

Universidade Paulista - UNIP

Luis Henrique Vinhali

PREDIÇÃO DE PICOS DO PROCESSAMENTO

Limeira

2020

Universidade Paulista - UNIP

Luis Henrique Vinhali

PREDIÇÃO DE PICOS DE PROCESSAMENTO

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em ciência da computação sob a orientação do professor Me. Mateus Locci, professor Me. Sergio Eduardo Nunes e PhD Thiago Bianchi

Limeira

2020

Universidade Paulista - UNIP

PREDIÇÃO DE PICOS DE PROCESSAMENTO

Trabalho de conclusão de curso apresentado à banca examinadora da Faculdade UNIP, como requisito parcial à obtenção do Bacharelado em ciência da Computação sob a orientação do professor Me. Mateus Locci, professor Me. Sergio Eduardo Nunes e PhD Thiago Bianchi

Aprovada em 26 de Novembro de 2020.

BANCA EXAMINADORA

DEDICATÓRIA

Dedico este trabalho a todos que contribuíram para minha formação acadêmica e profissional durante meus 4 anos de graduação em ciência da computação. Acima de tudo a minha família que sempre será guia e base do meu amadurecimento pessoal, acadêmico e profissional.

“O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.”.

José de Alencar

RESUMO

Este trabalho foi desenvolvido com objetivo de realizar a predição dos picos de processamento de uma unidade central de processamento (*CPU*), com intuito de solucionar a seguinte questão: *“Quantos picos de processamentos acima de 70% de uso total da CPU serão detectados nos próximos registros?”*. Para solução dessa incógnita foi utilizado uma rede neural com o método de classificação em um modelo sequencial, durante a dissertação desse trabalho será possível analisar a conversão de uma série temporal em um modelo atemporal com o objetivo de simplificar e normalizar os dados para inserção na camada de entrada e saída da rede. Buscando uma maior assertividade na predição, em seguida será utilizado uma engenharia reversa para retornar os dados normalizados entre 0 a 1 em decimais comuns utilizando o mesmo fator de divisão para multiplicação, e por fim gerar um histograma da previsão.

Palavras-Chaves: Rede Neural. Predição. Inteligência Artificial. Lógica

ABSTRACT

This work was developed with the objective of predicting the processing peaks of a central processing unit (CPU), in order to solve the following question: *“How many processing spikes above 70% of total CPU usage will be detected in the next logs?”*. To solve this unknown, a neural network was used with the classification method in a sequential model. During the dissertation of this work, it will be possible to analyze the conversion of a time series into a timeless model in order to simplify and normalize the data for insertion in the layer. incoming and outgoing network. Seeking greater assertiveness in the prediction, then reverse engineering will be used to return the normalized data between 0 to 1 in common decimals using the same division factor for multiplication, and finally generate a histogram of the forecast.

Keywords: Neural Network. Prediction. Artificial Intelligence. Logic

LISTA DE FIGURAS

Figura 1 - Fórmula sigmoide.....	15
Figura 2 - Visão macro de uma arquitetura inteligente.....	16
Figura 3 - Neurônios biológicos.....	17
Figura 4 - Modelo Perceptron.....	17
Figura 5 - Estudos de Minsky e Papert	18
Figura 6 - Fórmula de média de N valores	19
Figura 7 - Fórmula desvio padrão	20
Figura 8 – Exemplo de um modelo sequencial.....	21
Figura 9 - Leitura dos dados CSV (Python).....	22
Figura 10 - Modelagem dos blocos em 50 leituras (Python)	22
Figura 10 - Modelagem dos blocos em 50 leituras (Python)	23
Figura 11 - Cálculo das médias e desvios (Python)	23
Figura 12 – Matriz da camada de deslocamento.....	23
Figura 13 – Histograma	24
Figura 14 e 15 – Mesclagem das matrizes e visualização dos dados em uma tabela	24
Figura 16 - Camada de entrada	25
Figura 17 - Camada de saída.....	25
Figura 18 - Fatores de divisão.....	26
Figura 19 – Função de validação cruzada	26
Figura 20 – Estrutura do modelo de aprendizado	27
Figura 21 – Taxa de assertividade	28
Figura 22 – Sumário do modelo	28
Figura 23 – Função de predição.....	28
Figura 24 – Engenharia reversa	29
Figura 25 – Resultado	29

LISTA DE QUADROS

Quadro 1 – Tabela da matriz da camada de saída	20
--	----

LISTA DE ABREVIATURAS

ALL	Todos os agentes de software
AML	Antilavagem de dinheiro
ARIMA	Média móvel integrada autoregressiva
CPU	Unidade central de processamento
IDE	Ambiente de desenvolvimento integrado
MVP	Produto viável mínimo
VANTs	Veículo aéreo não tripulado

SUMÁRIO

1 INTRODUÇÃO	12
1.1. Objetivo.....	13
1.2. Justificativa	13
1.3. Metodologia	14
2 REDE NEURAL	17
2.1. Conceito e história	17
2.2. Atualmente.....	19
2.3. Explicando o fluxo do algoritmo	19
2.4. Criando um modelo de aprendizado.....	20
3 EXPERIMENTOS	22
3.1. Inicializando experimentos práticos	22
3.2. Modelagem	22
3.3. Observando os dados.....	24
3.4. Normalização	25
3.5. Predição.....	26
4 RESULTADOS	28
5 CONCLUSÃO	31
REFERÊNCIAS BIBLIOGRÁFICAS	32
APÊNDICE	33

1. INTRODUÇÃO

A Previsão de séries temporais tem sido realizada tradicionalmente usando métodos estatísticos como ARIMA ou suavização exponencial. (Hyndman e Athanasopoulos, 2014). Porém recentemente é possível notar o uso de modelos de aprendizado de máquina (Inteligência Artificial).

Toda via o padrão estocástico é algo cujo o estado é indeterminado, causado pela aleatoriedade, algo presente na predição de séries temporais, onde as variáveis podem sofrer uma mutação sem precedentes, isto leva ao entendimento que apenas analisar dados registrados em determinado intervalo de tempo, não é suficiente para resultados precisos em uma predição.

Para atingir um resultado satisfatório é necessário compreender qual o objetivo, e em cima disso arquitetar a manipulação de algumas variáveis para normalização dos dados de treinamento e teste, sendo assim possível aproveitar todo o potencial de uma rede neural, nesse trabalho será possível compreender alguns pontos:

- a) Qual é o problema?
- b) Qual é a lógica?
- c) Qual é o objetivo?

Sendo assim, possível validar se os resultados são aceitáveis considerando a taxa de assertividade e comportamento das antigas ocorrências, e além disso observar a convergência da solução para responder uma questão:

- a) Quantos picos de processamentos acima de 70% serão detectados nos próximos intervalos?

Essa questão responde o objetivo do trabalho, e é baseada no planejamento monotônico de taxa, onde a análise matemática determina que tarefas em tempo real (ou seja, tarefas com prazos específicos em tempo real) são agendáveis quando a utilização é inferior a cerca de 70%. (Daniel Bovet e Marco Cesati, 2014).

