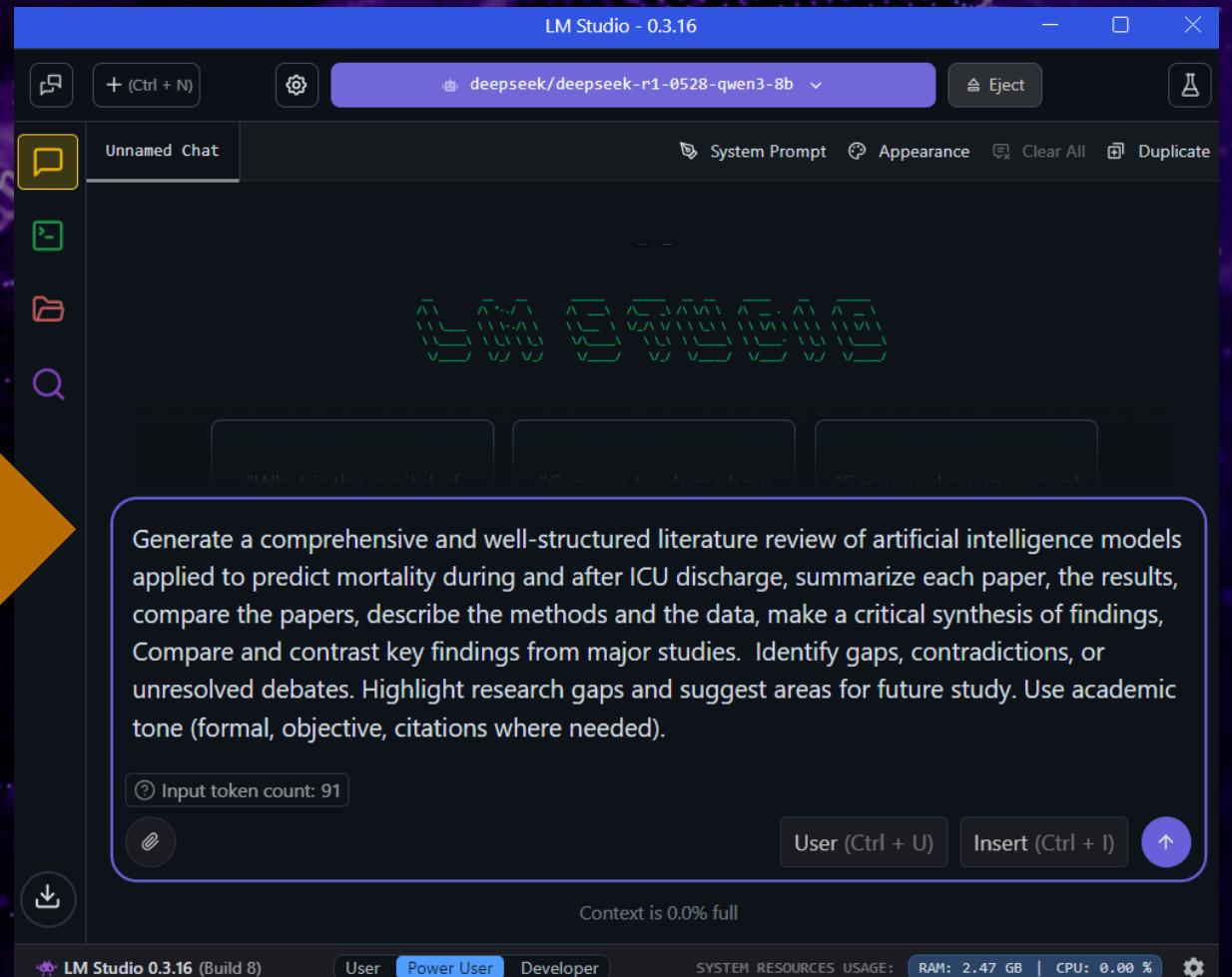
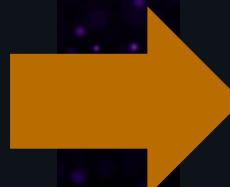
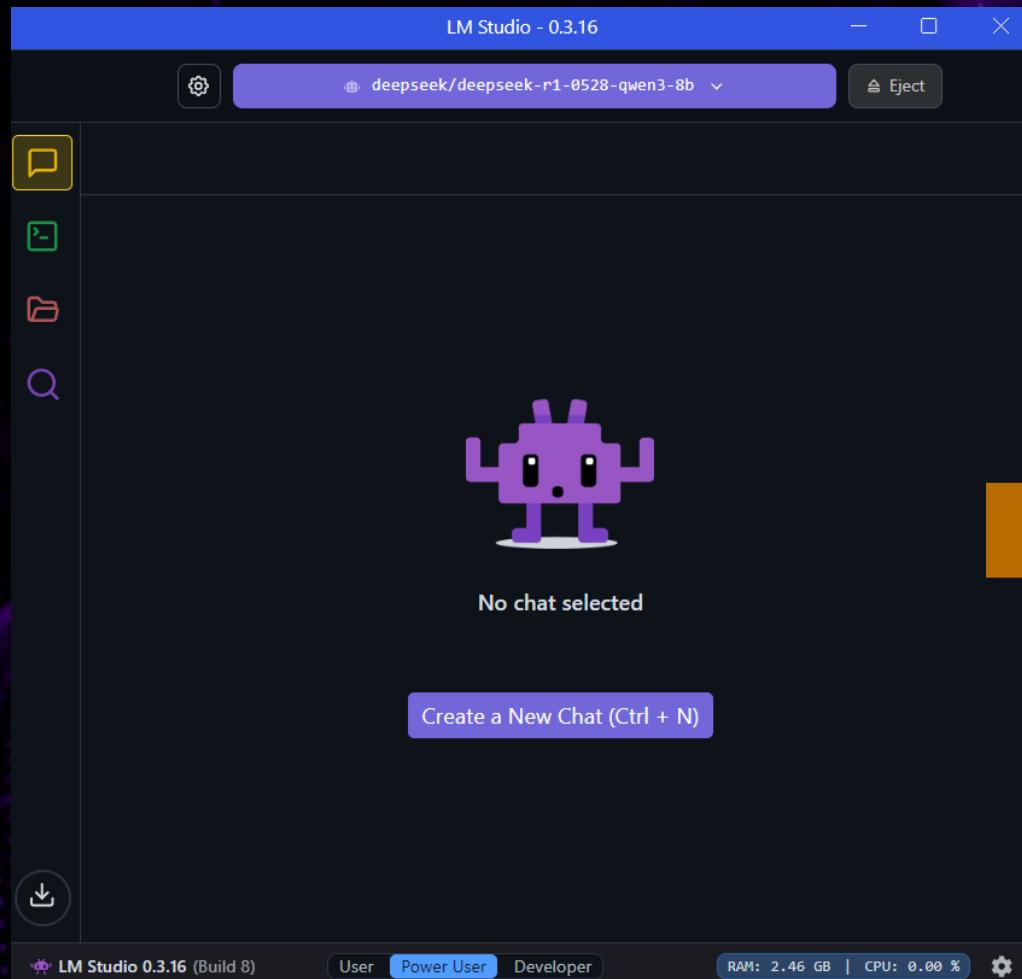


