Practica2-XarxesNeurals

November 15, 2019

1 Introducció a la pràctica 2: primers passos

1.1 Objectius

Els objectius d'aquesta sessió són:

- Aplicar models de xarxes neurals, posant l'èmfasi en:

  1. Aplicar diferents topologies de xarxes neurals i entendre els avantatges de cada una.
  2. Avaluar correctament l'error del model
  3. Visualitzar les dades i el model resultant

- Ésser capaç d'aplicar tècniques de xarxes neurals en casos reals
- Validar els resultats en dades reals
- Fomentar la capacitat per presentar resultats tècnics d'aprenentatge computacional de forma adequada davant altres persones

1.2 Materials per a la sessió

1. Base de dades amb imatges de digits (Caronte). Aquesta base de dades està separada en els següents conjunts:

- Train: la xarxa neural s'haurà d'entrenar sobre aquest conjunt.
- Validació: la xarxa entrenada en el conjunt de Train s'haurà d'evaluar sobre el conjunt de validació i les etiquetes predites per la xarxa es podran validar.
- Test: en la pàgina de la competició Codalab existeix un nou set de dades (sense etiquetar, ocult per a vosaltres) amb el que, amb les prediccions de la vostra xarxa, es generarà la puntuació definitiva de la competició.

2. Llibreria pytorch de python.
3. Codi d'exemple (Cerbero i aquest enunciat).
4. Apunts de l'assignatura.

1.3 Competició

Per a participar a la competició, primer de tot, un component del grup s'ha de registrar al Codalab. El nom d'usuari ha de ser "GrupXX", on XX es el numero del vostre grup. M'haureu de passar el user/pwd quan esteu donats d'alta (a Jordi.Gonzalez@uab.cat).

Per fer les submissions, heu de pujar un ZIP amb el fitxer de prediccions que genera l'script bash "run_tests.sh" en el codi d'exemple que podeu trobar a Caronte. Aquest fitxer, anomenat "predictions_class.pkl" el trobareu al directori competition/results/final/.

Alternativament, es pot generar l'arxiu .pkl directament des del propi codi, sense haver d'executar l'script `run_tests.sh`. Per això, es poden fixar els paràmetres (número d'epochs i target a aprendre) del codi en comtpes de passar-los com a arguments amb l'script, i executar el codi directament.

** Comprimiu únicament el fitxer .pkl, sense incloure la ruta a cap directori dins del ZIP. **

Després, a resultats, veureu la precisió que ha obtingut la vostra predicció, i el rànking respecte els vostres companys.

Molta sort, recordeu que teniu un Forum, i sobretot gaudiu de la competició!

1.4 Continguts que s'avaluaran

Hi hauran 2 entregues en un ZIP a Cerbero:

- 26/11 sessió de control: apartat (C), serà obligatori per aprovar la pràctica 2 (amb un 5.0);
- 10/12 sessió d'avaluació: apartats (B) i (A), opcionals.

La data d'entrega de tots els apartats es a la una de la matinada del dilluns anteriors a aquestes sessions, és a dir, una hora després de diumenge a les 23:59.

Els ZIPs contindran:

1. Memòria explicant els resultats trobats en la base de dades que heu treballat, detallant el passos seguits (60% de la nota).
2. Codi python desenvolupat (30% de la nota)
3. Presentació amb els resultats 10 min màxim (10% de la nota)

Per la sessió de control de la pràctica 2 del 26/11, no caldrà pujar la memòria de la pràctica 2.

1.5 Avaluació i entregues de la pràctica 2

En la pràctica 2, es presenten diversos problemes per comprendre les xarxes neuronals. Els problemes s'organitzen en tres nivells de dificultat: A, (sobre 10), dificultat alta; B, (sobre 7.5), dificultat mitjana i C, (sobre 5), dificultat baixa.

Per aprovar la pràctica és requisit necessari completar satisfactòriament els problemes d'assoliment baix (C), demostrant així una comprensió mínima de la matèria. Per tant, la superació de la pràctica 2 estarà condicionada a la presentació de la documentació sobre l'apartat (C), i serà opcional realitzar els apartats (B) i (A).

