Redes Neurais Artificiais

Fabrício Olivetti de França

Universidade Federal do ABC

Tópicos

1. Redes Neurais Biológicas
2. Neurônio Artificial
3. Rede Neural Artificial
4. Keras

UFABC

Redes Neurais Biológicas

Rede Neural

- Área de pesquisa da Neurociência.
- Composta de **neurônios** interconectados formando uma rede de processamento.
- Comunicação via sinais elétricos.

Rede Neural

A Rede Neural biológica é composta por **neurônios** conectados via axônios.

Essa rede pode ser entendida como um grafo de fluxo de informação.

O fluxo é determinado pela soma dos sinais elétricos recebidos por um neurônio, se este for maior que um limiar, o neurônio emite um sinal elétrico adiante, caso contrário ele bloqueia o sinal.

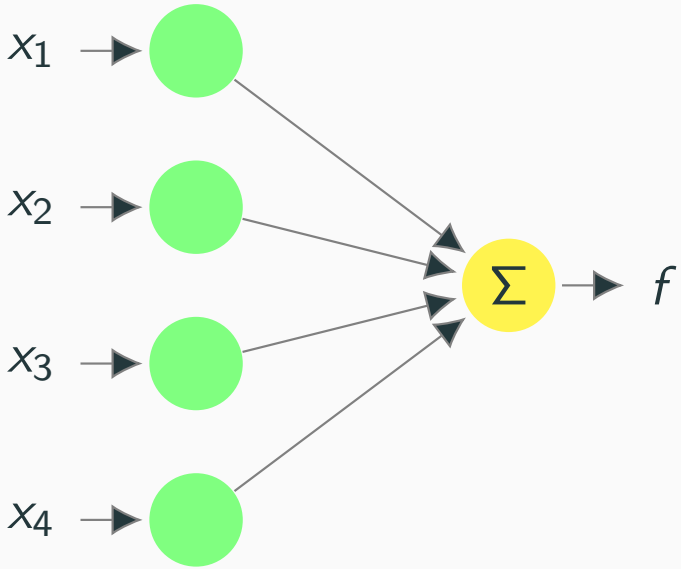
Neurônio Artificial

Modelo de Neurônio Artificial

Podemos criar um neurônio artificial como um *nó* de um Grafo que recebe múltiplas entradas e emite uma saída.

A saída é definida como a aplicação de uma função de ativação na soma dos valores de entrada.

Modelo de Neurônio Artificial



UFABC

Modelo de Neurônio Artificial

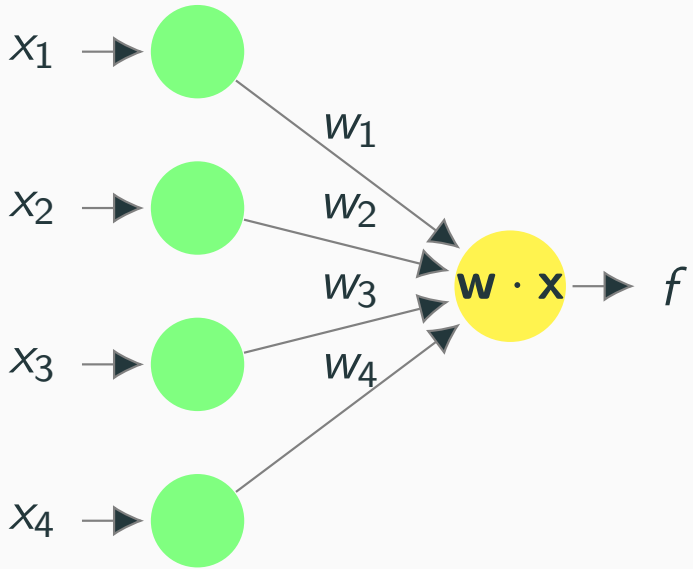
Pensando em entradas com valores reais entre 0 e 1, o neurônio é ativado sempre que a soma das entradas for maior que um determinado τ, ou seja, a função f é:

$$f(z) = \begin{cases} 0, & \text{se } z < \tau \\ g(z), & \text{c. c.} \end{cases},$$

com $g(z)$ a função de ativação que determinar o pulso a ser enviado e z a soma dos estímulos de entrada.

Modelo de Neurônio Artificial

Também é possível ponderar a importância dos estímulos de entrada do neurônio através de um vetor de pesos $\mathbf{w}$, substituindo a somatória pelo produto interno $\mathbf{w} \cdot \mathbf{x}$:



Modelo de Neurônio Artificial

O Neurônio Artificial é conhecido como Percéptron, visto em aulas anteriores.

Com isso, basta ajustar os pesos com os algoritmos que já aprendemos.

Também vimos em aulas anteriores que o Percéptron consegue apenas classificar amostras linearmente separáveis.

