

XI Encontro de Física e Astronomia da UFSC

Introdução às redes neurais com Python e Keras

Prof. Dr. Robson da Silva Oliboni



Introdução às redes neurais com Python e Keras

Sumário

1. A natureza da generalização
2. Seleção de modelos: Validação
3. Melhorando o ajuste do modelo
4. Melhorando a generalização do modelo

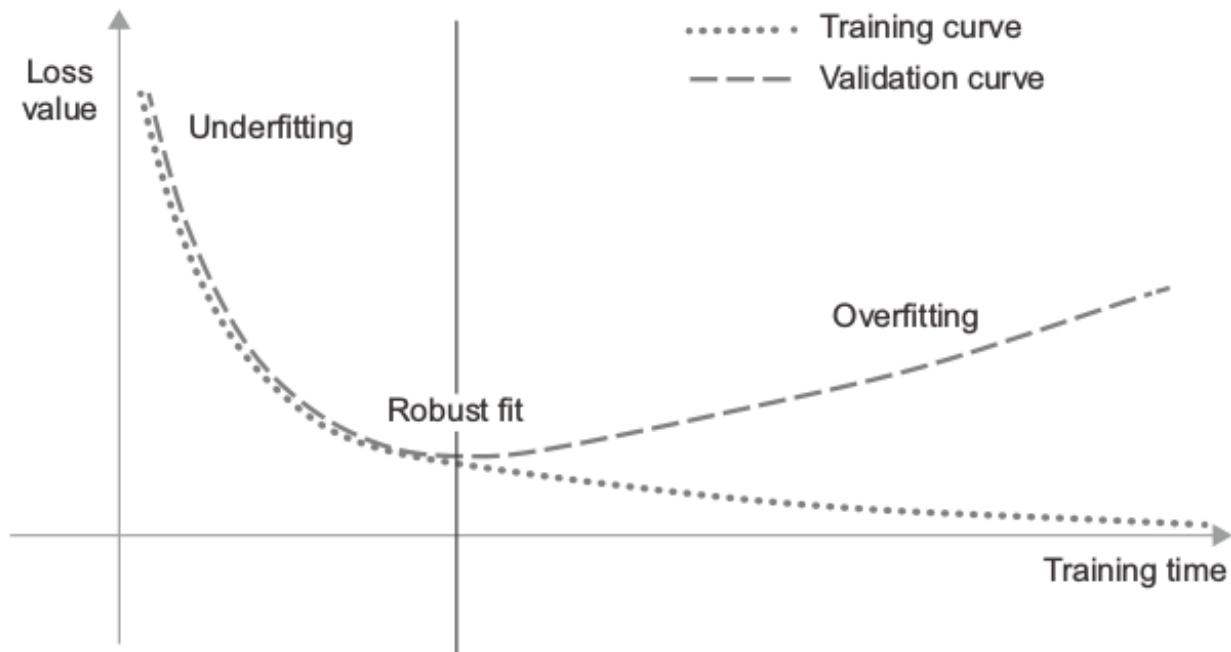
Generalização vs optimização

Geralmente o desempenho no conjunto de teste é menor que o obtido no conjunto de treinamento.

Obter um bom desempenho no conjunto de treinamento é relativamente simples; a rede estaria *memorizando*, e não *generalizando* o problema, que é o objetivo do aprendizado.

Normalmente separa-se uma parte do conjunto de treinamento como *conjunto de validação*, para avaliar o potencial de generalização do modelo.

Em todo modelo de ML, há um balanço entre otimização e generalização; entre *subajuste* (*underfitting*) e *sobreajuste* (*overfitting*).



[2]

Figure 5.1 Canonical overfitting behavior

Possíveis razões de sobreajuste nos dados de treinamento: Ruídos, incerteza, características raras...

Razões de sobreajuste

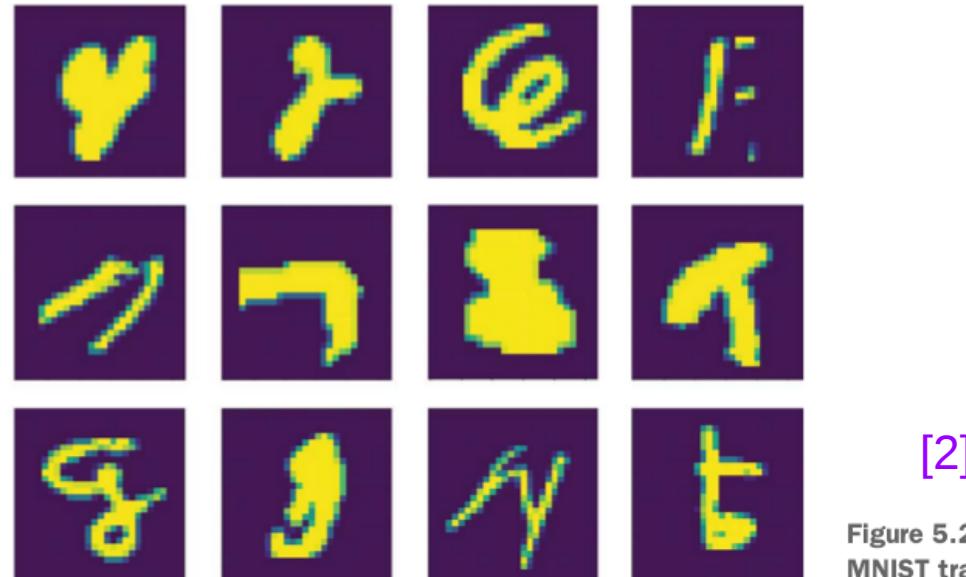


Figure 5.2 Some pretty weird
MNIST training samples

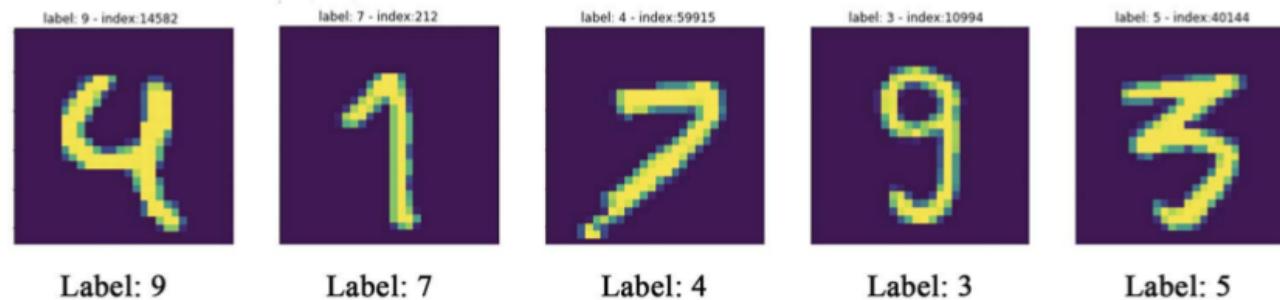


Figure 5.3 Mislabeled MNIST training samples

[2]