A més, s'ha dissenyat un concurs on els diferents grups competeixen per aconseguir ser els que millor classifiquin un conjunt d'instàncies secretes corresponents a imatges de digits de l'u al deu escrits a mà (MNIST).

Les entregues s'organitzen en tres nivells d'assoliment dels objectius, incrementals: apartat **C**, (sobre 5 punts), assoliment baix; apartat **B, (optimització, sobre 2.5 punts)**, assoliment mig; i apartat **A, (xarxes neurals multi-classe, sobre 2.5 punts)**, assoliment alt. La suma dels 3 apartats serà la nota final de la pràctica 2.

1.5.1 Sessió de control apartat (C), Aplicant xarxes neurals

En la sessió de control del 26 de novembre, serà opcional entregar en una presentació que demostri que **s'ha realitzat l'apartat (C) de la pràctica 2**, pujant al Cerbero la nit abans un ZIP amb el codi, el ppt (5 minuts) de l'apartat (C).

Així, aquesta sessió està orientada a que, durant 1 hora i mitja, els alumnes que vingueu pugueu preguntar i resoldre dubtes sobre la llibreria pytorch, preguntar sobre l'objectiu de cada apartat dels enunciats que no us hagi quedat clar, i preguntar sobre els resultats que esteu obtenint a l'hora d'aplicar les xarxes neurals.

Opcionalment, podeu pujar a Caronte el codi python d'aquest apartat. Fer la presentació en aquesta sessió de seguiment també és opcional. En definitiva, l'objectiu de la sessió és que al sortir tingueu clar com són les vostres dades, què cal entregar i com implementar cada apartat C, B i A. Per això, si ja ho teniu clar, l'assistència és voluntària.

Les presentacions en les sessions de control es faran davant de tota la classe, d'un màxim de 5 minuts. Porteu-les en pdf en una memòria USB. Es recomana una primera slide describint les arquitectures de xarxes neurals que heu provat, una altra amb les funcions del pytorch més importants, una tercera resumint els paràmetres de la xarxa neural que s'han provat, una altra amb les conclusions que s'hagin tret, i una última amb els problemes que us hàgiu trobat i quina solució proposeu, o si ja els heu solucionat, com ho vau solucionar per ajudar als vostres companys.

1.5.2 Sessió d'avaluació apartats (B) i (A), Optimització i xarxes multi-classe

En la següent sessió del 10 de desembre, la màxima puntuació per la **pràctica 2** s'aconsegueix abordant les preguntes dels apartats (B) i (A), d'entrega opcional. Caldrà pujar al Caronte abans de les 00:59 del dilluns 9 de desembre un ZIP amb el codi, la documentació i el ppt (10 minuts) d'aquests apartats. Podeu "recuperar" l'apartat (C) entregant-ho per aquesta sessió, ja que fer l'apartat (C) és obligagtori.

2 Sessió de control Pràctica 2: apartat (C) Aplicant Xarxes Neurals

Aquesta pràctica introdueix l'alumne en l'ús pràctic de les xarxes neuronals dins el marc de la classificació multi-classe. Per a això cal dominar els conceptes relacionats amb les xarxes neuronals i els paràmetres associats.

L'objectiu d'aquest apartat és entrenar una xarxa neural com a classificador per cada una de les categories de la base de dades MNIST. Concretament, s'haurà d'aprendre una xarxa neural per cadascun dels tipus de digit a la base de dades $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. En total s'obtindrà 10 classificadors binaris, que retornaran 1 si l'input pertany a la classe amb què s'ha entrenat el classificador o 0 si no hi pertany.

