UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

BRYAN ALVES DO PRADO RAIMUNDO

Redes Neurais: Estruturas De Um Mundo Moderno

Campinas
13/07/2022

BRYAN ALVES DO PRADO RAIMUNDO

Redes Neurais: Estruturas De Um Mundo Moderno

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. João Batista Florindo.

Resumo

Com vistas do avanço da tecnologia em áreas como inteligência artificial, modelos classificatórios e sistemas de recomendação, tal projeto visa apurar de forma analítica e crítica técnicas e abordagens clássicas e modernas amplamente utilizadas no mundo contemporâneo no âmbito de aprendizagem de máquina, principalmente no campo do aprendizado profundo.

Abstract

With a view to the advancement of technology in areas such as artificial intelligence, classificatory models and recommender systems, this project aims to analyze, in an analytical and critical way, classic and modern techniques and approaches widely used in the contemporary world in the field of machine learning, mainly in the field of deep learning.

Conteúdo

Lista de Figuras

1 Introdução

Em razão do avanço tecnológico amplo e irrestrito constatado no período das últimas duas décadas, ideias e concepções antes tidas como inviáveis acabaram por se tornar possíveis e amplamente utilizadas em pilares do mundo moderno, tais quais as redes sociais, aplicativos de serviços e entre outros, entre essas ideias, talvez a mais incidente em discussões acerca do presente, e futuro, é a aprendizagem de máquina. A ampla captação e capacidade de armazenamento de dados propiciou inovações e descobertas em um campo que outrora se encontrava estagnado pelas amarras tecnológicas de quando foi proposto, o aprendizado profundo, ou como é comumente conhecido globalmente, *Deep Learning*. Com vistas de tais fatos, compreender o funcionamento de tal campo, suas principais técnicas, abordagens e aplicações, é como de certa forma, entender o mundo moderno em todos os seus anseios.

2 Redes Neurais (FFN) - Um Mundo Conectado

2.1 Conceito Básico

Em geral, podemos definir o conceito de redes neurais como técnicas computacionais (e matemáticas), aplicadas principalmente no âmbito de aprendizado de máquinas, baseadas em modelos estruturais fortemente inspirados no modo de atuação do sistema nervoso de organismos inteligentes que adquirem conhecimento através da experiência. Tal sistema é formado por neurônios, células complexas com papel extremamente importante na definição do comportamento humano, seu aprendizado e sua memória. A seguir, um exemplo de neurônio:

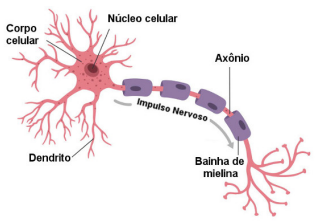

Figura 1: Exemplo de Neurônio

Fonte: https://mundoeducacao.uol.com.br/biologia/neuronios.htm.

Quando tratamos de tal modelo no âmbito matemático, conseguimos definir três grupos (ou camadas) com atuação específica:

- Camada de entrada, onde os dados são oferecidos à rede;
- Camadas intermediárias (mais conhecidas como camadas escondidas/ocultas), responsáveis pelo processamento e extração de características dos dados;
- Camada de saída, responsável por computar e apresentar o resultado final.

De forma resumida, a estrutura geral é dada da seguinte forma:

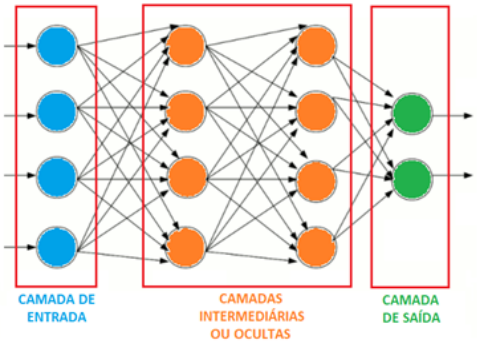

Figura 2: Exemplo Simples de Rede Neural

Fonte: http://www2.decom.ufop.br/imobilis/fundamentos-de-redes-neurais/

Cada camada é ligada a sua camada sequencial através de pesos, sendo que, estes indicam um valor de influência na saída da unidade. De forma simplificada, consideremos uma camada n e uma camada $n+1$: o peso incidiria sobre a saída da camada n, e tal saída atuaria como uma entrada da camada $n+1$. Consideremos então que em cada camada da rede neural teremos um número i de neurônios (que pode ser variável), e em cada neurônio teremos uma soma ponderada z composta pelos pesos anteriormente citados multiplicados por seus respectivos valores de entrada. Tal valor z será utilizado em uma função de ativação a, de forma que para uma função sigmoide, por exemplo, teríamos algo como $a = g(z) = \frac{1}{1+e^{-z}}$. A variedade de funções de ativação é enorme, entre elas, podemos citar a ReLU (*Rectified Linear Unit*), amplamente utilizada na parte de camadas escondidas/ocultas, a função de tangente hiperbólica e a já mencionada sigmoide, muito recorrente em modelos de classificação. A seguir, um exemplo detalhado de uma rede neural básica:

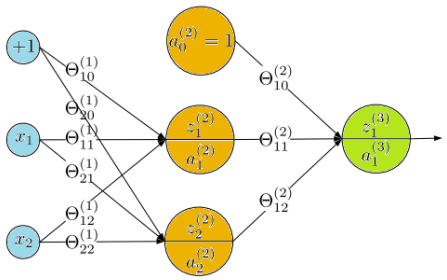

Figura 3: Exemplo Detalhado de Uma Rede Neural

Fonte: https://opencadd.com.br/events/como-se-treina-uma-rede-neural/

2.2 Primórdios:

A história do desenvolvimento e concepção de redes neurais é marcada por momentos de grandes rupturas e evoluções em relação a conceitos, técnicas e tecnologias já estabelecidas.

Se iniciando na década de 1940, mais especificamente no ano de 1943 na universidade de Illinois (EUA), Warren McCulloch, um neurofisiologista, e Walter Pits, um matemático, são os responsáveis pela criação do primeiro modelo computacional para redes neurais baseado em matemática e lógica, publicado em um *paper* (McCulloch and Pitts [1943]). Em seguida, na década de 1950, Frank Rosenblatt, inspirado pela linha de raciocínio apresentada por McCulloch e Pits, cria o algoritmo *Perceptron*, com a função de reconhecimento de padrões baseada em uma rede neural simples de duas camadas. Tal algoritmo foi publicado em um *paper* (Rosenblatt [1958]).

Esses dois pontos são de extrema importância e servem como pontos de inicialização para um amplo estudo e desenvolvimento posterior da área. Uma representação do algoritmo *Perceptron* original:

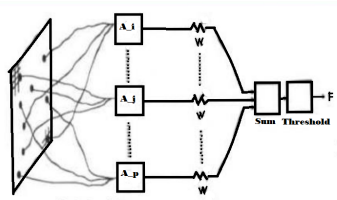

Figura 4: Perceptron

Fonte: https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

2.3 Forward-Propagation e Back-Propagation

Algoritmo essencial para as redes neurais, o *Forward-Propagation* é responsável por percorrer a rede neural desde sua camada de entrada até a camada de saída (como o próprio nome sugere), por meio do uso do valor calculado na função de ativação como valor de entrada para a camada sequente, a partir da primeira camada escondida (no caso da primeira camada, é utilizada a própria matriz X). Seu funcionamento pode ser exemplificado para uma rede de quatro camadas (2 ocultas) com os seguintes passos:

Algorithm 1 Forward-Propagation

$a^{(1)} = X$
$z^{(2)} = \theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)}) + a^{(2)}_{(0)}$
$z^{(3)} = \theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)}) + a^{(3)}_{(0)}$
$z^{(4)} = \theta^{(3)} a^{(3)}$
$a^{(4)} = h_\theta(x) = g(z^{(4)})$

O *Back-Propagation* (D. et al. [1986]), por sua vez, é um algoritmo que percorre a rede neural pelo caminho reverso, ou seja, da camada de saída até a camada de entrada, melhorando progressivamente os pesos utilizados nas camadas da rede durante o percurso, através do cálculo e uso do gradiente. De forma geral, o algoritmo se comporta da seguinte forma:

Algorithm 2 Back-Propagation

$\Delta^l_{ij} := 0$
for $i = 1 : m$ **do**
    $a^{(1)} := X^{(i)}$
    $\delta^{(L)} := a^{(L)} - Y^{(i)}$
    $\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} * a^{(l)} * (1 - a^{(l)})$, para $l = (L-1) : 2$
    $\Delta^l_{ij} := \Delta^l_{ij} + A^l_J \delta^{l+1}_i$
end for
$D^l_{ij} := \frac{1}{m}\Delta^l_{ij} + \lambda\theta^l_{ij}$, se $j \neq 0$
$D^l_{ij} := \frac{1}{m}\Delta^l_{ij}$, se $j = 0$
$\frac{\partial}{\partial\theta^l_{ij}} J(\theta) = D^l_{ij}$

Sua correta implementação é importante para o funcionamento pleno e correto de boa parte dos modelos desenvolvidos com base em redes neurais.

2.4 Função de Custo

Após implementar e testar uma dada rede neural, torna-se importante compreender se a mesma performa e funciona de maneira adequada para o problema a qual sua implementação visa resolver, sendo assim, uma visão de quão fidedigno são seus resultados é necessária, e a partir disso é possível constatar a importância da função de custo em um modelo. De forma geral, a mesma apresenta o quão próximo/igual estão os resultados gerados por uma rede neural em comparação a realidade dos dados. Além disso, a mesma é uma importante métrica acerca de quão rápido uma rede neural é treinada (E se é realmente treinada).

Como a quantidade de aplicações possíveis para redes neurais são amplas e diversas, existem também diferentes funções de custos que melhor se adequam a cada problema abordado, principalmente com o intuito de oferecer uma visão otimizada e correta perante os dados, e seus diferentes tipos, utilizados.

Entre alguns exemplos clássicos amplamente utilizados, podemos citar o erro quadrático médio, com aplicação rotineira em modelos de ajuste de curvas (Regressão linear e não linear), que possuí o seguinte formato:

$$J = \frac{1}{2m}\sum_{i=1}^{m}(\hat{Y}^{(i)} - Y^{(i)})^2$$

Onde $\hat{Y}$ é o vetor com os valores resultantes do modelo, Y é o vetor com os valores reais, m o número de dados no vetor Y (Que por padrão, tem o mesmo tamanho do vetor $\hat{Y}$) e J o custo resultante.

Para modelos que visam obter resultados de classificação, é muito comum a utilização de regressão logística, e para tal, é mais adequada a utilização de uma função de custo denominada *Cross Entropy Loss*, que possuí o seguinte formato:

$$J = -\frac{1}{m}\sum_{i=1}^{m}(Y^{(i)}log(\hat{Y}^{(i)}) + (1 - Y^{(i)})log(1 - \hat{Y}^{(i)}))$$

Sendo as variáveis correlatas as relatadas anteriormente. Tal função se adequa melhor a lógica de 0/1 geral para modelos de classificação, e ressalta a necessidade de se pensar uma função de custo adequada para cada aplicação.

2.5 Algoritmos de Otimização

A otimização é parte central de qualquer modelo baseado em redes neurais, sua correta e adequada implementação pode ser o diferencial entre um modelo que se ajuste de maneira ótima aos dados e um modelo fadado ao fracasso. Tal qual a função de custos, sua variedade também é ampla, e a escolha acerca de qual algoritmo utilizar passa por avaliações em parâmetros como o custo computacional e a quantidade, variedade e tipo dos dados aos quais se busca um ajuste. Com isso, os tipos mais populares de algoritmos com tal intuito de aplicação são baseados na lógica de gradientes, ou seja, a cada iteração busca-se obter a direção para a qual atualizar os valores de pesos θ (Também referenciado em muitos lugares como W, de *weight*), através do gradiente da função $J(\theta)$, de forma que tais coeficientes minimizem o custo final obtido. Importante constatar que o vetor (Ou matriz) θ é inicializado de forma randômica, e seus valores iniciais podem aumentar a velocidade de convergência geral.

Entre os algoritmos baseados em gradientes, o mais popular (E simples de implementar) é o gradiente de descida. Sua lógica consiste em atualizar a cada iteração os pesos θ utilizando como base o gradiente da função $J(\theta)$, sua fórmula geral é a seguinte:

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$$

Onde θ_i é o i-ésimo termo do vetor θ, α é uma variável pré definida que indica o tamanho do passo e $J(\theta)$ é a função de custos. Importante ressaltar que a escolha de um valor adequado de α é essencial para a convergência dos resultados de maneira correta, uma escolha errônea pode resultar em uma oscilação descontrolada ou uma explosão dos custo com o passar das iterações. Um gradiente de descida construído de maneira correta e com parâmetros adequados tende a se comportar da seguinte forma:

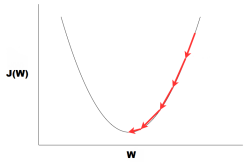

Figura 5: Gradiente de Descida

Fonte: https://makshay.com/neural-network-basics-gradient-descent

O gradiente de descida possui também uma variação denominada gradiente de descida estocástico, e sua utilidade se dá principalmente em casos onde a quantidade de dados (E o tamanho da rede) são muito extensos, ocasionando um custo computacional muito grande a cada iteração do modelo. Sua base de atuação é, ao invés de atualizar todos os pesos presentes na rede, selecionar randômicamente alguns (Ou um grupo, o que leva ao conceito de "*Mini-Batch*") e atualizar somente eles. Sua maior desvantagem se encontra na questão da convergência do custo para um valor mínimo, visto que o mesmo oscila consideravelmente mais em relação ao gradiente de descida padrão.

