DEEP LEARNING

UNIDADE IV
CONSIDERAÇÕES FINAIS

Elaboração

Natasha Sophie Pereira

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

CAPÍTULO 1

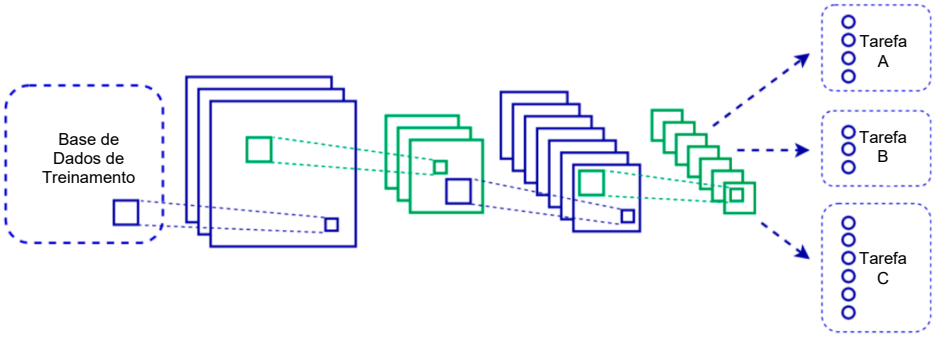
TRANSFERÊNCIA DE APRENDIZADO

Transferência de aprendizado é o processo de aplicar um modelo já treinado a um novo problema que seja correlacionado a este modelo. Por exemplo, no caso de imagens, é possível pegar uma rede já treinada utilizando a base de dados ImageNet e reaproveitá-la para a classificação de um tipo específico de imagens, como imagens de animais. É um conceito genérico que pode ser aplicado a qualquer modelo de aprendizado de máquina.

Para exemplificar, vamos utilizar uma rede já treinada com o conjunto de dados ImageNet (mas, reforçando, poderia ser um modelo treinado com qualquer conjunto de dados, de acordo com o problema a ser abordado). As bibliotecas TensorFlow, Keras, PyTorch, todas já possuem arquiteturas pré-treinadas na base de dados ImageNet.

Como já vimos anteriormente, a camada totalmente conectada em uma Rede Neural Convolucional atua como um tradutor entre a linguagem da rede (representações abstratas aprendidas durante o treinamento) e a linguagem desejada, ou seja, a classe de cada amostra. A transferência de aprendizado pode ser entendida como a tradução para outra linguagem desejada. Começamos com as características da rede, ou seja, a saída da última camada convolucional ou de agrupamento. Então, traduzimos estes valores a um novo conjunto de classe, relativas à nova tarefa. Isso pode ser feito ao remover a última camada totalmente conectada de uma rede pré-treinada e substituindo-a por outra camada, que representa as classes do novo problema. A Figura 29 mostra como a transferência de aprendizado funciona.

Figura 29. Transferência de Aprendizado.



Fonte: Adaptada de Vasilev *et al.*, 2019, p. 123.

Porém, não basta simplesmente substituir a última camada por outra, é necessário que o novo modelo adaptado seja treinado com dados referentes à nova tarefa. Este treinamento pode ser feito de duas formas:

- Usar parte da rede original como extrator de características e treinar apenas as novas camadas: é passado, à rede original, um lote de dados da nova tarefa, que passaram por ela a fim de verificar a saída da rede, da mesma forma que um treinamento normal. Porém, no passo para trás, os pesos da rede original são mantidos, e apenas os pesos das novas camadas serão atualizados. Esta forma é recomendada quando o novo conjunto de dados de treinamento é pequeno.
- Afinar o treinamento da rede inteira: neste caso, toda a rede será treinada, e não apenas as novas camadas. Ou seja, no passo para trás, todos os pesos serão atualizados. Em alguns casos, é possível manter os pesos das camadas iniciais, que são aquelas que detectam as características mais gerais dos dados. É recomentado o uso dessa forma quando o novo conjunto de treinamento é maior.

