Introdução a Machine Learning

1) Segundo a discussão feita em aula sobre *software* com e sem *Machine Learning*, em qual das situações abaixo *Machine Learning* é uma abordagem mais adequada do que escrever uma solução programada explicitamente?

   (a) Problemas em que as regras que transformam a entrada na saída são conhecidas e podem ser enumeradas como uma sequência finita de instruções.

   (b) Problemas em que a solução exata tem custo computacional baixo, mas a equipe deseja reduzir o tempo de desenvolvimento do sistema.

   (c) Problemas em que não sabemos escrever diretamente um algoritmo para a tarefa, mas dispomos de exemplos nos quais é possível encontrar padrões.

   (d) Problemas em que a saída esperada é determinística e uma mesma entrada deve produzir sempre exatamente o mesmo resultado.

2) A tabela abaixo apresenta o erro médio de quatro modelos no conjunto de treino e em um conjunto de teste formado apenas por dados novos, que não foram utilizados durante o treinamento. Considerando que o objetivo é obter o menor erro esperado para novos dados, qual modelo deve ser escolhido?

Modelo	Erro no treino	Erro no teste
A	12.0	12.4
B	5.8	6.3
C	1.5	13.9
D	0.2	19.7

   (a) Modelo A.

   (b) Modelo B.

   (c) Modelo C.

   (d) Modelo D.

3) Você precisa treinar uma Regressão Linear em um conjunto de dados com um número muito grande de atributos (M na ordem de centenas de milhares). Sobre a escolha entre a equação normal $\boldsymbol{\theta} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$ e a descida de gradiente nesse cenário, é correto afirmar que:

   (a) A equação normal é preferível, pois encontra exatamente o melhor $\boldsymbol{\theta}$ sem depender da escolha de uma taxa de aprendizado.

   (b) A descida de gradiente é preferível, pois a função de custo J_{SSR} deixa de ser convexa quando o número de atributos é grande.

   (c) A equação normal é preferível, pois seu custo cresce linearmente com o número de atributos, enquanto o da descida de gradiente cresce cubicamente.

   (d) A descida de gradiente é preferível, pois o custo de inverter a matriz $\mathbf{X}^\top \mathbf{X}$ cresce cubicamente com o número de atributos.

4) Ao adaptar a Regressão Linear para a tarefa de classificação binária obtemos a Regressão Logística, $\hat{y}^{(i)} = \frac{1}{1+e^{-\mathbf{x}^{(i)}\boldsymbol{\theta}}}$. Nessa adaptação, a função de custo J_{SSR} é substituída pela J_{BCE}. Qual alternativa apresenta a justificativa correta para essa substituição?

(a) A função J_{SSR} deixa de ser convexa para a Regressão Logística, e a otimização pode ficar presa em mínimos que não são o global.

(b) A função J_{SSR} deixa de ser diferenciável quando composta com a função sigmoide, e o gradiente fica indefinido em parte do domínio.

(c) A função J_{BCE} admite solução analítica semelhante à equação normal, e o treinamento passa a dispensar métodos iterativos.

(d) A função J_{BCE} torna a fronteira de decisão não linear, e o modelo passa a resolver problemas que não são linearmente separáveis.

5) Dentre as alternativas abaixo, qual melhor descreve o processo de *hyperparameter tuning*?

(a) É um processo de otimização de parâmetros da rede neural que é feito via decida de gradiente.

(b) É um processo de busca cujo objetivo é encontrar valores ideais para parâmetros que não são aprendidos durante o processo de treino.

(c) É o processo de busca pela hipótese que melhor descreve a relação entre um conjunto de atributos e a variável alvo conforme uma função loss.

(d) É o processo equivalente a regularização, que visa limitar a complexidade do modelo para melhorar a sua capacidade de generalizar.

6) No exemplo numérico do efeito tesoura (*Scissor Effect*) visto em aula, um polinômio de grau 4 passa exatamente pelos cinco pontos do conjunto de treino, obtendo erro de treino igual a zero, enquanto a reta $\hat{y} = 2x + 0.1$ apresenta erro de treino maior do que zero. No entanto, no ponto de teste $(6, 12)$ a reta prediz 12.1 e o polinômio prediz 27.5. Explique com suas próprias palavras a diferença entre ajustar bem o conjunto de treino e generalizar para novos dados. Em seguida, relacione o comportamento dos dois modelos com os termos da decomposição do erro de teste (viés, variância e ruído) e com o efeito tesoura: em qual regime de quantidade de dados a hipótese mais simples tende a ser a melhor escolha?

Perceptron

7) Por que o *Perceptron* como proposto originalmente ($\hat{y}^{(i)} = sign(\mathbf{X}^{(i)}\boldsymbol{\theta})$) não pode ser organizado em uma rede para formar um *Multi Layer Perceptron*?

(a) Pois a derivada da função de ativação em relação aos pesos ou é igual a zero ou indefinida.

(b) Pois o *dot product* dos pesos com as *features* não pode ser derivado.

(c) Pois **apenas** a descontinuidade na função de ativação $sign(x)$ em $x = 0$ impede o processo de otimização via descida de gradiente.

(d) Pois o perceptron possui a limitação de modelar apenas problemas que são linearmente separáveis.

8) Por que uma rede de Perceptrons de Rosenblatt não pode ser treinada via descida de gradiente e *backpropagation*?

9) Prove que um Perceptron com duas entradas $\hat{y} = sinal(\theta_0 + X_1\theta_1 + X_2\theta_2)$ gera uma fronteira de decisão linear.

10) Crie um Perceptron que se comporta como uma função *and*.

MLP e Backpropagation

11) Em uma unidade (neurônio) de uma rede neural artificial, qual a operação realizada entre as entradas e os pesos durante o *forward pass*?

(a) Produto Escalar (*Dot Product*).

(b) Função de Ativação (*Activation Function*).

(c) Retropropagação de Erros (*Backpropagation*).

(d) Descida de Gradiente (*Gradient Descent*).

12) Em uma rede perceptron multicamadas (*Multi Layer Perceptron*). Qual das alternativas abaixo melhor descreve o *Forward Pass* e o *Backward Pass* respectivamente?

(a) Computar as saídas de cada uma das unidades e as entradas.

(b) Computar as saídas de cada uma das unidades e o gradiente em relação aos pesos.

(c) Computar as entradas de cada uma das unidades e as saídas.

(d) Realizar a Retropropagação de Erros (*Backpropagation*) e a Descida de Gradiente (*Gradient Descent*).

(e) Realizar a Descida de Gradiente (*Gradient Descent*) e a Retropropagação de Erros (*Backpropagation*).

13) Considere um MLP com uma unidade na entrada, uma unidade na saída, K camadas ocultas e D unidades em cada camada. Quantos parâmetros esse MLP possui?

14) Quais são os quatro passos realizados para fazer o treinamento de um MLP?

15) Considerando a rede da Figura 1, resolva os itens abaixo:

$$\theta^{(1)} = \begin{bmatrix} 0.5 & -0.2 \\ 0.1 & 0.2 \end{bmatrix} \quad \theta^{(2)} = \begin{bmatrix} 0.2 & -0.1 \end{bmatrix}$$

$$\beta^{(1)} = \begin{bmatrix} 0.2 & -0.1 \end{bmatrix} \quad \beta^{(2)} = \begin{bmatrix} 0.4 \end{bmatrix}$$

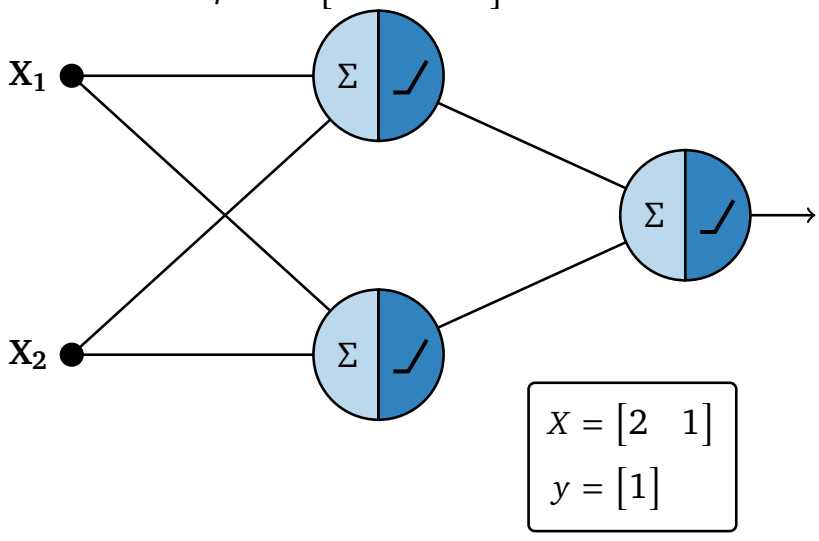


Figura 1: Multilayer Perceptron

a) Compute o *Forward Pass*.

b) Compute o valor da Loss $J = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$.

c) Compute o *Backward Pass*.

d) Realize a atualização de todos os parâmetros treináveis da rede conforme o otimizador SGD $\theta_{t+1} = \theta_t - \eta\nabla J$.

Softmax, Entropia Cruzada e Grafo Computacional

16) Em um grafo computacional, o *backward pass* percorre o grafo na ordem inversa aplicando a regra da cadeia localmente em cada nodo. Para calcular o gradiente que envia às suas arestas de entrada, o que cada nodo precisa conhecer?

(a) O gradiente *upstream* que chega pela sua saída e a derivada local da sua operação em relação a cada entrada.

(b) A expressão simbólica completa da função representada pelo grafo e a derivada algébrica dela em relação a cada folha.

(c) O valor da *loss* avaliada duas vezes, com cada entrada perturbada por um passo h, para estimar a derivada por diferença finita.

(d) O gradiente da *loss* em relação a todos os parâmetros da rede, obtido antes do início do percurso na ordem inversa.

17) Considere o grafo computacional da função $f(a, b, c) = (a + b) \cdot (b \cdot c)$, com nodos intermediários $u = a + b$ e $v = b \cdot c$. Note que a variável b alimenta dois nodos do grafo. Para $a = 1$, $b = 2$ e $c = 3$, após o *forward pass* e o *backward pass* iniciado com $\partial f/\partial f = 1$, qual o valor de $\partial f/\partial b$?

