Artificial neural networks

Olga Razuvaeva[1, 2], Sergey Korpachev[3, 4] and Stepan Zakharov[5]

[1]Institute for Theoretical and Experimental Physics
[2]National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
[3]Moscow Institute of Physics and Technology
[4]Lebedev Physical Institute of the Russian Academy of Sciences
[5]Novosibirsk State University

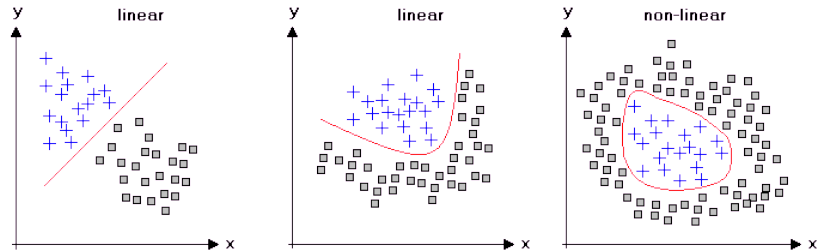
?????Outline?????

- Non-linearity in data
  - Feature extraction
    - NN structure
    - How to train?
    - Deep Learning
- The Achilles heel of the DL models
  - Evolution of GD
  - Reaching stability
- Network architectures

Modelling nonlinearities

Modelling nonlinearities

they exist

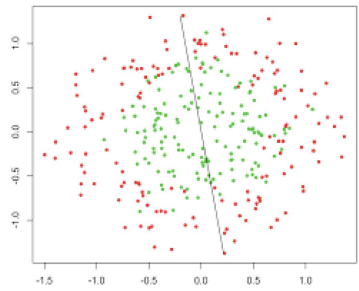


Do you know how to describe the non-linear data in the right picture?

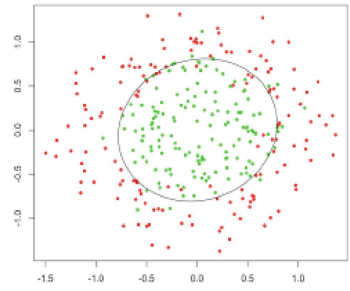
!!!!!Here goes picture of linear vs non-linear data!!!!!

Modelling nonlinearities

linear models



What we have



What we want

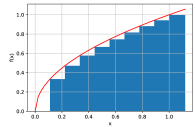
- The linear model does not describe complex nonlinear data
  - Need to use something non-linear

Modelling nonlinearities

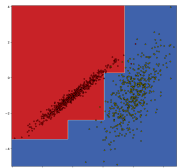
trees

- Trees were designed to approximate nonlinearities
- They do a pretty good job in it
- They are fast and interpretable
- But they are just ”brute-force” algorithms
- They don’t infer symmetries in data by design
- ad-hoc, cut-based and piecewise approximations of data at hand
- Hence, not differentiable and smooth

Regression:

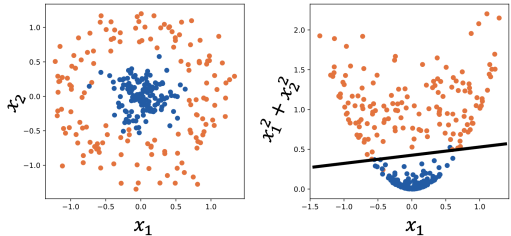


Classification:



Modelling nonlinearities

feature engineering



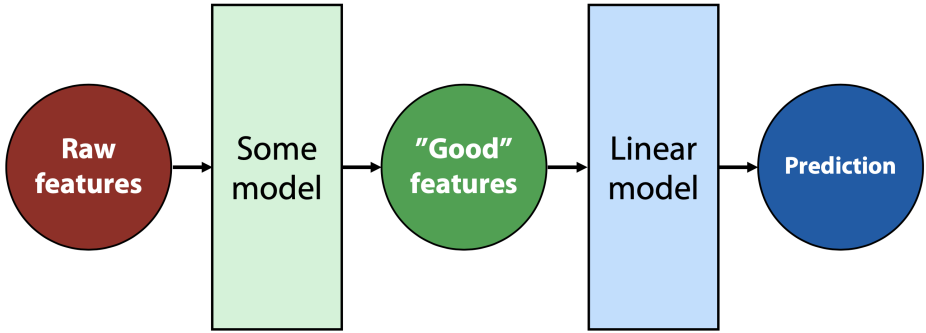
- But sometimes we know *a priori* that there are transformations simplifying the problem for our model
- So that even linear models can do the job
- But this **feature engineering** is non-trivial, requires domain knowledge and is time-consuming

