Inteligência artificial e sistemas cognitivos

Índice

Introdução

Projeto 1 – Redes Neuronais Artificiais

# Redes Neuronais Artificiais

Introdução

Para poder construir uma rede neuronal artificial (RNA) é primeiro preciso perceber a sua composição e comportamento.

Uma rede neural é um modelo computacional inspirado no funcionamento do cérebro humano. Assim como o cérbero humano, estas redes são compostas por múltiplas unidades individuais de processamento, designadas **neurónios**, interligadas entre si e organizadas em camadas.

As características das RNAs tornam-lhes uma ótima solução para a resolução de problemas complexos, onde métodos tradicionais enfrentam limitações, especialmente em contextos que envolvem dados não estruturados e relações não lineares, como reconhecimento de padrões e processamento de linguagem natural.

Neurónio artificial

O neurónio artificial é inspirado no funcionamento dos neurónios biológicos, o mesmo é composto por quatro elementos principais:

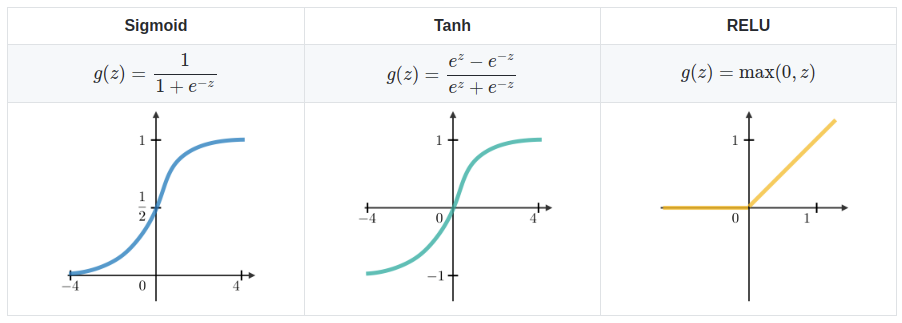
* **Entradas, x** – Sinais de entrada que chegam ao neurónio através de ligações com outros neurónios ou com as entradas da rede.
* **Pesos, w** – Valores que representam o peso, ou seja, a influência de cada ligação para a resposta do neurónio.
* **Função de transferência, f** – Função que determina o processamento das entradas para gerar a saída do neurónio, tipicamente são utilizadas funções degrau, sigmoid, tangente hiperbólica e ReLU (Rectified Linear Unit). Esta mal
* **Saída, y** – Sinal de saída do neurónio, o qual pode ser enviado a outros neurónios da rede ou ser diretamente a saída da rede.

Uma imagem com file, diagrama, Tipo de letra, círculo

Descrição gerada automaticamente

O neurónio tem como entrada os valores de x, tipicamente um vetor. Estes valores têm associados um vetor de pesos, w, responsável por determinar a influencia de cada entrada. h representa a soma ponderada da entrada pelos respetivos pesos ou o produto interno entre os vetores x e w. O valor do produto interno representa a semelhança entre os vetores de entrada, x, e o vetor de pesos, w. Ou seja, quanto mais semelhança entre estes dois vetores (maior produto interno) maior é a possibilidade de ativação do neurónio. Durante a aprendizagem os pesos das ligações vão sendo ajustados de forma que o valor de saída a aprender seja próximo do valor de saída gerado.

A função de transferência f é responsável por aplicar uma função de ativação, sigma, à soma de ativação h. Esta função desempenha um papel fundamental, é responsável por introduzir um caracter de resposta não linear, permitindo a rede neuronal lidar com problemas mais complexos. Para além disso, limita a saída do neurónio para um intervalo específico, como [0, 1] para a função sigmoid ou [-1, 1] para a função tangente hiperbólica. Isto faz com que a saída do neurónio não cresça indefinidamente, o que poderia causar problemas de estabilidade na rede.



Por último é também importante falar do pendor de ativação, b. Este pendor é adicionado à soma de ativação, antes de entrar na função de ativação, e permite ajustar o limiar de ativação da rede. Geometricamente representa a deslocação de fronteira de discriminação por . Este pendor permite que a rede tenha mais flexibilidade ao adaptar-se aos dados durante o treino.

Uma imagem com file, diagrama, círculo, Tipo de letra

Descrição gerada automaticamente

Redes Neuronais Multicamada

Percebendo agora o funcionamento de um neurónio, avançamos para as redes neuronais multicamada. Uma camada é composta por múltiplos neurónios, a sua função é processar informação e enviar uma resposta para outra camada. Portanto, uma Rede Neural Multicamada, como o nome sugere, é uma rede compostas por múltiplas destas camadas, onde o número de camadas, e o número de neurónios nas mesmas, varia conforme a complexidade do problema.

