Deep Learning

Treinamento e Avaliação de Modelos – Deep Learning

Prof. Bruno Martins, Msc.

📊 Treinamento e Avaliação de Modelos 📈

Ao treinar modelos de Machine Learning, buscamos o equilíbrio perfeito entre aprendizado e generalização. Porém, é comum enfrentarmos desafios como Overfitting e Underfitting.

Vamos entender melhor esses conceitos:

🚩 **Overfitting (Sobreajuste)**

⚠️ **O que é:**

•Modelo "memoriza" demais os dados de treino.

•Excelente no treino, ruim na validação/teste.

🎯 **Exemplo:**

•Aluno que decora todas as respostas do livro, mas falha em questões inéditas.

📊 Treinamento e Avaliação de Modelos 📈

🔍 Como identificar:

Tipo	Precisão no treino	Precisão no teste
✅ Ideal	90%	88%
⚠️ Overfitting	98%	60%

📉 Gráfico ilustrativo:

```
Desempenho
|    . Overfitting
|    .
|   .      Ideal
|  .
|.
└──────────────── Dados
```

📊 Treinamento e Avaliação de Modelos 📈

🚩 **Underfitting (Subajuste)**

⚠️ **O que é:**

•Modelo simples demais, não aprende o suficiente.

•Baixo desempenho tanto no treino quanto no teste.

🎯 **Exemplo:**

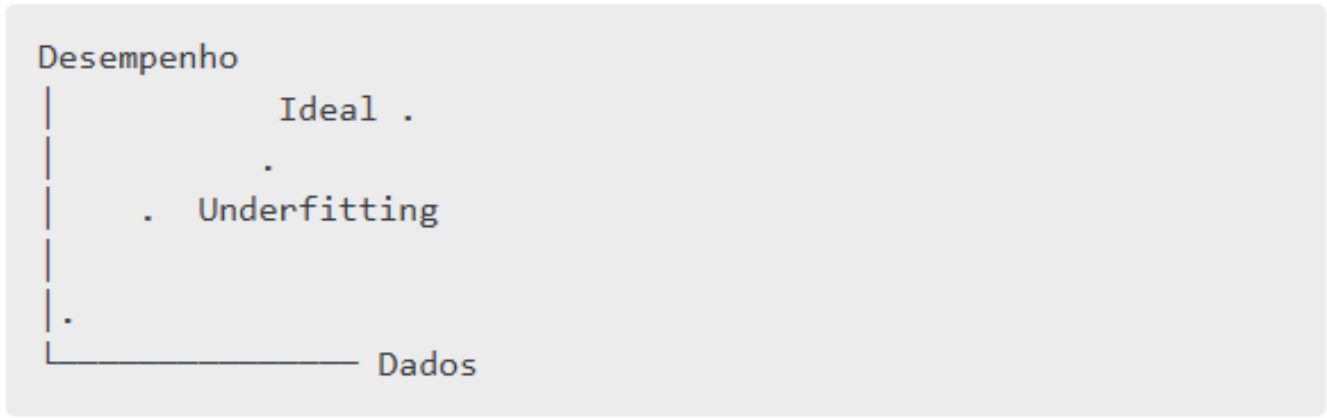
•Aluno que estuda apenas superficialmente, falhando em questões complexas.

📊 Treinamento e Avaliação de Modelos 📈

🔍 Como identificar:

Tipo	Precisão no treino	Precisão no teste
✅ Ideal	90%	88%
⚠️ Underfitting	65%	60%

📉 Gráfico ilustrativo:



📊 Treinamento e Avaliação de Modelos 📈

🎯 Estratégias para Melhorar o Desempenho

🚀 Reduzir Overfitting:

- 📌 Regularização (L1, L2)
- 📌 Cross-validation
- 📌 Mais dados 🗃️
- 📌 Early stopping ⏱️
- 📌 Dropout 🧩 (para redes neurais)
- 📌 Simplificação do modelo

🚀 Reduzir Underfitting:

- 📌 Modelos mais complexos 🔧
- 📌 Mais variáveis ou atributos 📋
- 📌 Ajuste de hiperparâmetros 🎚️
- 📌 Menos regularização 🛠️
- 📌 Melhorar dados de treinamento 📈

📊 Treinamento e Avaliação de Modelos 📈

🚀 **Reduzir Overfitting:**

- 📌 **Regularização (L1, L2)**:
Limita a complexidade do modelo penalizando pesos muito grandes, evitando que o modelo memorize os dados.
- 📌 **Cross-validation**:
Divide os dados em vários grupos, treina e testa múltiplas vezes. Ajuda a validar que o modelo realmente generaliza bem.
- 📌 **Mais dados** 📚:
Quanto mais dados, menos provável que o modelo memorize detalhes específicos. Melhora a generalização.
- 📌 **Early stopping** ⏱️:
Interrompe o treinamento antes do modelo começar a memorizar os dados, preservando melhor a capacidade de generalização.
- 📌 **Dropout** 🧩 **(para redes neurais)**:
Desativa aleatoriamente alguns neurônios durante o treino, evitando dependência excessiva em poucos neurônios específicos.
- 📌 **Simplificação do modelo**:
Reduz o número de parâmetros ou camadas, deixando o modelo menos propenso a memorizar.

