Neural Net Training with PyTorch

CS 759/859 Natural Language Processing Lecture 8

Samuel Carton, University of New Hampshire

Last lecture

Linear regression

- Learn $Wx + b$ from data
- Predict continuous values
- Optimize mean squared error

Logistic regression

- Learn $\sigma(Wx + b)$ from data
- Predict (close to) 0 or 1
- Optimize cross-entropy

Key concepts:

- Loss function
  - I.e. objective function
- Gradient of loss with respect to parameters
- Gradient descent
- Activation function

PyTorch

PyTorch is a **deep learning library**
- Define the structure of a neural net
- Use gradient descent to train it
- Implementations of common structural elements

PyTorch
- Created and maintained by Meta
- Competes primarily with TensorFlow (Google)
- Fairly dominant in research right now

All deep learning libraries are basically a lego kit for **tensor** operations

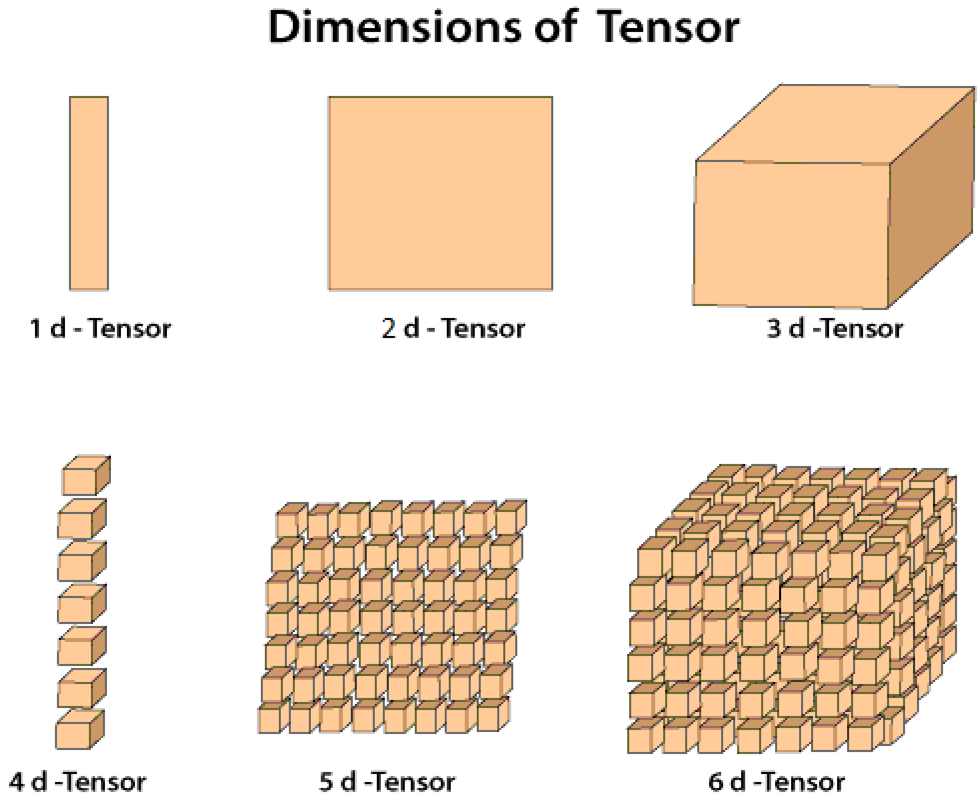
Tensors

A tensor is an N-dimensional array of values

- e.g. a scalar (0D), vector (1D), or matrix (2D)

Any neural net is basically just a bunch of tensor operations

GPUs happen to be good at doing tensor operations quickly



https://www.javatpoint.com/pytorch-tensors

Visualizing logistic regression

Recall our visualization of logistic regression as a matrix (i.e tensor) operation

$$\sigma\left(\begin{bmatrix} x_0^0 & x_0^1 & x_0^2 & \cdots & x_0^N \\ x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^N \\ & \ldots & & & \\ x_M^0 & x_M^1 & x_M^2 & \cdots & x_M^N \end{bmatrix}\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{bmatrix} + b\right) = \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_M \end{bmatrix}$$

