TareaRedesNeuronales

November 17, 2018

CONTROL INTELIGENTE
Maestro: Dr. Agustín Flores Novelo
Autor: Ing. José Alfonso Ureña Pajón

1 Objetivo:

Familiarizarse con la Red Neuronal BackPropagation.

2 Desarrollo:

Implementar una estructura nueronal backpropagation para reconocer las vocales (minúsculas)

1. Generar los patrones de entrada (vocales) en matrices de 5x7.
2. Generar el target como una matriz vector columna donde el '1' signifique la activación de la volcal correspondiente a la entrada.
3. Generar una estructura con 10 neuronas en la capa oculta.
4. Haga simulaciones para los diversos algoritmos de aprendizaje y compare sus resultados (que algoritmo converge mas rápido)
5. Repetir el inciso 4 para los siguientes casos
    - 5 neuronas en la capa oculta
    - 15 neuronas en la capa oculta
6. Del inciso en el punto 5 obtener las matrices de aprendizaje y elaborar un programa para obtener la respuesta de la Red Neuronal.
7. Investigar la definición de los diversos algoritmos usados.

3 Definición de los paquetes que se van a usar y de funciones auxiliares:

```
In [1]: # Se importan las librerias principales Numpy, MLPClassifier, matplotlib
        # se especifica que las gráficas son locales

        import numpy as np
        from sklearn.neural_network import MLPClassifier
        import matplotlib.pyplot as plt
        import random
```

```
%matplotlib inline

# Definición de funciones auxiliares

# Muestra los caracteres
def mostrarImagenes( vocales, fils, cols, titles ):
    x = 1
    withTitles = True
    if titles == []: withTitles = False
    plt.figure(figsize=(10,5))
    for vocal in vocales:
        plt.subplot(fils, cols, x)
        plt.axis('off')
        plt.xlim(-2, 7)
        plt.ylim(8, -1)
        plt.imshow(vocal, cmap='copper', interpolation='nearest')
        if withTitles: plt.title(titles[x-1])
        x = x + 1
    #plt.tight_layout()
    plt.show()
    return

# Muestra las graficas de comparacion de los clasificadores
def graficarAprendizaje(X, y, numNeurons, max_iter, name):
    mlps = []
    for label, param in zip(labels, params):
        print("training: %s" % label)
        mlp = MLPClassifier(verbose=0, random_state=0,hidden_layer_sizes=(numNeurons),
                            max_iter=max_iter,**param)
        mlp.fit(X, y)
        mlps.append(mlp)
        #print("Training set score: %f" % mlp.score(X, y))
        print("Training set loss: %f" % mlp.loss_)
    fig, ax = plt.subplots(figsize=(15, 10))

    for mlp, label, args in zip(mlps, labels, plot_args):
        ax.plot(mlp.loss_curve_, label=label, **args)
    ax.set_title(name)
    plt.legend(ax.get_lines(), labels, ncol=3, loc="upper center")
    plt.show()
    return mlps

# Genera patron de ruido aleatorio
def generarPatronDeRuido(numPixeles):
    px = []
    ruido = np.zeros(35).astype(int)
    for x in range(0,numPixeles):
        while True:
```

```
                tempPx = random.randint(0,34)
                if not tempPx in px:
                    ruido[tempPx] = 1
                    px.append(tempPx)
                    break
        return ruido

    # Genera reporte para un solo clasificador
    def reporteParcial(result):
        i = 0
        ok = 0
        for prediccion in result:
            onesIndexes = np.nonzero(prediccion)[0]
            v = vocalesCh[i]
            print("original: %s / " % v, end='')
            if len(onesIndexes) == 1:
                vD = vocalesCh[onesIndexes[0]]
                print("detectada: %s" % vD, end='' )
                if v == vD:
                    print(" -->  < OK >")
                    ok = ok + 1
                else:
                    print(" -->  < XX >")
            else:
                print("NO UNA VOCAL -->  < XX >")
            i = i + 1
        print("CORRECTOS: %d/5, EFECTIVIDAD: %.1f%% " % (ok, ok/5.0*100))
        print()
        return

    # Recibe un array de clasificadores y genera reporte de todos
    def printReporte(clasificadores, etiquetas):
        for label, cl in zip(etiquetas, clasificadores):
            print("clasificador: %s" % label)
            result = cl.predict(vocalesRuido)
            reporteParcial(result)
```

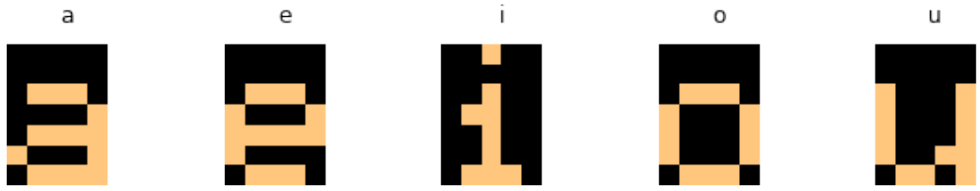
4 Generación de patrones de entrada (vocales)

```
In [2]: a2d = np.array([[0, 0, 0, 0, 0],
                       [0, 0, 0, 0, 0],
                       [0, 1, 1, 1, 0],
                       [0, 0, 0, 0, 1],
                       [0, 1, 1, 1, 1],
                       [1, 0, 0, 0, 1],
                       [0, 1, 1, 1, 1]])
```

```
e2d = np.array([[0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 1, 1, 1, 0],
                [1, 0, 0, 0, 1],
                [1, 1, 1, 1, 1],
                [1, 0, 0, 0, 0],
                [0, 1, 1, 1, 0]])

i2d = np.array([[0, 0, 1, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0],
                [0, 1, 1, 0, 0],
                [0, 0, 1, 0, 0],
                [0, 0, 1, 0, 0],
                [0, 1, 1, 1, 0]])

o2d = np.array([[0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 1, 1, 1, 0],
                [1, 0, 0, 0, 1],
                [1, 0, 0, 0, 1],
                [1, 0, 0, 0, 1],
                [0, 1, 1, 1, 0]])

u2d = np.array([[0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [1, 0, 0, 0, 1],
                [1, 0, 0, 0, 1],
                [1, 0, 0, 0, 1],
                [1, 0, 0, 1, 1],
                [0, 1, 1, 0, 1]])
```

4.1 Visualización de los patrones generados

In [3]:
```
vocalesCh = [ 'a', 'e', 'i', 'o', 'u' ]
vocales = [a2d, e2d, i2d, o2d, u2d ]
mostrarImagenes(vocales, 1, 5, vocalesCh)
```



5 Generacion de patrones de entrada y salida (punto 2)

In [4]:
```
# different learning rate schedules and momentum parameters

        a = a2d.flatten()
        e = e2d.flatten()
        i = i2d.flatten()
        o = o2d.flatten()
        u = u2d.flatten()

        X = [a,e,i,o,u]
        y = [[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],[0,0,0,1,0],[0,0,0,0,1]]
```

6 Comparación de Algoritmos/Clasificadores (puntos 3, 4, 5)

Se codifica una lista correspondientes a clasificadores propuestos en la pagina de scikit-learn, junto con respectivas listas auxiliares que sirven para generar información relacionada con las gráficas.

NOTA.- Donde se marca un error es porque los epoches propuestos no fueron suficientes para que el clasificador converja, esta información se puede percibir claramente en las gráficas.

