Introduction to Neural Networks and Deep Learning

Biljana Jonoska Stojkova, PhD Adapted by: Nikolas Krstic

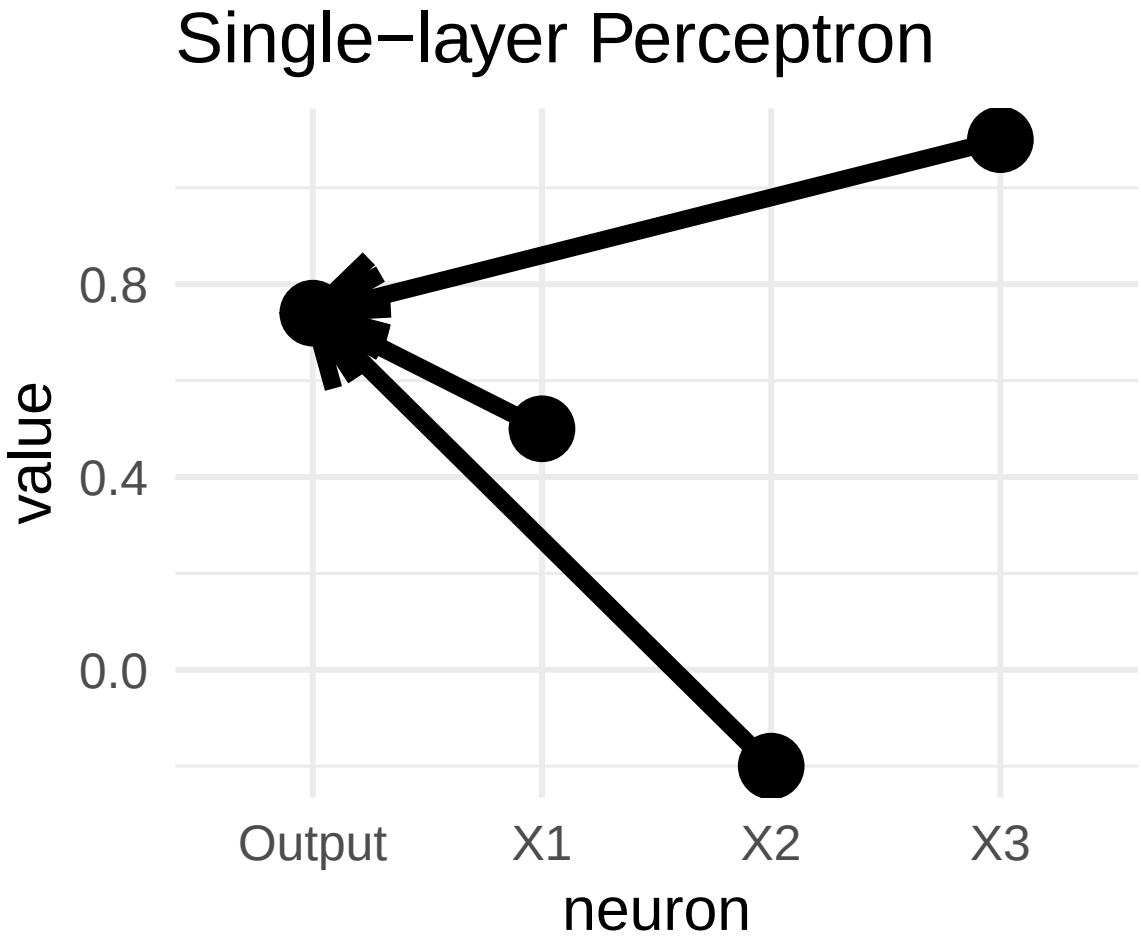
Department of Statisitcs, University of British Columbia

Aug 2, 2024

Section 1

The perceptron

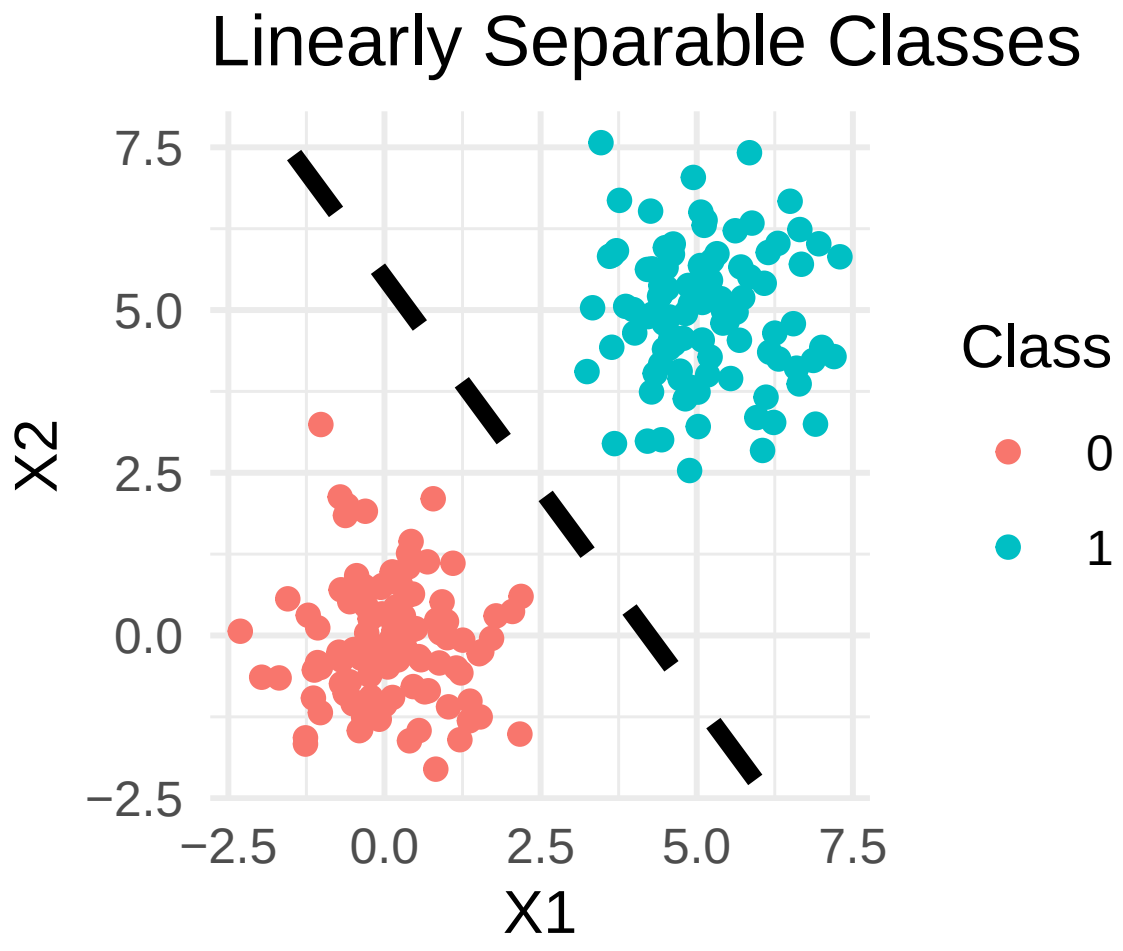
The earliest neural network model (Rosenblatt (1958))



The perceptron (Rosenblatt (1958)) - introduction

- The plot provides a simplified visual representation of how a perceptron takes inputs and produces an output based on the weights of its connections;
- Input layer represented by three circles labeled “X1,” “X2,” and “X3,” each with a value indicated by its position along the vertical axis;
- The output layer is represented by a single circle labeled “Output” with a value of 0.74 indicated by its position along the vertical axis;
- The arrow points from the input neuron to the output neuron, indicating that the output neuron receives input from the input neuron. The length of the arrow indicates the strength (weight) of the connection, with longer arrows indicating stronger connections.

The perceptron - how does it work?



The perceptron (Rosenblatt (1958)) - introduction

- The perceptron is a basic building block of a neural network for for linearly separable binary classification tasks;
- Modeled by the mid 19th century understanding of how the brain works The perceptron is like a simple decision-making machine that can make binary (yes or no) decisions based on a set of inputs;
  - Example: a light bulb that turns on if the temperature is above a certain threshold. The perceptron takes in inputs (like temperature) and decides whether to turn on or off (yes or no) based on a set of rules;
- The initial perceptron model was developed with numerical inputs, and the decision to turn on or off is based on whether the sum of the inputs times their respective weights is above a certain threshold using a step (threshold based) function as activation function.

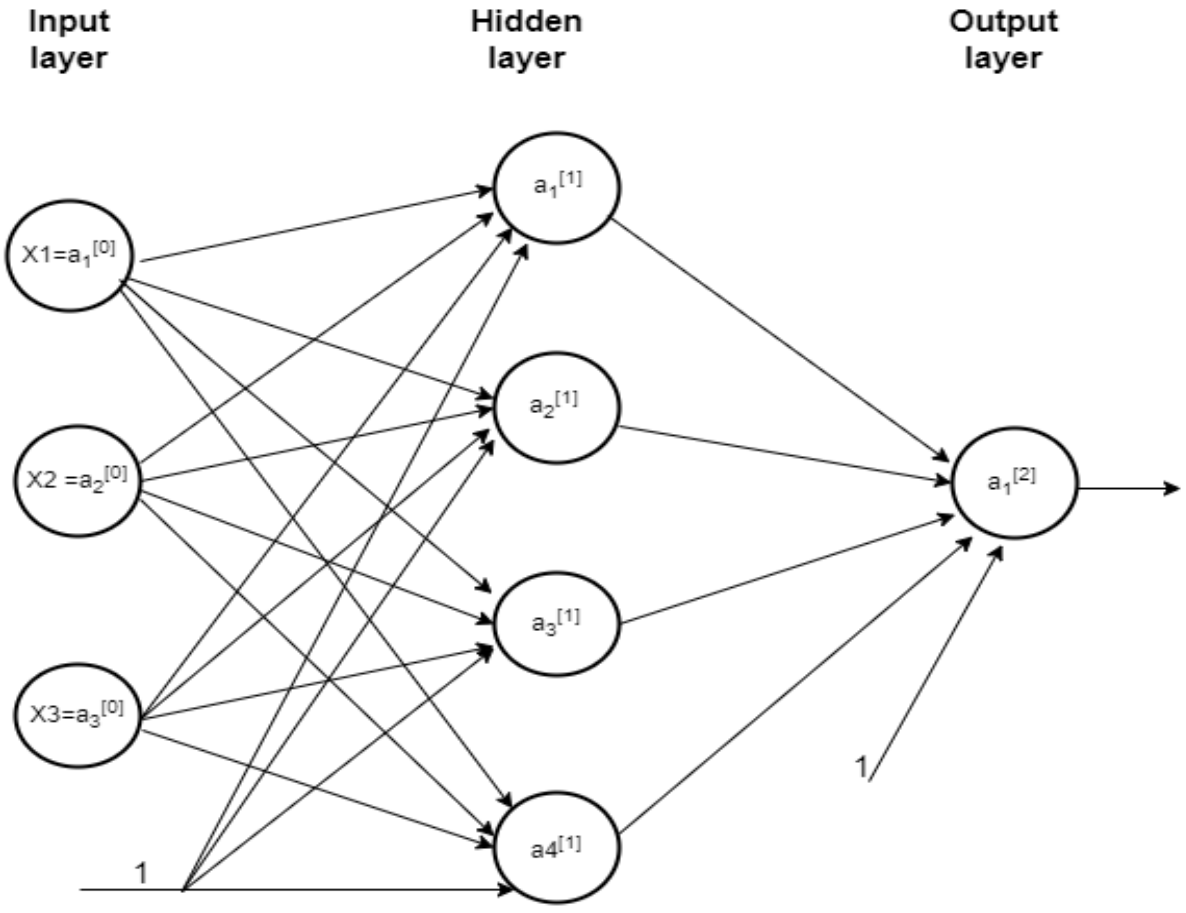
Section 2

NN

Neural Networks (NN)

- Machine learning algorithms that are inspired by the structure and function of the human brain;
- They consist of interconnected artificial neurons that process and transmit information;
- Used to learn patterns and relationships from data for a wide range of tasks;
  - image and speech recognition,
  - natural language processing,
  - prediction problems;
- Can be used for both supervised and unsupervised learning:
  - In supervised learning, the network is trained on labeled data, where the desired output is known for each input. The network learns to map the input to the output;
  - In unsupervised learning, the network is trained on unlabeled data, where the goal is to learn the underlying structure or patterns in the data (e.g., tasks such as clustering, dimensionality reduction, and anomaly detection).

