

ניסוי 56 – למידה עמוקה – דו"ח הכנה 2

ליאור וובצ'וק (207584715), אלמוג אדטו (318782976)

שאלה 1:

- א. $c_{out} = d$
 $w_{out} = w_{in} + 1 - k_w$
 $h_{out} = h_{in} + 1 - k_h$
- ב. $n = (k_h * k_w * c_{in} + 1) \cdot d$
- ג. $p = (c_{out} * w_{out} * h_{out}) * (c_{in} * w_{in} * h_{in} + 1)$

שאלה 2:

- א. $v_0 = 0$
 $x_0 = -1$
 $f = x^2, \quad \nabla f = 2x$
 $v_1 = 0.9 * 0 + 2 * (-1) = -2$
 $x_1 = -1 - 0.01 * (-2) = -0.98$
 $v_2 = 0.9 * (-2) + 2 * (-0.98) = -3.76$
 $x_2 = -0.98 - 0.01 * (-3.76) = -0.9424$
 $v_3 = 0.9 * (-3.76) + 2 * (-0.9424) = -5.2688$
 $x_3 = -0.9424 - 0.01 * (-5.2688) = -0.889712$
- ב. $v_0 = 0$
 $x_0 = -1$
 $f = x^2, \quad \nabla f = 2x$
 $v_1 = 0.9 * 0 + 2 * (-1) = -2$
 $x_1 = -1 - 2 * (-2) = 3$
 $v_2 = 0.9 * (-2) + 2 * (3) = 4.2$
 $x_2 = 3 - 2 * 4.2 = -5.4$
 $v_3 = 0.9 * 4.2 + 2 * (-5.4) = -7.02$
 $x_3 = -5.4 - 2 * (-7.02) = 8.64$

שאלה 3:

רגולריזציית L2 על המשקולות מגדילה את פונקציית ה-LOSS ככל שהמשקולות גדלות ולכן "מעדיפה" רשתות פשוטות יותר בעלות משקלים נמוכים. עבור $\alpha \rightarrow 0$ תקטן מאוד השפעת הרגולריזציה על פונקציית ה-LOSS ולכן כמעט ולא תהיה לה השפעה על קביעת המשקלים. עבור $\alpha \rightarrow \infty$ פונקציית ה-LOSS תהיה גדולה מאוד עבור כל משקל ששונה מ-0 ולכן הרשת לא תתאמן וכל המשקלים ישאפו ל-0.

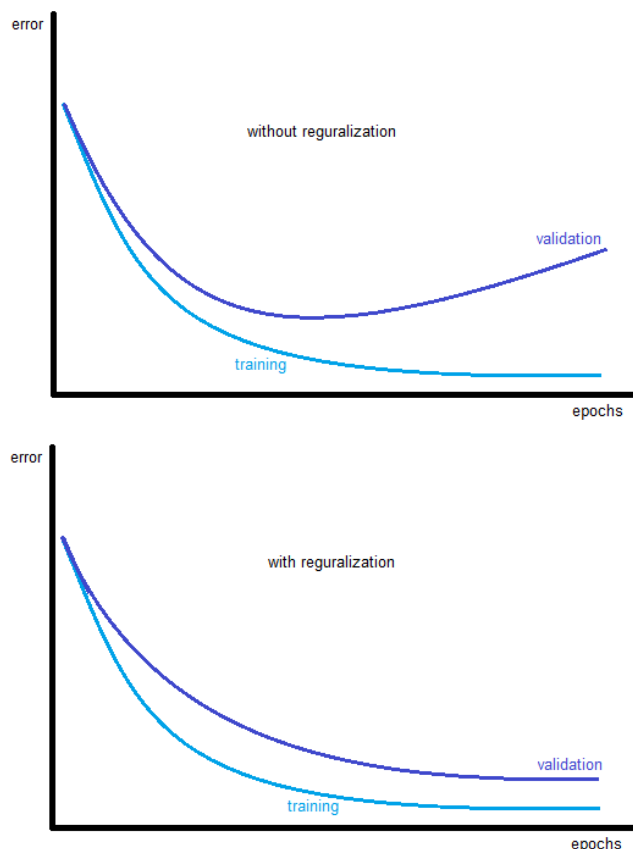
שאלה 4:

1. ישנם משמעותית פחות משקלים ולכן נצרך פחות זיכרון.
2. כל צומת ברשת יהיה תלוי בפחות צמדים הקודמים לו ולכן זמן החישוב יקטן.
3. בשיטת קונבולוציה נעבור עם כל המשקלים בצורה סימטרית על כל חלקי התמונה ולכן נקבל אינוריאנטיות להזזות.