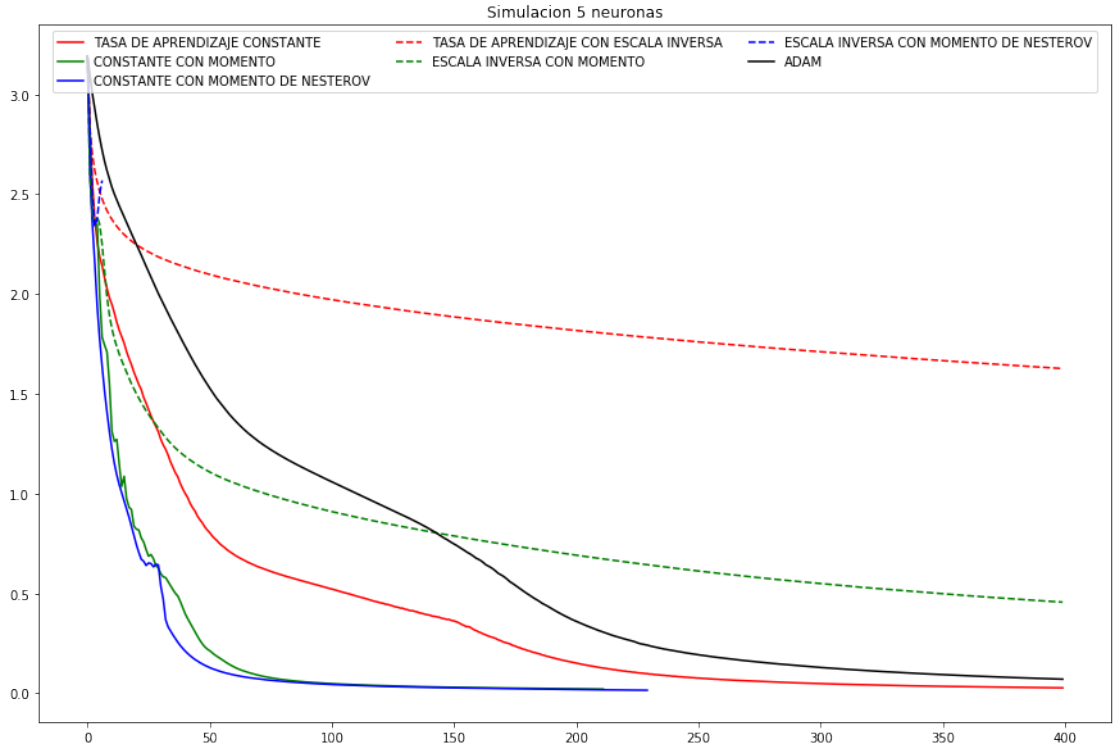
In [5]:
```
# Se definen los conjuntos necesarios para la comparación de cada uno de los clasificado
        params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0,
                   'learning_rate_init': 0.2},
                  {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
                   'nesterovs_momentum': False, 'learning_rate_init': 0.2},
                  {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
                   'nesterovs_momentum': True, 'learning_rate_init': 0.2},
                  {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': 0,
                   'learning_rate_init': 0.2},
                  {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
                   'nesterovs_momentum': True, 'learning_rate_init': 0.2},
                  {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
                   'nesterovs_momentum': False, 'learning_rate_init': 0.2},
                  {'solver': 'adam', 'learning_rate_init': 0.01}]

        labels = ["TASA DE APRENDIZAJE CONSTANTE", "CONSTANTE CON MOMENTO",
                  "CONSTANTE CON MOMENTO DE NESTEROV",
                  "TASA DE APRENDIZAJE CON ESCALA INVERSA", "ESCALA INVERSA CON MOMENTO",
                  "ESCALA INVERSA CON MOMENTO DE NESTEROV", "ADAM"]
        plot_args = [{'c': 'red', 'linestyle': '-'},
                     {'c': 'green', 'linestyle': '-'},
                     {'c': 'blue', 'linestyle': '-'},
                     {'c': 'red', 'linestyle': '--'},
                     {'c': 'green', 'linestyle': '--'},
                     {'c': 'blue', 'linestyle': '--'},
                     {'c': 'black', 'linestyle': '-'}]

        clasifiers5Neurons = graficarAprendizaje(X, y, 5, 400, "Simulacion 5 neuronas")
```

```
        clasifiers10Neurons = graficarAprendizaje(X, y, 10, 200, "Simulacion 10 neuronas")
        clasifiers15Neurons = graficarAprendizaje(X, y, 15, 100, "Simulacion 15 neuronas")
```

```
training: TASA DE APRENDIZAJE CONSTANTE
Training set loss: 0.026144
training: CONSTANTE CON MOMENTO
```

```
/home/ppurena/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```

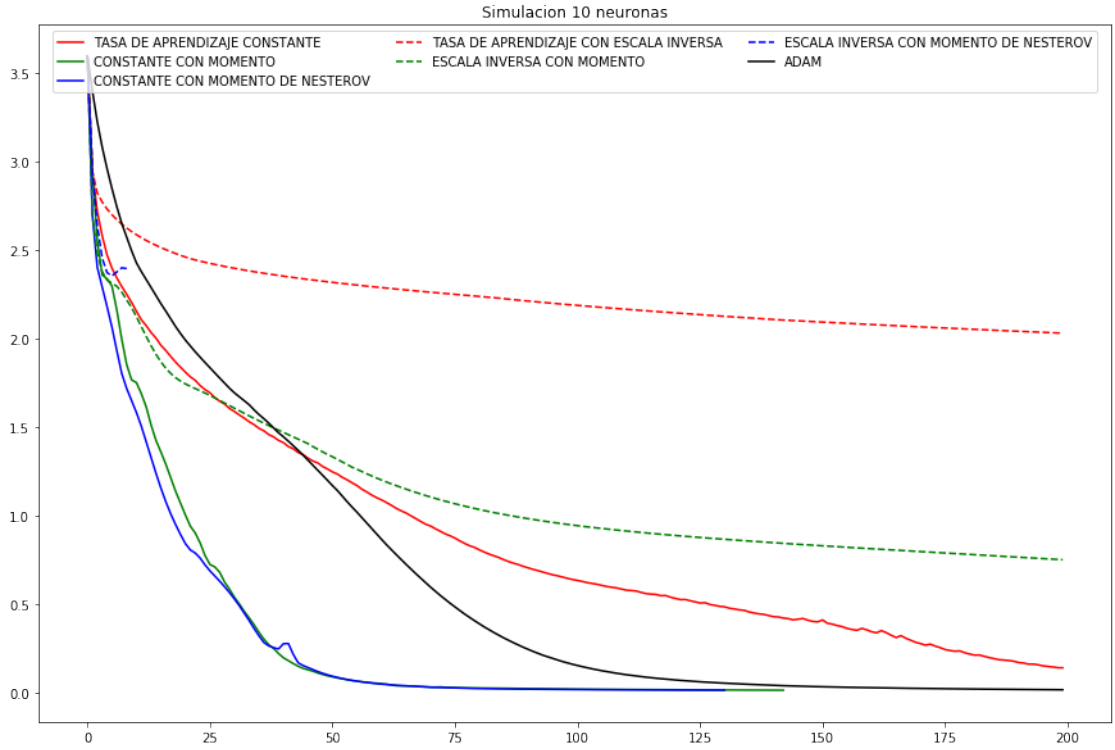
```
Training set loss: 0.020294
training: CONSTANTE CON MOMENTO DE NESTEROV
Training set loss: 0.014273
training: TASA DE APRENDIZAJE CON ESCALA INVERSA
Training set loss: 1.626314
training: ESCALA INVERSA CON MOMENTO
Training set loss: 0.455680
training: ESCALA INVERSA CON MOMENTO DE NESTEROV
Training set loss: 2.569188
training: ADAM
Training set loss: 0.069656
```