Neural Network with 2 layers



Parameters:

$\mathbf{W}_{4x3}^{[1]}$ $\mathbf{b}_{4x1}^{[1]}$ $\mathbf{W}_{1x4}^{[2]}$ $\mathbf{b}_{1x1}^{[2]}$

Neural networks ingredients

- Input layer;
- Output layer;
- Hidden layers;
- Neurons (interconnected nodes or perceptrons);
- Connections between neurons (weights and biases);
- Activation functions;
- Loss function;
- Optimization algorithm.

The Input Layer

- Is the first layer of a neural network and plays a crucial role in:
  - receiving input data;
  - processing input data;
  - passing it to the subsequent layers for further processing and feature extraction;
- The input layer consists of individual neurons, where each neuron corresponds to a specific feature or attribute of the input data (e.g., images, text, audio, or numerical data);
- The number of neurons in the input layer is determined by the dimensionality of the input data (image data, each neuron in the input layer represents a pixel or a specific feature of the image);
- Preprocessing steps may include scaling, normalization, one-hot encoding, or any other necessary transformations specific to the data and the problem domain.

The Output Layer

- The output layer is the final layer of a neural network that produces the predictions or outputs based on the processed input data;
- The design and structure of the output layer depend on the nature of the problem being solved, such as classification, regression, or generative tasks;
- The primary function of the output layer is to produce the desired output or predictions that are relevant to the specific problem domain;
- The number of neurons in the output layer corresponds to the number of output classes or the dimensionality of the desired output.

The Output Layer - Types of Output Layers

- Classification: e.g., in classification tasks with mutually exclusive classes, the output layer consists of one neuron for each class;
- Regression: e.g., for regression tasks, the output layer generally consists of a single neuron that outputs a continuous value, representing the predicted numerical value;
- Generative Tasks: e.g., in generative tasks like image generation, the output layer, the neurons correspond to higher-level features or attributes of the image.

The hidden layer

- Transforms the input data into a more abstract and high-level representation;
- Feeds the transformed data into the Output Layer;

The neurons

- Neuron, node or a perceptron, is a fundamental computational unit that processes and transforms input data;
- Neurons are organized in layers within the network, including the input layer, hidden layer(s), and output layer;
- Each neuron receives input from the previous layer, which is typically a weighted sum of the outputs from the neurons in the previous layer.

Connections between the layers (weights and biases)

- Represent links between neurons from one layer to neuron from another layer, that allow transition of information to process data and make predictions;
- The links are represented by weights and biases;
- These are the unknown parameters of the Neural Networks.

Activation functions

- Is a mathematical function applied to the output of a neuron;
- Introduce non-linearity to model complex relationships between input and output;
- The activation function determines whether the neuron will be activated or "fired" based on the weighted sum of its inputs;
- Different layers in the network can use different activation functions.

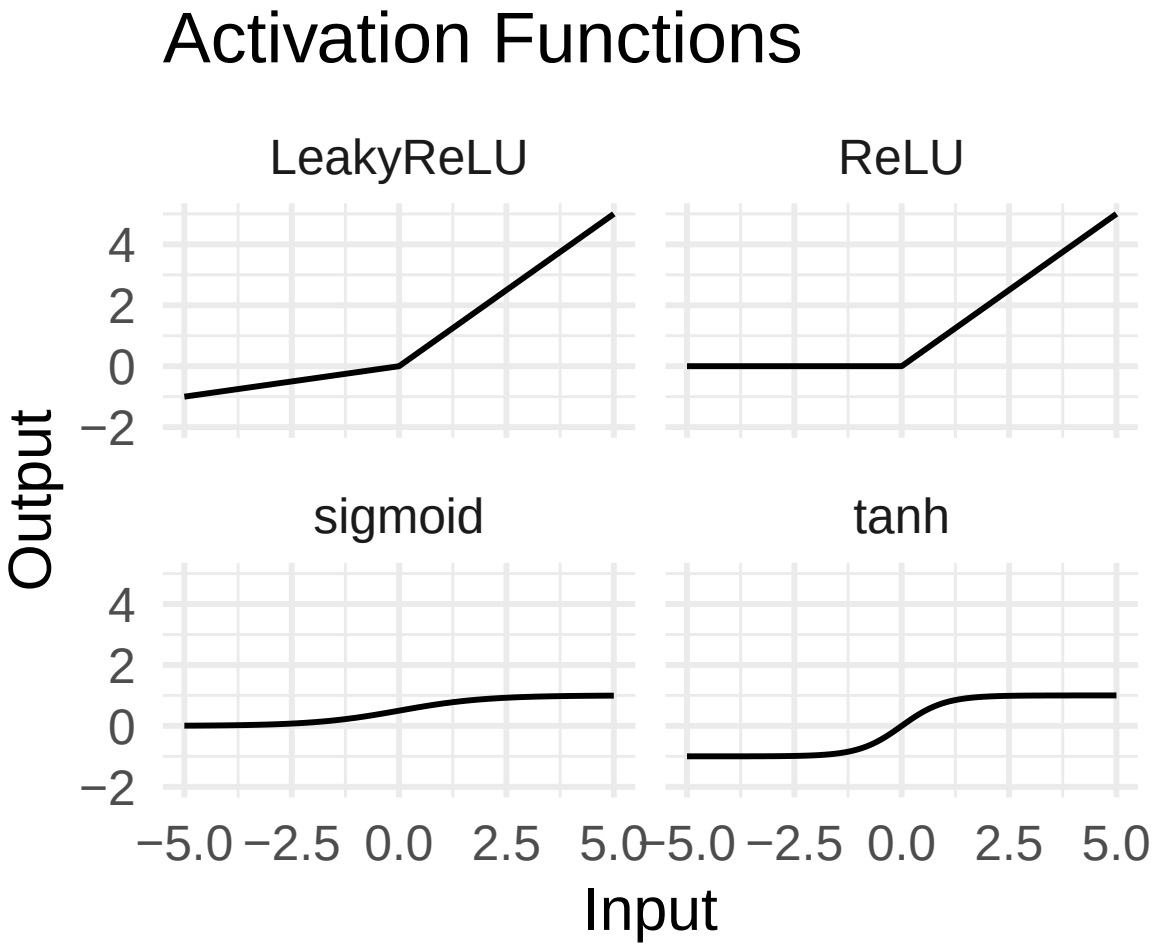
Activation functions - cont.

- Sigmoid function (Logistic function): The sigmoid function maps the input to a value between 0 and 1. It is defined as $f(x) = 1/(1 + exp(-x))$ (e.g., output layer binary classification problems);
- Softmax function: used in the output layer for multi-class classification problems. It converts the network's outputs into a probability distribution over multiple classes, ensuring that the probabilities sum up to 1.
- Hyperbolic tangent (Tanh) function: The tanh function maps the input to a value between -1 and 1. It is defined as $f(x) = (exp(x) - exp(-x))/(exp(x) + exp(-x))$ (classification tasks and can be used in hidden layers);

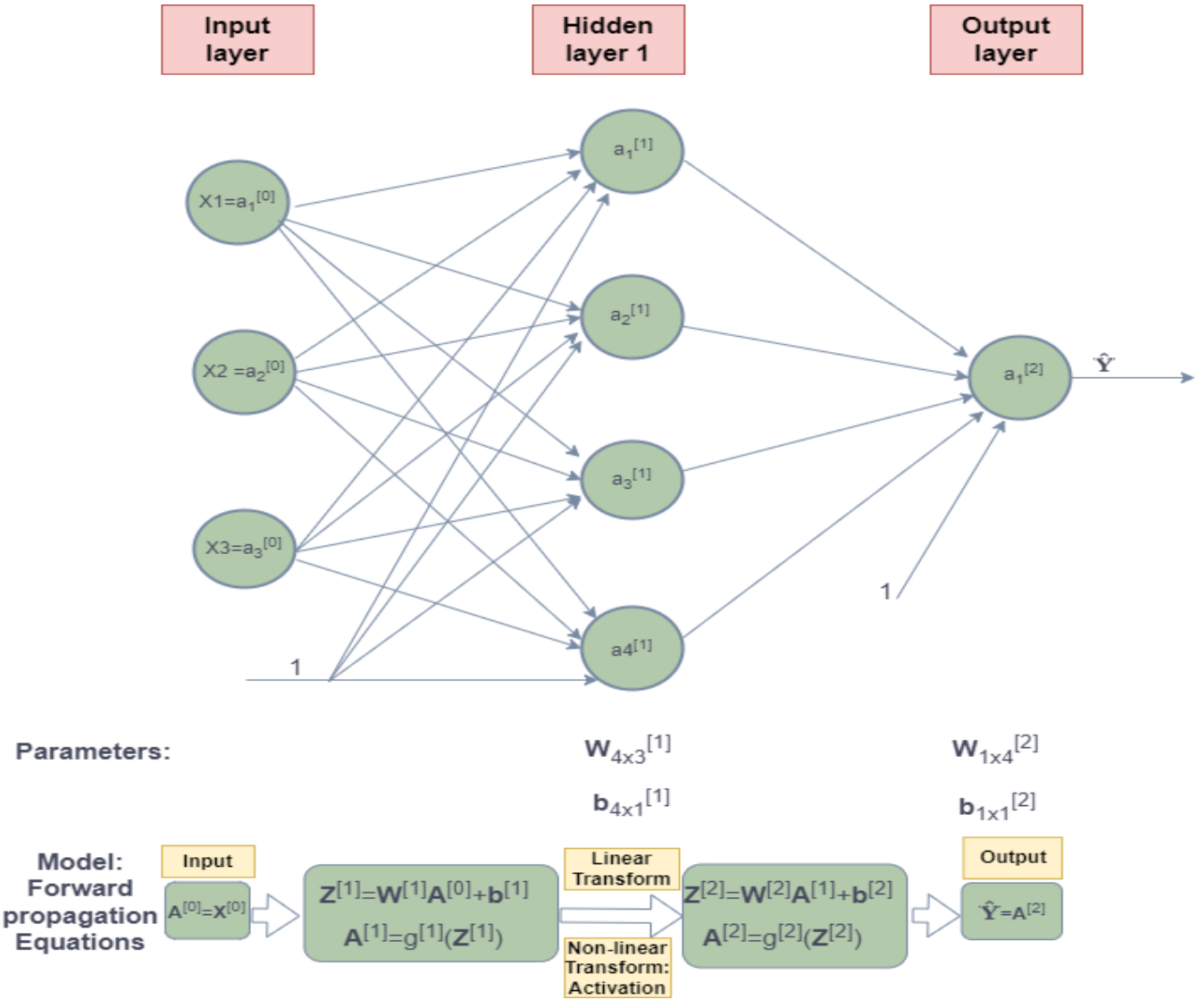
Activation functions - cont.

- Rectified Linear Unit (ReLU): The ReLU function is defined as $f(x) = max(0, x)$. It returns the input value if it is positive, and zero otherwise (widely used as an activation function in hidden layers, as it introduces sparsity and allows the network to learn complex representations more efficiently);
- Leaky ReLU: The Leaky ReLU function is similar to ReLU but allows small negative values for negative inputs. It is defined as $f(x) = max(\alpha x, x)$, where α is a small constant ($\$ < 1\$$). Leaky ReLU helps mitigate the "dying ReLU" problem where some neurons in the network become non-responsive.