📊 Treinamento e Avaliação de Modelos 📈

🚀 **Reduzir Underfitting:**

- 📌 **Modelos mais complexos** 🔧:
Usar algoritmos ou arquiteturas mais robustas capazes de capturar padrões mais sofisticados.
- 📌 **Mais variáveis ou atributos** 📋:
Adicionar novas informações que possam ajudar o modelo a entender melhor os padrões nos dados.
- 📌 **Ajuste de hiperparâmetros** 🎚️:
Ajustar parâmetros como taxa de aprendizado, número de camadas ou épocas para melhorar a capacidade de aprendizado.
- 📌 **Menos regularização** 🛠️:
Reduzir penalidades aplicadas ao modelo, permitindo que ele aprenda padrões mais complexos.
- 📌 **Melhorar dados de treinamento** 📈:
Garantir que os dados estejam corretos, limpos e representativos dos padrões que o modelo precisa aprender.

📊 Treinamento e Avaliação de Modelos 📈

📌 Resumo visual dos conceitos:

Conceito	Problema	Soluções práticas
🔴 Overfitting	Complexo demais 📉	Simplificação, Regularização, Dropout, Early stopping
▯ Underfitting	Simples demais 📉	Complexidade, Hiperparâmetros, Novos atributos

🎯 Dica final:

"Busque sempre um equilíbrio. Um modelo bom aprende padrões reais sem decorar detalhes irrelevantes!" 📌✨

📊 Treinamento e Avaliação de Modelos 📈

🚀 Primeiros passos com TensorFlow e PyTorch

🔧 Configuração e primeiros códigos:

- Instale as bibliotecas:

```
pip install tensorflow
pip install torch torchvision
```

- Confira se tudo está instalado corretamente:

```
import tensorflow as tf
print(tf.__version__)

import torch
print(torch.__version__)
```

📊 Treinamento e Avaliação de Modelos 📈

🖥️ **Conjunto de dados MNIST?**
O **MNIST** é um famoso conjunto de dados utilizado para testar algoritmos de aprendizado de máquina. É amplamente utilizado para aprendizado em Machine Learning e Redes Neurais.

📌 **Características principais:**
•**Conteúdo**: 70.000 imagens em tons de cinza (preto e branco) de dígitos escritos à mão (0 a 9).
•**Tamanho das imagens**: Cada imagem tem o tamanho padrão de 28x28 pixels.

•**Divisão**:
- 60.000 imagens para **treinamento**.
- 10.000 imagens para **teste** (validação do modelo).

📌 **Para que serve?** É muito utilizado para:

•Treinar redes neurais básicas.
•Ensinar conceitos de classificação.
•Avaliar desempenho de algoritmos.

📊 Treinamento e Avaliação de Modelos 📈

Normalização dos dados

normalização dos dados. Significa neste modelo que ela transforma os valores originais das imagens em valores entre 0 e 1

📌 **Por que dividir por 255?**

•As imagens originais do MNIST têm valores entre **0 e 255**, onde:

- **0** significa **preto total**.
- **255** significa **branco total**.
- Valores intermediários são tons de cinza.

Quando você divide por **255**, está transformando todos esses valores para uma escala entre **0 e 1**, facilitando o aprendizado da rede neural.

📊 Treinamento e Avaliação de Modelos 📈

Normalização dos dados

📌 Exemplo prático:

Se um pixel originalmente tem o valor `127` (cinza médio):

$$\frac{127}{255} \approx 0.498$$

Agora o valor será aproximadamente **0.498** na escala de 0 a 1.

📌 Benefícios da normalização:

- Acelera o processo de aprendizado.
- Melhora o desempenho e estabilidade da rede neural.
- Evita valores grandes que dificultam o treinamento.

📌 Por que fazer separadamente no treino e teste?

- Garantir que os dados usados para treino e teste estejam na mesma escala.
- Mantém coerência e evita erros na avaliação.