Existem diferentes tipos de camadas numa rede neuronal, cada uma com papeis diferentes, estas são:

* **Camada de Entrada -** Camada inicial da rede, responsável por receber os dados de entrada
* **Camada Escondida –** Esta camada, tipicamente existindo mais que uma, fica entre a camada de entrada e camada de saída. É responsável por processar os dados recebidos da camada anterior e enviar o resultado para a camada seguinte.
* **Camada de Saída –** Camada final da rede, que produz e exibe o resultado final da rede com base no processamento das camadas anterioresº

Uma imagem com texto, diagrama, captura de ecrã, file

Descrição gerada automaticamente

Camada de Entrada

Como previamente descrito é nesta camada que se dá o início do processamento dos dados de entrada. Cada neurónio, dessa camada, representa uma variável do problema em questão. No exemplo de processamento de uma imagem, cada pixel irá responder a um neurónio desta camada.

Camada Escondida

Esta camada, tipicamente existindo mais que uma, fica entre a camada de entrada e camada de saída. É responsável por processar os dados recebidos da camada anterior e enviar o resultado para a camada seguinte.

O principal objetivo das camadas escondidas é evitar a linearidade, a utilização das funções de ativação dos múltiplos neurónios das camadas escondidas, permite à rede capturar relações não lineares que um simples modelo linear não era capaz de detetar. Num exemplo de reconhecimento facial, utilizando uma rede neuronal, as camadas escondidas mais próximas da entrada aprendem características básicas como identificar formas simples (lábios, nariz), mas à medida que evoluímos para as camadas mais próximas da saída estas tornam-se capazes de aprender padrões mais complexos, como o rosto completo.

Camada Saída

É a camada final da rede, que produz e exibe o resultado final da rede com base no processamento das camadas anteriores, o número de neurónios desta camada depende da complexidade do problema.

A figura x, representa um exemplo de uma rede XOR, com o conhecimento prévio do comportamento do neurónio e com a introdução às redes neuronais multicamada é claro o funcionamento desta rede. O vetor de saída y é gerado após a função de ativação ser aplicada à soma ponderada das entradas pelos respetivos pesos, w, e somados os pendores, b.

Uma imagem com diagrama, texto, círculo, file

Descrição gerada automaticamente

Características

Para concluir é retirado algumas conclusões e características:

**Relações Não Lineares** – Graças às múltiplas camadas ocultas e funções de ativação, estas redes tem a capacidade de capturar relações não lineares complexas.

**Tolerância a Ruido** – A estrutura destas redes e capacidade de generalização ajudam a minimizar o impacto do ruido no resultado final.

**Escalabilidade e flexibilidade** – Estas redes podem ser ajustadas para redes mais complexas, aumentando o número de camadas e neurónios, o que lhes torna uma ótima solução para diversos tipos de problemas incluindo aprendizagem supervisionada e aprendizagem não-supervisionada.

Desafios

Contudo estas redes também apresentam alguns problemas como:

Necessidade de grande quantidade de dados – O treino destas redes exige conjuntos de dados muito extensos, para aumentar a sua eficiência.

Tempo de treino- Redes muito profundas e complexas tipicamente requerem longos períodos de treino, o que pode não ser ideal.

Sensibilidade a Exemplos Enganadores- Embora robustas a ruídos gerais, as redes neuronais podem ser vulneráveis a exemplos enganadores, dados criados especificamente para enganar o modelo.

Aprendizagem em Redes Neuronais

Dois tipos principais de aprendizagem em redes neuronais: Supervisionada e Não-Supervisionada

Aprendizagem Supervisionada

A aprendizagem supervisionada é um tipo de aprendizagem onde a rede é treinada com dados etiquetados. Por exemplo, um responsável pelo treino da rede fornece uma imagem com uma forma de três lados, e indica que a mesma é um triangulo. O mesmo é feito para outras formas, e após este treino o algoritmo eventualmente consegue receber uma nova entrada e independentemente prever a etiqueta(forma), correta. Este modelo é ideal para, por exemplo, classificação binaria e modelos de regressão.

Aprendizagem Não-Supervisionada

Como o nome sugere, ao contrário da aprendizagem supervisionada, esta não requer qualquer tipo de etiqueta para os dados. A rede aprende representações de agrupamentos (cluster) de dados similares ou características e padrões comuns. Este tipo de aprendizagem é ideal para *clustering*, deteção de anomalias e regras de associação.

