UNIVERZITET U BIHAĆU
TEHNIČKI FAKULTET
BIHAĆ

Auditorne vježbe iz predmeta

VJEŠTAČKA INTELIGENCIJA I EKSPERTNI SISTEMI

Una Drakulić, MA elektrotehnike
Viši asistent

NEURONSKE MREŽE ZA DEEP LEARNING

1) *Feed-forward neural network (FNN fully-connected)* je osnovni tip umjetne neuronske mreže, poznat i kao višeslojni perceptron (MLP Multi-Layer Perceptron). Njena glavna karakteristika je da informacija teče samo u jednom smjeru, od ulaznog sloja, kroz jedan ili više skrivenih slojeva, prema izlaznom sloju, bez povratnih veza (nema ciklusa). Kod višeslojne mreže, izlaz svakog sloja postaje ulaz narednom sloju. Mreža se sastoji od:

1. Ulaznog sloja – prima podatke (feature vektori).
2. Skrivenih slojeva – svaki neuron računa ponderisanu sumu ulaza, dodaje bias i prolazi kroz nelinearnu aktivacijsku funkciju (npr. sigmoid, tanh, ReLU).
3. Izlaznog sloja – daje predikciju (klasifikaciju ili vrijednost regresije).

Matematički, izlaz jednog neurona se računa kao: $y = f(Wx + b)$

gdje je:

W – matrica težina,
x – vektor ulaza,
b – bias (pomak),
$f(\cdot)$ – nelinearna aktivacijska funkcija.

Treniranje FNN mreže se zasniva na algoritmu *propagacije greške unazad* (*backpropagation*) **i optimizaciji težina** pomoću metoda kao što su:

- Levenberg–Marquardt (trainlm) – vrlo brza konvergencija za male do srednje mreže.
- Scaled conjugate gradient (trainscg) – efikasna za veće mreže i veće skupove podataka.
- Gradient descent (traingd) – jednostavnija, ali sporija metoda.

Tok treniranja:

1. Propagacija ulaza naprijed → izračun trenutnog izlaza.
2. Izračun greške:

$$e = y_{target} - y_{predicted}$$

3. Propagacija greške unazad i ažuriranje težina:

$$W_{new} = W_{old} + \Delta W$$

gdje se ΔW računa na osnovu gradijenta greške po težinama.
Proces se ponavlja kroz više epoha dok se greška ne minimizira (npr. MSE – *Mean Squared Error*).

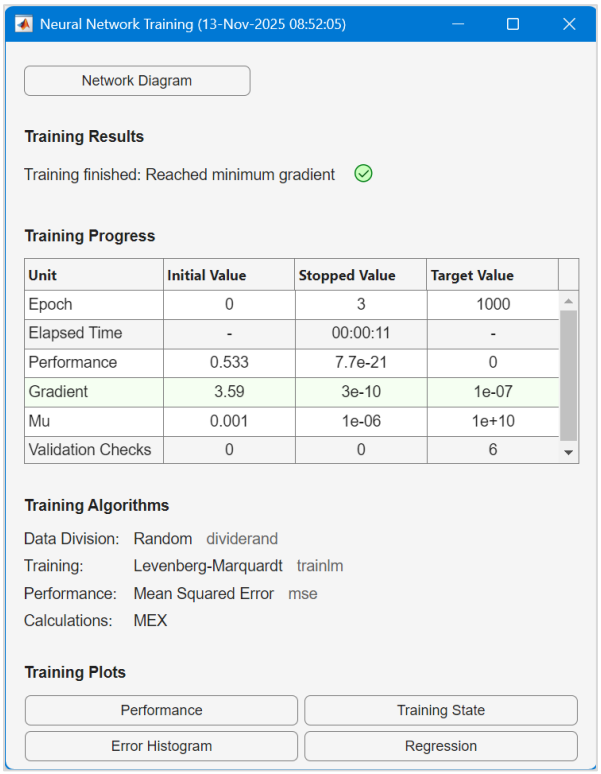
Feed-forward mreže su univerzalni aproksimatori i koriste se u širokom spektru problema:

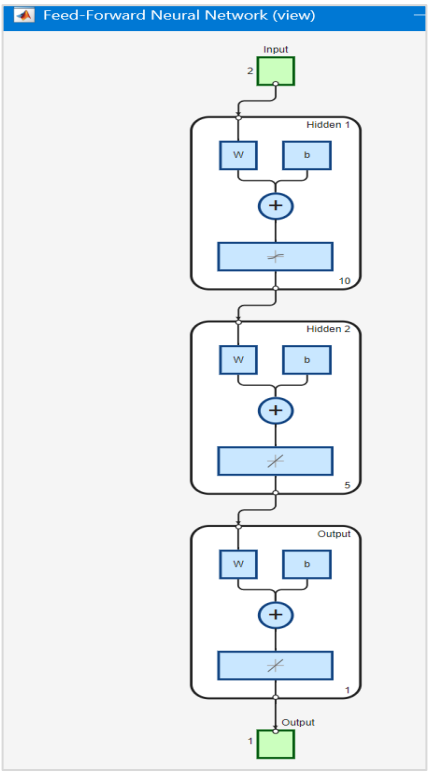
- Regresija (predviđanje numeričkih vrijednosti),
- Klasifikacija (prepoznavanje uzoraka, signala, slika),
- Upravljanje i automatizacija (npr. aproksimacija nelinearnih sistema),
- Predviđanje vremenskih serija (kada se koristi uz “lagged inputs”).

