"A verdadeira questão não é se as máquinas pensam, mas se os seres humanos pensam. O mistério que envolve uma máquina pensante já envolve os seres humanos" (B.F. Skinner, 1969).

Normalização

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Normalização

Em teoria, os pesos da rede podem se adaptar aos dados, independentemente de suas escalas.

Classe	Potência (cv)	Litros Motor (L)	Km Rodados	...
1	173	1,5	68000	...
1	115	1,6	135000	...
2	85	1,0	84000	...
...	...	...	...	...

Normalização

Em teoria, os pesos da rede podem se adaptar aos dados, independentemente de suas escalas.

Classe	Potência (cv)	Litros Motor (L)	Km Rodados	...
1	173	1,5	68000	...
1	115	1,6	135000	...
2	85	1,0	84000	...
...	...	...	...	...

As escalas são muito diferentes. A curvatura do espaço de erros varia muito nas diferentes dimensões.

Normalização

Na prática, normalizar os dados (ex.: para ter média zero e variância um) resulta em um espaço de erros mais bem comportado.

Facilita a convergência.

Vimos isso nas primeiras aulas com o treinamento do Perceptron.

Foram necessárias menos épocas para treinar o Perceptron com os dados não normalizados.

Normalização - Algoritmo

Considere o dataset de **treinamento**, contendo *N* instâncias.

Para cada dimensão i dos dados de treinamento, calcule a média μ_i e variância σ_i^2.

$$\mu_i = \frac{1}{N} \sum_{n=1}^{N} x_{ni}$$

$$\sigma_i^2 = \frac{1}{N} \sum_{n=1}^{N} (x_{ni} - \mu_i)^2$$

Normalização - Algoritmo

Antes de processar qualquer instância, durante o treino, validação, ou teste, redimensione-a:

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i}$$

Onde *i* representa a dimensão sendo normalizada.

Faça você mesmo

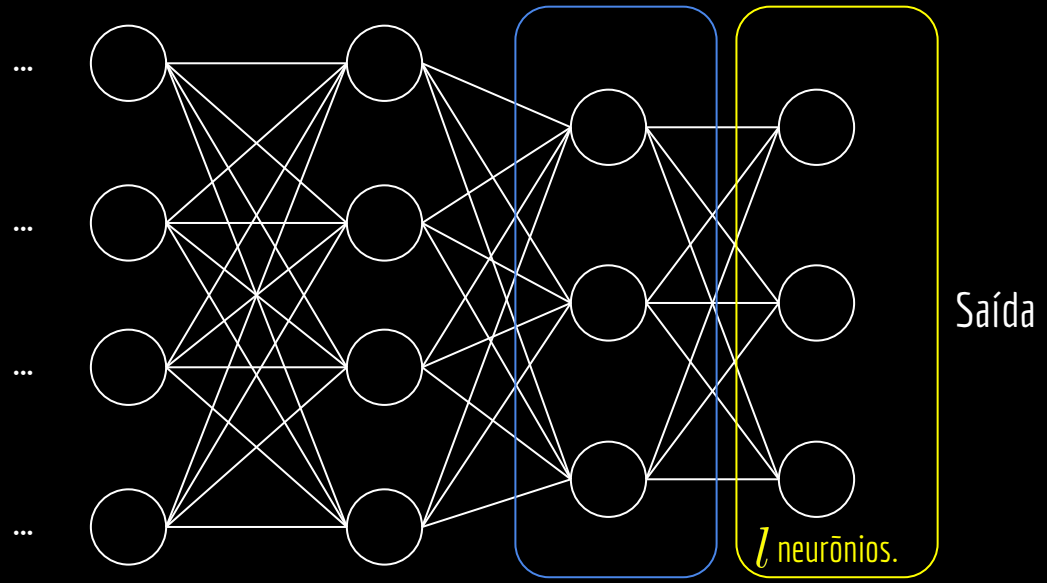
Normalize os dados de treinamento para o problema disponibilizado no Google Colab.

σ é a função de ativação.