What if we design a model which could automatically feature-engineer itself?

Building the beast

Neural Network

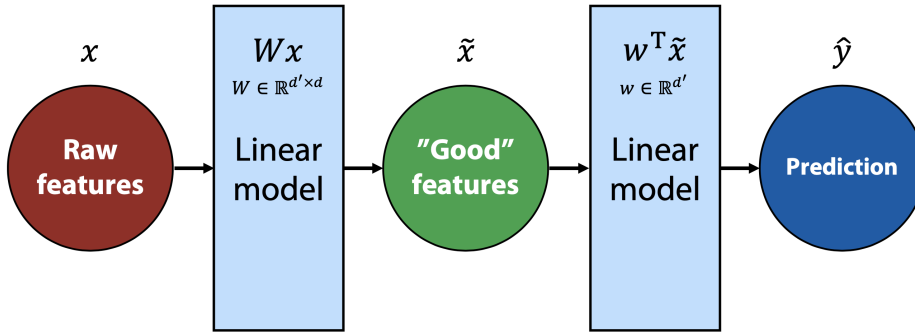
automating FE



- Add a block to a linear model to automatically generate new features
- Two blocks (unknown model and linear model) must work and update parameters as a single model

Neural Network

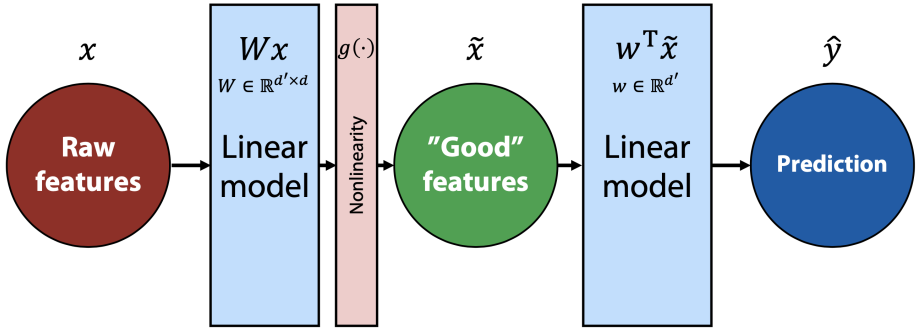
automating FE



- Add a linear model to an unknown block
- $\hat{y} = w^T \cdot \tilde{x} = w^T \cdot (W \cdot x) = (w^T \cdot W) \cdot x = w'^T \cdot x \Rightarrow$ it is still a linear model
- The input feature space has not changed, only the weights have changed

Neural Network

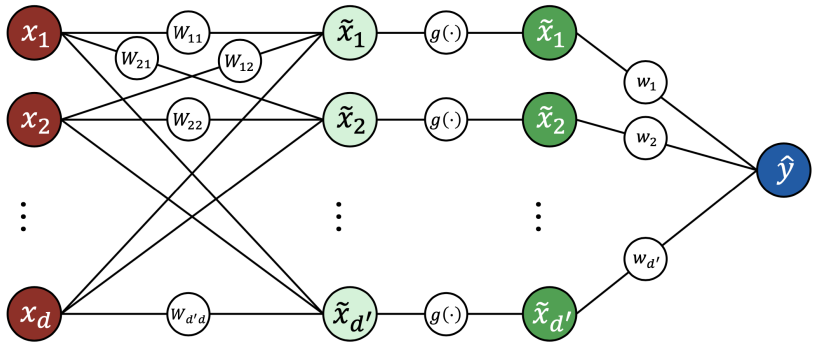
automating FE



- Add nonlinearity to our model
- $\hat{y} = w^T \cdot \tilde{x} = w^T \cdot g(W \cdot x)$,
  where $g(\cdot)$ some nonlinear scalar function (applied elementwise)
- This is the simplest example of a neural network

