Giải thích cú pháp (vi-style concise)

Mục đích: Xây dựng một mô hình neural network tuần tự bằng TensorFlow/Keras để phân loại ảnh (ví dụ MNIST 28×28).

Cấu trúc máyfangen (Sequential)

model = tf.keras.Sequential([ ... ])

Tạo một mô hình tuần tự, tức là các lớp được xếp chồng lên nhau theo thứ tự đã cho.

Ý nghĩa: dữ liệu chạy từ lớp này sang lớp kế tiếp một cách tuyến tính.

Các lớp (layers)

tf.keras.layers.Flatten(input\_shape=(28, 28))

Mục đích: chuyển đổi hình ảnh 2D 28×28 thành vector 1D có 784 phần tử.

input\_shape=(28, 28) là kích thước đầu vào của mỗi mẫu.

Ghi chú: không có tham số học tập ở lớp này; chỉ thay đổi hình dạng dữ liệu.

tf.keras.layers.Dense(128, activation='relu')

Mục đích: lớp ẩn đầy đủ (fully connected) với 128 neuron.

activation='relu' áp dụng hàm kích hoạt ReLU: relu(x)=max⁡(0,x)relu(x)=max(0,x).

Ý nghĩa: học được 128 đặc trưng phi tuyến từ đầu vào đã flatten.

tf.keras.layers.Dense(10)

Mục đích: lớp đầu ra gồm 10 neuron, mỗi neuron sẽ đại diện cho một lớp phân loại.

Ở đây chưa chỉ định activation, nên lớp này cho đầu ra dạng logit (giá trị tuyến tính) cho mỗi lớp.

Ghi chú: thường dùng với loss/số lượng lớp tương ứng (ví dụ 10 cho MNIST, sau đó có-softmax ở bước compile hoặc trong loss).

Kết luận ngắn gọn

Mô hình nhận ảnh 28×28, flatten thành vector 784, qua một lớp ẩn 128 neuron với activation ReLU, rồi ra 10 đầu ra logits cho các lớp.

Đọc như dòng chảy dữ liệu: input → Flatten → Dense(128, ReLU) → Dense(10) (logits).

Gợi ý thêm (nếu bạn wants)

Compile: thêm bước model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), metrics=['accuracy']) để huấn luyện.

Huấn luyện: model.fit(train\_images, train\_labels, epochs=…, validation\_data=(val\_images, val\_labels)).

Nếu muốn dùng softmax ở đầu ra ngay trong layer, có thể làm Dense(10, activation='softmax') và đổi loss tương ứng.

Bạn muốn tôi thêm phần hoàn thiện đầy đủ với compile và huấn luyện cho dataset cụ thể nào không?

TÍNH TOÁN BẰNG CÔNG THỨC SAO

Sider Fusion

Bạn đang hỏi “tính toán bằng công thức sao”—mình sẽ giải thích ngắn gọn cách viết công thức cho mô hình mạng neural được nêu trước đó, theo đúng ngữ cảnh tính toán với TensorFlow/Keras.

Cấu trúc mô hình (Sequential)

Input: hình ảnh 28×28

Các bước tiền xử lý (tùy chọn): Flatten

Lớp ẩn: Dense với 128 neuron, activation ReLU

Lớp đầu ra: Dense với 10 neuron (logits)

Các công thức liên quan:

Flatten

Kích thước đầu vào: 28×28 28×28

Đầu ra sau Flatten: 784 784 phần tử (vector hàng)

Công thức ánh xạ: với một ảnh X∈R28×28X∈R28×28,

Flatten(X)∈R784,moˆ˜i phaˆˋn tử laˋ Xi,j.Flatten(X)∈R784,moˆ˜i phaˆˋn tử laˋ Xi,j​.

Lớp Dense (Fully Connected)

Số lượng neuron: 128

Hàm kích hoạt: ReLU

Đầu ra của lớp ẩn: một vector h∈R128h∈R128

Công thức tuyến tính (trước kích hoạt):

z=Wx+b,W∈R128×784,  b∈R128z=Wx+b,W∈R128×784,b∈R128

Với x=Flatten(X)x=Flatten(X)

Sau đó áp dụng ReLU:

hi=ReLU(zi)=max⁡(0,zi),i=1,…,128hi​=ReLU(zi​)=max(0,zi​),i=1,…,128

Lớp Dense đầu ra

Số neuron: 10 (tương ứng 10 lớp/nhãn)

Đầu ra logits: l∈R10l∈R10

Công thức tuyến tính:

wpsKhông có activation đồng nghĩa với việc đây là logits (trước softmax).

Phần mềm và Loss (gợi ý thêm)

Nếu dùng loss cho phân loại 10 lớp với logits:

\mathcal{L} = \text{SparseCategoricalCrossentropy(from\_logits=True)}

Nếu muốn đồng thời đưa softmax ở lớp đầu ra:

python

Dense(10, activation='softmax')

Khi đó loss có thể là CategoricalCrossentropy.