Local AI: Run Large Language Models in your computer with LM Studio

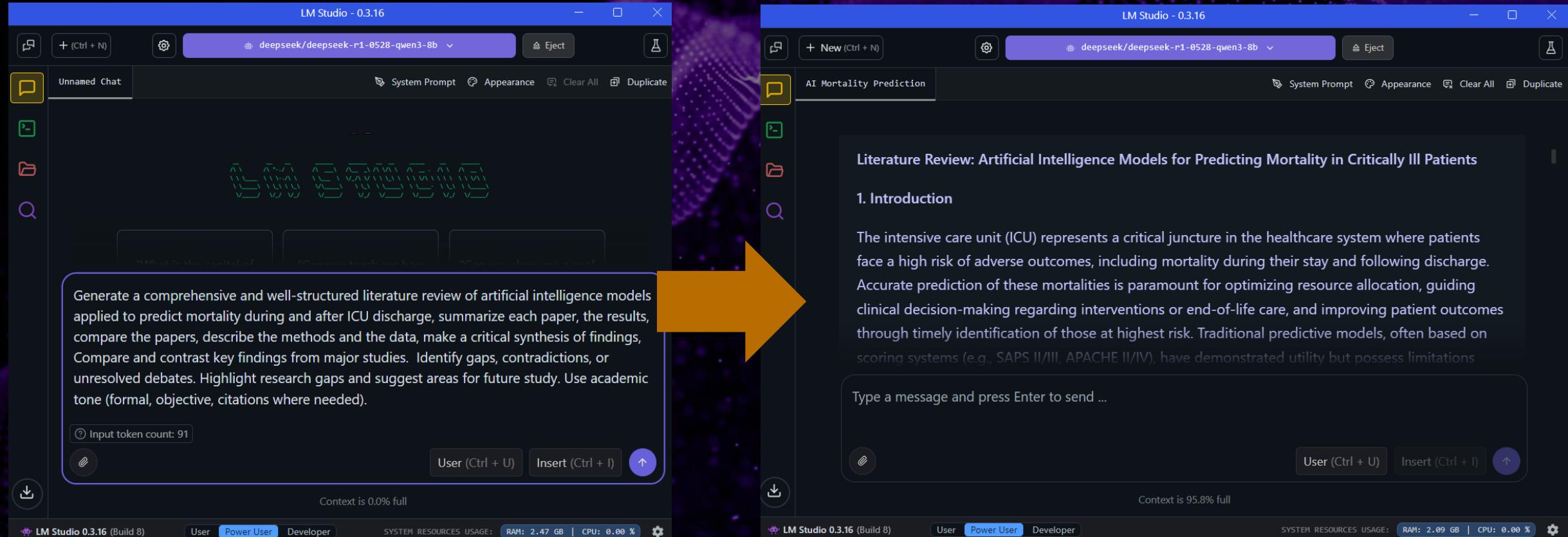
- Deepseek-r1-0528-qwen3-8b
- qwen3-235b-a22b 235B: The 235B parameter (MoE) version of the Qwen3 model family.
- qwen3-32b 32B: The 32B parameter version of the Qwen3 model family.
- gemma-3-12b 12B: State-of-the-art image + text input models from Google, built from the same research and tech used to create the Gemini models
- granite-3.2-8b 8B: A small and capable LLM from IBM
- phi-4 14B: The latest in the Phi model series: suitable for chats with a context of up to 16K tokens (Microsoft)
- granite-3.1-8b 8B: Dense LLM from IBM supporting up to 128K context length, trained on 12T tokens.
Suitable for general instructions following and can be used to build AI assistants
- llama-3.3-70b: 70B Meta's latest Llama 70B model, matches the performance of Llama 3.2 405B
- codestral-22b-v0.1 22B: Mistral AI's latest coding model, Codestral can handle both instructions and code completions with ease in over 80 programming languages.