```
training: TASA DE APRENDIZAJE CONSTANTE
Training set loss: 0.141417
training: CONSTANTE CON MOMENTO
Training set loss: 0.014991
training: CONSTANTE CON MOMENTO DE NESTEROV
Training set loss: 0.014874
training: TASA DE APRENDIZAJE CON ESCALA INVERSA
```

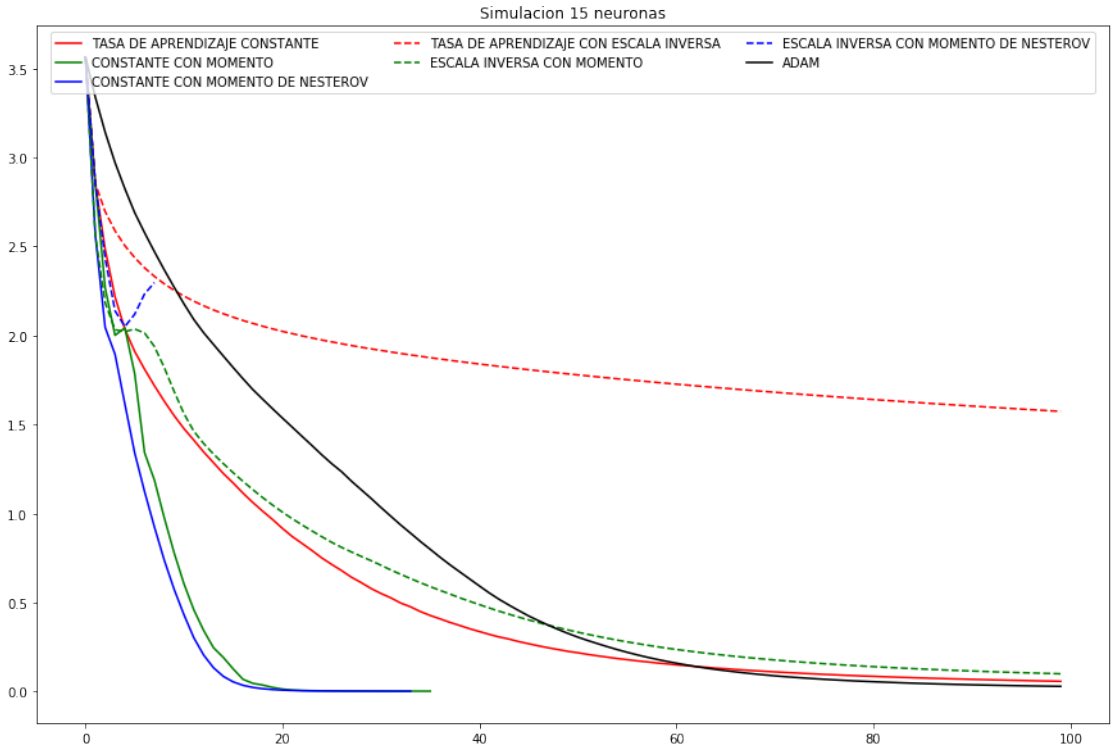
```
/home/ppurena/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```

```
Training set loss: 2.029087
training: ESCALA INVERSA CON MOMENTO
Training set loss: 0.751024
training: ESCALA INVERSA CON MOMENTO DE NESTEROV
Training set loss: 2.393224
training: ADAM
Training set loss: 0.017629
```




```
training: TASA DE APRENDIZAJE CONSTANTE
Training set loss: 0.058022
```

```
training: CONSTANTE CON MOMENTO
Training set loss: 0.002488
training: CONSTANTE CON MOMENTO DE NESTEROV
Training set loss: 0.002133
training: TASA DE APRENDIZAJE CON ESCALA INVERSA
Training set loss: 1.574069
training: ESCALA INVERSA CON MOMENTO
Training set loss: 0.100175
training: ESCALA INVERSA CON MOMENTO DE NESTEROV
Training set loss: 2.296936
training: ADAM
Training set loss: 0.029616


/home/ppurena/anaconda3/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron
  % self.max_iter, ConvergenceWarning)
```