Gợi ý nhanh để huấn luyện

python

model.compile(

optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy']

)

model.fit(train\_images, train\_labels, epochs=10, validation\_data=(val\_images, val\_labels))

Bạn muốn mình viết rõ thêm phần tính toán gradient hay backpropagation cho từng lớp không? Hoặc áp dụng cho một dataset cụ thể (ví dụ MNIST) và đi kèm công thức cập nhật tham số?

CÓ VIẾT CHI TIẾT CHO TÔI CÁCH GIẢI VÀ TÍNH TOÁN TRONG BÀI TOÀN PHÂN LOẠI ẢNH NÀY

Sider Fusion

Hướng dẫn chi tiết về cách giải và tính toán cho bài toán phân loại ảnh với mô hình đã cho

Bạn đang làm việc với một mô hình neural network đơn giản theo cấu trúc Sequential:

Input: hình ảnh 28×28

Flatten thành vector 784

Dense(128, activation='relu')

Dense(10) và sử dụng logits (không có activation ở lớp đầu ra)

Dưới đây là cách giải và tính toán từng bước, từ forward pass đến backpropagation và cập nhật tham số.

1) Chuẩn bị và notation

Gọi ảnh đầu vào x∈R28×28x∈R28×28 và sau Flatten ta được x∈R784x∈R784.

Lớp ẩn có 128 neuron.

Trước kích hoạt của lớp ẩn: z(1)=W(1)x+b(1)z(1)=W(1)x+b(1), với

W(1)∈R128×784W(1)∈R128×784

b(1)∈R128b(1)∈R128

Sau kích hoạt ReLU: a(1)=ReLU(z(1))a(1)=ReLU(z(1)), với

ReLU(t)=max⁡(0,t)ReLU(t)=max(0,t)

Lớp đầu ra có 10 neuron (logits):

Trước kích hoạt của lớp đầu ra: z(2)=W(2)a(1)+b(2)z(2)=W(2)a(1)+b(2), với

W(2)∈R10×128W(2)∈R10×128

b(2)∈R10b(2)∈R10

Đầu ra: z(2)∈R10z(2)∈R10 (logits)

2) Forward pass (thuật toán lan truyền tiến)

Cho một mẫu đơn có nhãn thật y∈{0,1,…,9}y∈{0,1,…,9}:

Flatten và chuẩn bị đầu vào:

x←Flatten(ảnh X)∈R784x←Flatten(ảnh X)∈R784

Lớp ẩn:

z(1)=W(1)x+b(1)z(1)=W(1)x+b(1)a(1)=ReLU(z(1))a(1)=ReLU(z(1))

Lớp đầu ra (logits):

z(2)=W(2)a(1)+b(2)z(2)=W(2)a(1)+b(2)y^=z(2)y^​=z(2)

Đây là vector logit cho 10 lớp, không softmax ở đây nếu bạn dùng từ loss từ\_logits=True.

Loss (ví dụ với sparse labels):

Nếu sử dụng Sparse Categorical Crossentropy từ\_logits=True:

L(y^,y)=−log⁡(ey^y∑j=09ey^j)=−y^y+log⁡(∑j=09ey^j)L(y^​,y)=−log(∑j=09​ey^​j​ey^​y​​)=−y^​y​+log(j=0∑9​ey^​j​)

Ghi chú: từ\_logits=True cho stabilization và thể hiện rõ bản chất logits.

Softmax (nếu muốn):

Nếu bạn đưa softmax ở đầu ra:

p=softmax(y^),pj=ey^j∑k=09ey^kp=softmax(y^​),pj​=∑k=09​ey^​k​ey^​j​​

Lúc này loss phổ biến là CategoricalCrossentropy với đối số from\_logits=False.

3) Backpropagation (lan truyền ngược) và cập nhật tham số

Mục tiêu: tối ưu hóa tham số Θ={W(1),b(1),W(2),b(2)}Θ={W(1),b(1),W(2),b(2)}.

Quy trình tổng quát:

Tính loss LL từ dự đoán và nhãn thật.

Tính gradient của loss với mọi tham số:

∂L∂W(2)∂W(2)∂L​, ∂L∂b(2)∂b(2)∂L​

∂L∂a(1)∂a(1)∂L​, sau đó

∂L∂z(1)∂z(1)∂L​, và cuối cùng

∂L∂W(1)∂W(1)∂L​, ∂L∂b(1)∂b(1)∂L​

Cập nhật tham số theo quy tắc tối ưu:

W(l)←W(l)−η∂L∂W(l),b(l)←b(l)−η∂L∂b(l)W(l)←W(l)−η∂W(l)∂L​,b(l)←b(l)−η∂b(l)∂L​

với ηη là learning rate.

