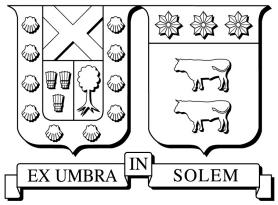


INF-477. Redes Neuronales Artificiales. I-2017.
Tarea 1 - Perceptrones Multicapa ó Redes Feed-Forward

Temas

- Diseño de Redes Feed-Forward para problemas de clasificación y regresión.
- Entrenamiento de Redes Feed-Forward vía GD y variantes (SGD, mini-batches).
- Momentum y estrategias de selección de paso en SGD.
- Evaluación de Redes Feed-Forward vía validación cruzada (cross-validation).
- Regularización de Redes Feed-Forward.

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: definida en calendario publicado en moodle.
- Formato de entrega: envío de link Github al correo electrónico del ayudante†, incluyendo al profesor en copia‡. Por favor especificar el siguiente asunto: [Tarea1-INF477-I-2017].

*Cada uno debe estar en condiciones de realizar una presentación y discutir sobre cada punto del trabajo realizado.

†joaquin.velasquez@alumnos.inf.utfsm.cl
‡jnancu@inf.utfsm.cl

1 Rol de las Capas Ocultas de una ANN

Como hemos discutido en clases, el modelo de neurona utilizando en la mayoría de las redes neuronales artificiales corresponde esencialmente a un discriminante lineal sobre el espacio original de atributos. Por lo tanto, si se entrena una neurona para resolver un problema cuya solución no es una función lineal de esos atributos debiésemos esperar un pobre rendimiento del modelo en la tarea. Un perceptrón multicapas en cambio agrega un nivel adicional de procesamiento entrenado para aprender automáticamente una representación adecuada al problema. En esta sección experimentaremos esta idea revisitando un problema "de juguete" denominado *moons*.

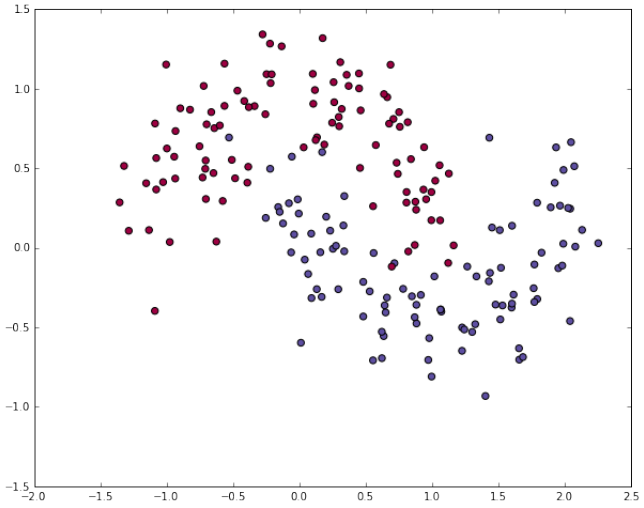


Fig. 1: Distribución deseada para la actividad 1. Los 2 colores representan 2 clases distintas.

(a) Implemente una función que genere 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 a la izquierda de la Fig.1 (se trata esencialmente de dos semi-círculos de radio $\sqrt{2}$, uno centrado en $(0.0, 0.5)$ y el otro en $(1.0, 0.0)$. Utilice la función para generar dos subconjuntos distintos de 500 puntos a ser usados como conjuntos de entrenamiento y pruebas (por el objetivo de la actividad, en este caso no utilizaremos un conjunto de validación). Haga un gráfico que muestre los datos obtenidos.

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise=0.20)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```

(b) Demuestre experimentalmente que una neurona artificial individual no puede resolver satisfactoriamente el problema anterior. Puede utilizar la función de activación (no-lineal) 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 Back-propagation (BP) from Scratch

BP (Back-propagation) es sin duda el paradigma dominante para entrenar redes neuronales feed-forward. En redes grandes, diseñadas para problemas reales, implementar BP eficientemente puede ser una tarea delicada que puede ser razonable delegar a una librería especializada. Sin embargo, construir BP *from scratch* es muy útil con fines pedagógicos.