שאלה 5:

1. הזזה: בעד - נצטרך לדעת לזהות אובייקט בכל מיקום בתמונה.
2. שיקוף: בעד - את רוב האובייקטים נרצה לזהות מכל זווית.
- נגד - אינו מתאים לכל האובייקטים, למשל לזיהוי ספרות.
3. סיבוב: בעד - את רוב האובייקטים נרצה לזהות מכל זווית.
- נגד - אינו מתאים לכל האובייקטים, למשל לזיהוי ספרות.
4. שינוי גודל: בעד - את רוב האובייקטים נרצה לזהות בכל גודל בתמונה.
- נגד - נוכל לחשוב על רשתות בהן הרשת תהיה מאומנת על גודל ספציפי של אובייקט בתמונה, למשל חישוב קלוריות במנה המונחת במרחק ספציפי מעדשת המצלמה.
5. שינוי בהירות: בעד - את רוב האובייקטים נרצה לזהות בכל בהירות.
- נגד - נוכל לחשוב על רשתות בהן הרשת תהיה מאומנת לזהות ולהבדיל בין סוגים שונים של בע"ח שההבדל ביניהם הוא בצבע.
6. הוספת רעש: בעד - נרצה לאמן את הרשת להתמודד עם תמונות לא אופטימליות.
- נגד - נוכל לחשוב על רשתות בהן הרשת תהיה רגישה מאוד (בהגדרתה) לאיכות התמונה ולשינויים, למשל זיהוי מחלה ברקמה אנושית.

שאלה 6:



ברשתות בהן אין רגולריזציה, המשקלים נוטים להגיע למצב *over-fitting*, כלומר מצב בו הרשת מתאימה בצורה כמעט מושלמת לסט האימון אך מתרחקת מהתאמה לסט הכללי של הדוגמאות. רגולריזציה מונעת זאת בכך שהיא מעודדת מודלים פשוטים ולכן הרשת לא יכולה להתאים את עצמה יתר על המידה לסט האימון ולהתרחק מהתאמה למקרה הכללי.

שאלה 7:

- א. Conv 5X5, no pad, stride=1, 10 kernels
ReLU
Conv 3X3, no pad, stride=3, 16 kernels
Max pulling 2X2
ReLU
Conv 3X3, pad=1, stride=1, 20 kernels
ReLU
Conv 7X7, no pad, stride=5, 15 kernels
ReLU
Conv 3X3, pad=1, stride=1, 3 kernels
ReLU
fc layer
Softmax
LOSS – nll