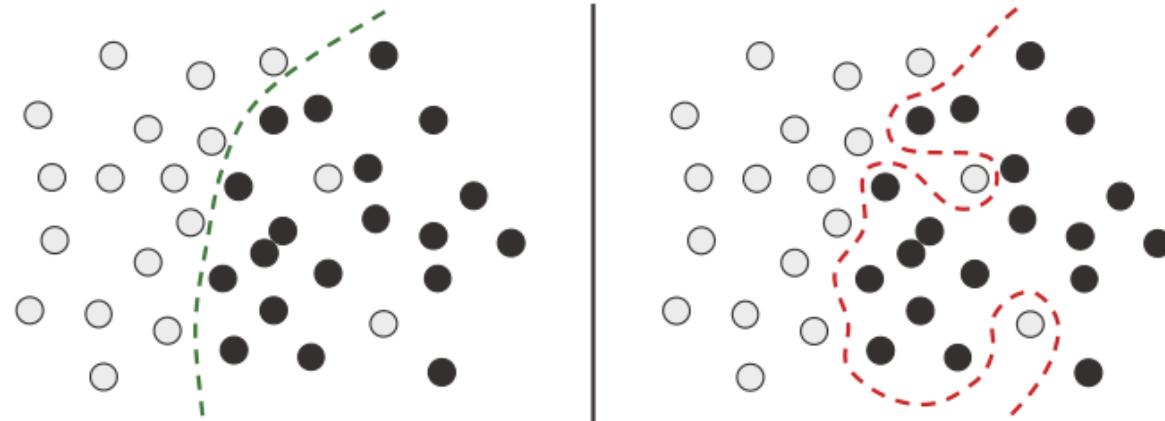


Figure 5.4 Dealing with outliers: robust fit vs. overfitting

[2]

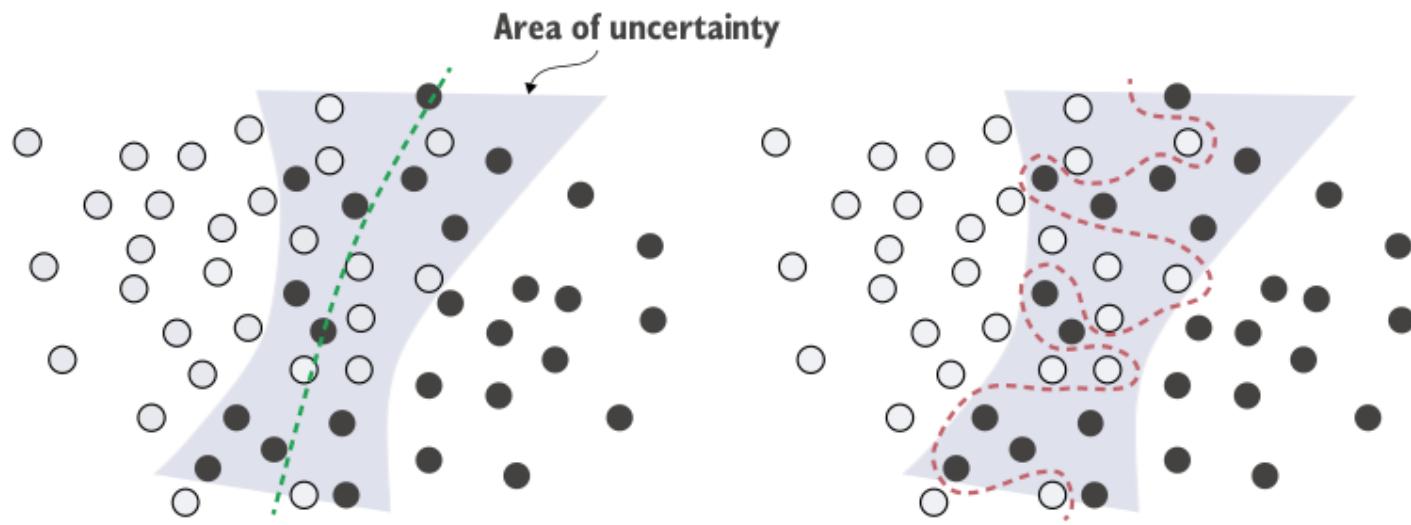


Figure 5.5 Robust fit vs. overfitting giving an ambiguous area of the feature space

[2]

A natureza da generalização

A generalização nos modelos de NNs tem pouco a ver propriamente com os modelos, mas com a estrutura da informação no mundo real.

A **hipótese de *manifold*** postula que todos os dados naturais se encontram em um manifold de menor dimensão em relação a um espaço de maior dimensão em que está codificada.

“*Manifolds are spatial regions (clusters) which represent the patterns essential to a concept.*”

A hipótese de *manifold* implica que:

- Os modelos de ML tem apenas que ajustar subespaços relativamente simples, de baixa dimensão e altamente estruturados dentro de seu potencial espaço de inputs (*manifolds latentes*);
- Dentro de um desses *manifolds*, é sempre possível interpolar entre dois inputs, isto é, metamorfosear um em outro via um caminho contínuo tal que todos os pontos se encontram no *manifold* (subespaço é contínuo, e os pontos no subespaço são conectados entre si).

Ao poder interpolar pontos no treinamento, podemos relacioná-los com pontos não vistos que são próximos no *manifold*; podemos entender a totalidade do espaço usando apenas uma amostra do espaço.

Mas a interpolação não é a totalidade da generalização; ela permite generalização *local*.

Seres humanos são capazes de generalização *extrema*, por mecanismos cognitivos como abstração, modelos simbólicos do mundo, raciocínio, lógica, senso comum, priores inatos sobre o mundo → racionalidade.

Os dados formam um *manifold* altamente estruturado de baixa dimensão dentro do espaço de input; durante a otimização, haverá um ponto em que o modelo se aproxima do manifold natural dos dados.