Neural Network

architecture

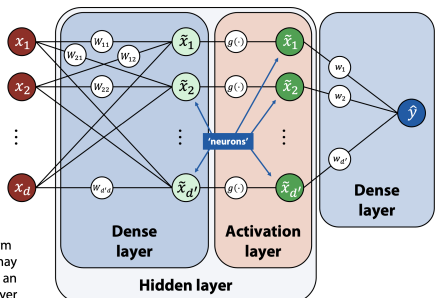


- $\hat{y} = w^T \cdot \tilde{x} = w^T \cdot g(W \cdot x) = \sum_{j=1} \left[w_j \cdot g(\sum_i W_{ji} \cdot x_i) \right]$
- NN is a function that maps a set of X to a set of Y

Neural Network

terminology

Feed-forward network:



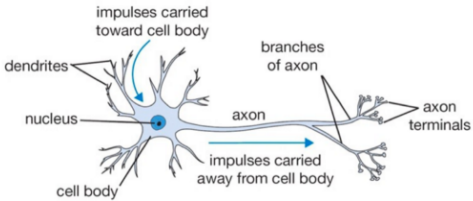
Note: the term "activation" may also stand for an output of a layer

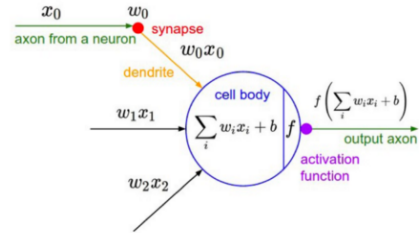
- Red nodes (vertices) $x_1, x_2, ..., x_d$ - features from an input layer of the ANN
- Green nodes (vertices) - neurons from a hidden layer of the ANN
- Blue node (vertex) $\hat{y}$ - neuron from a output layer of the ANN
- Straight lines (edges) correspond to weights (w) between neurons
- $g(\cdot)$ - nonlinear activation function, for example, a sigmoid function $\sigma(\cdot)$
- Each layer (except the output layer) has a bias neuron: $x_{bias} = 1$

Neural Network

human brain

It follows the way of human brain processing. ANN modelled how a neuron works in the human brain.





Neurons on human brain consist of a nucleus, dendrites, cell body, and axon. The number of neurons in humans is approximately 140 billion, consist of 100 billion neurons and 40 billion synapses in neurons.

Training the beast

Training

how to train?

- NN is basically a composite linear model (with nonlinearities)
- We can use gradient descent (GD) to train
- GD algorithm:
  $\theta_{i+1} = \theta_i - \eta \cdot \nabla Q(\theta_i)$
- Do you know how to calculate the gradient of the weights?

Training

chain rule

- But gradients are hard to derive analytically
- Writing down (into NN framework) all the derivatives is tough
- The approach doesn't generalize to architectures
- But NN is just a composite model $\Rightarrow$ can use chain rule for differentiating it

Training

chain rule: examples

- We know how to find the derivative of a simple function.
- Let's remember the rule for differentiating complex functions.

$$\frac{\partial f\big(t(x)\big)}{\partial x} = \frac{\partial f(t)}{\partial t(x)} \cdot \frac{\partial t(x)}{\partial x}$$

- What is the derivative of following function with respect to x: $sin(x^2 + 5)$?
- What is the derivative of the sigmoid function?

  Can you convert it to $\sigma'(x) = f\big(\sigma(x)\big)$?

- Sigmoid function: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

Training

chain rule: examples

- We know how to find the derivative of a simple function.
- Let's remember the rule for differentiating complex functions.

$$\frac{\partial f\big(t(x)\big)}{\partial x} = \frac{\partial f(t)}{\partial t(x)} \cdot \frac{\partial t(x)}{\partial x}$$

- What is the derivative of following function with respect to x: $sin(x^2 + 5)$?
- What is the derivative of the sigmoid function?