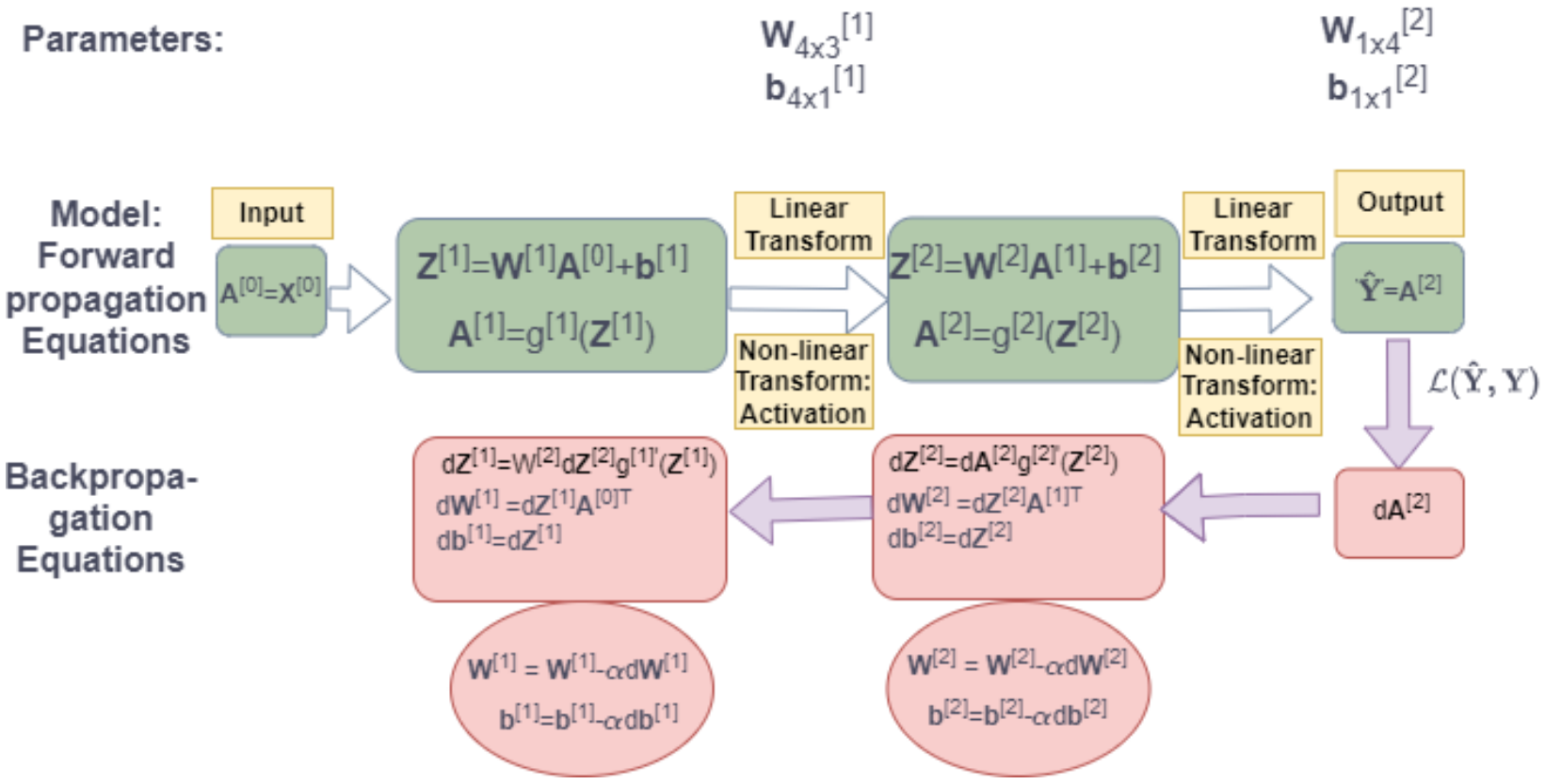
Activation functions - cont.



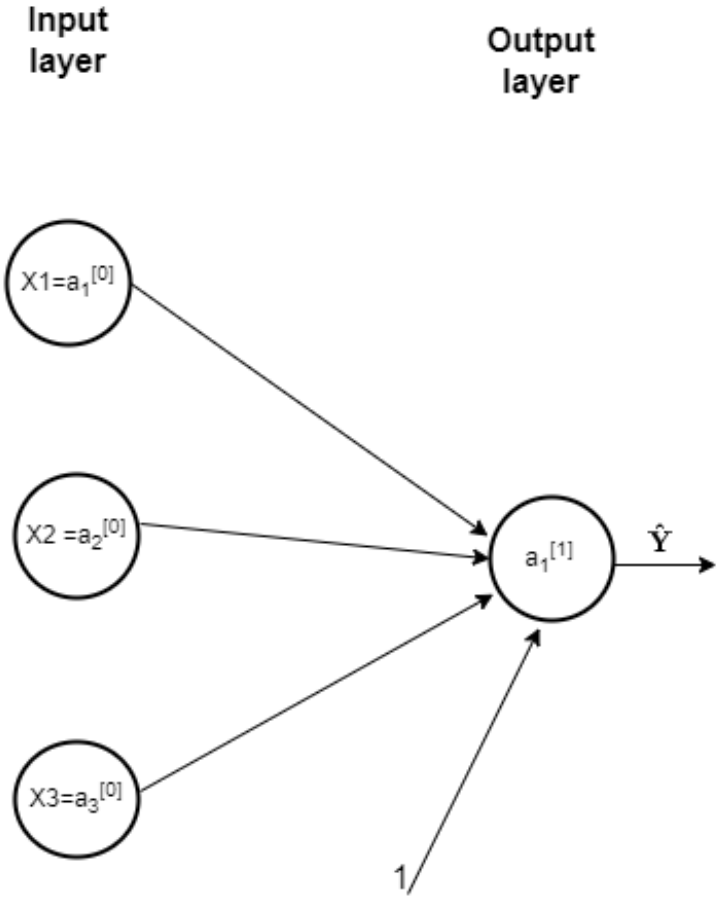
Forward propagation: classical NN model



Backpropagation and gradient descent used to train NNs



Logistic regression and Neural Networks



Parameters:

$\mathbf{W}_{1x3}$

b

$Z=WX+b$

$\mathbf{A} = \hat{\mathbf{Y}} = \sigma(\mathbf{Z})$

Loss function

$Z = W * X + b$, Sigmoid function: $\hat{Y} = \sigma(Z) = \frac{1}{1+e^{-Z}}$

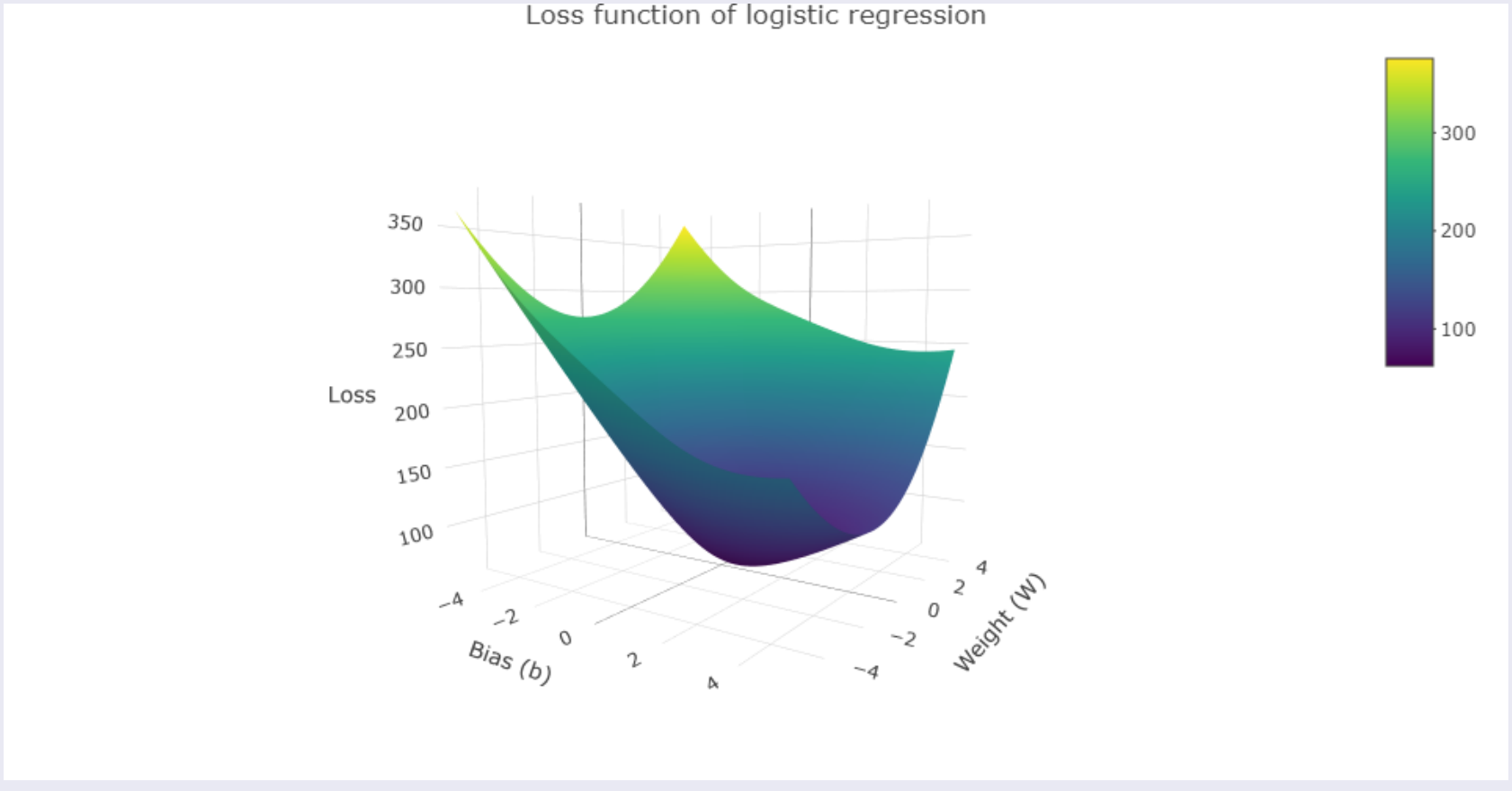
Binary Cross-Entropy Loss function:

$$J(W, b) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

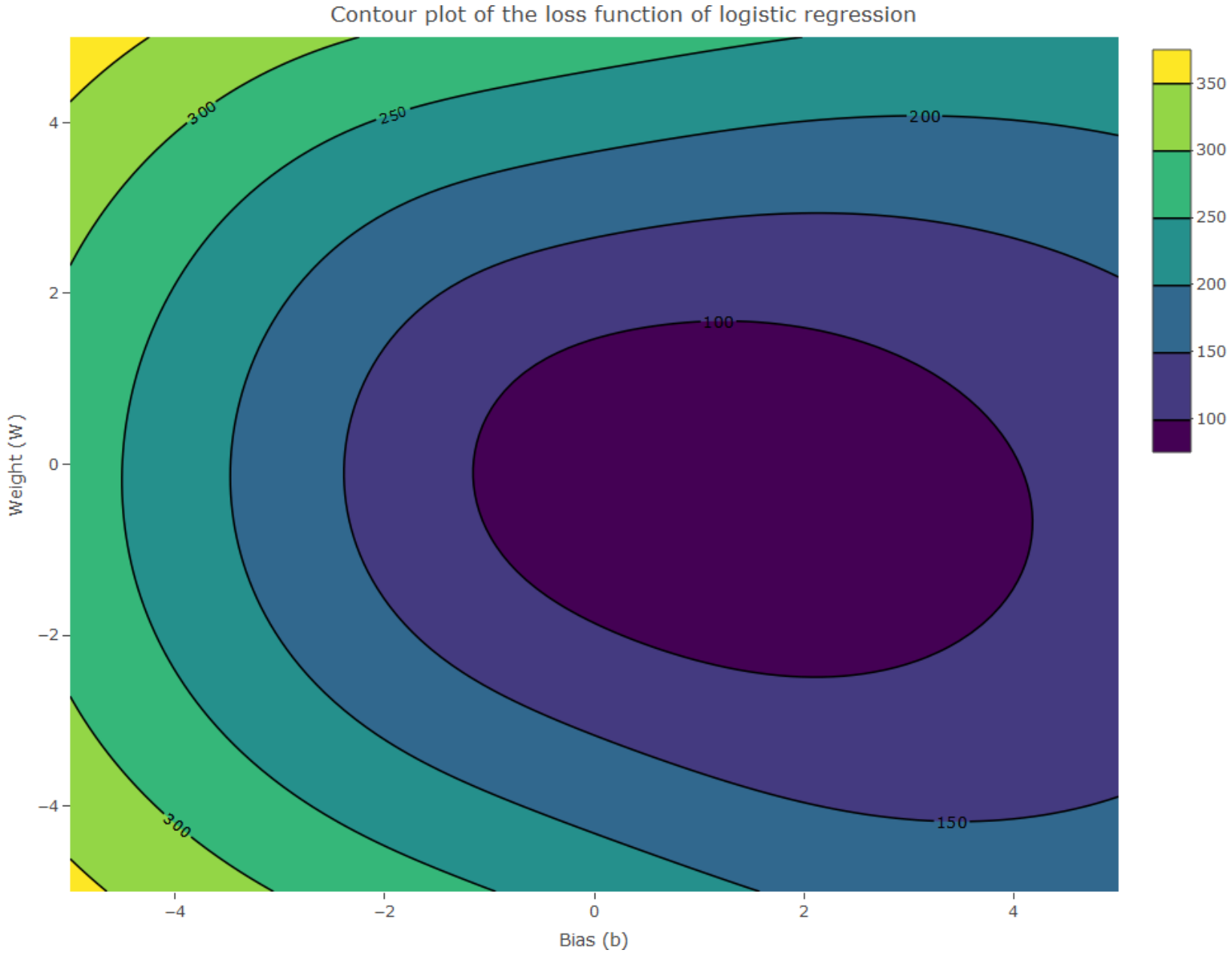
where m is the number of training examples, $y^{(i)}$ is the true label of the i-th example, $\hat{y}^{(i)}$ is the predicted probability of the i-th example being positive, and W and b are weights and biases of the logistic regression model.

Loss function - cont.

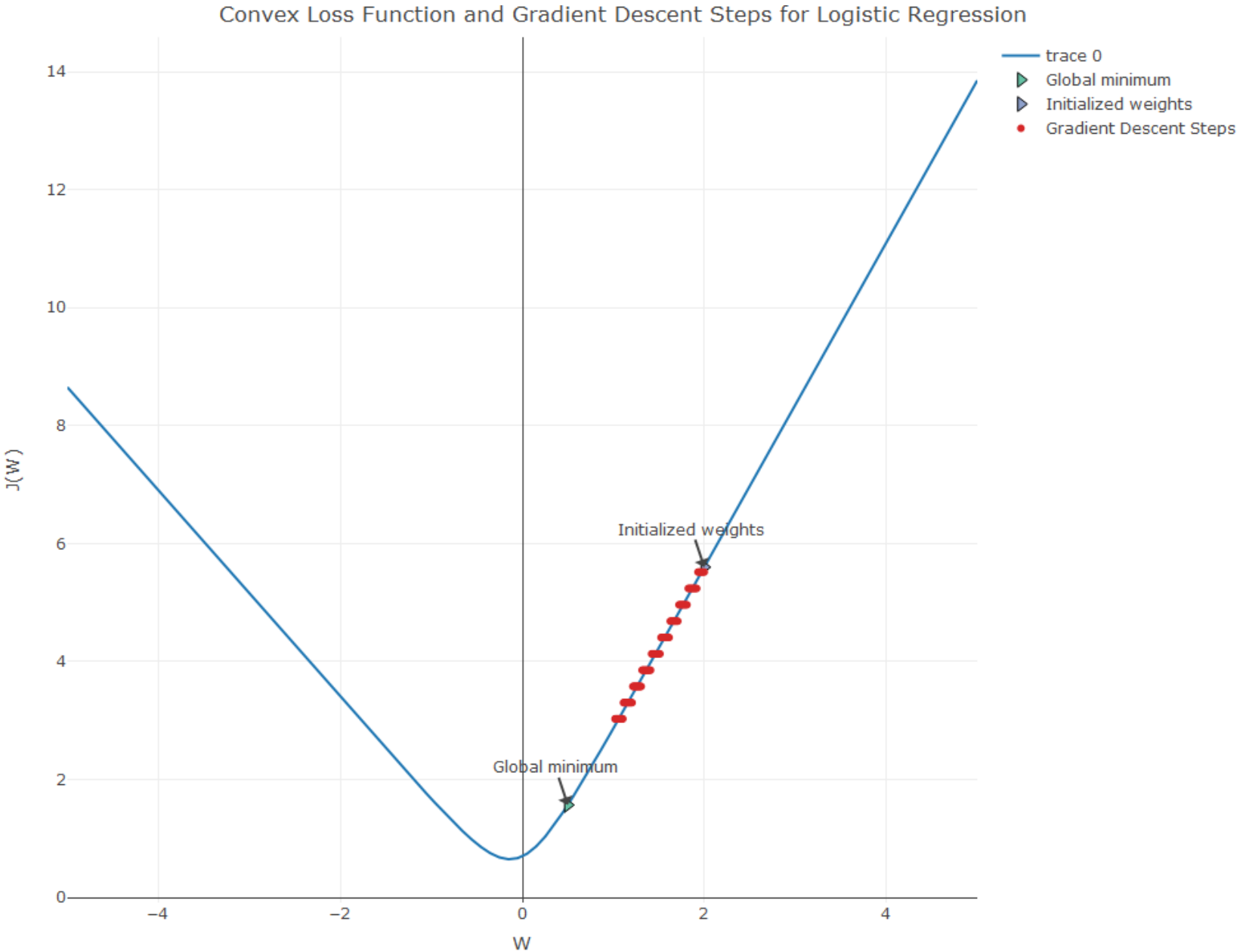
Interactive loss function (Click here)



Loss function - cont.



Optimization: Gradient descent illustrated



Section 3

NN in R

Example: Data set Infertility after Spontaneous and Induced Abortion

Load the packages and data, split the data into train (80%) and test (20%)

```
#  ADAPTED FROM:  neuralnet: Training of Neural Networks
#  by Frauke Günther and Stefan Fritsch The R Journal Vol. 2/1, June 2010, ISSN 2073-4859
#  AND MATT BOGARD
#  https://www.r-bloggers.com/2010/12/r-code-example-for-neural-networks
#load libraries
library(neuralnet)
library(caret)
library(MLmetrics)
library(datasets)
#Outcome for prediction case = 1 infertile, case=0 denotes control.
#Type ?infert to learn about the data set.
# Check for class imbalance problem, this might be important to judge the model's usefulness
# in predicting the infertility cases
table(infert$case)
```

```
##
##   0   1
## 165  83
```

```
# Split the data into train/test (80/20 split)
set.seed(123)  # Set a random seed for reproducibility
index <- createDataPartition(infert$case, p = 0.8, list = FALSE)
train_data <- infert[index, ]; test_data <- infert[-index, ]
```

Example: Data set Infertility after Spontaneous and Induced Abortion - cont.

Train the model, obtain F1-Score on the test (true holdout) data set

```
# Train the neural network on the training data
trained_nnet <- neuralnet(
    case ~ age + parity + induced + spontaneous,
    data = train_data,
    hidden = 2,
    err.fct = "ce",
    linear.output = FALSE
)
# Use the trained model to predict the probability of the 'case' on the test data
predicted <- predict(trained_nnet, test_data)
# use a threshold of 0.5 to convert the probability of 'case' into a binary prediction 'case/control' when
#the classes are balanced. In case of class imbalance, threshold needs to be
# optimized to maximize F1-score (e.g., nested cross-validation)
predicted_class <- ifelse(predicted > 0.4, 1, 0)
# F1-score balances recall (or sensitivity=TP/(TP + FN))
# and precision (or Positive Predictive Value = TP/(TP+FP))
# F1-score = TP/(TP+1/2*(FP+FN))
# where TP - True Positives, FP - False positive,
# FN - False Negative and TN - True Negative.
f1 <- F1_Score(predicted_class, test_data$case)
print(paste("F1 Score on Test Data:",round(f1,3),sep=""))
```

```
## [1] "F1 Score on Test Data:0.857"
```

Example: Data set Infertility after Spontaneous and Induced Abortion - cont.

Print Confusion matrix

```
xtab <- table(factor(test_data$case,levels=c(1,0)),factor(as.vector(predicted_class),levels=c(1,0)))
caret::confusionMatrix(t(xtab))
```

```
## Confusion Matrix and Statistics
##
##
##      1  0
##   1 13  7
##   0  2 27
##
##                Accuracy : 0.8163
##                  95% CI : (0.6798, 0.9124)
##     No Information Rate : 0.6939
##     P-Value [Acc > NIR] : 0.03963
##
##                   Kappa : 0.6045
##
##  Mcnemar's Test P-Value : 0.18242
##
##             Sensitivity : 0.8667
##             Specificity : 0.7941
##          Pos Pred Value : 0.6500
##          Neg Pred Value : 0.9310
##              Prevalence : 0.3061
##          Detection Rate : 0.2653
##    Detection Prevalence : 0.4082
##       Balanced Accuracy : 0.8304
```

Example: Data set Infertility after Spontaneous and Induced Abortion

Basic 10K Cross-validation to obtain accurate F1-Score

```
set.seed(124)  # Set a random seed for reproducibility
cv_f1_scores <- numeric(10)

for (i in 1:10) {
    index <- createDataPartition(train_data$case, p = 0.9, list = FALSE)
    cv_train_data <- train_data[index, ]
    cv_test_data <- train_data[-index, ]

    cv_nn <- neuralnet(
        case ~ age + parity + induced + spontaneous,
        data = cv_train_data,
        hidden = 2,
        err.fct = "ce",
        linear.output = FALSE
    )
    cv_predicted <- predict(cv_nn, cv_test_data)
    #probability threshold of 0.4 is chosen to increase sensitivity, but here nested cross-validation
    # step can be used to optimize the probability threshold by maximizing the F1-score.
    cv_predicted_class <- ifelse(cv_predicted > 0.4, 1, 0)
    cv_f1_scores[i] <- F1_Score(cv_predicted_class, cv_test_data$case)
}
# Calculate the mean F1 score from cross-validation
mean_cv_f1 <- mean(cv_f1_scores)
print(paste("Mean F1 Score from 10-fold Cross-Validation:",round(mean_cv_f1,3),sep=""))
```

```
## [1] "Mean F1 Score from 10-fold Cross-Validation:0.779"
```

Example: Data set Infertility after Spontaneous and Induced Abortion - cont.