Exemplo de transferência de aprendizado utilizando PyTorch

Vamos seguir o exemplo apresentado por Vasilev *et al.* (2019), que aplicou uma rede pré-treinada com ImageNet em imagens CIFAR-10.

Inicialmente, vamos importar as bibliotecas:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import models, transforms
```

Em seguida, vamos definir o tamanho do lote (*batch_size*):

```
batch_size = 50
```

Para definir o conjunto de dados de treinamento, vamos levar em consideração que as imagens CIFAR-10 têm tamanho 32x32, enquanto as redes ImageNet necessitam de dados de tamanho 224x224. Então, vamos aumentar as imagens do CIFAR-10 de 32x32 para 224x224. É necessário padronizar os dados do CIFAR-10 utilizando a média e o desvio-padrão da ImageNet, e é necessário adicionar um pequeno aumento de dados (*flip*):

```
# Dados de Treinamento
train_data_transform = transforms.Compose([
  transforms.Resize(224),
  transforms.RandomHorizontalFlip(),
  transforms.RandomVerticalFlip(),
  transforms.ToTensor(),
  transforms.Normalize([0.485, 0.456, 0.406], [0.229,
    0.224, 0.225])
])

train_set = torchvision.datasets.CIFAR10(root='./data',
  train=True, download=True,
transform=train_data_transform)

train_loader = torch.utils.data.DataLoader(train_set,
  batch_size=batch_size, shuffle=True, num_workers=2)
```

Realizar os seguintes passos com os dados de validação e teste:

```
val_data_transform = transforms.Compose([
  transforms.Resize(224),
  transforms.ToTensor(),
  transforms.Normalize([0.485, 0.456, 0.406], [0.229,
  0.224, 0.225])
])

val_set = torchvision.datasets.CIFAR10(root='./data',
  train=False, download=True,
```

```
transform=val_data_transform)

val_order = torch.utils.data.DataLoader(val_set,
  batch_size=batch_size, shuffle=False, num_workers=2)
```

Escolher um dispositivo, preferencialmente GPU:

```
device = torch.device("cuda:0" if torch.cuda.is_available()
  else "cpu")
```

Definir o modelo de treinamento. Ao contrário do Keras, com PyTorch é necessário realizar as repetições manualmente sobre os dados de treinamento. Este método repete uma vez por todo o conjunto de treinamento (uma iteração) e aplica um otimizador após cada passo:

```
def train_model(model, loss_function, optimizer,
  data_loader):
  # Definindo o modelo para o modo de treinamento
  model.train()

  current_loss = 0.0
  current_acc = 0

  # Repete sobre o conjunto de treinamento
  for i, (inputs, labels) in enumerate(data_loader):
    # Envia a entrada ou os rótulos para a GPU
    inputs = inputs.to(device)
    labels = labels.to(device)

    # reinicia os parâmetros do gradiente
    optimizer.zero_grad()

    with torch.set_grad_enabled(True):
      # Passo adiante
      outputs = model(inputs)
      _, predictions = torch.max(outputs, 1)
      loss = loss_function(outputs, labels)
```

```
        # passo para trás
        loss.backward()
        optimizer.step()

    # Estatística
    current_loss += loss.item() * inputs.size(0)
    current_acc += torch.sum(predictions ==
      labels.data)

 total_loss = current_loss / len(data_loader.dataset)
 total_acc = current_acc.double() /
   len(data_loader.dataset)

 print('Train Loss: {:.4f}; Accuracy:
   {:.4f}'.format(total_loss, total_acc))


def test_model(model, loss_function, data_loader):

   # Definindo o modelo para o modo de avaliação
   model.eval()
   current_loss = 0.0
   current_acc = 0

    # Repete sobre o conjunto de validação
   for i, (inputs, labels) in enumerate(data_loader):
       # Envia a entrada ou os rótulos para a GPU
       inputs = inputs.to(device)
       labels = labels.to(device)

       # Passo adiante
       with torch.set_grad_enabled(False):
           outputs = model(inputs)
           _, predictions = torch.max(outputs, 1)
           loss = loss_function(outputs, labels)
```