📊 Treinamento e Avaliação de Modelos 📈

Normalização dos dados

📌 Resumindo a lógica:

Antes (pixel original)	Depois da normalização
0 (preto total)	0.0
127 (cinza médio)	~0.498
255 (branco total)	1.0

✅ **Conclusão**:
Realizar a **normalização dos pixels das imagens** para uma escala entre 0 e 1, melhorando o desempenho da rede neural durante o treinamento neste modelo.

📊 Treinamento e Avaliação de Modelos 📈

📌 **Exemplo visual:**
Cada imagem do MNIST representa um dígito manuscrito:

```
Imagem (28x28 pixels)
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
...
```

Exemplo: um "7" escrito à mão.

📊 Treinamento e Avaliação de Modelos 📈

📚 **Resumindo**:

O MNIST é um conjunto de dados com milhares de imagens pequenas de números escritos à mão, utilizado amplamente para ensinar e praticar modelos simples de classificação de imagens em aprendizado de máquina.

📊 Treinamento e Avaliação de Modelos 📈

📌 **Atividade prática: Implementação de uma rede neural simples para classificação de dados**

•Utilize TensorFlow ou PyTorch para criar uma rede neural básica.
•Treine a rede utilizando um conjunto de dados simples (ex: MNIST).
•Avalie o desempenho e ajuste os parâmetros.
•Compare os resultados para verificar a generalização e identificar possíveis problemas (Overfitting ou Underfitting).

🎯 **Dica final:**
"Busque sempre um equilíbrio. Um modelo bom aprende padrões reais sem decorar detalhes irrelevantes!" 📌 ✨

Funções de ativação

🚩 1. Função de ativação ReLU (Rectified Linear Unit)

- **Fórmula**:

$$f(x) = \max(0, x)$$

- **Como funciona**:
  - Se o valor for **positivo**, mantém o valor.
  - Se o valor for **negativo**, substitui por **zero**.
- **Exemplo**:
  - ReLU(-5) = 0
  - ReLU(3) = 3
- **Uso**:
  - Usada principalmente em camadas ocultas.
  - Ajuda redes neurais a aprenderem rapidamente.

Funções de ativação

🚩 2. Função de ativação Softmax

- **Fórmula**:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- **Como funciona**:
  - Transforma valores em probabilidades (de 0 a 1), cuja soma total é sempre 1.
  - Muito usada na **última camada** em classificações com múltiplas classes.
- **Exemplo** (classe com 3 valores):
  - Entrada: [2.0, 1.0, 0.1]
  - Saída: [0.65, 0.24, 0.11] (probabilidades que somam 1).

Funções de ativação

🚩 3. Otimizador Adam

- **Adam (Adaptive Moment Estimation)**:
  - É um otimizador moderno que ajusta automaticamente a taxa de aprendizado durante o treino.
  - Combina vantagens de dois métodos populares: RMSProp e Momentum.
- **Como funciona**:
  - Ajusta parâmetros do modelo considerando a média e variância dos gradientes.
  - Permite treinos rápidos e estáveis.
- **Uso**:
  - Muito usado em redes neurais modernas devido à eficiência e facilidade.

Funções de ativação

🚩 4. Theta (θ)

- **O que é?**
  - "Theta" (θ) geralmente representa os parâmetros (pesos e vieses) em aprendizado de máquina.
  - Esses parâmetros são atualizados durante o treinamento.
- **Contexto**:
  - Durante o treino, a rede ajusta continuamente os valores de θ para reduzir os erros e melhorar a precisão do modelo.

Funções de ativação

🚩 5. Degrau (Step Function)

- **Fórmula básica**:

$$f(x) = \begin{cases} 0, & \text{se } x < 0 \\ 1, & \text{se } x \geq 0 \end{cases}$$

- **Como funciona**:
  - Retorna 0 se o valor for negativo.
  - Retorna 1 se o valor for zero ou positivo.
- **Uso**:
  - Muito utilizada nas primeiras redes neurais artificiais.
  - Hoje em dia menos usada, pois dificulta o aprendizado por não permitir pequenos ajustes (derivada zero quase em todo lugar).

Funções de ativação

📌 Resumindo visualmente:

Função	Tipo	Utilização principal
✅ ReLU	Não-linear	Camadas ocultas
✅ Softmax	Probabilidade	Última camada (classificação)
✅ Adam	Otimizador	Ajustar parâmetros automaticamente
✅ Theta (θ)	Parâmetros	Pesos e vieses
✅ Degrau	Binária (0 ou 1)	Primeiras redes neurais

Funções de ativação

🎚️ Taxa de Aprendizado (Learning Rate)

- Controla **quão rápido** o modelo ajusta os parâmetros (pesos).
- **Alta demais** pode causar instabilidade no treinamento.
- **Baixa demais** pode tornar o treinamento muito lento ou travar o aprendizado.