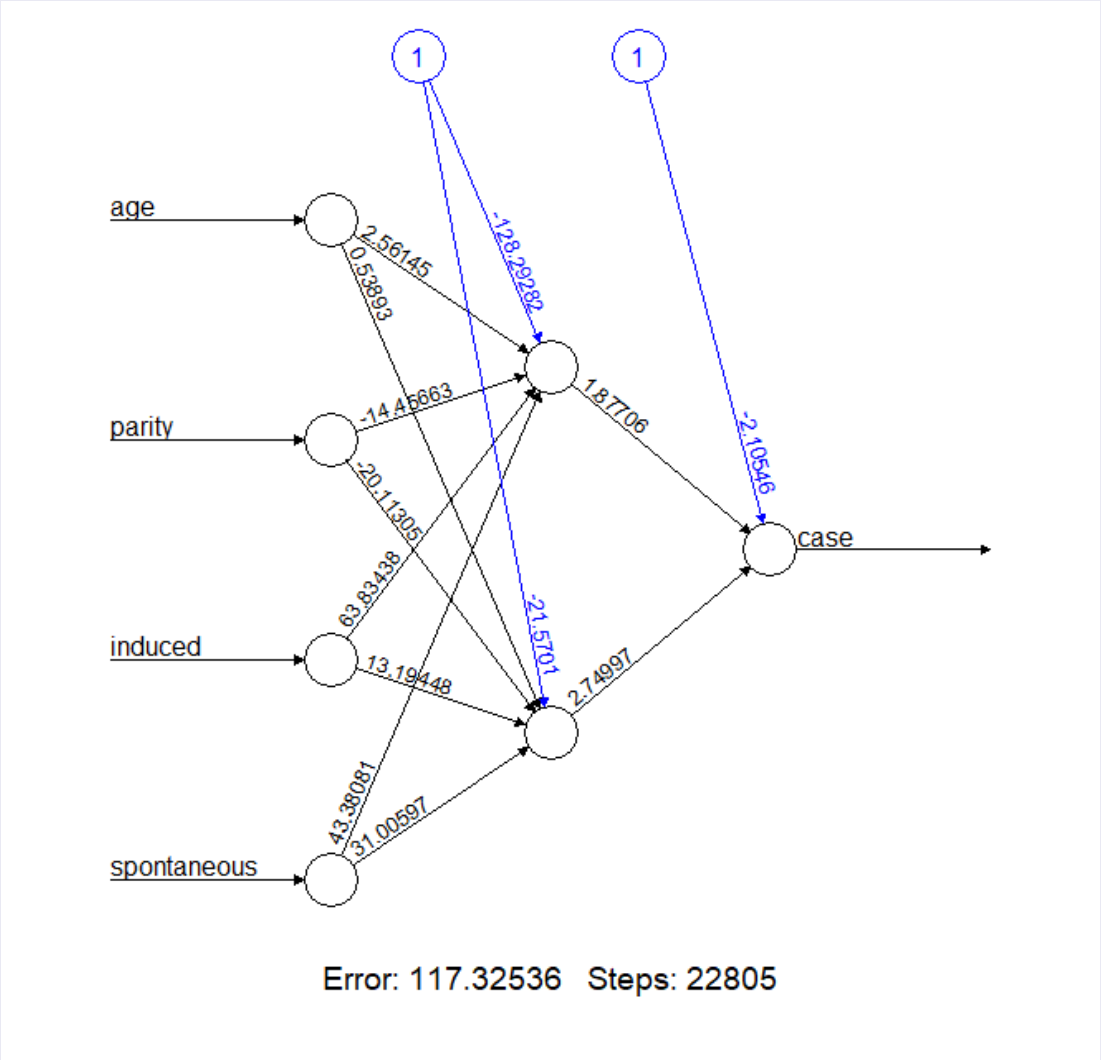
Refit the selected model using the entire data set

```
# The model predictive performance can be further improved
#depending on the problem that we are working on. For example,
# medical applications could require higher sensitivity as missing true positives
#could have fatal impacts on patients.
# Suggested steps to improve the model would be to use (e.g., cross-validation) to:
#1. Perform variable selection in nested cross-validation
#2. Perform model selection with cross-validation by fitting different types of machine learning approaches
#3. Collect more predictors and more observations if possible
#3. Perform hyperparameter tuning, such as learning rates, number of neurons in
#  hidden layers, probability threshold
# or test other machine learning approaches.
#   Once the model performance satisfies the objectives and needs
#   the final step is to refit the neural net using tuned hyperparametrs and selected features
# and model from the cross-validation step using the original train data
# and assess final performance (F1-score) on the true hold out test data set.
# In the final step the same model is refitted using the entire data set,
#   the model is implemented in production, and used
#   with new arriving data.
#   Model's predictive performance is monitored to determine
#   the frequency with which
#   the model need to be retrained.
fulldata_nnet <- neuralnet(
    case~age+parity+induced+spontaneous,
    data=infert, hidden=2, err.fct="ce",
    linear.output=FALSE)

plot(fulldata_nnet)
```

Example: Data set Infertility after Spontaneous and Induced Abortion - cont.

Visualization of the estimated NN



Example: Data set Infertility after Spontaneous and Induced Abortion - cont.

Interpret the final model

```
# Print the matrix of estimated weights and biases
# Values shown on the estimated NN graph
fulldata_nnet$result.matrix
```

```
##                                [,1]
## error                  1.252096e+02
## reached.threshold      8.146767e-03
## steps                  1.717000e+03
## Intercept.to.1layhid1  1.308941e+00
## age.to.1layhid1        1.725833e+00
## parity.to.1layhid1     1.005523e+00
## induced.to.1layhid1   -1.625591e-01
## spontaneous.to.1layhid1  1.497875e+00
## Intercept.to.1layhid2  5.500452e+00
## age.to.1layhid2       -1.142567e-01
## parity.to.1layhid2     1.716619e+00
## induced.to.1layhid2   -2.143374e+00
## spontaneous.to.1layhid2 -3.276646e+00
## Intercept.to.case      1.846294e+00
## 1layhid1.to.case       1.646391e+00
## 1layhid2.to.case      -5.297647e+00
```

Section 4

DNNs

What are the Deep Neural Networks (DNNs)?

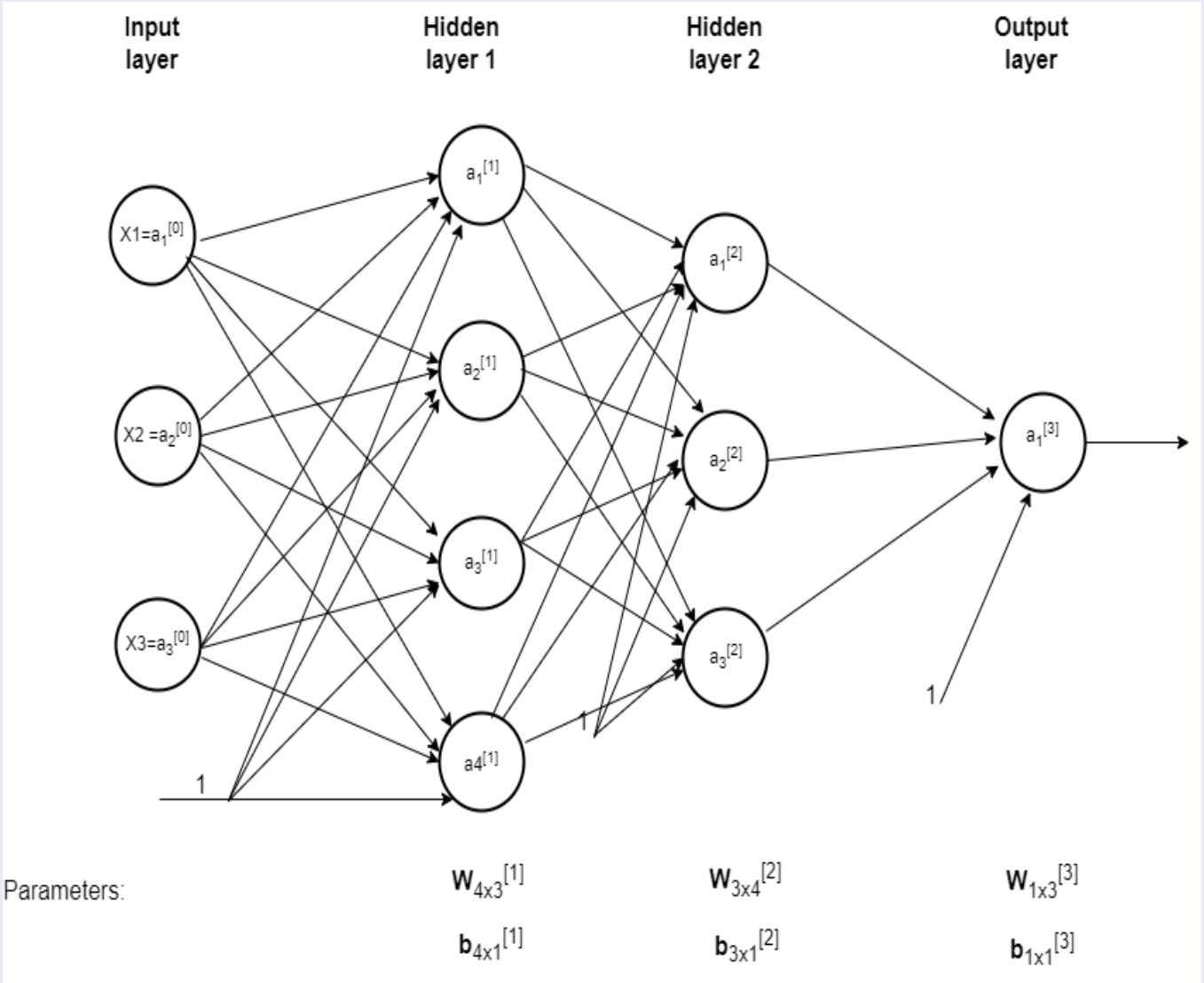
- In general, Neural Networks with more than one hidden layer are considered deep neural networks, but number of hidden layers can vary from field to field;
- The name Deep Neural Networks is given for branding.

Why Deep Neural Networks (DNN)?

- DNN are needed to solve complex problems:
  - image and speech recognition,
  - natural language processing,
  - game playing;
- Classical Neural Networks with only a few layers may not be able to learn all the complex representations in the data to accurately classify outcomes;
- DNN are more robust to noisy and incomplete data, as well as more efficient in their use of data and computation;
- Can extract more features from the data, use inferred features from the previous layer into the next layer.

Deep Neural Networks (DNNs) Intro

DNN with two hidden layers



Shallow Neural Nets versus Deep Neural Networks (DNNs)

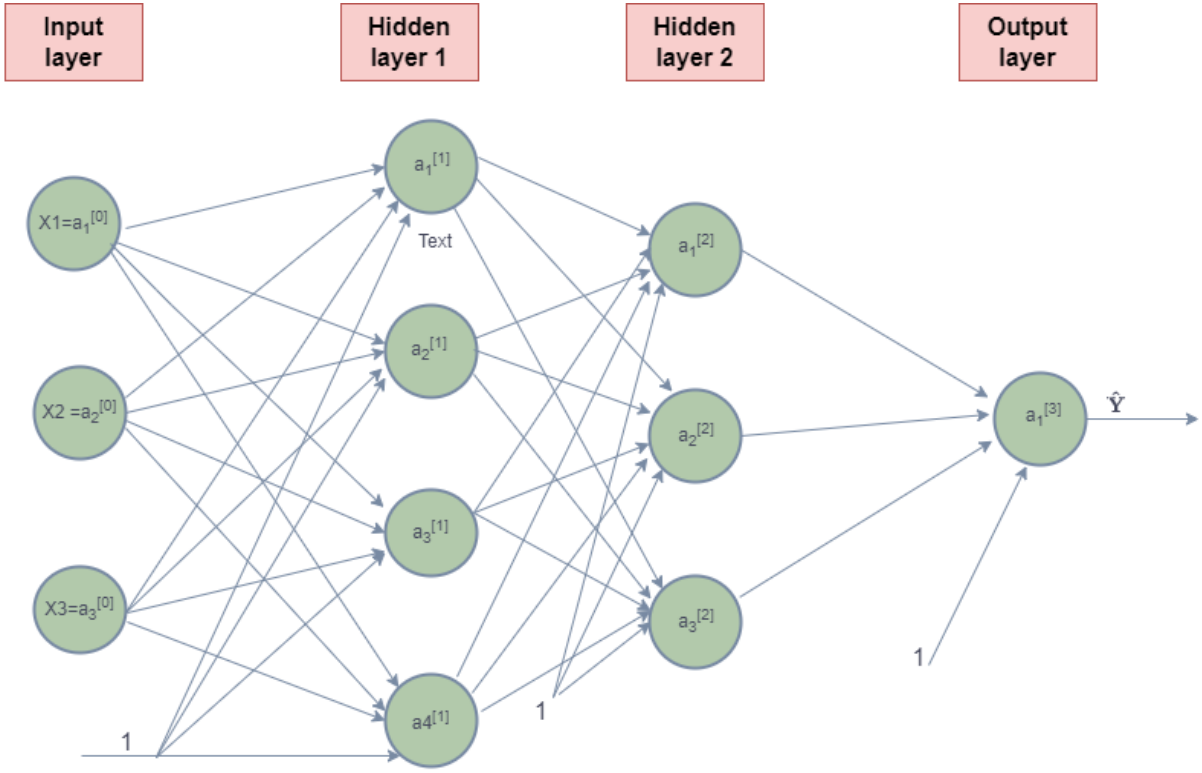
How DNNs can learn complex and hierarchical representations of data?