Outro algoritmo que se utiliza da lógica de gradientes para a otimização dos resultados finais é o gradiente conjugado, nesse caso, o algoritmo é baseado em uma solução numérica para um sistema de equações, dado por uma matriz positiva definida, de forma iterativa. Seu mecanismo de funcionamento tem como base encontrar um passo ótimo a cada iteração, gerando assim uma convergência em um menor número de passos em comparação ao método do gradiente de descida.

Para um sistema da forma $Ax = b$ com A sendo uma matriz positiva definida:

Algorithm 3 Gradiente Conjugado

1: $x^{(1)} = aprox.inicial$
2: $d^{(1)} = r^{(1)}$
3: $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$
4: $\alpha_k = -\frac{r^{(k)} \cdot d^{(k)}}{d^{(k)} \cdot Ad^{(k)}}$
5: $d^{(k+1)} = -r^{(k+1)} + \beta_k d^{(k)}$
6: $\beta^{(k)} = \frac{r^{(k+1)} \cdot Ad^{(k)}}{d^{(k)} \cdot Ad^{(k)}}$
7: $k = 1, 2, 3...\ e\ r^{(k)} = Ax^{(k)} - b$

Apesar de uma maior eficiência em relação ao gradiente de descida quando ambos se iniciam nos mesmos pontos, o gradiente conjugado possuí um custo computacional maior, sendo assim, seu tempo médio de processamento é superior ao do gradiente de descida (Principalmente pelos tipos de cálculos que o mesmo executa). Ou seja, a escolha de qual algoritmo é mais adequado para uma dada aplicação passa por uma ampla análise de quais pontos principais se deseja priorizar, e principalmente, passa pela consideração de qual *hardware* se tem acesso para a execução do modelo. Como já citado anteriormente, não existe modelo perfeito funcional para qualquer caso de aplicação, visto que, o desenvolvimento de redes neurais é, acima de tudo, uma análise criteriosa caso a caso.

Um exemplo ilustrado de como ambos os algoritmos averiguados se comportam em comparação um ao outro:

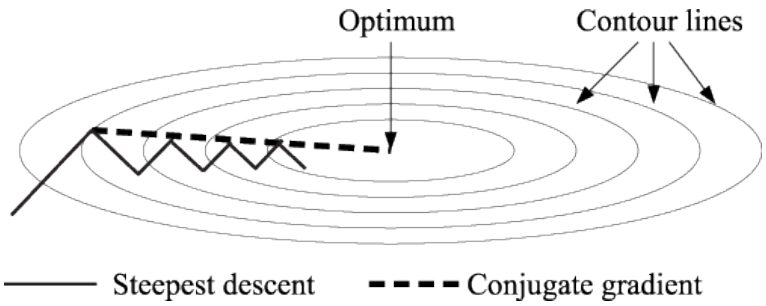

Figura 6: Gradiente Conjugado X Gradiente de Descida

Fonte: http://inf.ufes.br/mn1/metodo-dos-gradientes-e-gradientes-congugados.pdf

Entre as abordagens mais modernas para algoritmos de otimização, destaca-se o ADAM (*Adaptive Moment Estimation*); baseado no princípio do gradiente de descida estocástico, surgiu em 2015 (Kingma and Ba. [2015]) e é uma evolução e junção de dois modelos/métodos também baseados em gradiente de descida estocástico e contemporâneos do ADAM, o AdaGrad (*Adaptive Gradient Algorithm*), que tem como base a adaptação do coeficiente de aprendizado α de cada parâmetro θ a cada iteração de forma que o modelo possa convergir corretamente e em uma velocidade adequada, e o RMSprop (*Root Mean Square Propagation*), que se utiliza de uma média móvel do quadrado dos gradientes para normalizar um dado gradiente, evitando dessa forma problemas como a explosão do custo por conta de passos maiores do que o adequado e o fenômeno do "*Vanishing Gradient*", que consiste na mudança ínfima do parâmetro θ a cada iteração por conta da diminuição brusca do valor do gradiente. O AdaGrad foi apresentado em 2011 (Duchi et al. [2011]), enquanto o RMSprop surgiu em um curso ministrado na plataforma *Coursera* (Hinton et al. [2014]), e até o momento, não possui uma publicação em formato de paper.

O ADAM também se utiliza do conceito de "*Momentum*", técnica que consiste em utilizar, além do gradiente da iteração do momento, o acumulado de gradientes anteriores para guiar os próximos passos do modelo. Tal incremento permite um alcance mais rápido do valor ótimo buscado (Facilita a convergência), visto que a oscilação do custo a cada iteração é reduzida.

Como citado anteriormente, é visível que o ADAM é a evolução esperada de conceitos previamente estabelecidos, e de forma geral, é um modelo extremamente exitoso.

Dada sua alta eficiência em convergir para um resultado ótimo em poucas iterações, baixo custo computacional, adaptabilidade a diferentes problemas e aplicações, fácil escolha de parâmetros e, principalmente, sua simplicidade na implementação, o algoritmo ADAM se tornou extremamente popular e muito utilizado por entusiastas de redes neurais/*Deep Learning*.

Os passos do algoritmo ADAM podem ser visualizados da seguinte forma:

Algorithm 4 ADAM (Adaptive Moment Estimation)

$m_0 = 0$
$v_0 = 0$
$i = 0$
while θ_i not converged **do**
    $i := i + 1$
    $g_i := \nabla_\theta f_i(\theta_{i-1})$
    $m_i := \beta_1 m_{i-1} + (1 - \beta_1) g_i$
    $v_i := \beta_2 v_{i-1} + (1 - \beta_2) g_i^2$
    $\hat{m}_i := \frac{m_i}{1-\beta_1^i}$
    $\hat{v}_i := \frac{v_i}{1-\beta_2^i}$
    $\theta_i := \theta_{i-1} - \alpha \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}$
end while
return θ_i

Sendo m o vetor referente ao primeiro momento e v o vetor referente ao segundo momento, α o coeficiente de aprendizado, β_1 o coeficiente exponencial de decaimento estimado para o primeiro momento, β_2 o coeficiente exponencial de decaimento estimado para o segundo momento e ϵ um valor fixo (E extremamente baixo, em torno de 10^{-8}) com função de evitar divisões por zero durante as iterações.