Cụ thể more detail:

Lớp 2 (đầu ra):

∂L∂W(2)=∂L∂z(2)⋅(a(1))⊤∂W(2)∂L​=∂z(2)∂L​⋅(a(1))⊤∂L∂b(2)=∂L∂z(2)∂b(2)∂L​=∂z(2)∂L​

Với từ\_logits=True và loss là SparseCategoricalCrossentropy,:

∂L∂z(2)=p^−y∂z(2)∂L​=p^​−y

trong đó p^p^​ là phân phối softmax của z(2)z(2) hoặc tương ứng với logit nếu từ\_logits=True và y là one-hot vector của nhãn thực.

Lớp 1 (ẩn):

∂L∂W(1)=((W(2))⊤∂L∂z(2)⊙ReLU′(z(1)))x⊤∂W(1)∂L​=((W(2))⊤∂z(2)∂L​⊙ReLU′(z(1)))x⊤∂L∂b(1)=((W(2))⊤∂L∂z(2)⊙ReLU′(z(1)))∂b(1)∂L​=((W(2))⊤∂z(2)∂L​⊙ReLU′(z(1)))

ReLU derivative:

ReLU′(t)={1,t>00,t≤0ReLU′(t)={1,0,​t>0t≤0​

Lưu ý: ⊙⊙ là tích phần tử (element-wise).

Ghi chú thực tế khi dùng Keras:

Ta không cần tính tay các gradient nếu dùng model.compile(...) và model.fit(...).

Keras tự động tính gradient, áp dụng gradient descent/Adam, và thực hiện cập nhật tham số.

4) Quy trình học (ví dụ thực tế)

python

import tensorflow as tf

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10) # logits

])

model.compile(

optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy']

)

# Giả sử có dữ liệu train\_images (shape: [N,28,28]) và train\_labels (shape: [N])

model.fit(train\_images, train\_labels, epochs=10, validation\_data=(val\_images, val\_labels))

Loss:

Với loss SparseCategoricalCrossentropy(from\_logits=True), gradient được tính cho logits trực tiếp.

Cập nhật tham số:

Sử dụng optimizer, ví dụ Adam, với các tham số học tập riêng (learning rate, momentums, v.v.).

5) Công thức tổng quát cho từng bước cập nhật tham số

Lớp Dense đầu ra:

z(2)=W(2)a(1)+b(2)z(2)=W(2)a(1)+b(2)∂L∂W(2)=∂L∂z(2)(a(1))⊤∂W(2)∂L​=∂z(2)∂L​(a(1))⊤∂L∂b(2)=∂L∂z(2)∂b(2)∂L​=∂z(2)∂L​

Lớp Flatten và Lớp Dense ẩn:

z(1)=W(1)x+b(1),a(1)=ReLU(z(1))z(1)=W(1)x+b(1),a(1)=ReLU(z(1))∂L∂W(1)=((W(2))⊤∂L∂z(2)⊙ReLU′(z(1)))x⊤∂W(1)∂L​=((W(2))⊤∂z(2)∂L​⊙ReLU′(z(1)))x⊤∂L∂b(1)=((W(2))⊤∂L∂z(2)⊙ReLU′(z(1)))∂b(1)∂L​=((W(2))⊤∂z(2)∂L​⊙ReLU′(z(1)))

Update tham số với learning rate ηη:

W(l)←W(l)−η∂L∂W(l),b(l)←b(l)−η∂L∂b(l)W(l)←W(l)−η∂W(l)∂L​,b(l)←b(l)−η∂b(l)∂L​

6) Gợi ý tối ưu và thực tế

Để ổn định và nhanh convergence:

Sử dụng tối ưu như Adam hoặc RMSprop.

Chuẩn hóa dữ liệu đầu vào (ví dụ: chia 255 để đưa về [0,1]).

Sử dụng bộ dữ liệu đủ lớn và phân tách train/validation.

Đối với dataset MNIST:

10 nhãn (0–9).

Image shape: (28, 28, 1) hoặc (28, 28) sau Flatten.

Nếu bạn muốn:

Thêm dropout để giảm overfitting.

Thay đổi kích thước lớp ẩn (ví dụ Dense(256), Dense(64)).

Thêm lớp ở giữa như Dense(64) với activation.

Bạn muốn đi sâu vào phần nào?

Tính toánGradient thẳng tay cho một ví dụ cụ thể với một mẫu?

Hiển thị bước cập nhật tham số theo một vòng lặp training thực tế?

Áp dụng cho một dataset cụ thể (ví dụ MNIST) và đi kèm lời giải chi tiết với các số liệu ví dụ?

Cho mình biết bạn muốn chi tiết ở mức độ nào (vòng lặp một sample hay cả epoch, số liệu ví dụ cụ thể) để mình trình bày rõ ràng hơn.