1.1. OBJETIVO

O objetivo principal desse trabalho é responder a uma pergunta que é o coração do algoritmo, um histograma que demonstre a previsão não linear de picos de processamento acima de 70% de uso total de seus núcleos, removendo padrões indesejados para que o modelo de aprendizado convirja o mais rápido possível na solução da pergunta sem altas taxas de erro.

Para alcançar o objetivo o algoritmo foi modelado para estar apto a receber dados de séries temporais, e transformá-los em vetores com padrões aceitáveis para sua camada de entrada que será um modelo obrigatório para todas as entradas posteriores, que resultaram em um histograma como resposta à pergunta.

O algoritmo também permitirá sua replicação em infraestruturas de monitoramento como previsão não linear, assegurando a central de gerenciamento de eventos das ferramentas de monitoramento gerar alertas inteligentes.

1.2. JUSTIFICATIVA

A grande maioria dos cientistas hoje não analisam seu objetivo antes de criar um modelo de aprendizado, isso é como correr vendado, você segue uma direção, mas não tem certeza se está indo na direção correta.

Essa abordagem irá explicar e demonstrar como analisar um objetivo é fundamental para assertividade de um modelo de predição. Além disso esse trabalho contribuí para comunidade científica da computação como uma possível abordagem para análise de falhas em ambientes virtualizados, clusterizados ou híbridos, considerando que será eliminado ruídos dentro dos registros coletados que poderiam se tornar fatores de impedimento para resultados assertivos.

Como um exemplo de MVP, podemos analisar um e-commerce que possui uma determinada demanda durante dias normais, onde pode facilmente ser realizada uma predição de vendas com um modelo autorregressivo integrado de

médias móveis conhecido tradicionalmente como ARIMA, uma breve descrição é mencionada na citação a seguir:

“Na análise estatística de séries temporais, modelos autoregressivos de médias móveis (autoregressive-moving-average ou ARMA, na sigla em inglês) oferecem uma descrição parcimoniosa de um processo estocástico fracamente estacionário em termos de dois polinômios, um para a auto-regressão e outro para a média móvel.”
(Percival, Donald, 1993)

No entanto, em temporadas como Black Friday, época de compras natalícias com significativas promoções, essa demanda aumenta drasticamente, ou seja, sua previsão só pode ser feita por algoritmos não-lineares e estatísticos, vulgo inteligência artificial. (Zhang et al., 1998).

1.3. METODOLOGIA

Inicialmente foi feito um estudo bibliográfico para compreender uma rede neural em séries temporais, durante o tempo de pesquisa foi descoberto que utilizar dados de forma aleatória, sem objetivo definido, geram vícios dentro do modelo, após essa descoberta comecei a trabalhar buscando responder uma única pergunta que atendia meu objetivo final que é prever quantas leituras acima de 70% serão detectadas nos próximos intervalos.

Na primeira etapa busquei entender qual seria o melhor modelo a ser utilizado, onde encontrei o modelo sequencial que é uma pilha linear de camadas, uma estrutura de dados na qual elementos de um mesmo tipo de dado estão organizados de maneira sequencial. Não necessariamente, estes elementos estão fisicamente em sequência, mas a ideia é que exista uma ordem lógica entre eles. Logo isso atende o pré-requisito para uma série temporal, onde os dados tem uma ordem cronológica.

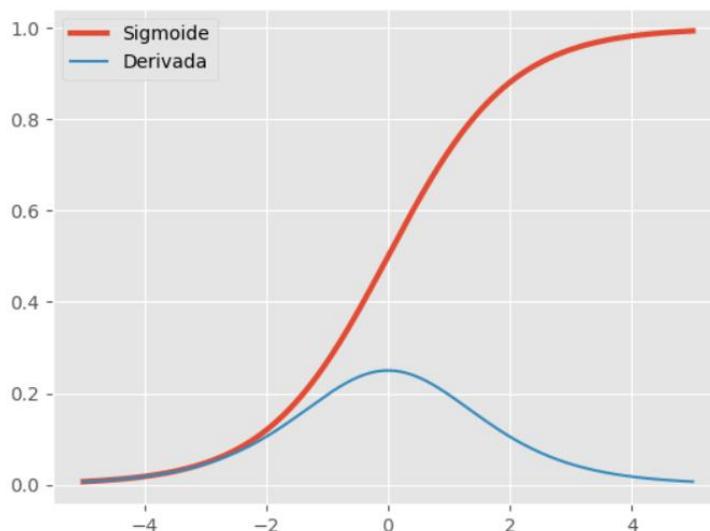
Na segunda etapa eu comecei a manipular algumas variáveis da arquitetura comum de uma rede neural com um modelo classificador. No entanto, algumas arquiteturas não são pilhas lineares. Dado este fato o modelo precisa saber qual formato de entrada deve esperar. Por esse motivo, a primeira camada

foi transformada em médias e desvios padrões e a segunda camada em histogramas.

Na terceira parte foi necessário realizar uma normalização dos dados, a inserção dos vetores sem a normalização pode resultar em previsões com altas taxas de erro. Uma rede neural deve receber os dados em um intervalo de 0 a 1. Logo utilizei o maior fator de cada coluna da matriz para dividir todos os elementos dessa mesma coluna, salvando em uma variável para realizar o processo reverso no final.

Na quarta parte foi necessário escolher qual função de ativação da camada seria utilizada no modelo, ela é responsável por escolher qual neurônio da rede é disparado para previsão não linear. Foi utilizada a função sigmoide que é uma equação diferencial, cuja incógnita aparece na equação com valores entre 0 a 1, abaixo na figura 1 podemos visualizar sua fórmula e função.

Figura 1 - Fórmula sigmoide



$$f(x) = \frac{1}{1 + e^{-x}}$$

Fonte: Matheus facure. Disponível em: <<https://matheusfacure.github.io/2017/07/12/ativ-func/>>. Acesso em 26 out. 2020.