(a) 6

(b) 9

(c) 15

(d) 18

18) Considere a função $f(a, b, c) = (a + b) \cdot \max(b, c)$, decomposta em um grafo computacional com três nodos de operação: $u = a + b$ (nodo de soma), $v = \max(b, c)$ (nodo de máximo) e $f = u \cdot v$ (nodo de produto). Repare que a variável b alimenta dois nodos. Para $a = 2$, $b = 3$ e $c = 4$, resolva os itens abaixo:

a) Compute o *forward pass*, informando o valor de cada nodo intermediário e da saída f.

b) Compute o *backward pass* nodo a nodo, iniciando com $\partial f/\partial f = 1$, e informe os gradientes $\partial f/\partial a$, $\partial f/\partial b$ e $\partial f/\partial c$. Indique, em cada nodo, qual padrão foi aplicado (distribuidor, comutador ou roteador) e como os gradientes que chegam em b são combinados.

19) A regressão logística $\hat{y} = \sigma(\theta_0+\theta_1 x_1+\theta_2 x_2)$, treinada com a *Binary Cross Entropy* $L = -\left(y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})\right)$, pode ser vista como um grafo computacional. Resolva os itens abaixo:

a) Decomponha a computação de L em nodos elementares, explicitando os nodos de produto, os nodos de soma, o nodo sigmoide (já agrupado em uma única operação, como visto em aula) e o nodo da *loss*.

b) Derive $\partial L/\partial\theta_0$, $\partial L/\partial\theta_1$ e $\partial L/\partial\theta_2$ aplicando a regra da cadeia nodo a nodo, no sentido inverso do grafo, informando o gradiente que atravessa cada nodo. Simplifique as expressões finais.

20) *Frameworks* como PyTorch representam a computação como um grafo construído automaticamente durante a execução do código *forward*, e é esse grafo que permite calcular as derivadas automaticamente, como discutido em aula. Explique por que essa representação é necessária para o cálculo automático de gradientes e descreva o que precisa ser armazenado durante o *forward pass* para que o *backward pass* seja possível. Ilustre com pelo menos dois nodos concretos cujas derivadas locais dependem de valores calculados no *forward*.

21) No nodo de produto de matrizes $Z = A\,\theta^\top$, visto na versão vetorizada do grafo computacional, temos $A \in \mathbb{R}^{N\times d_e}$, $\theta \in \mathbb{R}^{d_s\times d_e}$, e o *backward* recebe o gradiente *upstream* $G = \partial J/\partial Z \in \mathbb{R}^{N\times d_s}$. Quais expressões computam os gradientes das duas entradas do nodo?

(a) $\partial J/\partial A = G\,\theta$ e $\partial J/\partial\theta = G^\top A$.

(b) $\partial J/\partial A = G\,\theta^\top$ e $\partial J/\partial\theta = A^\top G$.

(c) $\partial J/\partial A = G^\top A$ e $\partial J/\partial\theta = G\,\theta$.

(d) $\partial J/\partial A = \theta^\top G$ e $\partial J/\partial\theta = A\,G^\top$.

22) Considere o MLP vetorizado representado como grafo computacional com os nodos *matmul*, sigmoide e *loss*, como visto em aula: $Z^{(1)} = X{\theta^{(1)}}^\top$, $A^{(1)} = \sigma(Z^{(1)})$, $Z^{(2)} = A^{(1)}{\theta^{(2)}}^\top$, $\hat{y} = \sigma(Z^{(2)})$ e $J = \frac{1}{2}\sum (y-\hat{y})^2$. Use uma única observação $X = \begin{bmatrix}1 & 2\end{bmatrix}$ com alvo $y = 1$ e pesos

$$\theta^{(1)} = \begin{bmatrix}0.5 & -0.5\\ 0.5 & 0.5\end{bmatrix}, \qquad \theta^{(2)} = \begin{bmatrix}0.5 & -0.5\end{bmatrix}.$$

a) Execute o *forward pass* nodo a nodo, informando $Z^{(1)}$, $A^{(1)}$, $Z^{(2)}$, $\hat{y}$ e J.

b) Execute o *backward pass* nodo a nodo, iniciando com $\partial J/\partial J = 1$, informando o gradiente na saída de cada nodo e, ao final, $\partial J/\partial\theta^{(1)}$ e $\partial J/\partial\theta^{(2)}$. Em cada passo, indique qual regra de *backward* foi aplicada (nodo *loss*, nodo de ativação ou nodo *matmul*).

Descida de Gradiente

23) Quando usamos a seguinte expressão $w_{(t+1)} = w_{(t)} - \eta\nabla\mathcal{L}$ para a atualização dos pesos de uma rede neural do tipo *multilayer perceptron*, considerando que o gradiente está sendo estimado com base em uma única observação, temos qual variação do algoritmo de descida de gradiente?

(a) *Batch Gradient Descent*.

(b) *Mini Batch Gradient Descent*.

(c) *Stochastic Gradient Descent*.

(d) *Deterministic Gradient Descent*.

24) Uma das modificações mais comuns no algoritmo de descida de gradiente envolve alterar a atualização dos pesos para que considere dois termos: um dos termos é o gradiente da função *loss* em relação aos pesos considerando *batch* atual; enquanto o segundo termo é um valor definido recursivamente que leva em consideração os gradientes anteriores. Essa modificação obedece a seguinte equação:

$$W_{(t+1)} = W_{(t)} - \eta \mathcal{V}_{(t)}$$
$$\mathcal{V}_{(t)} = \beta \mathcal{V}_{(t-1)} + (1-\beta)\nabla \mathcal{L}$$

Sendo que por definição $\mathcal{V}_{(0)} = 0$.

Qual nome se dá para essa modificação na descida de gradiente?

(a) *Batch Gradient Descent*

(b) *Momentum*

(c) *L2 Regularization*

(d) *Stochastic Gradient Descent*

(e) *Batch Normalization*

25) O valor da função *loss* da sua rede neural está oscilando ao invés de decrescer constantemente conforme esperado durante o treinamento. Supondo que você esteja utilizando *Mini-Batch Gradient Descent*, quais das opções abaixo são boas modificações que você pode introduzir para tentar resolver esse problema?

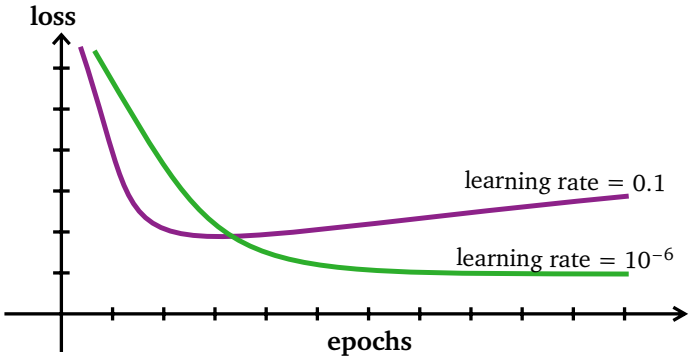
(a) Aparentemente isso é um problema dos dados, obter uma quantidade maior de dados para efetuar o treinamento é uma alternativa razoável.

(b) Aumentar o valor hiperparâmetro *learning rate* pode ajudar pois o processo de otimização vai chegar a um mínimo mais rápido.

(c) Diminuir o valor hiperparâmetro *learning rate* pode ajudar pois o processo de otimização para garantir que um mínimo local seja encontrado e não "pulado".

(d) Aumentar o hiperparâmetro *batch size* pode ajudar pois vai introduzir maior variabilidade em cada *mini-batch*. Assim as chances de cada um dos passos da otimização gerar uma diminuição no loss aumenta.

26) Você está experimentando com diferentes *learning rates* ao treinar uma rede neural específica. A evolução do valor da função *loss* no decorrer das épocas usando *learning rates* 0.1 e 10^{-6} estão retratados na figura abaixo. Com base nesses resultados, qual seria um bom valor de *learning rate* para testar a seguir?



(a) 0.2

(b) 10^{-4}

(c) 10^{-8}

27) François Chollet, o criador do *Keras*, comentou em 2022 que você não deve tentar aproveitar um *scheduler* de *learning rate* que funcionou bem em outro conjunto de dados para treinar em um novo conjunto de dados. A afirmação de Chollet vale mesmo nos casos nos quais a arquitetura da rede se mantém igual. Dentre as opções abaixo, quais são racionais razoáveis para essa afirmação?

(a) *Learning rate schedulle* deve variar conforme o número de observações no conjunto de dados. Dessa forma se os dados mudarem precisamos mudar o *scheduller*.

(b) O *learning rate schedulle* deve mudar sempre que a distribuição alvo mudar. Nos casos que ela se mantém igual não é necessário realizar modificação.

(c) Um novo conjunto de dados modifica o *tradeoff* entre otimização e regularização. Dessa forma, o *learning rate scheduller* é específico para um conjunto de dados.

28) Você se depara com uma situação na qual o seu time de pesquisa está usando o maior *batch size* possível, respeitando apenas a limitação da memória disponível na GPU. Você entende existem argumentos a favor de um *batch size* grande, contudo você quer trazer para a discussão argumentos contrários para incentivar o uso de *batch sizes* menores. Dentre as alternativas abaixo, quais seriam bons argumentos para diminuir o *batch size*?

(a) Um *batch size* menor torna o processo de treino mais ruidoso, auxiliando na diminuição de *overfitting*.

(b) Um *batch size* menor pode tornar a paralelização mais otimizada e portanto o treinamento mais rápido.

(c) Um *batch size* menor pode levar a uma maior generalização da rede, evitando mínimos locais.

(d) Um *batch size* menor reduz o ruído no treinamento, melhorando a generalização do modelo treinado.

(e) O hiperparâmetro *batch size* apenas diz respeito a limitações de *hardware* e não influencia a solução que será otimizada.

29) Dado que a descida de gradiente possui na sua formalização original a seguinte regra de atualização $\theta_{t+1} = \theta_t - \eta \nabla_\theta J$. Por que a atualização dos pesos com $-\nabla_\theta J$ diminui o valor da função de custo?