```
            # Estatística
            current_loss += loss.item() * inputs.size(0)
            current_acc += torch.sum(predictions == labels.data)

        total_loss = current_loss / len(data_loader.dataset)
        total_acc = current_acc.double() / en(data_loader.dataset)
      print('Test Loss: {:.4f}; Accuracy: {:.4f}'.format(total_loss,
total_acc))
```

Definir o primeiro cenário de transferência de aprendizado, em que a rede pré-treinada é utilizada como extratora de características. Será utilizada uma rede popular, chamada ResNet-18, e o PyTorch fará o *download* dos pesos pré-treinados automaticamente. Vamos substituir a última camada da rede por uma nova camada com 10 classes de saída (uma para cada classe do CIFAR-10). Vamos eliminar as camadas da rede original do passo para trás, e atualizar os pesos apenas da camada adicionada utilizando o otimizador ADAM. Vamos rodar o treinamento e verificar a acurácia da rede a cada iteração:

```
def tl_feature_extractor(epochs=3):
  # Carregar o modelo pré-treinado
  model = torchvision.models.resnet18(pretrained=True)

  # Excluir os parâmetros existentes do passo para trás
  for param in model.parameters():
    param.requires_grad = False

  # As novas camadas require, por padrão,
  # requires_grad=True
  num_features = model.fc.in_features
  model.fc = nn.Linear(num_features, 10)

  # Transferindo para a GPU (se disponível)
  model = model.to(device)

  loss_function = nn.CrossEntropyLoss()

  # Otimizar apenas os parâmetros da última camada
```

```
    optimizer = optim.Adam(model.fc.parameters())

    # treinamento
    for epoch in range(epochs):
      print('Epoch {}/{}'.format(epoch + 1, epochs))

      train_model(model, loss_function, optimizer,
        train_loader)
      test_model(model, loss_function, val_order)
```

Definir o segundo cenário para refinamento da rede original. Similar ao cenário anterior, porém, toda a rede será treinada novamente:

```
def tl_fine_tuning(epochs=3):
    # Carregar o modelo pré-treinado
    model = models.resnet18(pretrained=True)

