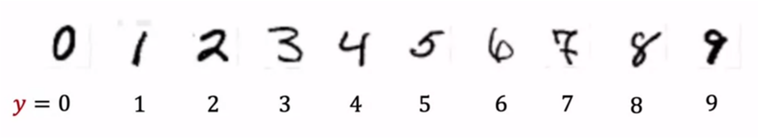
Problemas de Classificação com múltiplas classes

Onde estamos e para onde vamos

Nas aulas anteriores, aprendemos bastante sobre classificação binária. Agora falaremos sobre como resolver problemas de classificação onde temos múltiplas classes.

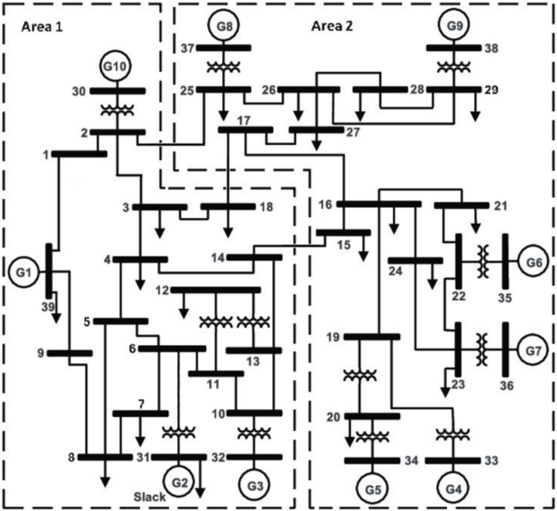
Classificação com múltiplas classes (multi-classe)

Voltando para o exemplo de reconhecimento de dígitos escritos à mão



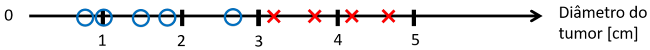
- Em problemas de classificação binária, y possui apenas dois valores possíveis: 0 ou 1.
- Em problemas de classificação com múltiplas classes, mais classes podem existir...

Localização de faltas em sistemas elétricos de potência



OBS: E se quisermos dividir o sistema em mais áreas? $y = 1, 2, 3, 4, 5, 6$**?**

- A Regressão **Softmax** é uma generalização do método de Regressão Logística, que se restringia a problemas de classificação binária.



Regressão Logística
(apenas duas saídas possíveis $y = 0, 1$)

Calculamos $z = \vec{w} \cdot \vec{x} + b$, e depois fazemos:

$$a_1 = g(z) = \frac{1}{1 + e^{-z}}$$

Onde a_1 pode ser interpretado como sendo

$$a_1 = P(y = 1|\vec{x})$$

Como calculamos $P(y = 0|\vec{x})$?

Simples,

$$a_2 = P(y = 0|\vec{x}) = 1 - a_1$$

Exemplo: Se $a_1 = 0{,}63$, quanto vale a_2?

Regressão Logística *versus* Softmax

Regressão Logística
(apenas duas saídas possíveis $y = 0, 1$)

Calculamos $z = \vec{w} \cdot \vec{x} + b$, e depois fazemos:

$$a_1 = g(z) = \frac{1}{1 + e^{-z}}$$

Onde a_1 pode ser interpretado como sendo

$$a_1 = P(y = 1|\vec{x})$$

Como calculamos $P(y = 0|\vec{x})$?

Simples,

$$a_2 = P(y = 0|\vec{x}) = 1 - a_1$$

Exemplo: Se $a_1 = 0{,}63$, quanto vale a_2?

Regressão Softmax
(exemplo com $y = 1, 2, 3, 4$)

Calculamos

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$$
$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2$$
$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$$
$$z_4 = \vec{w}_4 \cdot \vec{x} + b_4$$

Então fazemos:

$$a_1 = P(y = 1|\vec{x}) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
$$a_2 = P(y = 2|\vec{x}) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
$$a_3 = P(y = 3|\vec{x}) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$
$$a_4 = P(y = 4|\vec{x}) = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

Exemplo: Se $a_1 = 0.2$, $a_2 = 0.35$, $a_3 = 0.15$, quanto vale a_4?

Generalização da Regressão Softmax

(exemplo com N classes, tal que $y = 1, 2, \ldots, N$)

Calculamos

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \qquad \text{para } j = 1, \ldots, N$$

Então fazemos:

$$a_j = P(y = j|\vec{x}) = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$$

onde $a_1 + a_2 + \cdots + a_N = 1$

Como fica a função custo para a Regressão Softmax?