Vanishing Gradients

Pela regra da cadeia.

$$\frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w} \cdot \sum_{l} \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w}$$



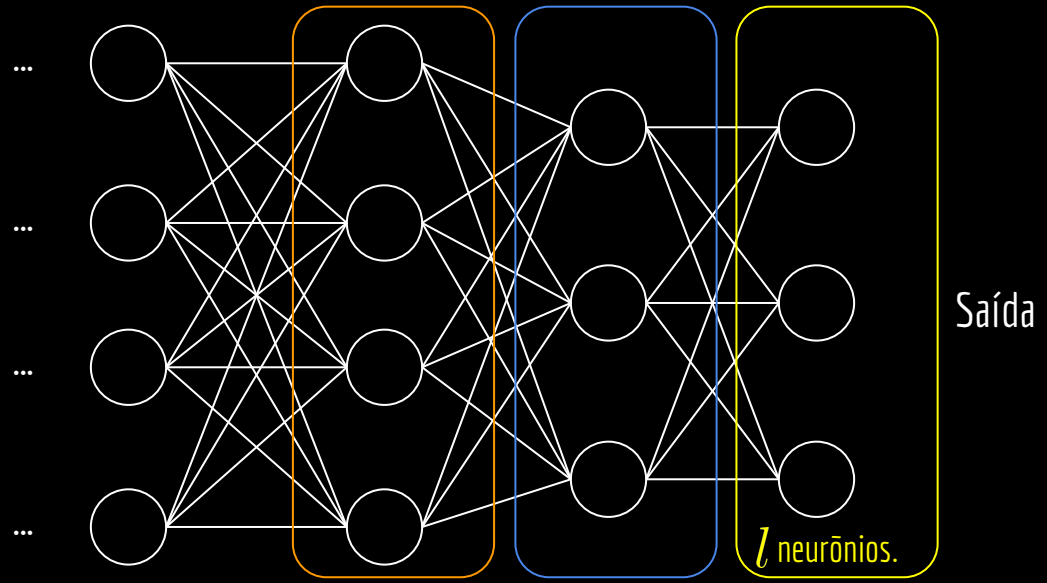
obs.: os índices foram omitidos.

σ é a função de ativação.

Vanishing Gradients

Pela regra da cadeia.

$$\cdots \sum \cdots \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w} \cdot \sum_{l} \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w}$$



obs.: os índices foram omitidos.

σ é a função de ativação.

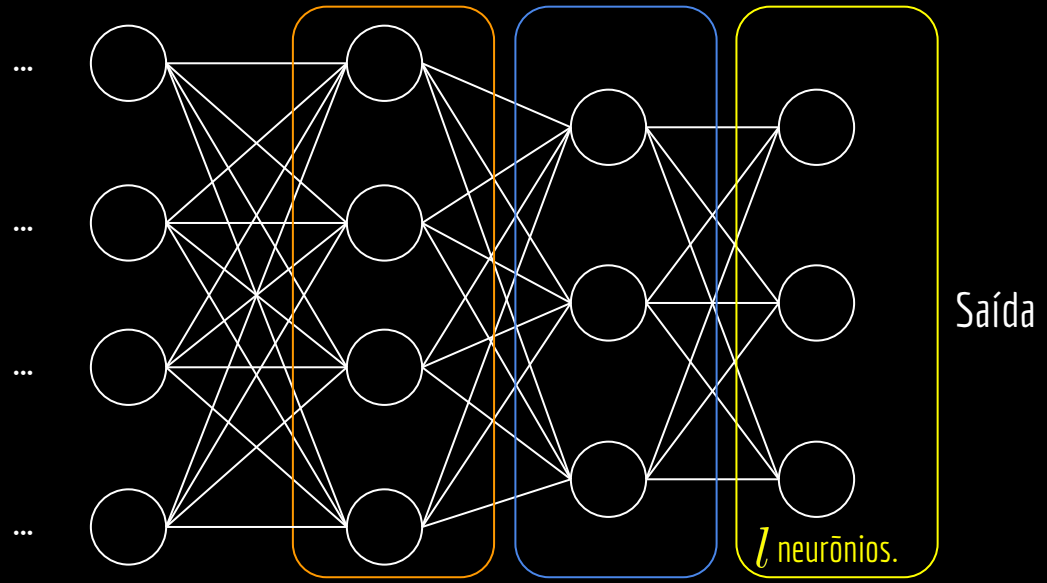
Vanishing Gradients

$$\cdots \sum \cdots \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w} \cdot \sum_{l} \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w}$$

Quanto mais camadas, mais termos são adicionados nos produtos.

Se a rede é profunda e:

1. Se os termos têm magnitude ‹ 1.
   O gradiente vai tender a zero.
   Desaparecer ou "*vanish*".



obs.: os índices foram omitidos.

σ é a função de ativação.

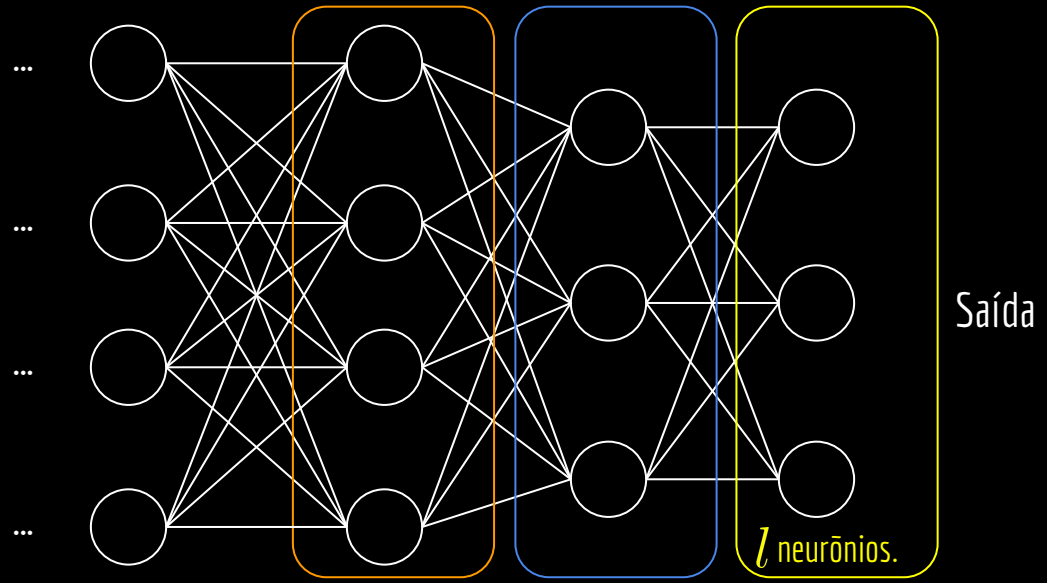
Vanishing Gradients

Quanto mais camadas, mais termos são adicionados nos produtos.

$$\cdots \sum \cdots \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w} \cdot \sum_{l} \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial out} \cdot \frac{\partial out}{\partial w}$$

Se a rede é profunda e:

1. Se os termos têm magnitude ‹ 1.
   O gradiente vai tender a zero.
   Desaparecer ou "*vanish*".
2. Se os termos têm magnitude › 1.
   O gradiente vai tender a ∞.
   Explodir.



obs.: os índices foram omitidos.