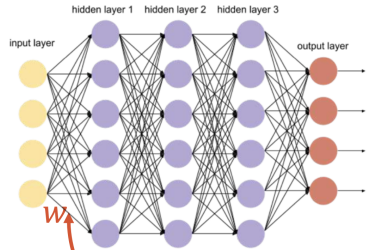
  Can you convert it to $\sigma'(x) = f\big(\sigma(x)\big)$?

- Sigmoid function: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

Answer: $\sigma'(x) = \sigma(x) \cdot \big(1 - \sigma(x)\big)$

Training

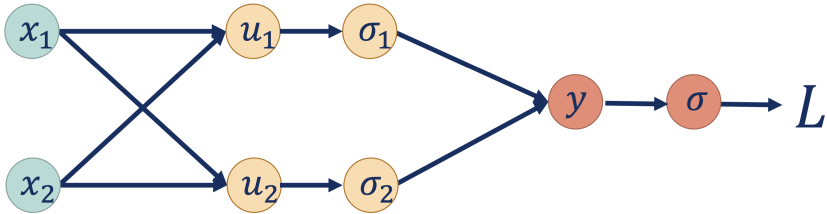
weights update



- We know how to update a weight
- How to find the partial derivative of the loss function with respect to some weight?

Training

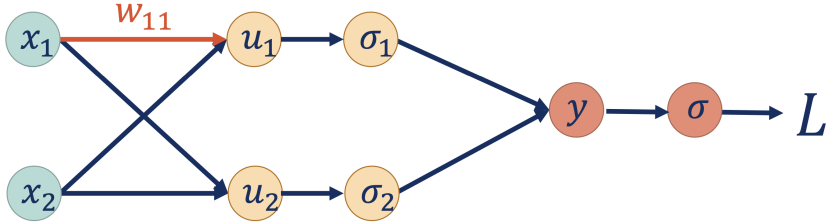
weights update



- Consider a simpler neural network

Training

weights update

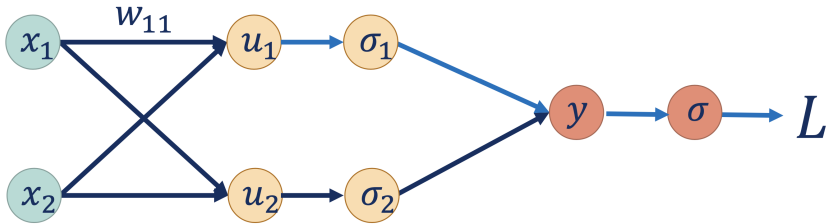


$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

- We know how to update a weight

Training

back propagation



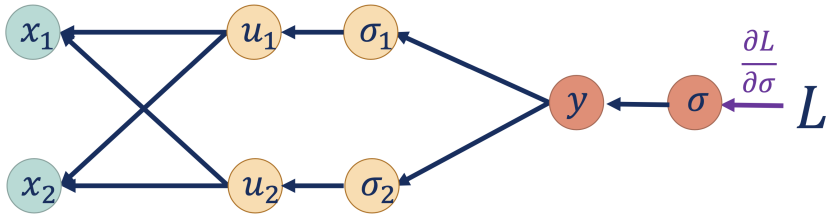
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1}\frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1$$

- We know how to find the partial derivative of the loss function with respect to some weight