Rede Neural Artificial

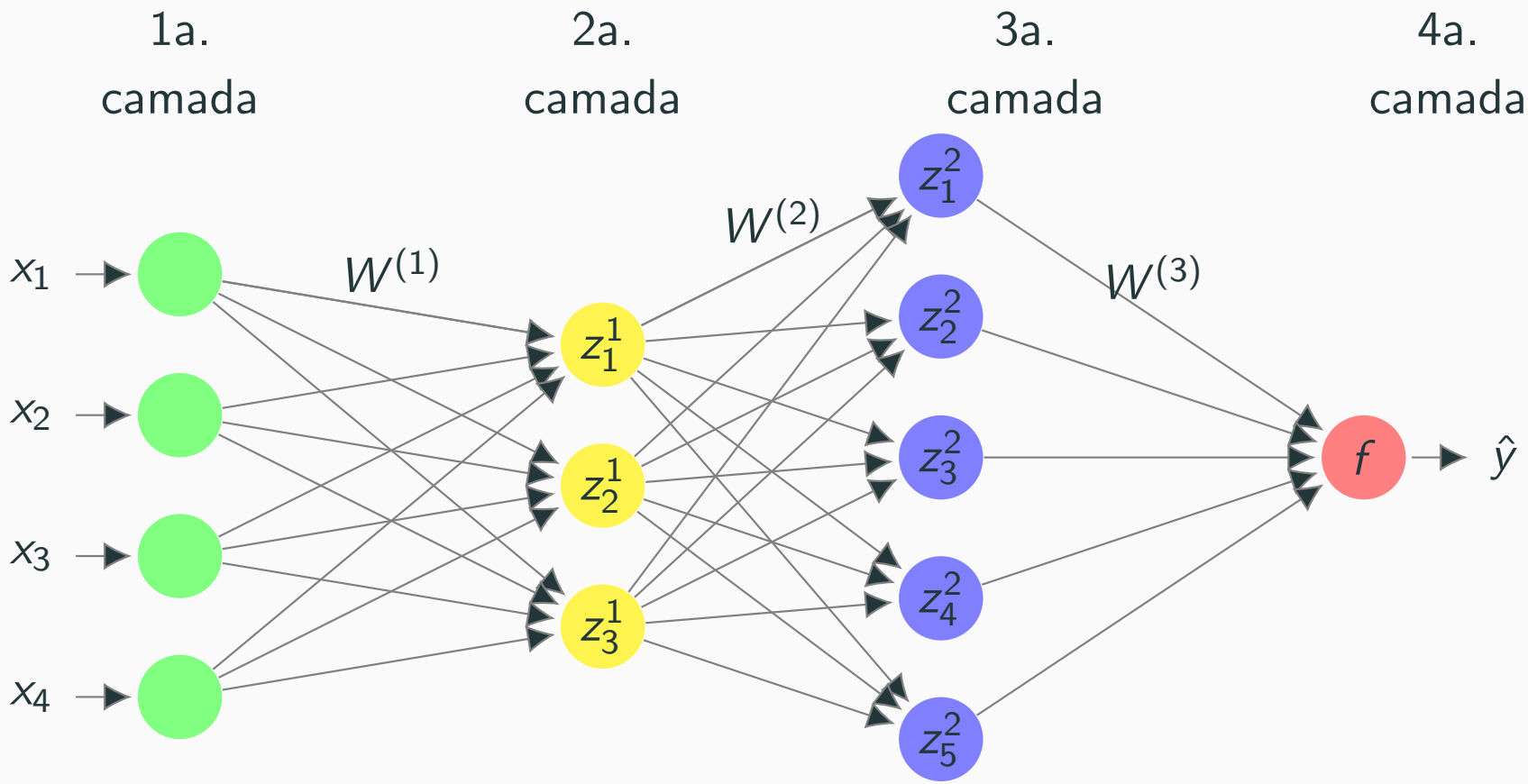
Percéptron de Múltiplas Camadas

Conhecido como Multi-Layer Percéptron (MLP) ou Feedforward Neural Network.

Rede composta de vários neurônios conectados por *camadas*.

O uso de camadas permite aproximar superfícies de separações não-lineares (considere o exercício 01 da aula 04).

Percéptron de Múltiplas Camadas



UFABC

Deep Learning

Essa é uma forma de Aprendizagem Profunda (Deep Learning).

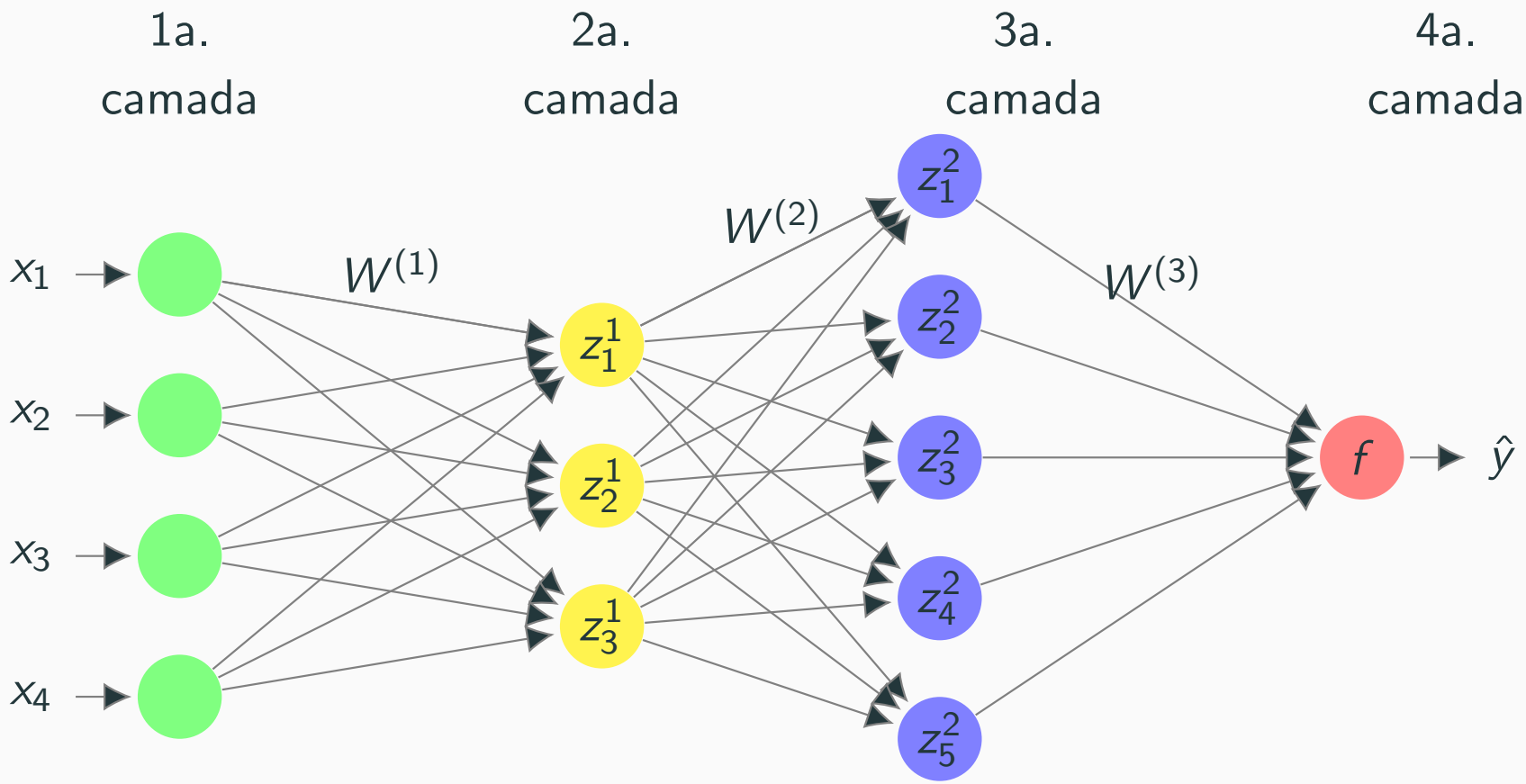
Quanto mais camadas, mais profunda é a rede.

Cada camada tem a função de criar novos atributos mais complexos.

Essa rede define um modelo computacional.