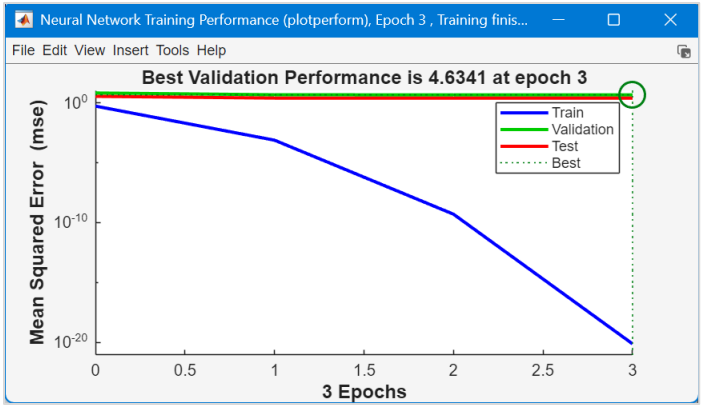
Primjer1

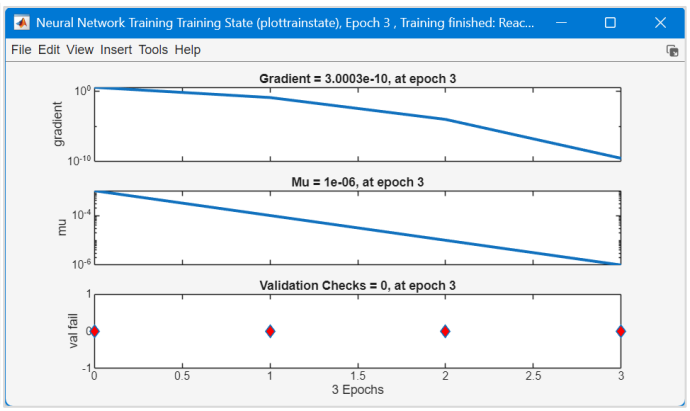
```
% Ulazni i cilj podaci
X = [0 0 1 1; 0 1 0 1];     % 2 ulaza, 4 uzorka
T = [0 1 1 0];              % ciljna vrijednost (XOR problem)

% Kreiranje mreže (2 ulaza, 2 skrivena sloja)
net = feedforwardnet([10 5]);
net.trainFcn = 'trainlm';          % algoritam učenja (Levenberg-Marquardt)
net.layers{1}.transferFcn = 'tansig';   % aktivacijska funkcija 1. sloja
net.layers{2}.transferFcn = 'purelin';  % izlazni sloj (linearan)

% Treniranje
net = train(net, X, T);

% Testiranje
Ypred = net(X);
perf = perform(net, T, Ypred);
disp(['Greška mreže (MSE): ', num2str(perf)]);
```

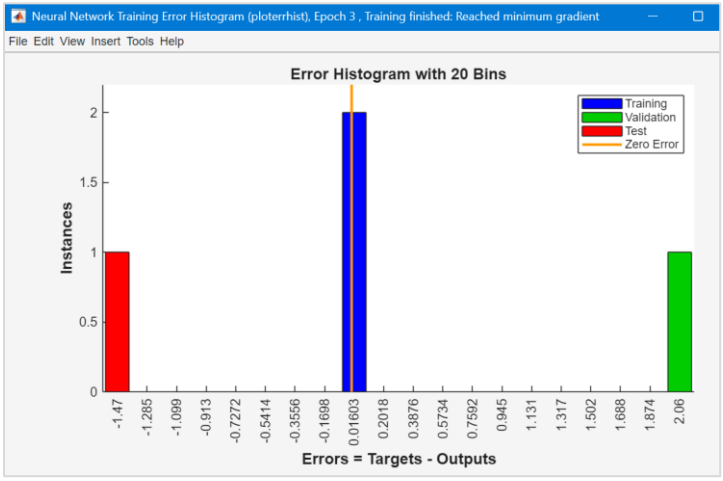
```
>> primjer1
Greška mreže (MSE): 1.7695
```