- DNNs have more layers, and each layer can learn to represent more abstract and high-level features of the data;
- Each layer in DNN uses the abstract features learned from the previous layer using the non-linear activation function;
- Non-linear activation functions in each layer allows the network to learn non-linear decision boundaries;
- Hierarchical learning allows deep neural networks to capture more nuanced and complex relationships between the input data and the output labels.

Shallow NN versus Deep Neural Networks (DNNs) example: image classification task: cats and dogs

Layer	**Shallow NN, one hidden layer**	**DNN**
Input	Learn image edges and basic shapes	Same
Hidden	Combine the edges and shapes in more complex feature: outline of the cat or dog	Learn more complex patterns: fur textures or body parts
Hidden	None	Continue learning abstract and high-level features: the shape of faces or fur textures
Output	Use learned features to classify the image: cat or dog	Same

Forward propagation: classical DNN model



Parameters: $W_{4x3}^{[1]}$ $b_{4x1}^{[1]}$ $W_{3x4}^{[2]}$ $b_{3x1}^{[2]}$ $W_{1x3}^{[3]}$ $b_{1x1}^{[3]}$

Model: Forward propagation Equations

Input: $A^{[0]}=X^{[0]}$

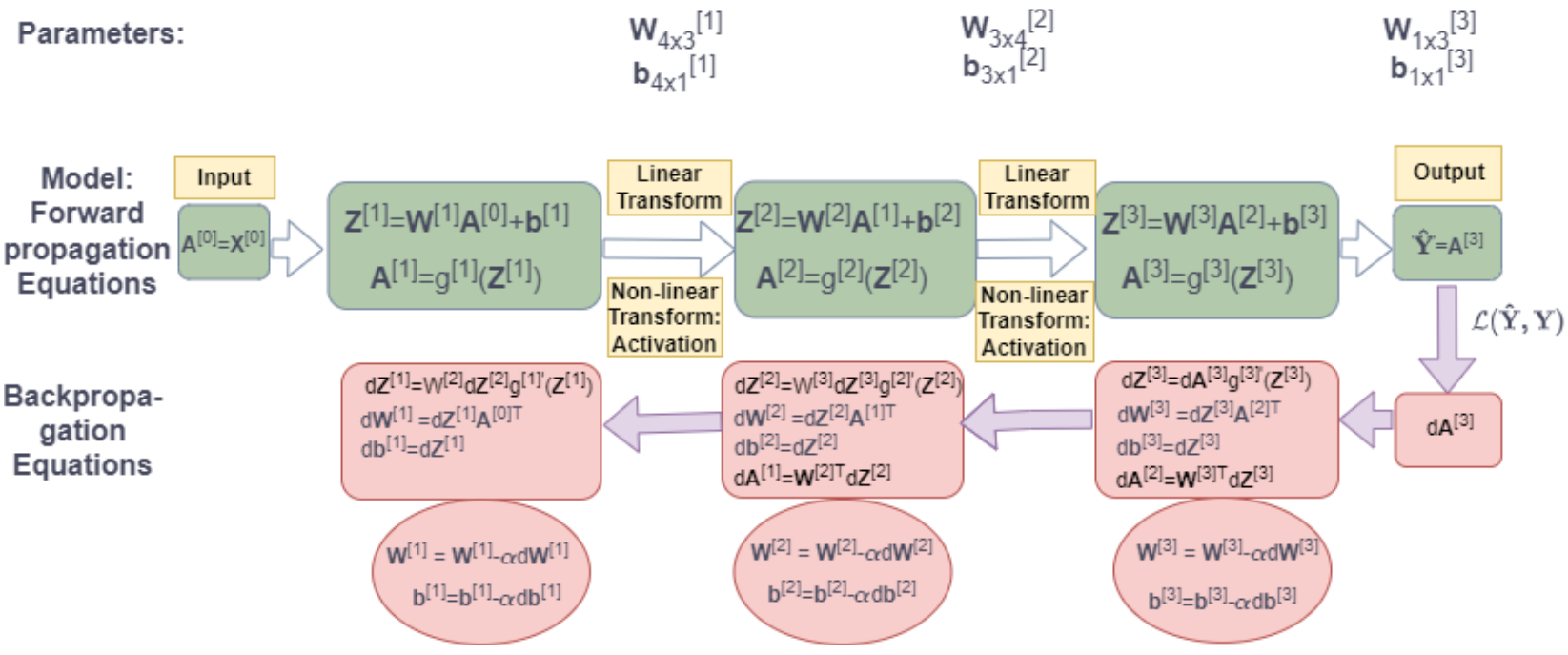
$Z^{[1]}=W^{[1]}A^{[0]}+b^{[1]}$, $A^{[1]}=g^{[1]}(Z^{[1]})$ — Linear Transform; Non-linear Transform: Activation

$Z^{[2]}=W^{[2]}A^{[1]}+b^{[2]}$, $A^{[2]}=g^{[2]}(Z^{[2]})$ — Linear Transform; Non-linear Transform: Activation

$Z^{[3]}=W^{[3]}A^{[2]}+b^{[3]}$, $A^{[3]}=g^{[3]}(Z^{[3]})$

Output: $\hat{Y}=A^{[3]}$

Backpropagation and gradient descent used to train DNNs



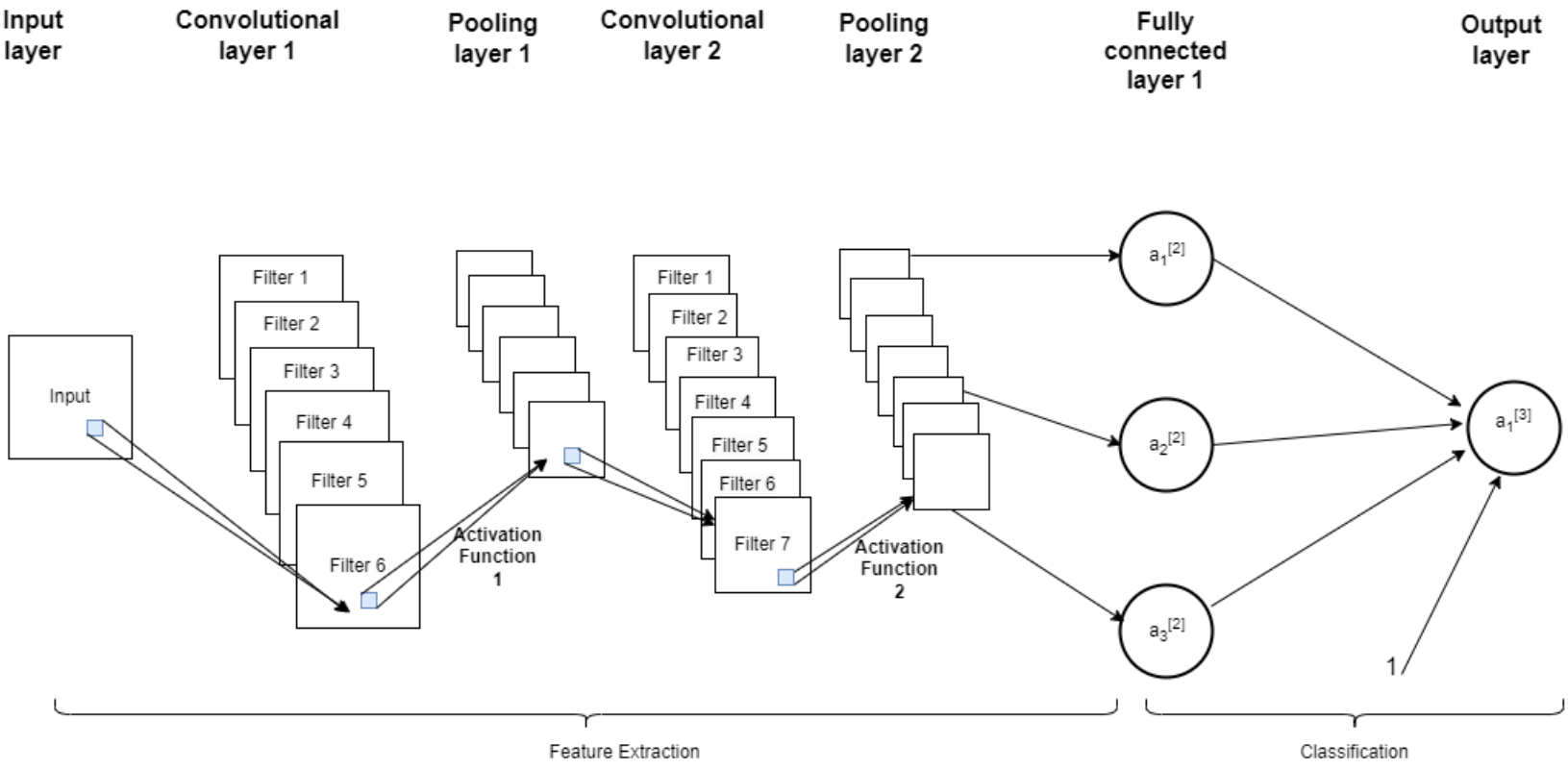
Section 5

CNNs

Convolutional neural networks (CNNs) - Intro

- Convolutional Neural Networks (CNNs) are a class of neural networks that are widely used for:
  - image processing,
  - video processing,
  - text classification,
  - text summarization;
- CNNs can automatically learn and extract features from input images, videos or text.