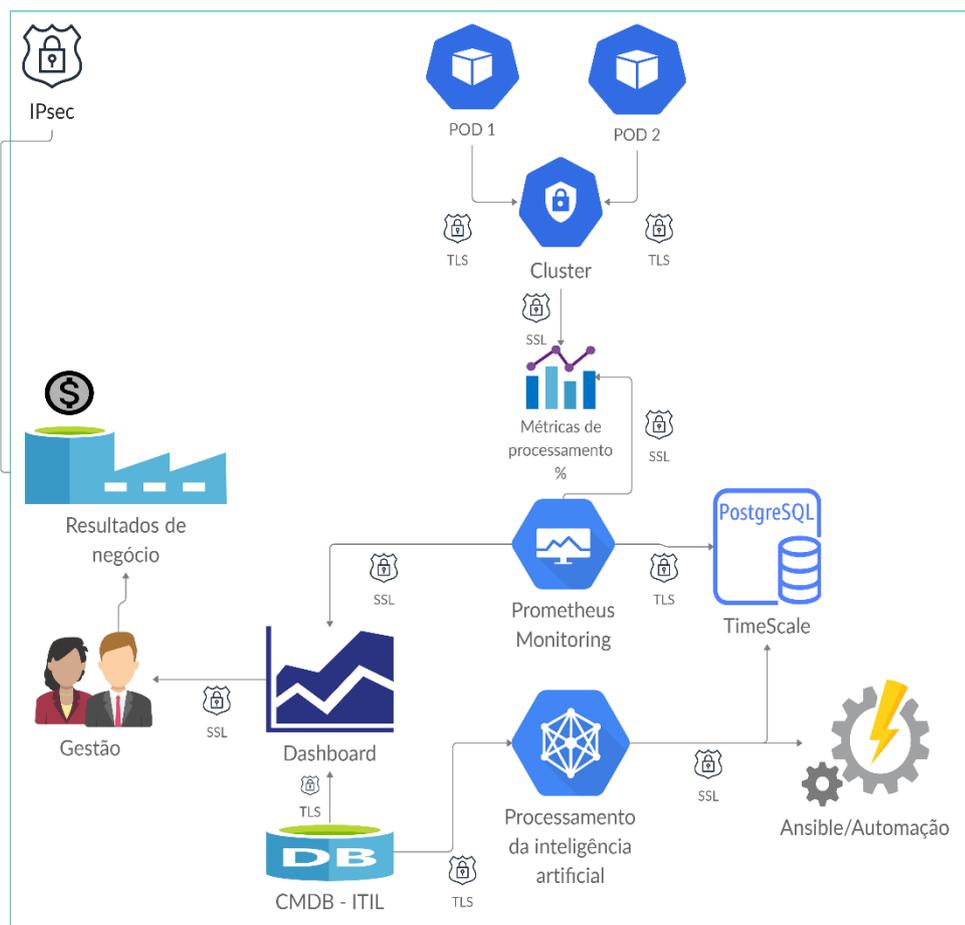
Na quinta parte, os vetores foram enviados para o modelo iniciar seu processo de aprendizado buscando retornar o mais breve possível ao resultado final. Após a conclusão do processo todos os valores recebidos na saída do

modelo estão salvos em um quadro de dados, que irá receber uma multiplicação de cada elemento de sua coluna, pelo maior elemento que foi utilizado na divisão da segunda etapa.

Na última etapa será utilizada a biblioteca Matplotlib da linguagem de programação Python para gerar um histograma dos valores, onde cada barra contém um limite de 20%, exemplo: “0% a 20% ... 80% a 100%”, isso permite o modelo responder a pergunta de quantos picos acima de 70% irão ocorrer nos próximos intervalos.

Abaixo podemos visualizar um fluxograma de como seria a inserção desse algoritmo em uma infraestrutura de monitoramento para que seus alertas se tornassem inteligentes e assertivos, observe a figura 2.

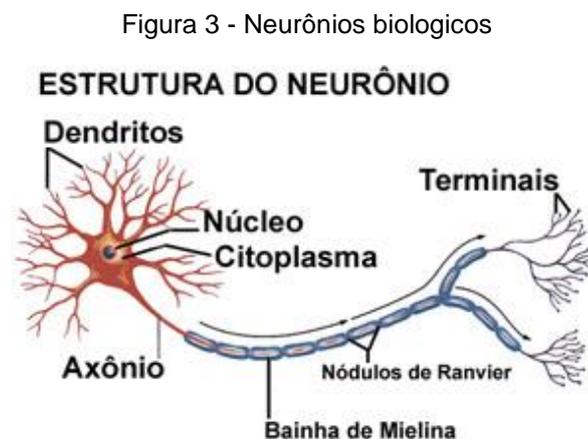
Figura 2 - Visão macro de uma arquitetura inteligente



Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

2. REDE NEURAL

Uma rede neural em computação é responsável pela simulação de neurônios humanos para identificar padrões em conjuntos de dados. Códigos são programados para mimetizar o comportamento de neurônios biológicos e conexões sinápticas que são responsáveis por encaminhar sinais vindos dos dendritos em direção ao axônio. (Wu et al., 2007). Para melhor compreensão observe a figura 3.



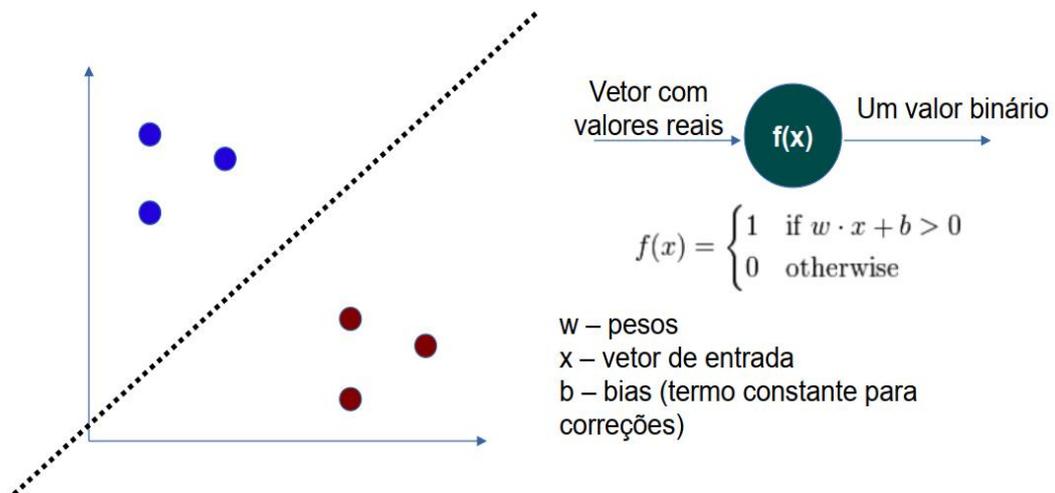
Fonte: [psiqweb.med.br](http://www.psiqweb.med.br). Disponível em: <<http://www.psiqweb.med.br/site/?area=NO/LerNoticia&idNoticia=290>>. Acesso em 26 out. 2020.

2.1. CONCEITO E HISTÓRIA

McCullouch e Pitts (1943) propuseram um modelo computacional baseado em redes neurais biológicas, denominado Threshold Logic. Hebb (década de 1940), psicólogo, propôs a hipótese de aprendizado baseado no mecanismo de plasticidade neural que tem a capacidade do cérebro em se remodelar em função de experiências do sujeito, e reformulação de conexões em função das necessidades e fatores do meio ambiente, que deu origem ao Aprendizado Hebbiano (emprego na Computação a partir de 1948).

Rosenblatt (1958) propôs o modelo Perceptron: um classificador linear e binário, como no exemplo abaixo na figura 4.

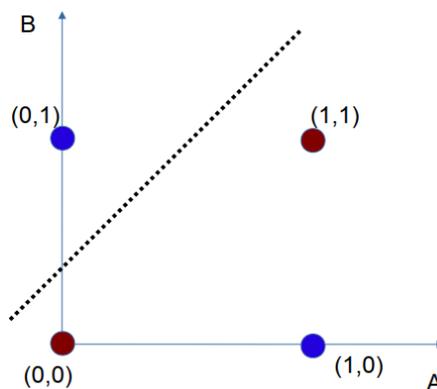
Figura 4 - Modelo Perceptron



Fonte: PhD Thiago Bianchi. Disponível nos repositórios IBM.COM para funcionários>. Criado em em 26 out. 2020.

Após a publicação de Minsky e Papert (1969) a área ficou estagnada, pois descobriram problemas como Ou-Exclusivo não poderiam ser resolvidos utilizando Perceptron e computadores não tinham capacidade suficiente para processar redes neurais artificiais de maior porte. Podemos observar a descoberta na figura 5.

