04-two_layer_net

June 14, 2020

1 Atividade 04: Implementando uma Rede Neural

Complete e entregue toda essa atividade (incluindo suas saídas e qualquer código adicional que você desenvolva) juntamente com a submissão de seu trabalho prático. Maiores detalhes podem ser vistos na página da disciplina.

Nesta atividade, você irá desenvolver uma rede neural com camadas completamente conectadas para realizar classificação de imagens, e irá testá-la utilizando o dataset CIFAR-10.

Nesta atividade, você irá:

- implementar uma **função de perda** (**loss function**) para uma rede neural de duas camadas
- implementar a expressão para seu **gradiente analítico**
- **verificar sua implementação** utilizando gradiente numérico
- **treinar** sua em um pequeno problema por meio de **SGD**
- **treinar e depurar** sua rede em um conjunto de **dados reais**
- usar um conjunto de validação para **ajustar hiperparâmetros**
- **visualizar** os pesos finais que foram obtidos

In [9]:
```
# Algum código de inicialização

import numpy as np
import matplotlib.pyplot as plt

from dl.classifiers.neural_net import TwoLayerNet

from __future__ import print_function

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returna erro relativo """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

```
        print('Okay!')
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
Okay!
```

Nesta atividade será utilizada a classe `TwoLayerNet` que está definida dentro do arquivo `dl/classifiers/neural_net.py` para representar instâncias de uma rede neural.

Os parâmetros da rede serão armazenados na variável de instância `self.params` que é um dicionário em que as chaves são os nomes (*strings*) de cada parâmetro e os valores são **arrays numpy**.

A seguir, você irá inicializar um pequeno conjunto de dados e um modelo simples que será usado para iniciar o desenvolvimento de sua implementação.

```
In [10]: # Cria um pequeno conjunto de dados aleatórios e um modelo simples para verificar sua i
         # Veja que foi fixado a 'semente aleatória' para possibilitar a repetição de experiment

         input_size = 4
         hidden_size = 10
         num_classes = 3
         num_inputs = 5

         def init_toy_model():
             np.random.seed(0)
             return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1)

         def init_toy_data():
             np.random.seed(1)
             X = 10 * np.random.randn(num_inputs, input_size)
             y = np.array([0, 1, 2, 2, 1])
             return X, y

         net = init_toy_model()
         X, y = init_toy_data()

         print('Okay!')
```

```
Okay!
```

2 Passo de Propagação (*Forward pass*): cálculo de *scores*

Abra o arquivo `dl/classifiers/neural_net.py` e analise o método `TwoLayerNet.loss`.

Esta função é muito similar as funções de perda que você escreveu anteriormente para as atividades envolvendo os modelos SVM e Softmax: ela utiliza os dados e os pesos (ou parâmetros) para calcular os *scores* de cada classe, o valor de perda/custo e os gradientes em relação aos parâmetros.

Você deve implementar a primeira parte do passo de propagação (*forward pass*) que utiliza os pesos e vieses (*biases*) para calcular os escores para todas as entradas.

```
In [11]: scores = net.loss(X)
         print('Your scores:')
         print(scores)
         print()
         print('correct scores:')
         correct_scores = np.asarray([
           [-0.81233741, -1.27654624, -0.70335995],
           [-0.17129677, -1.18803311, -0.47310444],
           [-0.51590475, -1.01354314, -0.8504215 ],
           [-0.15419291, -0.48629638, -0.52901952],
           [-0.00618733, -0.12435261, -0.15226949]])
         print(correct_scores)
         print()

         # A diferença deve ser bem pequena, algo < 1e-7
         print('Difference between your scores and correct scores:')
         print(np.sum(np.abs(scores - correct_scores)))

         print('Okay!')
```

```
Your scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
3.6802720496109664e-08
Okay!
```

3 Passo de Propagação (*Forward pass*): cálculo da perda/custo

Na mesma função, você deve implementar a segunda parte do passo de propagação (*forward pass*) responsável pelo cálculo da perda envolvendo os dados e a regularização.