Key Ingredients of CNNs - diagram



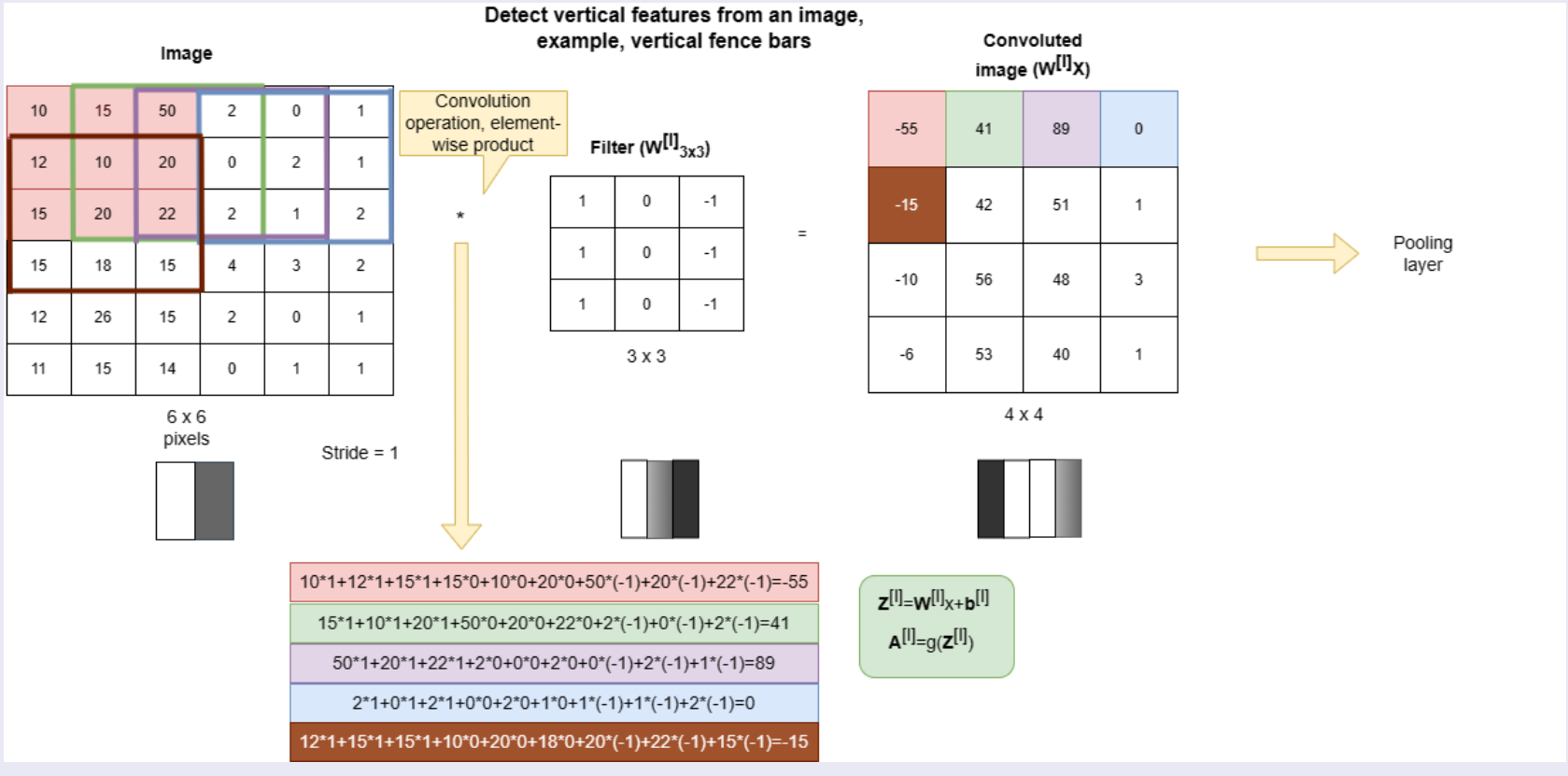
Key Ingredients of CNNs

Convolutional Layers

- Extract features from input images;
- Identify different visual patterns in the input image;
- Use convolution operations to learn filter matrices (also known as a kernels, or weights) by:
  - sliding a filter over the input image,
  - computing a dot product between the filter and the image pixels at each position,
  - producing a new feature map as output;
- The learned filters act as feature detectors, identifying different visual patterns in the input image;
- Convolution channels can incorporate different sources of information (e.g., colored image with three channels: red, green, and blue, each representing the intensity of the respective color at each pixel. CNN can recreate Full-color image using filter - width x height x nchannels = 3 x 3 x 3).

Key Ingredients of CNNs - cont

Convolutional operation



Key Ingredients of CNNs

Activation Functions

- Introduce non-linearity into the network, enabling it to learn complex patterns and relationships between features;
- Output value at a given spatial location in the convolutional layer is the sum of the filter outputs at that location;
- Convolutional layer output is passed through a non-linear activation function, such as ReLU, to introduce non-linearity in the model.
- The resulting feature maps are passed on to the pooling layer.

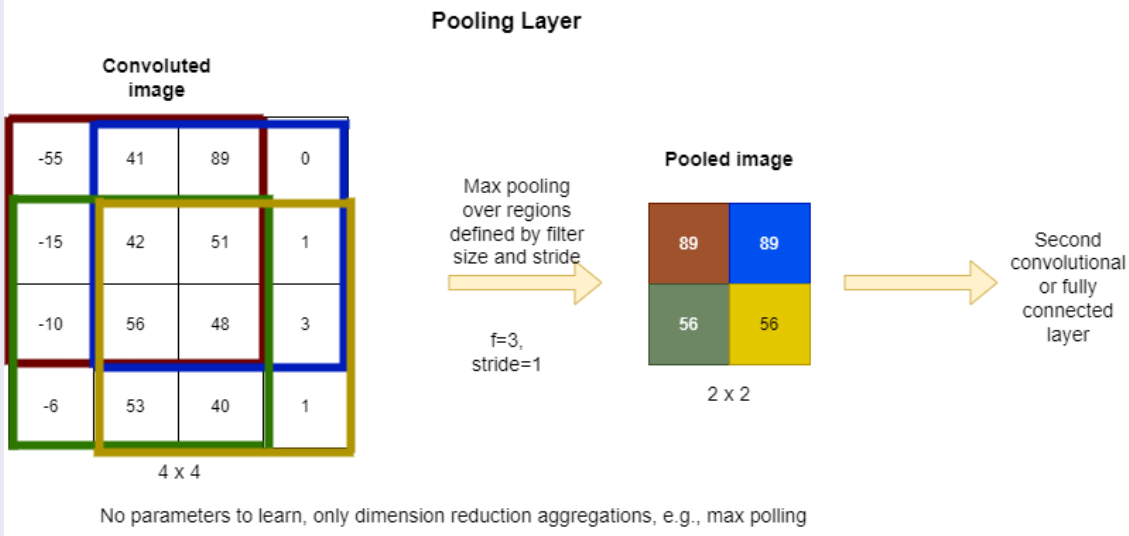
Key Ingredients of CNNs

Pooling Layers

- Used to downsample the feature maps produced by convolutional layers;
- The most common type of pooling operation is max pooling, which extracts the maximum value within a sliding window of pixels.
- This reduces the spatial size of the feature maps, making the network more computationally efficient and reducing overfitting.

Key Ingredients of CNNs

Pooling Layer illustrated



No parameters to learn, only dimension reduction aggregations, e.g., max polling

Key Ingredients of CNNs

Back-propagation

- Backpropagation in CNNs involves a combination of standard backpropagation and modifications with respect to the convolutional and pooling layers.
- The weights in the convolutional layer are shared across multiple spatial locations, which utilizes a "backpropagation through convolution" algorithm, which involves flipping the filter and sliding it across the input to calculate the gradients.
- The pooling operation in the pooling layers is not differentiable, so the gradients cannot be directly calculated. Approximation technique called "max pooling with switches" is used to store the index of the maximum element in each pooling window during the forward pass, and then using the indices gradients are backpropagated.

Advantages of CNNs for image and video processing tasks

- Parameter Sharing: By using the same filters across different locations in the input image, CNNs are able to learn more efficiently and generalize better to new data;
- Sparsity of the connections: CNNs employ sparse connections (localized connectivity pattern between neurons) to exploit the spatial structure and hierarchical nature of the data;
- Hierarchical Learning: features are learned in a hierarchical manner, with lower layers learning high-level features (such as edges and corners) and higher layers learning more complex features (such as object parts, angles and textures);
- Translation Invariance: CNNs are able to recognize objects in an image regardless of their position, rotation, or scale, making them highly robust to variations in the input.

CNN compared to the classical DNN

- Each neuron in classical DNNs is connected only to the neurons in the previous and next layer, hence, classical DNNs do not have a built-in mechanism to identify spatial structures in the data;
- In image recognition tasks, a classical deep neural net could potentially learn to recognize certain features of an object (such as the shape of its edges), but it would not be able to distinguish where in the image those features occur or how they relate to each other spatially.
- CNNs are particularly efficient in problems where there is a strong spatial structure to the input data:
  - image recognition,
  - natural language processing,
  - speech recognition (e.g., transcribe audio speaking into text, CNN can extract local acoustic features).

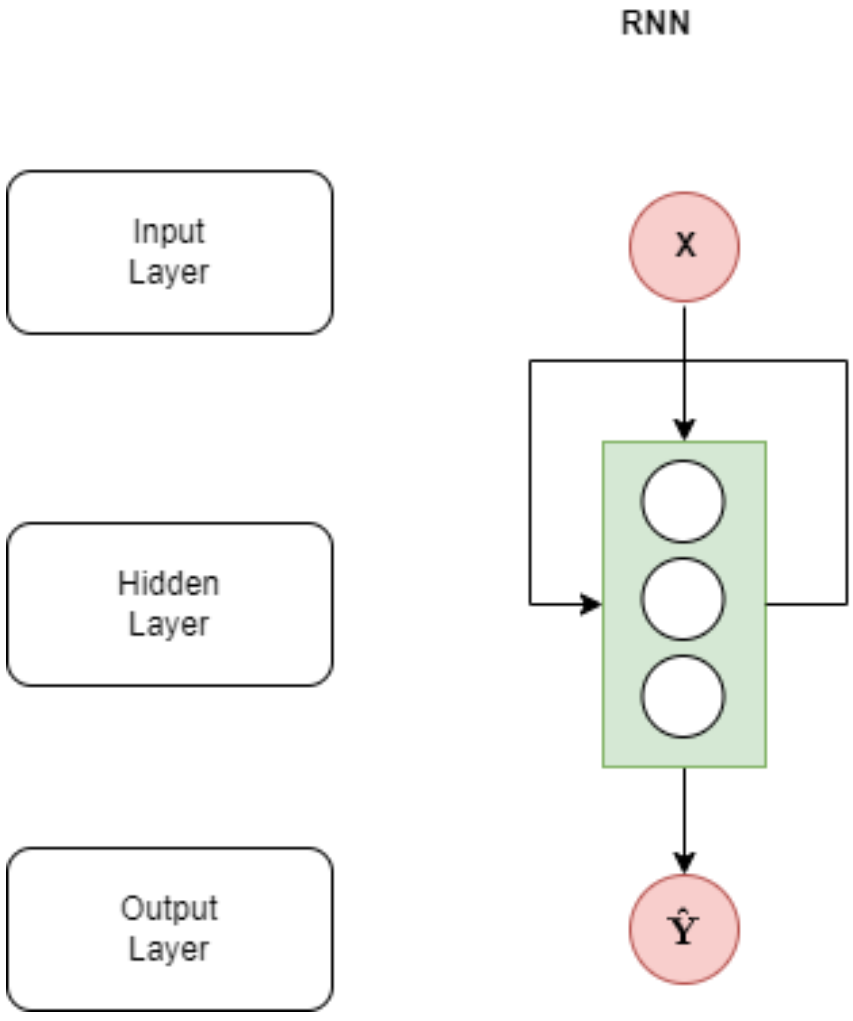
Section 6

RNNs

Recurrent neural networks (RNNs) - cont.

- RNNs are specialized for sequence processing tasks;
- RNNs are designed to handle sequential data, such as time series, text, and speech.
- Unlike feedforward neural networks, RNNs have feedback connections that allow them to store information about past inputs and use it to make predictions.
- RNNs are particularly effective in tasks that require capturing long-term dependencies and understanding context.