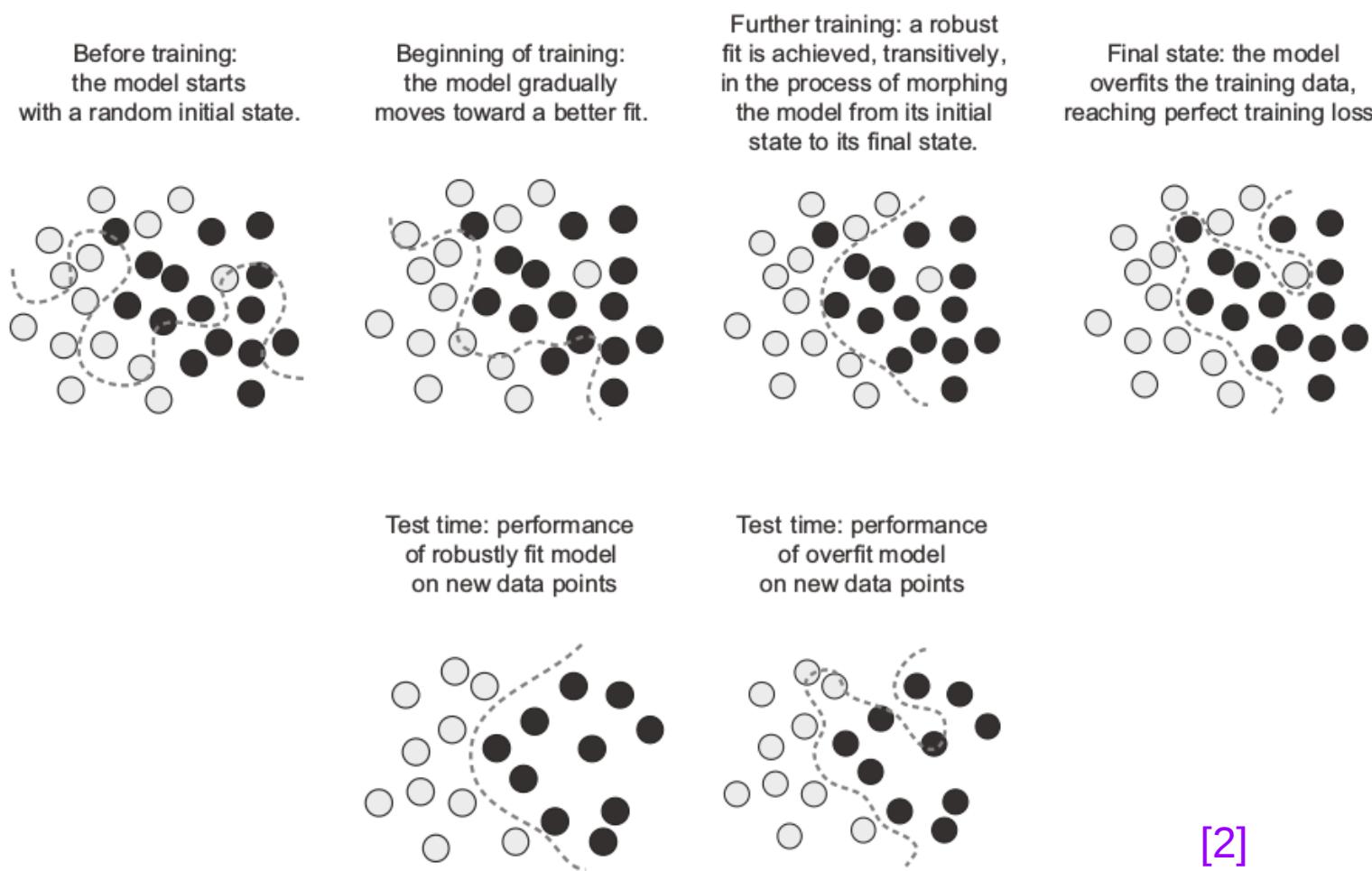


Figure 5.10 Going from a random model to an overfit model, and achieving a robust fit as an intermediate state

Why Deep Neural Networks? <https://www.youtube.com/watch?v=e5xKayCBOeU&t=601s>

O poder de generalização é mais uma consequência da estrutura natural de seus dados que uma consequência de qualquer propriedade do seu modelo. Apenas será capaz de generalizar se seus dados formam um *manifold* onde os pontos podem ser interpolados.

A curadoria de dados e *feature engineering* são essenciais à generalização. Além disso, para o modelo ter bom desempenho é necessário que seja treinado em uma amostra densa de seu espaço de input.

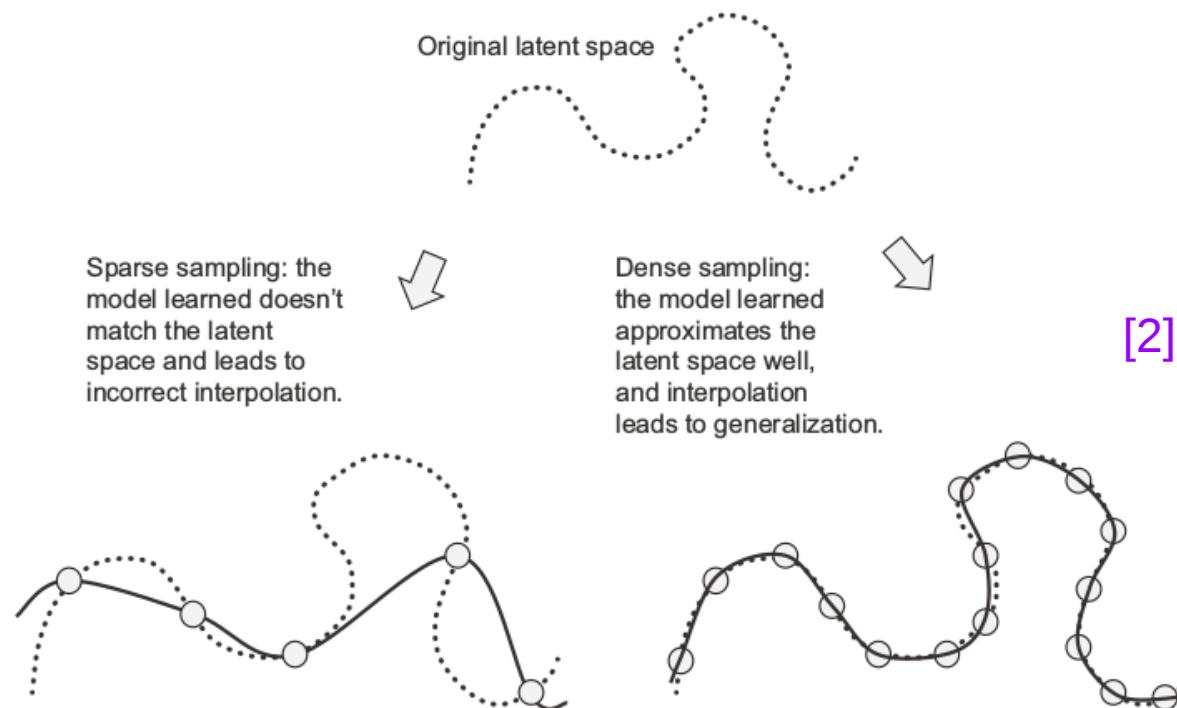


Figure 5.11 A dense sampling of the input space is necessary in order to learn a model capable of accurate generalization.

Seleção de modelos: Validação

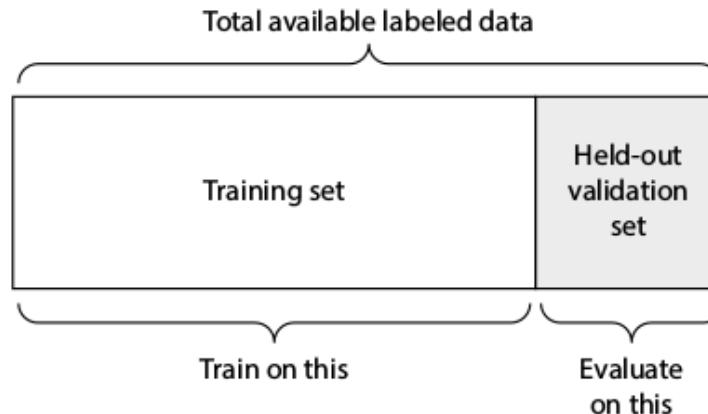
A validação de um modelo consiste em separar os dados em três conjuntos: treinamento, validação e teste.

O desenvolvimento de um modelo sempre envolve modificar sua configuração, como o número de camadas ou o tamanho de uma camada (seus hiperparâmetros).

Modelo: Linear ou redes neurais? → Redes neurais; Quantas camadas?

Três formas de validação: método holdout de validação cruzada, método K-fold de validação cruzada e método K-fold iterativo de validação cruzada.

Validação simples



[2]

Figure 5.12 Simple holdout validation split

Separase 20-30% do conjunto de dados para o conjunto de validação; utilizado quando se tem dados suficientes.

```
num_validation_samples = 10000
np.random.shuffle(data)                                ↗ Shuffling the data is
                                                       ↗ usually appropriate.

Defines the validation set   ↗ validation_data = data[:num_validation_samples]
                           ↗ training_data = data[num_validation_samples:]
                           ↗ model = get_model()
                           ↗ model.fit(training_data, ...)
                           ↗ validation_score = model.evaluate(validation_data, ...)

                           ↗ Defines the training set
                           ↗ Trains a model on the
                           ↗ training data, and evaluates
                           ↗ it on the validation data

                           ...                                     ↗ At this point you can tune your model,
                                                       ↗ retrain it, evaluate it, tune it again.

model = get_model()
model.fit(np.concatenate([training_data,
                         validation_data]), ...)
test_score = model.evaluate(test_data, ...)           ↗ Once you've tuned your
                                                       ↗ hyperparameters, it's common to
                                                       ↗ train your final model from scratch
                                                       ↗ on all non-test data available.
```

[2]

Validação K-fold

Este método é útil quando se tem poucos dados e a performance de seu modelo mostra variância significativa em relação à separação do conjunto de treinamento.

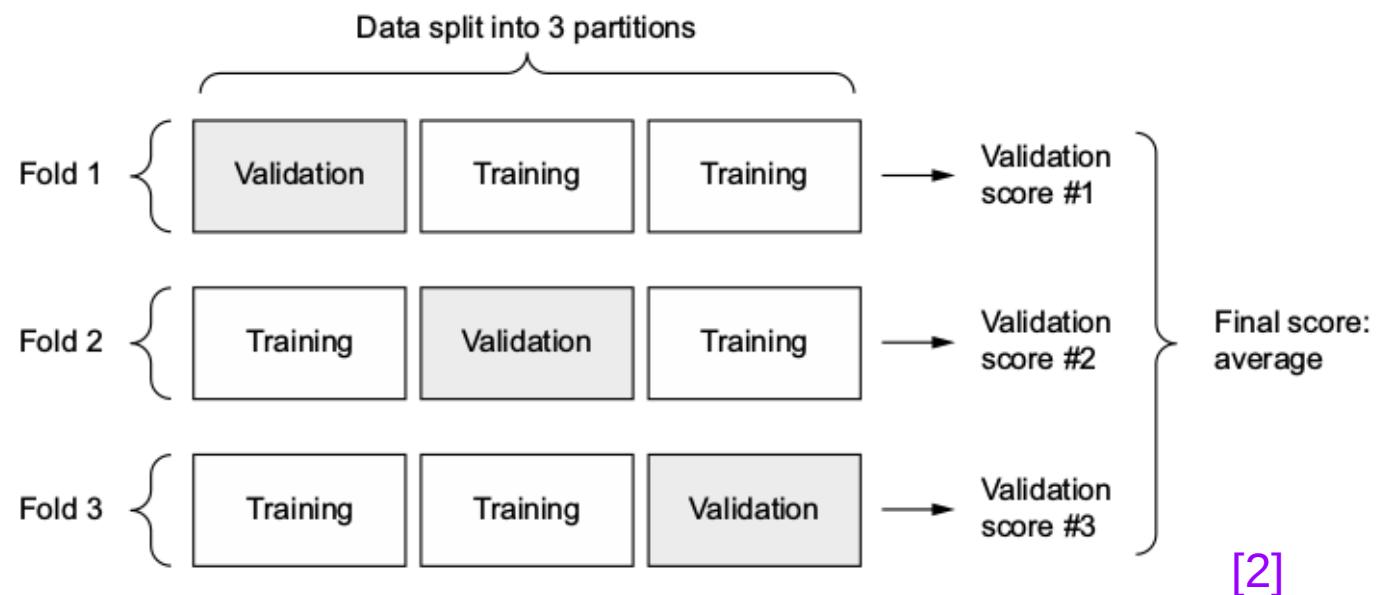


Figure 5.13 K-fold cross-validation with K=3

```

k = 3
num_validation_samples = len(data) // k
np.random.shuffle(data)
validation_scores = []
for fold in range(k):
    validation_data = data[num_validation_samples * fold:
                           num_validation_samples * (fold + 1)]
    training_data = np.concatenate(
        data[:num_validation_samples * fold],
        data[num_validation_samples * (fold + 1):])
    model = get_model()
    model.fit(training_data, ...)
    validation_score = model.evaluate(validation_data, ...)
    validation_scores.append(validation_score)
    validation_score = np.average(validation_scores)
    model = get_model()
    model.fit(data, ...)
    test_score = model.evaluate(test_data, ...)

    Selects the validation-data partition
    Creates a brand-new instance of the model (untrained)
    Validation score: average of the validation scores of the k folds
    Trains the final model on all non-test data available

```

Uses the remainder of the data as training data. Note that the + operator represents list concatenation, not summation.

[2]

Validação K-fold iterativa

Consiste em aplicar a validação K-fold várias vezes, misturando os dados toda vez antes de separá-los K vezes. O escore final é a média dos escores obtidos em cada rodada na validação K-fold. Acaba-se treinando e validando $P * K$ modelos.

Melhorando o ajuste do modelo

Você sempre pode sobreajustar os seus dados!

Normalmente encontra-se os seguintes problemas:

- O modelo não melhora o desempenho durante o treinamento;
- O desempenho é bom no treinamento, mas o modelo não generaliza;
- O desempenho diminui no treinamento e na validação, mas o modelo não sobreajusta.

– Ajustar parâmetros da otimização

Se a otimização não começar ou se estabilizar cedo, é necessário ajustar os parâmetros da otimização, em geral a taxa de aprendizado e/ou aumentar o tamanho do batch.

– Melhorar os priores da arquitetura da rede

Se o modelo otimiza, mas as métricas de validação não melhoram, o modelo não generaliza; é a pior situação possível!

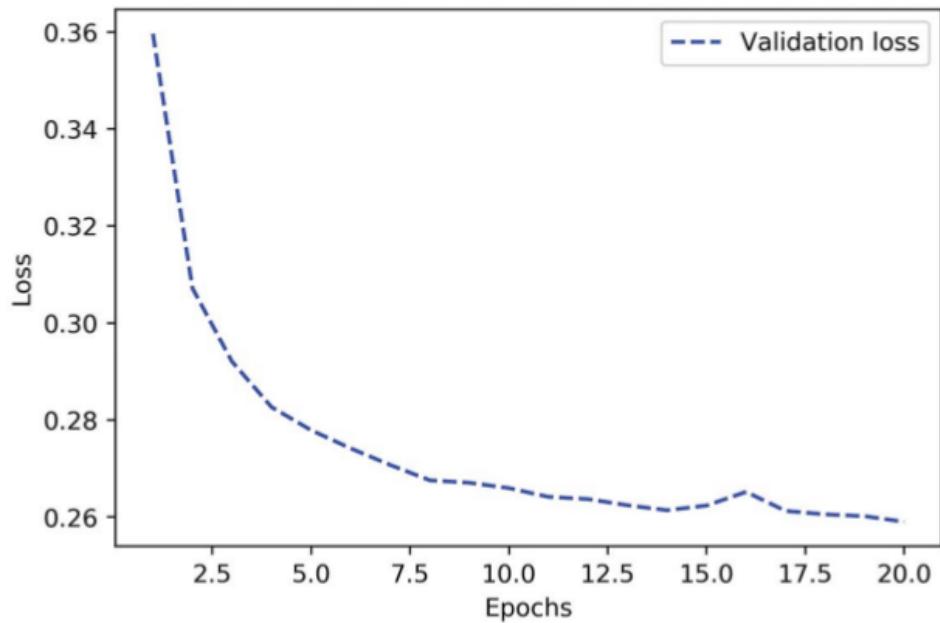
Há algo fundamentalmente errado com seu modelo: os dados de input não contém informação suficiente para prever os alvos; modelo não é adequado para o problema...