Per tant es demana als diferents grups que mentre competeixen per obtenir bons resultats abordin els següents punts clau:

1. Estudi de la influència dels diferents paràmetres de pytorch en els 10 classificadors obtinguts (un per cada classe). Amb especial èmfasi en:

    - Normalització de les features
    - Learning rate
    - Momentum

- Batch size
- Epochs
- Weight decay
- Dropout

2. Altres trobades en la documentació PyTorch (important revisar la documentació).
3. Recomanem que feu una taula amb els diferents paràmetres provats i els resultats obtinguts. Per exemple:

A continuació us proporcionem un exemple interactiu de codi i del funcionament de PyTorch. **Recordeu que cada celůla depen de l'anterior i cal executar-les totes en ordre**.

2.1 Exemple de PyTorch

- Instalůlació de la llibreria.
  1. Triar la configuració dessitjada i seguir la comanda especificada a la pàgina de pytorch.
  2. Instalůlar dependències per a executar aquest notebook (en cas que el volgueu executar interactivament).
     - `numpy`
     - `matplotlib`
  3. Jupyter notebook.
  4. (Opcional) Nvidia Drivers i CUDA per a permetre execució en paralůlel en una GPU.

Recordeu ser consistents en la versió de python que utilitzeu. Recomanem utilitzar python3.6.

- Carreguem les llibreries i binaritzem el dataset de training.

```
In [23]: %matplotlib inline

         import torch # Import main library
         from torch.utils.data import DataLoader # Main class for threaded data loading
         import torch.nn.functional as func
         import matplotlib.pyplot as plt
         import numpy as np
         import random
         from sklearn.metrics import confusion_matrix

         # Optimizaiton config
         target_class = 3 # Train a classifier for this class
         batch_size = 50 # Number of samples used to estimate the gradient (bigger = stable trai
         learning_rate = 0.05 # Optimizer learning rate
         epochs = 25 # Number of iterations over the whole dataset.

         # Prepare data
         train_data = np.load('base_dades_xarxes_neurals/train.npy')
         val_data = np.load('base_dades_xarxes_neurals/val.npy')
```

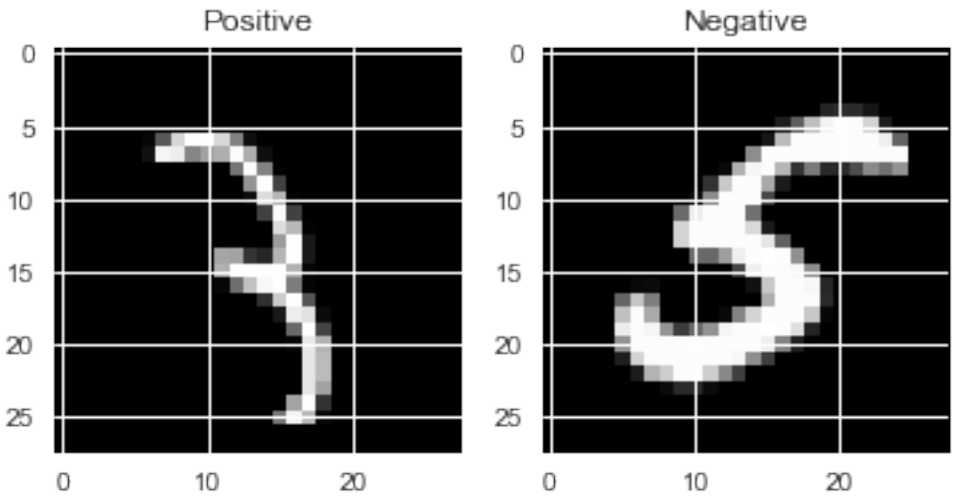
```
        def select_class(data, clss):
            images = np.array(data.item()["images"])
            labels = np.array(data.item()["labels"])
            labels = (labels == target_class).astype(int)
            return images, labels


        def getAllData(data, train=False):
            images = np.array(data.item()["images"])
            labels = np.array(data.item()["labels"])
            if train:
                indices = list(range(images.shape[0]))
                random.shuffle(indices)    # quan passeu les mostres del test de Codalab, NO FEU
                images = images[indices]
                labels = labels[indices]
            return images, labels

        ##=====Apartat B,C
        train_images, train_labels = select_class(train_data, target_class) # Binary case: here
        val_images, val_labels = select_class(val_data, target_class) # Binary case: here class

        ##=====Apartat A
        #train_images, train_labels = getAllData(train_data, train=True) # 10-class case (Apart
        #val_images, val_labels = getAllData(val_data, train=False) # 10-class case (Apartat A)

        train_size = train_labels.shape[0]
        val_size = val_labels.shape[0]

        print(train_size, "training images.")
```

```
47995 training images.
```