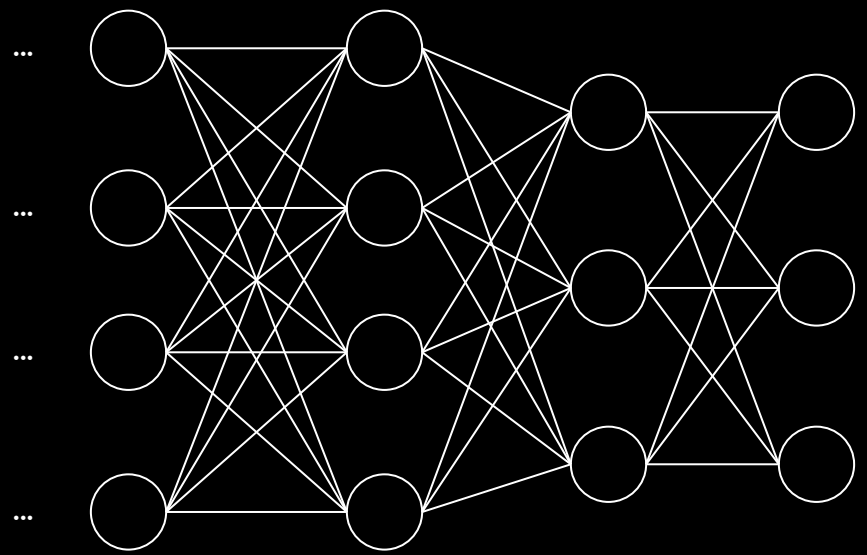
Vanishing Gradients e Normalização

A normalização, além de facilitar o aprendizado, mitiga o problemas de *vanishing gradients* e *exploding gradients*.

Mas para isso, precisamos normalizar a entrada de cada camada oculta.

Batch Normalization

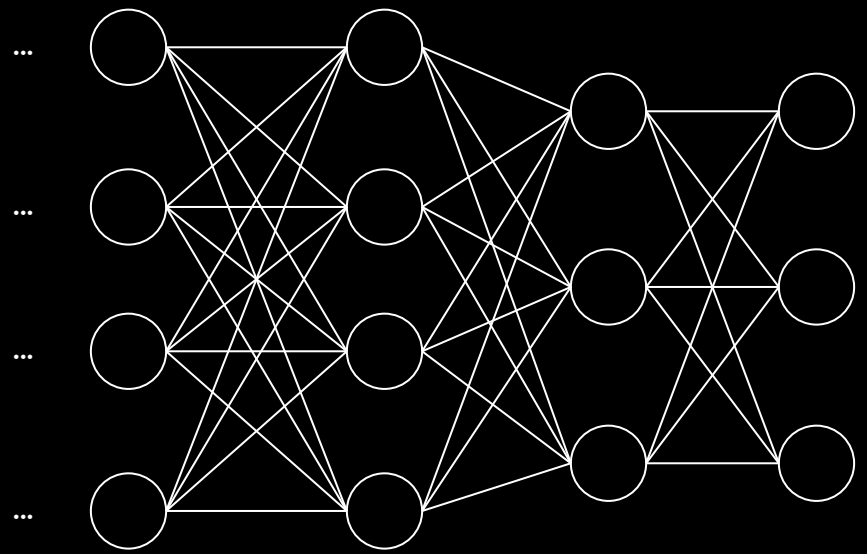
Ideia: em cada camada oculta, normalizar as ativações para obter ativações com média zero, e variância um.



Batch Normalization

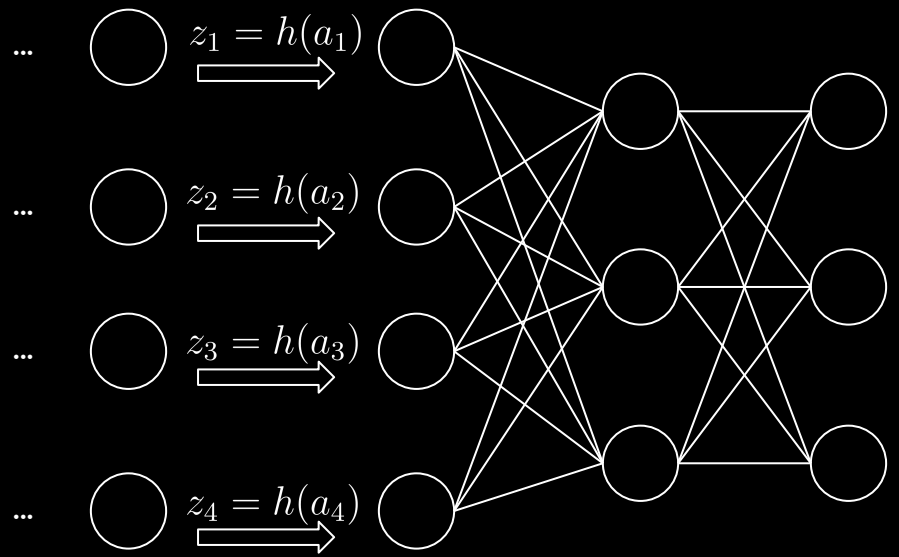
Ideia: em cada camada oculta, normalizar as ativações para obter ativações com média zero, e variância um.

Como a ativação depende dos pesos, e os pesos são atualizados a cada batch, será necessário realizar a normalização a cada batch.



Batch Normalization

Seja $z_i = h(a_i)$ a ativação do i-ésimo neurônio da camada oculta, gerada pela aplicação da não-linearidade $h(a_i)$, onde a_i é o produto escalar calculado para esse neurônio, dados os pesos e entradas da camada anterior.

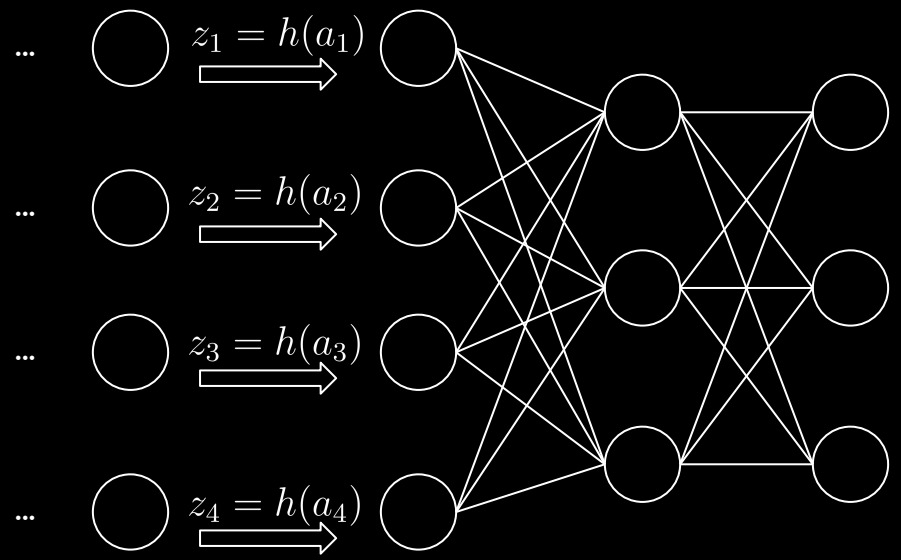


Batch Normalization

Seja $z_i = h(a_i)$ a ativação do i-ésimo neurônio da camada oculta, gerada pela aplicação da não-linearidade $h(a_i)$, onde a_i é o produto escalar calculado para esse neurônio, dados os pesos e entradas da camada anterior.

Vamos normalizar cada a_i.

Obs.: também seria correto normalizar z_i.



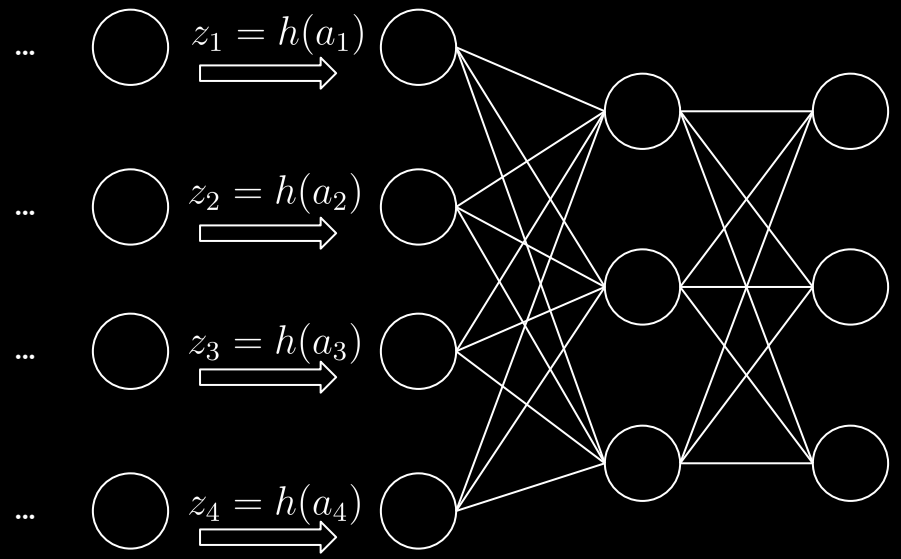
Batch Normalization

Para cada *minibatch* de tamanho *K*.

$$\mu_i = \frac{1}{K}\sum_{n=1}^{K} a_{ni}$$

$$\sigma_i^2 = \frac{1}{K}\sum_{n=1}^{K} (a_{ni} - \mu_i)^2$$

$$\hat{a}_{ni} = \frac{a_{ni} - \mu_i}{\sqrt{\sigma_i^2 + \delta}}$$



Batch Normalization

Problema: reduzimos os graus de liberdade dos parâmetros da camada, reduzindo a capacidade de representação.

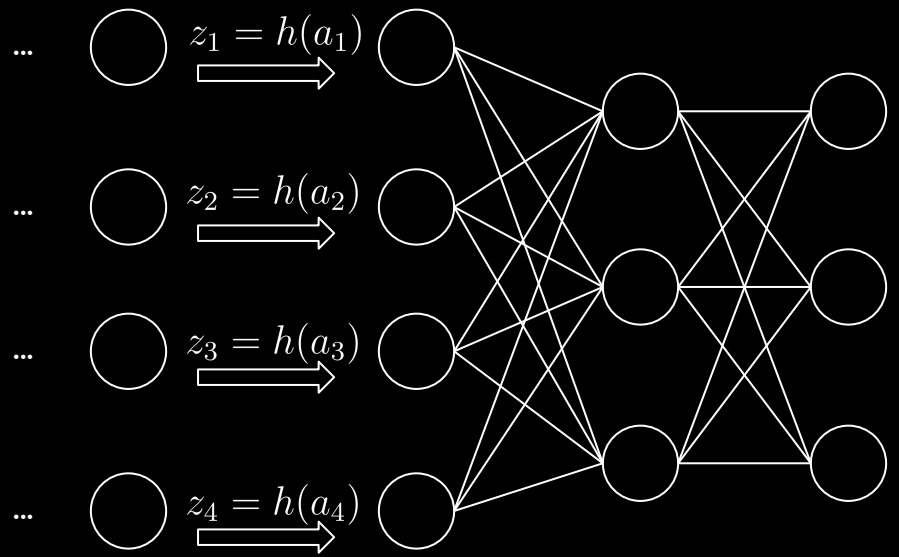
Vamos "desfazer" a operação:

$$\tilde{a}_{ni} = \gamma_i \hat{a}_{ni} + \beta_i$$

γ_i é a variância.

β_i é a média.

β_i e γ_i são aprendidos durante o treinamento.



Batch Normalization

Mesmo desfazendo a operação, tornamos o treinamento mais simples.

A média e variância originais eram determinadas pelos dados do batch de treinamento.

Agora, são determinados por dois parâmetros, que são aprendidos pela rede.

Como são aprendidos, podemos considerar o Batch Normalization como uma camada da rede.

Note que as equações são diferenciáveis em relação a β_i e γ_i.

Batch Normalization

Ao final do treino, **não** temos os valores de μ_i e σ_i^2 para as camadas ocultas.

Podemos:

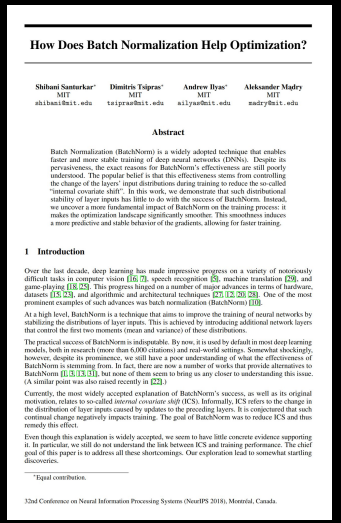
1. passar uma última vez por todos os dados de treinamento para computar esses valores, sem atualizar os pesos da rede.
2. Ou, estimar os valores durante o treinamento através de médias móveis.

$$\overline{\mu}_i^\tau = \alpha \overline{\mu}_i^{\tau-1} + (1-\alpha)\mu_i$$

$$\overline{\sigma}_i^\tau = \alpha \overline{\sigma}_i^{\tau-1} + (1-\alpha)\sigma_i$$

Batch Normalization

Uma das principais hipóteses é de que o batch normalization ajuda no treinamento por tornar a superfície de erro mais suave.