Basics of PyTorch tensors

Code description

- Demonstrations of basic PyTorch Tensor operations, including arithmetic, dimension inspection, and converting back and forth between Numpy arrays and PyTorch tensors

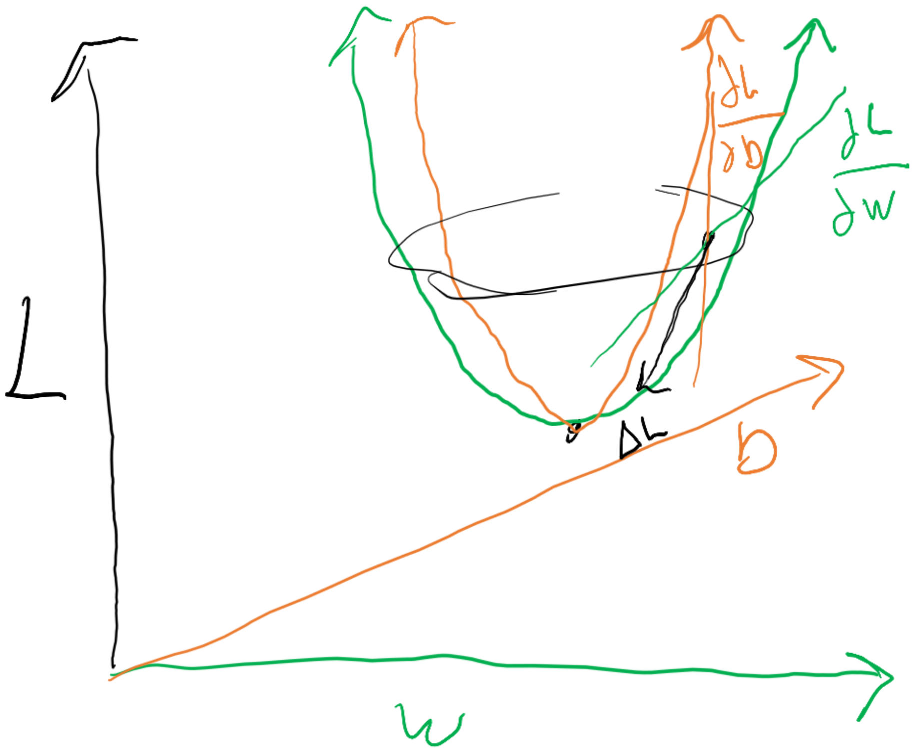
Notebook headings

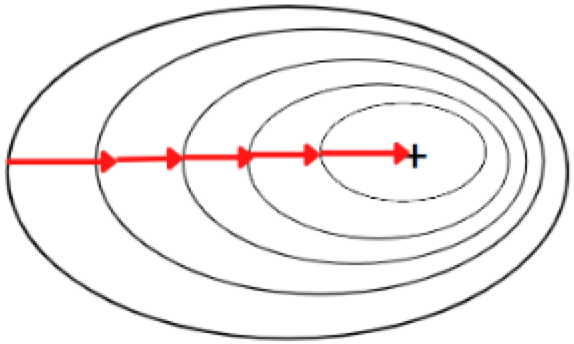
Tensors

- Basic operations
- Different dimensionalities
- Convenient functionality

Gradient Descent

Basic idea: Calculate the loss over the **whole training set**, do a step along the gradient, then recalculate the loss and so on