– Aumentar a capacidade do modelo

Se o modelo otimiza, as métricas de validação diminuem, você alcança um nível de generalização, mas não sobreajusta.

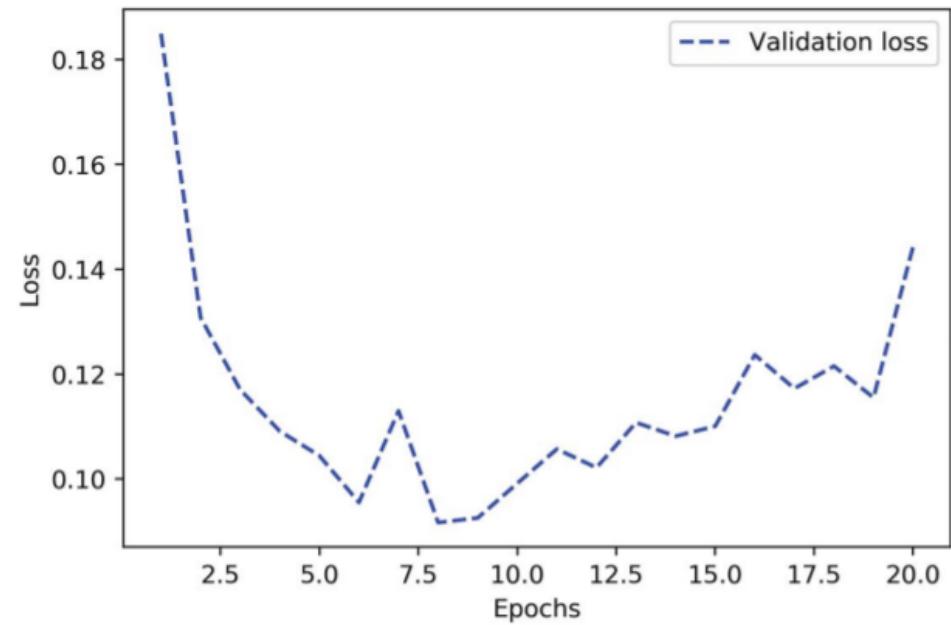
É necessário aumentar o poder de representação de seu modelo; você precisa de um modelo com maior capacidade, isto é, um modelo capaz de armazenar mais informação.

Você pode aumentar o poder de representação do modelo adicionando mais camadas, usando camadas maiores, ou usando outras arquiteturas de rede (melhores priorizes de arquitetura).



[2]

Figure 5.14 Effect of insufficient model capacity on loss curves



[2]

Figure 5.15 Validation loss for a model with appropriate capacity

Melhorando a generalização

– Curadoria de dados

A generalização surge de uma estrutura latente de seus dados; logo, é essencial trabalhar com um dataset apropriado.

Gastar mais esforço na coleção de dados quase sempre gera um maior retorno que gastar o mesmo no desenvolvimento de um modelo melhor.

– *Feature engineering*

É o processo de usar nosso próprio conhecimento sobre os dados e o modelo em mãos para tornar o modelo melhor, tornando o problema mais fácil ao expressá-lo de maneira mais simples. Torna o espaço latente mais suave, simples, melhor organizado.

Era essencial em modelos de ML mais “rasos”; porém, características melhores levam a um desempenho melhor ao usar menos recursos e precisar de menos dados.

– Utilizando *early stopping*

É uma técnica para interromper o treinamento assim que a métrica de validação parar de melhorar, utilizando o melhor estado do modelo (salva o modelo após o melhor desempenho em uma época).

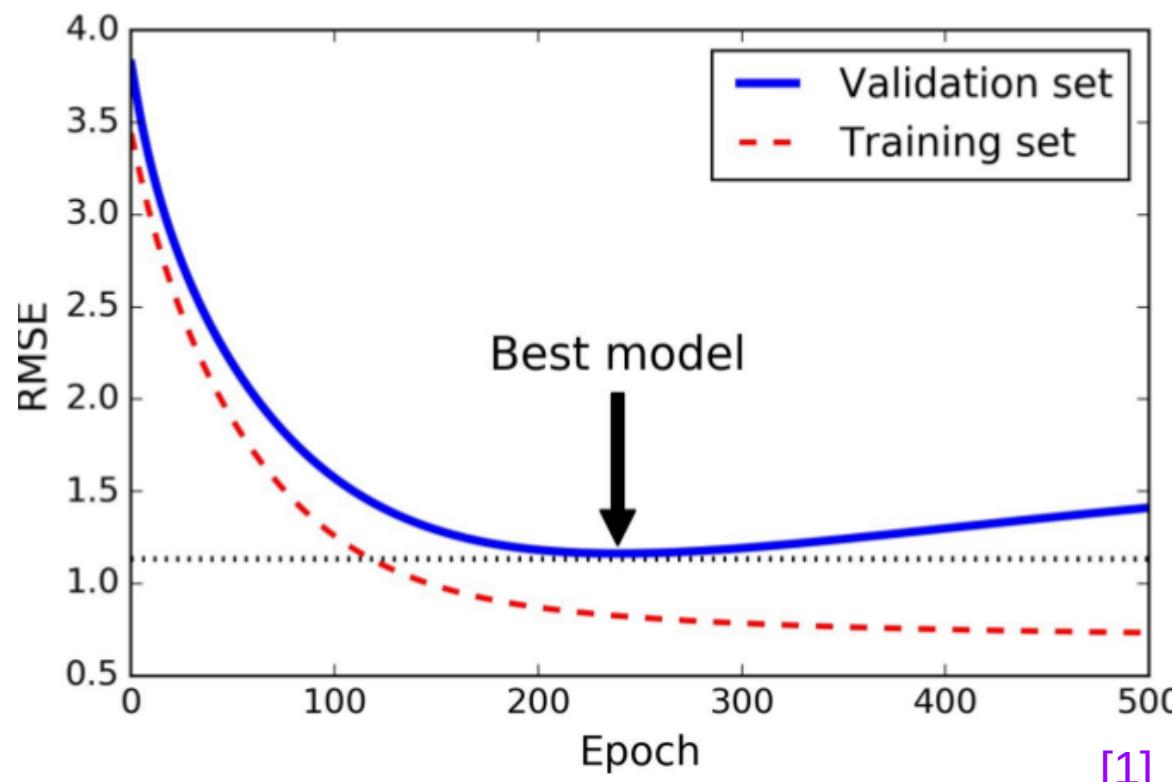


Figure 4-20. Early stopping regularization

[1]

