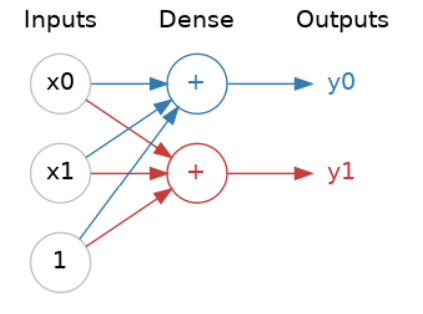
# Redes Neurais Profundas

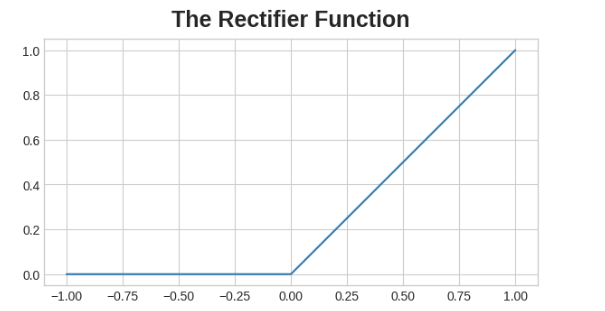
Como vimos anteriormente, a menor parte das redes neurais é o neurônio, para formar redes neurais mais eficientes e que aprendam melhor, temos que criar diversas camadas com vários neurônios, exemplo:



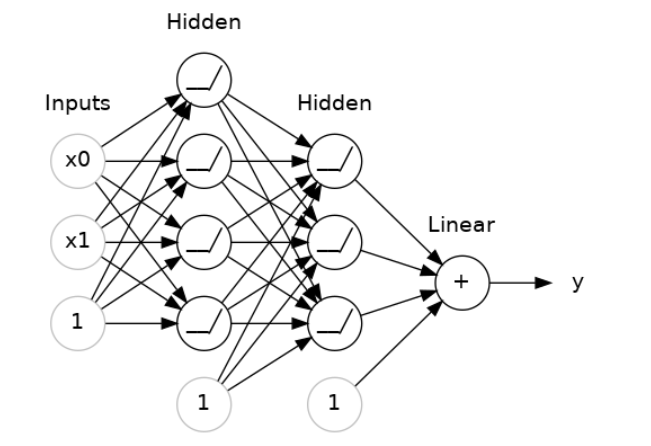
Como ambos os neurônios compartilham os mesmos dados de entrada, podemos chamar a camada que eles estão de camada densa. Em grandes redes neurais, teremos várias camadas, nas quais cada uma delas fazem uma transformação que nos aproxima mais do resultado final.

É importante ressaltar que como podemos descrever a saída de um neurônio como uma função linear das entradas, pesos e viés, ele consegue detectar relações lineares nos dados, o que é péssimo para um modelo de deep learning, para consertar isso, vamos colocar uma função de ativação em cada neurônio.

A função de ativação mais comum que colocamos em cada neurônio é max(0, y), eliminando os possíveis valores negativos.



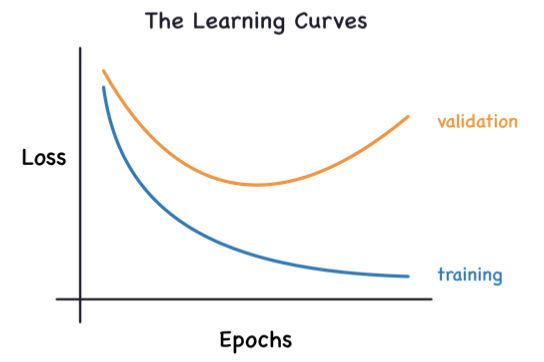
Só começa a ter valores diferentes de zero quando o output do neurônio fica maior que zero. Essa função de ativação é chamada de ReLU (Rectifier Linear Unit). Com isso, podemos formar camadas mais densas que serão diferentes e vão encontrar padrões não lineares.



Nesse exemplo maior, podemos identificar as camadas de neurônios hidden, que são aquelas em que não vemos o output diretamente, o output delas é usado em outras camadas de neurônios que vão mostrar outros outputs e por aí vai.

**TÉCNICAS PARA EVITAR OVERFITTING E UNDERFITTING**

Underfitting acontece quando o modelo não consegue se adaptar tanto a base de treino quanto a de teste, para isso, podemos aumentar o número de camadas ou o número de neurônios por camada. O overfitting acontece quando o modelo prevê muito bem os de treino mas muito mal os de teste, ele está apenas decorando os dados de treino. Quando deixamos o modelo treinar por muito tempo, ele começa diminuindo os erros, mas após um certo tempo, ele volta a aumentar, isso acontece pois ele começou a decorar os ruídos nos dados de treino, gerando um overfitting.



Podemos observar que ao aumentar o número de épocas, o modelo melhora tanto pro treino quanto para o teste, mas após passar de um certo limiar, ele começa a decorar os dados do treino, fazendo com que melhore a loss para o treino, mas pior para o teste.

**CAPACIDADE DO MODELO**

Redes neurais mais amplas são aquelas com mais neurônios por camadas, muito boas para pegar padrões lineares nos dados. Por outro lado, temos os modelos mais profundos, que são aqueles com menos neurônios por camada, mas com mais camadas, sendo melhores para pegar padrões não lineares.