https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/

Mini-batch gradient descent

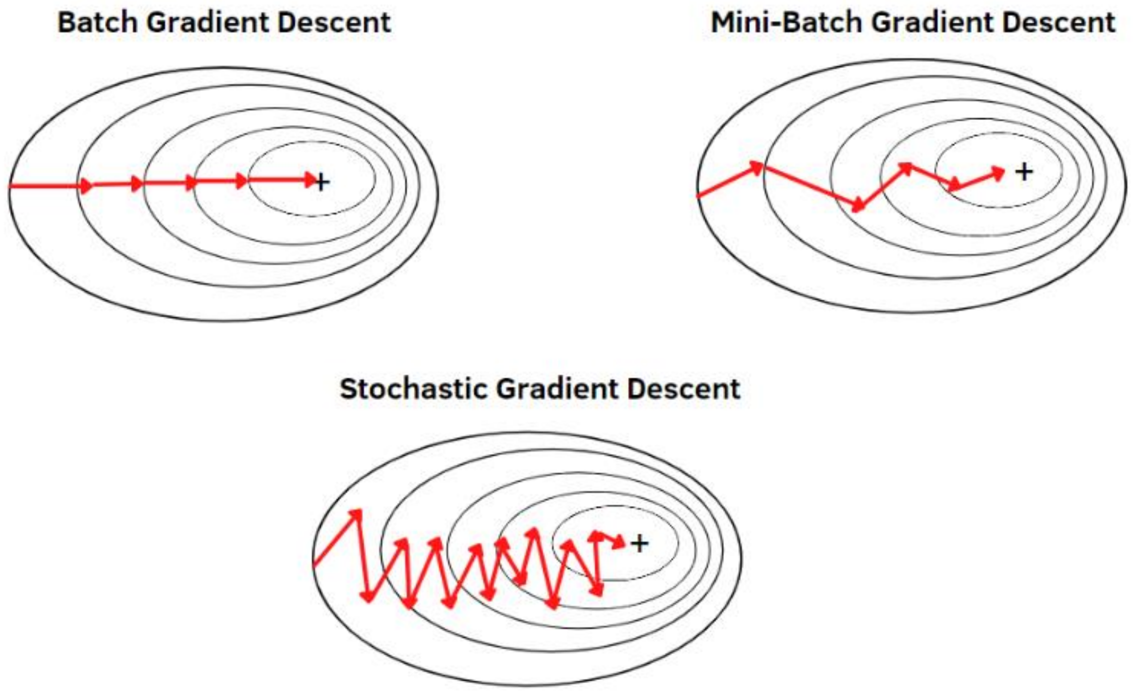
For big datasets/models, we can't fit all training gradients in memory.

So we do our steps on **batches** of the data, one at a time

When the batch size is 1, it's called **stochastic gradient descent**

Batch size is a **hugely important** hyperparameter in neural net training.

- Bigger usually better, but requires a bigger GPU
- Why Nvidia A100s are like $15k



Reading and preprocessing SST-2

Code description

- Reading, preprocessing and vectorizing SST-2. Should all be familiar at this point

Notebook headings

Reading and preprocessing SST-2 dataset

PyTorch Datasets and DataLoaders

PyTorch modules prefer to work with PyTorch **Datasets** and **DataLoaders**

A Pytorch Dataset

- Will extend torch.utils.data.Dataset
- Will primarily know how to yield one (x,y) item, given an index

A PyTorch DataLoader

- Will extend torch.utils.data.DataLoader
- Will know how to iterate over batches of items

For more info: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

PyTorch Datasets and DataLoaders

Code description

- Demonstration of PyTorch Dataset and DataLoader classes

Notebook headings

PyTorch Datasets and DataLoaders

Dataset

Dataloader

PyTorch models

PyTorch models always extend torch.nn.Module

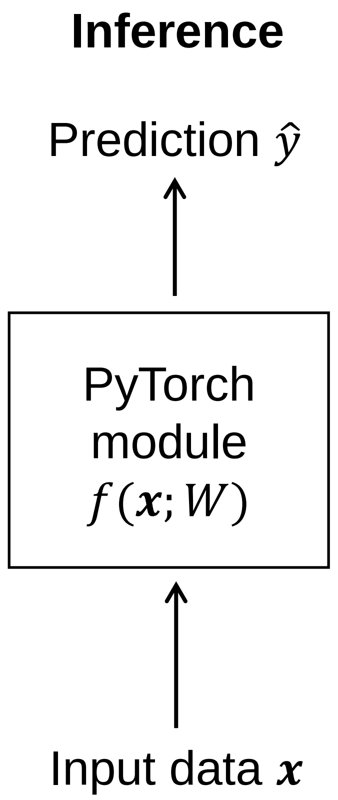
They always have:

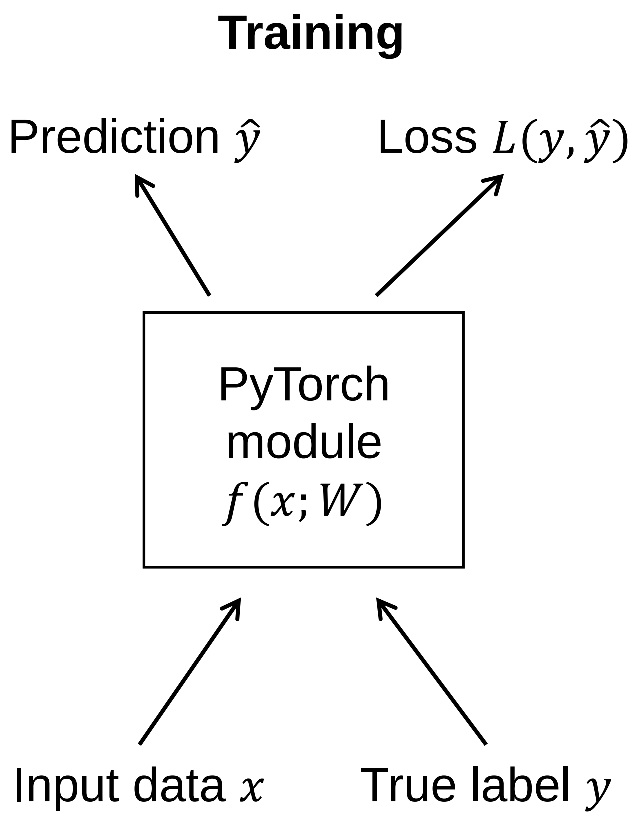
- An `__init__()` method which defines the structure of the model
- A `forward()` method which takes in the input and spits out the model output

As long as the output of forward() is composed of differentiable tensor-on-tensor operations, then PyTorch can use **automatic differentiation** to figure out $\Delta_{parameters} output$, and then subsequently do gradient descent.

PyTorch models

A PyTorch model is essentially a wrapper around its forward() function, taking in an input tensor $\boldsymbol{x}$ and producing a prediction $\hat{y}$





Pytorch Models

Code description

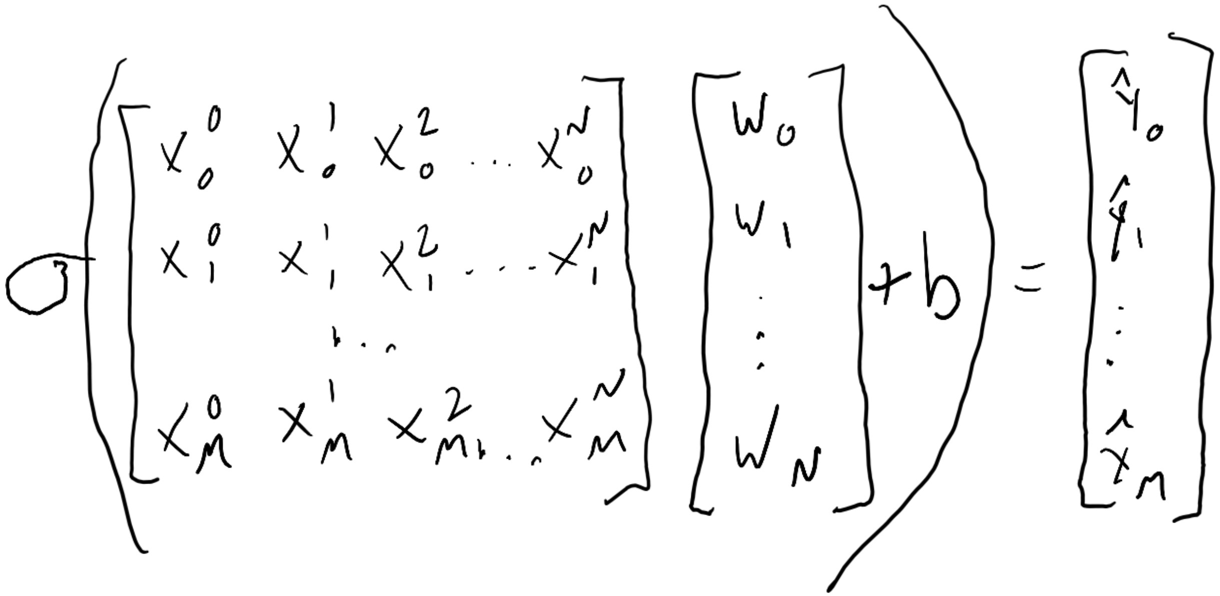
- Example of a PyTorch model for binary logistic regression

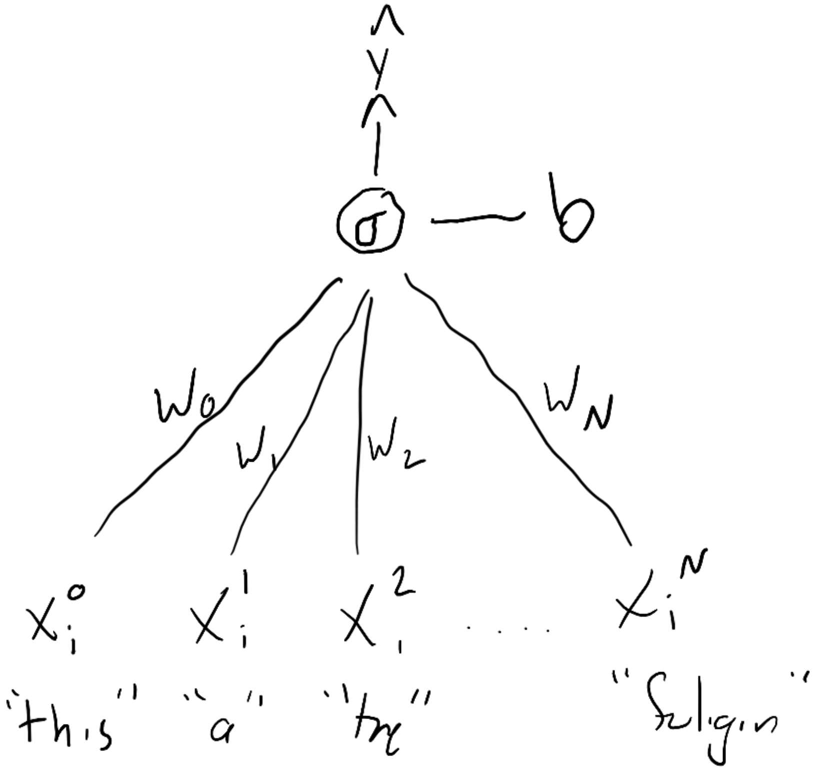
Notebook headings

Our PyTorch model

Visualizing logistic regression

You can do the same thing for logistic regression by adding the σ function





PyTorch training loop

Basic pseudocode:

For each epoch:

    For each training batch:

        Zero the accumulated grads

        Run model on training batch

        Calculate loss

        Perform gradient descent on step

    (optional)