Para alguns tipos de problemas (Principalmente Regressão Linear e Não Linear) é possível obtermos uma solução de forma direta, sem a necessidade de iterações recorrentes, através da equação normal. A fórmula, com X sendo a matriz de dados de entrada e Y o vetor de saída, é a seguinte:

$$\theta = (X^T X)^{-1} X^T Y$$

Como é possível constatar pelos tipos de cálculos realizados, o custo computacional é consideravelmente alto, o que inviabiliza sua aplicação para problemas grandes.

2.6 Hiperparâmetros

De forma geral, os hiperparâmetros funcionam como controles de um modelo, ditando como o mesmo se comporta perante uma dada aplicação, permitindo assim, o pleno funcionamento do mesmo. Os hiperparâmetros, assim como os próprios modelos, não são variáveis com valores fixos, ou seja, não existem valores perfeitos que funcionam para qualquer tipo de aplicação, e por conta dessa característica de "mutabilidade", se faz necessário com que tais variáveis passem por um processo de "*tunning*", ou seja, tenham seus melhores valores selecionados a partir de testes que avaliem o comportamento do modelo com suas inúmeras variações.

Para a realização dos testes, é adequada uma escolha randômica de possíveis valores, e a partir dos resultados iniciais, restringir os testes para faixas de valores que obtiveram um melhor resultado. Como forma de organização do modelo para os testes, é possível se adotar uma das duas estratégias mais populares, a "Panda", que consiste em focar em apenas um modelo por vez e analisar o seu comportamento com o passar das iterações, e a "Caviar", que se baseia no treino de vários modelos ao mesmo tempo, possibilitando assim uma seleção criteriosa a partir de inúmeros exemplares; o principal fator que dita qual das duas estratégias é mais interessante de se adotar é a capacidade computacional, se a mesma for ampla, a estratégia "Caviar" é muitíssimo interessante.

Os hiperparâmetros variam consideravelmente dependendo dos algoritmos e técnicas adotadas para uma dada aplicação, porém, para modelos baseados em redes neurais temos alguns hiperparâmetros padrões, como o coeficiente α de aprendizado, e também algumas escolhas estruturais, como o número de camadas e o número de neurônios em cada camada. Para modelos que se utilizam do algoritmo de otimização ADAM por exemplo, além dos já mencionados coefiente de aprendizado e variáveis estruturais, temos também outros hiperparâmetros, como o β_1, β_2 e ϵ.

Existem hiperparâmetros mais importantes para um bom funcionamento do modelo (Alguns são fixos e outros variam para cada modelo e também em relação a aplicação que se pretende fazer), como o coeficiente de aprendizado α e as métricas estruturais da rede neural em si por exemplo, assim, se a capacidade computacional disponível para o refino de tais parâmetros não for alta, é adequado priorizar os hiperparâmetros mais importantes para o processo de aprimoramento.

2.7 Problemas e Soluções

A implementação, em geral, de um modelo baseado em redes neurais é complexa e detalhista, dessa forma, alguns problemas podem surgir, e saber como identificar e tratar os mesmos é parte fundamental para a garantia de uma performance plena e duradoura.

Dois dos problemas mais recorrentes em implementações de redes neurais são relacionados ao quão bem um dado modelo se ajusta aos dados de treinamento, eles são conhecidos como "*Underfitting*" e "*Overfitting*".

O *Underfitting* é relacionado a má adequação do modelo para os dados oferecidos como entrada, ou seja, a predição não teve um resultado satisfatório; sua principal forma de detecção se dá através da visualização do erro final obtido nos grupos de teste e treino, mas também pode ser observado através de uma visualização gráfica da curva obtida em relação aos pontos dos dados de entrada (Não recomendado e em problemas maiores é completamente inviável). Alguns meios de abordagem para correção de tal problema consistem em verificar se as métricas dos hiperparâmetros (Principalmente o coeficiente de aprendizado α) estão adequadas, se o número de iterações realizadas é razoável, verificar se os dados de entrada e sua divisão em grupos de teste e treinamento estão corretas, sem problemas de desbalanceamento e baixas quantidades, e principalmente, averiguar se o modelo utilizado é o mais adequado para o problema tratado.

O *Overfitting* é relacionada a adequação exacerbada do modelo para os dados do grupo de treinamento oferecidos como entrada, ou seja, temos um ajuste quase perfeito para os dados de treino, mas uma péssima generalização para os dados de teste. A principal forma de identificação desse fenômeno é através da visualização do erro final no grupo de testes. Para tal problema, verificar a presença de valores extremos em relação ao todo no grupo de treinamento é pertinente, além da averiguação do número de iterações (Reduzir as mesmas pode evitar tal fenômeno) e da complexidade do modelo para o problema tratado. Outra estratégia válida é a adoção de regularização, em seu formato mais famoso, a regularização L2, a mesma força com que os pesos sejam pequenos, mas não nulos, fazendo com que todos os atributos do modelo contribuam, mesmo que de maneira pequena, ocasionando assim uma suavização da curva de predição, reduzindo mudanças bruscas de tendência da mesma, e por consequência, diminuí a probabilidade

de se ocorrer um *Overfitting*. Para o caso da regressão logística por exemplo, a função de custos é dada da seguinte forma:

$$J = -\frac{1}{m}\sum_{i=1}^{m}(Y^{(i)}log(\hat{Y}^{(i)}) + (1 - Y^{(i)})log(1 - \hat{Y}^{(i)})) + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

E por consequência, a função para atualização dos pesos θ é a seguinte:

$$\theta_0 = \theta_0 - \alpha(\frac{1}{m}\sum_{i=1}^{m}(\hat{Y}^{(i)} - Y^{(i)})x_0^{(i)})$$

$$\theta_j = \theta_j - \alpha(\frac{1}{m}\sum_{i=1}^{m}(\hat{Y}^{(i)} - Y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j)$$

Onde λ é o fator regularizante, $\hat{Y}$ é o vetor de resultados finais do modelo, α é o coeficiente de aprendizado e m é o tamanho do vetor θ. Importante ressaltar que o parâmetro λ se trata de um hiperparâmetro, sendo assim, testes para uma escolha adequada do mesmo devem e precisam ser realizados.