Figura 5 - Estudos de Minsky e Papert



Fonte: PhD Thiago Bianchi. Disponível nos repositórios IBM.COM para funcionários>. Criado em em 26 out. 2020.

Reinício das pesquisas na área após a proposta do algoritmo Backpropagation (Webos 1975). Resolveu o problema do Ou-Exclusivo e área continua avançando até os dias de hoje.

2.2. ATUALMENTE

A inteligência artificial faz parte da nossa vida cotidiana. Acessamos sistemas inteligentes para programar o itinerário com o Waze, pesquisar no Google e receber da Netflix e do Spotify recomendações de filmes e músicas. A Amazon captura nossas preferências no fluxo de dados que coleta a partir das nossas interações com a plataforma, alguns exemplos para aplicações são:

- a) Reconhecimento de Fala;
- b) Classificação de Imagens;
- c) Identificação de portadores de doenças;
- d) AML, ALL, etc.

Já para agentes de software que são agentes que podem operar de forma autônoma em um computador, são eles:

- a) Vídeo games;
- b) Robôs autônomos;
- c) VANTs.

2.3. EXPLICANDO O FLUXO DO ALGORITMO

Como mencionado na metodologia (1.3), para ter um resultado satisfatório em uma predição é necessário primeiro entender o que a máquina deveria aprender. A presença de variações que ocorrem em intervalos regulares é comum em uma série temporal. Logo podemos compreender que esse tipo de comportamento pode influenciar diretamente no resultado da predição. Para solução desse problema será calculado a média a cada 10 leituras nas janelas compostas por 50 leituras, apoiado na fórmula abaixo:

Figura 6 - Fórmula de média de N valores

$$M = \frac{x_1 + x_2 + \dots + x_n}{n}$$

Fonte: A Matemática do Ensino Médio. Disponível em bibliotecas físicas e ebooks>. Consultado em em 26 out. 2020.

Todavia, ainda existem ruídos que podem ser transferidos para as amostras do quadro de dados, afim de analisar o nível desses ruídos, em seguida será calculado o desvio padrão das mesmas 10 leituras, sendo assim possível analisar a uniformidade das amostras, ou seja, o quão homogêneo são os dados, quanto mais próximo de 0 forem as amostras, melhor será o treinamento da rede neural, para isso foi utilizado a seguinte fórmula:

Figura 7 - Fórmula desvio padrão

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

Fonte: Estatística Espacial. Disponível em bibliotecas físicas e ebooks>. Consultado em em 26 out. 2020.

Portanto existe um padrão dentro da camada de entrada, onde dias atípicos não afetaram o julgamento de predição da rede neural. Agora é necessário instruir a rede neural sobre qual objetivo da camada de saída, para isso será calculado os picos de ocorrência agrupados em intervalos de 20 casas decimais, como no exemplo do quadro 1:

Quadro 1 – Tabela da matriz da camada de saída

Intervalo	Contagem	Deslocamento	Agrupamento
1 ^o	0 á 20	10 leituras	[51 ... 69]
2 ^o	20 á 40	10 leituras	[101 ... 119]
3 ^o	40 á 60	10 leituras	[151 ... 179]
4 ^o	60 á 80	10 leituras	[201 ... 219]
5 ^o	80 á 100	10 leituras	[251 ... 279]

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

2.4. CRIANDO UM MODELO DE APRENDIZADO

Para esta abordagem será utilizado o modelo “*Sequential*” da biblioteca Keras com a linguagem de programação Python. (François Chollet, 2012) Esse modelo permite a inserção de camadas de uma rede neural em série, onde a saída da primeira camada serve como entrada da segunda, e assim sucessivamente. Para “instanciar” (criar um objeto alocando um espaço na memória para posteriormente usar os métodos e atributos que ele dispõe), devem ser usadas as seguintes linhas, como exemplo da figura 8:

Figura 8 – Exemplo de um modelo sequencial

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

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

A camada de saída deve ter como dimensão de output (saída) o número de classes a serem conhecidas – por exemplo, para uma classificação binária “Verdadeiro/Falso”, é usada somente 1 dimensão de saída. A compilação serve para configurar o processo de aprendizagem. Deve ser informada a função que define como os pesos da rede neural são atualizados, e por fim a predição pode ser realizada.

3. EXPERIMENTOS

O laboratório utilizado foi com a linguagem de programação python na versão 3.7.9 com a IDE Jupyter Notebook, a biblioteca de rede neural utilizada foi keras na versão 2.0.5.

3.1. INICIALIZANDO EXPERIMENTOS PRÁTICOS

Para treinamento da rede neural será utilizado dados de consumo total de CPU em unidade de porcentagem com intervalo de 1 minuto de coleta, totalizando 10.080 amostras, aproximadamente 1 semana, para isso será utilizado a biblioteca “*pandas*” como intermediário de leitura do arquivo de dados CSV, como no exemplo abaixo da figura 9:

Figura 9 - Leitura dos dados CSV (Python)

```
df = pd.read_csv('../data/cpu-7day.csv')
data = df.iloc[:, 1].values
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

3.2. MODELAGEM

A primeira etapa consiste na padronização dos dados, criando janelas de leituras dos vetores a cada 50 leituras que representam um registro por minuto, será criado uma nova janela. No trecho de código abaixo podemos analisar a implementação em Python da abordagem de janelas mencionada acima:

Figura 10 - Modelagem dos blocos em 50 leituras (Python)

```
blocks_data = []
for i in np.arange(0, int(data.shape[0]-40), 10):
    blocks_data.append(data[i:i+50])
blocks_data = np.array(blocks_data)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

A segunda parte é criar dentro de cada janela um limite de leitura para que haja uma congruência, logo divide-se as 50 leituras em 5 partes ($50/5=10$), como na figura 10.

Figura 10 - Modelagem dos blocos em 50 leituras (Python)

```
parts_data = blocks_data.reshape(-1, 5, 10)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Após a divisão será calculada a média e desvio padrão dentro de cada intervalo de 10 leituras como no exemplo abaixo:

Figura 11 - Cálculo das médias e desvios (Python)

```
mean_parts_data = np.mean(parts_data, axis = -1)
std_parts_data = np.std(parts_data, axis = -1, ddof = 1)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Para ajustar a camada de saída é necessário criar um padrão de deslocamento entre as amostras. Seleciona-se as próximas 20 leituras de cada bloco da camada de entrada, porém sempre deslocando 10 leituras para direita evitando repetições dentro dos blocos. Na imagem abaixo é possível analisar essa implementação na prática manipulando matrizes com Numpy, biblioteca de matrizes do python:

Figura 12 – Matriz da camada de deslocamento

```
next_data = []
for i in np.arange(50, int(data.shape[0]-10), 10):
    next_data.append(data[i:i+20])
next_data = np.array(next_data)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Após os blocos estarem definidos é necessário gerar uma classificação que consiste em uma seleção sequencial:

- a) Grupo: 0% a 20%;
- b) Grupo: 20% a 40%;

- c) Grupo: 40% a 60%;
- d) Grupo: 60% a 80%;
- e) Grupo: 80% a 100%

Exemplo da figura 13 demonstra o trecho de código responsável pelo processo que gera o histograma da camada de saída:

Figura 13 – Histograma

```
count_groups = np.array([np.sum(((0<=next_data) & (next_data<20))*1, axis = -1),
                        np.sum(((20<=next_data) & (next_data<40))*1, axis = -1),
                        np.sum(((40<=next_data) & (next_data<60))*1, axis = -1),
                        np.sum(((60<=next_data) & (next_data<80))*1, axis = -1),
                        np.sum(((80<=next_data) & (next_data<100))*1, axis = -1)]).T
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Após o fim de todo o processo de modelagem dos dados, para simplificar a utilização e visualização, é realizado uma mesclagem entre as matrizes formando um único quadro de dados, como exemplo das figuras 14 e 15.

