IA para Engenharia Ambiental

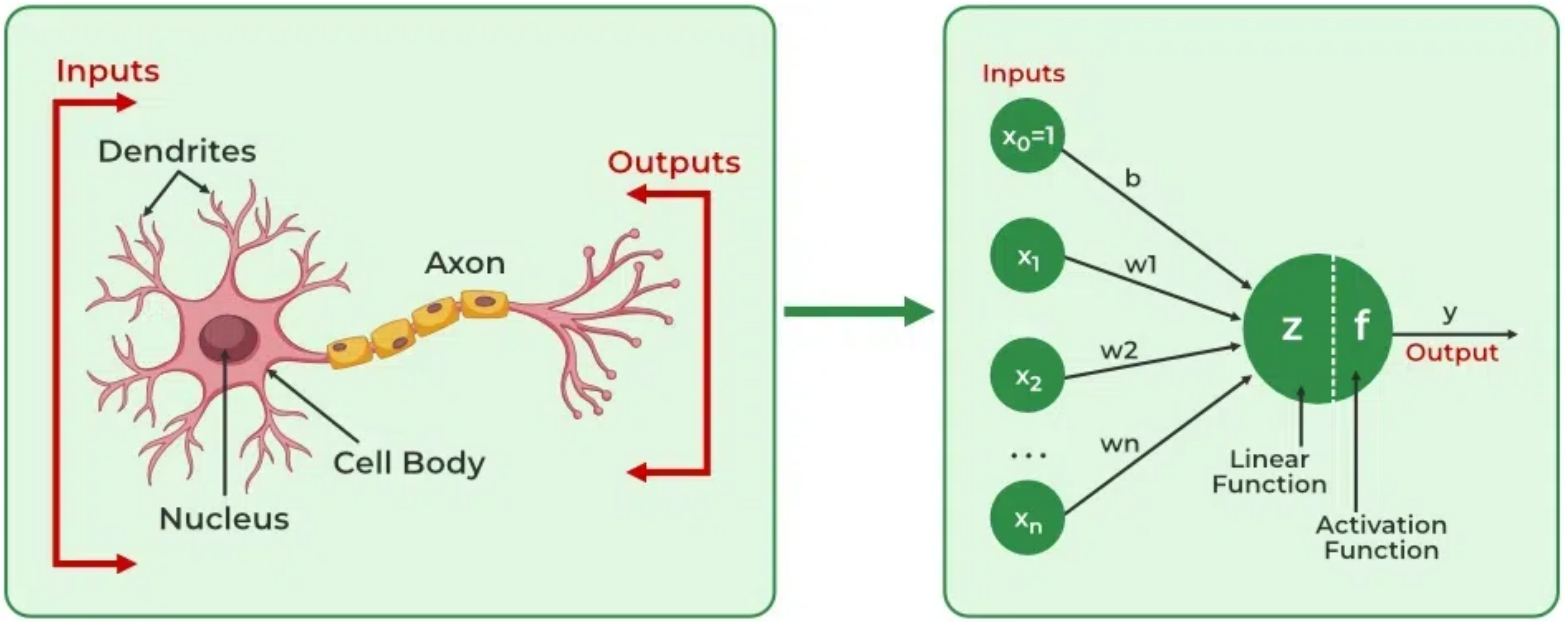
Redes Neurais Artificiais

Cesar de Oliveira F. Silva

Setembro / 2026

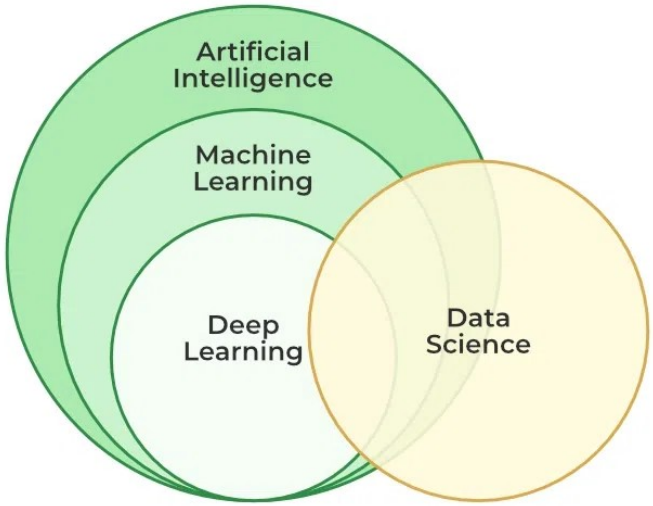
Roteiro

Parte 1: Bases para entender IA



IA, Aprendizado de Máquina e Deep Learning

- **Inteligência Artificial (IA)**: sistemas que executam tarefas que exigem inteligência humana.
- **Aprendizado de Máquina (ML)**: métodos estatísticos que aprendem a partir de dados.
- **Aprendizado Profundo (DL)**: subcampo de ML baseado em **redes neurais profundas**.
- **Ciência de Dados**: ciclo completo de coleta, preparação, modelagem e comunicação.



Subconjuntos: DL $\subset$ ML $\subset$ IA; interseção com Ciência de Dados.

Introdução e histórico

- Década de 1940: McCulloch & Pitts propõem um modelo lógico de neurônio artificial.
- Década de 1950–60: Perceptron (Rosenblatt) e regra de Hebb.
- Década de 1980: retropropagação do erro torna redes multicamadas treináveis em larga escala.
- Década de 2010 em diante: **redes profundas** impulsionadas por GPUs e grandes bases de dados.

Neurônio biológico: características gerais