Uma técnica recente de regularização que ganhou destaque dado seus resultados positivos é o "*Dropout*", com origem em um paper de 2012 (Hinton et al. [2012]), sua lógica consiste em "desligar" (Tornar nulo) neurônios das camadas ocultas de forma randômica a cada iteração do modelo, fazendo assim com que os neurônios ativos restantes tenham de adquirir uma capacidade maior de generalização, não dependendo tanto de seus neurônios vizinhos. De forma geral, é como se múltiplas redes estivessem sendo treinadas randomicamente ao mesmo tempo, e o resultado final obtido é tomado a partir da média dos resultados de cada rede.

Um exemplo gráfico de cada fenômeno pode ser visualizado a seguir:

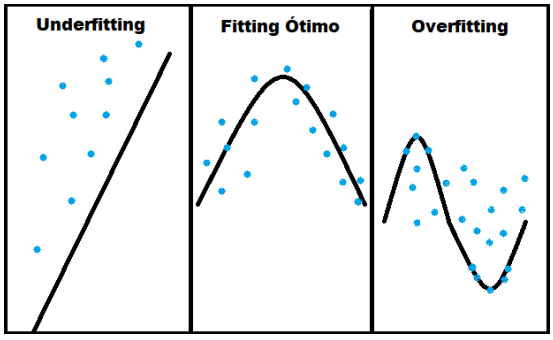

Figura 7: Exemplos de Ajustes

Os gradientes também são entes que, por alguma defasagem na implementação do modelo, podem apresentar problemas. Dois dos mais recorrentes problemas são o "*Vanishing Gradients*" e o "*Exploding Gradients*".

Para o caso do *Vanishing Gradients*, o problema ocorre quando o valor dos gradientes calculados através do *Back-Propagation* são extremamente pequenos, ocasionando uma situação de não atualização adequada dos pesos, dessa forma, ao longo do tempo a rede neural deixa de ser aprimorada, se mantendo estagnada em uma dada faixa de custo e vetor θ. A principal maneira de identificar tal fenômeno é a partir da visualização dos custos ao passar das iterações, e possíveis abordagens para tal problema consistem em avaliar a quantidade de camadas do modelo (Modelos com muitas camadas desnecessárias podem aumentar a chance de tal fenômeno ocorrer), avaliar se a implementação do *Back-Propagation* foi feita de forma correta e, principalmente, avaliar se as funções de ativação utilizadas nas camadas ocultas são adequadas para o modelo.

Como forma de avaliar se o *Back-Propagation* foi corretamente implementado, pode-se recorrer a um método denominado "*Gradient Checking*", que consiste em calcular o gradiente aproximado de um dado valor de θ via a definição de derivada, ou seja:

$$\lim_{\epsilon \to 0} = \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

Para um dado vetor θ, a derivada para cada componente será calculada da seguinte forma:

$$\hat{d\theta_i} = \frac{J(\theta_1, ..., \theta_i + \epsilon) - J(\theta_1, ..., \theta_i - \epsilon)}{2\epsilon}$$

Sendo J a função de custo e ϵ um valor infinitesimal (Algo em torno de 10^{-7}). A partir do ponto em que computamos todas as derivadas, nos utilizamos da seguinte fórmula para verificarmos se os valores gerados através do *Back-Propagation* são adequados:

$$\frac{||\hat{d\theta} - d\theta||_2}{||\hat{d\theta}||_2 + ||d\theta||_2}$$

Onde $\hat{d\theta}$ é o gradiente calculado através da definição formal e $d\theta$ o gradiente calculado através do *Back-Propagation*. Com o resultado desta fórmula, conseguimos averiguar se possivelmente existe algum problema com a implementação do *Back-Propagation*

(Valores inferiores a 10^{-7} indicam que a implementação está correta). É adequado desativar o *Gradient Checking* após uma iteração, dado que seu custo computacional é alto.

No caso dos *Exploding Gradients*, o problema é fruto do crescimento (Ou lento decrescimento) altíssimo dos gradientes com o passar das iterações, dessa forma, o modelo não consegue convergir para um resultado ótimo e todo o processo se torna falho. Tal qual o *Vanishing Gradients*, a maneira mais fácil de identificar tal problemática é através da visualização dos custos com o passar das iterações, e uma possível abordagem para correção é a averiguação da inicialização dos custos; se os mesmos forem inicializados de maneira errônea, com valores muito altos, a chance de falha é maior. Em 2010 foi proposto através de um *paper* (Glorot and Bengio [2010]) um modelo de inicialização de pesos que ficou conhecido como "*Xavier Initialization*", primeiro nome de seu principal autor propositor:

- Uma distribuição normal com média 0 e $\sigma^2 = \frac{1}{z}$
- Uma distribuição uniforme entre $-a$ e a com $a = \sqrt{\frac{3}{z}}$

Onde z é dado pela média simples entre o número de neurônios de entrada para uma dada camada e o número de neurônios em si desta camada. Tal lógica de inicialização se provou extremamente útil e performática, sendo amplamente utilizada.

Para problemas relacionados ao tempo requerido pelo modelo para um treinamento adequado, tem-se como abordagens úteis a adequação do coeficiente de aprendizado α para valores que permitam um aprendizado mais ágil, mas sem ocasionar uma oscilação muito brusca no custo, e a normalização dos valores de entrada, e se possível, dos vetores de ativação presentes nas camadas ocultas, técnica conhecida como "*Batch Normalization*" (Ioffe and Szegedy [2015]) que permite ao modelo uma maior agilidade e estabilidade para o processo de treinamento. Para o grupo de treino, o algoritmo da *Batch Normalization* é o seguinte:

Algorithm 5 Batch Normalization

1: $\mu = \frac{1}{n}\sum_{i=1}^{n} Z^{(i)}$
2: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(Z^{(i)} - \mu)^2$
3: $Z_{norm}^{(i)} = \frac{Z^{(i)}-\mu}{\sqrt{\sigma^2-\epsilon}}$
4: $\hat{Z} = \gamma Z_{norm}^{(i)} + \beta$

Sendo Z o vetor de ativação de uma dada camada oculta, γ e β, que são hiperparâmetros que devem ser ajustados para que o modelo performe adequadamente, são responsáveis pelo ajuste do desvio padrão e do "*bias*" respectivamente. Todo μ e σ^2 de uma dada camada deve ser armazenado, visto que, para o grupo de teste tais valores equivalem a média de todos os μ e σ^2 calculados para cada camada, dessa forma, para o grupo de teste só são realizadas as etapas 3 e 4 do algoritmo. O fator de normalização gerado pelo algoritmo tem como consequência uma rede neural menos sensível a más inicializações de pesos, e permite também a utilização de valores superiores ao padrão do coeficiente de aprendizado α, gerando assim, um treinamento mais rápido do modelo sem perda de estabilidade. Além dos efeitos previamente citados, temos também um efeito secundário na regularização (Goodfellow et al. [2016]), prevenindo um pouco a possibilidade de *Overfitting* (Caso se utilize *Batch Normalization*, é recomendada a não utilização da técnica de *Dropout*, visto que a mesma prejudica o desempenho da normalização).

