Redes neuronales

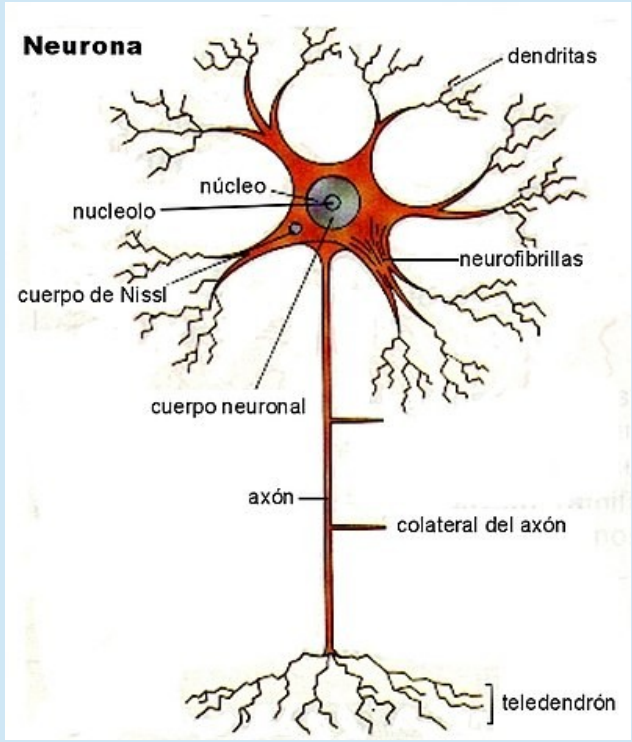
Deep learning

De entre los algoritmos de inteligencia artificial recientemente están resurgiendo los algoritmos de „aprendizaje profundo“.

El texto introduce estas redes a partir del perceptrón multicapa (MLP)

Biológicamente una neurona es una célula exitable que reacciona a las señales recibidas

- Por abstracción a este proceso, en la red neuronal, una neurona es la unidad base



A partir de la noción de neurona las podemos agrupar para generar distintas topologías

- Existen muchas estructuras de redes diseñadas para cada tipo de problema

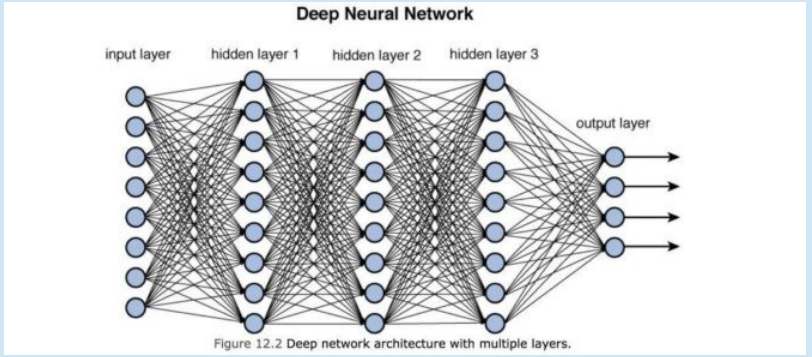


Figure 12.2 Deep network architecture with multiple layers.

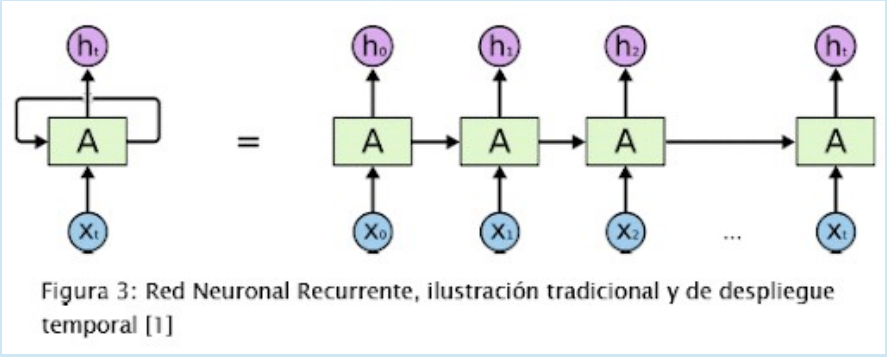
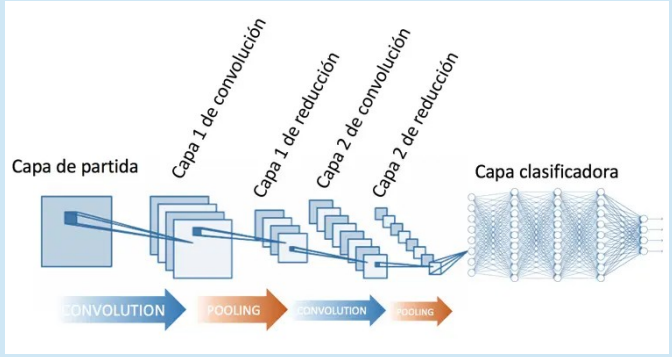


Figura 3: Red Neuronal Recurrente, ilustración tradicional y de despliegue temporal [1]



Plan

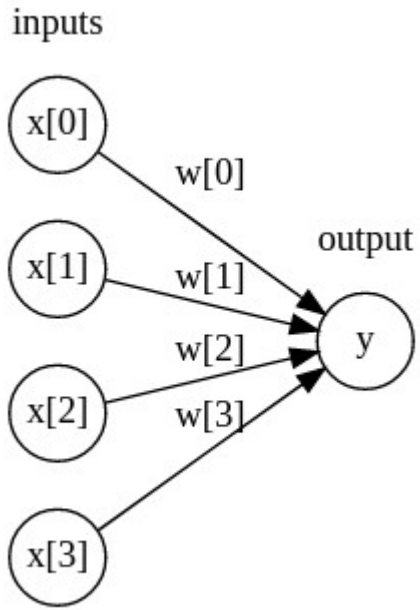
- **Redes neuronales**
- **Entrenamiento y ajustes**
- **Caso de estudio**

Redes neuronales

Perceptrón multicapa (MLP)

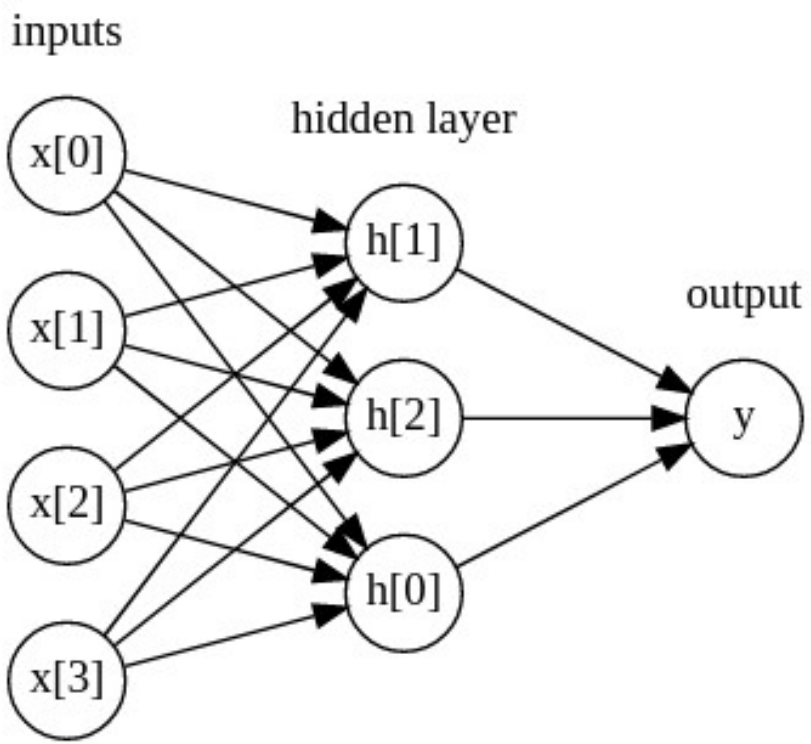
El MLP puede ser visto como una generalización de los modelos lineales en múltiples etapas.