7 Simulaciones

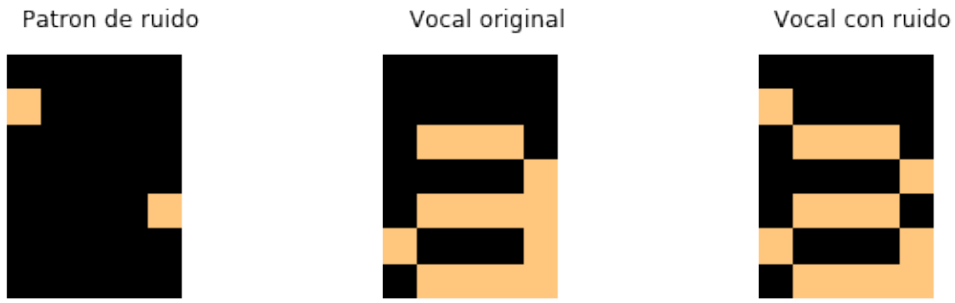
Para las simulaciones se genera un patron o mascara de pixeles ditribuidos aleatoriamente a forma de ruido y que al sobreponerse en la vocal original cambia el valor del pixel (si es 1 pasa a 0 y

viceversa), a continuación se muestra el código y las gráficas correspondientes de las vocales con ruido.