```
In [12]: loss, _ = net.loss(X, y, reg=0.1)
         correct_loss = 1.30378789133
```

```
        # Novamente, a diferença deve ser pequena, algo < 1e-12
        print('Difference between your loss and correct loss:')
        print(np.sum(np.abs(loss - correct_loss)))

        print('Okay!')
```

```
Difference between your loss and correct loss:
1.794120407794253e-13
Okay!
```

4 Passo de Retropropagação (*Backward pass*)

Você deve implementar os restante da função `TwoLayerNet.loss`, de modo que a função calcule o gradiente da perda em relação aos parâmetros `W1`, `b1`, `W2` e `b2`.

Agora, uma vez que você implementou corretamente o passo de propagação, você pode depurar o passo de retropropagação como feito anteriormente: por meio do uso de estimativas numéricas do gradiente.

```
In [13]: from dl.gradient_check import eval_numerical_gradient

        # Usa verificação numérica do gradiente para checar sua implementação do passo de retro
        # Se sua implementação estiver correta, a diferença entre a estimativa numérica do grad
        # e o valor obtido analiticamente deve ser menor que 1e-8 para cada um dor parâmetros.

        loss, grads = net.loss(X, y, reg=0.05)

        # O erros deve ser menores que 1e-8
        for param_name in grads:
            f = lambda W: net.loss(X, y, reg=0.05)[0]
            param_grad_num = eval_numerical_gradient(f, net.params[param_name], verbose=False)
            print('%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[pa

        print('Okay!')
```

```
W2 max relative error: 3.440708e-09
b2 max relative error: 3.865028e-11
W1 max relative error: 3.561318e-09
b1 max relative error: 2.738422e-09
Okay!
```

5 Treinamento de uma Rede Simples

Para treinar uma rede, você deve usar o método SGD (método de descida mais íngreme estocástico), similar ao realizado com os classificarores SVM e Softmax.

Analise a função `TwoLayerNet.train` e preencha as partes que faltam para implementar o procedimento de treinamento. Isso deve ser muito similar ao realizado para os classificadores SVM e Softmax.

Você também deve implementar a função `TwoLayerNet.predict` pois ela será necessária durante o treinamento, uma vez que periodicamente o método realiza predições para acompanhar a acurácia ao longo do processo.

Uma vez que você tenha implementados tais métodos, execute o código abaixo para treinar um rede de duas camadas sobre o pequeno conjunto de dados aletários. Você deverá obter uma perda ao final do treinamento inferior a 0.2

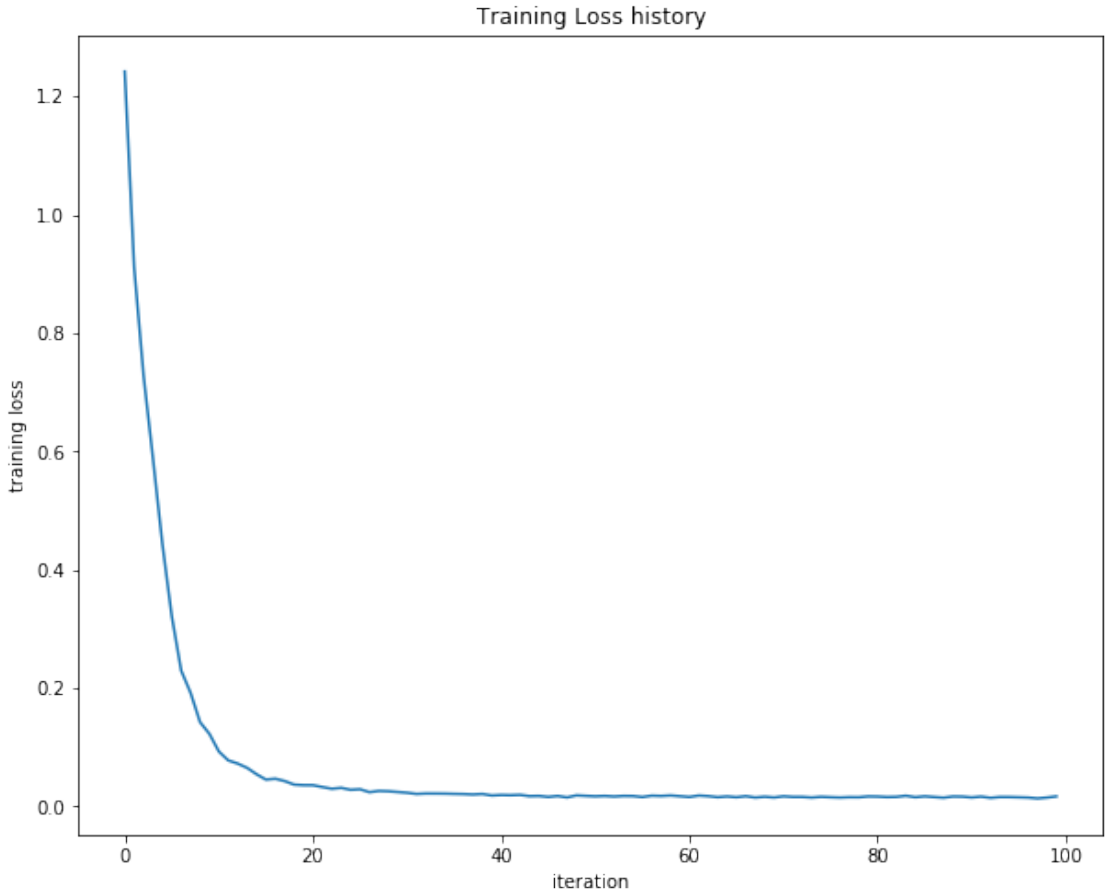
```
In [14]: net = init_toy_model()
         stats = net.train(X, y, X, y,
                     learning_rate=1e-1, reg=5e-6,
                     num_iters=100, verbose=False)