3 Redes Neurais (RNN) - Um Mundo Recorrente

3.1 Conceito Geral

Ao tratarmos de redes neurais recorrentes, estamos tratando inerentemente da capacidade de fazer com que uma dada informação (Ou conhecimento) persista, ou seja, de um fenômeno que se assemelha a memória. Análises de textos, áudios, fenômenos climáticos e entre outros inúmeros eventos, são todos dependentes de uma compreensão geral do ocorrido até um dado momento para que consigamos compreender o todo, propriedade que as redes neurais do tipo "*Feed-Forward*" não conseguem ou tem extrema dificuldade em modelar, para tal, surgem as redes neurais recorrentes, que através de sua arquitetura com "*loopings*", permite com que informações persistam e influenciem em momentos posteriores de análise. Com sua primeira menção ocorrendo em um *paper* de 1982 (Hopfield [1982]), tal estrutura de rede neural adquiriu imensa relevância em tempos contemporâneos por consequência da quantidade imensa de dados gerados, pelo aumento substancial do poder computacional, e principalmente, pela alta demanda de análises e modelagens que possuem um fator temporal intrínseco, como é o caso de modelos climáticos, e até mesmo, relacionados a surtos epidêmicos de doenças.

Quando comparamos as duas estruturas de redes neurais, FNN e RNN, temos o exemplo a seguir:

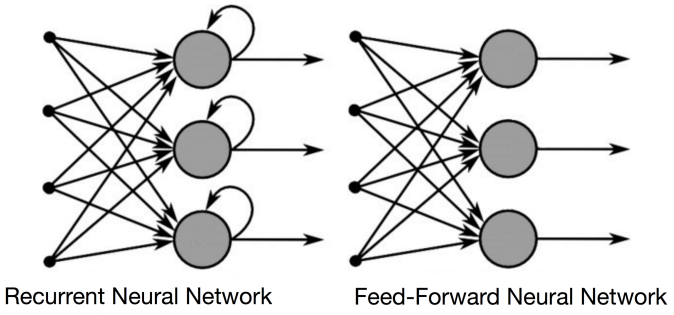

Figura 8: Comparação FNN e RNN

Fonte: https://github.com/Anna868/Arabic-Handwritten-OCR

Como é possível observar, nas RNN possuímos estruturas de *looping* que não ocorrem nas FNN; são exatamente tais estruturas que possibilitam a persistência de conhecimento/informação nas RNN. Os *loopings* funcionam como uma espécie de "*feedback*" para o estado anterior da camada de ativação, sendo assim, a saída é imediatamente utilizada como entrada a cada iteração.

Uma visão matemática desse processo de *looping* seria a seguinte (Schmidt [2019]):

$$g(t) = f(g(t-1), x(t))$$

Ou seja, o estado atual de um dado neurônio é definido pelo estado anterior do mesmo e pelos dados de entrada atuais, e isso se sucede recursivamente, estabelecendo uma relação de dependência com o passar do tempo.

Da mesma forma, ao aplicarmos uma função de ativação qualquer, teremos a seguinte configuração para um dado estado $g(t)$:

$$g(t) = h(Wx(t) + W_{-1}g(t-1) + b)$$

Sendo W o peso no neurônio de entrada, W_{-1} o peso no neurônio recorrente e b uma constante (*Bias*). Dessa forma, torna-se ainda mais evidente o fator de influência de estados anteriores na composição do estado atual de um dado neurônio.

3.2 Tipos De Redes RNN

Dado seu comportamento ajustável, com diferentes possibilidades de ligações e relações entre neurônios e camadas, as redes neurais do tipo RNN possibilitam configurações de fluxo da informação/aprendizado que são inviáveis de se adotar nas redes neurais do tipo FNN. Entre os tipos de organização possíveis para uma rede RNN, podemos destacar a "um-para-um", formato padrão da rede neural FNN onde para uma dada entrada x temos apenas uma saída y; "um-para-vários", onde temos uma entrada x e múltiplas saídas y, estrutura muito utilizada em modelos de geração de texto, música e entre outros; "vários-para-um", onde diversas entradas x corroboram para uma única saída y, muito utilizada no processo de classificação de sentimento; e por final, a "vários-para-vários", amplamente utilizada em processos de tradução de textos (Karpathy [2015]).

De forma visual, os tipos de redes RNN são os seguintes:

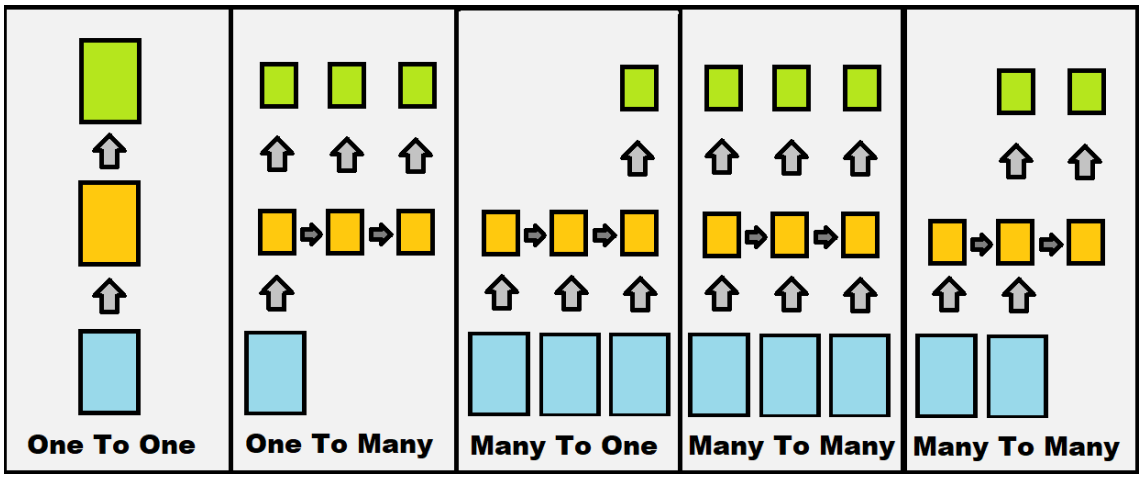

Figura 9: Tipos de RNN

Sendo assim, podemos estimar o custo iteração a iteração para um dado modelo através da seguinte equação:

$$J(\hat{Y}, Y) = \sum_{i=1}^{n} J(\hat{Y}^{(i)}, Y^{(i)})$$

Sendo $\hat{Y}^{(i)}$ o valor predito para uma dada saída, $Y^{(i)}$ o valor real de uma dada saída e J uma função de custo adequada para um dado problema. Como é fácil de observar, o custo total é dado pela soma do custo final de cada saída do modelo.