- ב. $n_{parameters} =$

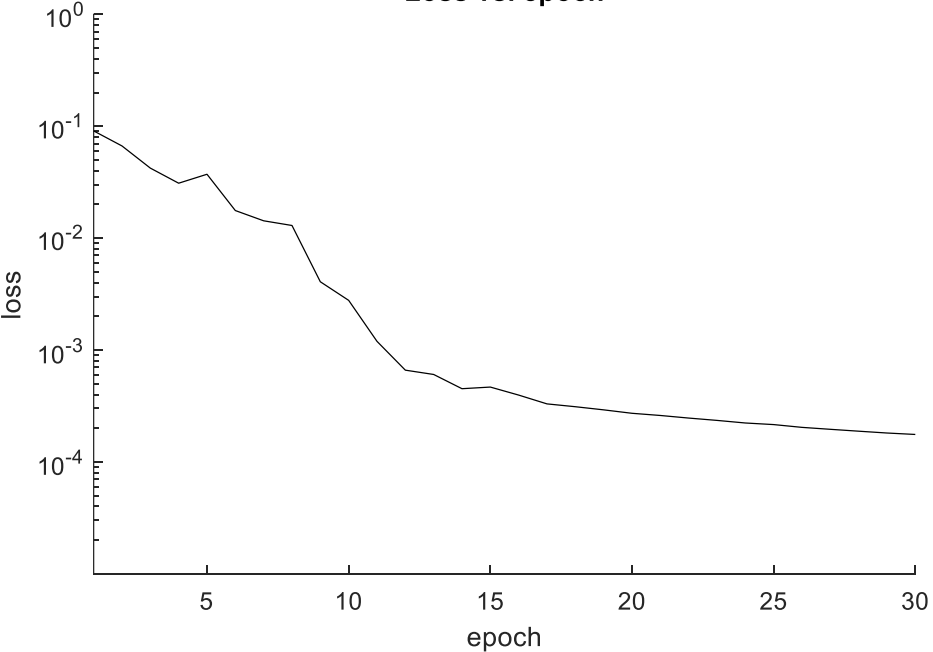
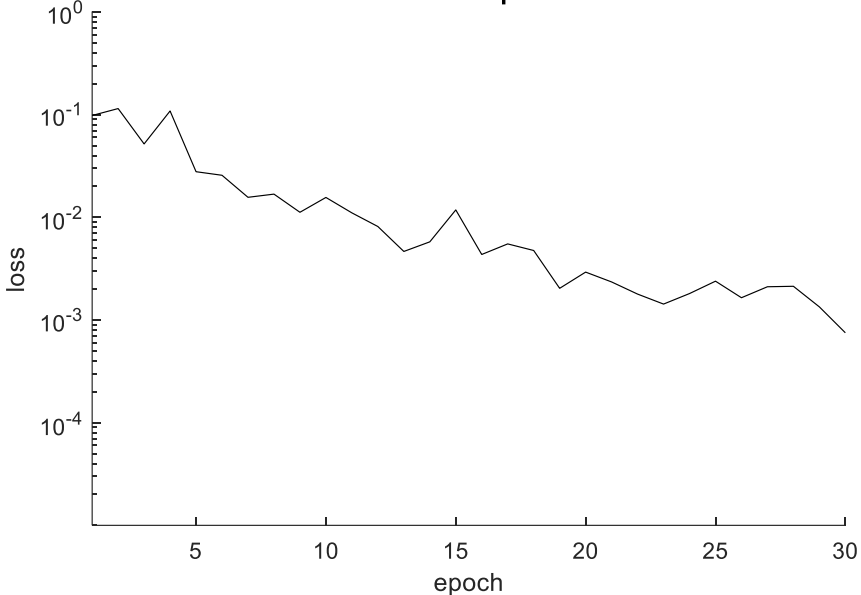
$$\begin{aligned} & (5 * 5 * 3 + 1) * 10 + \\ & (3 * 3 * 10 + 1) * 16 + \\ & (3 * 3 * 16 + 1) * 20 + \\ & (7 * 7 * 20 + 1) * 15 + \\ & (3 * 3 * 15 + 1) * 3 + \\ & (8 * 8 * 3 + 1) * 7 = 21590 \end{aligned}$$

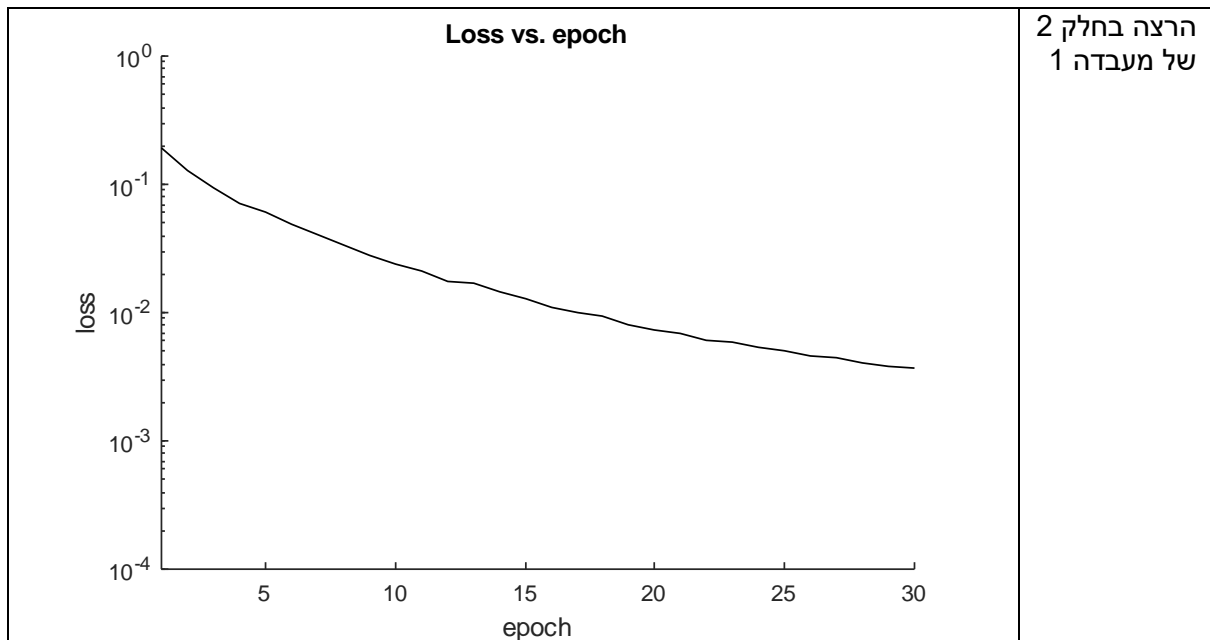
5 השורות הראשונות הן עבור שכבות הקונבולוציה והשורה האחרונה היא עבור שורת הfully_connected.

