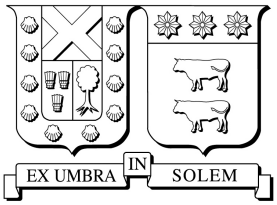


INF-477. Redes Neuronales Artificiales.
Tarea 1 - Perceptrones Multicapa ó Redes FF

Prof. Ricardo Ñanculef & Carlos Valle
jnancu@inf.utfsm.cl & *cvalle@inf.utfsm.cl*

Temas

- Manipulación de dataframes en *pandas*.
- Estandarización y normalización de datos.
- Manipulación básica de matrices en *numpy*.
- Construcción de Perceptrónes Multicapa usando *keras*.
- Métodos de entrenamiento de Redes Neuronales Artificiales usando *keras*: Batch, Mini-batches, Online, Learning Rate, Momentum.
- Validación cruzada (cross-validation) usando *sklearn*.

Formalidades

- Equipos de trabajo de: 2 personas.*.
- Se debe preparar un (breve) Jupyter/IPython notebook que explique la actividad realizada y las conclusiones del trabajo.
- Se debe preparar una presentación de 20 minutos. Presentador será elegido aleatoriamente.
- Se debe mantener un respaldo de cualquier tipo de código utilizado, informe y presentación en Github.
- Fecha de entrega y discusión: Lunes 12 de Septiembre.
- Formato de entrega: envío de link Github al correo electrónico del ayudante (joaquin.velasquez@alumnos.inf.utfsm.cl), incluyendo a todos los profesores en copia y especificando asunto: [TallerXXX-CursoXXX-Semestre-Año].

*La modalidad de trabajo en equipo nos parece importante, porque en base a nuestra experiencia enriquece significativamente la experiencia de aprendizaje. Sin embargo, esperamos que esto no se transforme en una cruda división de tareas. Cada miembro del equipo debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

1 El XOR

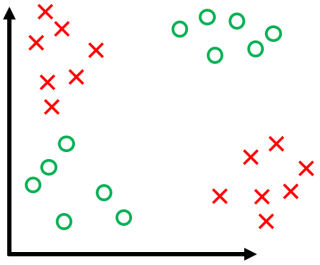


Fig. 1: Distribución deseada para la actividad 1. Los 2 colores representan 2 clases distintas.

(a) Escriba una función que genere (aleatoriamente) n datos etiquetados de la forma $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^2$, $y_i \in \{0, 1\}$, con una distribución de probabilidad que refleje la configuración linealmente inseparable que muestra la Fig.1[†]. Utilice esta función para generar un conjunto de 1000 datos de entrenamiento y 1000 datos de pruebas. El problema de clasificación obtenido se denomina en ocasiones "XOR". ¿Porqué?

(b) Demuestre experimentalmente que una neurona artificial individual[‡] no puede resolver satisfactoriamente el problema anterior. Puede utilizar la función de activación y el método de entrenamiento que prefiera. Sea convincente. Describa y explique lo que observa.

(c) Demuestre experimentalmente que un perceptron multicapas puede resolver satisfactoriamente el problema obtenido en (a). Puede utilizar la arquitectura y el método de entrenamiento que prefiera. Sea convincente. Describa y explique lo que observa.

2 Predicción del Precio de una Casa

En esta sección trabajaremos con un pequeño dataset conocido como *Boston Housing* que nos permitirá experimentar de modo más completo y exhaustivo con las técnicas bajo estudio. El problema consiste en predecir el precio de una casa en una zona/barrio de Boston (USA) a partir de una serie de atributos que describen el lugar que éste se ubica: tasa de criminalidad, proporción de zona residencial, proporción de zona industrial, si se encuentra junto al río ó no, contaminación atmosférica medida como la concentración de óxidos nítricos en el aire, etc. Para ver en detalle la descripción de la semántica asociada a los atributos de este problema, puede consultar `https://archive.ics.uci.edu/ml/datasets/Housing`.

[†]Por ejemplo: puede generar datos aleatorios distribuidos uniformemente en $[-1, +1] \times [-1, +1]$ para luego etiquetar aquellos ubicados en el primer y tercer cuadrante como 0 y aquellos ubicados en el primer y tercer cuadrante como 1.

[‡]Modelo discutido en clases

(a) Construya un dataframe con los datos a analizar descargando los datos desde la URL mantenida por los autores de [1]. Explique qué hacen las líneas 5 a 9.

```
1  import pandas as pd
2  url = 'http://mldata.org/repository/data/download/csv/regression-datasets-housing/'
3
4  df = pd.read_csv(url, sep=',',header=None, names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX',
5          'RM', 'AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT','MEDV'])
6  from sklearn.cross_validation import train_test_split
7  df_train,df_test= train_test_split(df,test_size=0.25, random_state=0)
```

(b) Describa brevemente el dataset utilizar.

```
1  df.shape
2  df.info()
3  df.describe()
```

(c) Normalice los datos antes de trabajar. Explique la importancia/conveniencia de realizar esta operación.

```
1  from sklearn.preprocessing import StandardScaler
2  scaler = StandardScaler().fit(df_train)
3  X_train_scaled = pd.DataFrame(scaler.transform(df_train),columns=df_train.columns)
```

(d) Muestre en un gráfico el error cuadrático (MSE) vs número de *epochs* de entrenamiento, para una red *feedforward* de 3 capas, con 200 unidades ocultas y función de activación *sigmoidal*. Entrene la red usando gradiente descendente estocástico con *learning rate* 0.2 y 300 epochs de entrenamiento, en el conjunto de entrenamiento y de test. Comente.

```
1  from keras.models import Sequential
2  from keras.layers.core import Dense, Activation
3  from keras.optimizers import SGD
4
5  model = Sequential()
6  model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
7  model.add(Activation('sigmoid'))
8  model.add(Dense(1, init='uniform'))
9  model.add(Activation('linear'))
10
11 sgd = SGD(lr=0.2)
12 model.compile(optimizer='sgd',loss='mean_squared_error')
13
14 hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300,
15         verbose=1, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
```

(e) Repita el paso anterior, utilizado 'Relu' como función de activación y compare con lo obtenido en d).

(f) Repita d) variando el learning rate. Comente.

```
1  import numpy as np
2
3  n_lr = 20
4  lear_rate = np.linspace(0,1,n_lr)
```

(g) Estime el error de predicción de los modelos d) y e) usando validación cruzada con un número de folds igual a $K = 5$ y $K = 10$. Recuerde que para que la estimación sea razonable debe ajustar los pesos del modelo de nuevo, cada vez que trabaja sobre un determinado fold. Mida el error real del modelo sobre el conjunto de pruebas, compare y concluya.

```
from sklearn import cross_validation

Xm = X_train_scaled.as_matrix()
ym = y_train_scaled.as_matrix()
kfold = cross_validation.KFold(len(Xm), 10)
cvscores = []
for i, (train, val) in enumerate(kfold):
    # create model
    model = Sequential()
    model.add(Dense(200, input_dim=Xm.shape[1], init='uniform'))
    model.add(Activation('relu'))
    model.add(Dense(1, init='uniform'))
    model.add(Activation('linear'))
    # Compile model
    sgd = SGD(lr=0.2)
    model.compile(optimizer='sgd',loss='mean_squared_error')
    # Fit the model
    model.fit(Xm[train], ym[train], nb_epoch=300)
    # evaluate the model
    scores = model.evaluate(Xm[val], ym[val])
    cvscores.append(scores)


mse_cv = np.mean(cvscores)
```

(h) Entrene el modelo obtenido en d) usando *progressive decay* Compare y comente.