**EARLY STOPPING**

Como vimos anteriormente, o ponto para pararmos de treinar o nosso modelo é quando os dados de validação começam a ter um aumento repentino na loss function, indicando overfitting. Devemos parar o nosso modelo quando o erro estiver começando a subir, justamente é isso que o early stopping faz.

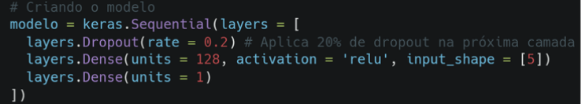


O tipo callback de função é aquele que só irá executar as vezes durante o treinamento, no caso do early stopping, vai executar em cada iteração(época).

**DROPOUT & BATCH NORMALIZATION**

Além de camadas de hidden layers, existem diversos outros tipos de camadas no deep learning como Convolucionais, LSTM, Recorrentes, Dropout e Batch Normalization, algumas podem até fazer pré-processamento e diversas outras ações.

Falando sobre a camada dropout, ela basicamente vai, a cada iteração, tirar um conjunto aleatório de neurônios, fazendo com que seja mais difícil o modelo aprender os ruídos dos dados testes e, por consequência, ajudar no overfitting. Uma vez que o overfitting é quando temos um conjunto de neurônios pesos e vieses que aprendem o ruído, então se retirarmos esse ruídos, o modelo tende a voltar ao normal.



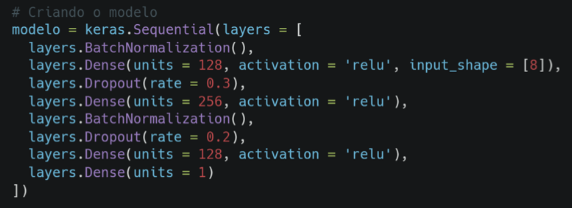
Sempre colocar a camada dropout antes da camada que queremos aplicar a técnica!

A técnica de Batch normalization é basicamente uma camada que vai normalizar os dados, como o nome já diz, uma vez que os algoritmos de deep learning e a otimização feita pelo gradiente descendente é muito sensível a escala dos dados, então é muito importante deixar os dados na mesma escala. O que faz essa camada ser melhor do que normalizar os dados antes com métodos como StandardScaler() é que vamos normalizar a cada iteração do modelo.



Poderíamos fazer a normalização antes da primeira camada densa também.

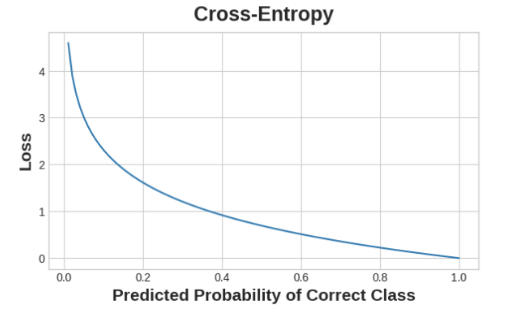
Juntando as duas novas camadas com um modelo de redes neurais de 3 camadas hidden, com 128, 256 e 128 neurônios cada.

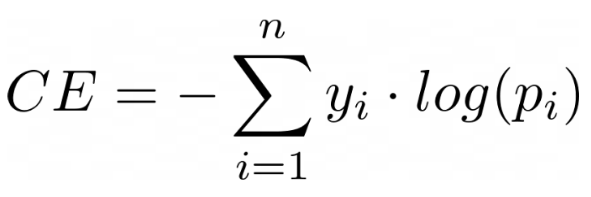


**PROBLEMAS DE CLASSIFICAÇÃO**

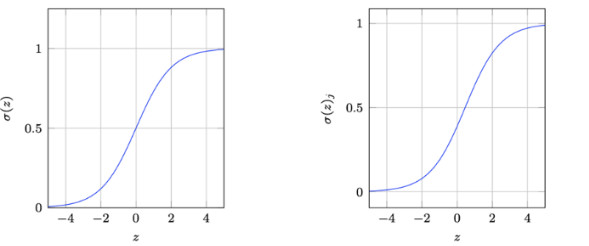
Até agora só tínhamos visto aplicações para modelos de regressão, para alterar para classificação, basta adicionar uma função de ativação na camada de output, uma vez que o modelo de Deep learning retorna um valor contínuo, então temos que converter para classe. Além disso, precisamos mudar a nossa loss function, estamos usando MAE.

cross-entropy Loss Function: Basicamente vai ver a probabilidade do modelo acertar uma certa classe(estilo predict proba). Fazendo isso vamos ter uma ideia maior do quanto o modelo está perto de classificar corretamente.



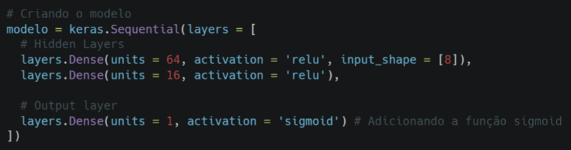


* Yi é a classe prevista(0 ou 1)
* Pi é a probabilidade de saída, geralmente uma sigmoid para apenas duas classes e softmax para multiclasse



Em ML o valor dessas funções é entre 0 e 1, estilo predict proba. Essa função sigmoid vai receber o output do modelo de deep learning e converter para uma certeza, que é a probabilidade dele ir para uma classe. Podemos também definir o threshold para ele ir para uma classe positiva(default é 0.5).

Primeiramente criamos a rede neural com a função de ativação sendo sigmoid, caso seja multiclasse botar softmax



Depois compilamos o modelo com a nova Loss Function para classificação e a métrica sendo acurácia:

