

Introduction to Machine Learning

Day 10: MLPs in PyTorch

Joanna Bieri DATA301

Get out your handwritten notes!

Mini Review - Three New Things

REVIEW

Last class we built one neuron by hand and taught it with gradient descent. Today we do three new things with that loop.

- 1 **More neurons.** We put neurons side by side in a **layer**, and stack layers into a **multilayer perceptron (MLP)**.
- 2 **PyTorch's tools.** PyTorch has ready made pieces for the layers, the loss and the update step.
- 3 **Mini-batches.** Instead of using every row for every step, we use a small batch of rows at a time.

Mini Review - A Number to Beat

REVIEW

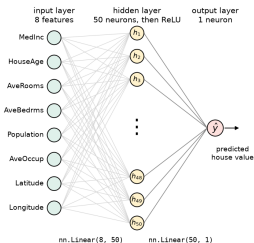
- The California housing data: each row is one small neighborhood. We predict the **median house value**, in units of \$100,000.
- **RMSE** (root mean squared error) is in the same units as the label, so an RMSE of 0.5 means the predictions are typically off by about \$50,000.

linear regression validation RMSE: 0.7356

So a straight line is typically off by about \$73,600. Can a neural network do better?

Mini Review - From One Neuron to a Layer

REVIEW



- `nn.Linear(8, 1)` is **one neuron** with 8 inputs.
- `nn.Linear(8, 50)` is **50 neurons side by side**, which is called a **layer**. Each neuron gets all 8 inputs and has its own 8 weights and its own bias.

Mini Review - A Multilayer Perceptron

REVIEW

```
model = nn.Sequential(  
    nn.Linear(8, 50),      # hidden layer: 50 neurons  
    nn.ReLU(),             # the activation  
    nn.Linear(50, 1))      # output layer: 1 prediction
```

- **ReLU** keeps positive numbers and turns negative numbers into 0.
- If we stack two linear layers with nothing between them, the result is still a straight line. A weighted sum of weighted sums is just another weighted sum.

Mini Review - Training with PyTorch's Tools

REVIEW

step	last class, by hand	today
clear	<code>w.grad.zero_()</code>	<code>optimizer.zero_grad()</code>
predict	<code>torch.sigmoid(X @ w + b)</code>	<code>model(X)</code>
loss	<code>F.binary_cross_entropy(p, y)</code>	<code>loss_fn(y_pred, y)</code>
backward	<code>loss.backward()</code>	<code>loss.backward()</code>
update	<code>w -= lr * w.grad</code>	<code>optimizer.step()</code>

Same five steps. The loss function measures how wrong we are, and the optimizer does the update for every weight at once.

Mini Review - Mini-Batches

REVIEW

- We use a small **batch** of rows for each step, and a different batch for the next step.
- A trip through all of the batches, so every row gets used once, is called an **epoch**.

```
train_loader = DataLoader(train_data,  
                           batch_size=32, shuffle=True)
```

```
for epoch in range(n_epochs):  
    for X_batch, y_batch in train_loader:  
        ...the same five steps, on one batch of 32 rows
```

15,480 training rows in batches of 32 is 484 steps per epoch.

Mini Review - Comparing the Three Models

REVIEW

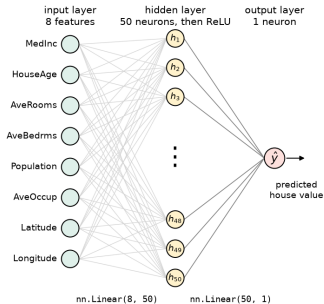
validation RMSE

model	RMSE	in dollars
linear regression	0.7356	\$73,600
MLP, every row, 1000 steps	0.602	\$60,200
MLP, batches of 32, 20 epochs	0.6067	\$60,700

Both networks beat the straight line by about \$13,000.

From One Neuron to a Network

ASK



Every line in this picture is one weight. How many numbers does this network have to learn?

Hint. Count the weights and the biases in each `nn.Linear`.

From One Neuron to a Network

ANSWER

Every line in this picture is one weight. How many numbers does this network have to learn?

```
nn.Linear(8, 50): 8 x 50 + 50 = 450
nn.Linear(50, 1): 50 x 1 + 1 = 51
                    total 501
```

Training means finding good values for all 501 numbers. A wider or deeper network has many more.

Why the Activation Matters

ASK

We predict house values.

Why is there no activation
after the output layer?

Hint. What numbers can a sigmoid give? What about ReLU?

Why the Activation Matters

ANSWER

We predict house values. Why is there no activation after the output layer?

- A sigmoid squashes every answer between 0 and 1, so a house value of 2.5 (\$250,000) would be impossible.
- ReLU could never predict below 0, and it gives no gradient when its input is negative.
- For regression the last layer is a plain weighted sum, so it can give any number.

The Training Loop, PyTorch's Way

ASK

Put the five steps in order
and say what each one does.

Hint. `step()`, `backward()`, `predict`, `zero_grad()`, `loss`

The Training Loop, PyTorch's Way

ANSWER

Put the five steps of the training loop in order, and say what each one does.

```
optimizer.zero_grad()           # clear
y_pred = model(X_batch)         # predict
loss = loss_fn(y_pred, y_batch) # how wrong?
loss.backward()                 # gradients
optimizer.step()                # step downhill
```

We will use this loop for every network for the rest of the semester.

The Training Loop, PyTorch's Way

ASK

Your loss prints nan
after a few steps.

What happened? What do you try first?

Hint. nan means "not a number". Think about the size of each step.

The Training Loop, PyTorch's Way

ANSWER

Your loss prints nan. What happened, and what do you try first?

- The steps were so big that the weights grew until the numbers were too big for the computer to hold.
- Try a smaller learning rate first (0.5 gave nan, 0.1 worked).
- Also check: unscaled features, a missing zero_grad, missing values in the data.

Mini-Batches

ASK

Why use small batches
instead of every row
at every step?

Hint. How many updates does one pass through the data give you each way?

Mini-Batches

ANSWER

Why use small batches instead of every row at every step?

- Every row at every step: one update per pass. It took 1000 passes to get to 0.602.
- Batches of 32: 484 updates per pass. After one pass it already beat linear regression, and 20 passes got to 0.607.
- Big data may not even fit in memory all at once.

Mini-Batches

ASK

Why do we shuffle the rows
every epoch?

Hint. What if the rows were sorted by house value?

Mini-Batches

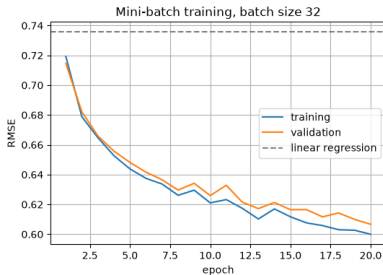
ANSWER

Why do we shuffle the rows every epoch?

- Sorted rows make one batch all cheap houses and the next all expensive, so each step pulls the model to one extreme.
- Shuffling makes every batch a small random sample.
- This randomness is the “stochastic” in SGD.

Did It Work?

ASK



Is the network worth it? Is it overfitting?

Hint. Find the dashed line. Then compare the two curves to each other.

Did It Work?

ANSWER

Look at the learning curve. Is the network worth it? Is it overfitting?

- Worth it: it beats linear regression after one epoch, and it trains in a few seconds.
- Not overfitting: after 20 epochs training is 0.600 and validation is 0.6067, and the curves stay close.
- Both curves are still going down slowly, so more epochs could help a little.

Questions?

Questions?