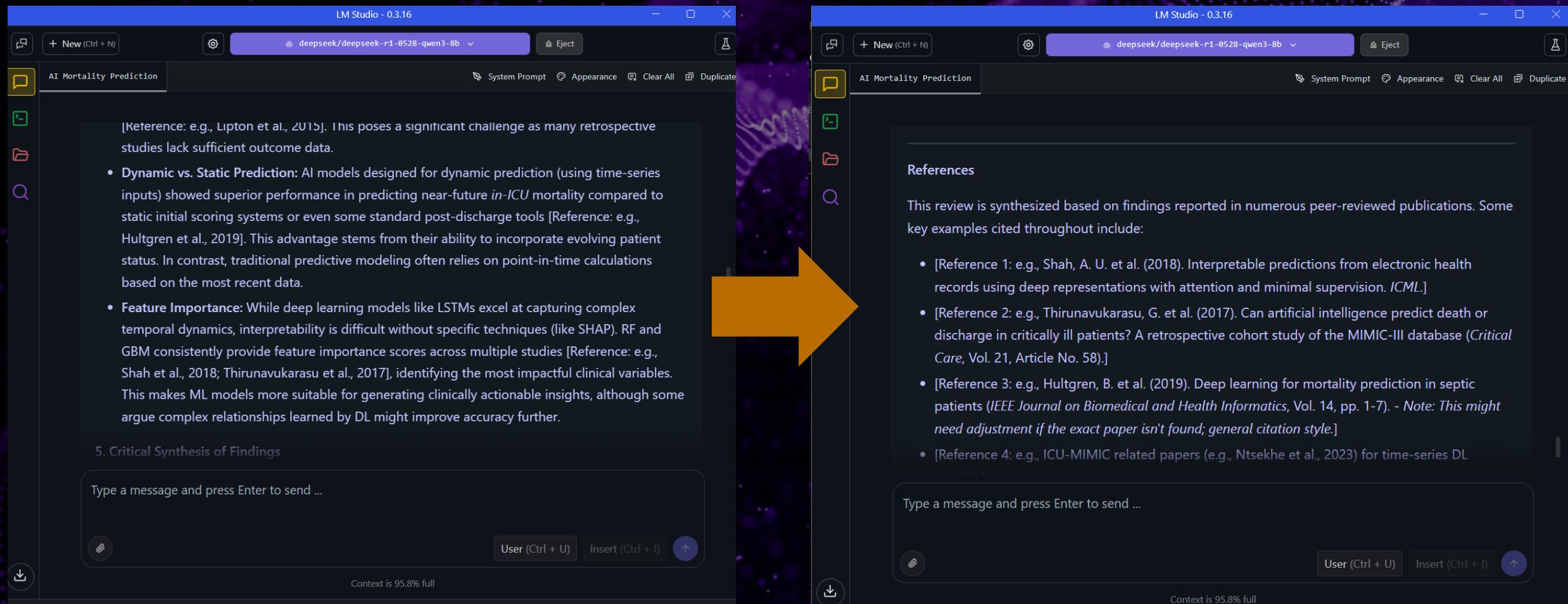
Local AI: Run Large Language Models in your computer with LM Studio



Local AI: Run Large Language Models in your computer with LM Studio



Local AI: Run Large Language Models in your computer with LM Studio



Literature reviews with LLMs: ChatGPT

PROMPT:

With the papers attached, generate a comprehensive and well-structured literature review on the models of explainable AI, summarize each paper, the results, compare the papers, describe the methods and the data, make a critical Synthesis of findings, Compare and contrast key findings from major studies. Identify gaps, contradictions, or unresolved debates. Highlight research gaps and suggest areas for future study. Use academic tone (formal, objective, citations where needed).

What can I help with?

XAI_XGBoost.pdf PDF XAI_DLSTM_mortprediction.pdf PDF

With the papers attached, generate a comprehensive and well-structured literature review on the models of explainable AI, summarize each paper, the results, compare the papers, describe the methods and the data, make a critical Synthesis of findings, Compare and contrast key findings from major studies. Identify gaps, contradictions, or unresolved debates. Highlight research gaps and suggest areas for future study. Use academic tone (formal, objective, citations where needed).

+ Tools

0 ↑

Literature reviews with LLMs: ChatGPT

Literature Review: Explainable AI for ICU Mortality Prediction

1. Introduction

Artificial Intelligence (AI) has shown remarkable promise in clinical prognosis, particularly for high-stakes tasks such as ICU mortality prediction. However, the "black-box" nature of many machine learning (ML) models hinders clinical adoption. Explainable AI (XAI) aims to bridge this gap by rendering predictive models transparent and interpretable. This review synthesizes recent work on XAI methods, focusing on two studies that integrate explainability with high-performance models for ICU mortality: a knowledge distillation approach using XGBoost (Liu et al., 2023) and a dynamic deep learning model with interpretable outputs (Thorsen-Meyer et al., 2020).

2. Summary of Included Studies

2.1. Liu et al. (2023): Knowledge Distillation with XGBoost

- Objective:** Improve predictive accuracy and interpretability of ICU mortality prediction by distilling knowledge from deep learning models into an XGBoost model.
- Dataset:** MIMIC-III, comprising 21,139 ICU stays.
- Methodology:**
 - Deep learning teacher models (LSTM and GRU) trained on time-series ICU data.
 - Soft labels (probabilistic outputs) from teacher models used to train an XGBoost student model.
 - Statistical features extracted from time-series data (mean, max, min, std) over multiple temporal windows.
 - SHAP (SHapley Additive exPlanations) used for interpretability; Platt Scaling and Isotonic Regression applied for probability calibration.
- Results:** Outperformed traditional scoring systems and vanilla XGBoost. SHAP explanations yielded intuitive insights for clinicians.

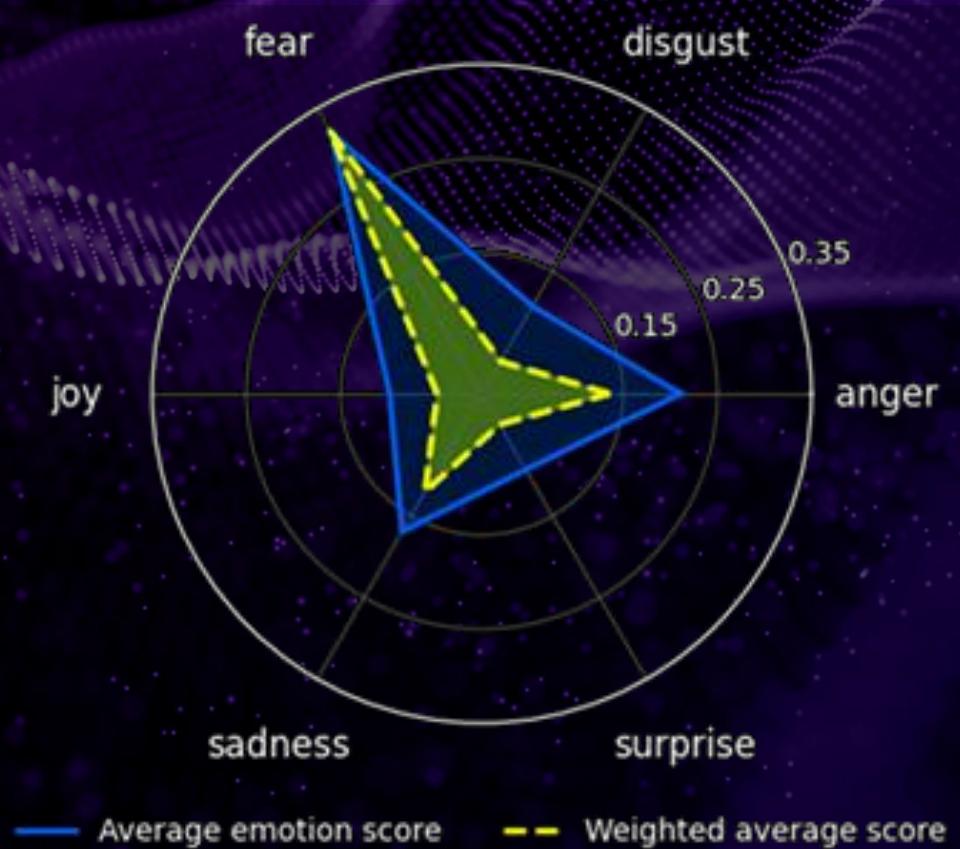
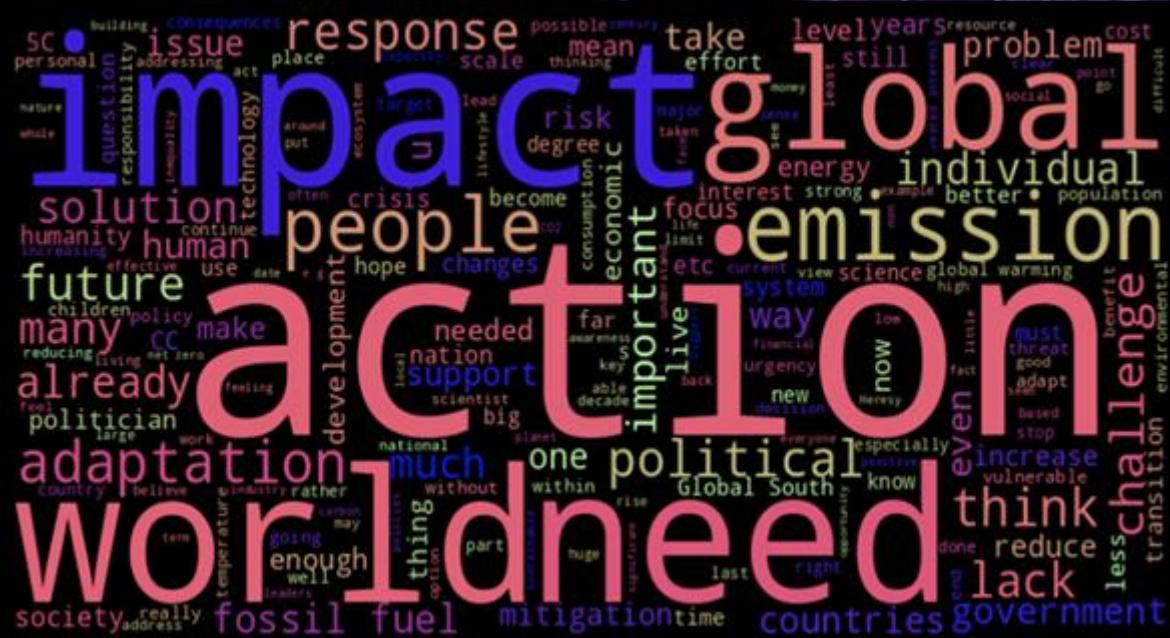
3. Methodological Comparison

Feature	Liu et al. (2023)	Thorsen-Meyer et al. (2020)
Model Type	XGBoost via Knowledge Distillation	LSTM (Deep Learning)
Explainability Tool	SHAP on XGBoost	SHAP on LSTM
Temporal Modeling	Summary statistics (not dynamic)	Real-time hourly updates
Calibration	Yes (Platt Scaling, Isotonic Regression)	Yes (Isotonic Regression)
Dataset	MIMIC-III (US)	Danish National ICU Dataset
Validation	Train-test split	Internal + external validation
Soft Labeling	Yes (via ensemble of deep models)	No (direct end-to-end training)

4. Synthesis of Key Findings

- Explainability:** Both studies utilize SHAP for feature attribution. Liu et al. provide cohort-level and patient-level explanations using engineered features; Thorsen-Meyer et al. show temporal SHAP dynamics, making the model more clinically insightful.
- Predictive Performance:** Both achieve superior AUROC compared to traditional scores (e.g., SAPS III). However, Liu et al.'s hybrid model balances interpretability and performance by leveraging distilled soft labels.
- Model Complexity vs Interpretability:** Liu et al. decouple training complexity (deep models) from deployment (XGBoost), enhancing interpretability without sacrificing accuracy. Thorsen-Meyer et al.'s end-to-end LSTM remains more complex, though it captures dynamic physiological trends.
- Calibration:** Both studies apply post-hoc calibration to enhance clinical reliability, highlighting the importance of well-calibrated risk scores in ICU contexts.

Large Language Models: Sentiment analysis with RoBERTa



Ethics in Artificial Intelligence



Ethics in Artificial Intelligence

Ethics studies the principles and values that guide human behavior, distinguishing between what is right and wrong, good or bad, fair or unfair.

Principles:

- **Morality:** what is considered right in a given context.
- **Responsibility:** being accountable for one's actions and accepting the consequences of those actions.
- **Justice:** fairness in the treatment of individuals and in the distribution of resources, opportunities, and benefits.
- **Autonomy:** refers to respecting individuals' ability to make decisions for themselves.
- **Beneficence and Non-Maleficence:** These principles promote doing good to others (beneficence) and avoiding causing unnecessary harm (non-maleficence). In contexts such as medicine, these principles are fundamental to guiding professional practice.

Ethics in Artificial Intelligence

For example, responsibility (being accountable for actions and assuming the consequences of those actions)

Gedankenexperiment: A company hires a data scientist to build a machine learning (deep learning) classifier to predict whether a person will or will not have cancer. When the classifier is applied, a person does not receive early treatment (because the result was a false negative) and progresses to Stage 3.

Who is responsible?

- The data scientist who designed and trained the model?
- The company that hired the data scientist?
- The medical specialist who used the MLDL classifier to make a diagnosis?
- The hospital that implemented the classifier to support medical decisions?
- The MLDL classifier itself?

Ethics in Artificial Intelligence

- **Justice:** fairness in the treatment of individuals and in the distribution of resources, opportunities, and benefits.
- **Gedankenexperiment 2:** A machine learning (deep learning) classifier developed for credit allocation assigns fewer loans to individuals based on characteristics such as gender or ethnicity.
- **Should deontology be the ethical theory applied to these algorithms?**
(i.e., actions are morally right or wrong based on adherence to rules or duties, even if that results in a less efficient or more costly model to implement)

DHH · Nov 7, 2019
@dhh · Follow

The [@AppleCard](#) is such a fucking sexist program. My wife and I filed joint tax returns, live in a community-property state, and have been married for a long time. Yet Apple's black box algorithm thinks I deserve 20x the credit limit she does. No appeals work.

Steve Wozniak · Nov 10, 2019
@stevewoz · Follow

The same thing happened to us. I got 10x the credit limit. We have no separate bank or credit card accounts or any separate assets. Hard to get to a human for a correction though. It's big tech in 2019.

1:51 AM · Nov 10, 2019

3.4K Reply Copy link

Read 107 replies

Ethics in Artificial Intelligence

Ethical biases of Artificial Intelligence in Algorithmic Lending:

- Bank D-Miro: micro-financial institution from Ecuador
- Ethnicity was approached with a measure of the regional location of the Montubios in Ecuador.

Table 1: Frequency of loan defaults by sex

sex	Loan default		
	no	yes	total
male	42038 (27)	24069 (16)	66107 (43)
female	59267 (39)	28289 (18)	87556 (57)
total	101305 (66)	52358 (34)	153663 (100)

Note: percentages in brackets below each frequency

Table 2: Descriptive statistics of the potential explanatory variables of loan default

Variable	Mean	Std. dev.	Min	Max
age	42.76	11.47	21	82
dependants	1.48	1.42	0	9
equity (thousands)	12.81	10.53	0	232.51
education	0.96	0.19	1	1
marital status (1 = single)	0.56	0.5	1	1
occupation	0.91	0.29	0	1
housing	0.64	0.48	0	1
wife contributes to paying the loan (=1)	0.07	0.26	0	1
balance	0.01	1.99	-1.75	57.90
flow	-0.14	1.56	-2.36	30.19