30) Por que dizemos que a formulação original do *momentum* calcula o gradiente "no local errado"?

31) Qual o efeito prático de normalizar o gradiente (conforme feito no Adam)?

32) Qual a diferença de uma iteração para uma época?

33) Considerando que θ_t denota o vetor de parâmetros treináveis de uma rede neural na iteração t e as seguintes informações:

$$\eta = 0.1, \quad \beta = 0.8, \quad \theta_0 = \begin{bmatrix} 5 \\ 10 \end{bmatrix}, \quad v_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$
$$\nabla_{\theta_0} J = \begin{bmatrix} -2 \\ 5 \end{bmatrix}, \quad \nabla_{\theta_1} J = \begin{bmatrix} -1 \\ 8 \end{bmatrix}, \quad \nabla_{\theta_2} J = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

a) Como fica θ_1 ao seguir a regra de atualização do SGD?

b) Como fica θ_1 ao seguir a regra de atualização da descida de gradiente com *normalização*?

c) Como fica θ_2 ao seguir a regra de atualização da descida de gradiente com *momentum*?

d) ☠️ Como fica θ_2 ao seguir a regra de atualização do otimizador Adam? (*Nota*: ignore o $\beta = 0.8$ usado nos itens anteriores; aqui usamos os hiperparâmetros padrão do Adam, $\beta_1 = 0.9$ e $\beta_2 = 0.999$)

Funções de Custo e Funções de Ativação

34) Dentre as funções de ativação abaixo, qual é a opção mais adequada para modelar um problema de classificação **multiclasse** no qual você quer interpretar a saída da rede como as probabilidades da observação pertencer a uma dada classe?

(a) ReLU: $f(x_i) = max(0, x_i)$

(b) Sigmoid: $f(x_i) = \frac{1}{1+e^{-x_i}}$

(c) Tanh: $f(x_i) = \frac{2}{1+e^{-2x_i}} - 1$

(d) Softplus: $f(x_i) = \log_e(1 + e^{x_i})$

(e) Softmax $f(x_i) = \frac{e^{x_i}}{\sum_z e^{x_z}}$

(f) Linear $f(x_i) = x_i$

35) Dentre as funções de ativação abaixo, qual é a opção mais adequada para modelar um problema de classificação **multilabel** no qual você quer interpretar a saída da rede como as probabilidades da observação pertencer a uma dada classe?

(a) ReLU: $f(x_i) = max(0, x_i)$

(b) Sigmoid: $f(x_i) = \frac{1}{1+e^{-x_i}}$

(c) Tanh: $f(x_i) = \frac{2}{1+e^{-2x_i}} - 1$

(d) Softplus: $f(x_i) = \log_e(1 + e^{x_i})$

(e) Softmax $f(x_i) = \frac{e^{x_i}}{\sum_z e^{x_z}}$

(f) Linear $f(x_i) = x_i$

36) "Para um problema de classificação **multilabel** onde a saída pode ter 10 classes diferentes (simultaneamente ou não) a função de ativação na camada de saída do MLP deve ser _____ e o treinamento deve buscar otimizar a função de perda _____". Quais opções abaixo melhor preenchem as lacunas da afirmação acima?

(a) ReLU: $f(x_i) = max(0, x_i)$

(b) Sigmoid: $f(x_i) = \frac{1}{1+e^{-x_i}}$

(c) Tanh: $f(x_i) = \frac{2}{1+e^{-2x_i}} - 1$

(d) Softplus: $f(x_i) = \log_e(1 + e^{x_i})$

(e) Softmax $f(x_i) = \frac{e^{x_i}}{\sum_z e^{x_z}}$

(f) Linear $f(x_i) = x_i$

(g) *Binary Cross Entropy Loss*: $L_{\text{bce}} = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$.

(h) *Categorical Cross Entropy Loss*: $L_{\text{cce}} = -\sum_{i=1}^{c} y_i \log \hat{y}_i$

(i) *Sum of Squared Residuals*: $L_{\text{ssr}} = (y - \hat{y})^2$.

(j) *Absolute Error Loss*: $L_{\text{abs}} = |y - \hat{y}|$.

(k) *Zero-One Loss*: $L_{01} = I(y \neq \hat{y})$.

37) Durante o treinamento de uma rede neural você percebe que os gradientes diminuem rapidamente até que se aproximam de zero, fazendo que os pesos da rede permaneçam praticamente sem mudança. Você rapidamente identifica que se trata de um caso de *vanishing gradient*, marque dentre as opções abaixo possíveis causas para esse problema.

(a) As camadas ocultas usam função de ativação *Tanh*.

(b) As camadas ocultas usam função de ativação *ReLU*.

(c) As camadas ocultas usam função de ativação *Sigmoid*.

(d) A rede usa *batch normalization*.

38) A função de ativação *Rectified Linear Unit (ReLU)* é uma das funções de ativação mais utilizadas em redes neurais. A *ReLU* é definida como $f(x) = \max(0, x)$. Sobre essa função de ativação, é correto afirmar que:

(a) Minimiza o problema de *vanishing gradient* em comparação com a função de ativação sigmoid.

(b) É continua e diferenciável em todo seu domínio.

(c) Não é contínua e não é diferenciável em todo seu domínio.

(d) É contínua e não é diferenciável em todo seu domínio.

39) Dentre as opções abaixo, quais adicionam não linearidades na rede neural.

(a) Adicionar camadas de convolução.

(b) Diminuir a taxa de aprendizado.

(c) Embaralhar o conjunto de treinamento.

(d) Usar função de ativação ReLU.

(e) Adicionar regularização.

40) O preço de um determinado veículo i pode ser modelado satisfatoriamente pela seguinte função $f(\mathbf{X}^{(i)}) = 10000X_1^{(i)} + 50000X_2^{(i)} + 3500$. Responda as questões abaixo supondo que você está utilizando um *MultiLayer Perceptron* para aproximar o preço do veículo e justifique sua resposta.

(a) Quantos neurônios/unidades são necessários no mínimo para modelar o preço dos veículos?

(b) Qual função de ativação você deveria utilizar?

41) Cite e explique duas características desejáveis de uma função de ativação em redes neurais.

42) Por que é usual trabalhar com funções de ativação não-lineares em redes neurais?

43) Calcule as derivadas das seguintes funções de ativação:

a) ReLU $\varphi(x) = max(0, x)$

b) Sigmoid $\varphi(x) = \frac{1}{(1+e^{-x})}$

c) ☠ Softmax $\varphi(\mathbf{x}) = S_i = \frac{e^{x_i}}{\Sigma e^{x_j}}$, (encontrar a matriz Jacobiana)

44) Qual a utilidade das funções de ativação nas camadas ocultas de um MLP?

45) Qual a utilidade das funções de ativação na camada de saída de um MLP?

46) Calcule as derivadas das seguintes funções de custo:

a) $L_{l2} = \sum \left(y^{(i)} - \hat{y}^{(i)}\right)^2$

b) $L_{BCE} = -yln(\hat{y}) - (1-y)ln(1-\hat{y})$

47) Desenvolva a função de custo do Erro Médio Quadrático a partir da distribuição Gaussiana.

$$Pr(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

48) Desenvolva a função de custo Entropia Binária Cruzada a partir da distribuição de Bernoulli.

$$Pr(y|\lambda) = (1-\lambda)^{(1-y)}\lambda^y$$

49) ☠ Suponha que você quer criar uma rede neural para estimar a direção y (em radianos) do vento a partir de medições de pressão $\mathbf{x}$. Uma distribuição que trabalha no domínio circular é a distribuição de von Mises:

$$Pr(y|\mu, k) = \frac{exp[k * cos(y-\mu)]}{2\pi * Bessel_0[k]}$$

onde μ é a direção média e k é o inverso da variância. O termo $Bessel_0[k]$ é uma função modificada de Bessel com grau o. Crie uma função de custo para aprender o parâmetro μ dessa distribuição.

50) ☠ Suponha que você quer criar uma rede neural para estimar o número de pedestres $y \in \{0, 1, 2, ...\}$ que passam em uma rua da cidade em algum determinado momento. Você possui um vetor de atributos $\mathbf{x}$ que contém informações sobre a hora do dia, sobre o clima e sobre o dia ser feriado ou não. Uma distribuição de probabilidade adequada para trabalhar com esse problema é a distribuição de Poisson:

$$Pr(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

onde λ é o único parâmetro, que representa a média da distribuição. Crie uma função de custo assumindo que temos acesso a I instâncias de treinamento compostas por pares $\{x^i, y^i\}$.

Inicialização de Pesos

51) Você inicializa todos os pesos de um MLP com um mesmo valor constante $c \neq 0$ e todos os bias com zero. As camadas ocultas usam a função de ativação tanh. Qual o principal problema que esse esquema de inicialização causa durante o treinamento?

(a) Os gradientes da primeira camada são nulos já na primeira iteração e os pesos dessa camada nunca são atualizados.

(b) A função de custo se torna não diferenciável nos pontos em que pesos de camadas distintas possuem o mesmo valor.

(c) As unidades de uma mesma camada computam a mesma saída e recebem gradientes idênticos, permanecendo cópias umas das outras.

(d) As pré-ativações crescem exponencialmente com a profundidade já no primeiro *forward pass*, causando *overflow* numérico.

52) Um MLP profundo com ativações sigmoid nas camadas ocultas tem os pesos inicializados com $\theta \sim \mathcal{N}(0, \sigma^2)$, onde σ é muito maior do que o recomendado pelas heurísticas de inicialização (He, Xavier). Qual comportamento você espera observar nas primeiras iterações do treinamento?

(a) Unidades mortas em todas as camadas ocultas, com saída exatamente zero independentemente da entrada apresentada.

(b) Pré-ativações com magnitude grande, unidades saturadas e gradientes próximos de zero nas camadas iniciais da rede.

(c) Pré-ativações próximas de zero, unidades operando na região aproximadamente linear e gradientes estáveis em todas as camadas.

(d) Convergência acelerada nas primeiras épocas, seguida de estagnação quando a magnitude dos pesos diminui ao longo do treino.