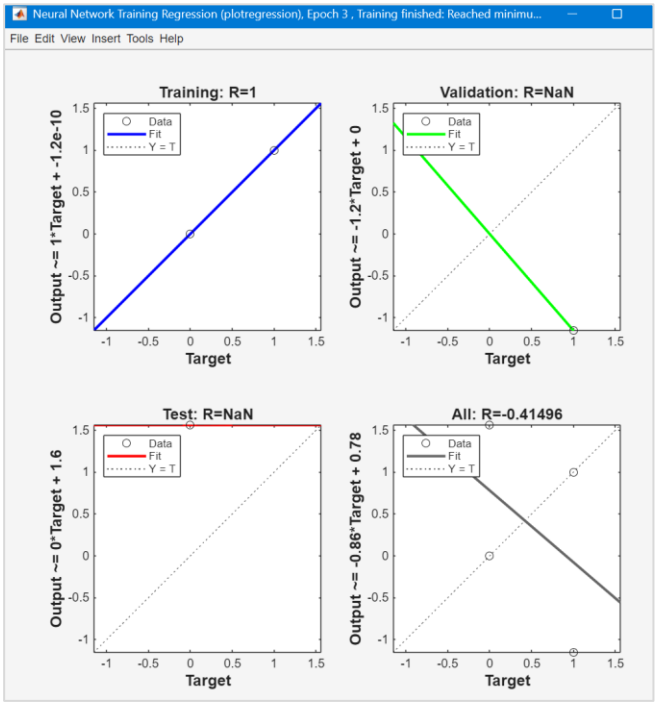








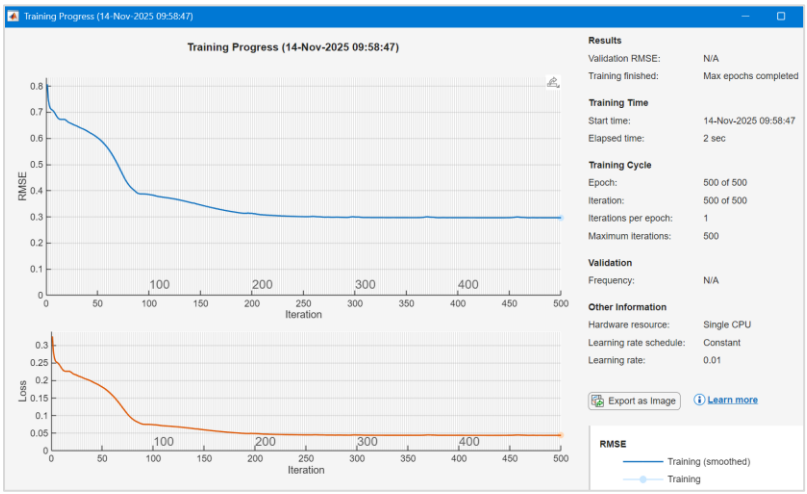


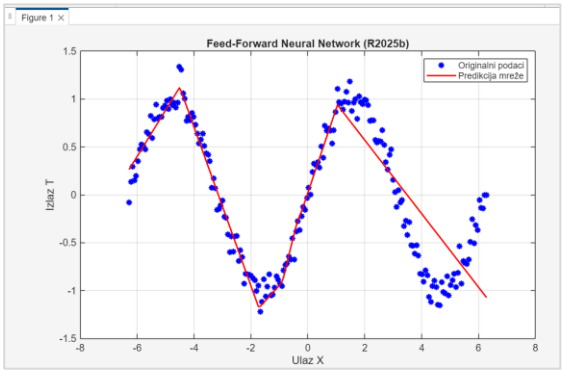


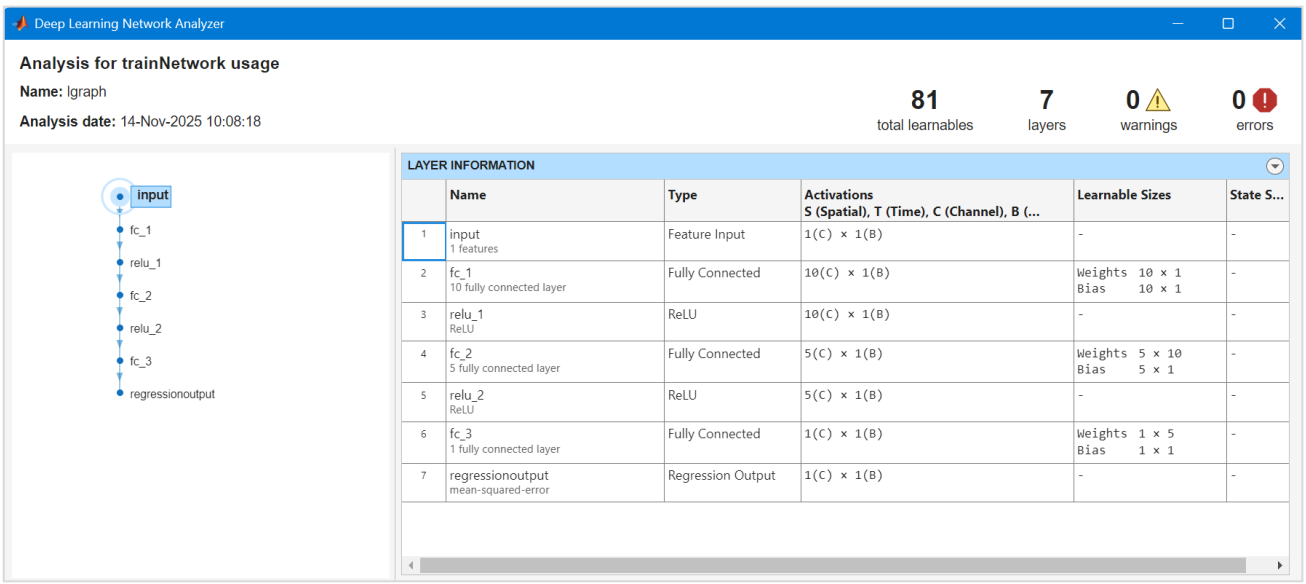
Primjer2 Primjer koristenjem Deep Learning Toolbox-a

```
% Generisanje podataka (regresija)
X = linspace(-2*pi, 2*pi, 200)';
T = sin(X) + 0.1*randn(size(X));

% Definicija slojeva FNN mreže
layers = [
    featureInputLayer(1)
    fullyConnectedLayer(10)
    reluLayer
    fullyConnectedLayer(5)
    reluLayer
    fullyConnectedLayer(1)
    regressionLayer
];

% Kreiranje layerGraph
lgraph = layerGraph(layers);
```

```
% Analiza mreže – direktno prikazati arhitekturu mreže naredbom:
analyzeNetwork(lgraph);

% Opcije treniranja
options = trainingOptions('adam', ...
    'MaxEpochs', 500, ...
    'InitialLearnRate', 0.01, ...
    'Plots','training-progress', ...
    'Verbose', false);

% Treniranje
net = trainNetwork(X, T, layers, options);

% Predikcija
Ypred = predict(net, X);

figure;
plot(X, T, 'b*', X, Ypred, 'r', 'LineWidth',1.5);
legend('Originalni podaci','Predikcija mreže');
grid on;
```







