Artificial Neural Networks

Artificial Intelligence for Digital Health (AID) M.Sc. in Digital Health – University of Pisa Davide Bacciu (davide.bacciu@unipi.it)



Lecture Outline

- Modeling the artificial neuron
- Artificial neural networks and the multilayer perceptron
 - Layered structure
 - Activation functions
 - Outputs and losses
- Training Artificial neural networks
 - Backpropagation algorithm
 - Loss optimization
 - Some basic tricks



Why Using Neural Networks?

- They are universal function approximators
- Non-parametric
- Scalable
- Heavily supported at SW and HW
- Fast solutions by Lego bricking

Neural Networks in Healthcare

Learn complex patterns from noisy and highly diverse data Handle vast amounts of data in a scalable way, possibly in real time

Provide highly accurate predictions

When to be careful in using them?

- Stringent interpretability and safety/security requirements
- Need to incorporate background/prior knowledge available
- Little-data (as opposed to Big-Data)



Essentially in our biomedical setting!

Modeling the artificial neuron

The Biological Neuron



• Dendrites: Receive electrical signals from other neurons

- Nucleus: Processes the electrical signal and generates an output signal
- Axons: Transmit the output signal to other neurons

The Neuron Metaphor



Looks like something we have already seen!

- Neurons receive information from multiple inputs and transmit information to other neurons
- Scale/enhance inputs
- Apply some activation function to received input information

The Return of the Logistic Regression



The Artificial Neuron





McCulloch & Pitts, 1943

- Input *x*
- Synaptic weights ${m heta}$
- Local potential $net(\mathbf{x}) = \sum_{i=1}^{k} \theta_i x_i$
- Activation function φ
- Output y

The Perceptron





Rosenblatt, 1953

The first learnable neuron (the deltarule)

The Perceptron -> The Issue

Still too similar to a logistic regression

$$y = \varphi(net(\mathbf{x})) = \sigma\left(\sum_{i=1}^{k} \theta_i x_i\right)$$

With its limitations



Artificial Neural Networks

Artificial neural networks – Interconnected neurons



Hidden layer

Multi-Layer Perceptron (MLP)

Three main components (layers):

- Input layer: feeds the input x
- Hidden layer: transforms the input x into a new (vectorial) representation h(x) (size of the number of hidden neurons) depending on the hidden layer parameters θ^h
- Output layer: generates the prediction y (output) by combining the intermediate representation of the input provided by the hidden layer and the output layer parameters θ^y

Two key aspects

- Can learn any decision boundary (even non-linear ones)
- Adaptable to any type of task by changing the output layer

MLP – Input Layer

- The input layer is simply a replica of the input data
- However, preprocessing is crucial for good performance
- Feature scaling (normalization or standardization)
 - If features have different scales, normalization is needed to ensure stable training.
- Handling categorical features
 - Convert categorical variables into numerical form using one-hot encoding
 - For large categories, embedding layers can be used instead

Categorical Features

- A categorical variable is a variable that can belong to one of k discrete categories
- Categorical variables are usually encoded using 1-out-of-k coding (one hot)
 - E.g. for three colors: red = (1 0 0), green =(0 1 0), Blue =(0 0 1)
 - I.e. the MLP above will have 3 input
- If we used red = 1, green = 2, blue = 3, then this type of encoding imposes a representational bias which is not semantically supported
- More recently dense embeddings have taken over (especially for natural language)

Numerical Features

- A numerical variable (continuous, ordinal) can be directly fed to a neural network
- However, it is good practice to normalize data so that the dynamic range of inputs is limited
 - Min-Max Scaling Scales values to [0,1] or [-1,1] $x' = \frac{x - x_{min}}{x_{max-x_{min}}} \rightarrow x' \in [0,1] \text{ or } x' = 2 \frac{x - x_{min}}{x_{max-x_{min}}} - 1 \rightarrow x' \in [-1,1]$
 - Z-score Standardization- Centers data around mean 0 and std 1

$$x' = \frac{x-\mu}{\sigma}$$

- Need also care in choosing w.r.t. whom you are normalizing population or individual (especially in biomedical applications)
- Normalization is applied also in timeseries and graphs

MLP – Hidden Layer

The hidden layer transforms the input \boldsymbol{x} into another vector \boldsymbol{h} of arbitrary size