- Unidade básica do sistema nervoso: célula especializada em receber, processar e transmitir informação.
- Sinais são transmitidos sob a forma de impulsos elétricos e químicos.
- Inspira diretamente o modelo de neurônio artificial usado em redes neurais.

Neurônio biológico: estrutura

- **Dendritos**: recebem sinais de outros neurônios.
- **Corpo celular (soma)**: integra os sinais recebidos.
- **Axônio**: conduz o impulso elétrico ao longo da célula.
- **Terminais sinápticos**: transmitem o sinal para outros neurônios por meio de sinapses.

Integração de sinais

O neurônio biológico soma excitações e inibições vindas dos dendritos e dispara um potencial de ação caso um limiar seja atingido.

Potencial de ação

- Em repouso, a membrana celular possui um potencial elétrico estável (ex.: -70 mV).
- Estímulos excitadores despolarizam a membrana; se o limiar é atingido, ocorre um **potencial de ação**.
- O potencial de ação é um fenômeno *tudo ou nada* e se propaga ao longo do axônio.

Analogia com o neurônio artificial

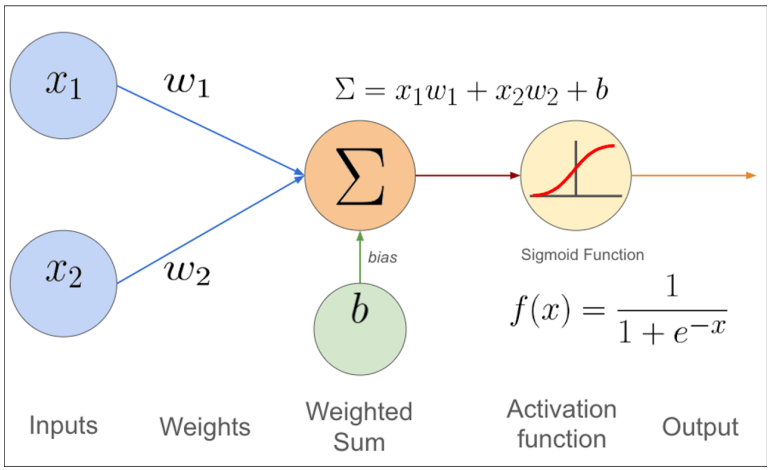
O limiar de disparo biológico inspira as funções de ativação e o limiar de decisão nos modelos artificiais.

Neurônio artificial

Um neurônio em uma rede neural artificial calcula uma soma ponderada das entradas e aplica uma função:

$$y = f\left(\sum_{i=1}^{d} w_i x_i + b\right) = f(\mathbf{w}^\top \mathbf{x} + b)$$

- $\mathbf{x} = (x_1, \ldots, x_d)$: saídas da camada anterior.
- $\mathbf{w} = (w_1, \ldots, w_d)$: pesos do neurônio.
- b: viés (intercepto).
- $f(\cdot)$: **função de ativação** (introduz não-linearidade).



Representação matemática de um neurônio.

Funções de ativação

Sem uma função de ativação não-linear, a rede inteira seria apenas uma transformação linear.

- **Sigmoide logística**

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  Saída em $(0, 1)$; usada em classificadores binários.

- **Tangente Hiperbólica**

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1$$

  Saída em $(-1, 1)$; centrada em zero.

- **ReLU** (*Rectified Linear Unit*)

$$\mathrm{ReLU}(z) = \max(0, z)$$

  Simples e eficiente em redes profundas.

Perceptron: descrição

- Modelo clássico de neurônio linear binário:

$$y = f(\mathbf{w}^\top \mathbf{x} + b), \quad f(\cdot) = \text{degrau}$$

- Usado para problemas de classificação linearmente separáveis.
- Limitação: não resolve problemas como XOR.

Princípios de Hebb e Regra do Perceptron

- **Regra de Hebb**: “neurônios que disparam juntos, conectam-se juntos”.

$$\Delta w_j = \eta \, x_j \, y$$

- **Regra de aprendizado do Perceptron**:

$$\Delta w_j = \eta \, (d - y) \, x_j$$

onde d é o rótulo desejado.

- As atualizações ocorrem apenas quando o perceptron erra.

Adaline e regra delta

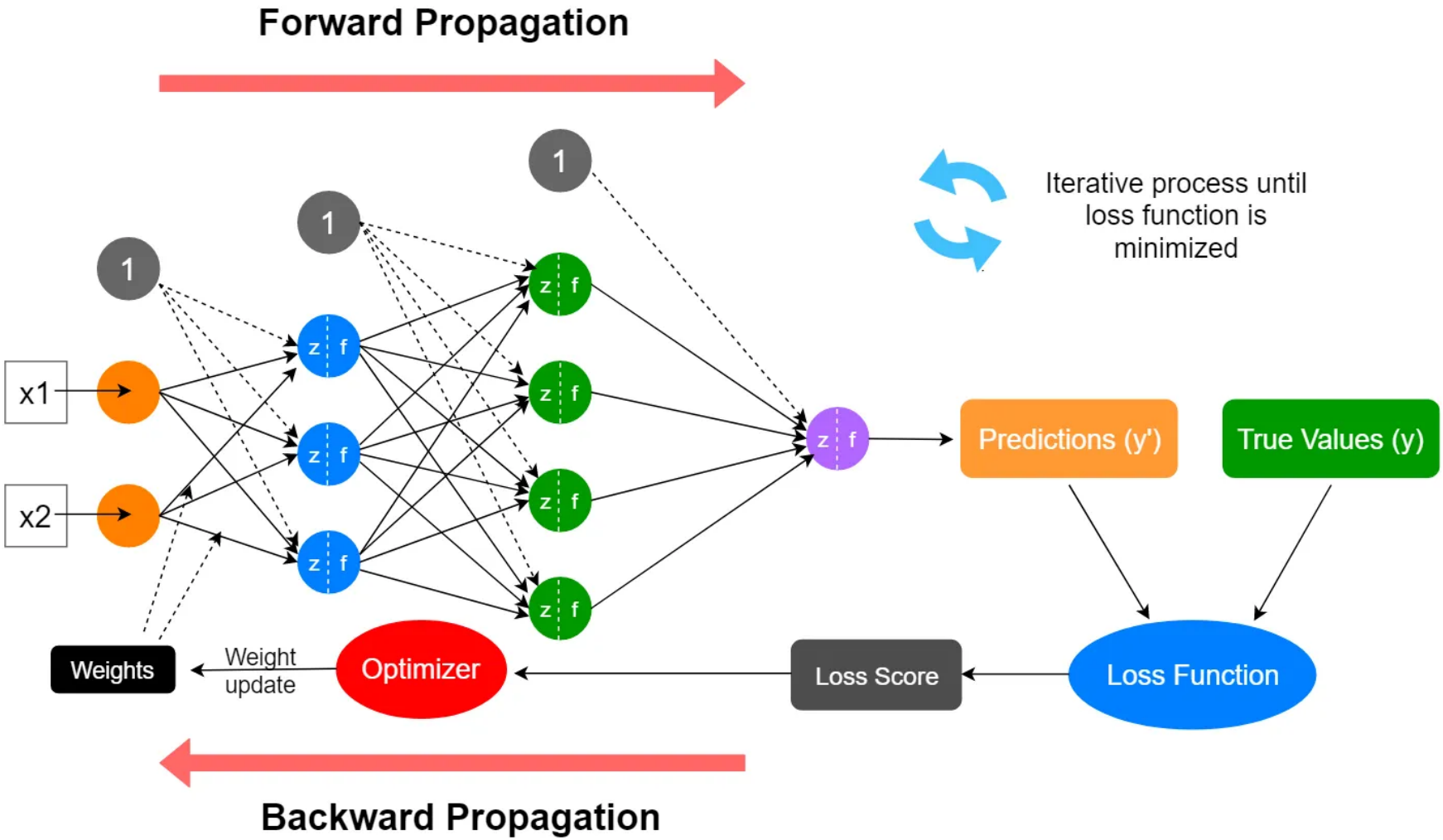
- **Adaline** (Adaptive Linear Neuron): saída linear (antes da função de passo).
- A função de perda é o erro quadrático:

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{N} (d_i - y_i)^2$$

- **Regra delta de Widrow** (gradiente descendente):

$$\Delta w_j = \eta \sum_{i=1}^{N} (d_i - y_i) x_{ij}$$

Parte 2: Como a Rede Neural Aprende?



Aspectos gerais de aprendizado

- Objetivo: ajustar os pesos $\{w_j\}$ para minimizar uma função de erro (ou maximizar desempenho).
- Em geral, utilizamos **gradiente descendente** ou variantes:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}(w)$$

  onde η é a taxa de aprendizado.
- O processo é iterativo, usando exemplos de treino.

Treinamento supervisionado

- Disponibilizamos pares entrada–saída $(\mathbf{x}, y)$.
- A função de perda típica em regressão: erro quadrático médio

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- Em classificação, usamos perda de entropia cruzada:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c} y_{ic} \log \hat{p}_{ic}$$

Treinamento não supervisionado

- Não temos rótulos *y*, apenas as entradas **x**.
- Objetivo: descobrir estrutura nos dados (clusters, projeções, padrões).
- Exemplos:
  - Autoencoders para redução de dimensionalidade.
  - Mapas auto-organizáveis (SOM) para visualização.

Treinamento com reforço

- O agente interage com o ambiente, recebendo **recompensas** r_t.
- Objetivo: aprender uma política que maximize a recompensa acumulada esperada:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

onde γ é o fator de desconto.

- Redes neurais podem aproximar funções de valor ou políticas em problemas ambientais dinâmicos.

Redes neurais multicamadas (MLP)

- Compostas por:
  - Camada de entrada
  - Uma ou mais camadas escondidas
  - Camada de saída
- Cada neurônio calcula:

$$y^{(\ell)} = f\left(W^{(\ell)} y^{(\ell-1)} + b^{(\ell)}\right)$$

- Permitem aproximar funções não-lineares complexas.

Retropropagação do erro

- Algoritmo padrão para treinar MLPs.
- Passos básicos:
  1. **Forward**: calcular saídas camada a camada.
  2. **Backward**: propagar o gradiente do erro a partir da saída.
  3. Atualizar pesos com gradiente descendente:

$$\Delta W^{(\ell)} = -\eta \frac{\partial \mathcal{L}}{\partial W^{(\ell)}}$$

- Usado com variantes como *mini-batch*, momentum, Adam etc.

Redes neurais profundas e arquiteturas

- **Redes profundas**: muitas camadas escondidas.
- Arquiteturas comuns:
  - **CNNs**: processamento de imagens (ex.: classificação de uso da terra).
  - **RNNs / LSTMs**: séries temporais ambientais (vazão, qualidade da água, poluentes).
  - **Autoencoders**: redução de dimensionalidade, detecção de anomalias.
- Exigem cuidado com overfitting (regularização, dropout, early stopping).

Rede neural como sequência de transformações

- Uma Rede Neural Profunda pode ser vista como uma **cadeia de transformações geométricas** em um espaço de alta dimensão.
- As **unidades** de uma camada definem a dimensionalidade dessa representação.
- Mais unidades $\Rightarrow$ maior capacidade de representar padrões complexos, porém maior custo computacional e maior risco de **overfitting**.

Fluxo de dados:

entrada $\rightarrow$ camadas $\rightarrow$ predições $\rightarrow$ função perda $\rightarrow$ otimizador.

Função de perda (Loss)

- Para treinar a rede, precisamos medir **o quão ruim** está a predição.
- A **função de perda** compara as saídas da rede com os valores verdadeiros.
- A meta do treinamento é minimizar essa perda.

Exemplos típicos:

- **Erro Quadrático Médio (MSE) –** regressão

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Crossentropy binária –** classificação 0/1

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- **Crossentropy categórica –** múltiplas classes.

O papel do Otimizador

- Dado um conjunto de pesos θ (vetor com todos os *w* e *b*), queremos minimizar $\mathcal{L}(\theta)$.
- O algoritmo mais usado é a **Descida por Gradiente Estocástico (SGD)**:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}_{\text{batch}}(\theta_t),$$

  onde η é a taxa de aprendizado.
- Variantes modernas: **Momentum**, RMSProp, Adam, etc.

Ciclo de treinamento

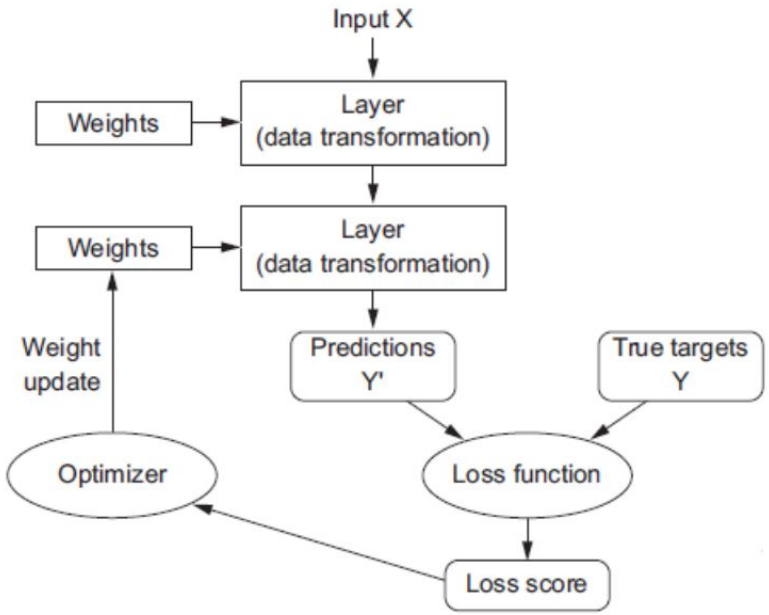
1. Propagação para frente (*forward*) e cálculo das predições;
2. Cálculo da perda;
3. *Backpropagation* para obter gradientes;
4. Atualização dos pesos pelo otimizador.

Ciclo de treinamento de uma rede neural

1. A rede recebe um **batch** de entradas **x**.
2. As camadas aplicam sucessivas transformações $\Rightarrow$ predições $\hat{\mathbf{y}}$.
3. A função de perda compara $\hat{\mathbf{y}}$ com os alvos **y**.
4. O **otimizador** calcula o gradiente da perda e atualiza os pesos:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta \mathcal{L}$$

   onde η é a taxa de aprendizado.
5. Repete-se por várias **épocas** até a perda parar de melhorar.



Fluxo geral de treino: entradas $\rightarrow$ camadas $\rightarrow$ perda $\rightarrow$ atualização.

Decisões de arquitetura

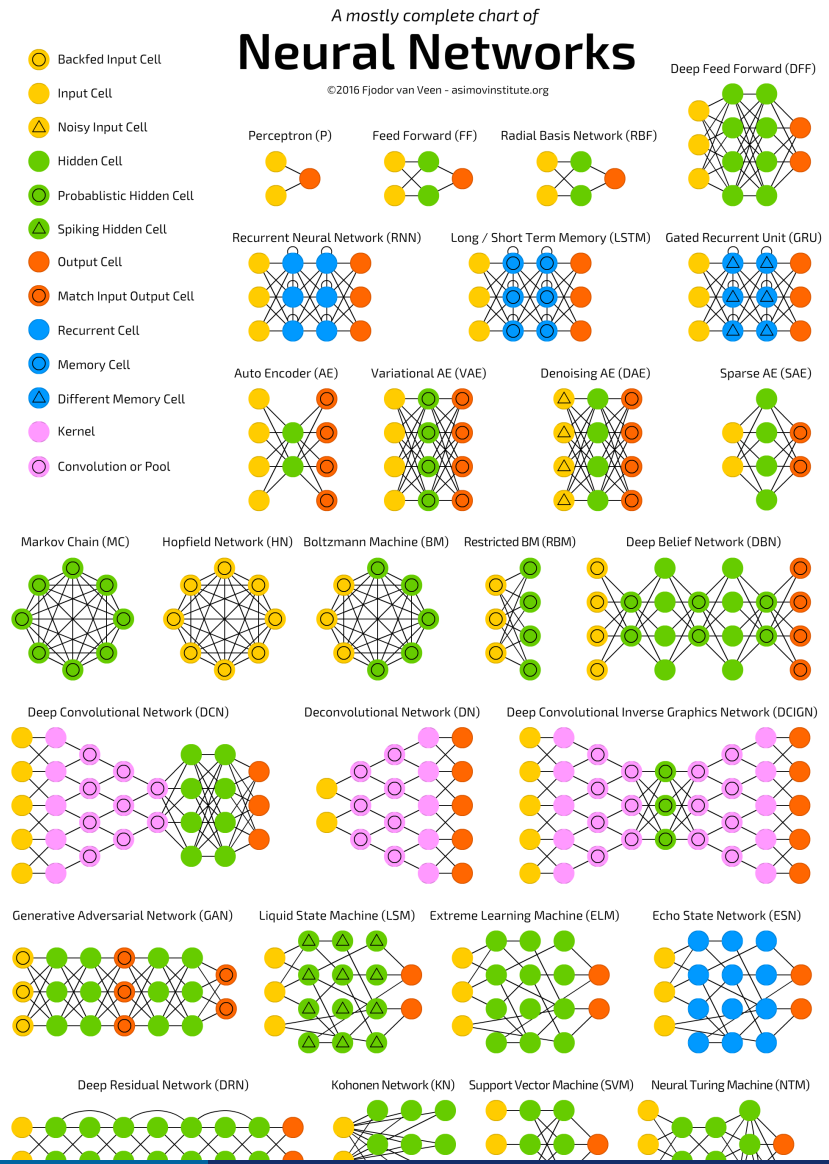
- **Número de camadas ocultas**
  - Mais camadas $\Rightarrow$ maior profundidade e capacidade de representação.
  - Profundidade excessiva sem dados suficientes pode levar a sobreajuste.
- **Número de unidades por camada**
  - Mais unidades $\Rightarrow$ mais liberdade para aprender padrões complexos.
  - Porém aumenta custo computacional e risco de sobreajuste.
- Regra prática: aumentar complexidade enquanto o erro em validação ainda melhora; usar regularização quando necessário.

Outros hiperparâmetros importantes

- **Momentum**
  - Usa o histórico dos gradientes para suavizar oscilações.
  - Valores típicos entre 0,5 e 0,9.
- **Número de épocas**
  - Quantas vezes o conjunto de treino completo é apresentado à rede.
  - Aumentar até a **loss de validação** começar a piorar (overfitting).
- **Tamanho do batch**
  - Número de amostras usadas em cada atualização de peso.
  - Valores comuns: 32, 64, 128, 256.