Verifiquem que les dades s'hagin carregat correctament:

```
In [24]: indices = np.arange(train_size)
         positive_indices = indices[train_labels == 1]
         negative_indices = indices[train_labels == 0]

         plt.figure()
         plt.subplot(1, 2, 1)
         plt.title("Positive")
         plt.imshow(train_images[positive_indices[0], :, :], cmap="gray")
         plt.subplot(1, 2, 2)
         plt.title("Negative")
         plt.imshow(train_images[negative_indices[0], :, :], cmap="gray")
```

```
Out[24]: <matplotlib.image.AxesImage at 0x1a188b2c50>
```



- Creem una xarxa neural amb 3 capes: input, output i una hidden layer (layer2). És important que les dimensions d'entrada i sortida siguin correctes: l'entrada ha de ser 28 * 28 (num de pixels de la imatge, que té mida 28 x 28)
- La sortida ha de ser de mida 1 en els apartats C i B (ja que és un classificador binari), i 10 en l'apartat A (és un classificador de 10 classes).

```
In [25]: def init_weights(net):
             if type(net) == torch.nn.Module:
                 torch.nn.init.xavier_uniform_(net.weight) # Xavier initialization
                 net.bias.data.fill_(0.01)  # tots els bias a 0.01

         class NeuralNet(torch.nn.Module):
             def __init__(self):
                 super().__init__() # Necessary for torch to detect this class as trainable
                 # Here define network architecture
                 self.layer1 = torch.nn.Linear(28**2, 32) # Linear layer with 32 neurons
                 self.layer2 = torch.nn.Linear(32, 64) # Linear layer with 64 neurons

                 ##=====Apartat B,C
                 self.output = torch.nn.Linear(64, 1) # Linear layer with 1 neuron (binary outpu

                 ##=====Apartat A
                 #self.output = torch.nn.Linear(64, 10) # Linear layer with 10 neurons (10-class


             def forward(self, x):
                 # Here define architecture behavior
                 x = torch.sigmoid(self.layer1(x)) # x = torch.nn.functional.relu(self.layer1(x)
                 x = torch.sigmoid(self.layer2(x))  # si dona error 'Cannot find reference 'sigm
```

```
                                                    #proveu x = func.sigmoid(self.layer1(x)), h

                ##=====Apartat B,C
                return torch.sigmoid(self.output(x)) # Binary output

                ##=====Apartat A
                #return torch.nn.functional.log_softmax(self.output(x), dim=1)  # 10 classes ne

        # Instantiate network
        model = NeuralNet()
```

- Creem l'optimitzador, declarem la funció a optimitzar, i la resta de funcions auxiliars per a optimitzar el model.

```
In [26]: # Try different initialization techniques, en aquest cas la de Xavier
         model.apply(init_weights)

         # Create optimizer for the network parameters
         optimizer = torch.optim.SGD(model.parameters(), learning_rate)

         # Instantiate loss function
         criterion = torch.nn.BCELoss() # Binary logistic regression

         # Function to iterate the training set and update network weights with batches of image
         def train(model, optimizer, criterion):

             model.train() # training mode

             running_loss = 0
             running_corrects = 0
             total = 0

             for idx in range(0, train_size, batch_size):
                 optimizer.zero_grad() # make the gradients 0
                 x = torch.from_numpy(train_images[idx:(idx + batch_size), ...]).float()
                 y = torch.from_numpy(train_labels[idx:(idx + batch_size), ...]).float()
                 output = model(x.view(-1, 28 ** 2)) # forward pass

                 ##=====Apartat B,C
                 preds = (output > 0.5).float()
                 loss = criterion(output.view_as(y), y) # calculate the loss value

                 ##=====Apartat A
                 # 10 classes neural network (Apartat A)
                 #preds = torch.argmax(output, 1)
                 #loss = torch.nn.functional.cross_entropy(output, y.long())

                 loss.backward() # compute the gradients
```