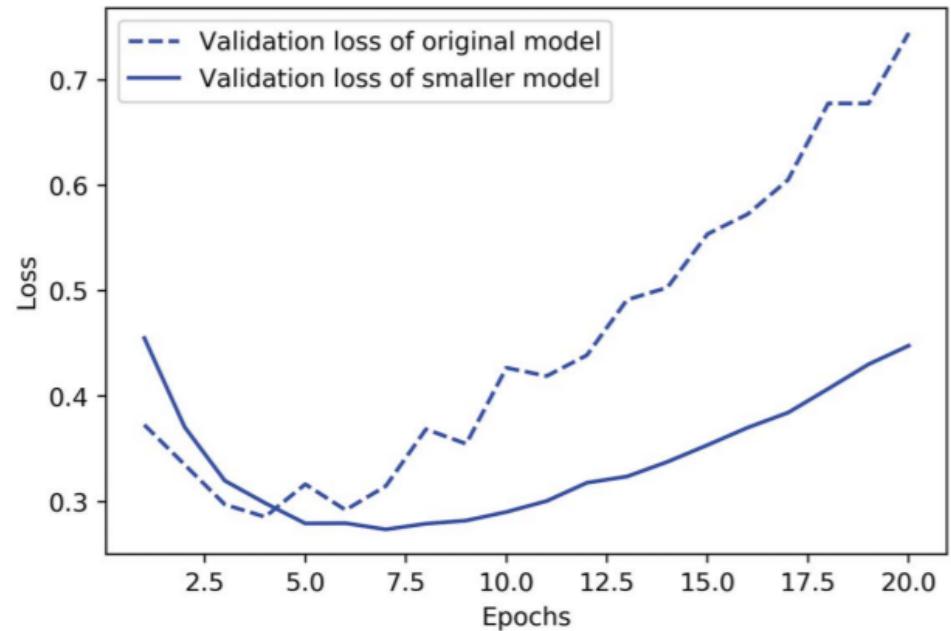
– Regularizando seu modelo

Técnicas de regularização são um conjunto de melhores práticas que ativamente impedem a capacidade do modelo de ajustar perfeitamente aos dados de treinamento; torna o modelo mais simples, mais “regular”, torna sua curva mais suave, mais genérica; torna o modelo mais capaz de generalizar.

Técnicas utilizadas:

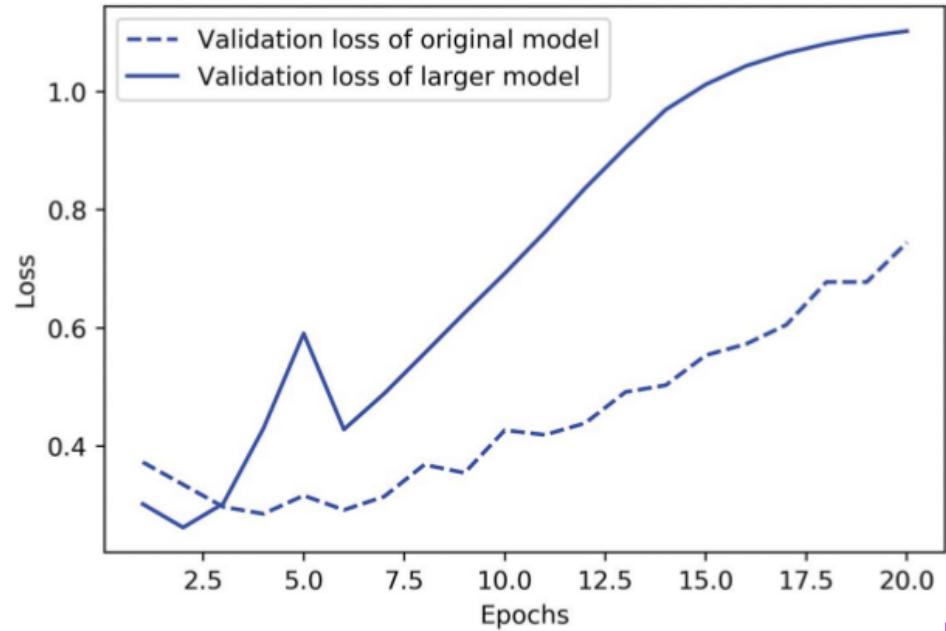
- Diminuição da capacidade do modelo;
- Regularização de pesos;
- Método *dropout*.

→ Reduzindo o tamanho da rede



[2]

Figure 5.17 Original model vs. smaller model on IMDB review classification



[2]

Figure 5.18 Original model vs. much larger model on IMDB review classification

– Adicionando regularização de pesos

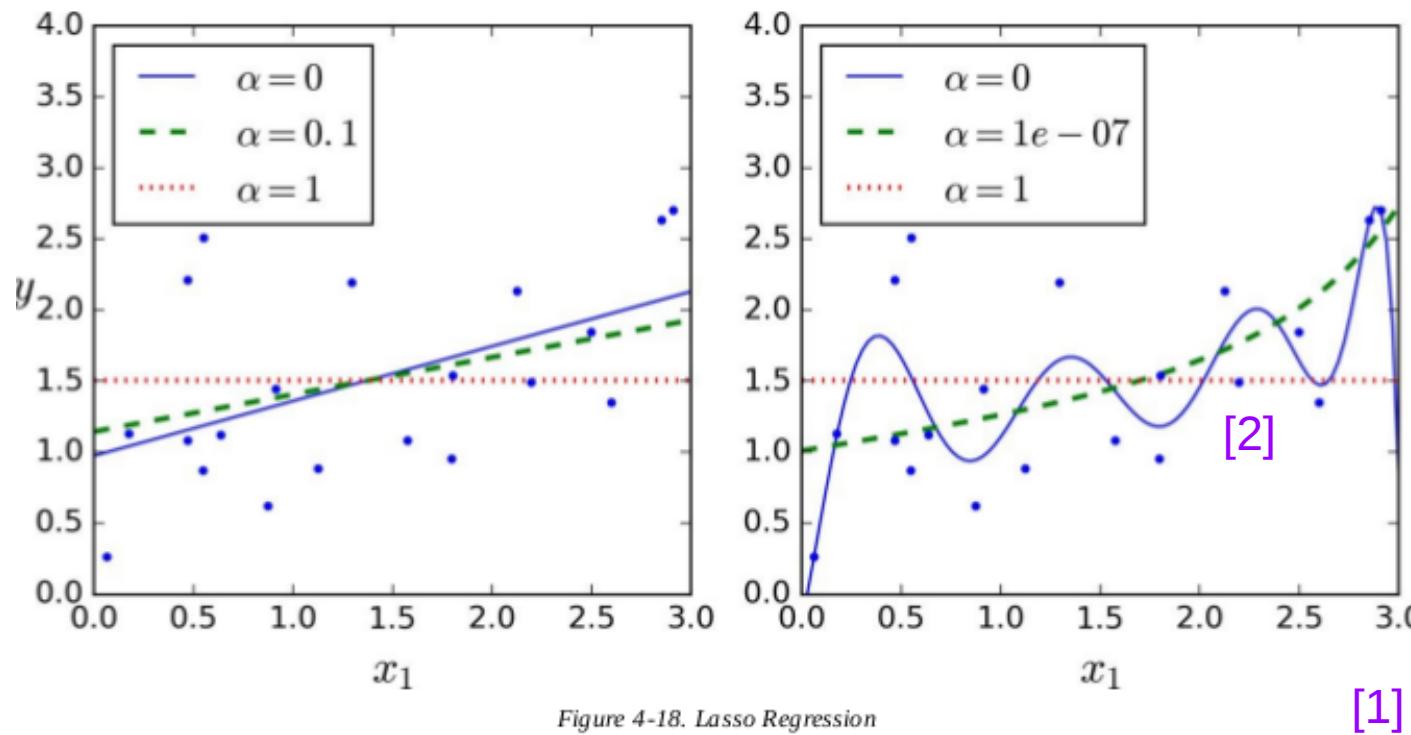
Uma forma comum de evitar o sobreajuste é colocar restrições à complexidade do modelo ao forçar seus pesos a terem apenas valores baixos, que torna a distribuição de valores de pesos mais regular. Isso é chamado de **regularização de pesos**, e é realizada ao adicionar à função de custo do modelo um custo associado a ter valores altos de pesos.