Training

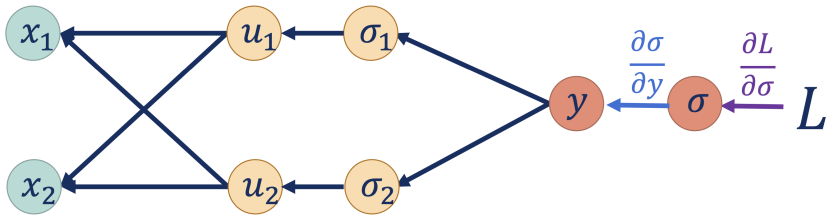
back propagation



- Let's try to apply our knowledge to this simple neural network

Training

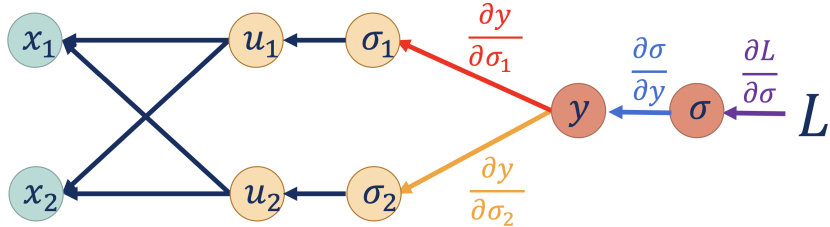
back propagation



- Going back over the network to the beginning

Training

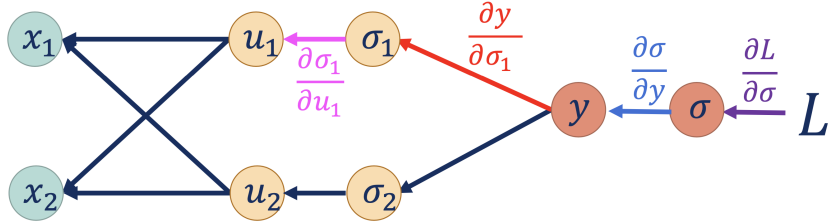
back propagation



- Let's go further to the beginning

Training

back propagation

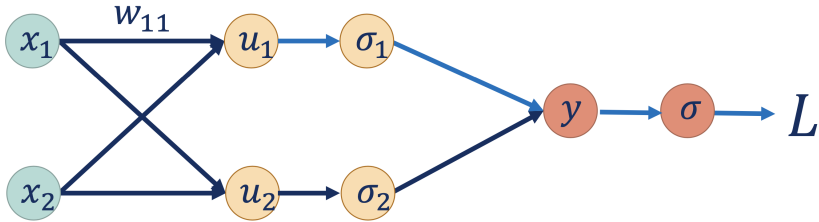


$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial y}\frac{\partial y}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial u_1}$$

- Now we know the value of the partial derivative of the loss function with respect to some weight

Training

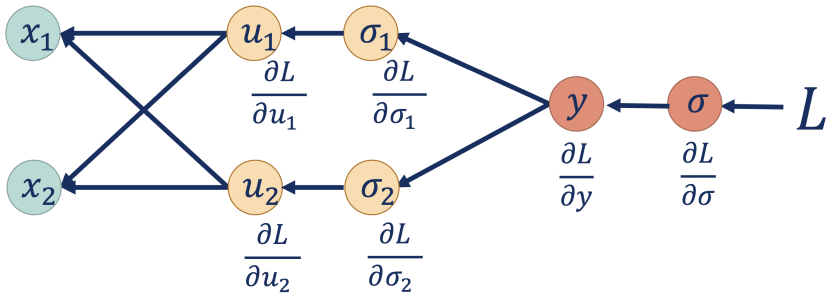
forward pass



- We use forward pass to get the output of each neuron and compute the loss function to then back propagate the error

Training

backward pass



- We use backward pass to compute the gradients in the neural network and get updated weight for each neuron

Training

NN training procedure

- Firstly, we make a forward pass through the neural network to calculate the output of each neuron and find the value of the loss function.
- Secondly, we go back through the neural network to calculate the value of the gradients to update the weights later.
- Thirdly, we repeat the given procedure at a new iteration with updated weights.
- One iteration (epoch) is a forward and backward pass of the neural network.
- It is necessary to make as many epochs (with pre-configured neural network hyperparameters) until the model produces the desired result.

Break time

Break time

check your understanding

- Is it necessary to use a bias neuron? Why?
- What values can the weights of neurons take?
- Should the input features be on the same scale (normalization of the input data) or not? Why?