Aprendizagem por Correção do Erro

Esta aprendizagem é do tipo supervisionada, é conhecida por ajustar os parâmetros do modelo, como os pesos, com base no erro observado entre a saída prevista pelo modelo e a saída desejada. O objetivo desta estratégia é minimizar o erro durante as iterações na rede, até que o modelo obtenha resultados mais precisos.

A figura x ilustra esse comportamento, onde para cada valor x e resposta yn é indicada à rede a resposta alvo y. A aprendizagem consiste na minimização da função de perda L(y, yn) entre a resposta yn(resposta atual da rede) e y (resposta desejada ou alvo).

Uma imagem com texto, file, diagrama, captura de ecrã

Descrição gerada automaticamente

Algoritmo de Retropropagação

O objetivo é treinar uma rede de forma a minimizar a função de perda L(teta) previamente vista. É um problema de otimização onde se procura encontrar os parâmetros ideais, teta, que minimizem a função de perda.

Descida de gradiente

Os parâmetros ideais teta são atualizados em função do gradiente da função de perda L(teta), no sentido oposto ao gradiente. Esta atualização é dada por:

θ←θ−α∇L(θ)

Onde:

alfa, é a taxa de aprendizagem, que determina a dimensão dos passos do algoritmo de otimização

deltaL(teta) é o gradiente da função de perda

Uma imagem com desenho, texto, esboço, desenhos de criança

Descrição gerada automaticamente

A atualização dos parâmetros é repetida até que a função de custo atinja um valor mínimo, idealmente global, ou até atingir o numero máximo de iterações. Nem sempre o algoritmo encontra o mínimo global, em alguns casos a solução converge para um mínimo local.

Concluindo, começa-se por fazer uma propagação direta. A entrada x passa pelas camadas até gerar a resposta yn. De seguida é calculado o erro usando a função de perda. Através da retropropagação o erro é propagado de trás para frente da rede e é calculado o gradiente da função de perda. Por último, através da descida do gradiente são ajustados os parâmetros (pesos e pendores) de cada camada da rede neuronal.

Algoritmo de Descida Gradiente

Como foi visto anteriormente, apesar de muito útil, este algoritmo apresenta alguns problemas comuns:

Ótimos Locais – O algoritmo pode convergir para mínimo local em vez do mínimo global.

Convergência Lenta – Em zonas de gradiente baixo ou nulo, ou se alfa por pequeno, o algoritmo pode ter uma convergência lenta.

Oscilações – Taxas de aprendizagem altas, podem provocar oscilações, impedindo a convergência.

Algoritmo de Descida Gradiente Com Momento

Para resolver o problema de oscilações e convergências em regiões planas é apresentado o algoritmo de descida gradiente com momento.

Semelhante ao algoritmo de descida gradiente, mas com a introdução do momento. Este momento adiciona uma memoria ao processo de atualização ao acumular as atualizações anteriores, funciona como uma inercia que ajuda a ultrapassar regiões planas, de gradiente baixou ou nulo.

Δx n = - α∇f(x) + βΔx n-1

Uma imagem com diagrama, file, texto, Gráfico

Descrição gerada automaticamente

Descida de Gradiente Estocástica

Este algoritmo utiliza subconjuntos aleatórios(batches) do conjunto de dados a cada iteração introduzindo aleatoriedade e visa ultrapassar o problema dos mínimos locais.

Em cada época de treino é utilizado apenas um batch do conjunto de dados, as atualizações são feitas após cada um destes batches resultando num treino mais rápido e introduzindo uma componente aleatória na exploração do espaço dos pesos.

Processo de aprendizagem

Para garantir um bom resultado com redes neuronais é necessário um pre-processamento dos dados, isto envolve: Limpeza de dados, normalização, codificação e divisão dos dados (separação do conjunto de treino e teste). Após este passo, é feito uma extração de características, isto é, identificar as variáveis mais importantes para o problema ex. transformações matemáticas, como a introdução de não linearidades para facilitar a aprendizagem de relações não lineares entre as características e as variáveis alvo. Por último, é realizado o treino da rede onde são ajustados os pesos e pendores para minimizar a função de perda e melhorar o desempenho da rede. Alguns aspetos principais:

Definir a estrutura da rede – Definir número de camadas, numero de neurónios e o tipo de função de ativação.

Função de perda- Escolher a função de perda mais adequada para o problema.