Temos dois tipos de regularização de pesos:

- Regularização l1, onde o custo adicionado é proporcional ao valor absoluto dos coeficientes dos pesos (a norma l1 dos pesos);
- Regularização l2 (ou decaimento de pesos): o custo adicionado é proporcional ao quadrado do valor dos coeficientes dos pesos (a norma l2 dos pesos).

→ Regularização L1 (Regressão lasso):

Função custo com a regularização L1: $J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|$



A regularização L1 tende a gerar um modelo esparso, ao zerar características pouco relevantes.

→ Regularização L2 (Regressão *ridge* ou de Tikhonov):

Função custo com a regularização L2: $J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$

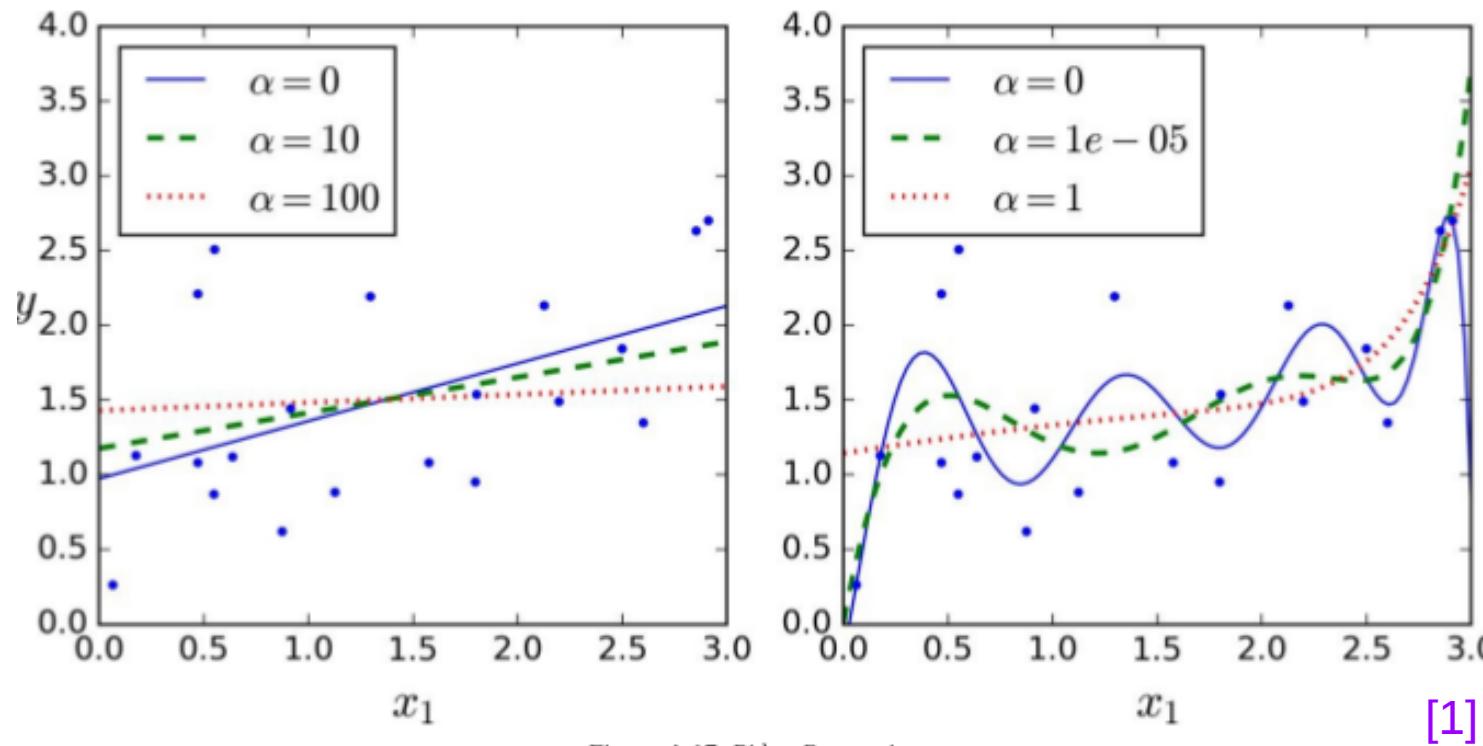


Figure 4-17. Ridge Regression

[1]

→ Regularização L1,L2 (Regularização *elastic net*):

Função custo com a regularização:
$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

Preferível: Ridge > elastic net > lasso; as duas últimas especialmente se apenas algumas características são úteis.

```

from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(16,
        kernel_regularizer=regularizers.l2(0.002),
        activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"])

history_l2_reg = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)

```

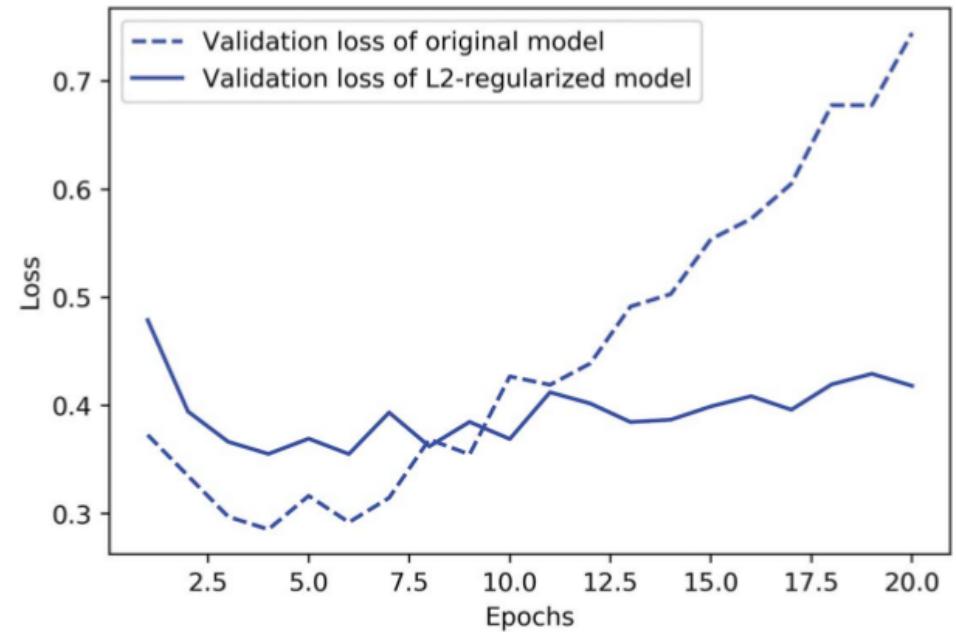


Figure 5.19 Effect of L2 weight regularization on validation loss [2]

Alternativas à regularização L2 no Keras:

```
from tensorflow.keras import regularizers
regularizers.l1(0.001)                         L1 regularization
regularizers.l1_l2(l1=0.001, l2=0.001)           Simultaneous L1 and
                                                L2 regularization
```

A regularização de pesos é tipicamente mais usada em redes menores. Redes maiores (profundas) tendem a ser tão sobreparametrizadas que impor restrições aos valores dos pesos não tem tanto impacto na capacidade do modelo e na generalização.

Em redes profundas, a técnica de *dropout* é preferida.

– Adicionando dropout

Dropout, aplicada a uma camada, consiste de aleatoriamente *dropping out* [descartar] um número de características da saída de uma camada durante o treinamento.