	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (...	Learnable Sizes	State S...
1	input 1 features	Feature Input	1(C) × 1(B)	-	-
2	fc_1 10 fully connected layer	Fully Connected	10(C) × 1(B)	Weights 10 × 1 Bias 10 × 1	-
3	relu_1 ReLU	ReLU	10(C) × 1(B)	-	-
4	fc_2 5 fully connected layer	Fully Connected	5(C) × 1(B)	Weights 5 × 10 Bias 5 × 1	-
5	relu_2 ReLU	ReLU	5(C) × 1(B)	-	-
6	fc_3 1 fully connected layer	Fully Connected	1(C) × 1(B)	Weights 1 × 5 Bias 1 × 1	-
7	regressionoutput mean-squared-error	Regression Output	1(C) × 1(B)	-	-

2) Konvolucijska neuronska mreža - *Convolutional Neural Network (CNN)*

je specijalizirana duboka neuronska mreža dizajnirana da efikasno obrađuje podatke koji imaju prostornu ili vremensku strukturu, kao što su slike (2D), videosignali (3D), vremenske serije (1D) i senzorski podaci. CNN je nastao s idejom da se neuronska mreža može naučiti prepoznati lokalne obrasce (edges, teksture, oblike), umjesto da uči direktno iz sirovih piksela kao FNN.

CNN neuronska mreža sadrži više filtra (kernela) koji klize preko slike i izvlače lokalne značajke. Filter je npr. 3×3 ili 5×5 matrica koja se konvoluira nad ulazom. CNN konvolucije uče ivice, strukture, uglove, teksture, složenije oblike u dubljim slojevima… Učenje se zasniva na gradijentima, filteri se automatski optimiziraju. Bez nelinearnosti CNN se svodi na linearnu transformaciju, ReLU (ili drugi) aktivacijski sloj dodaje nelinearnost, npr. *ReLU(x) = max(0, x)*

Pooling layer - Najčešće max-pooling (2×2), cilj ovog sloja je smanjenje dimenzije slike, smanjenje broja parametara, ekstrakcija dominatnih značajki (maximum pooling) i robustnost na translacije.

Fully-connected slojevi - Na kraju CNN-a nalazi se 1–3 potpuno povezana sloja koji izvode klasifikaciju i regresiju.

Output layer - softmaxLayer + classificationLayer (klasifikacija) i regressionLayer (regresija).

Postupak rada CNN mreže:

1. Ulazna slika prolazi konvolucijske filtere → detekcija lokalnih struktura
2. ReLU čisti negativne vrijednosti
3. Max-pooling smanjuje dimenziju → apstrahuje značajke
4. Više konvolucijskih blokova radi sve apstraktnije reprezentacije
5. Fully-connected slojevi vrše klasifikaciju ili regresiju
6. Mreža se trenira backpropagation algoritmom

CNN se posebno ističe jer automatski uči značajke, za razliku od tradicionalnog računarstva slike koje zahtijeva ručni feature engineering.

Primjer3

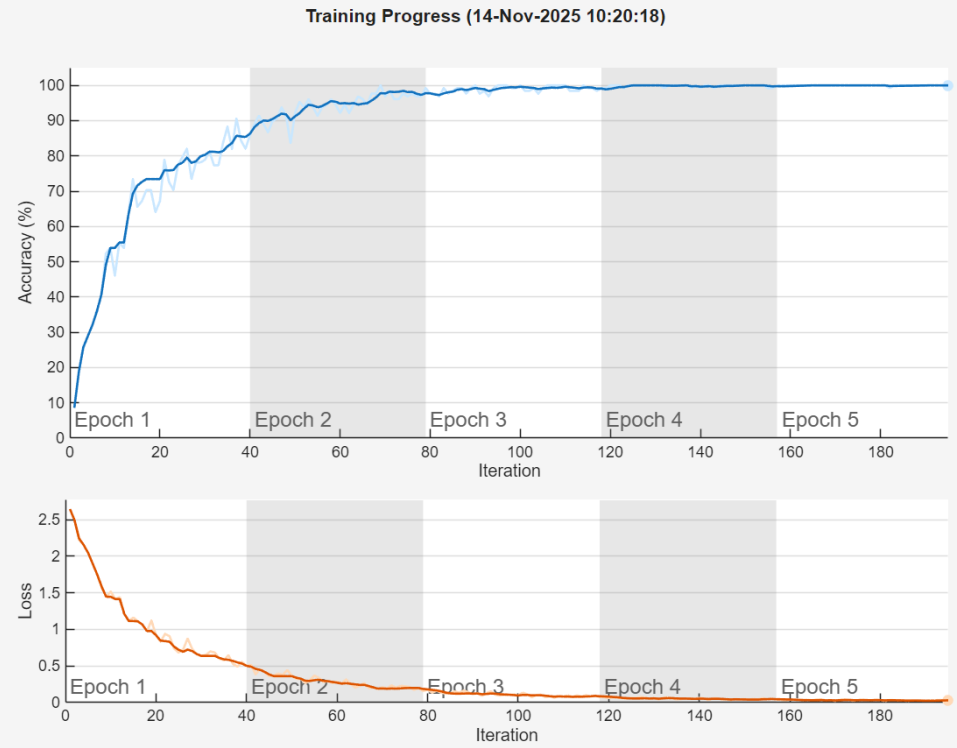
```
% Učitavanje MNIST dataseta
[XTrain, YTrain] = digitTrain4DArrayData;
[XTest,  YTest]  = digitTest4DArrayData;

% Definicija CNN arhitekture
layers = [
    imageInputLayer([28 28 1], 'Normalization','none') Ulazna slika 28×28×1
(sivi nivo)

    convolution2dLayer(3, 16, 'Padding','same') Konvolucijski sloj sa 16
filtera od 3×3 – detektuje lokalne ivice
    batchNormalizationLayer BatchNorm ubrzava treniranje i stabilizira
gradijente
    reluLayer  ReLU uvodi nelinearnost
```

```
    maxPooling2dLayer(2,'Stride',2)  Max-pooling smanjuje dimenziju slike na
14×14.

    convolution2dLayer(3, 32, 'Padding','same')  Drugi konvolucijski blok uči
složenije znake
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3, 64, 'Padding','same') Treći blok radi apstraktnije
reprezentacije (oblici cifri)
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)  Fully-connected dio klasificira cifru 0–9.
    softmaxLayer             Softmax daje vjerovatnoću za svaku cifru.
    classificationLayer
];

% Kreiranje grafičke strukture za analizu
lgraph = layerGraph(layers);

% Analiza mreže u R2025b (ispravno)
analyzeNetwork(lgraph);

% Opcije treniranja
options = trainingOptions('adam', ...
    'MaxEpochs', 5, ...
    'MiniBatchSize', 128, ...
    'Plots', 'training-progress', ...
    'Verbose', false);

% Treniranje CNN mreže
net = trainNetwork(XTrain, YTrain, layers, options);

% Evaluacija
YPred = classify(net, XTest);
accuracy = mean(YPred == YTest);

disp("Tačnost CNN mreže: " + accuracy);
```