53) Que problemas podemos ter com unidades na camada oculta que usam a função sigmoid?

54) Que problemas podemos ter com unidades na camada oculta que usam a função relu?

55) Qual a motivação das inicializações *He* e *Xavier*.

56) Seja a uma variável aleatória contínua, com distribuição simétrica em torno de $E[a] = 0$ e variância $\mathrm{Var}[a] = \sigma^2$. Mostre que

$$E\left[\mathrm{ReLU}(a)^2\right] = \frac{\sigma^2}{2}$$

Dica: escreva a esperança como uma integral, use a simetria da distribuição para relacionar a integral na metade positiva do domínio com $E[a^2]$ e lembre que $E[a^2] = \mathrm{Var}[a] + (E[a])^2$.

57) Considere a pré-ativação da unidade i na camada l de um MLP cuja camada anterior possui $n^{(l-1)}$ unidades:

$$z_i^{(l)} = b_i^{(l)} + \sum_{j=1}^{n^{(l-1)}} \theta_{ij}^{(l)} \, \varphi\left(z_j^{(l-1)}\right)$$

onde os bias são inicializados em zero, os pesos $\theta_{ij}^{(l)} \sim \mathcal{N}(0, \sigma_\theta^2)$ são independentes entre si e independentes de $\varphi(z_j^{(l-1)})$, e as pré-ativações $z_j^{(l-1)}$ têm distribuição simétrica em torno de zero com variância $\sigma^2_{z^{(l-1)}}$. Resolva os itens abaixo:

a) Mostre que $E[z_i^{(l)}] = 0$.

b) Supondo $\varphi = \mathrm{ReLU}$, use o resultado do exercício anterior para mostrar a recorrência $\sigma^2_{z^{(l)}} = \frac{n^{(l-1)}}{2} \, \sigma_\theta^2 \, \sigma^2_{z^{(l-1)}}$.

c) Imponha que a variância se preserve de uma camada para a outra, isto é, $\sigma^2_{z^{(l)}} = \sigma^2_{z^{(l-1)}}$, e derive a variância da inicialização *He*.

d) Supondo agora $\varphi = \tanh$ e usando a aproximação $\varphi(z) \approx z$ para z próximo da origem, refaça o argumento dos itens (b) e (c) e derive a variância da inicialização *Xavier*.

Regularização

58) *Label Smoothing* é uma técnica de regularização que visa diminuir a tendência de *overfitting* do modelo combatendo principalmente inferências com *overconfidence*. Essa técnica redefine os valores alvo da seguinte forma:

$$\mathbf{y}_{ls}^{(i)} = (1-\alpha)\mathbf{y}_{\text{oh}}^{(i)} + \frac{\alpha}{C}$$

onde $y_{\text{oh}}^{(i)}$ é a variável alvo codificada em *one-hot encoding*, α é a constante de *smoothing* e C é o número de classes do problema. Sobre *Label Smoothing*, marque as alternativas verdadeiras:

(a) Quando $\alpha = 1$ a variável alvo fica distribuída uniformemente entre as classes.

(b) Quando $\alpha = 0$ a variável alvo permanece inalterada em formato *one-hot encoding*.

(c) Para encontrar o valor ideal de α devemos considerar a quantidade de observações no conjunto de dados.

(d) Quando $\alpha = 0$ temos a maior intensidade de *smoothing*.

59) Sobre as penalidades de regularização L1 e L2 adicionadas à função de custo, $\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \lambda\,\Omega(\theta)$ com $\Omega_{L1} = \|\theta\|_1$ e $\Omega_{L2} = \|\theta\|_2^2$, é correto afirmar que:

(a) A penalidade L2 tende a zerar exatamente alguns componentes do vetor de pesos, induzindo soluções esparsas.

(b) A penalidade L1 tende a zerar exatamente alguns componentes do vetor de pesos, induzindo soluções esparsas.

(c) A penalidade L1 equivale, na visão probabilística, a adicionar um *prior* gaussiano sobre os pesos no critério de máxima verossimilhança.

(d) A penalidade L2 aumenta a magnitude dos pesos durante o treinamento para compensar o termo adicional na função de custo.

60) Uma rede foi treinada com *dropout*, desligando cada unidade oculta com probabilidade p a cada *forward pass*. Na inferência, todas as unidades permanecem ativas. Sobre o ajuste necessário nesse cenário, é correto afirmar que:

(a) As ativações devem ser multiplicadas por p na inferência, restaurando a fração de unidades que era desligada durante o treino.

(b) Nenhum ajuste é necessário, pois o *dropout* altera apenas o *backward pass* e não muda a escala das ativações.

(c) As ativações devem ser multiplicadas por $(1-p)$ na inferência, compensando a diferença de magnitude esperada em relação ao treino.

(d) Os pesos devem ser reamostrados com variância $(1-p)$ antes da inferência, igualando a distribuição das pré-ativações à do treino.

61) Demonstre que uma regularização L2 na função de custo de uma unidade com função de ativação identidade é equivalente a diminuir a magnitude do vetor de pesos antes de realizar a atualização. *Dica: escreva a função de custo com o termo de regularização $||\theta||^2$ e use essa função de custo na expressão de atualização dos pesos da descida de gradiente. Colocando o θ em evidência ficará claro que os pesos estão sendo multiplicados por um valor na faixa* $(0, 1)$.

62) Durante o treinamento com *dropout*, a ativação a de uma unidade oculta é multiplicada por uma máscara binária $m \sim \text{Bernoulli}(1 - p)$, onde p é a probabilidade de desligamento. Resolva os itens abaixo:

    a) Calcule $E[m \cdot a]$ e compare o resultado com a magnitude da ativação em inferência, quando a unidade está sempre ativa.

    b) Explique as duas formas vistas em aula de corrigir essa diferença de escala: o *rescaling* determinístico em inferência e o *inverted dropout* adotado como padrão por PyTorch e Keras.

63) Explique o funcionamento do *early stopping*: o que é monitorado durante o treinamento, qual o papel do hiperparâmetro *patience* e por que a técnica tem efeito análogo ao da regularização L2. Cite também o benefício adicional que ela traz além do efeito regularizador.

Gabarito

1) c. *Machine Learning* é indicado justamente quando não sabemos escrever o algoritmo que transforma a entrada na saída, mas dispomos de exemplos nos quais um padrão pode ser aprendido. As alternativas (a) e (d) descrevem cenários ideais para uma solução programada explicitamente, e em (b) o desejo de reduzir tempo de desenvolvimento não justifica a troca de paradigma quando a solução exata é conhecida e barata.

2) b. O erro no conjunto de teste é a estimativa do desempenho em dados novos, e o modelo B tem o menor erro de teste (6.3). O modelo A sofre de viés alto (erros altos em treino e teste), enquanto C e D exibem *overfitting*: erro de treino baixo com erro de teste alto.

3) d. A equação normal exige inverter $\mathbf{X}^\top\mathbf{X}$, uma matriz $M \times M$, operação de custo cúbico em M; com centenas de milhares de atributos isso é inviável, enquanto a descida de gradiente escala bem. A alternativa (a) afirma algo verdadeiro sobre a equação normal, mas não a torna preferível nesse cenário; (b) é falsa pois J_{SSR} permanece convexa na Regressão Linear; (c) inverte os custos dos dois métodos.

4) a. Ao compor J_{SSR} com a sigmoide, a função de custo deixa de ser convexa e a descida de gradiente pode estacionar em mínimos locais; a J_{BCE} é convexa para a Regressão Logística. Em (b), a composição continua diferenciável; em (c), a J_{BCE} não admite solução analítica; em (d), a fronteira de decisão da Regressão Logística continua linear.

5) b. Hiperparâmetros (taxa de aprendizado, *batch size*, arquitetura) não são aprendidos pela descida de gradiente, então seus valores são encontrados por um processo de busca externo ao treino. A alternativa (a) descreve o treinamento dos parâmetros, (c) descreve o próprio aprendizado da hipótese e (d) confunde o processo com regularização.

6) Ajustar bem o conjunto de treino significa encontrar uma hipótese que minimiza a função de custo sobre os exemplos já observados. Generalizar significa obter erro baixo em dados novos, vindos da mesma relação entre atributos e variável alvo, que é o objetivo real de um sistema de *Machine Learning*. As duas coisas não são equivalentes: o polinômio de grau 4 tem parâmetros suficientes para passar exatamente por todos os pontos de treino, então ele ajusta inclusive as flutuações particulares daquela amostra e não apenas a tendência geral. Fora dos pontos de treino ele oscila, e por isso erra muito no ponto de teste. Em termos da decomposição $\text{Erro}_{\text{teste}} = \text{Viés}^2 + \text{Variância} + \sigma^2$, o polinômio é um modelo de variância alta: se sorteássemos outro conjunto de treino de mesmo tamanho, a curva ajustada mudaria drasticamente. A reta pertence a uma família de hipóteses mais simples, então tem viés maior (não consegue representar qualquer padrão) e erro de treino diferente de zero, mas tem variância baixa e captura a tendência geral dos dados, o que resulta em erro de teste menor. O termo σ^2 é o ruído irredutível, que nenhum dos dois modelos consegue eliminar. Isso ilustra o efeito tesoura: com poucos dados, regras mais simples tendem a obter erros menores em novos dados, pois a variância domina o erro dos modelos complexos; com muitos dados, a variância dos modelos complexos diminui e eles passam a obter erros menores, pois seu viés é menor.

7) a. A derivada da função *sign* é zero em todo ponto onde é definida (e indefinida em $x = 0$), então nenhum gradiente flui pelas unidades e o *backpropagation* não tem como atualizar os pesos. A alternativa (c) erra ao afirmar que **apenas** a descontinuidade em $x = 0$ impede a otimização, pois a derivada nula no restante do domínio também impede; (d) é uma limitação verdadeira do Perceptron isolado, mas não é o que impede organizá-lo em rede.

8) Pois a derivada da função de ativação sinal é zero ou indefinida em todo domínio da função. Devido a isso, não conseguimos propagar os erros.

9) A fronteira de decisão é o conjunto de pontos onde o argumento da função sinal troca de sinal, isto é, onde $\theta_0 + X_1\theta_1 + X_2\theta_2 = 0$. Supondo $\theta_2 \neq 0$, isolamos $X_2 = -\frac{\theta_0+\theta_1 X_1}{\theta_2}$, que é a equação de uma reta no plano (X_1, X_2) com coeficiente angular $-\theta_1/\theta_2$ e intercepto $-\theta_0/\theta_2$. De um lado dessa reta a saída é $+1$ e do outro é -1, portanto a fronteira de decisão é linear.

10) $\hat{y} = sinal(\theta_0+X_1\theta_1+X_2\theta_2)$, onde $\theta_0 = -1, \theta_1 = 1, \theta_2 = 1$ e as entradas falsas são codificadas como -1 e as entradas verdadeiras como $+1$. Verificação nas quatro combinações: $(-1,-1) : z = -3 \Rightarrow -1$; $(+1,-1) : z = -1 \Rightarrow -1$; $(-1,+1) : z = -1 \Rightarrow -1$; $(+1,+1) : z = +1 \Rightarrow +1$, exatamente a tabela-verdade do *and*.

11) a. Cada unidade computa o produto escalar entre o vetor de entradas e o vetor de pesos (somado ao bias) para formar a pré-ativação; a função de ativação é aplicada depois, sobre esse resultado. *Backpropagation* e descida de gradiente pertencem ao treinamento, não ao *forward pass*.

12) b. O *forward pass* computa as saídas de cada unidade até produzir a predição, e o *backward pass* computa os gradientes da função de custo em relação aos pesos. As alternativas (d) e (e) confundem as duas fases com os algoritmos que as utilizam.

13) $3D + 1 + (K-1)D(D+1)$. Repare que essa expressão não é a única resposta possível para o exercício. Dependendo de como você desenvolver a questão poderá chegar em expressões equivalentes como por exemplo: $2D + (K-1)DD + KD + 1$. Contagem por partes: da entrada para a primeira camada oculta são D pesos e D bias; entre camadas ocultas são $(K-1)$ transições com D^2 pesos e D bias cada; da última camada oculta para a saída são D pesos e 1 bias. Somando: $2D + (K-1)D(D+1) + D + 1 = 3D + 1 + (K-1)D(D+1)$

14) *Forward Pass*: onde as saídas são calculadas; *Loss Function*: onde a diferença entre a saída e o valor esperado são comparadas; *Backward Pass*: onde os gradientes da função de custo em relação a cada um dos pesos é calculado; *Optimizer*: onde os pesos são atualizados.

15) Considerando ReLU $\varphi(z) = \max(0, z)$ em todas as unidades (inclusive na saída) e taxa de aprendizado $\eta = 0.1$.

Forward Pass:

$$z^{(1)} = \theta^{(1)}X + \beta^{(1)} = \begin{bmatrix}0.5 & -0.2\\ 0.1 & 0.2\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix} + \begin{bmatrix}0.2\\-0.1\end{bmatrix} = \begin{bmatrix}1.0\\0.3\end{bmatrix}, \quad a^{(1)} = \varphi(z^{(1)}) = \begin{bmatrix}1.0\\0.3\end{bmatrix}$$

$$z^{(2)} = \theta^{(2)}a^{(1)} + \beta^{(2)} = \begin{bmatrix}0.2 & -0.1\end{bmatrix}\begin{bmatrix}1.0\\0.3\end{bmatrix} + 0.4 = 0.57, \quad \hat{y} = a^{(2)} = \varphi(0.57) = 0.57$$

Loss: $J = (\hat{y} - y)^2 = (0.57 - 1)^2 = 0.1849$.

Backward Pass: como $\frac{\partial J}{\partial \hat{y}} = 2(\hat{y} - y) = -0.86$ e $\varphi'(z) = 1$ para $z > 0$,

$$\delta^{(2)} = 2(\hat{y} - y) \odot \varphi'(z^{(2)}) = -0.86$$

$$\frac{\partial J}{\partial \theta^{(2)}} = \delta^{(2)}a^{(1)\top} = \begin{bmatrix}-0.86 & -0.258\end{bmatrix}, \quad \frac{\partial J}{\partial \beta^{(2)}} = \delta^{(2)} = -0.86$$

$$\delta^{(1)} = (\delta^{(2)}\theta^{(2)}) \odot \varphi'(z^{(1)}) = \begin{bmatrix}-0.172 & 0.086\end{bmatrix}$$

$$\frac{\partial J}{\partial \theta^{(1)}} = \delta^{(1)\top}X^{\top} = \begin{bmatrix}-0.172\\0.086\end{bmatrix}\begin{bmatrix}2 & 1\end{bmatrix} = \begin{bmatrix}-0.344 & -0.172\\0.172 & 0.086\end{bmatrix}, \quad \frac{\partial J}{\partial \beta^{(1)}} = \delta^{(1)} = \begin{bmatrix}-0.172 & 0.086\end{bmatrix}$$

Atualização (SGD, $\theta_{t+1} = \theta_t - \eta \nabla J$):

$$\theta^{(1)} \leftarrow \begin{bmatrix} 0.5 & -0.2 \\ 0.1 & 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} -0.344 & -0.172 \\ 0.172 & 0.086 \end{bmatrix} = \begin{bmatrix} 0.5344 & -0.1828 \\ 0.0828 & 0.1914 \end{bmatrix}$$
$$\beta^{(1)} \leftarrow \begin{bmatrix} 0.2 & -0.1 \end{bmatrix} - 0.1 \begin{bmatrix} -0.172 & 0.086 \end{bmatrix} = \begin{bmatrix} 0.2172 & -0.1086 \end{bmatrix}$$
$$\theta^{(2)} \leftarrow \begin{bmatrix} 0.2 & -0.1 \end{bmatrix} - 0.1 \begin{bmatrix} -0.86 & -0.258 \end{bmatrix} = \begin{bmatrix} 0.286 & -0.0742 \end{bmatrix}$$
$$\beta^{(2)} \leftarrow 0.4 - 0.1(-0.86) = 0.486$$

16) a. A regra da cadeia local exige apenas o gradiente *upstream*, que resume todo o caminho da *loss* até a saída do nodo, e a derivada local da operação do nodo em relação a cada entrada. Essa localidade é o que torna o *backpropagation* eficiente. A alternativa (b) descreve diferenciação simbólica e (c) descreve diferenças finitas, métodos que a diferenciação automática substitui; (d) inverte a lógica, pois os gradientes dos parâmetros são o resultado do percurso, não um pré-requisito.

17) c. *Forward*: $u = 3$, $v = 6$, $f = 18$. No *backward*, b recebe contribuição por dois caminhos: via u, $\frac{\partial f}{\partial u} \cdot 1 = v = 6$; via v, $\frac{\partial f}{\partial v} \cdot c = u \cdot c = 9$. Somando, $\frac{\partial f}{\partial b} = 6 + 9 = 15$. Os distratores 6 e 9 são os caminhos isolados e 18 é o valor do *forward*. Verificação analítica: com $a = 1$ e $c = 3$, $f(b) = 3b + 3b^2$ e $f'(2) = 3 + 12 = 15$.

18) a) *Forward pass*, na ordem topológica:

$$u = a + b = 2 + 3 = 5$$
$$v = \max(b, c) = \max(3, 4) = 4$$
$$f = u \cdot v = 5 \cdot 4 = 20$$

b) *Backward pass*, iniciando com $\partial f / \partial f = 1$ e percorrendo o grafo na ordem inversa:

- Nodo $f = u \cdot v$ (comutador): cada entrada recebe o gradiente *upstream* multiplicado pelo valor da outra entrada, logo $\frac{\partial f}{\partial u} = 1 \cdot v = 4$ e $\frac{\partial f}{\partial v} = 1 \cdot u = 5$.
- Nodo $u = a + b$ (distribuidor): copia o gradiente *upstream* para as duas entradas, logo $\frac{\partial f}{\partial a} = 4$ e a contribuição para b por este caminho é 4.
- Nodo $v = \max(b, c)$ (roteador): como $c = 4 > b = 3$, a entrada vencedora é c, que recebe o gradiente inteiro, $\frac{\partial f}{\partial c} = 5$. A contribuição para b por este caminho é 0.
- Variável b usada em dois nodos (somador no *backward*): os gradientes que chegam são somados, $\frac{\partial f}{\partial b} = 4 + 0 = 4$.

Resultado: $\frac{\partial f}{\partial a} = 4$, $\frac{\partial f}{\partial b} = 4$ e $\frac{\partial f}{\partial c} = 5$. Verificação: na região em que $c > b$ vale $f = (a + b)\, c$, portanto $\frac{\partial f}{\partial a} = c = 4$, $\frac{\partial f}{\partial b} = c = 4$ e $\frac{\partial f}{\partial c} = a + b = 5$.

19) a) Decomposição em nodos elementares, na ordem do *forward*:

$$t_1 = \theta_1 \cdot x_1 \quad \text{(produto)}$$
$$t_2 = \theta_2 \cdot x_2 \quad \text{(produto)}$$
$$s_1 = \theta_0 + t_1 \quad \text{(soma)}$$
$$z = s_1 + t_2 \quad \text{(soma)}$$
$$\hat{y} = \sigma(z) \quad \text{(sigmoide agrupada)}$$
$$L = -\left(y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})\right) \quad (\textit{loss})$$

b) *Backward pass* nodo a nodo, iniciando com $\partial L/\partial L = 1$:

- Nodo da *loss*: derivando L em relação a $\hat{y}$,

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})}.$$

- Nodo sigmoide: a derivada local é $\sigma'(z) = \hat{y}(1-\hat{y})$, então

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \hat{y}(1-\hat{y}) = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y}) = \hat{y}-y.$$

- Nodos de soma (distribuidores): copiam $\hat{y}-y$ para todas as entradas, logo

$$\frac{\partial L}{\partial s_1} = \frac{\partial L}{\partial t_2} = \frac{\partial L}{\partial \theta_0} = \frac{\partial L}{\partial t_1} = \hat{y}-y.$$

- Nodos de produto (comutadores): multiplicam o gradiente *upstream* pelo valor da outra entrada, logo

$$\frac{\partial L}{\partial \theta_1} = (\hat{y}-y)\,x_1 \quad \text{e} \quad \frac{\partial L}{\partial \theta_2} = (\hat{y}-y)\,x_2.$$

Resumo: $\frac{\partial L}{\partial \theta_0} = \hat{y}-y$, $\frac{\partial L}{\partial \theta_1} = (\hat{y}-y)\,x_1$ e $\frac{\partial L}{\partial \theta_2} = (\hat{y}-y)\,x_2$. A simplificação ocorre porque o denominador $\hat{y}(1-\hat{y})$ da derivada da BCE cancela exatamente com a derivada da sigmoide, deixando o gradiente na pré-ativação igual ao erro $\hat{y}-y$.