- ג. על מנת לעבד את התמונה הזו נצטרך לבצע שינויים ברשת, ולכן לא ניתן לעבד תמונה זו בעזרת הרשת.

שינויים שנוכל לבצע הם: הוספת Padding בשכבות הלינאריות, שינוי מימדי המסגרים.

שאלה 8:

<p>Final train error: 0.00% Validation error: 1.72%</p> <p>Loss vs. epoch</p>  <p>loss</p> <p>epoch</p> <table border="1"><caption>Approximate data for Loss vs. epoch (dropout)</caption><thead><tr><th>epoch</th><th>loss</th></tr></thead><tbody><tr><td>1</td><td>1.0e-1</td></tr><tr><td>5</td><td>4.0e-2</td></tr><tr><td>10</td><td>1.0e-2</td></tr><tr><td>15</td><td>5.0e-3</td></tr><tr><td>20</td><td>3.0e-3</td></tr><tr><td>25</td><td>2.5e-3</td></tr><tr><td>30</td><td>2.0e-3</td></tr></tbody></table>	epoch	loss	1	1.0e-1	5	4.0e-2	10	1.0e-2	15	5.0e-3	20	3.0e-3	25	2.5e-3	30	2.0e-3	<p>הרצה ללא dropout</p>
epoch	loss																
1	1.0e-1																
5	4.0e-2																
10	1.0e-2																
15	5.0e-3																
20	3.0e-3																
25	2.5e-3																
30	2.0e-3																
<p>Final train error: 0.004% Validation error: 1.60%</p> <p>Loss vs. epoch</p>  <p>loss</p> <p>epoch</p> <table border="1"><caption>Approximate data for Loss vs. epoch (10% dropout)</caption><thead><tr><th>epoch</th><th>loss</th></tr></thead><tbody><tr><td>1</td><td>1.0e-1</td></tr><tr><td>5</td><td>3.0e-2</td></tr><tr><td>10</td><td>1.5e-2</td></tr><tr><td>15</td><td>1.0e-2</td></tr><tr><td>20</td><td>3.0e-3</td></tr><tr><td>25</td><td>2.0e-3</td></tr><tr><td>30</td><td>1.0e-3</td></tr></tbody></table>	epoch	loss	1	1.0e-1	5	3.0e-2	10	1.5e-2	15	1.0e-2	20	3.0e-3	25	2.0e-3	30	1.0e-3	<p>הרצה עם dropout 10%</p>
epoch	loss																
1	1.0e-1																
5	3.0e-2																
10	1.5e-2																
15	1.0e-2																
20	3.0e-3																
25	2.0e-3																
30	1.0e-3																



ניתן לראות בבירור שה-Loss קטן הרבה יותר בשתי ההרצות שהוספנו להן את השכבות החדשות. בנוסף לכך, ניתן לראות שברשת ללא dropout שגיאת ה-final train error היא 0 שכן הרשת התאמנה והתאימה את עצמה בצורה מושלמת לסט ה-train. כאשר הפעלנו את ה-dropout הרשת הצליחה פחות בלהתאים את עצמה בדיוק לסט ה-train אך התאימה יותר למקרה הכללי, שכן שגיאת ה-validation שלה קטנה יותר. מכאן ניתן להסיק, ששיטת ה-dropout מונעת מהרשת להתאים את עצמה יתר על המידה לסט ה-train (overfitting) אך עוזרת לאמן את הרשת להצליח במקרה הכללי, בסט ה-validation.

```

function z_out = relu_forward(z_in)
% z_out = relu_forward(z_in)
% This function performs elementwise ReLU function on the input z_in
%   Input:
%       z_in - input of the ReLU (nFeatures x nSamples)
%
%   Output:
%       z_out - output of the ReLU (nFeatures x nSamples)
%
% Last modified by Rotem Mulayoff 15/10/20

% TODO: compute the ReLU function.
z_out = max(0, z_in);

end