Figura 14 e 15 – Mesclagem das matrizes e visualização dos dados em uma tabela

```
mean_std = np.append(mean_parts_data.reshape(-1, 1), std_parts_data.reshape(-1, 1), axis = -1).reshape(-1, 10)
pad_count_groups = np.pad(count_groups, (0, mean_std.shape[0]-count_groups.shape[0]))[:, :5]
res_data = np.append(mean_std, pad_count_groups, axis = 1)
```

	mean_1	std_1	mean_2	std_2	mean_3	std_3	mean_4	std_4	mean_5	std_5	0_20	20_40	40_60	60_80	80_100
0	4.26	8.291388	4.86	4.741823	7.83	9.558248	3.14	2.254477	1.99	1.340357	18.0	0.0	2.0	0.0	0.0
1	4.86	4.741823	7.83	9.558248	3.14	2.254477	1.99	1.340357	5.90	12.272100	19.0	0.0	1.0	0.0	0.0
2	7.83	9.558248	3.14	2.254477	1.99	1.340357	5.90	12.272100	5.76	12.876093	19.0	0.0	1.0	0.0	0.0
3	3.14	2.254477	1.99	1.340357	5.90	12.272100	5.76	12.876093	2.97	2.762064	19.0	0.0	1.0	0.0	0.0
4	1.99	1.340357	5.90	12.272100	5.76	12.876093	2.97	2.762064	8.58	12.772262	20.0	0.0	0.0	0.0	0.0
...
999	3.58	3.813951	1.72	2.093270	8.48	14.654449	2.20	2.688659	3.13	3.694155	19.0	1.0	0.0	0.0	0.0
1000	1.72	2.093270	8.48	14.654449	2.20	2.688659	3.13	3.694155	6.27	10.241425	20.0	0.0	0.0	0.0	0.0
1001	8.48	14.654449	2.20	2.688659	3.13	3.694155	6.27	10.241425	2.70	3.612940	19.0	1.0	0.0	0.0	0.0
1002	2.20	2.688659	3.13	3.694155	6.27	10.241425	2.70	3.612940	1.33	1.255256	0.0	0.0	0.0	0.0	0.0
1003	3.13	3.694155	6.27	10.241425	2.70	3.612940	1.33	1.255256	6.92	9.062965	0.0	0.0	0.0	0.0	0.0

1004 rows × 15 columns

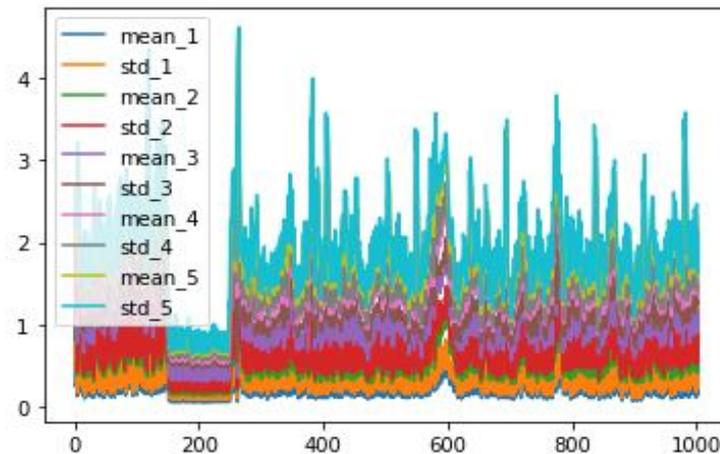
Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

3.3. OBSERVANDO OS DADOS

Para observação dos efeitos gerados nas manipulações de variáveis realizadas até o momento será utilizado o *matplotlib* do Python para uma visualização gráfica, como exemplo da figura 16 e 17.

a) Camada de entrada:

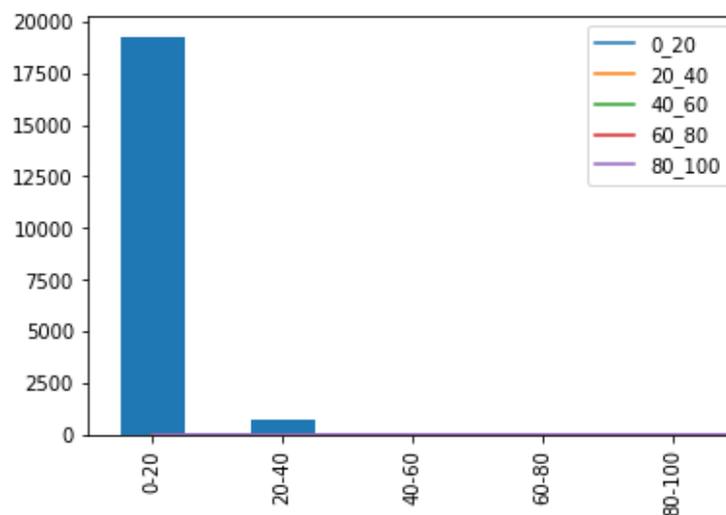
Figura 16 - Camada de entrada



Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

b) Camada de saída:

Figura 17 - Camada de saída



Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

3.4. NORMALIZAÇÃO

É importante que as leituras estejam em um range numérico padronizado, entre 0 a 1, sendo assim será selecionado o maior elemento de cada coluna e usado como fator de divisão para os elementos da respectiva coluna, dentro da rede neural será realizado um balanceamento de peso dos resultados dessa divisão. Abaixo na figura 18 pode-se analisar qual o maior fator de divisão de cada coluna do quadro de dados:

Figura 18 - Fatores de divisão

```
revert = DF.max()
DF.max()
mean_1    16.910000
std_1     31.063297
mean_2    16.910000
std_2     31.063297
mean_3    16.910000
std_3     31.063297
mean_4    16.910000
std_4     31.063297
mean_5    16.910000
std_5     31.063297
0_20     20.000000
20_40     5.000000
40_60     2.000000
60_80     1.000000
80_100    1.000000
dtype: float64

DF.divide(DF.max())
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

3.5. PREDIÇÃO

Em seguida é necessário dividir os dados em conjuntos de treino e teste, para treinar o modelo e o conjunto de teste avaliar o quão bom o modelo generaliza para dados que ele ainda não conhece, utilizando o conceito de validação cruzada dado pela função abaixo da figura 19.

Figura 19 – Função de validação cruzada

```
kfold = KFold(5, False, 1)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Após a validação cruzada ser definida será necessário percorrer dentro de um looping a camada de entrada e saída, utilizando o modelo sequencial para

estrutura da rede neural que será replicado 5 vezes ($K = 5$), como exemplo da figura 20.