How Does Batch Normalization Help Optimization?

Shibani Santurkar[*] MIT shibani@mit.edu
Dimitris Tsipras[*] MIT tsipras@mit.edu
Andrew Ilyas[*] MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

1 Introduction

Over the last decade, deep learning has made impressive progress on a variety of notoriously difficult tasks in computer vision [16, 7], speech recognition [9], machine translation [29], and game-playing [18, 25]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 23], and algorithmic and architectural techniques [27, 12, 20, 28]. One of the most prominent examples of such advances was batch normalization (BatchNorm) [10].

At a high level, BatchNorm is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions.

The practical success of BatchNorm is indisputable. By now, it is used by default in most deep learning models, both in research (more than 6,000 citations) and real-world settings. Somewhat shockingly, however, despite its prominence, we still have a poor understanding of what the effectiveness of BatchNorm is stemming from. In fact, there are now a number of works that provide alternatives to BatchNorm [1, 3, 13, 31], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [22].)

Currently, the most widely accepted explanation of BatchNorm's success, as well as its original motivation, relates to so-called *internal covariate shift* (ICS). Informally, ICS refers to the change in the distribution of layer inputs caused by updates to the preceding layers. It is conjectured that such continual change negatively impacts training. The goal of BatchNorm was to reduce ICS and thus remedy this effect.

Even though this explanation is widely accepted, we seem to have little concrete evidence supporting it. In particular, we still do not understand the link between ICS and training performance. The chief goal of this paper is to address all these shortcomings. Our exploration lead to somewhat startling discoveries.

[*]Equal contribution.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

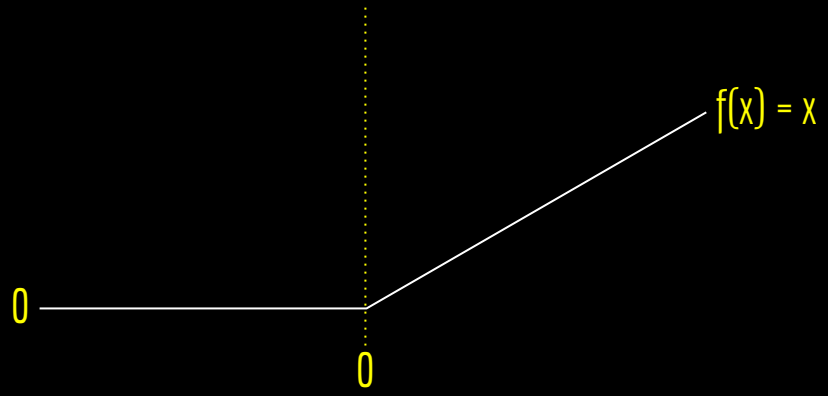
SANTURKAR, Shibani et al. How does batch normalization help optimization?. Advances in neural information processing systems, v. 31, 2018.

ReLU - Rectified Linear Unit

A função de ativação ReLU tem como uma de suas vantagens reduzir o problema do *vanishing gradients*.

Mas ainda apresenta problemas de *exploding gradients*. Por quê? Pesquise.