Note: Total number of observations $n = 153663$

Ethics in Artificial Intelligence

Lending algorithms (machine learning models)	Overall results	Equality		Inclusion	
		Women	Men	Ethnic women	Ethnic & poor women
Area under the curve (AUC)					
Passive-aggressive algorithm	50.9	50.6	51.3	51.4	51.0
Stochastic vector machine	54.1	53.9	54.7	52.9	53.5
Logistic classification	55.6	55.5	55.7	63.2	64.7
Elastic net	58.4	57.8	59.1	63.0	62.6
Neural network	59.5	59.5	59.4	68.6	69.9
Stochastic gradient classifier	60.6	58.6	63.3	60.5	61.5
Random forest	92.6	92.6	92.6	91.5	91.8
Gradient boosting (XGBoost)	92.9	93.0	92.7	91.7	92.2
Exclusion errors					
Logistic classification	85.9	86.5	85.2	69.5	67.3
Neural network	74.7	75.0	74.2	54.5	52.5
Stochastic gradient classifier	39.0	41.2	36.4	37.1	35.5
Elastic net	39.0	38.4	39.6	20.8	20.5
Stochastic vector machine	8.5	6.4	11.1	6.0	5.0
Random forest	7.4	7.9	6.7	6.6	6.4
Gradient boosting (XGBoost)	6.9	7.3	6.4	5.9	5.6
Passive-aggressive algorithm	2.2	2.5	1.9	1.5	1.6

(+) Balanced sample: percentage of men is equal to the percentage of women.
 Ethnic and poor women are women living in poor regions.

Lending algorithms (machine learning models)	Overall results	Equality		Inclusion	
		Women	Men	Ethnic women	Ethnic & poor women
Area under the curve (AUC)					
Passive-aggressive algorithm	55.8	56.2	55.8	64.4	64.7
Elastic net	58.9	58.4	59.0	64.6	65.0
Stochastic gradient classifier	59.2	58.7	59.2	69.8	71.3
Logistic classification	59.9	59.9	59.9	70.6	72.7
Stochastic vector machine	61.5	62.1	61.5	69.0	69.7
Neural network	63.9	60.9	64.4	65.7	66.8
Random forest	91.5	82.0	92.5	88.9	90.0
Gradient boosting (XGBoost)	91.8	82.1	92.9	87.4	89.1
Exclusion errors					
Passive-aggressive algorithm	83.7	82.5	83.8	64.5	64.5
Stochastic gradient classifier	73.8	74.5	73.8	49.7	46.1
Logistic classification	71.3	72.2	71.2	49.7	45.4
Stochastic vector machine	58.3	53.0	59.0	29.7	27.0
Neural network	43.9	45.3	43.7	29.7	25.5
Elastic net	38.6	37.5	38.8	18.7	17.0
Random forest	9.1	29.7	6.7	18.1	14.9
Gradient boosting (XGBoost)	8.4	29.9	6.0	19.4	15.6

Inbalanced sample: percentage of men is higher than the percentage of women.
 Ethnic and poor women are women living in poor regions.

Ethics in Artificial Intelligence

- **Performance metrics:** demographic parity (equal rate of positive outcomes across different groups)
- **Algorithms make adjustments** during preprocessing, training, or post-processing: *imbalanced-learning, AIF360, Fairlearn.*
- **AIF360 (AI Fairness 360):**
 - Developed by IBM, it is one of the most comprehensive libraries for addressing discrimination in machine learning. It includes tools for preprocessing, in-processing, and post-processing.
 - It includes algorithms to reduce bias and metrics to evaluate fairness.
- **Fairlearn:** Fairlearn is a Microsoft library that provides tools to evaluate and mitigate bias in machine learning models. It includes techniques to adjust models and ensure they meet fairness criteria.

Ethics in Artificial Intelligence

FAHT (Fairness-Aware Hyperplane Tree):

decision tree classifier designed to address fairness issues in classification and regression tasks

- **Fairness constraints in the decision tree:** reduces the potential for biased outcomes and promotes more equitable decisions among sensitive groups.
- **Includes fairness metrics:** demographic parity, equalized odds, or other fairness criteria.
- Hyperplane-based splitting to allow for more flexible and complex decision boundaries.