Figura 20 – Estrutura do modelo de aprendizado

```

cvcores = []
for train, test in kfold.split(X,Y):

    model = Sequential()
    model.add(Dense(10,
                    kernel_regularizer=l2(0.001),
                    kernel_initializer=VarianceScaling(),
                    activation='sigmoid',
                    name='hidden-input'))
    model.add(Dense(5,
                    kernel_regularizer=l2(0.001),
                    kernel_initializer=VarianceScaling(),
                    activation='sigmoid',
                    name='output'))

    model.compile(loss='mse', optimizer='adam', metrics=['mse'])

    model.fit(X.iloc[train].values, Y.iloc[train].values, epochs=100, batch_size=5, verbose = 0, shuffle=False,
              validation_data=(X.iloc[test].values, Y.iloc[test].values))

    scores = model.evaluate(X.iloc[test].values, Y.iloc[test].values, verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvcores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvcores), np.std(cvcores)))

```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Alguns pontos a serem notados nesse modelo são:

- a) Regularização dos pesos para 0.001
- b) Função *Sigmoid* para ativação
- c) Divisão de leitura de cada camada (5 e 10)
- d) Cálculo de perda utilizando uma função quadrática
- e) Otimização com a função *Adam*
- f) Embaralhamento desativado
- g) Modelo *fit* distribuído com os conjuntos de treino e teste
- h) Cálculo da pontuação do model

4. RESULTADOS

Abaixo é possível analisar o resultado da validação cruzada executada acima, onde aponta uma taxa de erro de 1.17% com desvio padrão de aproximadamente 0.24% para o modelo treinado, um resultado aceitável para o objetivo, como no exemplo da figura 21.

Figura 21 – Taxa de assertividade

```
mse: 1.60%
mse: 1.16%
mse: 0.84%
mse: 1.15%
mse: 1.10%
1.17% (+/- 0.24%)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

A biblioteca keras permite observar um resumo da rede neural chamando a função “*summary()*”, como no exemplo abaixo da figura 22.

Figura 22 – Sumário do modelo

```
model.summary()
Model: "sequential_5"
-----
Layer (type)                Output Shape              Param #
-----
hidden-input (Dense)        multiple                   110
-----
output (Dense)              multiple                   55
-----
Total params: 165
Trainable params: 165
Non-trainable params: 0
-----
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

O modelo está pronto e treinado, para predição a camada de entrada é enviada como amostras para o modelo, como no exemplo abaixo da figura 23.

Figura 23 – Função de predição

```
y = model.predict(X)
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Após o processo ser executado será realizada a reversão dos dados binários para decimais, usando os mesmos fatores de divisão para multiplicação, como no exemplo abaixo da figura 24.

Figura 24 – Engenharia reversa

```
maxOccurrence = revert[10:15]; maxOccurrence
```

```
0_20      20.0
20_40      5.0
40_60      2.0
60_80      1.0
80_100     1.0
dtype: float64
```

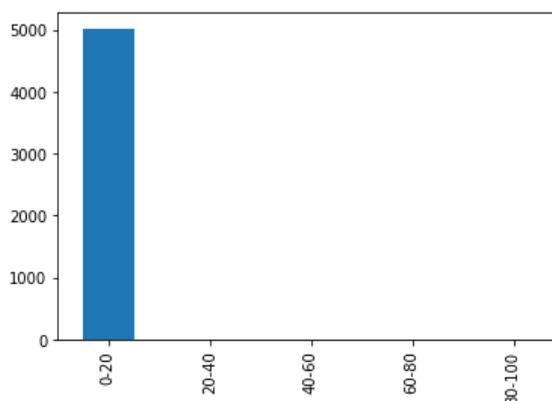
```
speak = result.multiply(maxOccurrence.values);speak
```

Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

Para responder à pergunta inicial: “Quantos picos de processamentos acima de 70% de uso total da CPU serão detectados nos próximos registros?”. Será gerado um histograma seguindo a regra de agrupamento utilizada para criar a camada de saída.

O resultado final é um histograma que responde à pergunta de quantos picos de processamento serão superiores a 70%, para esse modelo é zero, não existem picos superiores a 70, como exemplo final da figura 25 que é uma captura de tela da saída do algoritmo.

Figura 25 – Resultado



Fonte: Elaborado pelo autor Luis Vinhali. Disponível apenas no trabalho>. Criado em em 26 out. 2020.

5. CONCLUSÃO

Neste trabalho, foi apresentado o modelo sequencial keras que permite aos usuários prever uma série temporal usando uma rede neural. A interface do pacote é bastante simples, permitindo aos usuários especificar um modelo sequencial e prever uma série temporal. Além disso, vários gráficos podem ser gerados ilustrando como a previsão foi calculada. Toda via a abordagem principal desde o início do trabalho foi solucionar o problema baseado em uma análise, onde durante a discussão se pode compreender que a manipulação de algumas variáveis facilitam a inserção dos vetores no modelo de aprendizado, e impulsionam o modelo a uma previsão mais assertiva e rápida.

A normalização obrigatória para uma rede neural ajudou a criar a lógica do tratamento dos dados, onde com algumas operações matemáticas simples, uma série temporal com mais de 10.000 registros foi convertida em duas matrizes, uma para camada de entrada e outra para camada de saída, dentro de um intervalo de 0 a 1. O resultado final foi um histograma das leituras futuras (previsão) para análise de anomalias acima de 70%, onde o fator de pequenas oscilações não interfere no modelo de aprendizado graças ao tratamento dos dados.

Por fim este projeto provou que o mais importante antes de trabalhar com inteligência artificial, é entender o que é esperado que seu modelo de aprendizado solucione, além disso, o projeto gera a possibilidade de uma nova visão de solução de problemas de capacidade de infraestrutura utilizando modelos de aprendizado, gerando redução de eventos e incidentes em ferramentas de monitoramento.

REFERÊNCIAS BIBLIOGRÁFICAS

BOVET, Daniel P.; Cesati, Marco, **Understanding the Linux Kernel** – em inglês. (2014)

HAYKIN, Simon. **Redes neurais: princípios e prática**. trad. Paulo Martins Engel. - 2.ed. - Porto Alegre: Bookman. (2001).

KERAS. **Modelo sequencial – Aprendizado profundo**, keras.io, Documentação técnica. (2015-2020).

LLC, Zabbix. **Monitoramento de infraestrutura**, zabbix.com, Documentação técnica. (2001-2020).

LIU, C. L.; Layland, J. **Scheduling algorithms for multiprogramming in a hard-real-time environment** – em inglês. (1973).

PERCIVAL, Donald B.; Walden, Andrew T. **Spectral Analysis for Physical Applications** – em inglês. (1993).

R. R. Andrawis, A. F. Atiya, and H. El-Shishiny. **Forecast combinations of computational intelligence and linear models for the NN5 time series forecasting competition**. *International Journal of Forecasting* – em inglês. (2010).

SHIKIDA, Pery Francisco Assis e MARGARIDO, Mario Antonio. **Uma análise econométrica de sazonalidade dos preços da cana-de-açúcar, estado do Paraná**, (2007-2011).

SUN MICROSYSTEMS, INC. **Introduction to Cloud Computing Architecture**. 1ª edição. ed. White Paper, (2009).

VU, Ky M. **The ARIMA and VARIMA Time Series: Their Modelings, Analyses and Applications**, AuLac Technologies Inc. ISBN 9780978399610, (2007).

X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. **Top 10 algorithms in data mining. Knowl** – em inglês. (2007).

APÊNDICE

APÊNDICE A – Impressão digital do código fonte no software Jupyter Notebook

Conversion of a CPU consumption time series into means and standard deviation to calculate the next peak occurrences using a neural network

Imports