         print('Final training loss: ', stats['loss_history'][-1])

         # Plota o histórico da função de perda
         plt.plot(stats['loss_history'])
         plt.xlabel('iteration')
         plt.ylabel('training loss')
         plt.title('Training Loss history')
         plt.show()

         print('Okay!')
```

```
Final training loss:  0.017143643532923733
```




```
Okay!
```

6 Carregamento de dados

Agora que você implementou uma rede de duas camadas que passou pela verificação de gradientes e funcionado sobre o pequeno conjunto de dados, é hora de carregar os dados do CIFAR-10 dataset de modo que você possa usá-los no treinamento de um classificador sobre dados reais.

```
In [15]: from dl.data_utils import load_CIFAR10

         def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
             """
             Carrega CIFAR-10 dataset a partit do disco e realiza preprocessamento para preparar
             os dados para a rede neural de duas camadas. Estes são os mesmos passos usados para
             SVM, porém condensado em uma única função.
             """
             # Carregga os dados CIFAR-10 brutos
```

```
    cifar10_dir = 'dl/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subdivide os dados em conjuntos
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normaliza os dados: subtrai a imagem média
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Redimensiona as imagens de matrizes para vetores
    X_train = X_train.reshape(num_training, -1)
    X_val = X_val.reshape(num_validation, -1)
    X_test = X_test.reshape(num_test, -1)

    return X_train, y_train, X_val, y_val, X_test, y_test


# Usar a função definida acmina para obter os dados.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)

print('Okay!')
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
Okay!
```

7 Treinamento de uma Rede com Dados Reais

Para treinar sua rede, você deve usar **SGD+momentum**. Além disso, nesse processo a taxa de aprendizado será ajustada com um decaimento exponencial ao longo do processo de otimização, isto é, após cada época, a taxa de aprendizado é multiplicada pela taxa de decaimento (como esta última é menor que um, consequentemente a taxa de aprendizado é reduzida).