```

```

function dL_dz_in = relu_backward(z_in, dL_dz_out)
% dL_dz_in = relu_backward(z_in, dL_dz_out)
% This function computes the gradient of the elementwise ReLU function
%   Inputs:
%       z_in - the input of the ReLU (nFeatures x nSamples)
%       dL_dz_out - gradient of the loss w.r.t. output (nFeatures x nSamples)
%
%   Output:
%       dL_dz_in - gradient of the loss w.r.t. the inputs of the ReLU (nFeatures x nSamples)
%
% Last modified by Rotem Mulayoff 15/10/20

% TODO: Compute the gradient of the ReLU function w.r.t. it's inputs
dL_dz_in = dL_dz_out.*(0*(z_in<=0)+1*(z_in>0));

end

```

```

function [z_out, mask] = dropout_forward(z_in, p, active_flag)
% [z_out, mask] = dropout_forward(z_in, p, active_flag)
% This function implements a dropout layer, applied on the input z_in
%   Input:
%       z_in - input of the dropout layer (nFeatures x nSamples)
%       p - probability for getting zero at the output for each neuron (scalar)
%       active_flag - binary flag to active the dropout layer
%                   (1 - on for learning mode, 0 - off for validation, test and inference)
%
%   Output:
%       z_out - output of the dropout layer (nFeatures x nSamples)
%       mask - a binary matrix of the dropout (nFeatures x nSamples)
%
% Last modified by Rotem Mulayoff 15/10/20

% TODO: implement the dropout layer.
if active_flag && p>0
    mask = double(rand(size(z_in)) > p);
    z_out = z_in.*mask;
else
    mask = ones(size(z_in));
    z_out = z_in;
end

end

```

```

function dL_dz_in = dropout_backward(mask, dL_dz_out)
% dL_dz_in = dropout_backward(mask, dL_dz_out)
% This function computes the gradient of the dropout layer
% Inputs:
%     mask - a binary matrix of the dropout given by the forward function (nFeatures x nSamples)
%     dL_dz_out - gradient of the loss w.r.t. output (nFeatures x nSamples)
%
% Output:
%     dL_dz_in - gradient of the loss w.r.t. the inputs of the dropout layer (nFeatures x nSamples)
%
% Last modified by Rotem Mulayoff 15/10/20

% TODO: Compute the gradient of the dropout layer w.r.t. it's inputs
dL_dz_in = mask.*dL_dz_out;

end

```

```

Editor - C:\Users\almoga\Downloads\DL-prep-lab2\trainTwoLayerPerceptron_prep2.m
+5 dropout_backward.m dropout_forward.m affine_backward.m relu_backward.m trainTwoLayerPerceptron_prep2.m mean.m mu
54 % TODO: Activate dropout in training
55 dropFlag = 1;
56
57 % Select a batch from the training set
58 startInd = (n-1)*batchSize+1;
59 stopInd = min(n*batchSize,trainingSetSize);
60 x = X(:,randInds(startInd:stopInd));
61 y = Y(:,randInds(startInd:stopInd));
62
63 % TODO: complete a training pass of the network
64 [xw1,sxw1,sxw1_drop,mask,sxw1w2,~,~] = forwardPass_prep2(x, w1, b1, w2, b2, y, p, dropFlag);
65 [dL_dw1, dL_db1, dL_dw2, dL_db2] = backwardPass_prep2(x, w1, b1, w2, b2, y, xw1, sxw1, sxw1w2, mask, sxw1_drop);
66
67 w2 = w2 - learningRate*dL_dw2;
68 b2 = b2 - learningRate*dL_db2;
69 w1 = w1 - learningRate*dL_dw1;
70 b1 = b1 - learningRate*dL_db1;
71 end
72
73 % TODO: deactivate dropout in evaluation
74 dropFlag = 0;
75
76 % TODO: calculate the the loss average on ALL examples in the dataset
77 [~,~,~,~,ssxw1w2,loss] = forwardPass_prep2(X, w1, b1, w2, b2, Y, p, dropFlag);
78
79 % plot the loss function
80 addpoints(h, t, loss);
81 drawnow
82