Estas vocales con ruido son los datos que se van a ingresar en las neuronas entrenadas correspondientes a los incisos anteriores y de los cuales se va a verificar su desempeño para cada uno de los clasificadores.

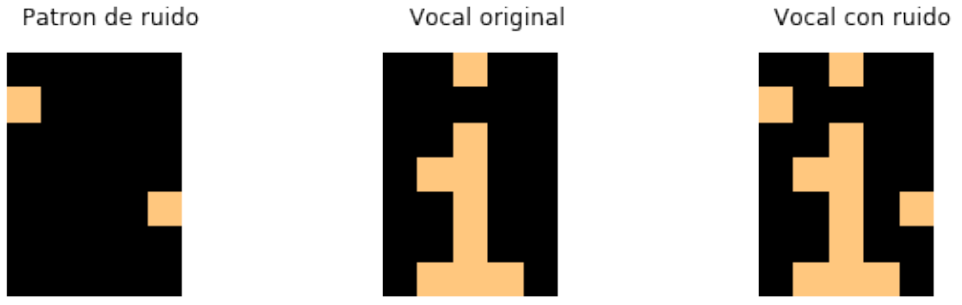
```
In [6]: # Generación del ruido aleatorio
        ruido = generarPatronDeRuido(2)

        # Se agrega el ruido a las vocales
        vocalesRuido = (X + ruido)% 2
        titulos = ['Patron de ruido', 'Vocal original', 'Vocal con ruido']

        # Generación de gráficas
        i = 0
        for v,vr in zip(X,vocalesRuido):
            print("Vocal %s" % vocalesCh[i])
            setVocales = [np.reshape(ruido,(7,5)), np.reshape(v,(7,5)), np.reshape(vr,(7,5))]
            mostrarImagenes(setVocales, 1, 3,titulos)
            i = i + 1
```

```
Vocal a
```