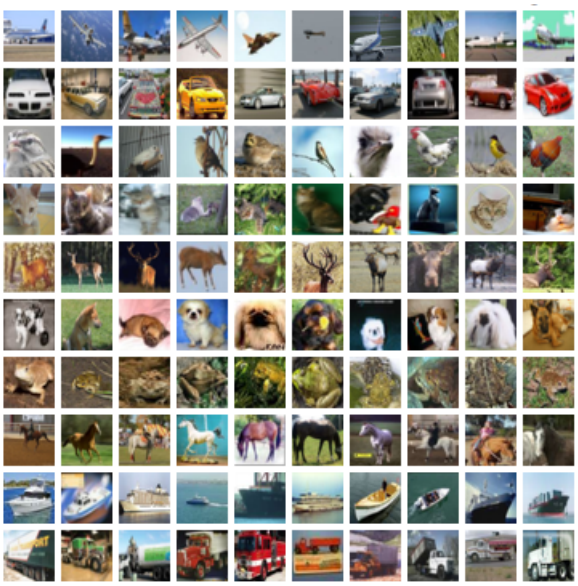
```
n_decay = 10
lear_decay = np.logspace(-6,0,n_decay)
sgd = SGD(lr=0.2, decay=1e-6)
```

(i) Entrene el modelo obtenido en d) usando momentum

```
n_decay = 21
momentum = np.linspace(0,1,n_decay)
sgd = SGD(lr=0.2,momentum=0.9)
```

(j) Entrene los modelos obtenidos en d) y e) cambiando el tamaÒo del batch. Compare SGD, batch y mini-batch.

```
n_batches = 21
batch_sizes = np.round(np.linspace(1,X_train_scaled.shape[0],n_batches))
model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), batch_size=50, nb_epoch=300)
```



3 Reconocimiento de Imágenes en CIFAR10

En esta sección trabajaremos con un dataset bastante conocido y utilizado por la comunidad para experimentar con reconocimiento de objetos en imágenes: CIFAR10. Se trata de un conjunto de 60.000 imágenes RGB de 32×32 pixeles que contiene 10 clases de objetos y 6000 ejemplos por clase. La versión utilizada se atribuye a A. Krizhevsky, V. Nair y G. Hinton [3] y viene separada en 50000 ejemplos de entrenamiento y 10000 casos de prueba. El conjunto de pruebas fue obtenido seleccionando 1000 imágenes aleatorias de cada clase. Los datos restantes han sido ordenados aleatoriamente y están organizados en 5 bloques de entrenamiento (batches). Las clases son mutuamente excluyentes y corresponden a las siguientes categorías: gatos, perros, ranas, caballos, pájaros, ciervos, aviones, automóviles, camiones y barcos.

Los datos asociados a esta actividad podrán ser obtenidos utilizando los siguientes comandos en la línea de comandos (sistemas UNIX)

```
wget http://octopus.inf.utfsm.cl/~ricky/data.tar.gz
tar -xzvf data.tar.gz
rm data.tar.gz
```

En la carpeta generada encontrarán 5 archivos denominados 'data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5' y 'test_batch' correspondientes a los 5 bloques de entrenamiento y al conjunto de pruebas respectivamente. Los archivos corresponden a diccionarios serializados de python y pueden ser "extraídos" utilizando la siguiente función:

```
def unpickle(file):
    import cPickle
    fo = open(file, 'rb')
    dict = cPickle.load(fo)
    fo.close()
    return dict
```

Una vez extraído, cada diccionario contendrá 2 elementos importantes: *data* y *labels*. El primer elemento (*data*) es un matriz de 10000×3072 (numpy array). Cada fila de esa matriz corresponde a una imagen RGB: los primeros 1024 valores vienen del canal R, los siguientes 1024 del canal G, y los últimos 1024 del canal B. Para cada canal, las imágenes han sido vectorizadas por filas, de modo que los primeros 32 valores del canal R corresponden a la primera fila de la imagen. Por otro lado, el elemento (*labels*) del diccionario contiene una lista de 1000 valores enteros entre 0 y 9 que identifican las clases antes enumeradas.

```
label_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',\
                        'frog', 'horse', 'ship', 'truck']
```