```

```

Editor - C:\Users\almoga\Downloads\DL-prep-lab2\trainTwoLayerPerceptron_prep2.m
+5 dropout_backward.m dropout_forward.m affine_backward.m relu_backward.m trainTwoLayerPerceptron_prep2.m
66
67     w2 = w2 - learningRate*dL_dw2;
68     b2 = b2 - learningRate*dL_db2;
69     w1 = w1 - learningRate*dL_dw1;
70     b1 = b1 - learningRate*dL_db1;
71 end
72
73 % TODO: deactivate dropout in evaluation
74 dropFlag = 0;
75
76 % TODO: calculate the the loss average on ALL examples in the dataset
77 [~,~,~,~,ssxw1w2,loss] = forwardPass_prep2(X, w1, b1, w2, b2, Y, p, dropFlag);
78
79 % plot the loss function
80 addpoints(h, t, loss);
81 drawnow
82
83 Y_hat = zeros(size(ssxw1w2));
84 [~,max_inds] = max(ssxw1w2);
85 linearInd = sub2ind(size(ssxw1w2), max_inds, 1:trainingSetSize);
86 Y_hat(linearInd) = 1;
87
88 % TODO: calculate the classification error
89 error = mean(mean(Y_hat~=Y))*100;
90
91 fprintf('Train error = %2.2f%%\n', error);
92 end
93
94 end
95

```

```

function [xw1,sxw1,sxw1_drop,mask,sxw1w2,ssxw1w2,loss] = forwardPass_prep2(x, w1, b1, w2, b2, y, p, dropout_flag)
% [xw1,sxw1,sxw1_drop,mask,sxw1w2,ssxw1w2,loss] = forwardPass(x, w1, b1, w2, b2, y, p, dropout_flag)
% This function propagates the input vector through the network.
% Last modified by Ori Bryt 11/11/20

% TODO: Complete the forward pass.
xw1 = affine_forward(x, w1, b1);
sxw1 = relu_forward(xw1);
[sxw1_drop,mask] = dropout_forward(sxw1, p, dropout_flag);
sxw1w2 = affine_forward(sxw1_drop, w2, b2);
ssxw1w2 = softmax_forward(sxw1w2);
loss = multi_nll_loss_forward(ssxw1w2, y);

end

function [dL_dw1, dL_db1, dL_dw2, dL_db2] = backwardPass_prep2(x, w1, b1, w2, b2, y, xw1, sxw1, sxw1w2, mask, sxw1_drop)
% [dL_dw1, dL_db1, dL_dw2, dL_db2] = backwardPass(x, w1, b1, w2, b2, y, xw1, sxw1, sxw1w2)
% This function computes the gradients for all layers using backpropagation technique.
% Last modified by Ori Bryt 11/11/20

% TODO: Complete the backward pass.
dL_dsxw1w2 = nll_and_softmax_backward(sxw1w2, y);
[dL_dsxw1_drop, dL_dw2, dL_db2] = affine_backward(sxw1_drop, w2, b2, dL_dsxw1w2);
dL_dsxw1_drop = dropout_backward(mask, dL_dsxw1_drop);
dL_dxw1 = relu_backward(xw1, dL_dsxw1);
[~, dL_dw1, dL_db1] = affine_backward(x, w1, b1, dL_dxw1);

end

```



```
Editor - C:\Users\almoqa\Downloads\DL-prep-lab2\MNISTClassification_prep2.m
backwardPass_prep2.m forwardPass_prep2.m MNISTClassification_prep2.m dropout_backward.m dropout_forward.m affine_backward.m relu_backward.m trainTwoLayerPerceptron_prep2.m
42 hiddenUnits = 200;
43
44 % Choose appropriate parameters.
45 learningRate = 0.3;
46
47 % Choose number of epochs
48 epochs = 30;
49
50 % Choose batch size
51 batchSize = 64;
52
53 % TODO: Set dropout probability (0 when not using dropout)
54 p = 0.1;
55
56 fprintf('Train two layer perceptron with %d hidden units.\n', hiddenUnits);
57 fprintf('Learning rate: %2.2f\n', learningRate);
58 fprintf('Batch size: %d\n', batchSize);
59
60 % Network training
61 [hiddenWeights, outputWeights, b1, b2, error] = trainTwoLayerPerceptron_prep2(hiddenUnits, inputValuesTrain, targetValuesTrain, epochs, learningRate, batchSize, p);
62
63 fprintf('Final train error: %2.2f%%\n', error);
64
65 %% Validate
66
67 fprintf('Validation:\n');
68 validationError = validateTwoLayerPerceptron_prep2(hiddenWeights, outputWeights, b1, b2, inputValuesTest, targetValuesTest);
69
70 fprintf('Validation error: %2.2f%%\n', validationError);
71
```