Zoo de Arquiteturas de Redes Neurais

- Há dezenas de arquiteturas: Perceptron, MLP, CNN, RNN, LSTM, Autoencoders, GANs, etc.
- Todas seguem a mesma ideia básica: **camadas** de neurônios conectadas por pesos.
- A escolha da arquitetura depende do tipo de dado (imagem, série temporal, texto, tabela) e da tarefa (classificar, regredir, gerar, comprimir).



O que é Aprendizado Profundo?

- Subcampo de ML que aprende **representações em múltiplas camadas**.
- Cada camada transforma o vetor de entrada em uma representação mais **abstrata e significativa**.
- A profundidade do modelo é o número de camadas que contribui para o mapeamento entrada $\rightarrow$ saída.
- Modelos modernos de DL podem ter dezenas ou centenas de camadas.

Camadas, Pesos e Conhecimento

- **Camada**: módulo de processamento que recebe um ou mais vetores e devolve vetores transformados.
- Cada camada possui **pesos** (e vieses) que definem a transformação aplicada nos dados.
- Durante o treinamento, esses pesos são ajustados (via **Descida por Gradiente Estocástico**) para maximizar o poder preditivo.
- O “conhecimento” de uma rede profunda é exatamente o conjunto de pesos e vieses de todas as camadas.

Medidas de desempenho em regressão

- Erro Quadrático Médio (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Raiz do MSE (RMSE):

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- Erro Absoluto Médio (MAE):

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Matriz de confusão e métricas de classificação

- Matriz de confusão binária:

	Pred. Positivo	Pred. Negativo
Real Positivo	TP	FN
Real Negativo	FP	TN

- Acurácia: $\frac{TP + TN}{TP + TN + FP + FN}$
- Precisão: $\frac{TP}{TP + FP}$
- Recall (sensibilidade): $\frac{TP}{TP + FN}$
- F1-score: média harmônica entre precisão e recall.

Validação cruzada

- Objetivo: estimar desempenho em dados não vistos.
- **k-fold CV**: divide os dados em k partes, treina em $k - 1$ e testa na parte restante, repetindo k vezes.
- Média e desvio dos indicadores ao longo dos folds dão uma visão da variabilidade do modelo.
- Em dados espaciais e temporais, usar variantes que respeitem dependências (ex.: *blocked CV*).

Resumo

- Aprendizado Profundo aprende **representações em múltiplas camadas**.
- Redes neurais artificiais são composições de neurônios:

$$y = f(w^\top x + b),$$

  com funções de ativação não lineares.
- O treinamento ajusta pesos para minimizar uma **função perda** via métodos de otimização como SGD e Adam.
- Decisões de arquitetura e hiperparâmetros têm impacto direto em capacidade, custo e generalização.

Construindo uma rede em Keras (visão conceitual)

- No Keras, um **modelo sequencial** é uma pilha de camadas.
- Exemplo: rede totalmente conectada para 5 classes de uso da terra:
  - Camada de entrada (pixels).
  - 4 camadas ocultas densas com, por exemplo, 200 neurônios cada.
  - Camada de saída **softmax** com 5 unidades (probabilidades).
- As camadas extraem representações progressivamente mais úteis para a tarefa.

Etapas: compilar e treinar o modelo

Compilar o modelo: definir como ele vai aprender.

- **Função perda** (p. ex. crossentropy categórica).
- **Otimizador** (SGD, Adam, etc.).
- **Métricas** de acompanhamento (acurácia, F1, etc.).

Ajustar o modelo aos dados (método `fit()`):

- `epochs`: número de passagens completas sobre o conjunto de treino.
- `batch_size`: tamanho dos lotes usados em cada atualização.
- `validation_data`: conjunto de validação para monitorar overfitting.

Ao final, obtemos uma rede treinada, isto é, um conjunto de pesos que minimiza a perda e produz predições próximas dos valores reais.