    # Substituir a última camada
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, 10)

    # Transferir o modelo para a GPU
    model = model.to(device)

    # Função de Perda
    loss_function = nn.CrossEntropyLoss()

    # Otimização de todos os parâmetros
    optimizer = optim.Adam(model.parameters())

    # treinamento
    for epoch in range(epochs):
      print('Epoch {}/{}'.format(epoch + 1, epochs))

      train_model(model, loss_function, optimizer,
        train_loader)
      test_model(model, loss_function, val_order)
```

É possível rodar a rede de acordo com o cenário escolhido. Para o primeiro cenário, chamar a função tl_feature_extractor(epochs=5) e para o segundo cenário, chamar a função tl_fine_tuning(epochs=5). Para o primeiro cenário, a acurácia obtida foi de 76%, enquanto, no segundo cenário, a acurácia foi de 87%.

CAPÍTULO 2

POSSIBILIDADES E LIMITAÇÕES

Deep Learning não é sinônimo de Inteligência Artificial ou de Aprendizado de Máquina. AI é um campo antigo e amplo que intenta automatizar o processo cognitivo, ou seja, automatizar o pensamento humano, seja através de implementação de funções lógicas matemáticas, até a construção de robôs que podem interagir com o mundo. Aprendizado de Máquina, por sua vez, é um subcampo da Inteligência Artificial, que visa à automatização do desenvolvimento de modelos a partir da análise de dados, processo conhecido como aprendizado. Finalmente, *Deep Learning* é uma das várias abordagens de ML, em que os modelos são várias cadeias de funções geométricas aplicadas aos dados uma após a outra. As funções em DL são organizadas em módulos, conhecidos como camadas, então, DL são pilhas de camadas parametrizadas por pesos que são os parâmetros aprendidos durante o treinamento. O conhecimento de um modelo é armazenado em seus pesos e o processo de aprendizagem, neste caso, consiste em encontrar os melhores valores para estes pesos.

Em poucos anos, *Deep learning* tem alcançado excelentes resultados na solução de tarefas consideradas muito complexas para os computadores, em especial no campo de "percepção" da máquina, ou seja, extração de informação de imagens, vídeos, sons, entre outros. Ao ser treinado com os parâmetros adequados, é possível que os algoritmos de DL consigam interpretar os dados de forma muito similar a um ser humano.

Atualmente, *Deep Learning* vem sendo alvo de muito interesse e investimento, visto que permite um reconhecimento de fala e a classificação de imagens em nível praticamente humano, permite o desenvolvimento de assistentes inteligentes em diversas áreas, melhorou sensivelmente o processo de tradução da máquina etc. De acordo com Chollet (2018), *Deep Learning* é uma revolução que está progredindo rapidamente, principalmente devido ao aumento de investimento e de pesquisadores na área, porém, apesar de as expectativas serem ótimas para essa tecnologia, abordar todo o seu potencial poderá levar vários anos.

Em *Deep Learning* tudo pode ser considerado como um ponto em um espaço geométrico. As entradas (texto, imagens etc.) e os alvos do modelo são vetorizados em espaços geométricos, e cada uma das camadas do modelo executa uma transformação geométrica simples nos dados, de modo que, juntas, o conjunto de camadas do modelo forma uma transformação geométrica complexa, que mapeia o vetor de entradas de acordo com o vetor de alvos. Essa transformação é parametrizada pelos pesos de cada camada, que é atualizado de acordo com a performance do modelo. A principal característica

dessa complexa transformação é que ela deve ser suave e contínua, para permitir que o algoritmo aprenda.

Em resumo, *Deep Learning* transforma os dados em vetores, e em espaços geométricos, e então, incrementalmente aprende as transformações geométricas complexas que mapeiam um espaço (entrada) de acordo com o outro (alvos). Para funcionar, é necessário que sejam fornecidos dados suficientes para capturar todas as características presentes nos dados originais.

Deep Learning não surgiu de repente, houve vários fatores que influenciaram essa tecnologia, alguns dos principais são:

- Inovações nos algoritmos incrementais.
- Disponibilização de uma grande quantidade de dados parametrizados.
- Disponibilização de *hardware* mais rápido, com processamento paralelo, a preços acessíveis.