Método de Otimização – Escolher o algoritmo de otimização que mais se enquadre no problema, como, Descida Gradiente Com Momento ou Descida de Gradiente Estocástica

Hiperparâmetros – Definir, taxa de aprendizagem, número de épocas e tamanho dos batches.

Arquitetura da Rede Neuronal

A definição da arquitetura de uma rede neuronal é uma etapa crucial para o desenvolvimento de modelos capazes de resolver os problemas em questão. A escolha da rede neuronal inclui uma componente experimental, já que as características das representações a serem aprendidas nem sempre são totalmente conhecidas.

Principais aspetos da arquitetura

**Número de neurónios de entrada**-Corresponde ao número de variáveis do conjunto de dados, ou seja, cada neurónio da camada mapeia diretamente uma característica dos dados. Ex: Num problema que envolva uma classificação de uma imagem de 10X10 pixels (100 características), o número de neurónios deve ser 100.

**Número de neurónios de saída** – Essencialmente depende da tarefa que se vai realizar, podendo estas ser: classificação binária, onde existe um único neurónio, multi-classe, em que o número de neurónio de saída é igual ao numero de classes e regressão, onde existe apenas um neurónio de saída que representa o valor a prever.

**Número de neurónios escondidos**-Resumidamente o numero ideal de neurónios escondidos depende do problema e dos dados, e geralmente é adaptado com experiencia. Contudo, uma prática comum é número de neurónios escondidos ≈ (número de entradas + número de saídas) / 2.

**Número de camadas escondidas**- Depende da complexidade do problema, mais camadas permitem que a rede aprenda representações mais complexas. No entanto, redes com muitas camadas podem sofrer sobreparametrização(overfitting).

Sobreparametrização

Este é um problema comum que acontece quando a rede se adapta demasiado bem aos dados de treino, aprendendo não só padrões gerais, mas como também ruido e discrepâncias levando assim a uma diminuição da capacidade de generalização da rede. A figura x ilustra um exemplo de sobreparametrização

Algumas soluções para evitar este problema são reduzir o numero de camadas, interromper o treino quando o erro de validação começa a aumentar e aumentar o conjunto de dados.

Aprendizagem profunda (deep learning)

É um subcampo de aprendizagem automática que utiliza redes neuronais profundas (múltiplas camadas escondidas) para aprender padrões e relações complexas a partir de um grande conjunto de dados. É inspirada na estrutura de córtex cerebral, que possibilita a formação de representações dos dados em diferentes níveis de abstração. São tipicamente utilizadas em reconhecimento de imagem, reconhecimento de voz chatbots e veículos autónomos.

Redes Convolucionais (CNNs)

São redes especializadas no processamento de dados estruturados espacialmente, como imagens, sendo por isso muito utilizadas em tarefas relacionadas com o processamento de imagem, por exemplo, deteção de objetos. Algumas vantagens das redes convolucionais são: Invariância à Translação, Capacidade de reconhecer padrões independentemente da posição nos dados. Redução de parâmetros, a reutilização de filtros reduz o número total de parâmetros. Extração automática de características, dispensa a necessidade de projetar manualmente características relevantes.

Conclusão

As redes neuronais artificiais são um modelo inspirado na estrutura e no funcionamento do sistema nervoso biológico. São compostas por elementos interligado entre si, neurónios artificiais, que processam informações em paralelo e colaboram para resolver problemas de alta complexidade. Algumas características são: capacidade de aprendizagem, capacidade de generalização, adaptabilidade, computação e representação distribuída, paralelismo massivo e robustez.

Implementação

Conhecendo agora a composição e comportamento de uma rede neuronal artificial, foi desenvolvida uma biblioteca em python, de uma rede neuronal, baseada no conhecimento que foi estudado.

Começou-se por criar a classe **Camada de Entrada**, o ponto inicial da rede neuronal, e sua função é simplesmente armazenar e propagar os valores de entrada para a camada seguinte.