- Size of *h*
 - h can be smaller or larger than x, depending on the complexity of the task
 - More neurons → More capacity to learn complex patterns (but risk of overfitting)
 - Fewer neurons → Simpler models, better generalization
- Helps solve the task by learning representations that make classification, regression, or other tasks easier
- The transformation from x to h is governed by trainable parameters θ^h fitted during training

 $\boldsymbol{h} = g_{\theta^h}(\boldsymbol{x})$ with $g_{\theta^h}(\cdot)$ nonlinear

Hidden layer step-by-step (I)

- Input: A vector \boldsymbol{x} with k = 2 features $\rightarrow \boldsymbol{x} \in \mathbb{R}^2$
- Output: A vector \boldsymbol{h} with s = 3 features $\rightarrow \boldsymbol{h} \in \mathbb{R}^3$

 $h_0 = \varphi(\theta_0 x) = \varphi(\theta_{00} x_0 + \theta_{10} x_1)$ $h_1 = \varphi(\theta_1 x) = \varphi(\theta_{01} x_0 + \theta_{11} x_1)$ $h_2 = \varphi(\theta_2 x) = \varphi(\theta_{02} x_0 + \theta_{12} x_1)$

We transformed a 2D input vector into a 3D hidden representation by a linear combination of x features + an activation function φ



Hidden layer step-by-step (II)

• θ is a parameter matrix with k=2 rows and s=3 columns

$$\boldsymbol{\theta} = \begin{bmatrix} 2 & 3 & -1 \\ 2 & 6 & -8 \end{bmatrix}^T$$

• $x = \begin{bmatrix} -1 & 1 \end{bmatrix}^T$ is a sample input vector

$$h_{0} = \varphi \left(\begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \varphi(0)$$

$$h_{1} = \varphi \left(\begin{bmatrix} 6 & 3 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \varphi(-3)$$

$$h_{2} = \varphi \left(\begin{bmatrix} -8 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \varphi(7)$$

$$h = \varphi(\begin{bmatrix} 0 - 3 7 \end{bmatrix})$$

Linear activation function



It returns the input without changing it $h = \varphi([0 - 37]) = [0 - 37]$

Sigmoid activation function



Squashes the input in $[0,1] \rightarrow \varphi([0 - 37]) = [0.5 \ 0.047 \ 0.999]$

Hyperbolic tangent activation function



Rectified Linear Unit (ReLU)

 $\varphi(\boldsymbol{\theta} \boldsymbol{x}) = \max(0, \boldsymbol{\theta} \boldsymbol{x})$



Zeroes the negative components of the input vector, leaving the rest unchanged $\rightarrow \varphi([0 - 37]) = [007]$

MLP – Output Layer

The output layer transforms the hidden layer output \boldsymbol{h} into the prediction \boldsymbol{y}

- Provides the final prediction of the MLP
- The transformation from \boldsymbol{h} to \boldsymbol{y} is parameterized by $\boldsymbol{\theta}^{\boldsymbol{y}}$ which are learned during training
- It implements a parameterized function $y = g_{\theta^y}(h)$
- Computes similarly to the hidden layer, but it is associated to activation functions which are task-specific and are linked to a loss function

Output layer for binary classification

- Goal: classify a sample x into either class $y \in \{0,1\}$
- Linear combination of the inputs
- Followed by sigmoid activation $\hat{y} = \sigma(\theta^y h)$



Binary cross entropy loss $BCE(\hat{y}, y) = -y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$

Output layer for multiple binary classification

Goal: predict a d-dimensional vector \boldsymbol{y} of 0 and 1 for a sample \boldsymbol{x}

- Output vector $\mathbf{y} \in \{0,1\}^d$
- Repeated application of the sigmoid for the d-dimensions $\widehat{y} = \sigma(\theta^y h)$

Average binary cross entropy loss

$$BCE_{multi}(\widehat{y}, y) = \frac{1}{d} \sum_{d} BCE(\widehat{y_d}, y_d)$$



Output layer for regression

Goal: predict a real value y in response to sample \boldsymbol{x}

- Just a linear combination of the inputs
- Followed by linear (identity) activation $\hat{y} = (\theta^y h)$

Mean squared error loss $MSE(\hat{y}, y) = (y - \hat{y})^2$



Output layer for multiple regression

Goal: predict a real valued vector $y \in \mathbb{R}^d$ in response to sample x

- Multiple linear combination of \boldsymbol{h} , one for each component of \boldsymbol{y}
- Followed by multiple application of identity

Mean squared error loss

$$MSE_{multi}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{d} \sum_{d} MSE(\widehat{y_{d}}, y_{d})$$



Output layer for multiclass classification

Goal: assign a sample x to one of d classes $\{c_1, \dots, c_d\}$

- Output $y \in \{0,1\}^d$ as a one-hot encoding of the class
- Linear combination of \boldsymbol{h} followed by identify
- Output generated by a softmax function

 $\hat{y} = softmax(\boldsymbol{\theta}^{y}\boldsymbol{h})$

Cross-Entropy loss

 $CE(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{d} y_{d} \log(\widehat{y_{d}})$



Softmax function

- Transforms a (dense) vector in a categorical (discrete) probability distribution
- Given a vector **x** with *n* components

softmax_i(**z**) =
$$\frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

In other words

- Take the exponentiation of each component of $oldsymbol{z}$
- Compute the sum of exponentiated components
- Divide each exponentiated component by the overall sum
- The resulting softmax vector will sum-to-1 (like a probability distribution)

 $softmax([-1\ 0\ 1]) = [0.09\ 0.244\ 0.666]$

MLP: putting things back together



- Transform x into h through the hidden layer $h = g_{\theta^h}(x) = \varphi(\theta^h x)$
- Transform **h** into y through the output layer $y = g_{\theta^y}(h) = \varphi(\theta^y h)$
- The model is $MLP_{\theta}(x) = g_{\theta^{y}}(g_{\theta^{h}}(x))$ with $\theta = \{\theta^{h}, \theta^{y}\}$ being learned parameters

What do we need to build an MLP?

- Input preprocessing
- Size of the hidden layer s
- Hidden layer activation functions arphi
- Configuration of the output layer (guided by the task)

Model selection choices

Training an Artificial Neural Network

Let's recall logistic regression training

For a certain number of iterations (epochs):

For each training pair $(x^i, y^i) \in D_{train}$:

1.Compute the prediction $h_{\theta}(\mathbf{x}^i) = \sigma(\mathbf{x}^i \boldsymbol{\theta})$

2.Compute the loss L of the prediction $h_{ heta}(x^i)$ compared to the true label y^i

3.Compute the gradient of the loss $\nabla_{\theta} L = x(y^i - h_{\theta}(x^i))$

4. Update the parameters $\boldsymbol{\theta}_{new} = \boldsymbol{\theta} - \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \boldsymbol{L}$

5.Use the updated parameters in the next iteration $\boldsymbol{\theta} = \boldsymbol{\theta}_{new}$

MLP training

For a certain number of iterations (epochs): For each training pair $(x^i, y^i) \in D_{train}$: 1.Compute the prediction $MLP_{\theta}(x^i)$ 2.Compute the loss L of $MLP_{\theta}(x^i)$ compared to the true label y^i 3.Compute the gradient of the loss $\nabla_{\theta} L$ 4.Update the parameters $\theta_{new} = \theta - \eta \nabla_{\theta} L$ 5.Use the updated parameters in the next iteration $\theta = \theta_{new}$

Actually, it is a bit more articulated than that

MLP training, more realistically

For a certain number of iterations (epochs):

- Shuffle D_{train}
- For each subsample *B* of pairs (x^B, y^B) extracted from D_{train} :
 - 1.Compute the prediction $MLP_{\theta}(x^B)$
 - 2.Compute the loss L of $MLP_{\theta}(x^{B})$ compared to the true labels y^{B}
 - 3.Compute the gradient of the loss $\nabla_{\theta} L$
 - 4. Update the parameters $\boldsymbol{\theta}_{new} = \boldsymbol{\theta} \eta \boldsymbol{\nabla}_{\boldsymbol{\theta}} \boldsymbol{L}$
 - 5.Use the updated parameters in the next iteration $\boldsymbol{\theta} = \boldsymbol{\theta}_{new}$

The gradient of the loss needs to be computed w.r.t. all parameters, including those from the hidden layer!

Minibatching

Mini-batching is a technique used to split the dataset into smaller subsets, each containing B examples. Instead of updating the parameters after processing all examples or after each single example, updates are done after processing each mini-batch

Divide the dataset into multiple mini-batches of size B (batch size, hyperparameter)
 Shuffle the dataset at the beginning of each epoch to avoid bias in training.
 Update model parameters once per mini-batch.

- Why is Mini-Batching Useful? It is a compromise between:
 - Stochastic Gradient Descent (SGD) (B = 1): Updates after each example → fast but noisy updates.
 - Batch Gradient Descent (B = n): Updates after the entire dataset → stable but slow updates.
- Mini-batching provides a balance
 - Faster convergence than full batch updates
 - More stable updates than SGD
 - Efficient computation by leveraging GPU parallelism

Gradient of the loss

We have different loss functions based on the output layer, and we can compute their gradients as follows

Loss Function	Formula	Gradient
Binary Cross-Entropy (BCE)	$-[y\log(\hat{y})+(1-y)\log(1-\hat{y})]$	$x(\hat{y}-y)$
Mean Squared Error (MSE)	$(\hat{y}-y)^2$	$x(\hat{y}-y)$
Categorical Cross-Entropy (CE)	$-\sum_{k=1}^{K}y_k\log(\hat{y}_k)$	$\hat{y}-y$

We can update the output layer parameters (θ^{γ}) using these gradients

However, we don't yet know how to compute the gradient for the hidden layer parameters (θ^h)

Backpropagation at the rescue

Key Idea: We apply the Chain Rule to propagate the gradient from the output layer to the hidden layer

- 1. Compute the gradient for the output layer
- Example: For BCE, the gradient is

$$\nabla_{\theta^{\mathcal{Y}}}L = x(\hat{y} - y)$$

2. Propagate the gradient back to the hidden layer

- Using the Chain Rule, the gradient for the hidden layer parameters θ^h is: $\nabla_{\theta^h} L = \nabla_h L \cdot \nabla_{\theta^h} h$
- $abla_h L$ is the gradient of the loss with respect to the hidden layer output $m{h}$
- $\nabla_{\theta^h} h$ is the gradient of **h** with respect to θ^h
- 3. Compute the gradient of h
- If the hidden layer transformation is:
- Then, using the chain rule:

$$\nabla_{\theta^h} L = \nabla_h L \cdot \left(\varphi'(\theta^h x) \right) \cdot x$$

 $\boldsymbol{h} = \varphi(\boldsymbol{\theta}^h \boldsymbol{x})$

Luckily for you backpropagation is automatically handled by the NN libraries

• Where φ' is the derivative of the activation function

Convergence Criteria

- Learning is obtained by iteratively supplying shuffled training data and adjusting by backpropagation
 - Typically, 1 training set presentation = 1 epoch
- We need a stopping criteria to define convergence
 - Validation for generalization performance: stop when generalization performance reaches a peak

Early Stopping

- Keep a hold-out validation set and assess accuracy after (every/some) epoch.
- Maintain weights for best performing network on the validation set and stop training when error increases beyond this
- Always let the network run for some epochs before deciding to stop (patience parameter), then backtrack to best result



Wrap-up

Take home lessons

- Artificial Neural Networks are universal function approximators that can learn complex patterns from noisy and diverse data
 - In healthcare they allow handling vast amounts of heterogenous data in a scalable way, providing highly accurate predictions
- The artificial neuron is loosely inspired by biological neurons
 - Artificial neurons receive and aggregate inputs, apply activation functions, and transmit outputs
- The MLP has a layered structure consisting of input, hidden, and output layers
 - Capable of learning any decision boundary and adaptable to various tasks
 - Several architectural/model selection choices need to be taken even for simple networks
- Training ANNs involves a specialized gradient descent algorithm (backpropagation)
 - Optimization aspects needs to be carefully curated
 - Stopping conditions are relevant as it is an iterative process
- Optimize neural networks from the start
 - Applying the right weight initialization strategies significantly impacts effectiveness
 - Needs to be tailored to the activation function

Next Lecture

- Laboratory tutorial
- Next week lecture
 - Risk scoring and stratification for population health management
 - Machine learning for risk stratification
 - Identifying risk factors
 - Assessment and validation of risk predictors
 - Censoring and its impact in risk scoring