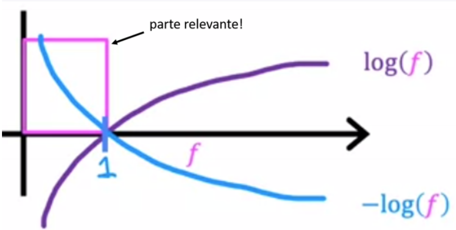
Lembrando que, na Regressão Logística, tínhamos

Duas classes $(N = 2)$:

$$a_1 = P(y = 1|\overrightarrow{x}) \quad \text{e} \quad a_2 = P(y = 0|\overrightarrow{x}) = 1 - a_1$$

$$\textbf{perda} = \textbf{função de entropia cruzada binária} = \begin{cases} -\log a_1 & \text{, se } y^{(i)} = 1 \\ -\log a_2 & \text{, se } y^{(i)} = 0 \end{cases}$$

$$\textbf{função custo} = J(\overrightarrow{w}, b) = \textbf{média das perdas}$$



Como fica a função custo para a Regressão Softmax?

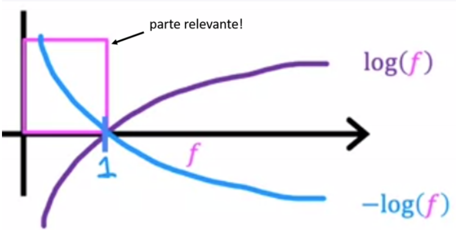
Transferindo essa ideia para a Regressão Softmax, temos

N **classes**:

$$a_1 = P(y = 1|\overrightarrow{x}) \qquad a_2 = P(y = 2|\overrightarrow{x}) \qquad \cdots \qquad a_N = P(y = N|\overrightarrow{x})$$

$$\textbf{perda} = \textbf{função de entropia cruzada para } N \textbf{ classes} = \begin{cases} -\log a_1 & \text{, se } y^{(i)} = 1 \\ -\log a_2 & \text{, se } y^{(i)} = 2 \\ \cdots & \cdots \\ -\log a_N & \text{, se } y^{(i)} = N \end{cases}$$

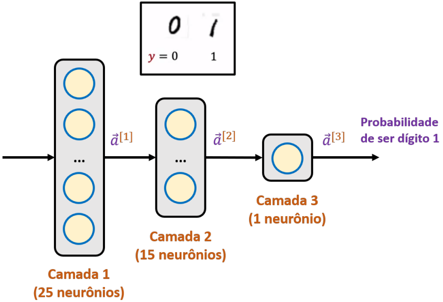
$$\textbf{função custo} = J(\overrightarrow{w}_1, b_1, \cdots, \overrightarrow{w}_N, b_N) = \textbf{média das perdas}$$



Redes Neurais com saída Softmax

Redes Neurais com saída Softmax

Para que seja possível usar redes neurais no contexto de classificação multi-classe, podemos inserir o modelo de **regressão Softmax na camada de saída** da rede.



Pergunta:

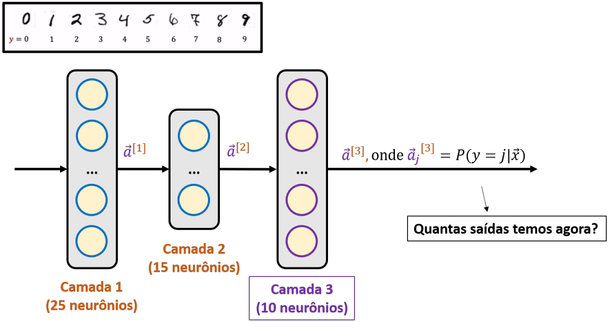
Como fazíamos para calcular $\vec{a}^{[3]}$ quando a camada de saída tinha a função sigmoide?