Exemplo prático:

- Taxa alta: `0.1` (rápido, mas pode instabilizar)
- Taxa moderada: `0.001` (mais estável, geralmente ideal)
- Taxa baixa: `0.00001` (lento, mas seguro)

Funções de ativação

⚙ Otimizadores

São algoritmos que ajustam automaticamente os parâmetros do modelo para reduzir erros.

- **Adam**: Ajusta automaticamente a taxa de aprendizado, combinando vantagens de RMSProp e Momentum (moderno e recomendado).
- **SGD (Stochastic Gradient Descent)**: Simples, tradicional, pode ser lento, mas funciona bem com ajustes adequados.
- **RMSProp**: Ajusta a taxa de aprendizado conforme o histórico de gradientes. Bom para problemas complexos.
- **Adamax**: Uma versão modificada do Adam que pode funcionar melhor em algumas situações específicas.

Funções de ativação

📦 Batch Size (Tamanho do Lote)

- Número de amostras (imagens, dados) usados em cada atualização do modelo.
- **Menor batch**: Mais atualizações, pode melhorar precisão, mas treinamento lento.
- **Maior batch**: Menos atualizações, mais rápido, porém pode diminuir precisão.

Exemplos comuns:

- `Batch size = 16`: Pequeno, mais atualizações.
- `Batch size = 64`: Moderado, equilíbrio entre velocidade e precisão.
- `Batch size = 256`: Grande, rápido, menos atualizações.

Funções de ativação

Hiperparâmetros vamos testar

📌 1. Número de neurônios na camada oculta:

- Original: `128`
- Sugestões: `32`, `64`, `256`, `512`
- Como testar:

```
Dense(128, activation='relu')
```

📌 2. Número de épocas (epochs):

- Original: `epochs=5`
- Sugestões: `epochs=3`, `epochs=10`, `epochs=20`
- Como testar:

  Altere o número de épocas na linha:

```
model.fit(x_train, y_train, epochs=5)
```

Funções de ativação

Hiperparâmetros testar

📌 1. Número de neurônios na camada oculta:

- Original: `128`
- Sugestões: `32`, `64`, `256`, `512`
- Como testar:

```
Dense(128, activation='relu')
```

📌 2. Número de épocas (epochs):

- Original: `epochs=5`
- Sugestões: `epochs=3`, `epochs=10`, `epochs=20`
- Como testar:

  Altere o número de épocas na linha:

```
model.fit(x_train, y_train, epochs=5)
```

Funções de ativação

Hiperparâmetros testar

📌 3. Taxa de aprendizado (learning rate):

- **Original**: Utilizando padrão do otimizador Adam.
- **Sugestões**: `0.1`, `0.01`, `0.001`, `0.0001`
- **Como testar**: Modifique o otimizador com taxas diferentes:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

📌 4. Otimizadores diferentes:

- **Original**: `optimizer='adam'`
- **Sugestões**: `'sgd'`, `'rmsprop'`, `'adamax'`
- **Como testar**:

  Altere o otimizador:

```
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Funções de ativação

Hiperparâmetros testar

📌 5. Batch size (tamanho do lote):

- **Original**: padrão (geralmente `32`)
- **Sugestões**: `16`, `64`, `128`, `256`
- **Como testar**: Modifique o tamanho do lote na função `fit`:

```
model.fit(x_train, y_train, epochs=5, batch_size=64)
```

Funções de ativação

Hiperparâmetros testar

🎯 **Como avaliar os testes?**

Criem uma tabela simples para registrar e comparar resultados como:

Teste	Neurônios	Épocas	Otimizador	Learning rate	Batch size	Acurácia Final (%)
1	128	5	Adam	0.001	32	97%
2	64	5	Adam	0.001	32	95%
3	128	10	SGD	0.01	64	93%

Funções de ativação

Hiperparâmetros testar

📌 O que observar?

- Como o número de neurônios afeta a **precisão** e o **tempo de treinamento**.
- Como o número de épocas influencia o **overfitting**.
- Qual otimizador apresenta melhores resultados.
- Como o tamanho do lote afeta a rapidez e desempenho.
- Como diferentes taxas de aprendizado influenciam o aprendizado.

📌 Discussão final:

•Qual combinação de hiperparâmetros foi mais eficiente e por quê.
•É importante experimentar para encontrar um equilíbrio ótimo entre precisão, rapidez e capacidade de generalização.