20) O *backward pass* é a aplicação repetida da regra da cadeia sobre uma composição de operações elementares. Para automatizar esse processo, o *framework* precisa saber quais operações foram executadas e em que ordem de dependência, e é exatamente isso que o grafo (um DAG) registra: cada operação executada no *forward* vira um nodo, e as arestas indicam de quais valores cada nodo depende. Com o grafo montado, basta percorrê-lo na ordem topológica inversa multiplicando o gradiente *upstream* pela derivada local de cada nodo, o que é eficiente no caso típico de redes neurais, com uma *loss* escalar e muitos parâmetros.

Durante o *forward pass*, três coisas precisam ser armazenadas: (1) a estrutura do grafo, ou seja, quais nodos alimentam quais, pois sem ela não há como saber para onde propagar cada gradiente; (2) a operação realizada em cada nodo, pois cada operação elementar conhece a própria regra de derivada local; (3) os valores intermediários calculados no *forward* que as derivadas locais consomem. Exemplos do item (3): o nodo de produto $z = x \cdot y$ precisa guardar os valores das entradas, pois no *backward* cada entrada recebe o gradiente multiplicado pelo valor da outra ($\partial z/\partial x = y$ e $\partial z/\partial y = x$); o nodo sigmoide precisa guardar a própria saída, pois $\sigma'(z) = \sigma(z)(1-\sigma(z))$; o nodo de máximo precisa registrar qual entrada venceu para rotear o gradiente inteiro para ela.

21) a. Com $Z = A\,\theta^\top$, o nodo *matmul* segue a mesma regra do nodo $\times$ escalar: cada entrada recebe o gradiente *upstream* multiplicado pela outra entrada, com as transpostas ajustando as dimensões. Conferindo: $G\,\theta$ é $(N \times d_s)(d_s \times d_e) = N \times d_e$, a forma de A, e $G^\top A$ é $(d_s \times N)(N \times d_e) = d_s \times d_e$, a forma de θ. A alternativa (b) aplica a regra do produto sem transposta $C = AB$, ignorando a convenção $Z = A\,\theta^\top$, e produz até uma multiplicação indefinida ($G\,\theta^\top$ tenta multiplicar $N \times d_s$ por $d_e \times d_s$); em (c) as duas expressões estão trocadas e nenhuma tem a forma da entrada correspondente; em (d) nenhuma das multiplicações é definida em geral.

22) a) *Forward pass*, nodo a nodo:

$$Z^{(1)} = X{\theta^{(1)}}^\top = \begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}0.5 & 0.5\\ -0.5 & 0.5\end{bmatrix} = \begin{bmatrix}-0.5 & 1.5\end{bmatrix}$$
$$A^{(1)} = \sigma(Z^{(1)}) = \begin{bmatrix}0.3775 & 0.8176\end{bmatrix}$$
$$Z^{(2)} = A^{(1)}{\theta^{(2)}}^\top = 0.5 \cdot 0.3775 - 0.5 \cdot 0.8176 = -0.2200$$
$$\hat{y} = \sigma(-0.2200) = 0.4452$$
$$J = \tfrac{1}{2}(1 - 0.4452)^2 = 0.1539$$

b) *Backward pass*, iniciando com $\partial J/\partial J = 1$:

- Nodo *loss*: $\frac{\partial J}{\partial \hat{y}} = \hat{y} - y = -0.5548$.
- Nodo sigmoide da saída: $\delta^{(2)} = \frac{\partial J}{\partial \hat{y}} \odot \hat{y}(1-\hat{y}) = -0.5548 \cdot 0.2470 = -0.1370$.
- Nodo *matmul* da camada 2: $\frac{\partial J}{\partial \theta^{(2)}} = {\delta^{(2)}}^\top A^{(1)} = \begin{bmatrix}-0.0517 & -0.1120\end{bmatrix}$ e $\frac{\partial J}{\partial A^{(1)}} = \delta^{(2)}\theta^{(2)} = \begin{bmatrix}-0.0685 & 0.0685\end{bmatrix}$.
- Nodo sigmoide da camada oculta: $\delta^{(1)} = \frac{\partial J}{\partial A^{(1)}} \odot A^{(1)} \odot (1 - A^{(1)}) = \begin{bmatrix}-0.0685 & 0.0685\end{bmatrix} \odot \begin{bmatrix}0.2350 & 0.1491\end{bmatrix} = \begin{bmatrix}-0.0161 & 0.0102\end{bmatrix}$.
- Nodo *matmul* da camada 1: $\frac{\partial J}{\partial \theta^{(1)}} = {\delta^{(1)}}^\top X = \begin{bmatrix}-0.0161\\ 0.0102\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}-0.0161 & -0.0322\\ 0.0102 & 0.0204\end{bmatrix}$.

X e y são dados do problema, então não recebem gradiente.

23) c. Estimar o gradiente com uma única observação caracteriza o *Stochastic Gradient Descent*. O *Batch Gradient Descent* usa o conjunto de treino inteiro em cada passo e o *Mini Batch* usa subconjuntos com mais de uma observação; *Deterministic Gradient Descent* não é uma variação padrão do algoritmo.

24) b. A equação define uma média móvel exponencial dos gradientes: $\mathcal{V}_t$ acumula o histórico com peso β e o gradiente atual com peso $1 - \beta$, que é exatamente o *momentum*. Não há termo de penalidade sobre os pesos (L2) nem normalização de ativações (*batch normalization*) na expressão.

25) c, d. A oscilação indica passos grandes ou ruidosos demais. Diminuir a taxa de aprendizado reduz o tamanho do passo e permite assentar no vale (c), e aumentar o *batch size* coloca mais exemplos em cada estimativa do gradiente, tornando os passos mais estáveis (d). Obter mais dados (a) não muda a dinâmica do passo, e aumentar a taxa de aprendizado (b) agrava a oscilação.

26) b. Com 0.1 a *loss* cai rápido mas se comporta de forma instável (taxa alta demais) e com 10^{-6} o decaimento é lento demais. O próximo palpite razoável fica entre os dois extremos em escala logarítmica, como 10^{-4}. Testar 0.2 (a) aumentaria a instabilidade e 10^{-8} (c) seria ainda mais lento que o caso já inviável.

27) c. O *scheduler* interage com o equilíbrio entre otimizar bem o conjunto de treino e regularizar o modelo, e esse equilíbrio muda quando os dados mudam; por isso o *scheduler* é específico do conjunto de dados. O número de observações (a) e a distribuição alvo (b), isoladamente, não capturam essa dependência.

28) a, c. O ruído da estimativa do gradiente com *batches* menores atua como regularização implícita, dificultando a memorização do conjunto de treino (a) e ajudando a escapar de mínimos ruins (c). A alternativa (b) é falsa pois *batches* menores aproveitam pior o paralelismo da GPU, (d) afirma o oposto de (a) e (e) ignora o efeito do *batch size* na solução encontrada.

29) O gradiente aponta na direção de maior crescimento local da função de custo, então atualizar os pesos na direção contrária diminui o custo. Formalmente, pela expansão de primeira ordem, $J(\theta - \eta\nabla_\theta J) \approx J(\theta) - \eta \|\nabla_\theta J\|^2$, e como $\|\nabla_\theta J\|^2 \geq 0$, um passo suficientemente pequeno na direção $-\nabla_\theta J$ reduz o valor de J.

30) Dizemos que o gradiente está atrasado pois, devido ao *momentum*, já existe uma atualização dos pesos conforme o histórico dos gradientes. Essa observação leva a definição do *Nesterov Accelerated Momentum*: como a atualização vai deslocar os pesos pelo termo de histórico βv_t de qualquer forma, é mais informativo avaliar o gradiente no ponto para onde os pesos estão indo (o ponto *lookahead*) do que no ponto atual, e o Nesterov faz exatamente essa correção.

31) Ao normalizar o gradiente, fica mais fácil definir uma taxa de aprendizado que funcione igualmente bem para todos os parâmetros da rede. Sem normalização, parâmetros com gradientes de escalas muito diferentes exigiriam taxas de aprendizado diferentes; ao dividir cada componente pela raiz do segundo momento acumulado, o passo efetivo de cada parâmetro fica com magnitude comparável, próxima de η.

32) Uma iteração consiste na apresentação de algumas instâncias para o modelo, sob as quais são realizados *Forward Pass*, cálculo da função de custo, *Backward Pass* e otimização. Uma época ocorre quando foram feitas iterações suficientes para ver todas as instâncias do conjunto de treinamento 1 vez.