Definições

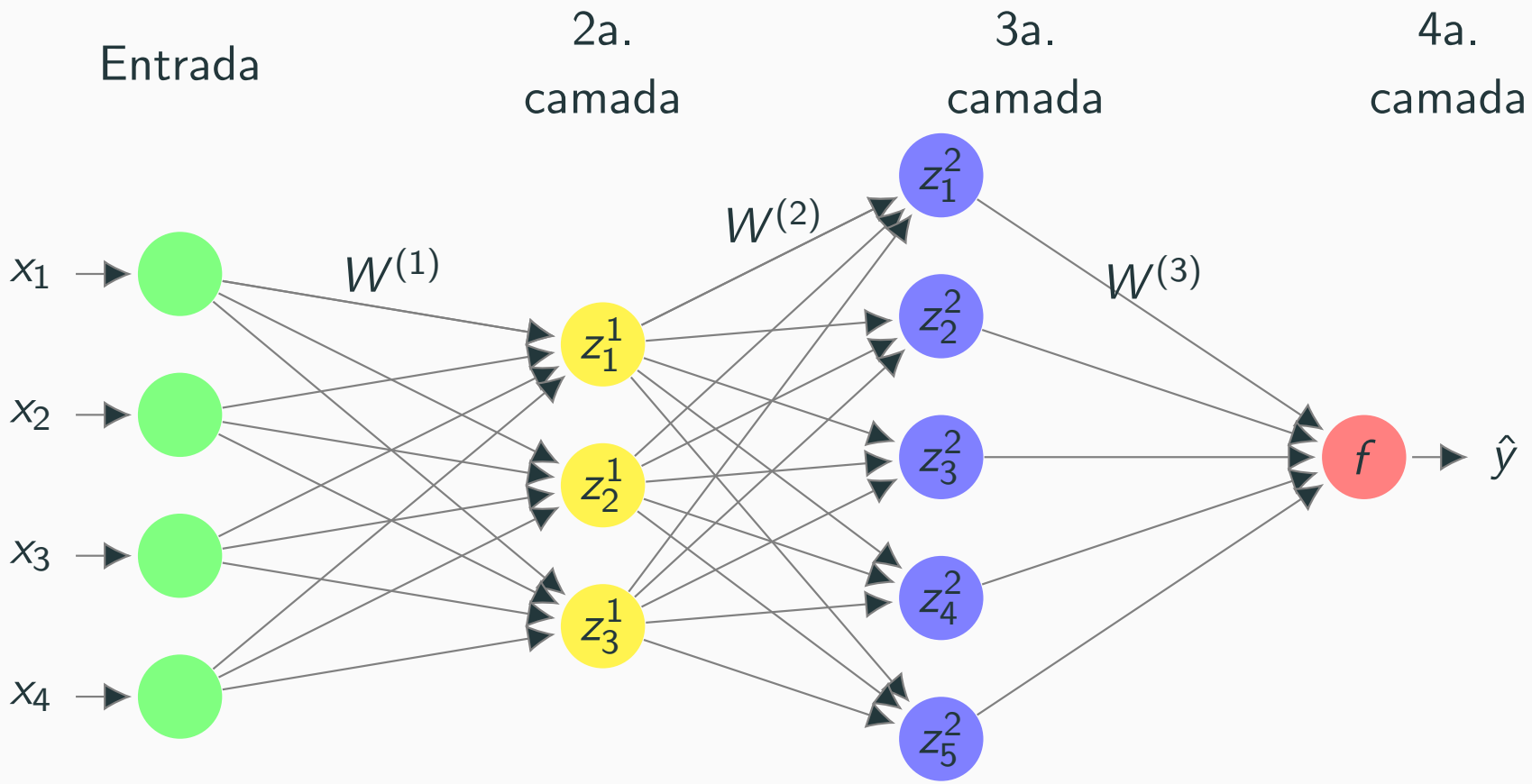
Cada conjunto de nós é denominado como uma *camada*.



UFABC

Definições

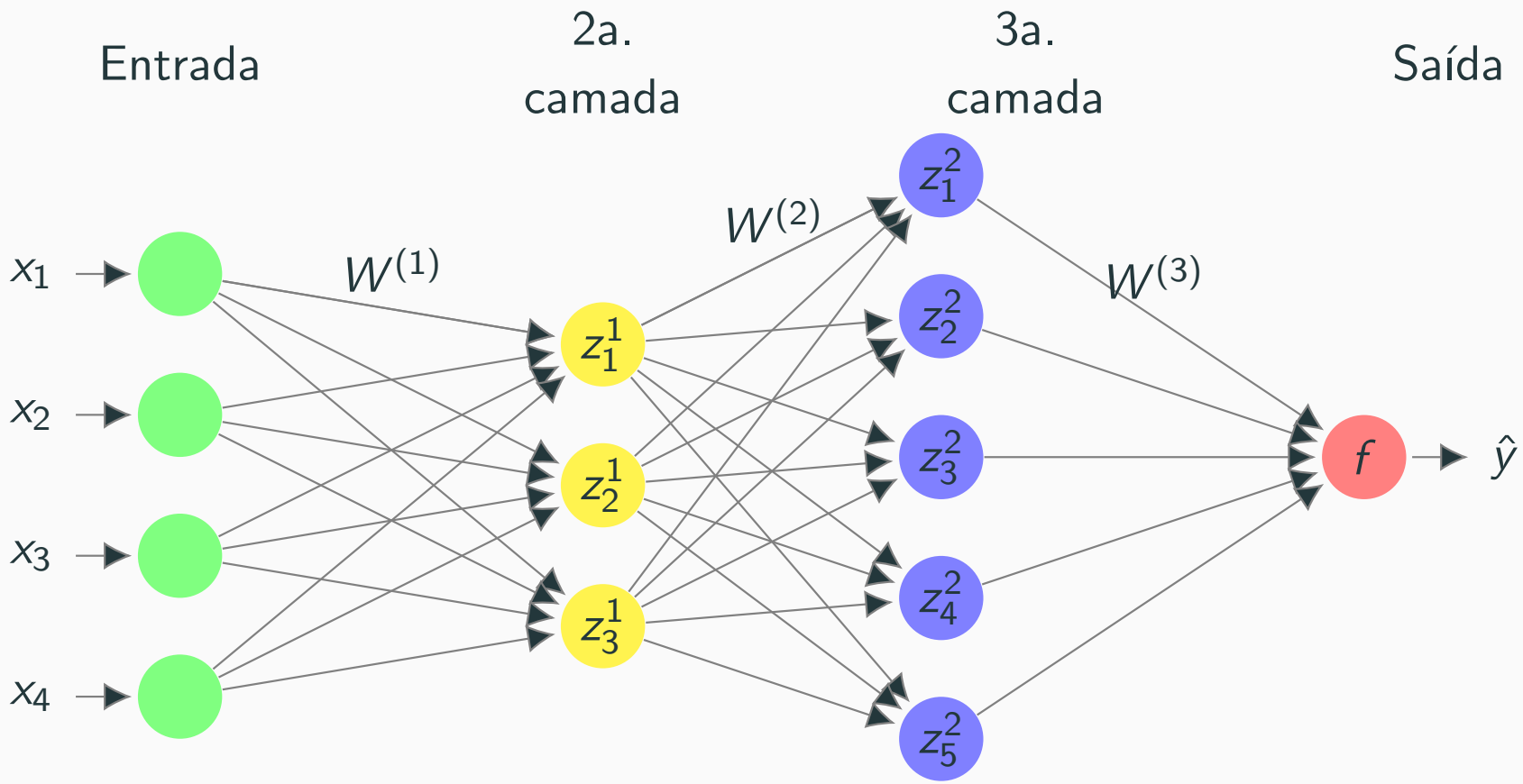
A primeira camada é conhecida como *entrada*.