(a) Construya una función que cargue todos los bloques de entrenamiento y pruebas del problema CIFAR generando como salida: (i) dos matrices X_{tr}, Y_{tr}, correspondientes a las imágenes y etiquetas de entrenamiento, (ii) dos matrices X_t, Y_t, correspondientes a las imágenes y etiquetas de pruebas, y finalmente (iii) dos matrices X_v, Y_v, correspondientes a imágenes y etiquetas que se usarán como conjunto de validación, es decir para tomar decisiones de diseño acerca del modelo. Este último conjunto debe ser extraído desde el conjunto de entrenamiento original y no debe superar las 5000 imágenes.

```
import cPickle as pickle
import numpy as np
import os
from scipy.misc import imread

def load_CIFAR_one(filename):
  with open(filename, 'rb') as f:
    datadict = pickle.load(f)
    X = datadict['data']
    Y = datadict['labels']
    Y = np.array(Y)
    return X, Y

def load_CIFAR10(PATH):
  xs = []
  ys = []
  for b in range(1,6):
    f = os.path.join(PATH, 'data_batch_%d' % (b, ))
    X, Y = load_CIFAR_one(f)
    xs.append(X)
    ys.append(Y)
  Xtr = np.concatenate(xs)
  Ytr = np.concatenate(ys)
  del X, Y
  Xte, Yte = load_CIFAR_batch(os.path.join(PATH, 'test_batch'))
  return Xtr, Ytr, Xte, Yte
```

(b) Construya una función que escale apropiadamente las imágenes antes de trabajar. Experimente sólo centrando los datos y luego centrando y escalándolos como en actividades anteriores.

(c) Diseñe, entrene y evalúe una red neuronal con salida softmax para el problema CIFAR a partir de la representación original de las imágenes (píxeles RGB). Experimente con distintas arquitecturas y métodos de entrenamiento, midiendo el error de clasificación sobre el conjunto de validación. En base a esta última medida de desempeño, decida qué modelo, de entre todos los evaluados, evaluará finalmente en el conjunto de test. Reporte y discuta los resultados obtenidos.[§] Se espera que logre obtener un error de pruebas menor o igual a 0.5.

(d) Repita la actividad anterior, pero mejorando los atributos utilizados para representar las imágenes. Para esta parte, se distribuirá junto a esta tarea una función denominada *extract_features.py* que extraerá 2 tipos de representaciones sobre una imagen y conjunto de imágenes: (i) histogramas de tono [6], (ii) descriptores HOG [5]. Reporte y discuta los resultados obtenidos utilizando las distintas representaciones por separado o todas simultáneamente. La función *extract_features.py* estará definida

[§]Para empezar, puede considerar una pequeña arquitectura de 1 capa escondida con 50 neuronas y con funciones de activación ReLu. Como método de entrenamiento puede usar BP estocástico o en mini-batches, con *weight decay* y tasa de aprendizaje decreciente.

en un script denominado *top_level_features.py* y puede ser importada y utilizada como se muestra a continuación.

```
from top_level_features import hog_features
from top_level_features import color_histogram_hsv
from top_level_features import extract_features
Xtr, Ytr, Xte, Yte = load_CIFAR10("datasets/")
features = extract_features(Xtr,[hog_features]) #extrae hog features
features = extract_features(Xtr,[color_histogram_hsv]) #extrae histogramas de color
features = extract_features(Xtr,[hog_features, color_histogram_hsv]) #extrae todo
print Xtr.shape
print features.shape
```

References

[1] Hastie, T.; Tibshirani, R., Friedman, J. (2009), The Elements of Statistical Learning, Second Edition. Springer New York Inc.

[2] Bishop, Christopher M. (1995). Neural Networks for Pattern Recognition, Clarendon Press.

[3] Krizhevsky, A., Hinton, G. (2009). Learning multiple layers of features from tiny images.

[4] Harrison, D. and Rubinfeld, D. (1978). Hedonic prices and the demand for clean air, Journal of Environmental Economics and Management, 5, 81-102

[5] Dalal, N., Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE.

[6] Forsyth, D. A., Ponce, J. (2002). Computer vision: a modern approach. Prentice Hall Professional Technical Reference.

LaTeX