Command Window

```
Tačnost CNN mreže: 0.9952
```

Training Progress (14-Nov-2025 10:20:18)



Results

Validation accuracy:	N/A
Training finished:	Max epochs completed

Training Time

Start time:	14-Nov-2025 10:20:18
Elapsed time:	22 sec

Training Cycle

Epoch:	5 of 5
Iteration:	195 of 195
Iterations per epoch:	39
Maximum iterations:	195

Validation

Frequency:	N/A

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

Export as Image Learn more

Accuracy

Training (smoothed)

Training

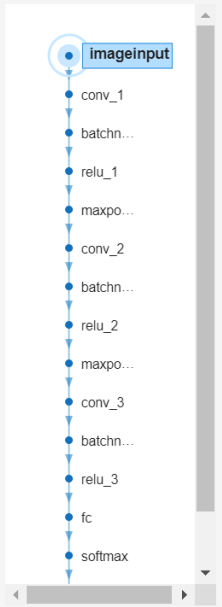
Deep Learning Network Analyzer

Analysis for trainNetwork usage

Name: lgraph

Analysis date: 14-Nov-2025 10:20:15

54.8k	15	0	0
total learnables	layers	warnings	errors



LAYER INFORMATION

	Name	Type	Activations S (Spatial), T (Time), C (Channel), B (…	Learnable Sizes	State Sizes
1	imageinput 28×28×1 images	Image Input	28(S) × 28(S) × 1(C) × 1(B)	-	-
2	conv_1 16 3×3 convolutions with stride [1 1] and…	2-D Convolution	28(S) × 28(S) × 16(C) × 1(B)	Weig… 3 × 3 × 1 … Bias 1 × 1 × 16	-
3	batchnorm_1 Batch normalization	Batch Normalization	28(S) × 28(S) × 16(C) × 1(B)	Offset 1 × 1 × 16 Scale 1 × 1 × 16	TrainedMean 0 … TrainedVar… 0 …
4	relu_1 ReLU	ReLU	28(S) × 28(S) × 16(C) × 1(B)	-	-
5	maxpool_1 2×2 max pooling with stride [2 2] and pa…	2-D Max Pooling	14(S) × 14(S) × 16(C) × 1(B)	-	-
6	conv_2 32 3×3 convolutions with stride [1 1] and…	2-D Convolution	14(S) × 14(S) × 32(C) × 1(B)	Weig… 3 × 3 × 16… Bias 1 × 1 × 32	-
7	batchnorm_2 Batch normalization	Batch Normalization	14(S) × 14(S) × 32(C) × 1(B)	Offset 1 × 1 × 32 Scale 1 × 1 × 32	TrainedMean 0 … TrainedVar… 0 …
8	relu_2 ReLU	ReLU	14(S) × 14(S) × 32(C) × 1(B)	-	-
9	maxpool_2 2×2 max pooling with stride [2 2] and pa…	2-D Max Pooling	7(S) × 7(S) × 32(C) × 1(B)	-	-
10	conv_3 64 3×3 convolutions with stride [1 1] and…	2-D Convolution	7(S) × 7(S) × 64(C) × 1(B)	Weig… 3 × 3 × 32… Bias 1 × 1 × 64	-
11	batchnorm_3 Batch normalization	Batch Normalization	7(S) × 7(S) × 64(C) × 1(B)	Offset 1 × 1 × 64 Scale 1 × 1 × 64	TrainedMean 0 … TrainedVar… 0 …

3) Generativna suparnička neuronska mreža – *Generative Adversarial Neural Network (GAN)*

- sastoji se od dvije neuronske mreže koje se treniraju istovremeno, ali sa suprotnim ciljevima:

1. *Generator (G)* – stvara nove primjere (slike, podatke) na osnovu slučajnog ulaza (latentnog prostora).
2. *Discriminator (D*) – pokušava razlikovati stvarne podatke od lažnih koje generiše G.

Obje mreže se takmiče u igri nulte sume, tj. generator nastoji prevariti discriminator, dok discriminator pokušava prepoznati lažne podatke. Rezultat je generator koji s vremenom postaje izuzetno dobar u stvaranju podataka sličnih originalnim.

Način rada GAN neuronske mreže:

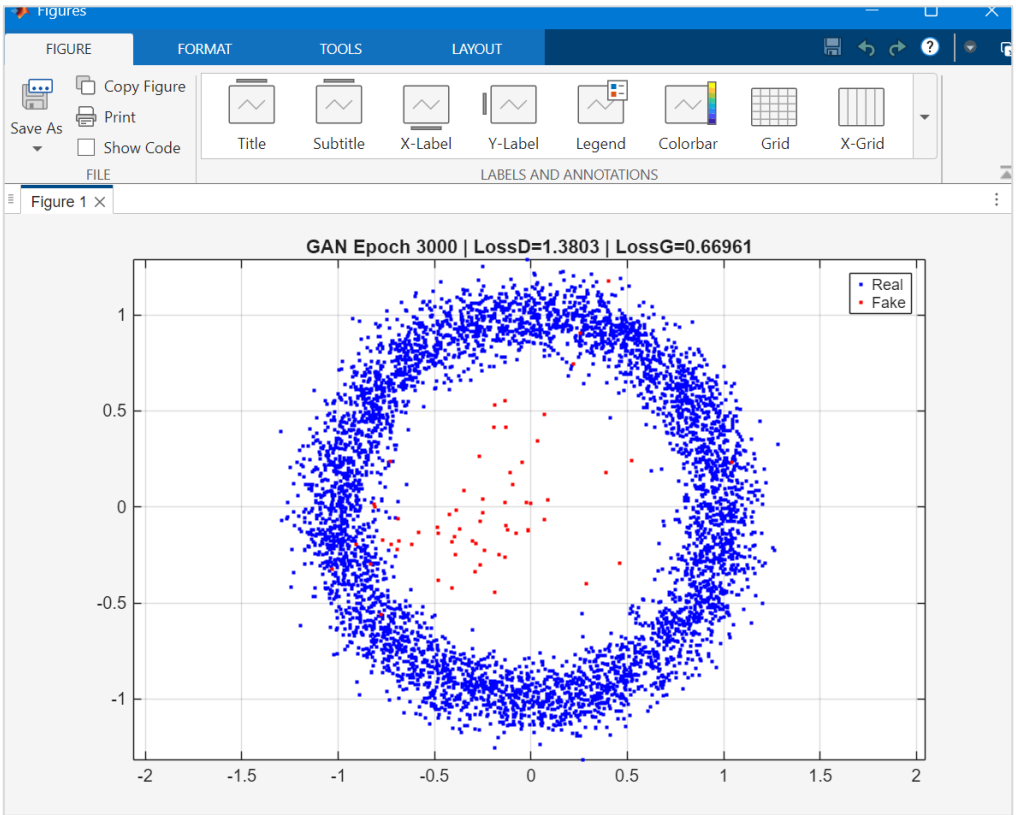
Uzimaju se stvarni podaci (npr. slike), zatim generator uzima random vektor z (latent vector) i iz njega pokušava generisati lažni primjer, discriminator dobija stvarne primjere i označi ih kao *real* i generisane primjere i označi ih kao *fake.* Nakon toga generator i diskriminator se treniraju naizmjenično, discriminator maksimalno pokušava povećati vjerovatnoću ispravne klasifikacije, dok generator minimizuje sposobnost discriminatora da otkrije lažne primjere.

Primjer4

```
%% ============================================================
%      GAN primjer – MATLAB R2025b
%      Generiše 2D tačke u obliku kruga
%      Fully functional sa dlfeval za gradient
% ============================================================

clear; clc; close all;

%% 1) Dataset: 2D tačke u obliku kruga (REAL DATA)
numReal = 5000;
theta = 2*pi*rand(numReal,1);
r = 1 + 0.1*randn(numReal,1);
XReal = [r.*cos(theta), r.*sin(theta)];   % 2D kružnica

%% 2) DEFINICIJA GENERATORA
latentDim = 10;  % dimenzija latentnog vektora

layersG = [
    featureInputLayer(latentDim, 'Normalization','none')
    fullyConnectedLayer(32)
    reluLayer
    fullyConnectedLayer(16)
    reluLayer
    fullyConnectedLayer(2)         % generisana 2D tačka
];

lgraphG = layerGraph(layersG);
dlnetG = dlnetwork(lgraphG);
```

```
%% 3) DEFINICIJA DISKRIMINATORA
layersD = [
    featureInputLayer(2, 'Normalization','none')
    fullyConnectedLayer(32)
    leakyReluLayer(0.2)
    dropoutLayer(0.3)

    fullyConnectedLayer(16)
    leakyReluLayer(0.2)

    fullyConnectedLayer(1)
    sigmoidLayer                    % izlaz = vjerovatnoća real/fake
];

lgraphD = layerGraph(layersD);
dlnetD = dlnetwork(lgraphD);

%% 4) OPCIJE TRENIRANJA
numEpochs = 3000;
learningRate = 0.0005;
miniBatchSize = 64;

%% 5) TRENIRANJE GAN-A
figure;

for epoch = 1:numEpochs

    % === MINI-BATCH REALNIH PODATAKA ===
    idx = randperm(numReal, miniBatchSize);
    XBatch = XReal(idx,:);
    dlXReal = dlarray(single(XBatch'), 'CB');  % C = feature, B = batch

    % === GENERISANJE LATENT VEKtORA ===
    Z = randn(latentDim, miniBatchSize, 'single');
    dlZ = dlarray(Z, 'CB');  % C = latentDim, B = batch

    % === Izračun gradijenata sa dlfeval ===
    [gradientsD, gradientsG, lossD, lossG] = dlfeval(@modelGradients, dlnetD,
dlnetG, dlXReal, dlZ);

    % === Update mreža ===
    dlnetD = dlupdate(@(p,g) p - learningRate*g, dlnetD, gradientsD);
    dlnetG = dlupdate(@(p,g) p - learningRate*g, dlnetG, gradientsG);

    % === PRIKAZ PROGRESIJE ===
    if mod(epoch, 100) == 0
        dlXFake = forward(dlnetG, dlZ);
        fake = extractdata(dlXFake)';
        plot(XReal(:,1), XReal(:,2), 'b.'); hold on;
        plot(fake(:,1), fake(:,2), 'r.');
        legend('Real','Fake');
        title("GAN Epoch " + epoch + ...
              " | LossD=" + gather(extractdata(lossD)) + ...
              " | LossG=" + gather(extractdata(lossG)));
```

```
        grid on; axis equal;
        drawnow;
        hold off;
    end
end

disp('Trening GAN-a je završen.');

%% ============================================================
%  Funkcija za gradijente generatora i diskriminatora
%% ============================================================
function [gradientsD, gradientsG, lossD, lossG] = modelGradients(dlnetD,
dlnetG, dlXReal, dlZ)
    % Forward pass generatora
    dlXFake = forward(dlnetG, dlZ);

    % Forward pass discriminatora
    dReal = forward(dlnetD, dlXReal);
    dFake = forward(dlnetD, dlXFake);

    % Loss funkcije
    lossD = -mean(log(dReal + 1e-8) + log(1 - dFake + 1e-8));
    lossG = -mean(log(dFake + 1e-8));

    % Gradijenti
    gradientsD = dlgradient(lossD, dlnetD.Learnables);
    gradientsG = dlgradient(lossG, dlnetG.Learnables);
end
```



4) Rekurentna neuronska mreža (RNN) – *Recurrent Neural Network (GAN)*

- je vrsta neuronske mreže dizajnirana za obradu sekvenci podataka, vremenskih nizova, govora, signalnih serija, tekstova itd. RNN ima memoriju. Svaki izlaz zavisi od trenutnog ulaza i od stanja iz prethodnog vremenskog koraka (hidden state). Matematički izraz za RNN:

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$
$$y_t = W_y h_t + c$$

Ova povratna petlja daje mreži sposobnost da:

- uči vremenske obrasce
- modelira dinamiku sistema
- radi predikciju sekvenci

Klasična RNN predstavlja osnovni model za obradu sekvencijalnih podataka, ali posjeduje značajna ograničenja kada se radi o učenju dugoročnih zavisnosti unutar vremenskih serija. Najveći izazov je takozvani problem nestajućeg gradijenta (vanishing gradient). Tokom treniranja, RNN propagira grešku kroz veliki broj vremenskih koraka. Zbog višestrukog množenja malih vrijednosti gradijenta, informacija o udaljenim događajima u sekvenci se brzo gubi, pa mreža postaje nesposobna da nauči dugoročne obrasce. Ovo rezultira:

- slabom memorijom mreže,
- nemogućnošću učenja dugih sekvenci,
- otežanim treniranjem i sporom konvergencijom.

Da bi se riješio ovaj problem, razvijene su naprednije arhitekture LSTM (Long Short-Term Memory) i GRU (Gated Recurrent Unit), koje koriste unutrašnje kontrolne mehanizme – „vrata" (gates) – za regulaciju protoka informacija kroz vrijeme.