- „Y“ es una suma ponderada y w los pesos de cada conexión
- Cada nodo a la izquierda representa una característica de entrada



$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \ldots + w[p] * x[p] + b$$

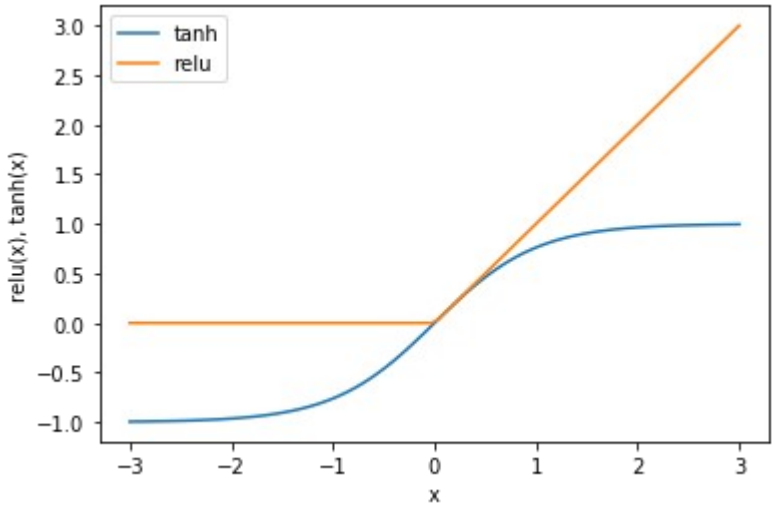
En un MLP este proceso de sumas ponderadas se repite multiples veces



- Cada suma ponderada intermedia es llamada capa oculta
- Estas capas ocultas se combinan mediante sumas ponderadas
- Pueden existir múltiples capas ocultas

Para diferenciar el modelo neuronal de un modelo lineal se suele hacer un cambio adicional

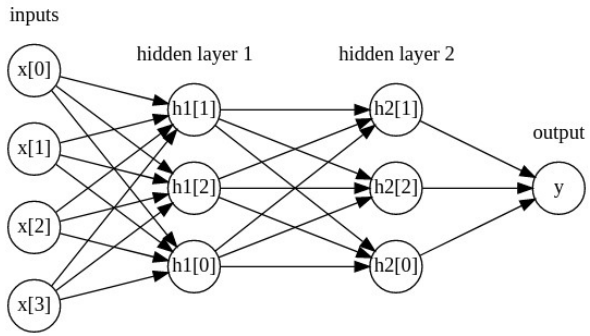
- Tras calcular la suma ponderada se suele aplicar una función (de activación) no lineal al resultado
- Dos funciones usuales son la unidad lineal rectificada y la tangente hiperbólica



Este MLP da origen a un modelo no lineal con múltiples parámetros de ajuste

$$h[0] = \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])$$
$$h[1] = \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])$$
$$h[2] = \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3])$$
$$\hat{y} = v[0] * h[0] + v[1] * h[1] + v[2] * h[2]$$

Una red neuronal puede tener múltiples capas ocultas



- Grandes redes con múltiples capas ocultas inspiraron el nombre „aprendizaje profundo“