33)

$$\eta = 0.1, \quad \theta_0 = \begin{bmatrix} 5 \\ 10 \end{bmatrix}, \quad v_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$
$$\nabla_{\theta_0} J = \begin{bmatrix} -2 \\ 5 \end{bmatrix}, \quad \nabla_{\theta_1} J = \begin{bmatrix} -1 \\ 8 \end{bmatrix}, \quad \nabla_{\theta_2} J = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

a)

$$\theta_1 = \begin{bmatrix} 5 \\ 10 \end{bmatrix} - 0.1 \begin{bmatrix} -2 \\ 5 \end{bmatrix} = \begin{bmatrix} 5.2 \\ 9.5 \end{bmatrix}$$

b)

$$\theta_1 = \begin{bmatrix} 5 \\ 10 \end{bmatrix} - 0.1 \left(\begin{bmatrix} -2 \\ 5 \end{bmatrix} \div \begin{bmatrix} 2 \\ 5 \end{bmatrix} \right) = \begin{bmatrix} 5.1 \\ 9.9 \end{bmatrix}$$

c)

$$v_1 = 0.8 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + (1 - 0.8) \begin{bmatrix} -2 \\ 5 \end{bmatrix} = \begin{bmatrix} -0.4 \\ 1 \end{bmatrix}$$
$$\theta_1 = \begin{bmatrix} 5 \\ 10 \end{bmatrix} - 0.1 \begin{bmatrix} -0.4 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.04 \\ 9.9 \end{bmatrix}$$
$$v_2 = 0.8 \begin{bmatrix} -0.4 \\ 1 \end{bmatrix} + (1 - 0.8) \begin{bmatrix} -1 \\ 8 \end{bmatrix} = \begin{bmatrix} -0.52 \\ 2.40 \end{bmatrix}$$
$$\theta_2 = \begin{bmatrix} 5.04 \\ 9.9 \end{bmatrix} - 0.1 \begin{bmatrix} -0.52 \\ 2.40 \end{bmatrix} = \begin{bmatrix} 5.092 \\ 9.660 \end{bmatrix}$$

d) ☠️ Considerando as regras de atualização do Adam com $\eta = 0.1$, $\beta = 0.9$ e $\gamma = 0.999$, podemos calcular o valor de θ_2.

Primeiro, calculamos θ_1 para a iteração $t = 1$:

Momento (m_1):

$$m_1 = \beta m_0 + (1-\beta)\nabla_{\theta_0} J = 0.9 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + (1-0.9) \begin{bmatrix} -2 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.1 \begin{bmatrix} -2 \\ 5 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}$$

Termo normalizador (v_1):

$$v_1 = \gamma v_0 + (1-\gamma)(\nabla_{\theta_0} J)^2 = 0.999 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + (1-0.999) \begin{bmatrix} (-2)^2 \\ 5^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.001 \begin{bmatrix} 4 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.004 \\ 0.025 \end{bmatrix}$$

Correção de viés (*bias correction*):

$$\hat{m}_1 = \frac{m_1}{1-\beta^1} = \frac{\begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}}{1-0.9} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}$$

$$\hat{v}_1 = \frac{v_1}{1-\gamma^1} = \frac{\begin{bmatrix} 0.004 \\ 0.025 \end{bmatrix}}{1-0.999} = \begin{bmatrix} 4 \\ 25 \end{bmatrix}$$

Atualização de θ_1:

$$\theta_1 = \theta_0 - \eta \frac{\hat{m}_1}{\sqrt{\hat{v}_1} + \epsilon} = \begin{bmatrix} 5 \\ 10 \end{bmatrix} - 0.1 \begin{bmatrix} \frac{-2}{\sqrt{4}} \\ \frac{5}{\sqrt{25}} \end{bmatrix} = \begin{bmatrix} 5 \\ 10 \end{bmatrix} - 0.1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.1 \\ 9.9 \end{bmatrix}$$

Agora, calculamos θ_2 para a iteração $t = 2$:

Momento (m_2):

$$m_2 = \beta m_1 + (1-\beta)\nabla_{\theta_1} J = 0.9 \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix} + (1-0.9) \begin{bmatrix} -1 \\ 8 \end{bmatrix} = \begin{bmatrix} -0.18 \\ 0.45 \end{bmatrix} + 0.1 \begin{bmatrix} -1 \\ 8 \end{bmatrix} = \begin{bmatrix} -0.28 \\ 1.25 \end{bmatrix}$$

Termo normalizador (v_2):

$$v_2 = \gamma v_1 + (1-\gamma)(\nabla_{\theta_1} J)^2 = 0.999 \begin{bmatrix} 0.004 \\ 0.025 \end{bmatrix} + (1-0.999) \begin{bmatrix} (-1)^2 \\ 8^2 \end{bmatrix} = \begin{bmatrix} 0.003996 \\ 0.024975 \end{bmatrix} + 0.001 \begin{bmatrix} 1 \\ 64 \end{bmatrix} = \begin{bmatrix} 0.004996 \\ 0.088975 \end{bmatrix}$$

Correção de viés (*bias correction*):

$$\hat{m}_2 = \frac{m_2}{1-\beta^2} = \frac{\begin{bmatrix} -0.28 \\ 1.25 \end{bmatrix}}{1-0.9^2} = \frac{\begin{bmatrix} -0.28 \\ 1.25 \end{bmatrix}}{0.19} \approx \begin{bmatrix} -1.47368 \\ 6.57895 \end{bmatrix}$$

$$\hat{v}_2 = \frac{v_2}{1-\gamma^2} = \frac{\begin{bmatrix} 0.004996 \\ 0.088975 \end{bmatrix}}{1-0.999^2} = \frac{\begin{bmatrix} 0.004996 \\ 0.088975 \end{bmatrix}}{0.001999} \approx \begin{bmatrix} 2.49925 \\ 44.50975 \end{bmatrix}$$

Atualização de θ_2:

$$\theta_2 = \theta_1 - \eta \frac{\hat{m}_2}{\sqrt{\hat{v}_2} + \epsilon} = \begin{bmatrix} 5.1 \\ 9.9 \end{bmatrix} - 0.1 \begin{bmatrix} \frac{-1.47368}{\sqrt{2.49925}} \\ \frac{6.57895}{\sqrt{44.50975}} \end{bmatrix} \approx \begin{bmatrix} 5.1 \\ 9.9 \end{bmatrix} - 0.1 \begin{bmatrix} -0.9322 \\ 0.9869 \end{bmatrix} = \begin{bmatrix} 5.1932 \\ 9.8013 \end{bmatrix}$$

Portanto, o valor de θ_2 ao seguir a regra de atualização do Adam é:

$$\theta_2 \approx \begin{bmatrix} 5.193 \\ 9.801 \end{bmatrix}$$

34) e. A softmax normaliza os *logits* para que as saídas sejam positivas e somem 1, produzindo uma distribuição de probabilidade sobre classes mutuamente exclusivas, que é o que o problema multiclasse exige. As demais ativações produzem valores por unidade sem a restrição de somarem 1, então não formam uma distribuição sobre as classes.

35) b. No problema *multilabel* cada classe é uma decisão binária independente, então cada saída precisa de uma probabilidade própria em $[0, 1]$, o que a sigmoid fornece. A softmax (e) forçaria as classes a competir entre si, pois as probabilidades somariam 1, o que impede múltiplas classes simultâneas.

36) b, g. Sigmoid em cada unidade de saída fornece probabilidades independentes por classe, e a *Binary Cross Entropy* avalia cada uma dessas decisões binárias separadamente. A combinação softmax + *Categorical Cross Entropy* seria a escolha para o caso multiclasse com classes mutuamente exclusivas, que não é o cenário do enunciado.

37) a, c. Tanh e sigmoid saturam para entradas de magnitude grande e têm derivadas menores que 1 (a da sigmoid vale no máximo 0.25); no *backpropagation* essas derivadas se multiplicam camada após camada e o gradiente encolhe exponencialmente. A ReLU (b) tem derivada 1 na região ativa, e o *batch normalization* (d) é uma técnica que combate o problema em vez de causá-lo.

38) a, d. Na região ativa ($x > 0$) a derivada da ReLU é constante igual a 1, então ela não satura como a sigmoid e minimiza o *vanishing gradient* (a). Ela é contínua em todo o domínio, mas não é diferenciável em $x = 0$, onde as derivadas laterais diferem (d), o que exclui (b) e (c).

39) d. A ReLU é uma função não linear aplicada às pré-ativações, e é isso que impede a rede de colapsar em um modelo linear. Convoluções (a) são operações lineares, e taxa de aprendizado (b), embaralhamento dos dados (c) e regularização (e) afetam o treinamento, não a classe de funções que a rede consegue representar.

40) (a) Uma única unidade é suficiente. A função alvo é linear nos atributos, e um neurônio computa exatamente uma combinação linear das entradas mais um viés: basta fazer $\theta_1 = 10000$, $\theta_2 = 50000$ e $\theta_0 = 3500$.

(b) A ativação linear (identidade). O problema é de regressão e a saída deve poder assumir qualquer valor real; uma ativação não linear na saída restringiria ou distorceria a imagem da função modelada.

41) Exemplos de características desejáveis (citar e explicar duas): ser não linear, pois é o que permite que a composição de camadas represente funções que um modelo linear não representa; ser diferenciável (ou quase em todo ponto), pois o treinamento depende da propagação de gradientes; não saturar em grande parte do domínio, pois derivadas próximas de zero causam *vanishing gradient*; ter baixo custo computacional, pois a ativação é avaliada em todas as unidades a cada *forward* e *backward*.

42) Porque sem não linearidade entre as camadas a composição de transformações lineares colapsa em uma única transformação linear ($\theta^{(2)}(\theta^{(1)}X) = (\theta^{(2)}\theta^{(1)})X$). Nesse caso a rede, por mais profunda que seja, teria o mesmo poder de representação de um modelo linear. As ativações não lineares quebram essa equivalência e permitem aproximar funções complexas.

43) a)

$$\varphi'(x) = \begin{cases} 0, & x \leq 0, \\ 1, & x > 0 \end{cases}$$

b) $\varphi'(x) = \varphi(x)(1 - \varphi(x))$

c) ☠️

$$\frac{\partial S_i}{\partial x_j} = \begin{cases} S_i(1 - S_i), & i = j, \\ -S_i S_j, & i \neq j. \end{cases}$$

Forma compacta: $\frac{\partial S_i}{\partial x_j} = S_i\,(\delta_{ij} - S_j)$, onde δ_{ij} é o delta de Kronecker.

44) Adicionar não linearidades. Sem elas, a composição das camadas colapsaria em uma única transformação linear e a profundidade não acrescentaria poder de representação; com ativações não lineares nas camadas ocultas, cada camada pode compor funções progressivamente mais complexas.

45) Ajustar a saída da rede neural à imagem da função modelada: a ativação da última camada mapeia as pré-ativações para o domínio que a tarefa exige, como $[0, 1]$ com a sigmoid para probabilidade binária, uma distribuição que soma 1 com a softmax no caso multiclasse, ou os reais com a ativação linear em regressão. Essa escolha também casa a saída com a função de custo correspondente.