Recurrent Neural Networks (RNNs) - vanilla

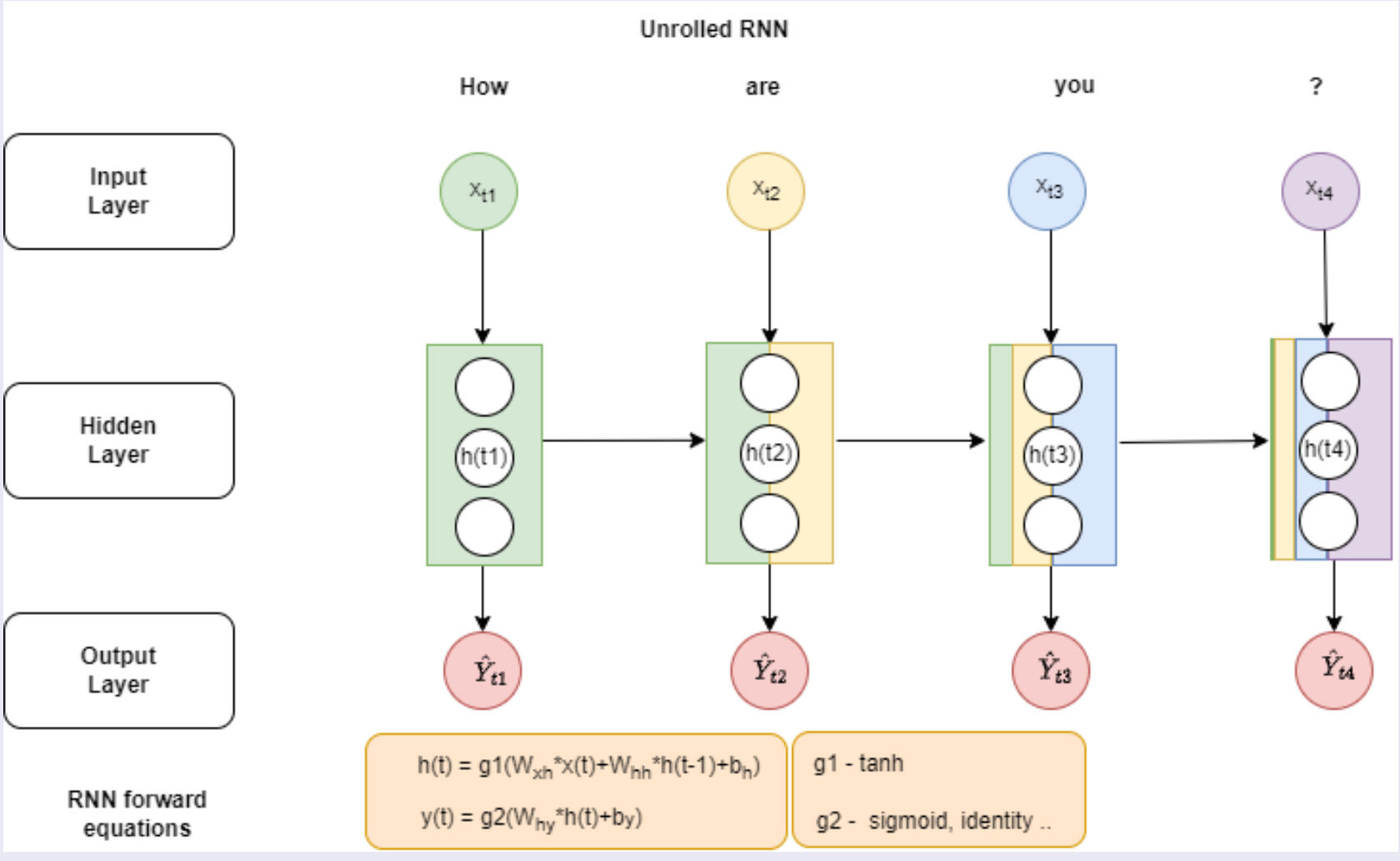


Key components of RNNs

- RNNs have a recurrent layer that consists of recurrent units (often referred to as cells);
- Each recurrent unit takes an input and a hidden state from the previous time step and produces an output and a new hidden state;
- The hidden state serves as the memory of the network, enabling it to retain information from previous inputs. The output of each time step can be used for prediction or passed as input to the next time step;
- Can run into the problem of vanishing gradient descend, as the vanilla RNN suffers from the short memory issue, i.e., it tends to retain info from most recent time points, but forgets the information from the previous time info.

Unrolled Recurrent Neural Networks (RNNs) - vanilla

Coloring schemes indicate portions of info retained at each time



Section 7

LSTM

Long Short-Term Memory (LSTM)

- Is a type of recurrent neural network (RNN) model designed to address the vanishing gradient problem in traditional RNNs;
- Can capture long-term dependencies in sequential data;
- LSTM networks have (long) memory cells that can selectively retain or forget information over time using three gates, making them well-suited for tasks involving long sequential or time-series data;
- Hidden states used as short-memory.

LSTM Key components

- The memory cell;
- The input gate;
- The forget gate;
- The output gate;
- The hidden state;
- The memory cell and the three gates are composed of neural sub-networks with their own weight parameters;
- The hidden state summarizes the information at the current time step and passes it on to the next step.

LSTM Key components

The memory cell

- Stores and updates the information at each time point.
- It's current input is from the:
  - input gate,
  - forget gate, and
  - previous memory cell value;
- Output is:
  - updated memory cell value.

LSTM Key components - cont.

The memory cell is controlled by the three gates

- The input gate filters and selects relevant information to remember based on the current input and the previous hidden state;
- The forget gate controls what information should be forgotten and discarded from the the previous time step of the memory cell;
- The output gate filters memory cell content based on the the current input and the previous hidden state that is passed as the hidden state or the final output,
  - the output gate is the connection between the memory cell (long-term memory) and the hidden state (short term memory).

Section 8

Applications

CNNs

- CNNs are used for image classification to leverage their ability to capture spatial relationships in images;
- Object Detection: CNN can classify each pixel in the image, thus identifying and localizing objects in images;
- Healthcare: CNNs are employed for medical image analysis.

Vanilla RNNs

- Simple Sequential Tasks that do not require capturing long-term dependencies:
  - Predicting the next element in a short sequence,
  - Generating short sequences of text,
  - Simple pattern recognition,
  - Basic time series analysis;
- Language Modeling, the goal is to predict the next word or character in a sequence of text (e.g., autocomplete);
- Music Generation, by learning patterns and dependencies in musical sequences;
- Video Analysis, predict the next frame in a video sequence.

LSTM

- Natural Language Processing (NLP): LSTM are used for:
  - text classification,
  - sentiment analysis,
  - machine translation,
  - language generation;
- Speech Recognition (transcribe audio into text, LSTM models are commonly used for capturing the longer temporal dependencies in speech);
- Healthcare: RNNs/LSTMs are utilized for time-series data analysis and predicting patient outcomes.

Combined use of CNNs and RNNs (LSTMs)

The CNNs learn local spatial features, while the RNNs capture the temporal features

- Image Captioning, where the goal is to generate textual descriptions for images, CNNs can be used to extract visual local features from the images, these features can be fed into RNNs to generate the corresponding captions;
- Video Classification, CNNs can extract spatial features in each individual frame, while RNNs can capture temporal dependencies across frames;
- Speech Recognition, CNNs can extract the local acoustic features, which can can then be fed into RNNs to model the sequential dependencies in the speech data;
- Natural Language Processing, such as text classification or sentiment analysis, CNNs can extract local text features (n-gram patterns), which then can be fed into RNNs (LSTMs) to model long-range dependencies and contextual information.

Section 9

Examples

EXAMPLE 1: Binary classification on textual data

Social science problem formulation and data description

- Input: journalistic reports of potential protests
- Output: classify protests versus non-protests.
- Data: One year worth of data were labeled manually, 7000 reports of non-protests and 300 into protests. The social scientists used 100 keywords (words and phrases) to label manually each report into protest and non-protest.

EXAMPLE 1: Binary classification on textual data - cont.

Data set preparation and model training

- Using the expert domain keywords we created a data set with manually created features representing the words and phrases frequencies of appearances within each report;
- Using this data set, random forest model was trained to classify the reports into protests/non-protests;
- Nested cross-validation with repetitions was used to select the features that maximize the predictive performance based on the f1-score and to assess predictive performance of each of the models;
- Predictive performance was assessed on a hold out test data set that the model has not seen during the training;

EXAMPLE 1: Binary classification on textual data - cont.

Challenges

- Class-imbalance issue;
- Over estimated predictive performance: the hold out data set from the same year yield correlated samples between training and test data sets within the same year;

EXAMPLE 1: Binary classification on textual data - cont.

Adressing the challenges

- Class-imbalance issue was addressed by training the random forest on the data set with over-sampled classes, while test data set was used in it's original state;
- A subset of another year worth of data was labeled to test the predictive performance of the trained model,
- We used both years with labeled data to train another random forest model and produce a predictive performance measures using a hold out data set from the second labelled year;
- The resulting predictive performance was comparable from both trained models (one year versus two year combined training dataset);
- The model was then applied to data worth of 10 additional years.

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Problem formulation

- Medical lab in a hospital installed a new instrument and needed to re-calibrate its threshold with an aim to reduce the cost by reducing the number of samples sent for testing using results from a pre-screening test.
- The pre-screening test consisted of two continuous measurements;
- The main (high cost) test outcome was binary: positive or negative to an infection;
- The goal was to find the most optimal threshold for each of the two pre-screening measures, such that the sensitivity of these two measures correctly detecting an infection is at least 94%, with a lower 95% confidence interval bound of 92%.

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Data

- The lab data consisted of ~ 3500 samples from patients (with known outcome, i.e., labeled binary response).
- The prevalence of the infection among the patents about $\sim 40\%$.
- Only $\sim 20\%$ of the patients had repeated tests (samples).

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Challenges

- Repeated measures within patient create correlations in the observations within patient. Data splitting into train and test required careful consideration of the observational units in the data set for splitting (train, test). If the train and test data sets are correlated, then the test data set would not consist of new samples predictive power of the model could be overestimated.
- The data set was small to use any neural net models;
- There were only 2 predictors, so not enough variables to improve the predictive power as well;

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Challenges -cont.

- Predictive power assessed via cross-validation likely overestimated; the data set was small to conduct many K-fold cross-validation;
- The sheer amount of models with various thresholds would also lead to over-estimated predictive performance;
- The decision rule had to be simplified as the instrument could only accept logical statements of the type if "threshold for measure 1 is larger than it's optimal value AND/OR threshold for the second measure is larger than the it's optimal value" the sample will be sent for further testing.

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Solution

- Patient samples (repeated measures) are correlated within patient, while patients are independent from each other. Hence, the train set was constructed by selecting 70% of the patients and all their samples, while the rest of the patients and all their samples were assigned in the test data set;
- Logistic mixed effect model was used to correctly estimate the standard errors of the model estimates;
- Nested cross validation was used to fit models with various threshold combinations;
- The choice of the predictive performance measures should reflect the objective to maximize the sensitivity while minimizing the cost;

EXAMPLE 2: Finding optimal thresholds in a lab instrument to meet pre-defined sensitivity of a binary classification problem

Solution - cont.

- Predictive performance was assessed for each combination of the thresholds for the two predictors;
- Simple decision rule was derived based on the selected model and respective predictions.

Section 10

Conclusion

Conclusion

- NN are a class of machine learning models inspired by the structure and function of biological neural networks. They consist of interconnected layers of artificial neurons, with each neuron performing a weighted sum of its inputs and applying an activation function to produce an output;
- DNNs are a subset of NN class with multiple hidden layers between the input and output layers. These hidden layers allow DNNs to learn more complex and abstract representations of the data;
- CNNs are a subclass of neural network designed to capture spatial correlations in the data. Convolutional and pooling layers are helping learn these relationships.

Conclusion - cont.

- RNNs are a subclass of neural network designed for processing sequential data. The recurrent connections in RNNS allow information to be propagated across time steps, enabling them to capture temporal dependencies in the data. RNNs cannot capture long-term dependencies due to the vanishing gradient;
- Long Short-Term Memory (LSTM) is a specific subclass of RNN that addresses the vanishing gradient problem and allows for better capturing of long-term dependencies. LSTM are commonly used for tasks such as natural language processing, speech recognition, and time series analysis.

Section 11

References

References

- UBC Research Commons Hands-on Workshops on Neural Nets using Python Skikit-learn

Free Audit Coursera courses

- Neural networks and Deep Learning
- Convolutional neural networks
- RNN, NLP, sequence models
- Hyperparameter tuning

Free ebooks:

- Beginner-friendly: introduction to Neural Networks and Deep Learning
- Pattern Recognition and Machine Learning, Bishop 2006
- Deep Learning, Goodfellow, I, Bengio, Y, and Courville, Aaron Courville, 2016

References

Other:

RNNs explained, (Hochreiter and Schmidhuber, 1997)(https://www.bioinf.jku.at/publications/older/2604.pdf)

References

Bishop, **C M** 2006 *Pattern recognition and machine learning*. Springer.

Cox, **D R** 1958 On the theory of logistic regression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2): 215–242.

Goodfellow, **I**, **Bengio**, **Y**, and **Courville**, **A** 2016 *Deep learning*. MIT Press.

Hochreiter, **S** and **Schmidhuber**, **J** 1997 Long short-term memory. *Neural Computation*, 9(8): 1735–1780.

Rosenblatt, **F** 1958 The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6): 386.