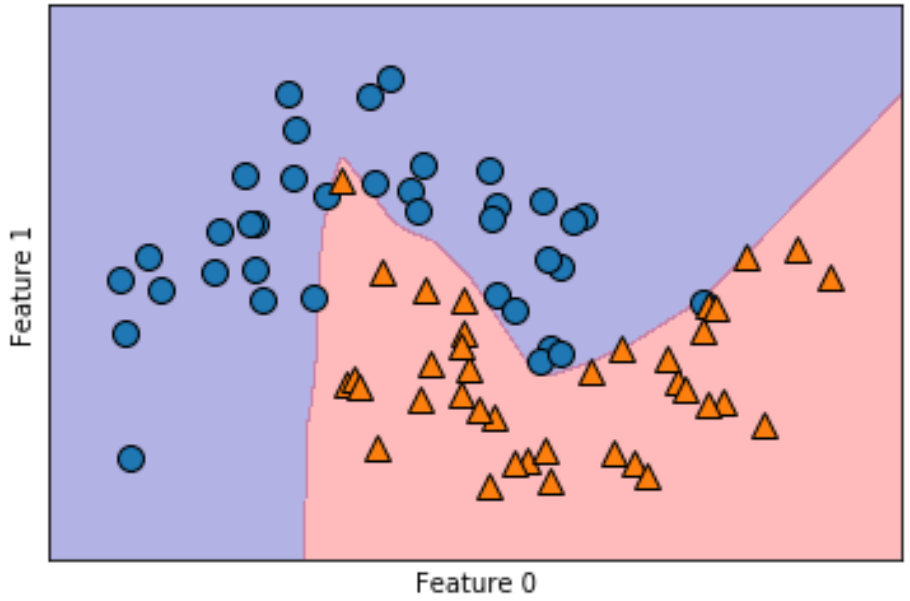
- La cantidad de neuronas puede ser tan pequeño o grande como la aplicación lo requiera
- Es posible aumentar el número de capas ocultas tanto como sea necesario

Entrenamiento y ajustes

Partamos del dataset two_moons tomando 100 puntos

- Este conjunto será la base para explorar el comportamiento del MLP

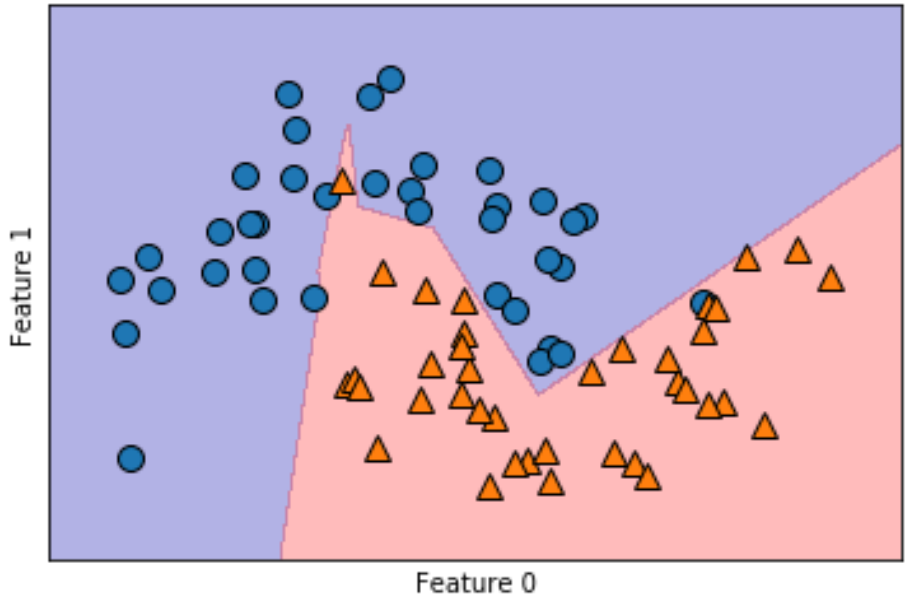
```
#In[93]:
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=42)