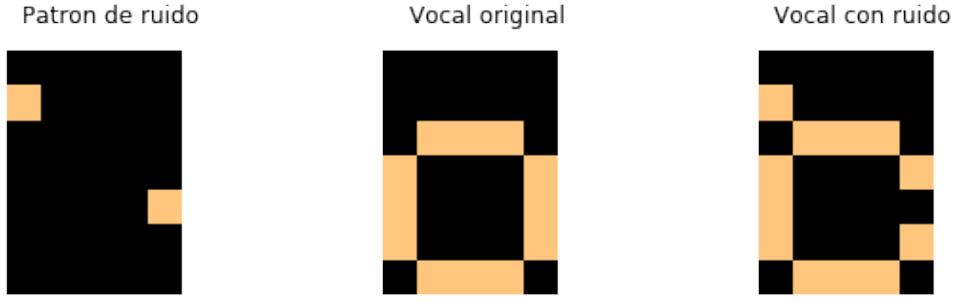



```
Vocal e
```

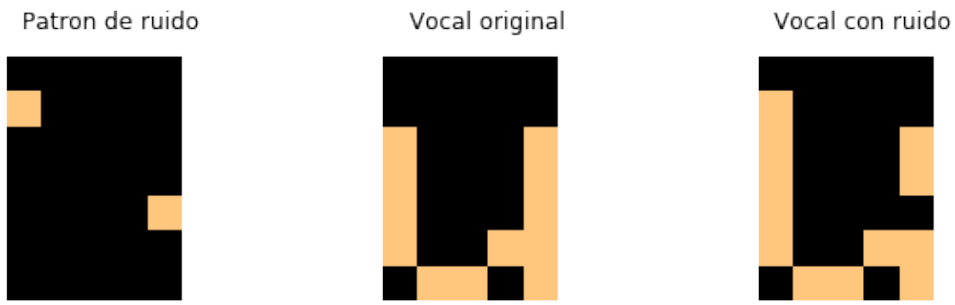
Vocal i



Vocal o



```
Vocal u
```



7.1 Simulacion para clasificadores con 5 neuronas en la capa interna

```
In [7]: printReporte(clasifiers5Neurons, labels)

clasificador: TASA DE APRENDIZAJE CONSTANTE
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: CONSTANTE CON MOMENTO
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / NO UNA VOCAL -->  < XX >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 4/5, EFECTIVIDAD: 80.0%

clasificador: CONSTANTE CON MOMENTO DE NESTEROV
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / NO UNA VOCAL -->  < XX >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 4/5, EFECTIVIDAD: 80.0%
```

```
clasificador: TASA DE APRENDIZAJE CON ESCALA INVERSA
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / NO UNA VOCAL -->  < XX >
original: o / NO UNA VOCAL -->  < XX >
original: u / detectada: u -->  < OK >
CORRECTOS: 1/5, EFECTIVIDAD: 20.0%

clasificador: ESCALA INVERSA CON MOMENTO
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: ESCALA INVERSA CON MOMENTO DE NESTEROV
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / NO UNA VOCAL -->  < XX >
original: o / NO UNA VOCAL -->  < XX >
original: u / NO UNA VOCAL -->  < XX >
CORRECTOS: 0/5, EFECTIVIDAD: 0.0%

clasificador: ADAM
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / NO UNA VOCAL -->  < XX >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 4/5, EFECTIVIDAD: 80.0%
```

7.2 Simulacion para clasificadores con 10 neuronas en la capa interna

```
In [8]: printReporte(clasifiers10Neurons, labels)
```

```
clasificador: TASA DE APRENDIZAJE CONSTANTE
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: CONSTANTE CON MOMENTO
```

```
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: CONSTANTE CON MOMENTO DE NESTEROV
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: TASA DE APRENDIZAJE CON ESCALA INVERSA
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / detectada: i -->  < OK >
original: o / NO UNA VOCAL -->  < XX >
original: u / NO UNA VOCAL -->  < XX >
CORRECTOS: 1/5, EFECTIVIDAD: 20.0%

clasificador: ESCALA INVERSA CON MOMENTO
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / NO UNA VOCAL -->  < XX >
original: u / detectada: u -->  < OK >
CORRECTOS: 4/5, EFECTIVIDAD: 80.0%

clasificador: ESCALA INVERSA CON MOMENTO DE NESTEROV
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / NO UNA VOCAL -->  < XX >
original: o / NO UNA VOCAL -->  < XX >
original: u / NO UNA VOCAL -->  < XX >
CORRECTOS: 0/5, EFECTIVIDAD: 0.0%

clasificador: ADAM
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%
```

7.3 Simulacion para clasificadores con 15 neuronas en la capa interna

In [9]: printReporte(clasifiers15Neurons, labels)

```
clasificador: TASA DE APRENDIZAJE CONSTANTE
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: CONSTANTE CON MOMENTO
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: CONSTANTE CON MOMENTO DE NESTEROV
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: TASA DE APRENDIZAJE CON ESCALA INVERSA
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / detectada: i -->  < OK >
original: o / NO UNA VOCAL -->  < XX >
original: u / NO UNA VOCAL -->  < XX >
CORRECTOS: 1/5, EFECTIVIDAD: 20.0%

clasificador: ESCALA INVERSA CON MOMENTO
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%

clasificador: ESCALA INVERSA CON MOMENTO DE NESTEROV
original: a / NO UNA VOCAL -->  < XX >
original: e / NO UNA VOCAL -->  < XX >
original: i / NO UNA VOCAL -->  < XX >
original: o / NO UNA VOCAL -->  < XX >
```

```
original: u / NO UNA VOCAL -->  < XX >
CORRECTOS: 0/5, EFECTIVIDAD: 0.0%

clasificador: ADAM
original: a / detectada: a -->  < OK >
original: e / detectada: e -->  < OK >
original: i / detectada: i -->  < OK >
original: o / detectada: o -->  < OK >
original: u / detectada: u -->  < OK >
CORRECTOS: 5/5, EFECTIVIDAD: 100.0%
```

8 Definición de los diversos algoritmos usados

El **MLP** (Multilayer Perceptron) se entrena usando "Descenso de Gradiente Estocástico" (SGD, por sus siglas en ingles), Adam, o L-BFGS (Limited-memory BFGS).