```
In [16]: input_size = 32 * 32 * 3
         hidden_size = 50
         num_classes = 10
         net = TwoLayerNet(input_size, hidden_size, num_classes)

         # Treinamento da rede
         stats = net.train(X_train, y_train, X_val, y_val,
                     num_iters=1000, batch_size=200,
                     learning_rate=1e-4, learning_rate_decay=0.95,
                     reg=0.25, verbose=True)

         # Predição sobre o conjunto de validação
         val_acc = (net.predict(X_val) == y_val).mean()
         print('Validation accuracy: ', val_acc)
```

```
iteration 0 / 1000: loss 2.302762
iteration 100 / 1000: loss 2.302358
iteration 200 / 1000: loss 2.297404
iteration 300 / 1000: loss 2.258897
iteration 400 / 1000: loss 2.202975
iteration 500 / 1000: loss 2.116816
iteration 600 / 1000: loss 2.049789
iteration 700 / 1000: loss 1.985711
iteration 800 / 1000: loss 2.003726
iteration 900 / 1000: loss 1.948076
Validation accuracy:  0.287
```

8 Depuração do treinamento

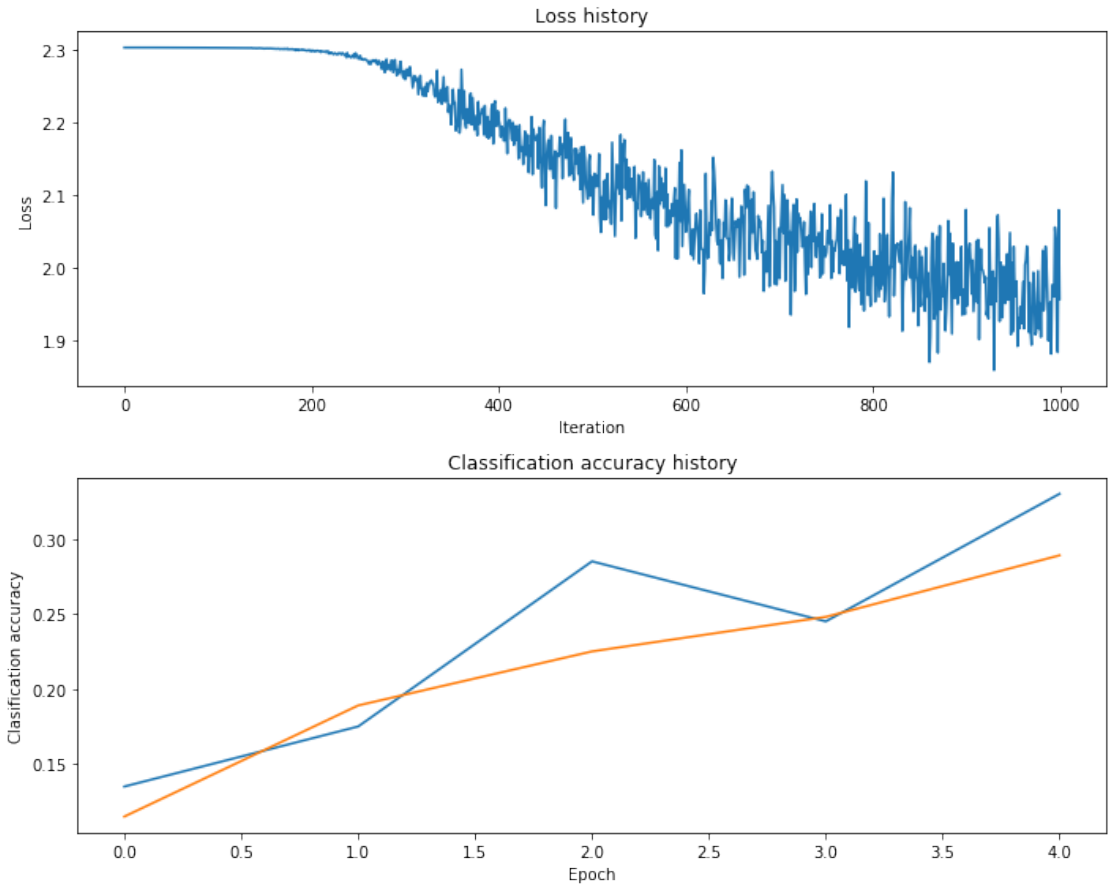
Com os valores de parâmetros fornecidos acima, você deve ter obtido uma acurácia no conjunto de validação em torno de 0.29. O que não representa um resultado muito bom...

Uma estratpegia para melhorar o entendimento (fornecer *insigths*) sobre o que pode estar errado é traçar os gráficos de evolução da função de perda e das acurácias de treinamento e validação ao longo do processo de otimização.