```
In [22]: import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.initializers import VarianceScaling
from keras.regularizers import l2
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import KFold
from sklearn.metrics import cohen_kappa_score
```

Read the data

```
In [23]: df = pd.read_csv('../data/cpu-7day.csv')
data = df.iloc[:, 1].values
```

```
In [24]: pd.DataFrame(data)
```

Out[24]:

	0
0	27.6
1	0.7
2	1.0
3	2.7
4	0.9
...	...
10075	29.8
10076	8.7
10077	13.6
10078	2.1
10079	0.5

10080 rows × 1 columns

Divide the data into blocks of 50 elements, where each new block it moves 10 elements

```
In [25]: blocks_data = []
for i in np.arange(0, int(data.shape[0]-40), 10):
    blocks_data.append(data[i:i+50])
blocks_data = np.array(blocks_data)
```

Then divide the blocks into 5 parts (10x5 = 50)

```
In [26]: parts_data = blocks_data.reshape(-1, 5, 10)
```

Calculate average and deviation of each part

```
In [27]: mean_parts_data = np.mean(parts_data, axis = -1)
std_parts_data = np.std(parts_data, axis = -1, ddof = 1)
```

Take the next 20 elements from each block of 50 elements

```
In [28]: next_data = []
for i in np.arange(50, int(data.shape[0]-10), 10):
    next_data.append(data[i:i+20])
next_data = np.array(next_data)
```

Calculate how many readings there are from 0 to 100, grouping every 20

```
In [29]: count_groups = np.array([np.sum(((0<=next_data) & (next_data<20))*1, axis = -1),
    np.sum(((20<=next_data) & (next_data<40))*1, axis = -1),
    np.sum(((40<=next_data) & (next_data<60))*1, axis = -1),
    np.sum(((60<=next_data) & (next_data<80))*1, axis = -1),
    np.sum(((80<=next_data) & (next_data<100))*1, axis = -1)]).T
```

Collect all and merge in new dataframe

```
In [30]: mean_std = np.append(mean_parts_data.reshape(-1, 1), std_parts_data.reshape(-1, 1), a
axis = -1).reshape(-1, 10)
pad_count_groups = np.pad(count_groups, (0, mean_std.shape[0]-count_groups.shape[0]))
[:, :5]
res_data = np.append(mean_std, pad_count_groups, axis = 1)
```

View data

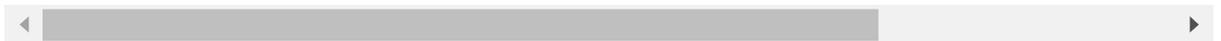
```
In [31]: columns = ['mean_1', 'std_1', 'mean_2', 'std_2', 'mean_3', 'std_3', 'mean_4', 'std_4'
, 'mean_5', 'std_5',
    '0_20', '20_40', '40_60', '60_80', '80_100']
DF = pd.DataFrame(res_data, columns = columns)
```

In [32]: DF

Out[32]:

	mean_1	std_1	mean_2	std_2	mean_3	std_3	mean_4	std_4	mean_5	std_5
0	4.26	8.291388	4.86	4.741823	7.83	9.558248	3.14	2.254477	1.99	1.340357
1	4.86	4.741823	7.83	9.558248	3.14	2.254477	1.99	1.340357	5.90	12.272100
2	7.83	9.558248	3.14	2.254477	1.99	1.340357	5.90	12.272100	5.76	12.876093
3	3.14	2.254477	1.99	1.340357	5.90	12.272100	5.76	12.876093	2.97	2.762064
4	1.99	1.340357	5.90	12.272100	5.76	12.876093	2.97	2.762064	8.58	12.772262
...
999	3.58	3.813951	1.72	2.093270	8.48	14.654449	2.20	2.688659	3.13	3.694155
1000	1.72	2.093270	8.48	14.654449	2.20	2.688659	3.13	3.694155	6.27	10.241425
1001	8.48	14.654449	2.20	2.688659	3.13	3.694155	6.27	10.241425	2.70	3.612940
1002	2.20	2.688659	3.13	3.694155	6.27	10.241425	2.70	3.612940	1.33	1.255256
1003	3.13	3.694155	6.27	10.241425	2.70	3.612940	1.33	1.255256	6.92	9.062965

1004 rows × 15 columns



Normalization, dividing all column elements by the highest occurrence of the column itself (0,1)

In [33]: revert = DF.max()
DF.max()

Out[33]: mean_1 16.910000
std_1 31.063297
mean_2 16.910000
std_2 31.063297
mean_3 16.910000
std_3 31.063297
mean_4 16.910000
std_4 31.063297
mean_5 16.910000
std_5 31.063297
0_20 20.000000
20_40 5.000000
40_60 2.000000
60_80 1.000000
80_100 1.000000
dtype: float64

```
In [34]: DF.divide(DF.max())
```

Out[34]:

	mean_1	std_1	mean_2	std_2	mean_3	std_3	mean_4	std_4	mean_5	std_5
0	0.251922	0.266919	0.287404	0.152650	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149
1	0.287404	0.152650	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068
2	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511
3	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511	0.175636	0.088917
4	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511	0.175636	0.088917	0.507392	0.411169
...
999	0.211709	0.122780	0.101715	0.067387	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923
1000	0.101715	0.067387	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695
1001	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309
1002	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309	0.078652	0.040410
1003	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309	0.078652	0.040410	0.409225	0.291758

1004 rows × 15 columns



```
In [35]: DF = DF.divide(DF.max())
```

Dividing averages/deviation in X and counts in Y

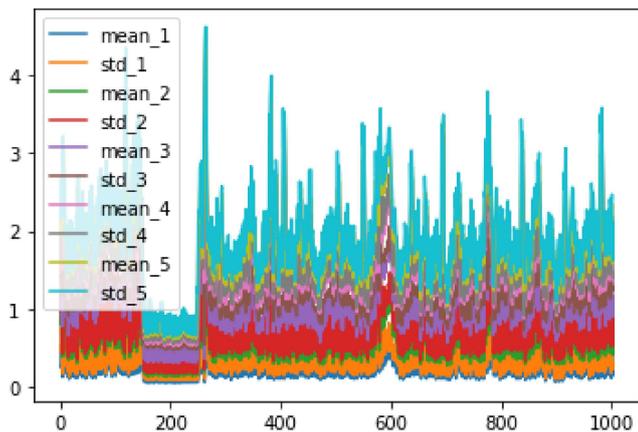
Viewing the neuron input