UFABC

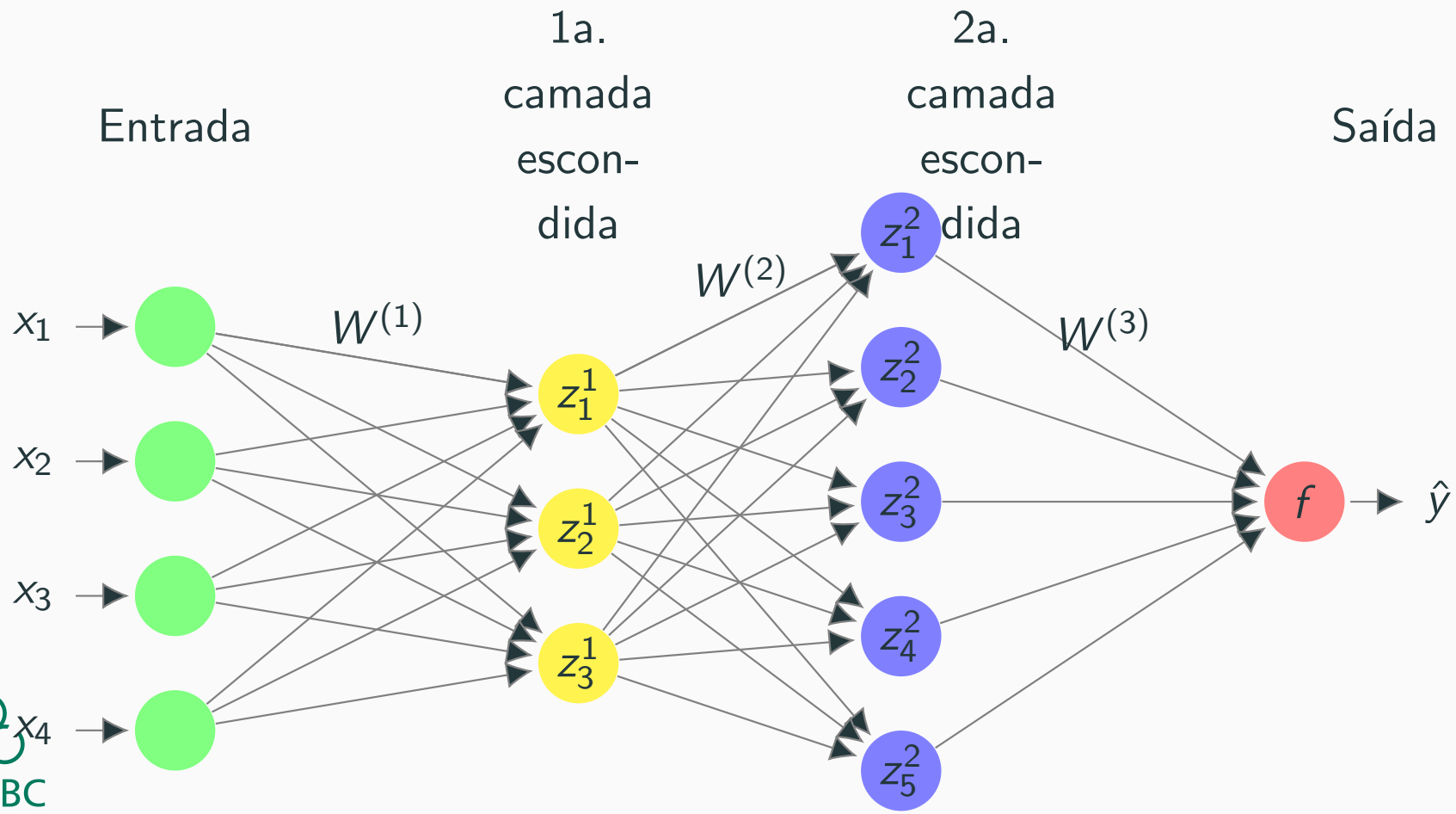
Definições

A última camada é a *saída* ou *resposta* do sistema.



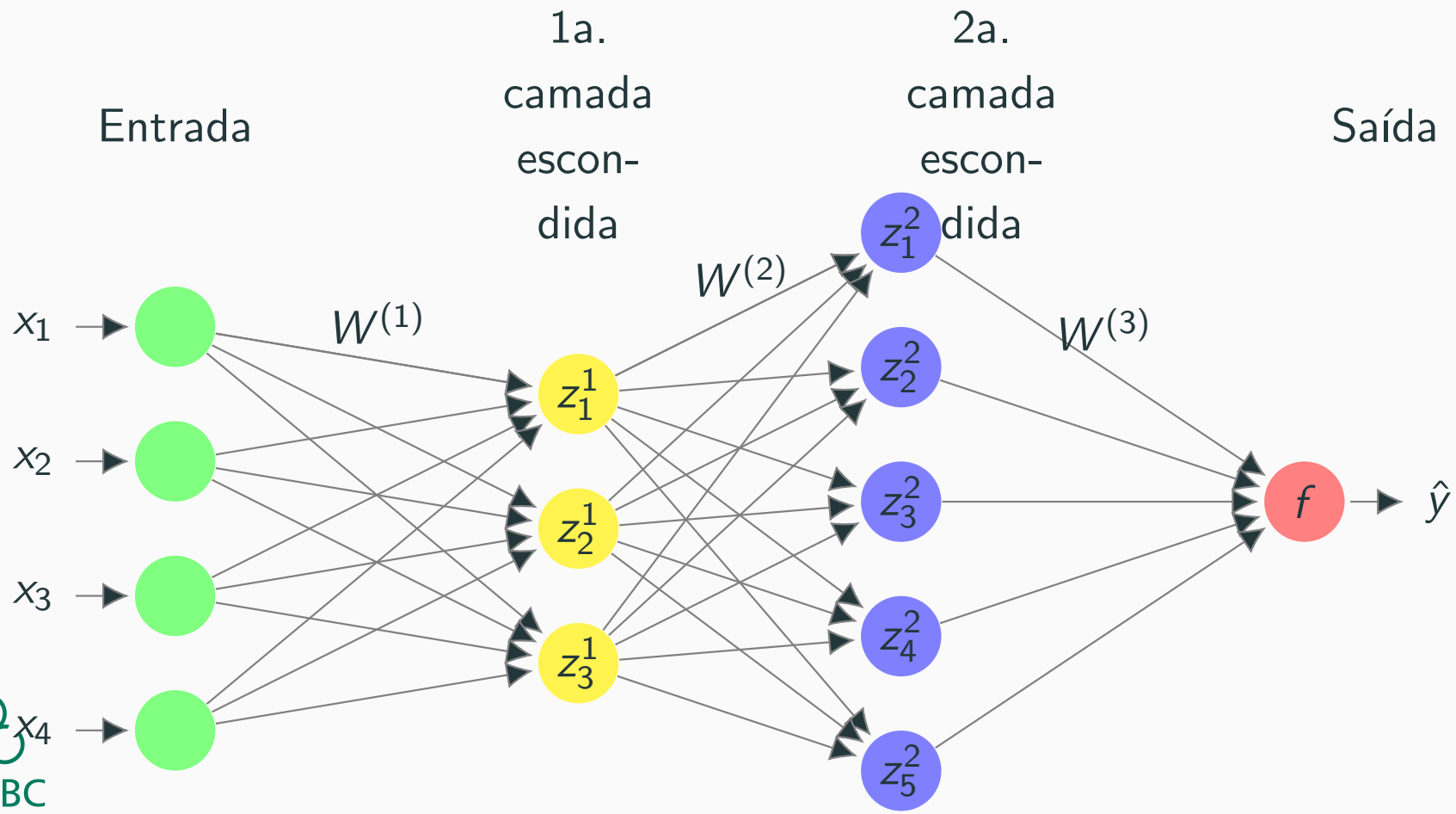
Definições

As camadas restantes são numeradas de 1 a m e são conhecidas como *camadas escondidas* ou *hidden layers*. Esse nome vem do fato de que elas não apresentam um significado explícito de nosso problema.