Métodos:

***\_\_init\_\_(sefl, ds):*** Inicializa a camada com ***ds***, (numero de neurónios de entrada).

***Propagar(self, x):*** Responsável por propagar a entrada, ***x***, para a próxima camada.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

De seguida foi criada a classe **Neurónio**, que tem como objetivo representar um neurónio artificial

**Atributos:**

***Phi*** -Função de ativação.

***w -*** Vetor de pesos das ligações de entrada

***b-*** Pendor interno do neurónio.

***delta\_w, delta\_b-*** Mudanças acumuladas para pesos e pendores usadas no cálculo do momento.

***h -***Soma ponderada da ativação das entradas

***y-*** Saída do neurônio apos a aplicação da função de ativação

***y\_linha-***Derivada da função de ativação em relação a h.

**Métodos:**

***\_\_init\_\_(self, d, phi):*** Inicia os atributos do neurónio e tem como entrada a dimensão(número ligações de entrada), ***d***, e a função de ativação, ***phi***. Recorre também à biblioteca numpy para gerar valores aleatórios para os pesos, ***w***, e pendor, ***b***, desta forma evita-se aprendam padrões idênticos permitindo que cada neurónio aprenda de forma diferente. É garantido também que ***delta\_w*** e ***delta\_b*** sejam iniciados a zero pois estas variáveis são responsáveis por armazenar as atualizações.

***propagar(self, x):*** Responsável por calcula a soma ponderada h, e aplicar a função de ativação, e armazena a o calculo da derivada.

***adaptar(self, delta, y\_menos\_1, alpha, beta):*** Atualiza os pesos e pendor aplicando algoritmo de descida gradiente com momento.

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamente

É importante referir que também foram criadas funções globais:

***sig(x):*** Função sigmoid, usada como função de ativação padrão

***deriv(f, x):*** Calcula a derivada de uma função

Uma imagem com texto, Tipo de letra, captura de ecrã, escrita à mão

Descrição gerada automaticamente

Nesta biblioteca caso o utilizador queira utilizar outra função de ativação terá de alterar o código da função ***sig(x)*** para a função correspondente.

Com isto, foi criada a classe Camada Densa, que representa uma camada densa com múltiplos neurónios, é responsável por processar os dados recebidos e ajustar os parâmetros.

**Atributos:**

**de**-Dimensão das entradas da camada (neurónios na camada anterior).

**ds-** Dimensão das saídas da camada (número de neurónios nesta camada).

**phi**-Função de ativação.

**y**-Saída da camada.

**neuronios**: Lista de objetos do tipo Neurónio

**Métodos:**

***\_\_init\_\_(self, de, ds, phi):*** Inicializa a camada recebendo de, ds e phi. Cria uma lista de neurónios usando a classe Neuronio, previamente implementada.

***propagar(self, x):*** Propaga os valores de entrada, x, para todos os neurónios da camada, também calculando as respetivas saídas.

***adaptar(self, delta\_n, y\_n\_menos\_1, alpha, beta):***

**(Parâmetros deste método):**

**delta\_n-**Erro associado a cada neurónio.

**y\_n\_menos\_1-**Saídas da camada anterior (entradas para a camada atual).

**alfa-**Taxa de aprendizagem.

**beta-**Termo de momento.

Este método, chama o método adaptar de neurónio aplicando os respetivos atributos.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Por último foi criado uma classe RedeNeuronal, responsável por criar a rede neuronal multicamada.

**Atributos:**

**forma-** É responsável por definir o numero de neurónios em cada cama, ex: [2, 3, 1] isto significa que a rede terá dois neurónios de entrada, cinco na camada escondida e dois na camada de saída.

**phi-** Função de ativação.

**Métodos:**

***\_\_init\_\_(self, forma, phi):*** Inicializa a rede neuronal, constrói a rede com as camadas com base na lista ***forma***.

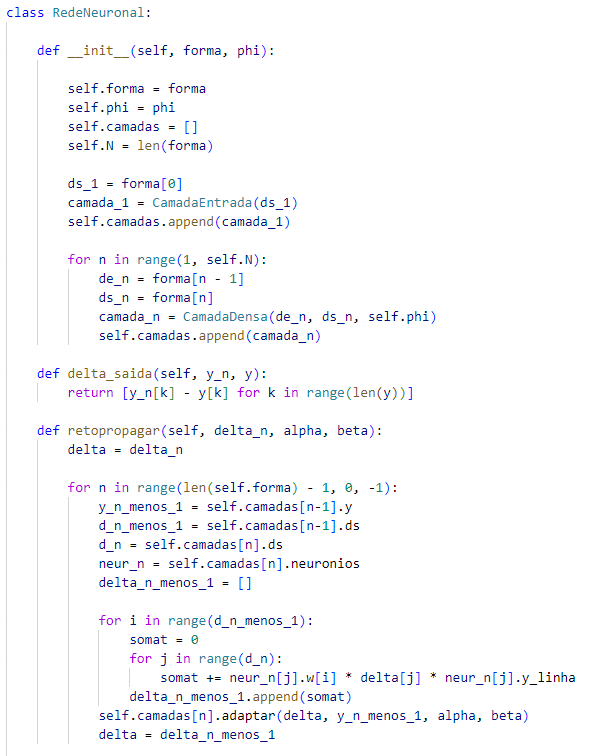
***def delta\_saida(self, y\_n, y):*** Este método calcula o erro entre a diferença da resposta atual da rede, yn, e a resposta alvo, y.