mlp = MLPClassifier(solver='lbfgs', random_state=0).fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



Definiendo la red podemos ajustar el número de neuronas según nuestras necesidades

- Este modelo ejemplifica el uso de una capa oculta con diez neuronas

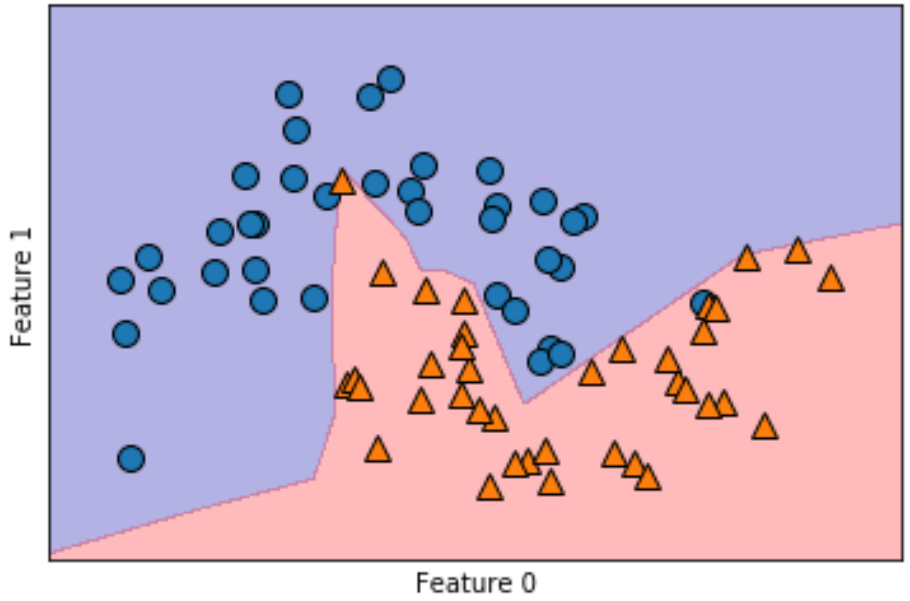
```
#In[94]:
mlp = MLPClassifier(solver='lbfgs', random_state=0, hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