3.3 Long Short Term Memory (LSTM)

Entre as inúmeras propostas de abordagem para RNN, podemos destacar a LSTM, referenciada primeiramente em um *paper* (Hochreiter and Schmidhuber [1997]), tal proposta de estrutura de RNN surge como uma solução para o problema de "*Vanishing Gradients*" observado em algumas aplicações de modelos, mas principalmente, surge como uma maneira de estimar iteração a iteração quais informações são relevantes ou não para um dado estado do neurônio, permitindo assim, um armazenamento prolongado e relevante das principais informações necessárias para um dado processo ou decisão, implicando na minimização da relevância de informações com baixa influência ao longo do passar de estados.

Para a realização de tal análise a cada estado e iteração, a abordagem LSTM se faz uso de "portões", espécies de funis de avaliação para a manutenção e preparação (Ou não) de dadas informações (Lu and Salem [2017]). Ao todo, são 3 portões, sendo o primeiro o "*Forget Gate*", responsável por indicar quais informações de estados anteriores de um dado neurônio devem ser mantidas ou descartadas, sendo assim, tal processo pode ser representado pelas seguintes fórmulas:

$$F_t = \sigma(Wx(t) + W_{-1}g(t-1))$$

$$B(t-1).F_t = \ 0 \ ou \ 1$$

Ou seja, aplica-se uma função sigmoide para um dado estado do neurônio, e multiplica-se o estado anterior com o resultado de tal função, dessa forma, caso o resultado seja nulo, a informação é descartada, do contrário, a informação é mantida.

Para o "*Input Gate*", temos o seguinte processo:

$$I_t = \sigma(Wx(t) + W_{-1}g(t-1))$$

$$A_t = h(Wx(t) + W_{-1}g(t-1))$$

$$B(t) = B(t-1).F_t + I_t.A_t$$

Sendo h uma função de ativação (Comumente utiliza-se a $tanh$). Assim, o es-

tado atual de um dado neurônio é composto pela agregação de antigas e novas informações.

Por fim, o "*Output Gate*":

$$O_t = \sigma(Wx(t) + W_{-1}g(t-1))$$

$$g(t) = O_t.h(B(t))$$

Assim, define-se a informação que será predita pelo modelo (Caso tal processo ocorra na última camada), ou quais dados serão utilizados de entrada para a camada seguinte.

Com a execução de todos esses passos, torna-se possível a obtenção de um resultado refinado para uma dada predição, além do contorno do problema de "*Vanishing Gradients*".

Em um exemplo ilustrado de todos os processos, teríamos o seguinte fluxo:

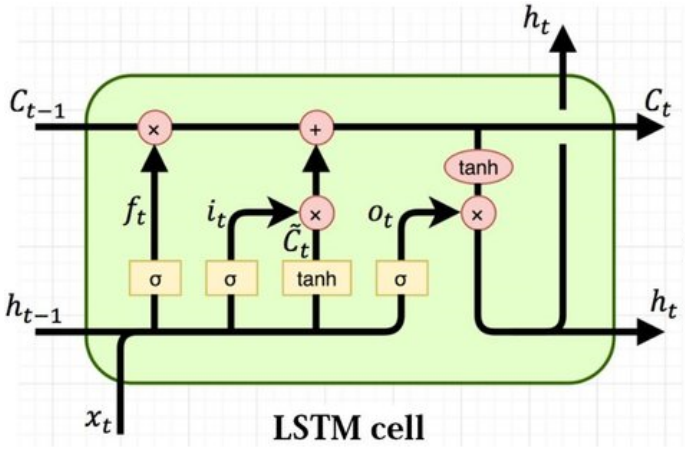

Figura 10: Fluxo LSTM

Fonte: https://hackinganalytics.wordpress.com/2018/12/09/series-temporais-com-lstm/

4 Aplicações Possíveis e Problemas Reais

Dada sua vasta versatilidade e maleabilidade para problemas diversos, redes neurais se tornaram estruturas recorrentes em aplicações do mundo moderno; desde processos cotidianos envolvendo previsões financeiras e climáticas, até sistemas robustos de recomendação que funcionam como pilares de receita para redes sociais e plataformas de

streaming como "*Netflix*", "*Amazon Prime*" e "*HBO Max*".

No contexto e realidade brasileira, a utilização de sistemas de inteligência artifical são comuns em setores tais como o agropecuário, onde aplicações como a análise do solo e de amplos terrenos através de imagens geradas por satélites, além do constante monitoramento climático, propiciam uma potencialização no aproveitamento e resultado de colheitas. Além de aplicações com vistas analíticas, tem se aumentado gradativamente com o passar dos anos a utilização de sistemas e maquinários que propiciam uma automatização dos processos práticos do campo agropecuário, como tecnologias de colheita e plantio automáticos; e também drones para a aplicação de fertilizantes e inseticidas.

Em um âmbito científico, a utilização de inteligência artificial é encontrada em todas as áreas de estudo e campos do conhecimento, desde análises de textos com o auxílio de redes neurais para classificação e identificação de entes e características importantes (Prasanna and Rao [2018]), até a utilização de *softwares* com embasamento em técnicas e conceitos de *Deep Learning* para a construção e refinamento de imagens através de dados obtidos via coleta de satélites, como é o caso da primeira imagem de um buraco negro apresentada em um *paper*(Bouman et al. [2019]). No contexto da pandemia de COVID-19, inúmeras pesquisas se utilizaram de sistemas de inteligência artificial com vistas de otimizar o processo de estudo e obtenção de potenciais remédios e vacinas (Abubaker Bagabir et al. [2022]); análises com o intuito de se obter potenciais previsões de picos e modelagens de comportamento para a evolução epidêmica se tornaram também usuárias recorrentes de técnicas e preceitos fundamentalmente ligados ao âmbito de estudos de IA (Musulin et al. [2021]). De forma geral, sua versatilidade e potencial quase ilimitado de aplicações em inúmeras áreas e contextos do mundo científico nos fornece um vislumbre da importância de tais tecnologias e técnicas para o que concebemos para nosso mundo contemporâneo e futuro, visto que, uma boa parte das pesquisas e processos realizados atualmente com o uso de IA para auxílio (Ou em seu cerne principal), serão responsáveis por impactos no desenvolvimento e na concepção de sociedade que temos atualmente em suas principais estruturas e pilares, tais como o mundo do trabalho e suas organizações e hierarquias, as soluções no âmbito de segurança e justiça, e até mesmo conceitos abstratos, como a privacidade e direito ao anonimato; assim, o desenvolvimento e aplicação de avanços das tecnologias de IA geram e irão continuar gerando debates acerca do papel e

responsabilidade da humanidade frente a essas mudanças (Chattopadhyay [2020]).

Para o caso das redes sociais, sua ampla utilização foi responsável inclusive por debates acerca de seu prejuízo a longo prazo para a preservação de ambientes diversos para convívio e vivência humana, visto que, com o aprimoramento dos sistemas de recomendação, que se utilizam amplamente da enorme quantidade de dados geradas a todo momento (Fenômeno que foi impulsionado pela popularização dos *smartphones*), o surgimento e preservação de "bolhas" se tornou cada vez mais recorrente (Indriani et al. [2020]), ocasionando com que grupos de pessoas passassem a compreender que a realidade se resume apenas ao que sua "*Timeline*" apresenta. Discussões modernas no âmbito de ciências sociais e políticas encontram, inclusive, correlação direta do fenômeno de "bolhas" em relação a ideologias políticas com a degradação e retrocesso de modelos lidos como democráticos nos últimos anos (Bozdag and van den Hoven [2015]), além de debates acerca do poder que as chamadas "*Big Techs*" possuem no mundo contemporâneo, uma capacidade de influência jamais vista anteriormente na história humana (Gurumurthy and Bharthur [2018]).

5 Conclusão

Com uma longa história de desenvolvimento anterior até se tornar um ponto central do debate moderno acerca de tecnologias inovadoras e com potencial de influência gigantesco para o que concebemos como futuro, as redes neurais se mostram como entes que não possuem um formato perfeito com variáveis e estruturas pré definidas para um dado resultado, dessa forma, sua compreensão em detalhes, e estudos acerca de possíveis aprimoramentos, é essencial, e tem acontecido com certa frequência e rápidez em tempos recentes. É necessário também reforçar a influência que tais estruturas já possuem (E tende a aumentar com o tempo) em nossa vida cotidiana, desde pequenas coisas, como uma simples recomendação de produtos para compra em um *e-commerce*, até amplos debates acerca do estado democrático de direito e liberdade de expressão; sendo assim, é necessário o reforço de discussões e análises acerca da ética e moral na utilização de novas tecnologias, com fins de prevenção para possíveis cenários catastróficos para o desenvolvimento da sociedade e até mesmo, do próprio indivíduo.

Bibliografia

Sali Abubaker Bagabir, Nahla Khamis Ibrahim, Hala Abubaker Bagabir, and Raghdah Hashem Ateeq. Covid-19 and artificial intelligence: Genome sequencing, drug development and vaccine discovery. *Journal of Infection and Public Health*, 15(2):289–296, 2022. ISSN 1876-0341. doi: https://doi.org/10.1016/j.jiph.2022.01.011. URL `https://www.sciencedirect.com/science/article/pii/S1876034122000144`.

Katherine L. Bouman, Kazunori Akiyama, and etal. First m87 event horizon telescope results. i. the shadow of the supermassive black hole. *The Astrophysical Journal Letters,*, 2019. doi: https://doi.org/10.3847/2041-8213/ab0ec7.

Engin Bozdag and Jeroen van den Hoven. Breaking the filter bubble: democracy and design. *Ethics Inf Technol*, 17:249–265, 2015. doi: https://doi.org/10.1007/s10676-015-9380-y.

H.K. Chattopadhyay. Artificial intelligence and its impacts on the society. *International Journal of Law*, 6:306–310, 11 2020.

Rumelhart D., Hinton, and G. Williams R. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. doi: https://doi.org/10.1038/323533a0.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. URL `http://jmlr.org/papers/v12/duchi11a.html`.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of Machine Learning Research*, 15:315–323, 2010. URL `https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 8.7.1. The MIT Press, 2016.

Anita Gurumurthy and Deepti Bharthur. Democracy and the algorithmic turn: Issues, challenges and the way forward. *Informal or Other Publication*, 07 2018.

G. E. Hinton, N. Srivastava, J. A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *Informal or Other Publication*, pages 1–18, 2012. URL https://arxiv.org/pdf/1207.0580.pdf.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Rmsprop. *Informal or Other Publication*, 2014. URL http://www.cs.toronto.edu/t̃ijmen/csc321/slides/lecture_slides_lec6.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

J.J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc.Natl.Acad.Sci.USA*, 79:2554—-2558, 1982. doi: https://doi.org/10.1073/pnas.79.8.2554.

Sri Indriani, Ditha Prasanti, and Rangga Permana. Analysis of the filter bubble phenomenon in the use of online media for millennial generation (an ethnography virtual study about the filter bubble phenomenon). *Nyimak: Journal of Communication*, 4: 199, 09 2020. doi: 10.31000/nyimak.v4i2.2538.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Informal or Other Publication*, 2015. doi: https://doi.org/10.48550/arXiv.1502.03167.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. *Informal or Other Publication*, 2015. URL http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *Published as a conference paper at ICLR 2015*, pages 1–15, 2015. URL https://arxiv.org/pdf/1412.6980.pdf.

Yuzhen Lu and Fathi M. Salem. Simplified gating in long short-term memory (lstm) recurrent neural networks. *Informal or Other Publication*, 2017. URL https://arxiv.org/ftp/arxiv/papers/1701/1701.03441.pdf.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115—-133, 1943. doi: https://doi.org/10.1007/BF02478259.

Jelena Musulin, Sandi Baressi Šegota, Daniel Štifanić, Ivan Lorencin, Nikola Anđelić, Tijana Šušteršič, Anđela Blagojević, Nenad Filipović, Tomislav Ćabov, and Elitza Markova-Car. Application of artificial intelligence-based regression methods in the problem of covid-19 spread prediction: A systematic review. *International Journal of Environmental Research and Public Health*, 18(8), 2021. ISSN 1660-4601. URL `https://www.mdpi.com/1660-4601/18/8/4287`.

P. Prasanna and Dr Rao. Text classification using artificial neural networks. *International Journal of Engineering and Technology(UAE)*, 7:603–606, 01 2018. doi: 10.14419/ijet.v7i1.1.10785.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386—-408, 1958. doi: https://doi.org/10.1037/h0042519.

Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *Informal or Other Publication*, pages 1—-16, 2019. doi: https://doi.org/10.48550/arXiv.1912.05911.