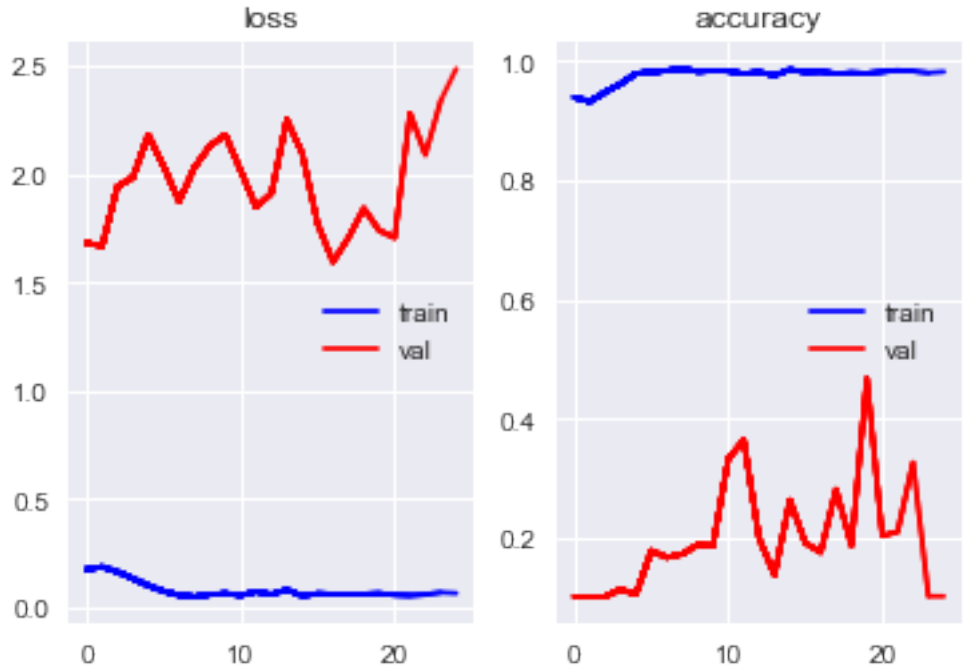
```
            optimizer.step() # uptade network parameters

            # statistics
            running_loss += loss.item() * x.size(0)


            ##=====Apartat B,C
            running_corrects += torch.sum(preds.data.view(-1)==y.data.view(-1)).item() # .i
            # La raó és que preds té mida (batch, 1) i labels té mida (batch).
            # Pel què al fer ==, torch repeteix preds i labels per a que tinguin mida (batc
            # el .data, l'únic que fa es dir a torch que no cal fer backpropagation amb aix

            ##=====Apartat A
            #running_corrects += torch.sum(preds == y.data.view(-1).long()).item()

            total += float(y.size(0))

        epoch_loss = running_loss / total # mean epoch loss
        epoch_acc = running_corrects / total # mean epoch accuracy

        return epoch_loss, epoch_acc

    # Function to iterate the validation set and update network weights with batches of ima
    def val(model, criterion):

        model.eval() # validation mode

        running_loss = 0
        running_corrects = 0
        total = 0

        predicted = []

        with torch.no_grad(): # We are not backpropagating trhough the validation set, so u
            for idx in range(0, val_size, batch_size):
                x = torch.from_numpy(val_images[idx:(idx + batch_size), ...]).float()
                y = torch.from_numpy(val_labels[idx:(idx + batch_size), ...]).float()
                output = model(x.view(-1, 28 ** 2)) # forward pass

                ##=====Apartat B,C
                # Binary case (Apartat B, C)
                preds = (output > 0.5).float()
                loss = criterion(output.view_as(y), y) # calculate the loss value

                ##=====Apartat A
                # 10-classes nbeural network (Apartat A)
                #preds = torch.argmax(output, 1)
                #loss = torch.nn.functional.cross_entropy(output, y.long())
```

```
                predicted += preds.tolist()

                # statistics
                running_loss += loss.item() * x.size(0)

                ##=====Apartat B,C
                running_corrects += torch.sum(preds.data.view(-1)==y.data.view(-1)).item()
                # La raó és que preds té mida (batch, 1) i labels té mida (batch).
                # Pel què al fer ==, torch repeteix preds i labels per a que tinguin mida (
                # el .data, l'únic que fa es dir a torch que no cal fer backpropagation amb

                ##=====Apartat A
                #running_corrects += torch.sum(preds == y.data.view(-1).long()).item()

                total += float(y.size(0))

        targets = np.array(val_labels)
        confusionMatrix = confusion_matrix(targets, np.array(predicted))

        epoch_loss = running_loss / total # mean epoch loss
        epoch_acc = running_corrects / total # mean epoch accuracy

        return epoch_loss, epoch_acc, confusionMatrix
```

- Iterem sobre el dataset sencer *epochs* vegades i mostrem el train loss i accuracy.

```
In [27]: # Main training loop
         train_loss = []
         train_accuracy = []
         val_loss = []
         val_accuracy = []

         # Remove this line out of jupyter notebooks
         from IPython import display
         for epoch in range(epochs):
             t_loss, t_acc = train(model, optimizer, criterion)

             v_loss, v_acc, confusionMatrix = val(model, criterion)

             train_loss.append(t_loss)
             train_accuracy.append(t_acc)
             val_loss.append(v_loss)
             val_accuracy.append(v_acc)

             plt.subplot(1,2,1)
             plt.title("loss")
             plt.plot(train_loss, 'b-')
```

```
        plt.plot(val_loss, 'r-')
        plt.legend(["train", "val"])
        plt.subplot(1,2,2)
        plt.title("accuracy")
        plt.plot(train_accuracy, 'b-')
        plt.plot(val_accuracy, 'r-')
        plt.legend(["train", "val"])
        display.clear_output(wait=True)
        display.display(plt.gcf())

display.clear_output(wait=True)
```



Hauria d'acabar a 90+% train accuracy i val accuracy. Recordeu que evaluem les imatges d'una classe contra la resta. Per tant, hi haurà moltes més imatges negatives que positives i és possible obtenir bona accuracy aprenent a predir sempre negatiu.

Sabent això, és l'accuracy una bona mesura per a aquest apartat? Podrieu calcular quin accuracy s'obtindria si el model tingués output constant a 0? I si fos aleatori? Com és la matriu de confusió?

2.2 Important

Un cop entrenats 10 models, un per cada classe, haurem de generar la predicció final (de 0 a 10) per a cada imatge. Una manera és mirar quin score assigna cada classificador a la imatge i assignar la classe corresponent al classificador amb màxim score.

Així es podrà contestar a aquestes preguntes:

- Quina influència té el learning rate sobre el resultat final?
- Quina arquitectura es comporta millor?
- Té sentit normalitzar les cada mostra?
- S'aconsegueix sempre el mateix resultat per a una mateixa arquitectura? Per què?

3 Sessió d'avaluació Pràctica 2: apartat (B) Optimització de l'Arquitectura i l'Aprenentatge

Un cop s'ha provat diferents configuracions en l'entrenament de les xarxes neurals, es tractarà d'avaluar com de bona és una arquitectura en concret, canviant la inicialització.

Es tractarà de desenvolupar l'avaluació i comparació de les diferents configuracions provades en l'apartat anterior, estudiant ara els efectes de l'overffiting i underfitting segons el valor dels diferents paràmetres (es recomana incloure gràfiques comparant error en training i validació, com les mostrades a l'apartat anterior), a partir de canviar els següents paràmetres:

- Nombre de capes ocultes
- Nombre de neurones per capa
- Estudiar la influència dels diferents mètodes d'inicialització de la xarxa (aleatòria, Xavier uniforme, Nguyen-Widrow, Kaiming, Ortogonal, ...):

```
torch.nn.init.xavier_uniform_(m.weight)
```

- Funcions de transferència/activació (ReLu, TanH, Sigmoid,...)