Definições

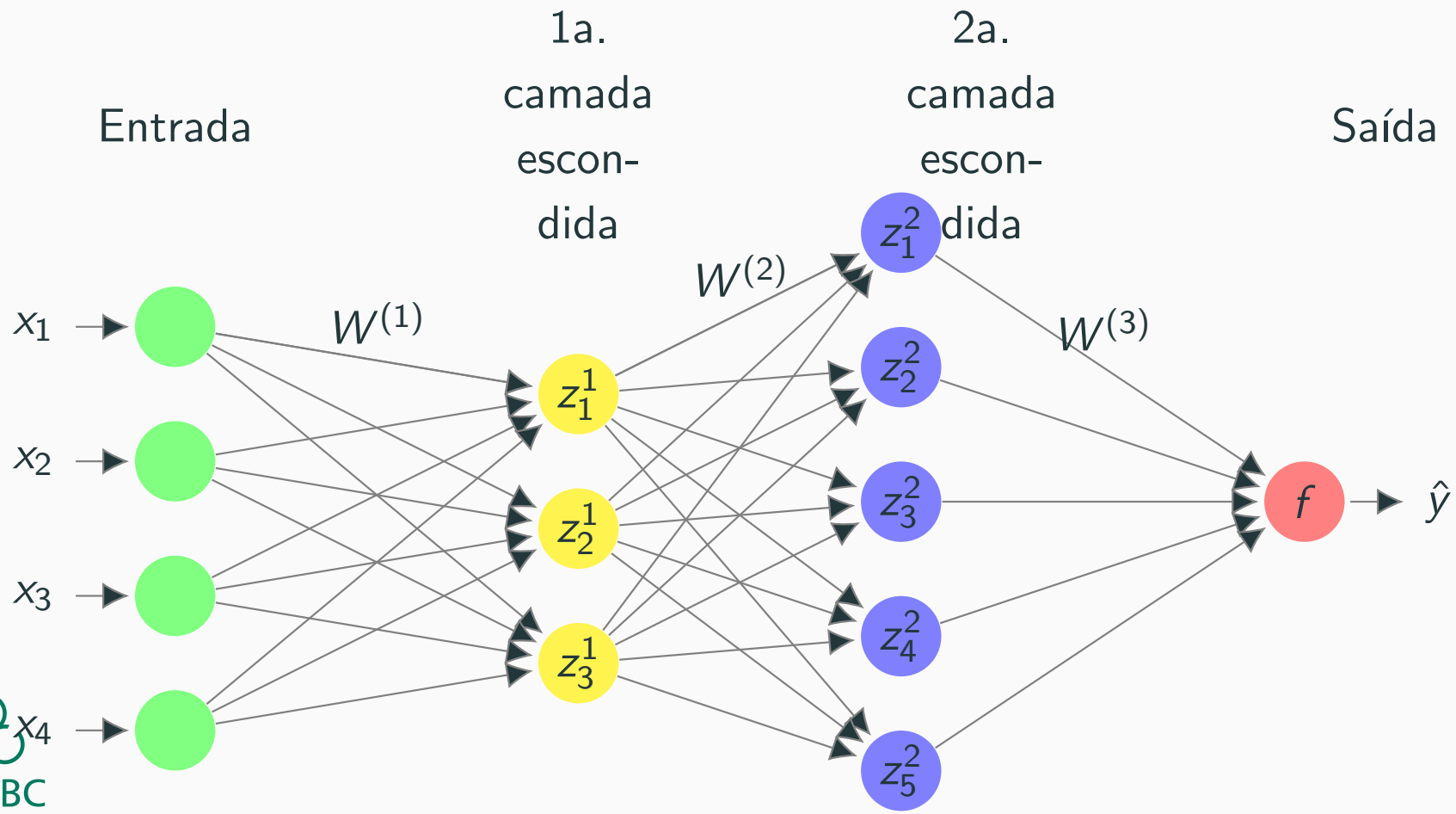
Duas camadas vizinhas costumam ser totalmente conectadas formando um grafo bipartido (em algumas variações podemos definir menos conexões).



UFABC

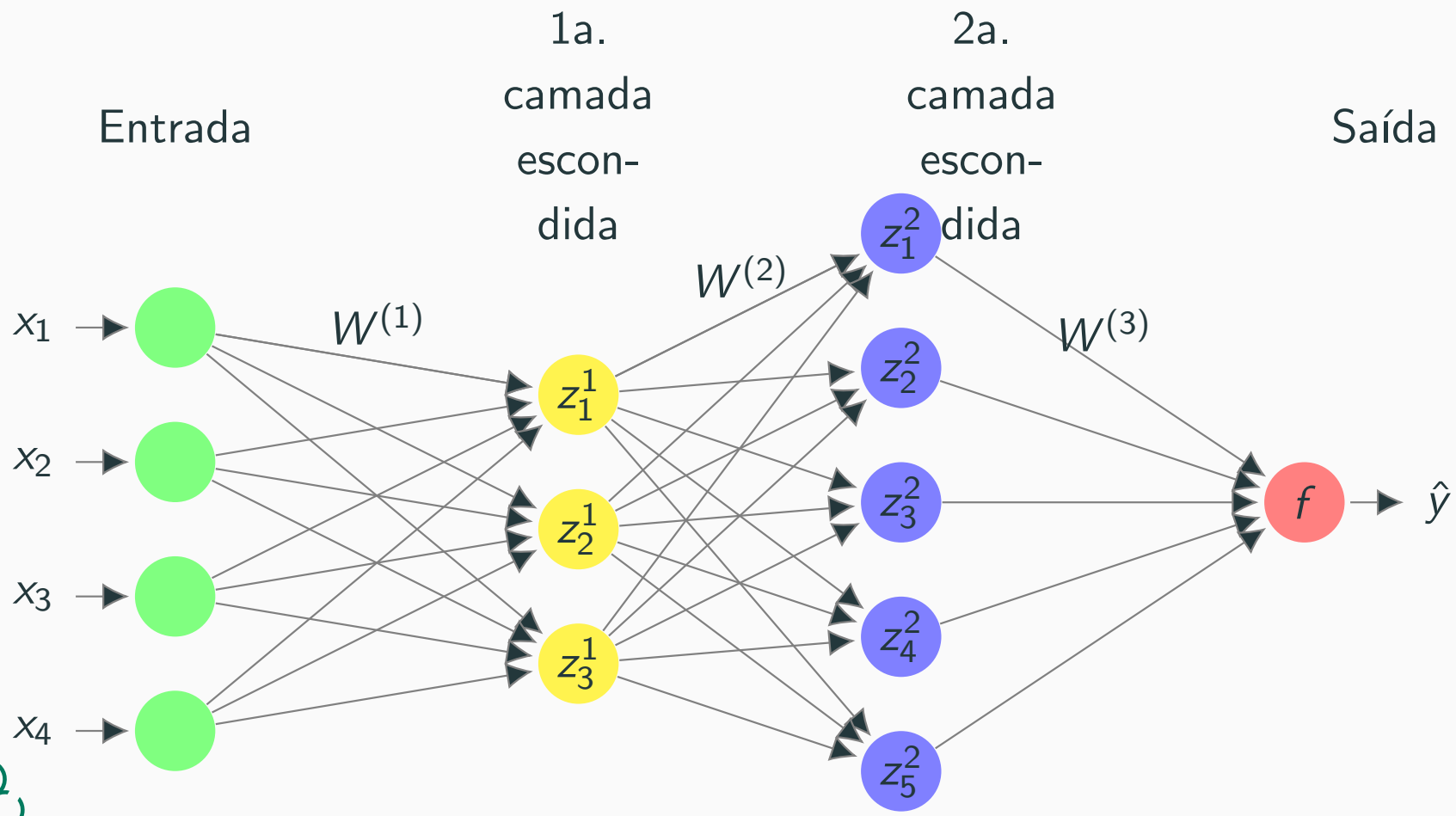
Definições

Esse grafo é ponderado e os pesos são definidos por uma matriz W de dimensão $n_o \times n_d$, com n_o sendo o número de neurônios da camada de origem e n_d da de destino.