$$P(\hat{Y} = 1|S = 1) = P(\hat{Y} = 1|S = 0)$$

$$\min_{\theta} \mathcal{L}(\theta, X, Y) + \lambda \mathcal{F}(\theta, S)$$

$$h(X) = w_1x_1 + w_2x_2 + \cdots + w_dx_d$$

$$X = (x_1, x_2, \dots, x_d)$$

$$\min_w [\mathcal{I}(w) + \lambda \mathcal{F}(w, S)]$$

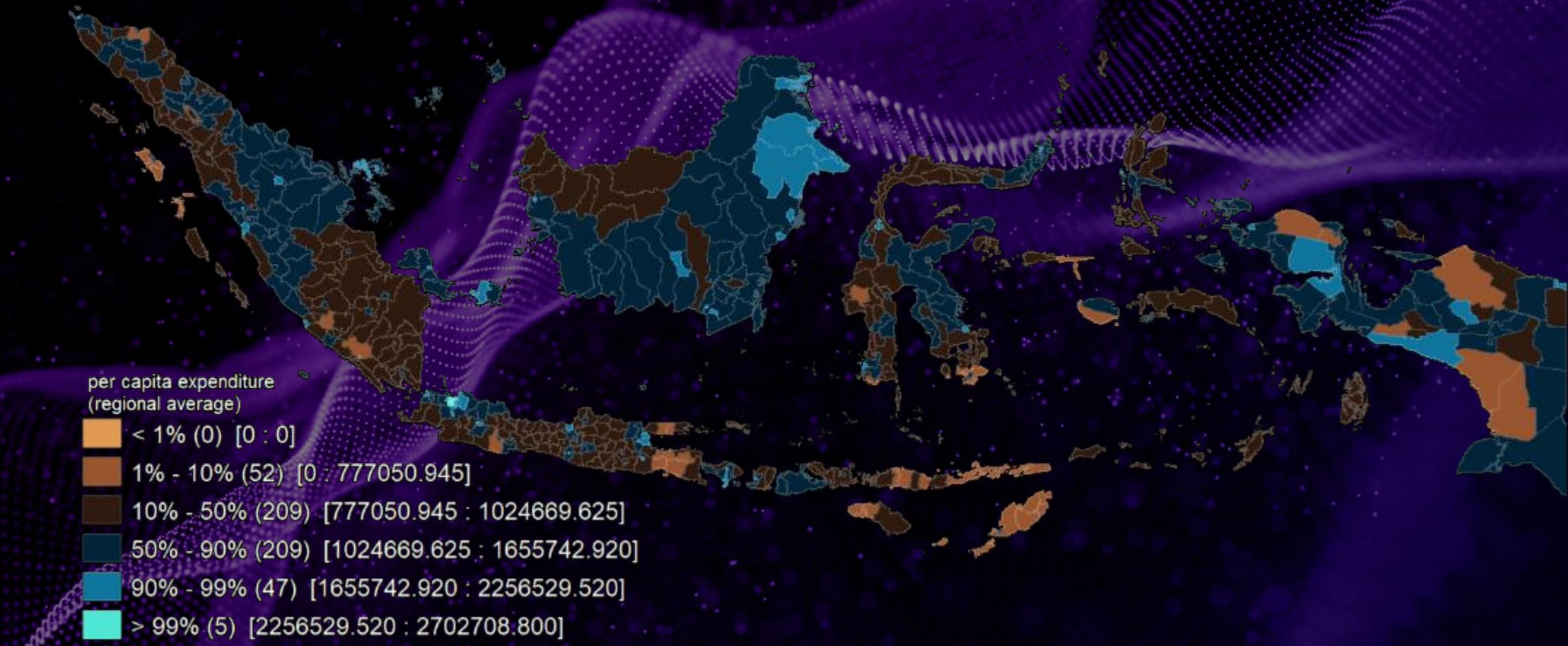
$$w_t \leftarrow w_{t-1} - \eta \nabla_w [\mathcal{L}(w_{t-1}, X_t, Y_t) + \lambda \mathcal{F}(w_{t-1}, S_t)]$$

Recent developments in Artificial Intelligence

- Spatial machine learning
- Quantum computing
- Neuromorphic computing

Spatial machine learning

First Law of Geography (Waldo Tobler): "everything is related to everything else, but near things are more related than distant things."



Spatial machine learning

Spatial contiguity by region



Spatial machine learning

Spatial clustering with machine learning



Spatial machine learning

$$EE = \frac{fn}{tp + fn}$$

$$IE = \frac{fp}{tp + fp}$$

Table 2: Machine learning results with and without spatial analysis, and with and without principal component analysis: exclusion errors and inclusion errors in the test sample. Data from the year 2016 to 2021

Benchmark ML models*	Spatial machine learning models (updated data 2016-2021)					
	Without principal component analysis			With principal component analysis		
	4 clusters	6 clusters	12 clusters	4 clusters	6 clusters	12 clusters
Exclusion error						
Elastic net	29.78	29.55	29.47	29.46	29.56	29.50
Gradient boosting	31.12	28.38	28.33	28.49	30.81	30.65
Linear regression	29.32	29.07	29.00	28.95	29.14	29.06
Logistic classification	35.48	27.85	27.91	29.99	31.75	31.90
Naive Bayes	26.73	24.19	24.41	24.28	32.13	32.17
Neural network	31.77	34.54	34.93	36.65	31.58	29.36
Random forest	60.96	50.18	50.12	50.55	55.03	55.06
Stochastic gradient	40.79	59.49	59.87	55.92	55.14	54.06
Inclusion error						
Elastic net	29.38	32.70	32.67	32.73	32.74	32.72
Gradient boosting	30.72	31.57	31.55	31.72	33.69	33.64
Linear regression	28.92	32.30	32.24	32.26	32.35	32.29
Logistic classification	27.53	46.30	45.61	45.45	39.82	38.91
Naive Bayes	48.44	48.38	48.34	48.85	44.14	43.98
Neural network	28.77	57.16	57.12	57.81	52.49	50.12
Random forest	37.07	31.63	31.23	32.21	35.06	34.80
Stochastic gradient	41.70	51.99	51.46	54.36	48.90	46.51
Minimum values						
Exclusion error (EE)	26.73	24.19	24.41	24.28	29.14	29.06
Inclusion error (IE)	27.53	31.57	31.23	31.72	32.35	32.29

(*) Machine learning without spatial component

Spiking Neural Networks (SNN)

$$\tau_m \frac{dV(t)}{dt} = -V(t) + RI(t)$$

$V(t)$: Membrane potential at time t

τ_m : Membrane time constant

R : Membrane resistance

$I(t)$: Input current (can be sum of post-synaptic currents from other neurons)

When $V(t) \geq V_{\text{th}}$ (a threshold), the neuron emits a spike:

$$S(t) = \delta(t - t_{\text{spike}})$$

And resets:

$$V(t) \leftarrow V_{\text{reset}},$$

Synaptic Dynamics

$$I_{\text{syn}}(t) = \sum_i w_i \alpha(t - t_i)$$

w_i : Synaptic weight

$\alpha(t)$: Synaptic kernel (e.g., exponential decay)

$$\alpha(t) = \frac{1}{\tau_s} e^{-t/\tau_s} \cdot \mathbf{1}_{t \geq 0}$$

Spiking Neural Networks (SNNs): neural networks that aim to more closely mimic the biological behavior of neurons.