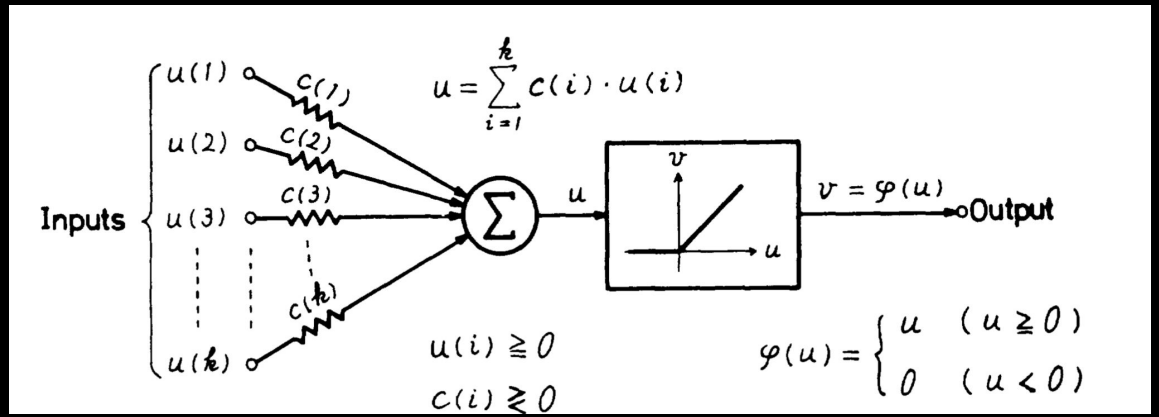
A função é popular e, na prática, leva a uma convergência mais rápida do que funções tradicionais, como sigmóide.

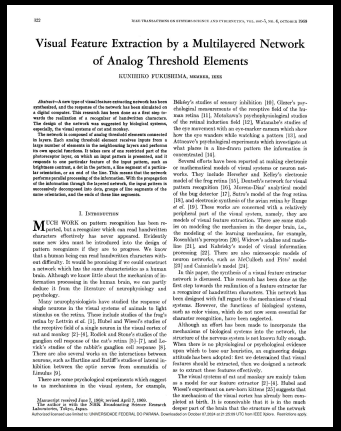


ReLU - Rectified Linear Unit

Problema da derivativa no ponto $x = 0$. Não há derivada, mas podemos aplicar alguns truques. Pesquise.

Proposta em 1969. Uso popularizado em CNNs a partir de 2010.





Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements

KUNIHIKO FUKUSHIMA, MEMBER, IEEE

FUKUSHIMA, Kunihiko. Visual feature extraction by a multilayered network of analog threshold elements. IEEE Transactions on Systems Science and Cybernetics, v. 5, n. 4, p. 322-333, 1969.

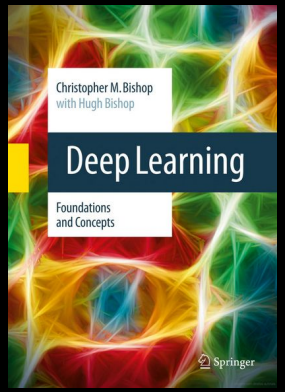
Exercícios

Considerando o programa disponibilizado no Google Colab, execute várias vezes e compare os resultados.

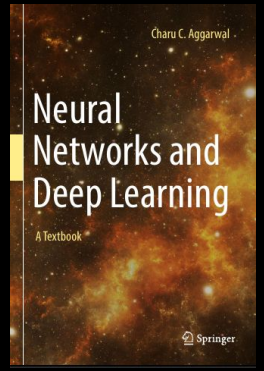
1. Sem utilizar normalização dos dados, e sem *batch normalization*.
2. Normalizando apenas os dados.
3. Normalizando os dados e incluindo *batch normalization*.

Referências

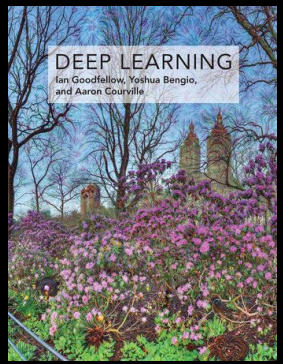
Bishop, C. M., Bishop, H. Deep Learning: Foundations and Concepts. 2023.



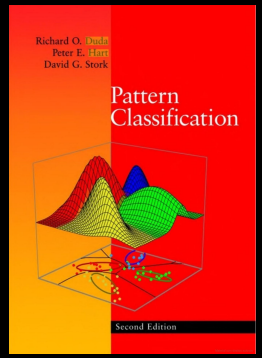
Aggarwal, C. C. Neural Networks and Deep Learning. 2018.



Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. 2016.



Duda, R. O., Hart, P. E., Stork, D. G. Pattern Classification. 2012.



Licença