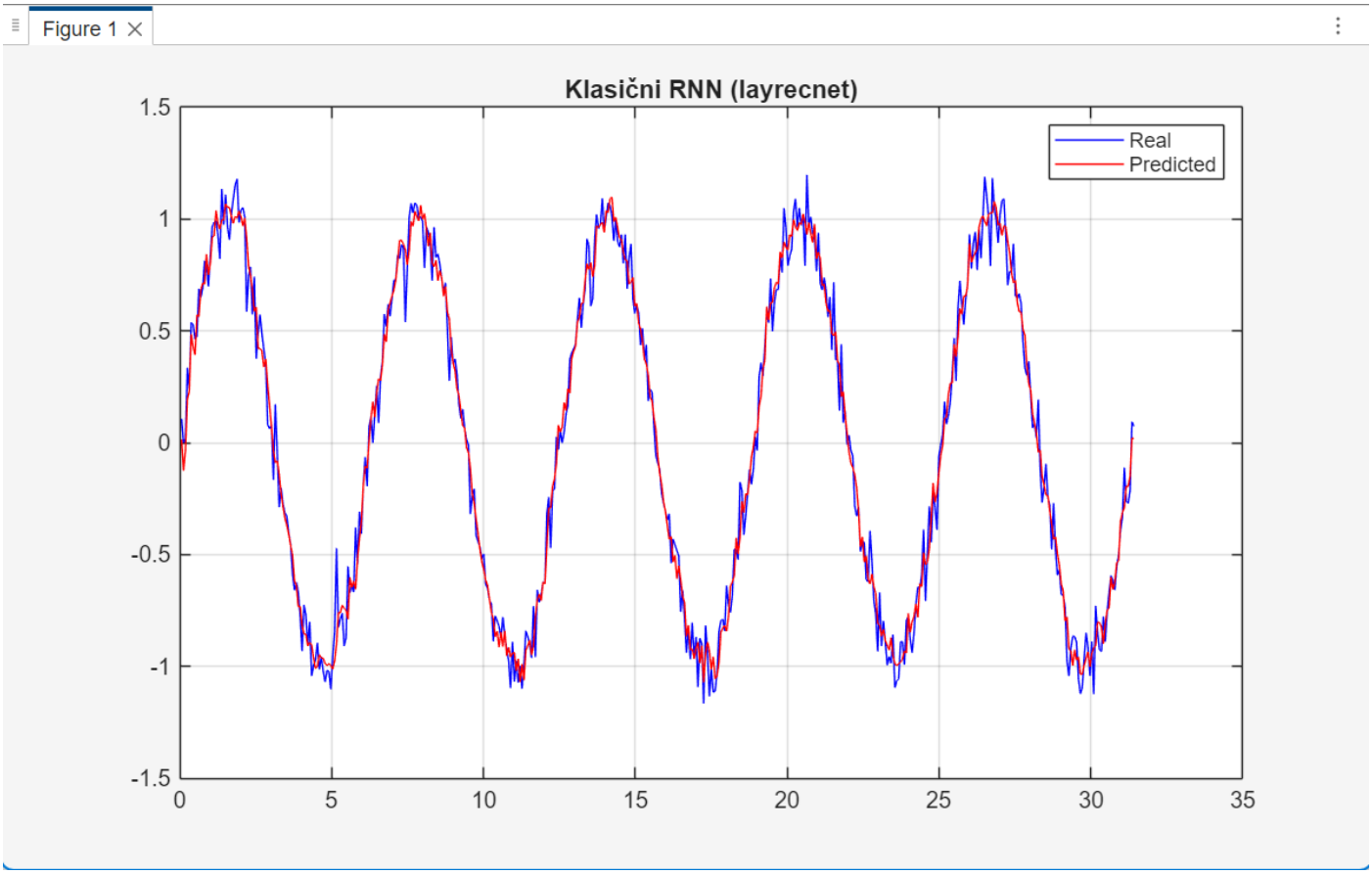
LSTM mreža uvodi strukture nazvane *ćelijsko stanje* i tri vrste vrata (input, forget i output gates). Ova vrata omogućavaju mreži da selektivno pamti ili zaboravlja informacije kroz duge vremenske intervale. Zahvaljujući tome, LSTM značajno ublažava problem nestajućeg gradijenta i omogućava učenje kompleksnih dugoročnih zavisnosti u sekvencama.

GRU je pojednostavljena verzija LSTM arhitekture, ali sa sličnim performansama. Koristi dva vrata (reset i update gate) i nema posebno ćelijsko stanje, što je čini bržom i računski efikasnijom u odnosu na LSTM. GRU daje odlične rezultate u sistemima gdje je potreban kompromis između tačnosti i brzine treniranja.

Primjer5

```
clear;
clc;
close all;

t = linspace(0,10*pi,500);
Y = sin(t)+0.1*randn(size(t));
```

```
% layrecnet očekuje sekvence u cell-u
Xseq = num2cell(Y(1:end-1));
Tseq = num2cell(Y(2:end));

% Kreiraj rekuretntnu mrežu (jedan skriveni sloj, 20 neurona)
net = layrecnet(1:1,20);   % delay 1, 20 neurona
net.trainFcn = 'trainlm';
net.divideFcn = 'dividetrain';
net = train(net, Xseq, Tseq);
Ypred_seq = net(Xseq);
Ypred = cell2mat(Ypred_seq);

figure;
plot(t(2:end), Y(2:end),'b'); hold on;
plot(t(2:end), Ypred,'r');
legend('Real','Predicted');
title('Klasični RNN (layrecnet)');
grid on;
```



5) Transformeri

Transformeri su arhitektura zasnovana na mehanizmu self-attention koja zamjenjuje rekurentne i konvolucijske pristupe u zadacima sekvenci (NLP) i, adaptacijom (ViT), u računalnom vidu. Osnovna ideja: model uči odnose *svakog* elementa sekvence sa *svim* ostalim elementima kroz težinske pozorne mehanizme, paralelizirano.

NLP (Natural Language Processin)*Obrada prirodnog jezika* je oblast umjetne inteligencije koja se bavi razumijevanjem i generisanjem ljudskog jezika: analiza teksta, prevođenje, klasifikacija emocija, chatbotovi, sažimanje teksta, detekcija namjere, entiteta itd.

ViT (Vision Transformer)*Vizijski transformer* je arhitektura transformera prilagođena za obradu slika. Umjesto riječi (tokena), slika se dijeli na male blokove, *patch-e*. Svaki patch se tretira kao token u transformer arhitekturi. ViT se koristi za: klasifikaciju slika, detekciju objekata, segmentaciju, generisanje slika itd.

Ključni elementi transformesa:

- Self-attention (scaled dot-product) Za ulazne reprezentacije Xtransformer računa upite (Q), ključeve (K) i vrijednosti (V):

$$Q = XW_Q, K = XW_K, V = XW_V$$

Attention izračun:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$$

gdje $d_k$je dimenzija ključeva.

- Multi-head attention: paralelno izvršavanje više self-attention "glava", koje se potom spoje i projektuju — omogućava modelu da uči različite vrste odnosa.
- Feed-forward block: nakon attention bloka ide pozicioni-nezavisan MLP (obično dva FC sloja sa ReLU), sa residual vezama i layer normalization.
- Positional encoding: budući da attention ne nosi informaciju o poretku elemenata, dodaju se položajne kodove (sinusoidalne ili učene) $PE(pos)$na ulazne tokene. (MathWorks ima sinusoidalPositionEncodingLayer)