In [1]:

```
%matplotlib inline
```

MLP

Uma das formas mais tradicionais de redes neurais é a MLP, ou Multilayer Perceptron. A MLP é um algoritmo de aprendizado supervisionado que aprende uma função que relaciona entradas e saídas através do seu treinamento em um conjunto de dados.

Tipicamente, uma MLP é formada por:

- uma camada de entrada, que corresponde aos M atributos a serem usados para a classificação
- uma camada de saída, que possui um neurônio para cada uma das N classes em que a amostra de atributos será classificada
- uma ou mais camadas ocultas de neurônios, responsáveis por tornar a MLP um classificador não linear

A instanciação de uma MLP

Para criarmos uma MLP no `scikit-learn` usamos a classe `MLPClassifier` do pacote `sklearn.neural_network` através do código:

```
from sklearn.neural_network import MLPClassifier
```

A instanciação de um objeto representando a MLP depende da definição de **hiperparâmetros**, ou seja parâmetros da rede que não serão aprendidos durante a fase de treinamento. Até agora, os hiperparâmetros que aprendemos são:

1. Número de camadas ocultas

Indica quandas camadas ocultas serão usadas na rede neural. Há provas matemáticas de que uma rede do tipo MLP com função de ativação sigmóide pode aproximar qualquer função contínua com apenas uma única camada oculta de neurônios, porém para funções descontínuas, duas são necessárias (ver livro do Russel, pág. 720). Na prática, para dados simples, não precisamos de mais do que uma camada, e partimos para a decisão do número de neurônios nessa camada. O inconveniente dessa abordagem é que o número de neurônios necessários para realizar a classificação corretamente pode ter que crescer muito dependendo da complexidade dos dados.

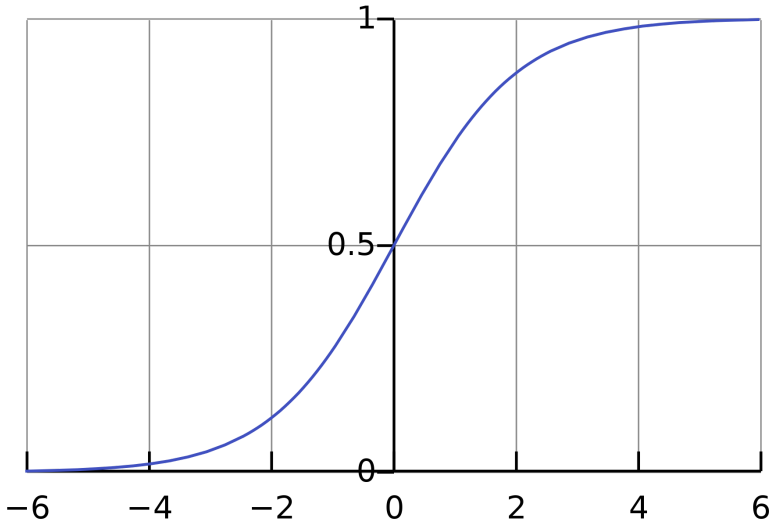
2. Número de neurônios por camada oculta

No caso de escolhermos usar apenas uma camada oculta, é preciso escolher o número de neurônios a ser empregado. Há algumas "receitas de bolo" para essa decisão, porém a melhor forma de determinar esse parâmetro é realizar diversos testes com os dados de treinamento, através da *validação cruzada*. Uma dessas receitas de bolo indica que o número de neurônios da camada oculta seja o dobro do número de elementos de entrada mais um, que nós podemos usar como ponto de partida para a realização de testes.

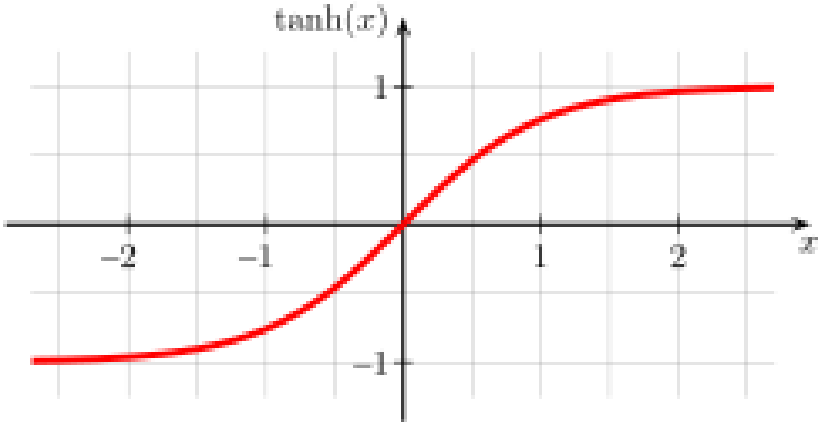
3. Função de ativação $f(a)$

A função de ativação processa a entrada líquida do neurônio, resultante do somatório das entradas multiplicadas pelos pesos. Para a tarefa de classificação, as funções de ativação mais comumente empregadas são:

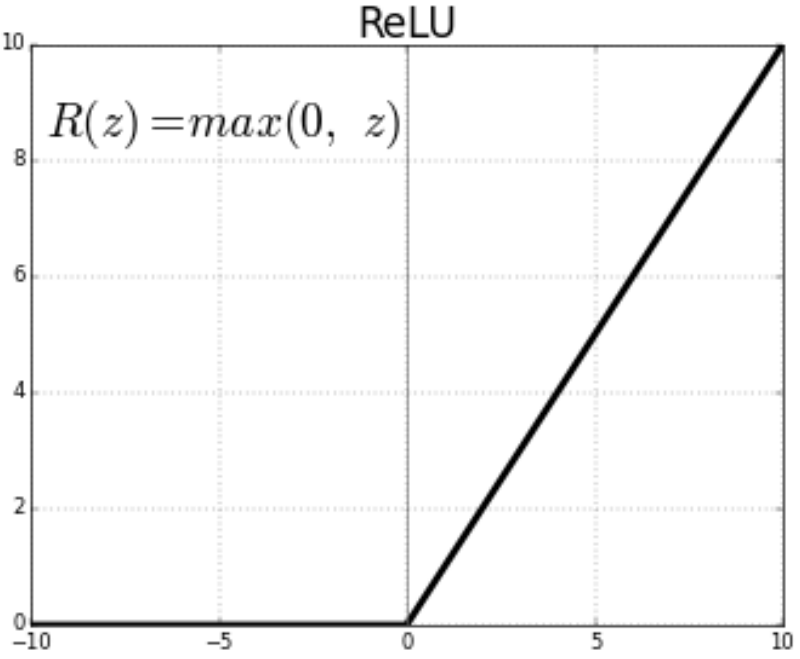
- Logística ou sigmóide: $f(a) = \frac{1}{1+e^{-a}}$



- Tangente hiperbólica: $f(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$



- Linear retificada (ReLU): $f(a) = \begin{cases} a, & \text{se } a >= 0 \\ 0, & \text{caso contrário} \end{cases}$



Há também parâmetros relacionados ao processo de treinamento:

- Taxa de aprendizado η: valor maior do que zero que representa a velocidade em que ocorre o aprendizado

- *Batch size*: número de amostras a serem usadas a cada atualização de pesos
- Algoritmo de otimização: embora o algoritmo Backpropagation e suas variações seja o mais empregado, há outros algoritmos disponíveis para o trienamento. Aqui vamos trabalhar com o Backpropagation tradicional.
- Número de épocas: determina quantas épocas, ou seja, quantas vezes cada amostra de entrada será utilizada, mesmo que o algoritmo de otimização não tenha encontrado um resultado confiável

Dessa forma, a instanciação da nossa rede MLP fica:

In [113]:

```
from sklearn.neural_network import MLPClassifier
# Definição dos hiperparâmetros
# Função de ativação: pode ser uma dentre: {'identity', 'logistic', 'tanh', 'rel
u'}, default 'relu'
activation = 'logistic'
# Tamanhos das camadas ocultas
hidden_layer_sizes = [10] # Apenas uma camada com quatro neurônios
# Algoritmo de otimização: pode ser um dentre {'lbfgs', 'sgd', 'adam'}, default 'ad
am'
solver = 'sgd' #Stochastic Gradient Descent
# Valor da taxa de aprendizado, default 0.001
learning_rate =  0.03
# Batch size, default é 'auto', ou seja, min(200, n_samples)
batch_size=50
# Número máximo de épocas, default 200
max_iter = 1000
mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                    activation=activation, solver=solver,
                    learning_rate_init=learning_rate,
                    batch_size=batch_size,
                    max_iter=max_iter)
```

Treinamento da MLP

Executar a classificação com MLP através do pacote de Machine Learning mais conhecido do Python, o `scikit-learn'.

Passos que seguiremos:

1. Carregar o arquivo de dados do iris dataset em um DataFrame do `pandas`.
2. Normalizar os dados de entrada
3. Separar aleatoriamente o dataframe em amostras de treinamento e de validação
4. Treinar o classificador MLP instanciado
5. Exeutar o classificador e visualizar os resultados obtidos

In [81]:

```
#Importando o módulo pandas, e invocando como pd
import pandas as pd

# 1. Carregar o arquivo na variável df
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
header = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
df = pd.read_csv(url, header=None, names=header)
```

In [94]:

```
# 2.Encontrar um dataframe numérico normalizado
df_norm = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
df_norm = (df_norm - df_norm.mean())/df_norm.std()
# Aqui acrescentamos a coluna de espécies
df_norm['species'] = df['species']

# 3. Separando as amostras de treinamento e validação
frac_validacao = 0.2 # Separamos 20% para a validação
# Primeiro embaralhamos o dataframe...
from sklearn.utils import shuffle
df_shuffle = shuffle(df)
# ... depois separamos em porção de treinamento e validação
import math
icut = math.floor(frac_validacao * df_shuffle.shape[0])
df_valid = df_shuffle.iloc[:icut, :]
df_treino  = df_shuffle.iloc[icut:, :]
```

In [95]:

```
# 4. Treinar o classificador MLP instanciado anteriormente, seprando as entradas e
 saídas
entradas = df_treino[['sepal_length', 'sepal_width', 'petal_length', 'petal_width'
]]
saidas = df_treino['species']
mlp.fit(entradas, saidas)
```

Out[95]:

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size=50, beta
_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=[100], learning_rate='constant',
       learning_rate_init=0.03, max_iter=1000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.
1,
       verbose=False, warm_start=False)
```

Uma observação sobre a sintaxe dos classificadores do `scikit-learn`

- O método fit(X,Y) recebe uma matriz ou dataframe X onde cada linha é uma amostra de aprendizado, e um array Y contendo as saídas esperadas do classificador, seja na forma de texto ou de inteiros
- O método predict(X) recebe uma matriz ou dataframe X onde cada linha é uma amostra de teste, retornando um array de classes

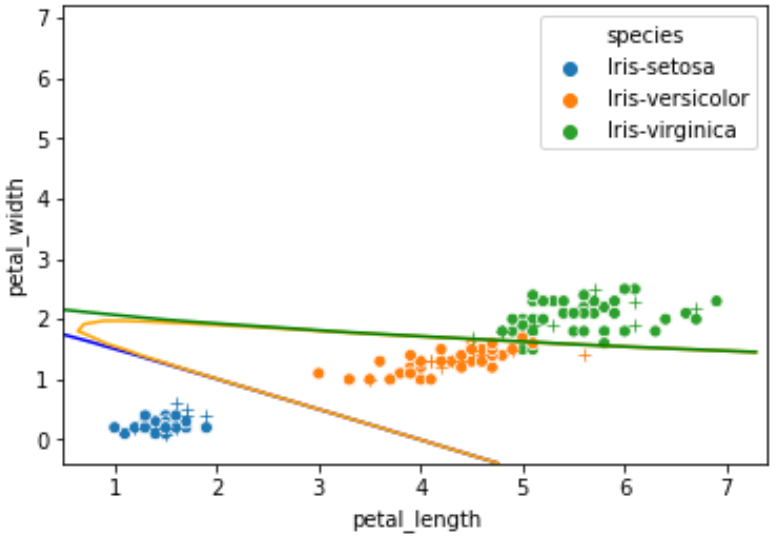
In [96]:

```
# 5. Executar o classificador nas amostras de validação
entradas_valid = df_valid[['sepal_length', 'sepal_width', 'petal_length', 'petal_wi
dth']]
classes = mlp.predict(entradas_valid)

#Cálculo do erro
erro = np.sum(classes != df_valid['species'])/df_valid.shape[0]
print("Erro médio de classificação: ", erro)
```

```
Erro médio de classificação:  0.03333333333333333
```

In [114]:

```
# 8. Visualizando o resultado usando o seaborn e matplotlib
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def visualizar_resultado(data_learn, data_valid, colunas_dados, coluna_classes, cla
ssifier):
    h = 0.2 # Step size in the mesh
    #Definimos a faixa de valores de X e Y
    X = data_learn[colunas_dados[0]]
    Y = data_learn[colunas_dados[1]]
    x_min = min(X.min(), data_valid[colunas_dados[0]].min()) - .5
    x_max = max(X.max(), data_valid[colunas_dados[0]].max()) + .5
    y_min = min(Y.min(), data_valid[colunas_dados[0]].min()) - .5
    y_max = max(Y.max(), data_valid[colunas_dados[0]].max()) + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Treina o classificador para as features de visualização
    classifier.fit(data_learn[colunas_dados], data_learn[coluna_classes])
    # Resultado da classificação
    classes = classifier.predict(data_valid[colunas_dados])

    # Zonas de classificação
    i = 0
    cores = ['blue', 'orange', 'green']
    for cls in np.unique(data_learn[coluna_classes]):
        Z = classifier.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, i]
        # Put the result into a contour plot
        Z = Z.reshape(xx.shape)
        plt.contour(xx, yy, Z, 1, colors=cores[i%len(cores)])
        i += 1

    # Plot also the training points
    unique_classes = np.unique(data_learn[coluna_classes])
    sns.scatterplot(*colunas_dados, hue=coluna_classes, hue_order=unique_classes, d
ata=data_learn)
    # and testing points
    sns.scatterplot(*colunas_dados, hue=classes, hue_order=unique_classes, data=dat
a_valid, marker='+', legend=False)

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)


visualizar_resultado(df_treino, entradas_valid, ['petal_length', 'petal_width'], 's
pecies', mlp)
```


species
Iris-setosa
Iris-versicolor
Iris-virginica
petal_width
petal_length
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7