(a) Escriba un programa que permita entrenar una red FF con 1 capa escondida (H neuronas) y O neuronas de salida, sin usar librerías, excepto eventualmente *numpy* para implementar operaciones básicas de algebra lineal. Por simplicidad, asuma que todas las neuronas implementan una función de activación diferenciable y que la función de error (loss function) también lo es. Especifique explícitamente las funciones anteriores, así como sus gradientes. Escriba funciones para: (i) dar valores iniciales a los pesos de la red, (ii) implementar el *forward pass*, (iii) implementar el *backward pass* y (iv) implementar la rutina principal de entrenamiento, adoptando, por simplicidad, la variante cíclica de SGD (un ejemplo a la vez, pero iterando cíclicamente sobre el conjunto de entrenamiento) con una tasa de aprendizaje y número de ciclos fijos (epochs).

(b) Escriba una función que permita hacer predicciones mediante una red FF con 1 capa escondida (H neuronas) y O neuronas de salida, sin usar librerías, excepto eventualmente *numpy*. Escriba una función vectorizada que implemente el *forward pass* sobre un conjunto de n_{test} ejemplos.

(c) Demuestre que sus programas funcionan en un problema de clasificación, eligiendo funciones de error y de activación apropiadas. Para esto utilice el dataset *seeds*, disponible en UCI [1] y correspondiente a la clasificación de distintos tipos de semillas (3 clases) (recuerde que usualmente es conveniente normalizar los datos antes de trabajar con el modelo).

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00236/seeds_dataset.txt'
df = pd.read_csv(url, sep=r'\s+',header=None)
X_train = df.ix[:,0:6]
y_train = df.ix[:,7]
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
```

Para evaluar los resultados, construya un gráfico correspondiente al error de clasificación versus número de epochs, utilizando sólo el conjunto de entrenamiento (el objetivo de esta sección es familiarizarse con el algoritmo BP, no encontrar la mejor red). Grafique también la evolución de la función objetivo utilizada para el entrenamiento.

(d) Construya, sin usar librerías, excepto eventualmente *numpy* para implementar operaciones básicas de algebra lineal, una variante de su programa anterior que entrene la red utilizando *weight-decay*.

3 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`.

(a) Construya un dataframe con los datos a analizar descargando los datos desde la URL mantenida por los autores de [2]. Describa brevemente el dataset utilizar.

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

(b) Normalice los datos antes de trabajar. Determine la conveniencia de realizar esta operación.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(df_train)
X_train_scaled = pd.DataFrame(scaler.transform(df_train),columns=df_train.columns)
y_train = df_train.pop('MEDV')
```

(c) 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 tasa de aprendizaje (*learning rate*) 0.01 y 300 epochs de entrenamiento, en el conjunto de entrenamiento y de test. Comente. Si observara divergencia durante el entrenamiento, determine si esto ocurre para cada repetición del experimento.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(200, input_dim=X_train_scaled.shape[1], init='uniform'))
model.add(Activation('sigmoid'))
```

```
model.add(Dense(1, init='uniform'))
model.add(Activation('linear'))

sgd = SGD(lr=0.01)
model.compile(optimizer=sgd,loss='mean_squared_error')

hist = model.fit(X_train_scaled.as_matrix(), y_train_scaled.as_matrix(), nb_epoch=300,
        verbose=1, validation_data=(X_test_scaled.as_matrix(), y_test_scaled.as_matrix()))
```

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

(e) Repita c) y d) variando la tasa de aprendizaje (learning rate) en un rango sensible. Comente. Si observara divergencia durante el entrenamiento, determine si esto ocurre para cada repetición del experimento.

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

(f) Estime el error de predicción de los modelos c) y d) 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)
```

(g) Entrene los modelos considerados en c) y 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)
```

(h) Entrene los modelos considerados en c) y d) usando momentum. Experimente usando momentum clásico y momentum de Nesterov. ¿Observa un mejor resultado final? ¿Observa una mayor velocidad de convergencia sobre el dataset de entrenamiento? ¿Sobre el dataset de pruebas?

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

(i) Entrene los modelos considerados en c) y d) utilizando SGD en mini-batches. Experimente con diferentes tamaños del batch. Comente.