Proposta em 2012 por Geoffrey Hinton [<https://homl.info/64>], melhor detalhada em [<https://homl.info/65>].

Por exemplo, se uma camada normalmente retorna um vetor [0.2, 0.5, 1.3, 0.8, 1.1], depois de aplicar um *dropout* esse vetor terá alguns zeros distribuídos aleatoriamente: [0, 0.5, 1.3, 0, 1.1].

The diagram shows two 4x4 matrices. The left matrix represents the original activation matrix with values: 0.3, 0.2, 1.5, 0.0; 0.6, 0.1, 0.0, 0.3; 0.2, 1.9, 0.3, 1.2; and 0.7, 0.5, 1.0, 0.0. An arrow labeled "50% dropout" points to the right matrix. The right matrix shows the result after applying dropout, where 50% of the values have been set to zero: 0.0, 0.2, 1.5, 0.0; 0.6, 0.1, 0.0, 0.3; 0.0, 1.9, 0.3, 0.0; and 0.7, 0.0, 0.0, 0.0.

0.3	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.2	1.9	0.3	1.2
0.7	0.5	1.0	0.0

50% dropout →

0.0	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.0	1.9	0.3	0.0
0.7	0.0	0.0	0.0

* 2

Figure 5.20 Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time the activation matrix is unchanged.

[2]

A taxa de *dropout* é a fração de características que são zeradas; geralmente tem o valor de 0.2 e 0.5.

A cada etapa do treinamento, todos os neurônios (incluindo os de entrada, excluindo os de saída) têm uma probabilidade p de serem temporariamente “omitidos”; são ignoradas durante a etapa de treinamento, mas podem ser ativados na próxima etapa.

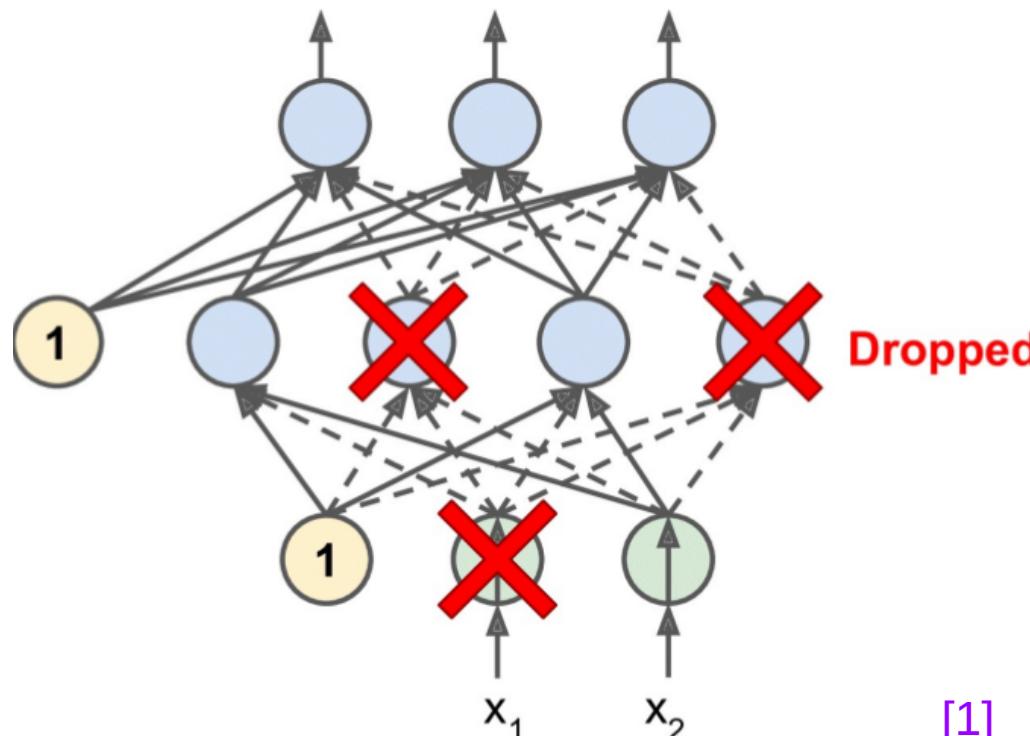


Figure 11-9. Dropout regularization

Taxa de *dropout* p : Entre 10% e 50%; 20-30% para RNNs, 40-50% CNNs.

No Keras, pode-se introduzir o *dropout* via uma camada Dropout:

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_dropout = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

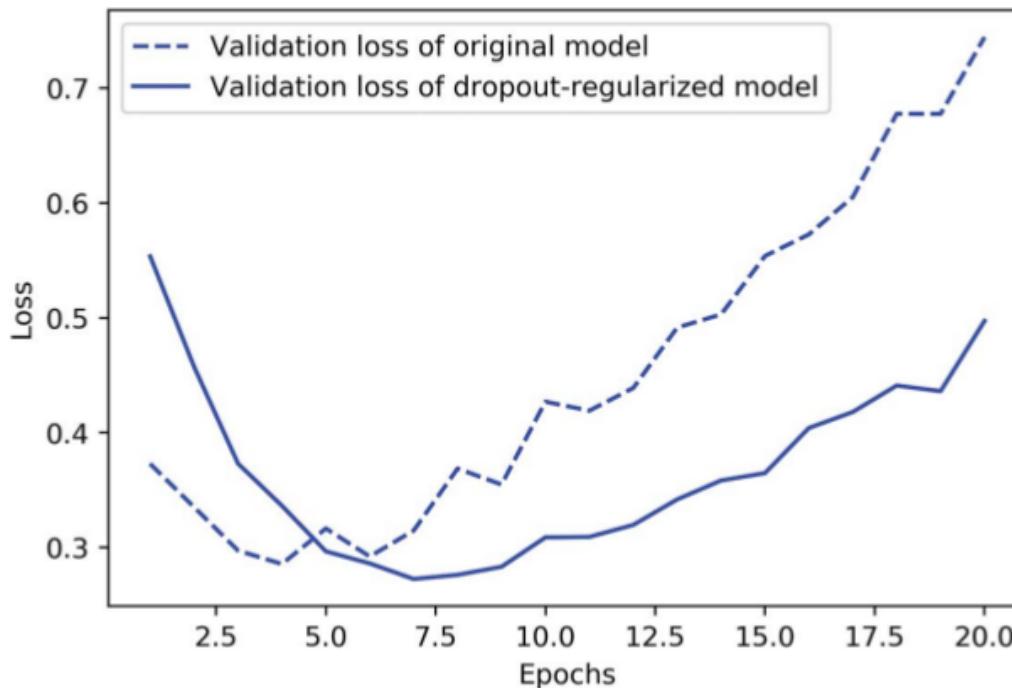


Figure 5.21 Effect of dropout on validation loss

Diretrizes práticas

Table 11-2. Default DNN configuration

[1]

Initialization	He initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

- Normalizar as características de entrada!
- Se precisar de um modelo esparso, usar a regularização l1;
- Caso precise de um modelo de baixa latência: Usar menos camadas, dobrar as camadas de normalização em batch e usar uma função de ativação rápida, como a leaky ELU ou ReLU.

Computer Scientists Prove Why Bigger Neural Networks Do Better

<https://www.quantamagazine.org/computer-scientists-prove-why-bigger-neural-networks-do-better-20220210/>

New Theory Cracks Open the Black Box of Deep Learning

<https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921/>