46) a) $2(\hat{y} - y)$

b) $\frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$ (esta é $\partial L_{BCE}/\partial \hat{y}$, não $\partial L_{BCE}/\partial \theta$; ao multiplicar pela derivada da sigmoid $\hat{y}(1 - \hat{y})$ e por $X^\top$, simplifica para $X^\top(\hat{y} - y)$)

47) Aplicando o logaritmo negativo à verossimilhança da Gaussiana com $\mu = \hat{y}^{(i)}$: $-\ln Pr(y^{(i)}|\hat{y}^{(i)}, \sigma^2) = \frac{1}{2}\ln(2\pi\sigma^2) + \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$. O primeiro termo e o fator $\frac{1}{2\sigma^2}$ não dependem de $\hat{y}$, então maximizar a verossimilhança equivale a minimizar $L_{l2} = \sum \left(y^{(i)} - \hat{y}^{(i)}\right)^2$.

48) Aplicando o logaritmo negativo à verossimilhança da Bernoulli com $\lambda = \hat{y}$: $-\ln\left[(1 - \hat{y})^{(1-y)}\,\hat{y}^{\,y}\right] = -y\,ln(\hat{y}) - (1 - y)\,ln(1 - \hat{y}) = L_{BCE}$.

49) ☠️ $-\sum_i cos(y^{(i)} - f(x^{(i)}, \theta))$ (a NLL é $-\sum_i \ln \Pr(y^{(i)} \mid \mu, k)$. O denominador $2\pi \cdot \text{Bessel}_0[k]$ e o k não dependem de $\mu = f(x^{(i)}, \theta)$, então caem como constantes; sobra apenas o termo $-k\sum_i \cos(y^{(i)} - f(x^{(i)}, \theta))$, que com k fixo é proporcional ao apresentado)

50) ☠️ $\sum_{i=1}^{I}(f[x^{(i)}, \theta] - y^{(i)}ln(f[x^{(i)}, \theta]))$ (a NLL de cada observação é $-\ln Pr(y^{(i)}) = -y^{(i)}\ln\lambda + \lambda + \ln(y^{(i)}!)$; com $\lambda = f[x^{(i)}, \theta]$, o termo $\ln(y^{(i)}!)$ não depende de θ e cai como constante, restando a soma apresentada)

51) c. Com todos os pesos iguais, todas as unidades de uma mesma camada computam a mesma pré-ativação e a mesma saída; no *backward*, recebem gradientes idênticos e são atualizadas de forma idêntica. A simetria nunca é quebrada e a camada se comporta como se tivesse uma única unidade. Note que os gradientes não são nulos (a): o problema não é ausência de atualização, e sim atualização idêntica em todas as unidades.

52) b. Pesos com variância grande produzem pré-ativações de magnitude grande, que colocam a sigmoid nas regiões saturadas onde a derivada é praticamente zero; no *backpropagation* isso dissipa o gradiente antes de chegar às camadas iniciais. Unidades mortas (a) são um fenômeno da ReLU, não da sigmoid, e (c) descreve o cenário oposto, de σ pequeno demais.

53) *Vanishing Gradient*: a sigmoid satura para entradas de magnitude grande e sua derivada vale no máximo 0.25; ao multiplicar essas derivadas através das camadas no *backpropagation*, o gradiente encolhe exponencialmente e as primeiras camadas praticamente param de aprender.

54) Unidades mortas (*dead units*): se os pesos e o bias de uma unidade a levam a produzir pré-ativação negativa para todas as entradas do conjunto, a saída da ReLU é sempre zero e sua derivada também; o gradiente que passa pela unidade é nulo e ela nunca mais é atualizada.

55) Inicializar os pesos da rede neural de modo que não ocorra explosão nem dissipação de gradiente. A ideia central é escolher a variância dos pesos de forma que a variância das pré-ativações (e, por consequência, dos gradientes) se preserve de camada para camada: a He usa $\sigma_\theta^2 = 2/n^{(l-1)}$ para compensar o fator $\frac{1}{2}$ que a ReLU introduz no segundo momento, e a Xavier usa $\sigma_\theta^2 = 1/n^{(l-1)}$ para ativações aproximadamente lineares perto da origem, como a tanh.

56) Pela definição da ReLU,

$$E[\mathrm{ReLU}(a)^2] = \int_{-\infty}^{\infty} \max(0,a)^2\, p(a)\, da = \int_0^{\infty} a^2\, p(a)\, da.$$

Pela simetria de p em torno de zero, $\int_{-\infty}^{0} a^2\, p(a)\, da = \int_0^{\infty} a^2\, p(a)\, da$, então $E[a^2] = 2\int_0^{\infty} a^2\, p(a)\, da$. Como $E[a] = 0$, temos $E[a^2] = \mathrm{Var}[a] = \sigma^2$, e portanto $E[\mathrm{ReLU}(a)^2] = \frac{E[a^2]}{2} = \frac{\sigma^2}{2}$.

57) a) Pela linearidade da esperança, $E[z_i^{(l)}] = E[b_i^{(l)}] + \sum_j E[\theta_{ij}^{(l)}\, \varphi(z_j^{(l-1)})]$. O bias é inicializado em zero e, pela independência entre pesos e ativações, cada termo do somatório fatora em $E[\theta_{ij}^{(l)}]\, E[\varphi(z_j^{(l-1)})] = 0 \cdot E[\varphi(z_j^{(l-1)})] = 0$. Logo $E[z_i^{(l)}] = 0$.

b) Como $E[z_i^{(l)}] = 0$, vale $\sigma^2_{z^{(l)}} = E[(z_i^{(l)})^2]$. Ao expandir o quadrado do somatório, os termos cruzados desaparecem pela independência dos pesos e por $E[\theta] = 0$, restando $\sigma^2_{z^{(l)}} = \sum_j E[(\theta_{ij}^{(l)})^2]\, E[\varphi(z_j^{(l-1)})^2] = \sigma_\theta^2 \sum_j E[\varphi(z_j^{(l-1)})^2]$. Pelo exercício anterior, $E[\mathrm{ReLU}(z_j^{(l-1)})^2] = \sigma^2_{z^{(l-1)}}/2$ para cada uma das $n^{(l-1)}$ unidades, o que dá $\sigma^2_{z^{(l)}} = \frac{n^{(l-1)}}{2}\, \sigma_\theta^2\, \sigma^2_{z^{(l-1)}}$.

c) Impondo $\sigma^2_{z^{(l)}} = \sigma^2_{z^{(l-1)}}$ na recorrência, a constante multiplicativa deve valer 1: $\frac{n^{(l-1)}}{2}\, \sigma_\theta^2 = 1$, ou seja, $\sigma_\theta^2 = \frac{2}{n^{(l-1)}}$, que é a inicialização *He*.

d) Com $\varphi(z) \approx z$, o segundo momento da ativação passa a ser $E[\varphi(z)^2] \approx \sigma^2_{z^{(l-1)}}$, sem o fator $\frac{1}{2}$ da ReLU. A recorrência vira $\sigma^2_{z^{(l)}} = n^{(l-1)}\, \sigma_\theta^2\, \sigma^2_{z^{(l-1)}}$ e a condição de preservação dá $\sigma_\theta^2 = \frac{1}{n^{(l-1)}}$, que é a inicialização *Xavier*.

58) a, b. Basta substituir na fórmula: com $\alpha = 1$ o termo *one-hot* é anulado e sobra $\frac{1}{C}$ em todas as classes, o alvo uniforme (a); com $\alpha = 0$ a fórmula devolve o alvo *one-hot* original, sem nenhuma suavização (b), o que também mostra que (d) está invertida. O valor de α (c) é um hiperparâmetro de intensidade da suavização, não uma função da quantidade de observações do conjunto.

59) b. A penalidade L1 empurra componentes pequenos até exatamente zero (os vértices do losango da bola L1 ficam sobre os eixos), induzindo esparsidade; a L2 encolhe as magnitudes sem zerá-las, o que exclui (a). Na visão probabilística, L1 corresponde a um *prior* laplaciano e L2 ao gaussiano, o que exclui (c); e ambas diminuem a magnitude dos pesos, o contrário de (d).

60) c. Durante o treino cada unidade fica ativa com probabilidade $1 - p$, então a magnitude esperada das ativações que as camadas seguintes aprenderam a receber é $(1 - p)$ vezes a magnitude sem *dropout*. Com todas as unidades ativas na inferência, multiplicar as ativações por $(1 - p)$ restaura essa escala. Multiplicar por p (a) corrige na direção errada, e (b) ignora que o *dropout* altera a escala do próprio *forward*.

61) $\theta_{(t+1)} = (1 - 2\eta\lambda)\theta_t - \eta\nabla_\theta J$, onde λ é a constante de regularização L2.

62) a) Como a máscara é independente do valor da ativação e $E[m] = 1 - p$, temos $E[m \cdot a] = E[m]\, a = (1 - p)\, a$. Em inferência a unidade está sempre ativa e produz a, ou seja, uma magnitude $\frac{1}{1-p}$ vezes maior do que a magnitude esperada que as camadas seguintes viram durante o treino.

b) *Rescaling* determinístico: multiplicar as ativações por $(1 - p)$ na inferência, trazendo a escala de teste para a escala esperada do treino. *Inverted dropout*: dividir as ativações por $(1 - p)$ já durante o treino, de modo que a magnitude esperada fique igual à de inferência e nenhum ajuste seja necessário ao testar; é o padrão em PyTorch e Keras.

63) O *early stopping* monitora o desempenho da rede em um conjunto de validação ao longo do treinamento e interrompe o processo antes que a *loss* de validação volte a subir, ainda que a *loss* de treino continue caindo. O hiperparâmetro *patience* define quantas iterações sem melhoria na validação são toleradas antes de parar, evitando interromper o treino por flutuações passageiras. O efeito é análogo ao da regularização L2 porque, ao limitar o número de atualizações, os pesos não têm tempo de crescer indiscriminadamente e permanecem com magnitude contida, assim como o *weight decay* os mantém próximos da origem. O benefício adicional é a economia de tempo de computação, já que o treinamento termina mais cedo.