- Grande quantidade de camadas de *software* e *frameworks* especializados, que tornam essa tecnologia mais acessível.

A sequência de passos para o desenvolvimento de uma boa arquitetura de Aprendizado de Máquina, e consequentemente, de *Deep Learning* é:

- Definir o Problema: quais dados estão disponíveis, o que você está tentando descobrir nestes dados.
- Definir uma forma de verificar, medir, sua taxa de sucesso: medir a acurácia de seu modelo de predição.
- Preparar o processo de validação para seu modelo: definir um conjunto de treinamento, um conjunto de validação e um conjunto de teste, de modo que os rótulos dos dados dos conjuntos de validação e teste sejam desconhecidos ao conjunto de treinamento, ou seja, as etapas de validação e teste devem ser posteriores ao treinamento.
- Vetorizar os dados e pré-processá-los, por normalização, por exemplo, de modo a torná-los mais adequados para a rede.
- Desenvolver um modelo que demonstre que o aprendizado de máquina poderá ser utilizado para solucionar seu problema.
- Refine a arquitetura de seu modelo ajustando os parâmetros e regularização. Faça mudanças baseadas na performance durante a fase de validação.

- Cuidado com o exagero de ajustes em seu modelo (*overfitting*), para que seu modelo não seja especializado apenas nos dados do conjunto de validação.

As três principais arquiteturas de rede são as redes densamente conectadas (DenseNet, do inglês, *densely connected networks*), as redes Convolucionais (CNN ou ConvNet, do inglês, *convolutional neural networks*) e as redes recorrentes (RNN, do inglês, *recurrent neural networks*), cada uma delas é projetada para um tipo de entrada de dados. Chollet (2018, p. 319) apresenta alguns exemplos de entradas de dados, e as redes que melhor se adequam a elas:

- Dados Vetoriais: redes densamente conectadas (camadas densas).
- Imagens: redes convolucionais 2D.
- Som: redes convolucionais 1D (preferencialmente) ou redes recorrentes.
- Texto: redes convolucionais 1D (preferencialmente) ou redes recorrentes.
- Dados de séries temporais: redes recorrentes (preferencialmente) ou redes convolucionais 1D.
- Outros tipos de dados sequenciais: redes recorrentes ou redes convolucionais 1D. Preferencialmente RNN se a sequência dos dados for uma questão relevante.
- Vídeo: redes convolucionais 3D (se há necessidade de capturar efeitos de movimento) ou uma combinação de ConvNet 2D para cada frame, para extração de características, com redes recorrentes ou ConvNet 1D para processar a sequência resultante.
- Dados volumétricos: redes convolucionais 3D.

Possibilidades

Chollet (2018, pp. 323-324) apresenta alguns exemplos de possibilidade de aplicação de Aprendizado de Máquinas, classificados de acordo com a forma de mapeamento das entradas para as saídas:

- Mapeamento de dados vetoriais para dados vetoriais
  - Predição de tratamentos médicos: mapear os registros médicos e predizer resultados.
  - Segmentação do comportamento: mapear um conjunto de atributos de um *website* e predizer quanto tempo um usuário gastará nele.

- Controle de qualidade de produtos: mapear um conjunto de características de um produto e predizer quanto tempo ele ficará no mercado.

» Mapeamento de imagens para dados vetoriais

- Assistentes de saúde: mapeamento de imagens médicas para predizer a presença de tumores.
- Veículos autônomos: mapear o ambiente onde o carro está inserido e comandar a direção.
- Jogos: mapear tabuleiros e predizer jogadas.
- Ajudante de dietas: mapear imagens de comida e predizer a quantidade de calorias.
- Predição de imagens: mapear imagens e prever a idade da pessoa.

» Mapeamento de dados temporais para dados vetoriais

- Previsão do tempo: mapear informações climáticas de uma região e predizer o tempo nos próximos dias.
- Interfaciar comandos do cérebro para o computador: mapear séries temporais do magnetoencefalograma (MEG) em comandos para o computador.
- Segmentação do comportamento: mapear a sequência de interações de uma pessoa em um *website* e prever a probabilidade de ela comprar algo.

» Mapear texto para texto

- Resposta Inteligente: mapear *e-mails* e sugerir respostas rápidas.
- Responder a perguntas: mapear respostas a questões de conhecimentos gerais.
- Resumir: mapear longos textos em textos mais curtos.

» Mapear imagens para texto

- Criar Legendas: mapear imagens e criar pequenas descrições de seu conteúdo.

» Mapear texto para imagens

- Geração de imagem: mapear uma pequena descrição para imagens que a descrevam.
- Geração ou seleção de logomarca: mapear o nome e descrição de uma empresa e criar uma logomarca.

- Mapear imagens para imagens
  - Super-resolução: mapear imagens de baixa qualidade para imagens de alta resolução.
  - Profundidade visual: mapear imagens em ambientes internos e predizer a profundidade dos elementos.
- Mapear imagem e texto para texto
  - Responder a Perguntas: mapear imagens e questões sobre o conteúdo da imagem para respostas textuais.
- Mapear vídeo e texto para texto
  - Responder a perguntas: mapear vídeos e questões sobre o conteúdo do vídeo para respostas textuais.

Limitações

Chollet (2018, p. 325-329) afirma que há uma infinidade de aplicações para as técnicas de *Deep Learning*, porém, alguns problemas não podem ser solucionados por essa arquitetura:

- Quando a solução de um problema requer raciocínio, planejamento prévio, manipulação algorítmica de dados.
- Uma das características dos seres humanos é a capacidade de projetar intenções, crenças e conhecimento em situações do dia a dia. A máquina não possui essa capacidade, o que pode levar à má interpretação dos dados.
- O ser humano pode antecipar acontecimentos ou sensações que nunca experimentou antes, visto que armazena experiências próprias e de outros, e as mescla com outras situações. A máquina simplesmente aprende a partir de um conjunto limitado de exemplos sobre determinada situação, podendo, simplesmente, prever desfechos para este cenário específico. Caso a arquitetura seja apresentada a um novo conjunto de exemplos, ela terá que aprender novamente.

Sendo assim, apesar dos grandes avanços da Inteligência Artificial e Aprendizado de Máquina, em especial *Deep Learning*, a máquina está muito longe de atingir um nível humano de pensamento, visto que ela não consegue raciocinar, nem abstrair conhecimento a partir de situações não vivenciadas.

REFERÊNCIAS

AREL, I.; ROSE, D. C.; KARNOWSKI, T. P. Deep machine learning – a new frontier in artificial intelligence research. **IEEE Computational intelligence magazine**, v. 5, n. 4, pp. 13-18, 2010.

BENGIO, Y. Learning deep architectures for AI. **Foundations and trends® in machine learning**, v. 2, n.1, p. 1-127, 2009.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. representation learning: a review and new perspectives. **IEEE Transactions on pattern analysis and machine intelligence**, v. 35, n.8, pp. 1.798-1.828, 2013.

BRYNJOLFSSON, E.; MITCHELL, T. M. What can machine learning do? Workforce Implications. **Science**, v. 358, n. 6.370, pp. 1.530-1.534, 2017.

BUDUMA, N. **Fundamentals of deep learning:** designing next-generation machine intelligence algorithms. Sebastopol, CA: O'Reilly Media, 2017.

CHO, K.; MERRIËNBOER, B. van; GULCEHRE, C.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *In*: Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar. **Anais**... Doha, Qatar: 2014.

CHOLLET, F. **Deep learning with python**. New York: Manning Publications, 2018.

DATA SCIENCE ACADEMY. **Deep learning book**. [s.l.] Data Science Academy, 2017.

DECHTER, R. Learning while searching in constraint-satisfaction-problems. *In*: Conference Proceedings of the Association for the Advancement of Artificial Intelligence 1986, **Anais**...1986.

DENG, L. An overview of deep-structured learning for information processing. *In*: Proceedings of Asian-Pacific Signal & Information Processing Annual Summit and Conference (APSIPA-ASC), Xi'an. **Anais**... Xi'an: 2011. Disponível em: https://www.microsoft.com/en-us/research/publication/an-overview-of-deep-structured-learning-for-information-processing/. Acesso em: 27/05/2019.

DENG, L. *et al*. Recent advances in deep learning for speech research at Microsoft. *In*: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing – Proceedings, Vancouver, BC, Canadá. **Anais**... Vancouver, BC, Canadá: 2013.

DENG, L.; YU, D. Deep Learning: Methods and Applications. **Foundations and trends® in signal processing**, v. 7, n. 3-4, pp. 197-387, 2013.