```
In [36]: X = DF.iloc[:, 0:10]
X.plot(kind='line', stacked=True); X
```

Out[36]:

	mean_1	std_1	mean_2	std_2	mean_3	std_3	mean_4	std_4	mean_5	std_5
0	0.251922	0.266919	0.287404	0.152650	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149
1	0.287404	0.152650	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068
2	0.463040	0.307702	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511
3	0.185689	0.072577	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511	0.175636	0.088917
4	0.117682	0.043149	0.348906	0.395068	0.340627	0.414511	0.175636	0.088917	0.507392	0.411169
...
999	0.211709	0.122780	0.101715	0.067387	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923
1000	0.101715	0.067387	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695
1001	0.501478	0.471761	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309
1002	0.130101	0.086554	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309	0.078652	0.040410
1003	0.185098	0.118923	0.370787	0.329695	0.159669	0.116309	0.078652	0.040410	0.409225	0.291758

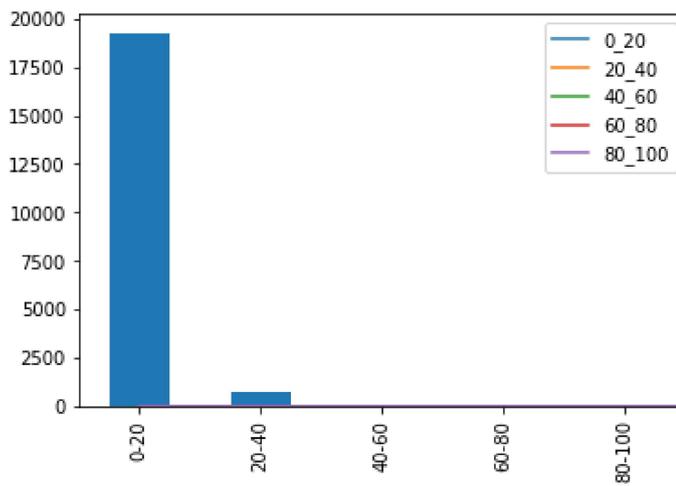
1004 rows × 10 columns



Visualizing the output of the neuron

```
In [37]: Y = DF.iloc[:, 10:15];  
  
Y.plot(kind='line', stacked=True);  
pd.DataFrame(count_groups, columns = ['0-20', '20-40', '40-60', '60-80', '80-100']).sum()  
.plot(kind='bar', stacked=True)
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7f99f5b35d90>



Visualizing the shape of the axes

```
In [38]: X.shape, Y.shape
```

Out[38]: ((1004, 10), (1004, 5))

Preparing cross validation k = 5

```
In [39]: kfold = KFold(5, False, 1)
```

Training:

```
In [40]: cvscores = []
for train, test in kfold.split(X,Y):

    model = Sequential()
    model.add(Dense(10,
                    kernel_regularizer=l2(0.001),
                    kernel_initializer=VarianceScaling(),
                    activation='sigmoid',
                    name='hidden-input'))
    model.add(Dense(5,
                    kernel_regularizer=l2(0.001),
                    kernel_initializer=VarianceScaling(),
                    activation='sigmoid',
                    name='output'))

    model.compile(loss='mse', optimizer='adam', metrics=['mse'])

    model.fit(X.iloc[train].values, Y.iloc[train].values, epochs=100, batch_size=5, verbose = 0, shuffle=False,
              validation_data=(X.iloc[test].values, Y.iloc[test].values))

    scores = model.evaluate(X.iloc[test].values, Y.iloc[test].values, verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
mse: 1.60%
mse: 1.16%
mse: 0.84%
mse: 1.15%
mse: 1.10%
1.17% (+/- 0.24%)
```

Viewing the network structure

```
In [41]: model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
hidden-input (Dense)	multiple	110
output (Dense)	multiple	55
Total params: 165		
Trainable params: 165		
Non-trainable params: 0		

Generating prediction based on X readings

```
In [42]: y = model.predict(X)
```

WARNING:tensorflow:Layer hidden-input is casting an input tensor from dtype float64 to the layer's dtype of float32, which is new behavior in TensorFlow 2. The layer has dtype float32 because it's dtype defaults to floatx.

If you intended to run this layer in float32, you can safely ignore this warning. If in doubt, this warning is likely only an issue if you are porting a TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.keras.backend.set_floatx('float64')`. To change just this layer, pass `dtype='float64'` to the layer constructor. If you are the author of this layer, you can disable autocasting by passing `auto_cast=False` to the base Layer constructor.

```
In [43]: len(X)
```

```
Out[43]: 1004
```

```
In [44]: result = pd.DataFrame(y); result
```

```
Out[44]:
```

	0	1	2	3	4
0	0.962622	0.135029	0.032634	0.012159	0.020513
1	0.962622	0.135020	0.032632	0.012159	0.020512
2	0.962622	0.135025	0.032633	0.012159	0.020512
3	0.962622	0.135030	0.032634	0.012159	0.020513
4	0.962622	0.135023	0.032633	0.012159	0.020512
...
999	0.962622	0.135030	0.032634	0.012159	0.020513
1000	0.962622	0.135019	0.032632	0.012159	0.020512
1001	0.962622	0.135029	0.032634	0.012159	0.020513
1002	0.962622	0.135028	0.032634	0.012159	0.020512
1003	0.962622	0.135020	0.032633	0.012159	0.020512

1004 rows × 5 columns

Reverting binary values to decimals

```
In [45]: maxOccurrence = revert[10:15]; maxOccurrence
```

```
Out[45]: 0_20      20.0  
         20_40      5.0  
         40_60      2.0  
         60_80      1.0  
         80_100     1.0  
         dtype: float64
```

```
In [46]: speak = result.multiply(maxOccurrence.values);speak
```

Out[46]:

	0	1	2	3	4
0	19.252441	0.675144	0.065267	0.012159	0.020513
1	19.252449	0.675102	0.065265	0.012159	0.020512
2	19.252443	0.675125	0.065266	0.012159	0.020512
3	19.252438	0.675151	0.065268	0.012159	0.020513
4	19.252445	0.675116	0.065266	0.012159	0.020512
...
999	19.252438	0.675150	0.065268	0.012159	0.020513
1000	19.252449	0.675095	0.065265	0.012159	0.020512
1001	19.252438	0.675145	0.065267	0.012159	0.020513
1002	19.252441	0.675140	0.065267	0.012159	0.020512
1003	19.252449	0.675102	0.065265	0.012159	0.020512

1004 rows × 5 columns

Grouping data into occurrences for sum

```
In [47]: next_data = np.array(speak)
count_groups = np.array([np.sum(((0<=next_data) & (next_data<20))*1, axis = -1),
                          np.sum(((20<=next_data) & (next_data<40))*1, axis = -1),
                          np.sum(((40<=next_data) & (next_data<60))*1, axis = -1),
                          np.sum(((60<=next_data) & (next_data<80))*1, axis = -1),
                          np.sum(((80<=next_data) & (next_data<100))*1, axis = -1)]).T
```

Forecasting next peaks based on grouping

```
In [48]: count_groups.sum(axis=0)
```

Out[48]: array([5020, 0, 0, 0, 0])

Generating histogram of readings

```
In [49]: columns = ['0-20', '20-40', '40-60', '60-80', '80-100']
peaks = pd.DataFrame(count_groups, columns = columns)
```

```
In [50]: peaks.sum().plot(kind='bar', stacked=True)
```

```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f99b84998d0>
```