Definições

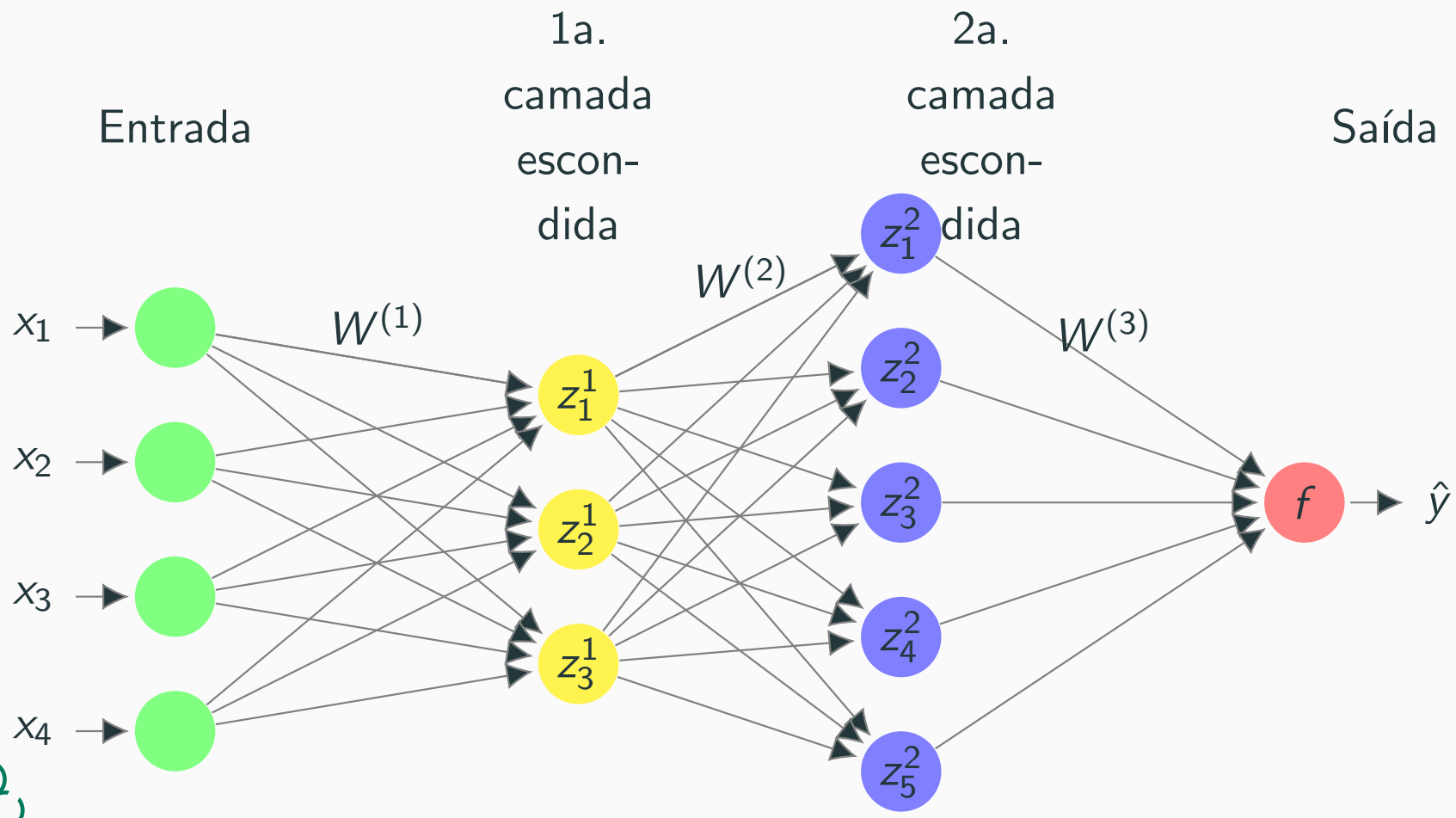
Assumindo que as entradas são representadas por um vetor linha $\mathbf{x}$, o processamento é definido como: $\mathbf{z^1} = f^{(1)}(\mathbf{x} \cdot W^{(1)})$.



UFABC

Definições

A camada seguinte calcula a próxima saída como $\mathbf{z^2} = f^{(2)}(\mathbf{z^1} \cdot W^{(2)})$, e assim por diante.



UFABC

Funções de Ativação

As funções de ativação comumente utilizadas em Redes Neurais são:

- **Linear:** $f(z) = z$, função identidade.
- **Logística:** $f(z) = \frac{1}{1+e^{-z}}$, cria uma variável em um tipo sinal, com valores entre 0 e 1.
- **Tangente Hiperbólica:** $f(z) = \tanh(z)$, idem ao anterior, mas variando entre -1 e 1.
- **Rectified Linear Units:** $f(z) = \max(0, z)$, elimina os valores negativos.
- **Softmax:** $f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$, faz com que a soma dos valores de z seja igual a 1.

Ajustando os Parâmetros

Para determinar os valores corretos dos pesos, utilizamos o Gradiente Descendente, assim como nos algoritmos de Regressão Linear e Logística.

Note porém que o cálculo da derivada da função de erro quadrático é igual aos casos já estudados apenas na última camada.

Para as outras camadas precisamos aplicar o algoritmo de **Retropropagação** que aplica a regra da cadeia.