```
In [17]: # Plota a funcão de pedar e as acurácias de treinamento e validação
         plt.subplot(2, 1, 1)
         plt.plot(stats['loss_history'])
         plt.title('Loss history')
         plt.xlabel('Iteration')
```

```
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(stats['train_acc_history'], label='train')
plt.plot(stats['val_acc_history'], label='val')
plt.title('Classification accuracy history')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
plt.tight_layout()
plt.show()
```



Uma outra estratégia é construis uma visualização dos pesos que foram obtidos na primeira camada da rede. Por trás disto, está o fato de que na maioria das redes neurais treinadas sobre dados visuais, os pesos da primeira camada geralmente exibem algum tipo de estrutura visível.

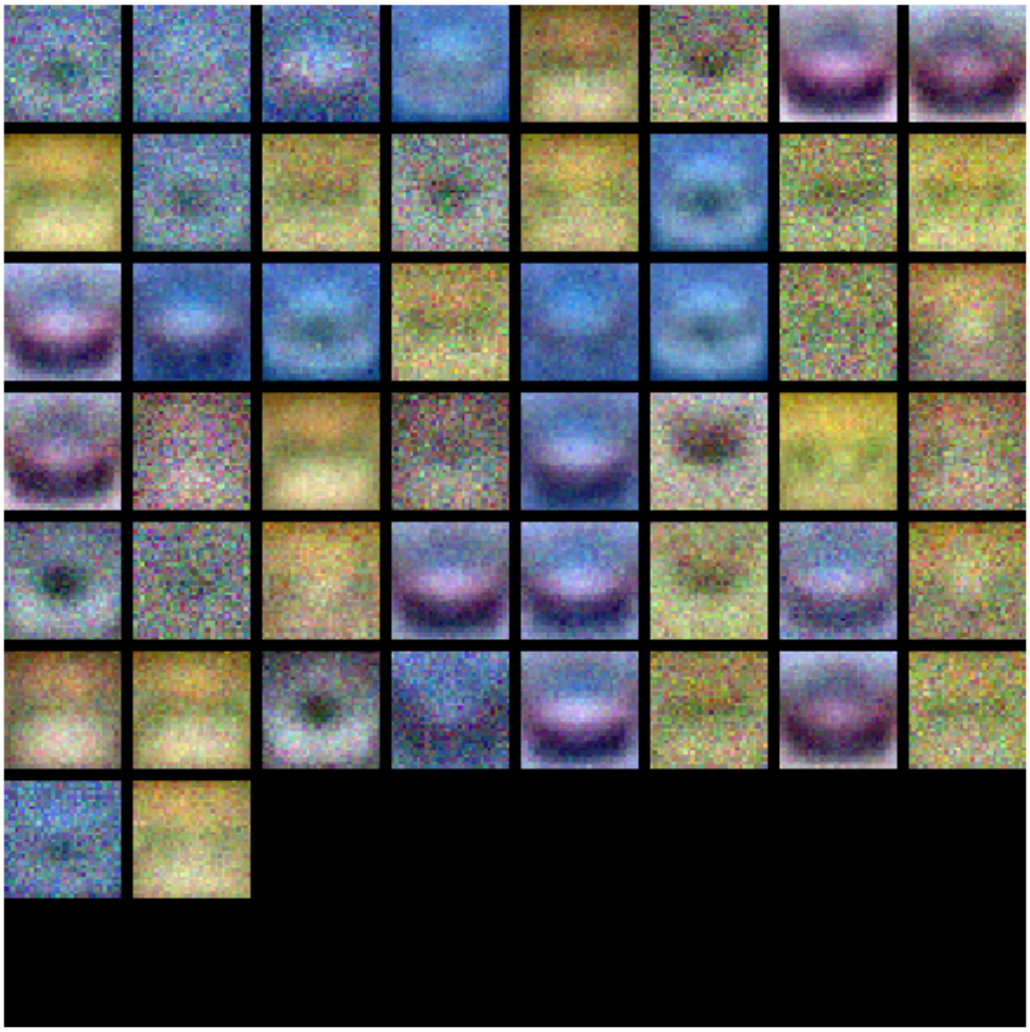
```
In [18]: from dl.vis_utils import visualize_grid