Resposta:

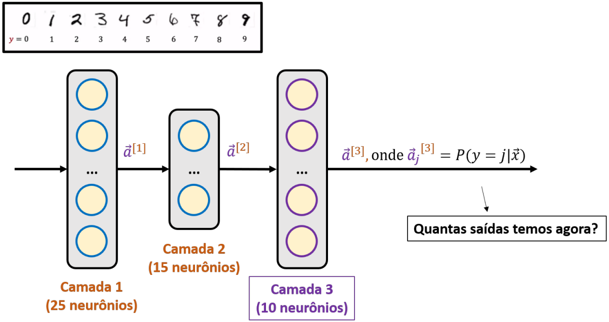
Fazíamos $z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}$ e depois

$$\vec{a}_1^{[3]} = g(z_1^{[3]}) = P(y = 1|\vec{x})$$

Agora, no problema multi-classe, podemos usar a camada de saída Softmax

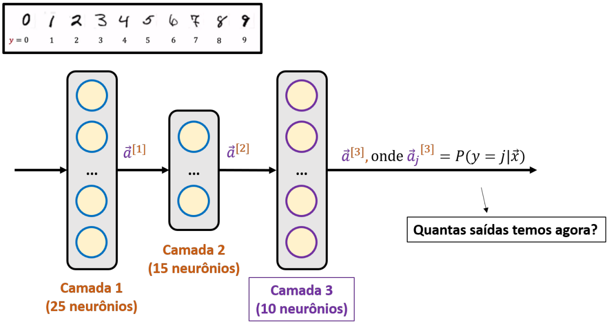


Agora, no problema multi-classe, podemos usar a camada de saída Softmax



Agora, com a função de ativação Softmax na camada de saída, calculamos $z_j^{[3]} = \vec{w}_j^{[3]} \cdot \vec{a}^{[2]} + b_j^{[3]}$, para $j = 1, \cdots, 10$ e depois calculamos cada $\vec{a}_j^{[3]} = P(y = j|\vec{x})$ como sendo

$$\vec{a}_j^{[3]} = \frac{e^{z_j}}{e^{z_1} + e^{z_2} + \cdots + e^{z_{10}}}$$



Observação final (*nível hard*):

Note que $\vec{a}_j^{[3]}$ é função de $z_1, z_2, \cdots, z_N$.

- Isso é uma característica interessante que diferencia a ativação Softmax das demais ativações vistas anteriormente (sigmoide, relu e linear), onde $\vec{a}_j$ é função tão somente de z_j. Por exemplo, para a função sigmoide, teríamos:

$$\vec{a}_j^{[3]} = \frac{1}{1 + e^{-z_j}}$$

Softmax no Tensorflow

Implementação intuitiva:

```
modelo = Sequential(
    [
        Dense(units=25, activation="relu"),
        Dense(units=15, activation="relu"),
        Dense(10, activation="softmax")
    ]
)
modelo.compile(
    loss=SparseCategoricalCrossEntropy()
)
modelo.fit(
    X,y,epochs=50
)
```

10 unidades softmax na camada de saída

Função custo que definimos anteriormente

Implementação mais robusta numericamente:

```
modelo = Sequential(
    [
        Dense(units=25, activation="relu"),
        Dense(units=15, activation="relu"),
        Dense(10, activation="linear")
    ]
)
modelo.compile(
    loss=SparseCategoricalCrossEntropy(from_logits=True)
)
modelo.fit(
    X,y,epochs=50
)
logits = model(X)
f_x    = tf.nn.softmax(logits)
```

Definimos uma função linear aqui

Isso faz com que as saídas da rede neural agora sejam z_j (logit numbers)

Mudamos aqui para que a ativação softmax passe a ser feita diretamente junto ao cálculo da função de perda

Aqui calculamos as saídas z_j e aplicamos a softmax para obter de fato as probabilidades

De olho no código!

Iremos agora verificar como implementamos a função Softmax na prática, assim como sua integração junto ao Tensorflow.

Nome do arquivo que trabalharemos agora:

codigo - classificacao multi-classe com tensorflow.ipynb

Atividade de aula

Parte 1

Rode todo o "codigo - classificacao multi-classe com tensorflow.ipynb" sem fazer qualquer tipo de alteração. Certifique-se de que você o compreendeu.

Parte 2

1. Compare a taxa de acerto obtida pela implementação mais robusta com a taxa de acerto obtida pela implementação intuitiva. Há garantia de que a implementação mais robusta proverá maior taxa de acerto? Para responder a essa pergunta, lembre-se, por exemplo, da inicialização aleatória dos parâmetros de uma rede neural ao se iniciar seu treinamento.