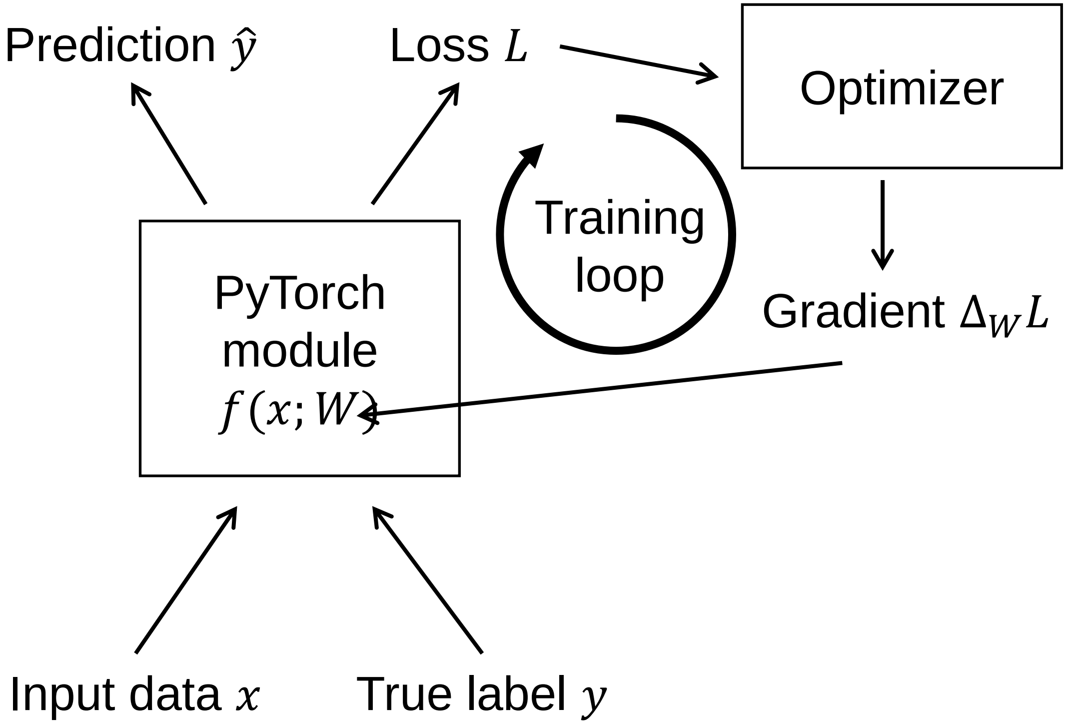
    For each validation batch:

        Run model on validation batch

    Report overall validation accuracy

PyTorch training loop

A PyTorch model is essentially a wrapper around its forward() function, taking in an input tensor x and producing a prediction y



PyTorch training loop

Code description

- Example of PyTorch training loop, including an inner loop for performing dev set evaluation

Notebook headings

Training loop

L1/L2 Regularization

Basic idea: discourage any one feature from having too much of an impact on the model output by punishing the sum (L1) or squared-sum (L2) of the model parameters

Standard way to discourage overfitting

Done automatically by most scikit-learn models

```
# We can see here that "kangaroo" totally sabotaged the prediction here
explain_binary_linear_model_prediction('the movie was wonderful and had a kangaroo in it.',
                                        lin_reg_model,
                                        vectorizer)
```

```
Prediction: -0.698151062283332
Word coefficients:
        Word: the - Coef: -0.005
        Word: movi - Coef: 0.002
        Word: wa - Coef: -0.079
        Word: wonder - Coef: 0.184
        Word: and - Coef: 0.024
        Word: had - Coef: -0.061
        Word: kangaroo - Coef: -1.353
        Word: in - Coef: -0.015
        Word: it - Coef: 0.003
Model intercept: 0.6021082139311205
```

Regularized PyTorch model

Code description

- Example of construction and training of a PyTorch model with L2 regularization on the weight vector

Notebook headings

Regularized model

Early stopping

Basic idea: Keep an eye on the development set performance (either loss or accuracy), and stop the training loop early when the improvement seems to level off

- Often save model checkpoints only on improvement, and then reload best checkpoint at the end of training

Another way to avoid overfitting

https://neptune.ai/blog/early-stopping-with-neptune

Early stopping in PyTorch

Code description

- Example of a training loop that has been adjusted to perform early stopping based on the dev set performance

Notebook headings

Early stopping

Concluding thoughts

New tool: PyTorch

- Machine learning Legos

Mini-batch gradient descent

- Batch size very important

Training loop

Avoid overfitting by:

- Regularization
- Early stopping