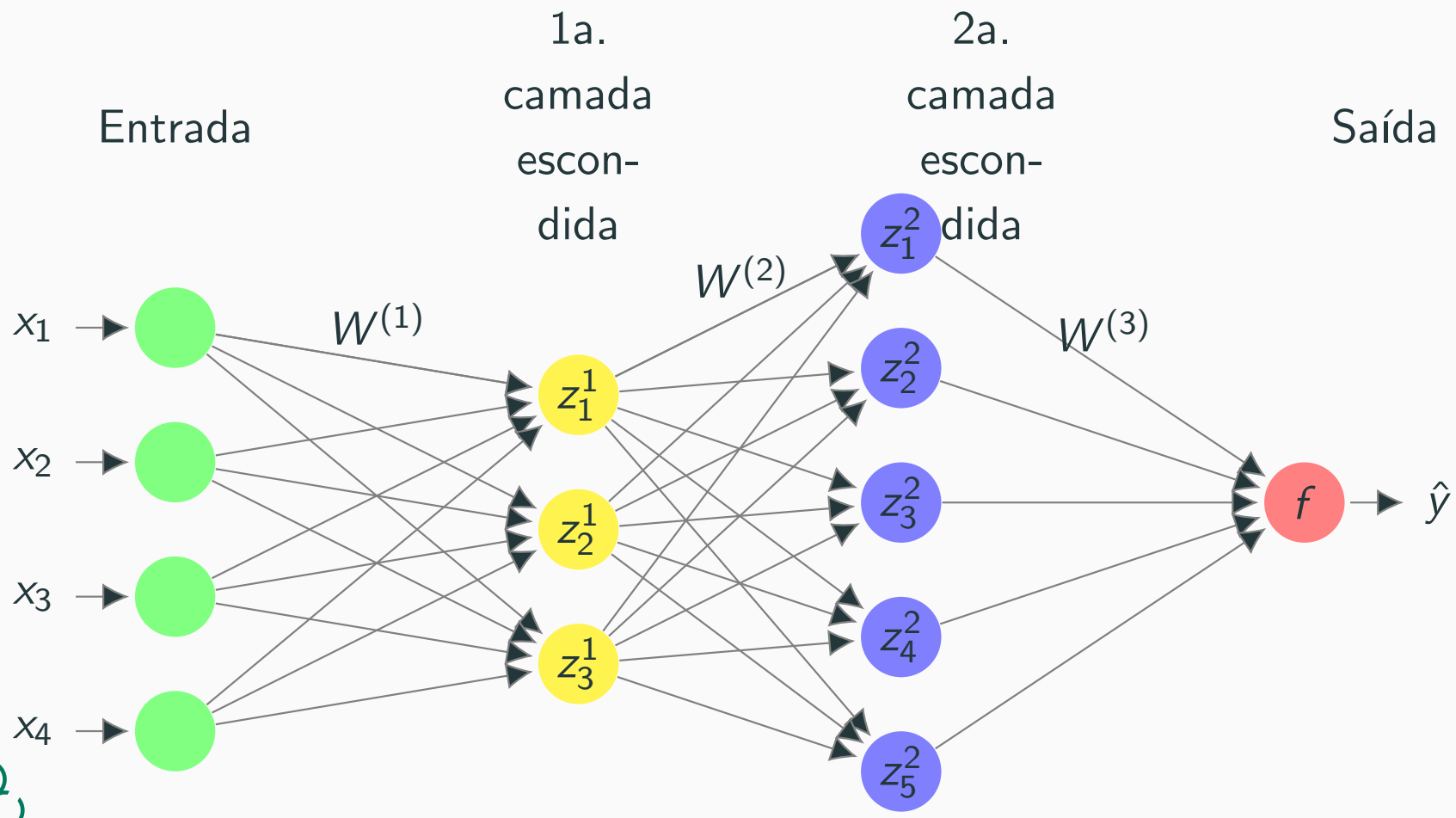
Retropropagação

O algoritmo segue os seguintes passos:

- Calcula a saída para uma certa entrada.
- Calcula o erro quadrático.
- Calcula o gradiente do erro quadrático em relação a cada peso.
- Atualiza pesos na direção oposta do gradiente.
- Repita.

Retropropagação

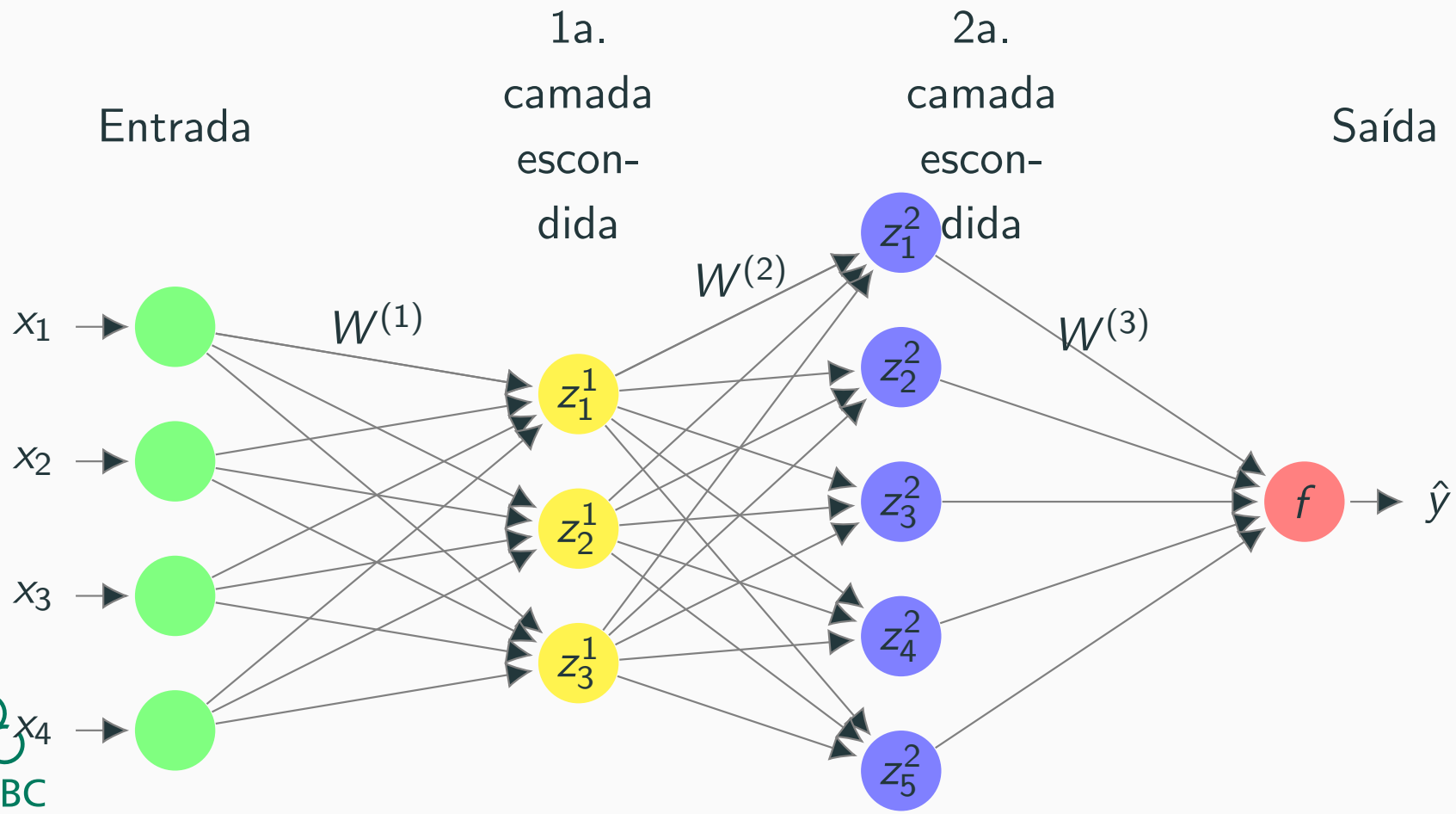
Assumindo a função de ativação logística $\sigma(z)$, cuja derivada é $\sigma(z)(1-\sigma(x))$, o gradiente da camada de saída é $\frac{\partial e}{\partial W^{(3)}} = (\hat{y} - y) \cdot z^2$.



UFABC

Retropropagação

O gradiente da camada anterior é calculado como: $\frac{\partial e}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'(x \cdot W^{(1)}) \cdot z^1$. E assim por diante até a camada de entrada.



UFABC

Dicas para Melhorar o desempenho do ajuste

- Utilize tanh como função sigmoidal.
- Utilize *softmax* para multi-classes.
- Escale as variáveis de saída para a mesma faixa de valores da segunda derivada da função de ativação (ex.: para tanh deixe as variáveis entre -1 e 1).
- Ajuste os parâmetros utilizando mini-batches dos dados de treinamento.
- Inicialize os pesos como valores aleatórios uniformes com média zero e desvio-padrão igual a $\frac{1}{\sqrt{m}}$, com m sendo o número de nós da camada anterior.