DENTON, E.; CHINTALA, S.; SZLAM, A.; FERGUS, R. Deep generative image models using a laplacian pyramid of adversarial networks. *In*: Neural Information Processing Systems (NIPS), Montreal, Canadá. **Anais**... Montreal, Canadá: 2015. Disponível em: http://arxiv.org/abs/1506.05751.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [s.l.] MIT Press, 2016.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial nets. In: Advances in Neural Information Processing Systems 27 (NIPS 2014), **Anais**...2014. Disponível em: https://papers.nips.cc/paper/5423-generative-adversarial-nets. Acesso em: 27/5/2019.

HINTON, G. E. Learning Multiple Layers of Representation. **Trends in cognitive sciences**, v. 11, n. 10, pp. 428-434, 2007.

HINTON, G. E. *et al*. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. **IEEE Signal Processing Magazine**, v. 29, n. 6, pp. 82-97, 2012.

HINTON, G. E.; OSINDERO, S.; TEH, Y. W. A fast learning algorithm for deep belief nets. **Neural computation**, v. 18, n. 7, pp. 1.527–1.554, 2006.

JAIN, S. **An Overview of Regularization Techniques in Deep Learning (with Python code)**. 2018. Disponível em: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/. Acesso em: 27/05/2019.

JOSHI, P. **Artificial intelligence with python:** build real-world artificial intelligence applications with python to intelligently interact with the world around you. Birmingham, UK: Packt Publishing, 2017.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Convolutional neural networks imagenet classification with deep convolutional neural network. **Communications of the Acm**, v. 60, n. 6, pp. 84-90, 2017.

LECUN, Y. **Generalization and network design strategiestechnical report CRG-TR-89-4**. [s.l: s.n.].

LEE, H.; GROSSE, R.; RANGANATH, R.; NG, A. Y. Unsupervised learning of hierarchical representations with convolutional deep belief networks. **Communications of the ACM**, v. 54, n. 10, pp. 95-103, 2011.

LIMA, I.; PINHEIRO, C. A. M.; SANTOS, F. A. O. **Inteligência artificial**. Rio de Janeiro: Elsevier, 2014.

MCCULLOCH, W. S.; PITTS, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, pp. 115-133, 1943.

MIRZA, M.; OSINDERO, S. **Conditional generative adversarial nets**. [s.l: s.n.]. Disponível em: http://arxiv.org/abs/1411.1784. Acesso em: 27/05/2019.

MITCHELL, T. M. **Machine learning**. [s.l: s.n.]

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of machine learning**. Cambridge, MA: The MIT Press, 2012.

OHLSSON, S. **Deep learning:** how the mind overrides experience. Cambridge: Cambridge University Press, 2011.

PATTERSON, J.; GIBSON, A. **Deep Learning:** A practitioner's approach. Sebastopol, CA: O'Reilly Media, 2017.

PÉREZ CASTAÑO, A. **Practical artificial intelligence:** machine learning, bots, and agent solutions using C#. New York, NY: Apress, 2018.

RADFORD, A.; METZ, L.; CHINTALA, S. **Unsupervised representation learning with deep convolutional generative adversarial networks**. [s.l: s.n.]. Disponível em: http://arxiv.org/abs/1511.06434. Acesso em: 27/05/2019.

RUSSELL, S.; NORVIG, P. **Artificial intelligence:** a modern approach. 3ª ed. New Jersey: Pearson Education, 2010.

SIMEONE, O. **A brief introduction to machine learning for engineers**. London, England: King's College London, 2018.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to Sequence Learning with Neural Networks. *In*: NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems, Montreal, Canadá. **Anais**... Montreal, Canadá: 2014. Disponível em: http://arxiv.org/abs/1409.3215. Acesso em: 27/05/2019.

THE ASIMOV INSTITUTE. **The neural network zoo**. Disponível em: http://www.asimovinstitute.org/neural-network-zoo/. Acesso em: 27/5/2019.

THRUN, S. *et al*. Stanley: The robot that won the DARPA grand challenge. **Journal of Field Robotics**, v. 23, n. 9, pp. 661-692, 2006.

TURING, A. M. Computing Machinery and Intelligence. **MIND**, v. 59, n. 236, pp. 433-460, 1950.

VASILEV, I.; SLATER, D.; SPACAGNA, G.; ROELANTS, P.; ZOCCA, V. **Python deep learning:** exploring deep learning techniques and neural network architectures with pytorch, keras and tensorflow. 2ª ed. [s.l.] Packt Publishing, 2019.

VINYALS, O.; TOSHEV, A.; BENGIO, S.; ERHAN, D. **Show and tell:** a neural image caption generator. *In*: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA. **Anais**... Boston, MA: 2015.

WERBOS, P. J. **Backpropagation Through time:** what it does and how to do it. *In*: Proceedings of IEEE, **Anais**...1990.

YU, D.; DENG, L. Deep Learning and its applications to signal and information processing. **IEEE Signal Processing Magazine**, v. 28, n. 1, pp. 145-150, 2011.