** En definitiva, caldrà estudiar les formes d'agilitar la velocitat de la xarxa, reduint el temps de convergència de la mateixa, comparat amb l'apartat anteriur, a partir d'una selecció adequada dels paràmetres**

Recordeu que esteu entrenant 10 classificadors, un per a cada dígit. Per cada classificador, te un 10% de mostres positives (el número concret que esteu aprenent) i un 90% de mostres negatives (la resta de números). Per tant, al no tenir el mateix número de mostres per cada classe (positiva i negativa), si pugeu la predicció dels 10 classificadors aplicats al test set de Codalab, cada classificador tendirà a predir negatiu per tal de tenir un accuracy alt pel seu número. però això comportarà obtenir un valor molt baix d'accuracy en el test set de codalab, ja que hi han mostres de cada número. ** La solució passa per balancejar el número de positius/negatius, o implementar el següent apartat **

Així es podrà contestar a aquestes preguntes:

- Quina relació hi ha entre el número de pesos i l'overfitting/underfitting?
- Per quina inicialització de pesos, l'aprenentatge millora o és més ràpid en convergir?
- Hi ha diferència entre la mida òptima/teòrica de la xarxa i la que millor resultats us ha donat?

Per aquesta pregunta, teniu en compte que es tracta de fixar els paràmetres (coeficient d'aprenentatge, inicialitzacio, etc) i variar el tamany de la capa oculta i el número de capes ocultes per veure quan va pitjor en validació (overfitting) o en training (underfitting). Caldrà posar a l'informe tots els experiments que s'han fet, no únicament els que han donat el millor resultat.

De totes formes, tingueu present que degut al tamany reduit del conjunt d'aprenentatge que us hem passat, molts dels conceptes de teoria de les xarxes neuronals els veureu quan treballeu amb grans volums de dades: l'objectiu d0aquest apartat és més familiaritzar-se amb l'estratègia per identificar els millors paràmetres pel problema.

4 Sessió d'avaluació Pràctica 2: apartat (A) Aprendre 10 Classes amb una Única Xarxa Neural

Un cop s'ha provat diferents inicialitzacions i arquitectures de xarxes neurals, identificant la millor per a cada classe, es tractarà de fer aprendre una única xarxa neural amb 10 neurones de sortida, una per a cada classe (per exemple, una sortida [1,0,0,0,0...] correspon a la primera classe (0); [0,1,0,0,0] vol dir que la xarxa neural classifica la mostra d'entrada com a la classe (1); i així).

Per tant, cal aprendre en totes les mostres de totes les classes a la vegada, i a l'hora de classificar ja no es tractarà d'agafar el màxim d'entre les sortides de les 10 xarxes neurals que teniem fins ara per a cada una de les classes, sino que la classificació a la classe m correspondrà a aquella neurona de sortida que s'hagi activat amb més força.

Els canvis que s'hauran de fer en l'arquitectura son trivials, tot i que és possible que l'arquitectura que funcionava bé amb una sortida s'hagi de canviar per a que funcioni optimament amb 10 sortides:

`self.output = torch.nn.Linear(64, 10) # Linear layer with 10 neurons (num of classes)`

I no caldrà binaritzar la sortida:

`return self.output(x)`

També s'ha de canviar la funció de loss per a que accepti distribucions de probabilitat:

`criterion = torch.nn.CrossEntropy()`

O modificar únicament la funció d'activació de la capa de sortida:

`torch.nn.functional.log_softmax`

Recordeu que l'input de `criterion` per a cada imatge és un vector de 10 scores, una per a cadascuna de les 10 classes, per tant heu de fer la modificació corresponent per a que tingui mida *batchsize* $\times$ 10 (mirar el `view_as`).