SNNs operate with discrete pulses called spikes or impulses.

Third generation of neural network models:

1. **First generation:** perceptron, multilayer perceptron
2. **Second generation:** non-linear activation functions and deep learning
3. **Third generation:** SNNs and neuromorphic computing

Neuromorphic computing

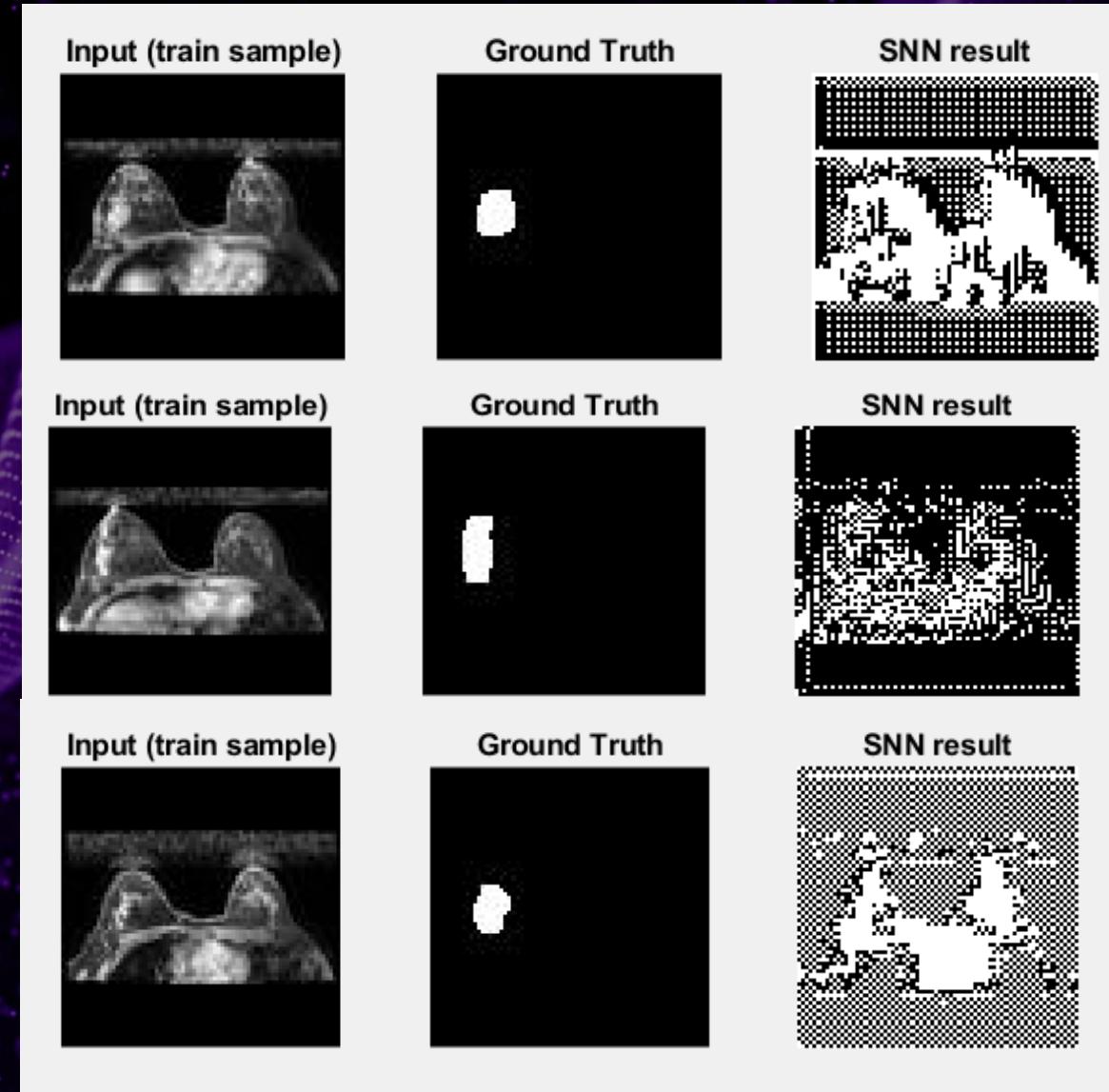
SNNs can be implemented as software, but also as hardware:

- In neuromorphic hardware, computations are carried out by many small units that communicate with each other through bursts of activity (spikes) and adjust their behavior based on the spikes they receive from others.
- Modeled after the systems of the human brain and nervous system. They form a cognitive computer that connects artificial intelligence and machine learning algorithms in an integrated circuit that faithfully replicates the behavior of the human brain.



SNN for classifying breast cancer images

Low accuracy (~53%),
low detection



High accuracy (~79%),
low detection

Low accuracy
(~43%), better
detection

Quantum computing

- **Based on quantum bits (qubits).** Qubits play a similar role in quantum computing as 0/1 bits play in classical computing, but qubits can hold superposition.
- Probabilistic computation instead of deterministic.
- **Entanglement:** Entangled qubits have correlated states that cannot be described independently. If qubit A is entangled with qubit B, measuring A instantaneously affects B, regardless of the distance.
- **Measurement:** When a qubit is measured, the quantum state collapses to one of the basis states:

Challenges:

- Decoherence: Quantum states are fragile and lose information due to environment interactions.
- Error correction: Needs advanced quantum error-correcting codes.
- Scalability: Building many reliable qubits is technically hard.


$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{where}$$
$$\alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1$$

$|0\rangle$ with probability $|\alpha|^2$,
 $|1\rangle$ with probability $|\beta|^2$.