```
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)
```

(j) Entrene los modelos obtenidos en c) y d) utilizando estrategias modernas para adaptar la tasa de aprendizaje. Compare los desempeños de *adagrad*, *adadelta*, *RMS prop* y *adam*. ¿Se observa en algún caso un mejor resultado final? ¿Se observa en algún caso una mayor velocidad de convergencia sobre el dataset de entrenamiento? ¿Sobre el dataset de pruebas?

```
from keras.optimizers import SGD, Adam, RMSprop, Adagrad, Adadelta
moptimizer = Adagrad(lr=0.01)
model.compile(optimizer=moptimizer)
model.fit(X_train_scaled.as_matrix(),y_train_scaled.as_matrix())
```

(k) Entrene los modelos obtenidos en c) y d) utilizando regularizadores ℓ_1 y ℓ_2 (weight decay). Compare los desempeños de prueba obtenidos antes y después de regularizar. Experimente con distintos valores del parámetro de regularización y comente. Luego, agregue una capa al modelo y aumente significativamente el número de neuronas escondidas en cada capa. Evalúe nuevamente el efecto de los regularizadores.

```
model = Sequential()
#la regularization se debe incorporar a cada capa separadamente
idim=X_train_scaled.shape[1]
model.add(Dense(200,input_dim=idim,init='uniform',W_regularizer=l2(0.01)))
model.add(Activation('sigmoid'))
model.add(Dense(1, init='uniform',W_regularizer=l2(0.01)))
model.add(Activation('linear'))
```

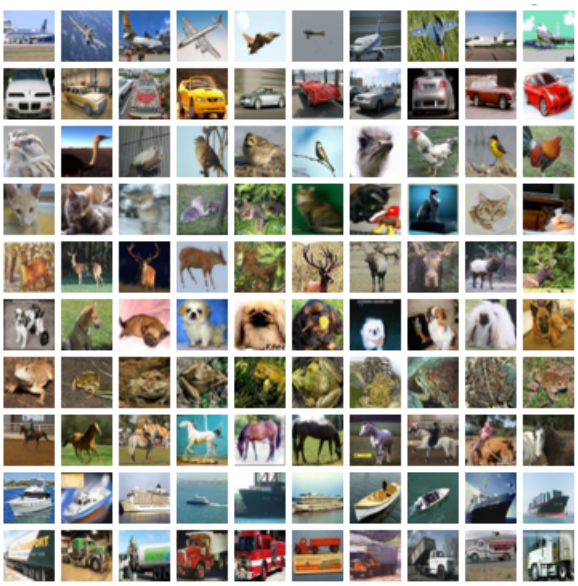
(l) Entrene los modelos obtenidos en c) y d) utilizando *Dropout*. Compare los desempeños de prueba obtenidos antes y después de regularizar. Experimente con distintos valores del parámetro de regularización y comente. Luego, agregue una capa al modelo y aumente significativamente el número de neuronas escondidas en cada capa. Evalúe nuevamente el efecto de la regularización, evaluando también el efecto de regularizar sólo la capa inicial versus la alternativa de regularizar todas las capas.

```
from keras.layers import Dropout
model = Sequential()
model.add(Dropout(0.2))
idim=X_train_scaled.shape[1]
model.add(Dense(200,input_dim=idim,init='uniform'))
model.add(Activation('sigmoid'))
model.add(Dense(1, init='uniform'))
model.add(Activation('linear'))
```

(m) Fijando todos los demás hiper-parámetros del modelo definido en c) y d), utilice validación cruzada para determinar el mejor valor correspondiente a un parámetro que usted elija (tasa de aprendizaje, número de neuronas, parámetro de regularización, etc).

4 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 [4] 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 [7], (ii) descriptores HOG [6]. 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] M. Lichman. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

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

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

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

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

[6] 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.

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