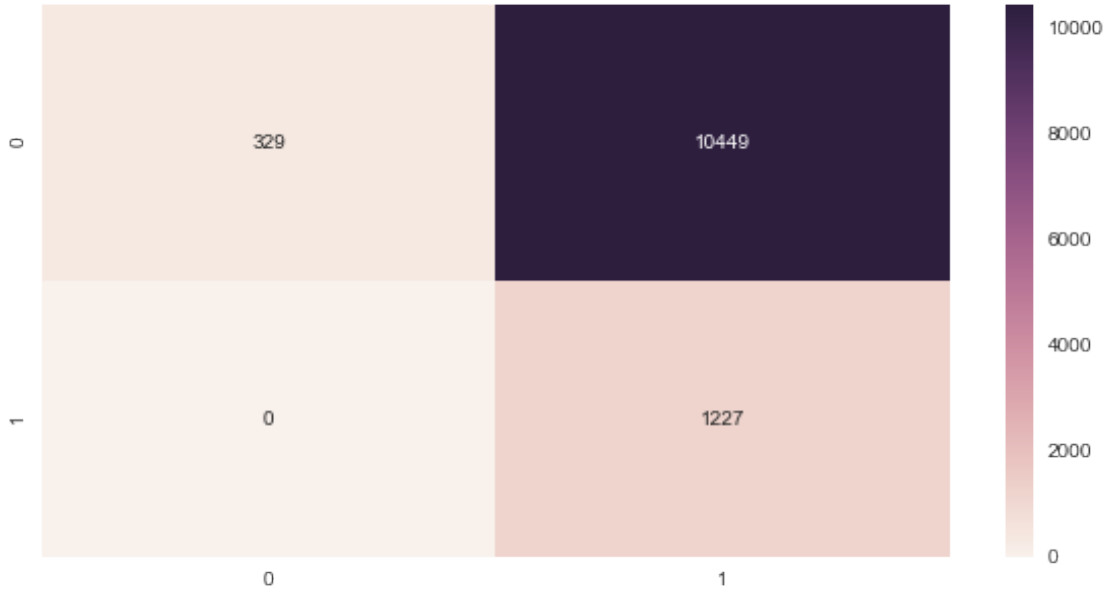
Pel que fa a l'anàlisi de resultats es recomana la utilització de matrius de confusió:

Així es podrà contestar a aquesta pregunta:

- Per quina inicialització de pesos (Xavier?), i valors vistos en els anteriors apartats de learning rate, momentum, funció d'activació, dropout, etc, l'aprenentatge millora o és més ràpid en convergir?
- Què és millor, utilitzar una xarxa neural especialitzada en una classe en particular, o una única xarxa neural que discrimini entre diferents classes? Per què?

```
In [21]: import pandas as pd
         import seaborn as sn

         ##=====Apartat B,C: Visualització de la matriu de confusio pel cas binari
```

```
plotConfusionMatrix = pd.DataFrame(confusionMatrix, index=[i for i in range(2)], column
plt.figure(figsize=(10, 5))
sn.heatmap(plotConfusionMatrix, annot=True, fmt='d')
plt.show()


##=====Apartat A: Visualització de la matriu de confusio per l'apartat A (10-class neur
#plotConfusionMatrix = pd.DataFrame(confusionMatrix, index=[i for i in range(10)], colu
#plt.figure(figsize=(12, 7))
#sn.heatmap(plotConfusionMatrix, annot=True, fmt='d')
#plt.show()
```



5 Resultats que s'avaluaran

1. Memòria explicant els resultats trobats en la base de dades que heu treballat, detallant el passos seguits (60% de la nota).
2. Codi python desenvolupat (30% de la nota)
3. Presentació amb els resultats (10% de la nota)
4. Classificació final en la competició (rànquing a partir d'un conjunt desconegut de test), ja que s'otorgaran punts addicionals als equips millor classificats: **1 punt addicional en la nota final de l'assignatura pels 3 equips que acabin entre els 3 primers**