Keras

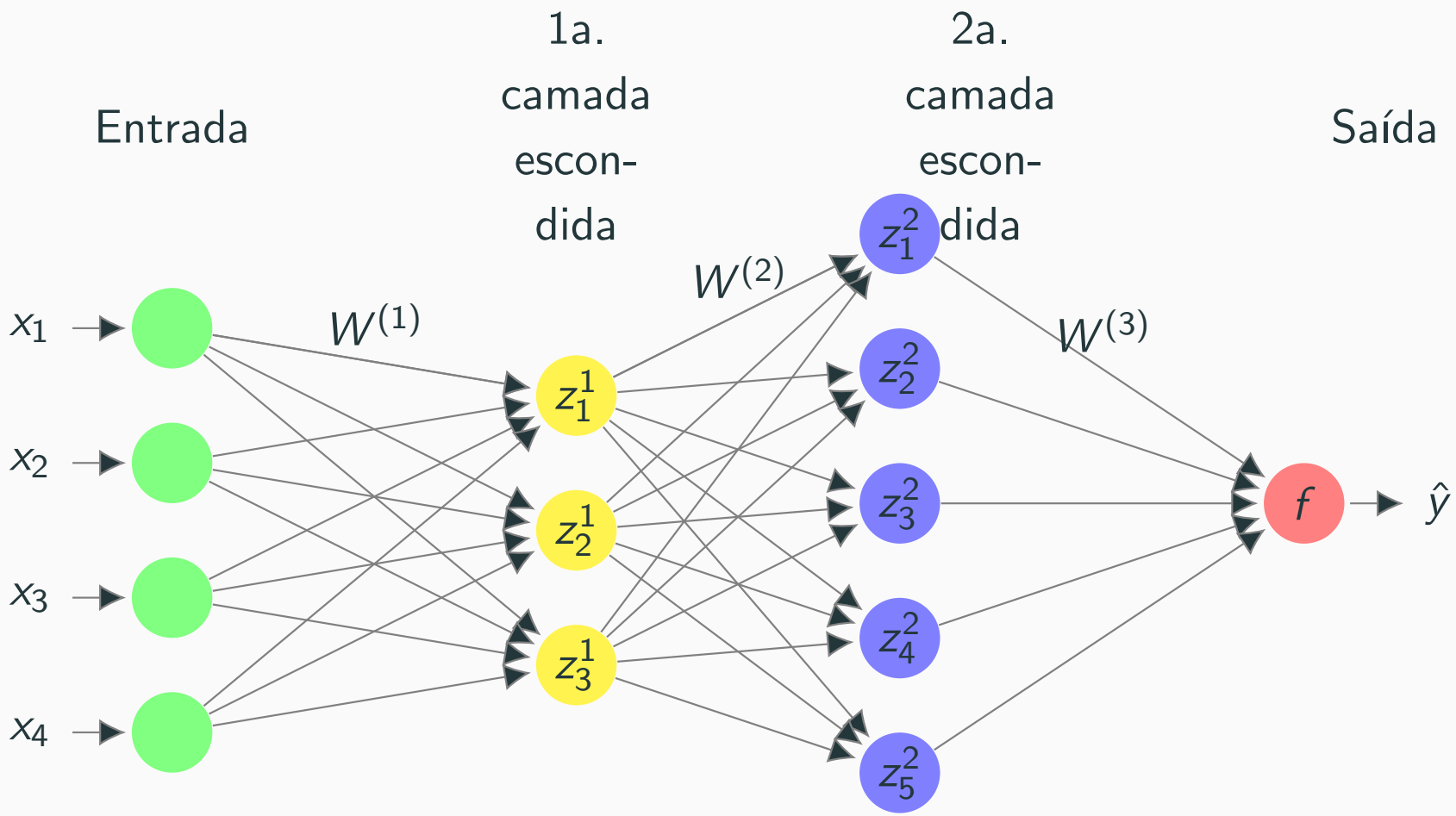
Biblioteca Keras

Utilizaremos a biblioteca Keras do Python:

- Permite construir, treinar e executar Redes Neurais diversas (Deep Learning).
- Escrita em Python e permite modelos de configurações avançados.
- Usa Tensorflow ou Theano, serve apenas como *frontend*.
- Permite o uso de CPU ou GPU para processar.
- Usa estrutura de dados do *numpy* e estrutura de comandos similar ao *scikit-learn*.

Keras

Vamos criar a seguinte rede de exemplo no Keras:



UFABC

Keras

Primeiro criamos o modelo do tipo *Sequential* que cria cada camada da rede em sequência:

```
from keras.models import Sequential
model = Sequential()
```

UFABC

Keras

Em seguida importamos as funções *Dense* para criação de camadas densas e *Activation* que define a função de ativação nos neurônios:

```
from keras.layers import Dense, Activation
```

Keras

Finalmente, criamos as camadas da primeira até a última na sequência. Note que para a primeira camada devemos especificar a dimensão de entrada:

```
model.add(Dense(units=3, input_dim=4))
model.add(Activation('tanh'))

model.add(Dense(units=5))
model.add(Activation('tanh'))
```

Keras

A camada de saída para problemas de regressão deve ter um neurônio e ativação linear:

```
model.add(Dense(units=1))
model.add(Activation('linear'))
```

Keras

A camada de saída para problemas de classificação binária deve ter um neurônio e ativação sigmoid:

```
model.add(Dense(units=1))
model.add(Activation('sigmoid'))
```

Keras

A camada de saída para problemas de classificação multi-classes deve ter um neurônio e ativação softmax:

```
model.add(Dense(units=1))
model.add(Activation('softmax'))
```

Keras

Uma vez que a rede está construída, podemos *compilá-la* e definir a função de erro e o algoritmo de treinamento:

```
model.compile(loss='mean_squared_error',
              optimizer='sgd'
              metrics=['accuracy']
             )
```

UFABC

Keras

Possíveis funções de erro:

- mean_squared_error: erro quadrático médio.
- squared_hinge: maximiza a margem de separação.
- categorical_crossentropy: minimiza a entropia, para multiclasses.

Veja mais em:
https://github.com/keras-team/keras/blob/master/keras/activations.py.

Keras

Possíveis algoritmos de otimização:

- sgd: Stochastic Gradient Descent.
- rmsprop: ajusta automaticamente a taxa de aprendizado.
- adam: determina a taxa de aprendizado ótima de cada iteração.

Veja mais em:
https://github.com/keras-team/keras/blob/master/keras/optimizers.py.

Keras

Possíveis métricas de avaliação:

- sgdmean_squared_error: Erro quadrático médio.
- mean_absolute_error: Erro absoluto médio.
- binary_accuracy: Acurácia para classes binárias.
- categorical_accuracy: Acurácia para multi-classes.

Veja mais em:
https://github.com/keras-team/keras/blob/master/keras/metrics.py.

Keras

Finalmente, o ajuste da rede é feita com:

```
model.fit(x_train, y_train, batch_size=n, epochs=max_it)
score = model.evaluate(x_val, y_val)
y_pred = model.predict(x_test)
```

UFABC

Notas

Não se esqueçam:

- Use a classe *StandardScaler* da biblioteca *sklearn.preprocessing* para escalonar as variáveis e deixá-las com média 0.
- Atributos categóricos e rótulos de classes devem ser transformados em vetores binários utilizando a técnica One-Hot-Encoding (classe OneHotEncoder).

Próxima Aula

Na próxima aula aprenderemos os conceitos básicos de Aprendizado Não-Supervisionado.

UFABC

Atividade 06

Complete os Laboratórios:

04_Keras_First_NN.ipynb

UFABC