1. **El Descenso por Gradiente Estocástico (SGD)** es un método optimizado para solucionar problemas de optimización sin restricciones. Eseé actualiza los parámetros utilizando el gradiente de la función de pérdida con respecto a un parámetro que necesita adaptación considerando un solo evento de entrenamiento a la vez, en otras palabras, SGD trata de encontrar minimos o maximos por iteración. Tanto en las estimaciones estadisticas como en el aprendizaje por computadora se considera el problema de minimizar la función objetivo que tiene la forma de:

   $$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

   Donde el parametro w que minimiza Q(w) es el que se estima y cada sumando Qi esta asociado con la i-esima observación en el conjunto de datos usado para el entrenamiento. Cuando se usa el algoritmo SGD, el gradiente efectivo de Q(w) es aproximado por un gradiente de un solo ejemplo:

   $$w := w - \eta \nabla Q_i(w).$$

   Conforme el algoritmo barre el conjunto de datos, ejecuta una actualización de la formula anterior para cada ejemplo de entrenamiento. Varios iteraciones pueden ser ejecutadas hasta que el algoritmo converja.

2. **Adaptive Moment Estimation (Adam)** es una actualización del algoritmo RMSProp (Root Mean Square Propagation). Para este algoritmo promedios del gradiente y el segundo momento del gradiente son usados. Adam es similar a SGD en el sentido de que es un optimizador estocástico, pero puede ajustar automáticamente la cantidad para actualizar los parámetros basándose en estimaciones adaptativas de los momentos de orden inferior. Algunas de las ventajas de Adam son:

   - Fácil de implementar.
   - Computacionalmente eficiente.
   - Requiere poca memoria.

- Invariante para la reescala diagonal de los gradientes.
- Apropiado para problemas de gran tamaño en términos de datos y/o parámetros.
- Apropiado para objetivos no estacionarios.
- Apropiado para problemas con pendientes muy ruidosas o escasas.
- Los hiperparámetros tienen una interpretación intuitiva y normalmente requieren poca afinación.

3. **Limited-memory BFGS (L-BFGS)** es un algoritmo de optimización en la familia de métodos cuasi-Newtonianos que se aproxima al algoritmo de Broyden-Fletcher-Goldfarb-Shanno (BFGS) usando una cantidad limitada de memoria de la computadora. Es un algoritmo popular para la estimación de parámetros en el aprendizaje automático.El problema objetivo del algoritmo es minimizar f(x) sobre valores no restringidos del vector real "x" donde f es una funció escalar diferenciable. Al igual que el algoritmo BFGS, L-BFGS utiliza una estimación de la matriz hessiana inversa para dirigir su búsqueda a través del espacio variable, pero donde BFGS almacena una densa aproximación nŒn al Hessian inverso (siendo n el número de variables en el problema), L-BFGS almacena sólo unos pocos vectores que representan la aproximación implícitamente. Debido a su requerimiento de memoria lineal resultante, el método L-BFGS es particularmente adecuado para problemas de optimización con un gran número de variables. En lugar del Hk Hessian inverso, L-BFGS mantiene una historia de las últimas actualizaciones m de la posición x y del gradiente f(x), donde generalmente el tamaño m de la historia puede ser pequeño (a menudo m<10). Estas actualizaciones se utilizan para realizar operaciones que requieren el producto Hk-vector.

9 Conclusiones

Con los avances tecnológicos y especialmente en el ramo de la computación, todos como sociedad tenemos mucho que ganar con el uso de las redes neuronales. Su capacidad de aprender a partir de muestras de datos reales obtenidos por medio de la experiencia, las convierte en herramientas muy flexibles y poderosas. Parte de su gran poder radica en no requerir del diseño de un algoritmo para realizar una tarea específica, liberando al diseñador de la necesidad de contar con conocimientos particulares asociados con los mecanismos del problema a resolver. También son muy adecuados para sistemas de tiempo real debido a su rápida respuesta y tiempos computacionales debidos a su naturaleza paralela. Por lo tanto conocer como funcionan y contar con ellas como una herramientas más, nos permite tener una perspectiva distinta de solución de problemas.