         # Visualiza os pesos da primeira camada da rede

         def show_net_weights(net):
```

```
    W1 = net.params['W1']
    W1 = W1.reshape(32, 32, 3, -1).transpose(3, 0, 1, 2)
    plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(net)
```



9 Ajuste de hiperparâmetros

O que está errado? Observando os gráficos e as visualizações acima, pode-se ver que a perda está reduzindo de forma *mais ou menos* linear, o que parece sugerir que a taxa de aprendizado pode estar muito baixa. Além disso, não há uma separação grande entre as acurácias de treinamento e

validação, sugerindo que o modelo usado tem baixa capacidade e que talvez seu tamanho devesse ser aumentado. Por outro lado, com um modelo muito grande deve-se esperar observar mais *overfitting*, que se manifesta por meio de uma distância muita grande entre acurácias de treinamento e de validação.

Ajuste. Realizar o ajuste de hiperparâmetros e desenvolver uma nocão intuitiva de como eles afetam o resultado final é uma parte importatnte do uso de Redes Neurais. Dessa forma, deseja-se que você realize várias práticas envolvendo o ajuste de hiperparâmetros. A seguir, você deve realizar experimentos com diferentes valores para os hiperparâmetros incluindo: tamanho da camada escondida, taxa de aprendizado, taxa de decaimento, número de épocas de treinamento e regularização.

Resultados aproximados. Você deve tentar alcançar uma acurácia de classificação (taxa de acerto) maior que 48% no conjunto de validação. (OBS: minha solução obteve uma acurácia acima de 52% no conjunto de validação!)

Experimento: O objetivo desse exercício é que você tente obter o melhor resultado sobre a base CIFAR-10, usando uma rede neural completamente conectada. Para cada 1% acima de 52% sobre a acurácia no **conjunto de teste**, você irá receber um ponto extra. Sinta-se livre para implementar quaisquer técnicas que desejar (p.ex., redução de dimensionalidade via PCA, *dropout*, ou outras estratégias ao otimizador, etc).

OBS: Lembre-se de deixar documentado tudo que foi feito! Caso necessário, acrescente mais células a sua vontade.

In [23]:
```
best_net = None # Armazenar o melhor modelo encontrado nesse variável pois será
                # usado no teste final (ver final desse notebook)

################################################################################
# TODO: Ajustar hiperparâmetros usando o conjunto de validação. O melhor modelo #
# obtido ao longo do treinamento deve-se armazenado em best_net.               #
#                                                                              #
# Para auxiliar a depurar sua rede, pode ser interessante se utilizar de       #
# visualizações similares as usadas acima. Essas visualizações irão apresentar #
# diferenças qualitativas significativas especialmente para redes com ajustes  #
# ruins.                                                                       #
#                                                                              #
# O ajuste fino de hiperparâmetros feito manualmente pode ser divertido, mas   #
# provavelmente você deverá considerar a possibilidade de escrever código que  #
# percorra todas as combinações possíveis de hiperparâmetros de forma a tornar #
# automático o processo de busca (similar ao que foi feito nas atividades      #
# anteriores).                                                                 #
################################################################################
best_val = -1
best_stats = None
learning_rates = [1e-1, 1e-2, 1e-3, 1e-4]
regularization_strengths = [1e-1, 1e-2, 1e-3, 1e-4]
batch_sizes = [200, 400, 800]
hidden_sizes = [80, 160, 320]
results = {}
iters = 2000
total_size = 144
```