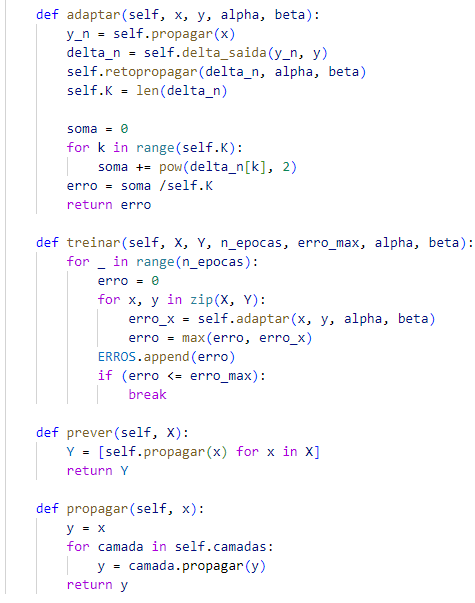
***retopropagar(self, delta\_n, alpha, beta):*** O objetivo deste método é percorrer as camadas de trás para a frente executando o processo de retro propagação visto anteriormente, com os valores taxa de entrada e termo de momento correspondentes.

***adaptar(self, x, y, alpha, beta):*** Este método faz a propagação direta, o calculo do erro de saída, a retroprogacação e por fim o calculo do erro médio quadrático(MSE), este erro é útil para avaliar o desempenho da rede.

***treinar(self, X, Y, n\_epocas, erro\_max, alpha, beta):*** Tem como objetico treinar a rede neuronal, ajustando pesos e pendores, o treino é feito através da repetição do método adaptar, ou seja, através de varias proagacoes diretas e inversas. ***erro\_max*** serve para interromper o treino caso o erro da época seja muito significativo.

***prever(self, X):*** Faz predições usando a rede neuronal já treinada, recebe entradas, ***X***, e retorna a previsão das mesmas.

***def propagar(self, x):*** Semelhante aos métodos das outras classes propaga os valores de entrada para a próxima camada.



Exemplo prático

De modo a verificar o funcionamento da biblioteca, decidiu-se fazer um teste com um dataset com dados fictícios. A ideia foi criar um cenário hipotético onde uma empresa de telecomunicações forneceu um dataset com a informação relativa aos clientes renovarem, ou não, o contrato do serviço conforme alguns atributos.

* Idade: Idade do cliente em anos
* Tempo de Contrato: Tempo de contrato em meses (há quanto tempo o cliente está com a empresa)
* Planos Adicionais: Número de planos adicionais (como internet, TV a cabo, telemóvel, etc.)
* Reclamações: Número de reclamações realizadas pelo cliente no último ano
* Fatura Mensal: Valor da fatura mensal do cliente
* Leave: Target, indica se o cliente cancelou o serviço (Sim (1) ou Não(0))

A ideia é criar um algoritmo capaz de prever se o cliente irá cancelar o serviço, desta forma a empresa pode antecipar-se e sugerir uma melhor oferta.



Tratamento de dados

Com base neste dataset definiu-se como valores de entrada: Idade, Tempo de Contrato, Planos Adicionais, Reclamações e Fatura Mensal. E como saída: Leave

Com os valores definidos foi feita uma divisão do dataset, onde, 80% foi utilizado para treino e os restantes 20% para teste. Para tal utilizou-se o modulo train\_slpit\_test de sklearn.

De seguida, foi feita uma normalização dos dois datasets, mais uma vez recorrendo a sklearn e utilizando StandartScaler.

Arquitetura da Rede

Tendo os dados prontos, foi definida qual a estrutura da rede a usar. O numero de neurónios da camada de entrada foi cinco, respeitando os cinco atributos do dataset e respetivamente para a saída foi utilizado um neurónio. Para a camada escondida foi utilizada a prática vista anteriormente (neurónios escondidos ≈ (número de entradas + número de saídas) / 2), ou seja três neurónios.

Para os restantes valores escolheu-se o seguinte:

n\_epocas = 1000

erro\_max = 0.0001

alpha = 0.2

beta = 0.4

Parte 2 - Processo de decisão sequencial