Un segundo ajuste posible es variar el número de capaz ocultas

- Cada capa puede tener diferente número de neuronas
- Este ejemplo tiene dos capas ocultas con diez neuronas

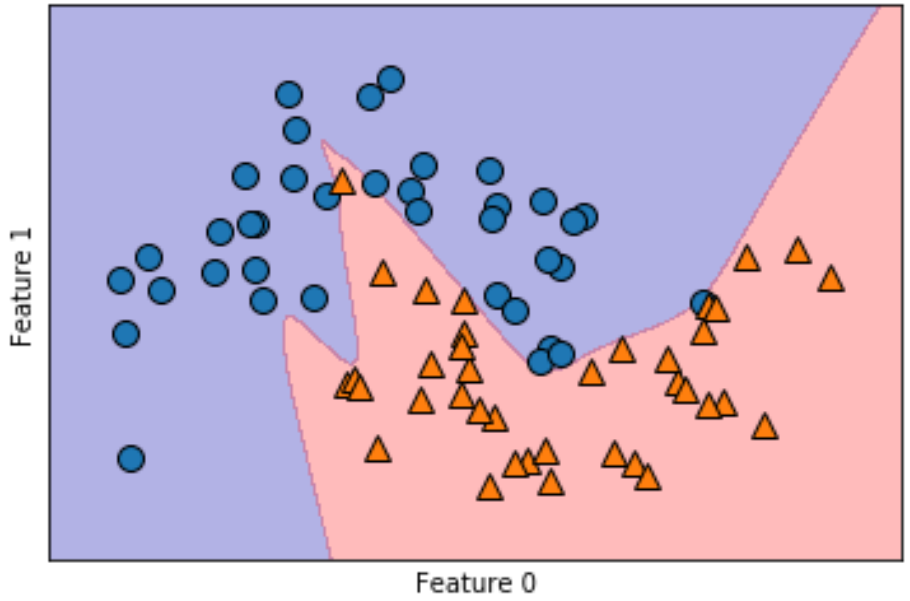
```
#In[95]:
# using two hidden layers, with 10 units each
mlp = MLPClassifier(solver='lbfgs', random_state=0,
hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



La función de activación es otro parámetro ajustable para el modelo

- Cada capa puede tener diferente función de activación
- Este ejemplo implementa la función tanh

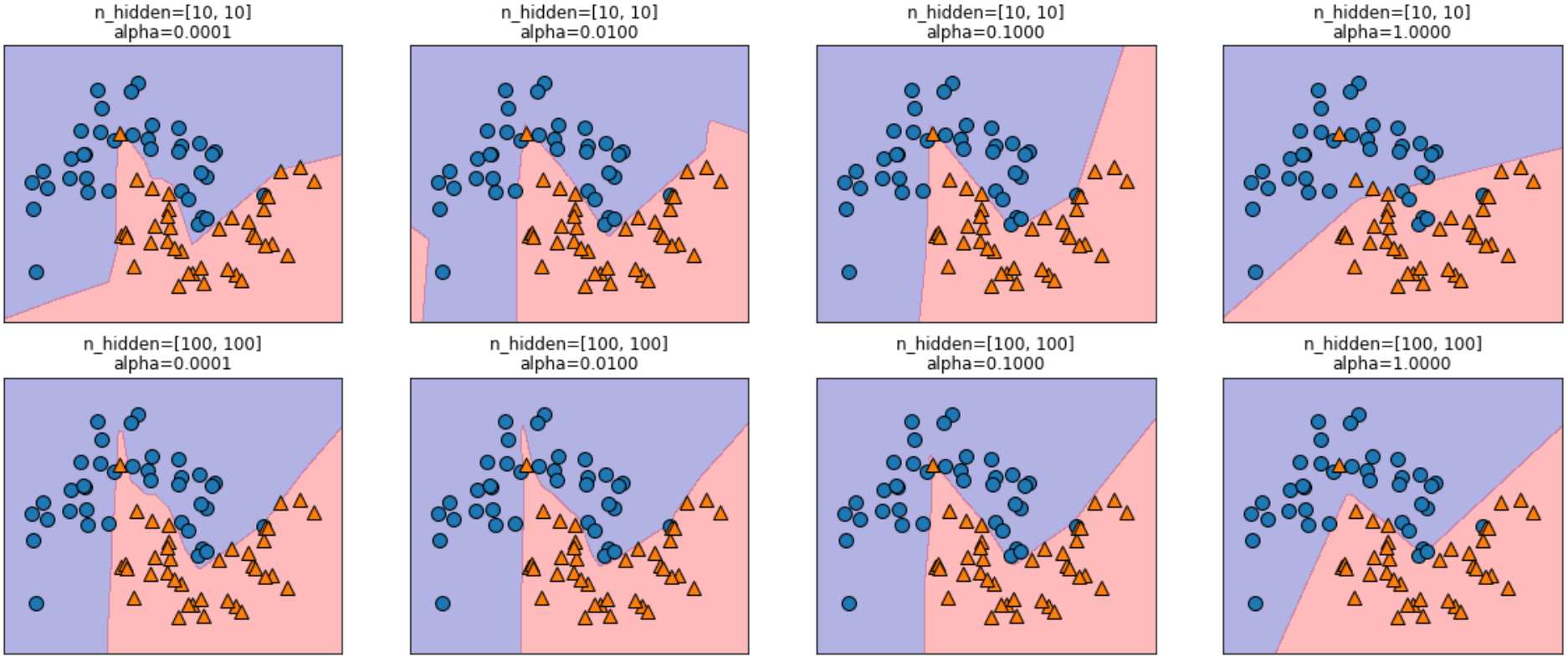
```
#In[96]:
# using two hidden layers, with 10 units each, now with tanh nonlinearity
mlp = MLPClassifier(solver='lbfgs', activation='tanh',
random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



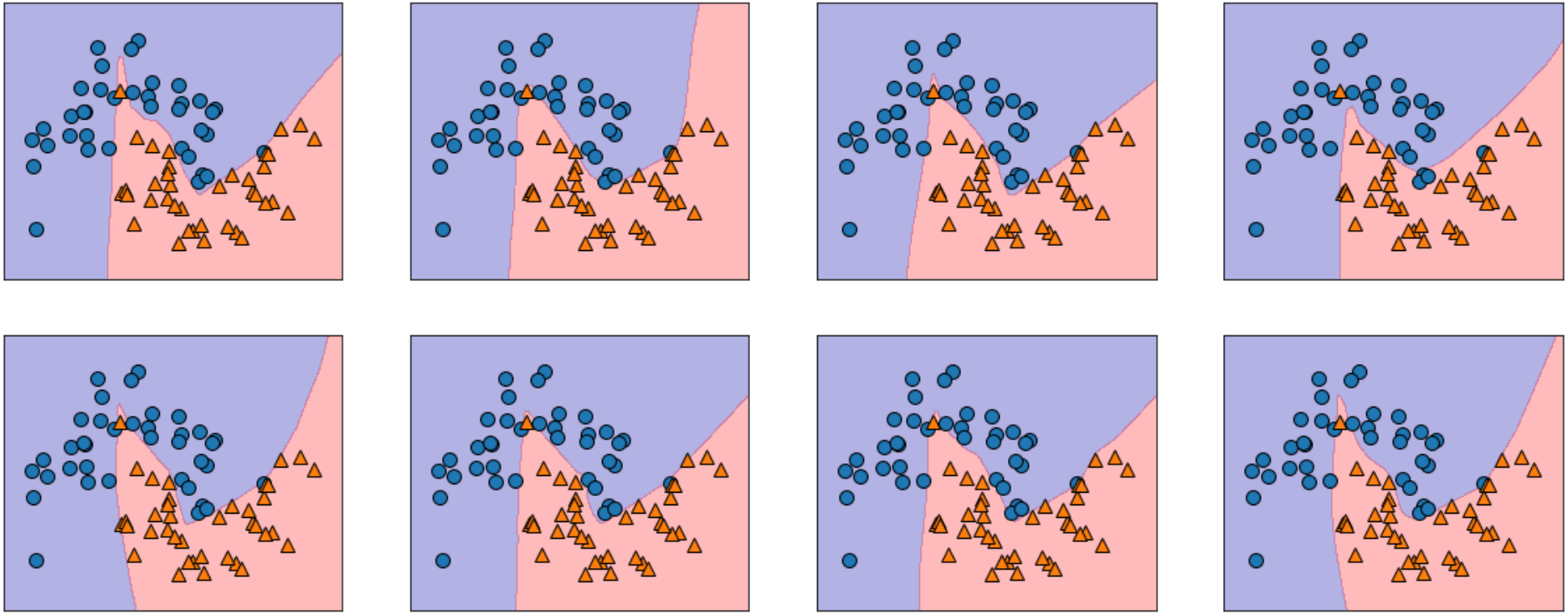
El parámetro de regularización alfa restringe la variación entre los pesos

- La variabilidad entre los pesos de la red se ve acotada por el parámetro alfa

Variando estos parámetros se generan distintas fronteras de decisión



Las fronteras dependen de la semilla aleatoria con que fue inicializada la red



Caso de estudio

- Para un mejor entendimiento tomemos datos del mundo real (dataset de cancer de mama)

Tomemos el dataset de cancer de mama

```
#In[99]:
print("Cancer data per-feature maxima:\n{}".format(cancer.data.max(axis=0)))
```

```
Cancer data per-feature maxima:
[2.811e+01 3.928e+01 1.885e+02 2.501e+03 1.634e-01 3.454e-01 4.268e-01
 2.012e-01 3.040e-01 9.744e-02 2.873e+00 4.885e+00 2.198e+01 5.422e+02
 3.113e-02 1.354e-01 3.960e-01 5.279e-02 7.895e-02 2.984e-02 3.604e+01
 4.954e+01 2.512e+02 4.254e+03 2.226e-01 1.058e+00 1.252e+00 2.910e-01
 6.638e-01 2.075e-01]
```

Usemos los parámetros base del clasificador

- La presición del MLP es buena
- No supera a los modelos estudiados en clases previas

```
#In[100]:
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, random_state=0)
mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)
print("Accuracy on training set: {:.2f}".format(mlp.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(mlp.score(X_test, y_test)))
```

```
Accuracy on training set: 0.94
Accuracy on test set: 0.92
```

Las redes neuronales esperan que las características de entrada varíen de manera similar

- Idealmente esperamos media 0 y varianza 1
- Podemos reescalar los datos para lograr estos objetivos
- La presición ha mejorado pero arroja advertencias

```
#In[101]:
# compute the mean value per feature on the training set
mean_on_train = X_train.mean(axis=0)
# compute the standard deviation of each feature on the training set
std_on_train = X_train.std(axis=0)
# subtract the mean, and scale by inverse standard deviation
# afterward, mean=0 and std=1
X_train_scaled = (X_train - mean_on_train) / std_on_train
# use THE SAME transformation (using training mean and std) on the test set
X_test_scaled = (X_test - mean_on_train) / std_on_train
mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Accuracy on training set: {:.3f}".format(
mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.991
Accuracy on test set: 0.965
```

Podemos indicar al modelo límites mas grandes de repeticiones para entrenar el modelo

- Esto es parte del algoritmo adam de entrenamiento

```
#In[102]:
mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Accuracy on training set: {:.3f}".format(
mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.972
```

Modificando el parámetro alpha controlamos la velocidad con que los pesos son entrenados

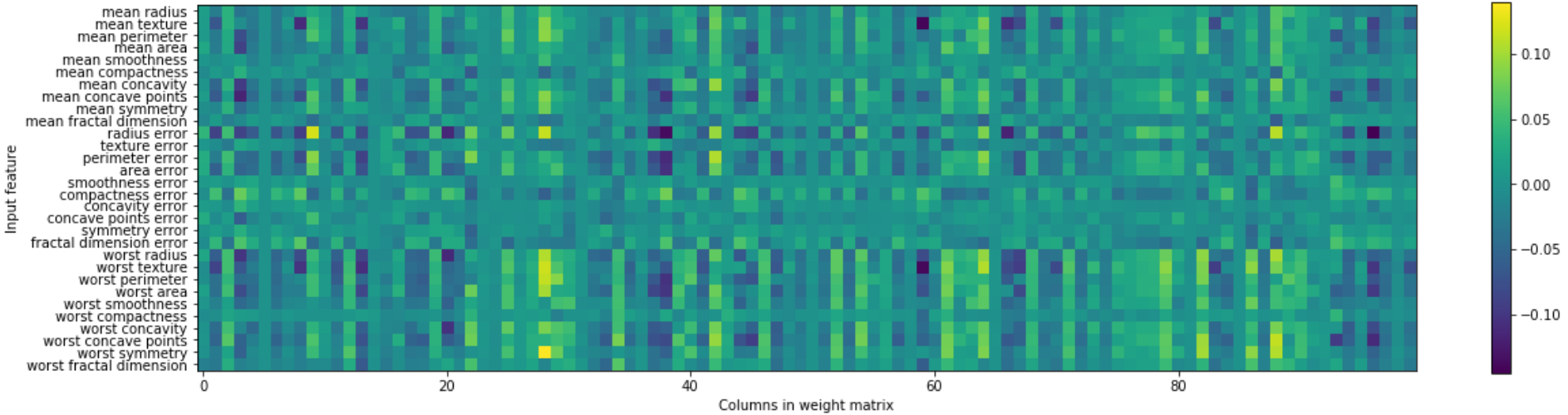
- Este parámetro también puede repercutir en la estabilidad con que se explora la solución

```
#In[103]:
mlp = MLPClassifier(max_iter=1000, alpha=1, random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Accuracy on training set: {:.3f}".format(
mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.988
Accuracy on test set: 0.972
```

Con un cuidadoso análisis es posible explorar lo que la red podría estar aprendiendo

- En la práctica este enfoque resulta ser impráctico
- Como alternativa es mas viable estudiar la importancia que tienen las características de entrada
- Este enfoque parte del estudio de los pesos de cada neurona para cada parámetro de entrada


mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
Input feature
0
20
40
60
80
Columns in weight matrix
0.10
0.05
0.00
−0.05
−0.10