```
        i = 0
        for lr in learning_rates:
            for rs in regularization_strengths:
                for bs in batch_sizes:
                    for hs in hidden_sizes:
                        i += 1
                        print (i, '/', total_size)
                        net = TwoLayerNet(input_size, hs, num_classes)
                        # Train the network
                        stats = net.train(X_train, y_train, X_val, y_val,
                                    num_iters=iters, batch_size=bs,
                                    learning_rate=lr, learning_rate_decay=0.95,
                                    reg=rs)
                        y_train_pred = net.predict(X_train)
                        acc_train = np.mean(y_train == y_train_pred)
                        y_val_pred = net.predict(X_val)
                        acc_val = np.mean(y_val == y_val_pred)
                        results[(lr, rs, bs, hs)] = (acc_train, acc_val)
                        if best_val < acc_val:
                            best_stats = stats
                            best_val = acc_val
                            best_net = net

        print ('best validation accuracy achieved during cross-validation: %f', best_val)
        ################################################################################
        #                              FIM DE SEU CÓDIGO                               #
        ################################################################################
```

```
1 / 144
```

```
C:\Users\Digão\Google Drive\Mestrado\Disciplinas\Deep Learning\Atividade 1\dl\classifiers\neural
  a2 = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
C:\Users\Digão\Google Drive\Mestrado\Disciplinas\Deep Learning\Atividade 1\dl\classifiers\neural
  correct_log_probs = -np.log(a2[range(N), y])
C:\Users\Digão\Google Drive\Mestrado\Disciplinas\Deep Learning\Atividade 1\dl\classifiers\neural
  dhidden[z1 <= 0] = 0
```

```
2 / 144
3 / 144
4 / 144
5 / 144
6 / 144
7 / 144
8 / 144
9 / 144
10 / 144
```

11 / 144
12 / 144
13 / 144
14 / 144
15 / 144
16 / 144
17 / 144
18 / 144
19 / 144
20 / 144
21 / 144
22 / 144
23 / 144
24 / 144
25 / 144
26 / 144
27 / 144
28 / 144
29 / 144
30 / 144
31 / 144
32 / 144
33 / 144
34 / 144
35 / 144
36 / 144
37 / 144
38 / 144
39 / 144
40 / 144
41 / 144
42 / 144
43 / 144
44 / 144
45 / 144
46 / 144
47 / 144
48 / 144
49 / 144
50 / 144
51 / 144
52 / 144
53 / 144
54 / 144
55 / 144
56 / 144
57 / 144
58 / 144

59 / 144
60 / 144
61 / 144
62 / 144
63 / 144
64 / 144
65 / 144
66 / 144
67 / 144
68 / 144
69 / 144
70 / 144
71 / 144
72 / 144
73 / 144
74 / 144
75 / 144
76 / 144
77 / 144
78 / 144
79 / 144
80 / 144
81 / 144
82 / 144
83 / 144
84 / 144
85 / 144
86 / 144
87 / 144
88 / 144
89 / 144
90 / 144
91 / 144
92 / 144
93 / 144
94 / 144
95 / 144
96 / 144
97 / 144
98 / 144
99 / 144
100 / 144
101 / 144
102 / 144
103 / 144
104 / 144
105 / 144
106 / 144

```
107 / 144
108 / 144
109 / 144
110 / 144
111 / 144
112 / 144
113 / 144
114 / 144
115 / 144
116 / 144
117 / 144
118 / 144
119 / 144
120 / 144
121 / 144
122 / 144
123 / 144
124 / 144
125 / 144
126 / 144
127 / 144
128 / 144
129 / 144
130 / 144
131 / 144
132 / 144
133 / 144
134 / 144
135 / 144
136 / 144
137 / 144
138 / 144
139 / 144
140 / 144
141 / 144
142 / 144
143 / 144
144 / 144
best validation accuracy achieved during cross-validation: %f 0.56
```

```
In [24]: # visualizar os pesos da primeira camada da melhor rede obtida
         show_net_weights(best_net)
```

10 Executar predições sobre o conjunto de teste

Quando você terminar com seus experimentos acima (**nunca antes!!!**), você deve avaliar sua rede final sobre o conjunto de teste e o resultado de acurácia (taxa de acerto) deve ser acima de 48%.

Você irá ganhar ponto extra para cada 1% de acurácia obtido acima de 52%.

```
In [25]: test_acc = (best_net.predict(X_test) == y_test).mean()
         print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.541
```