



Deep Learning Workshop

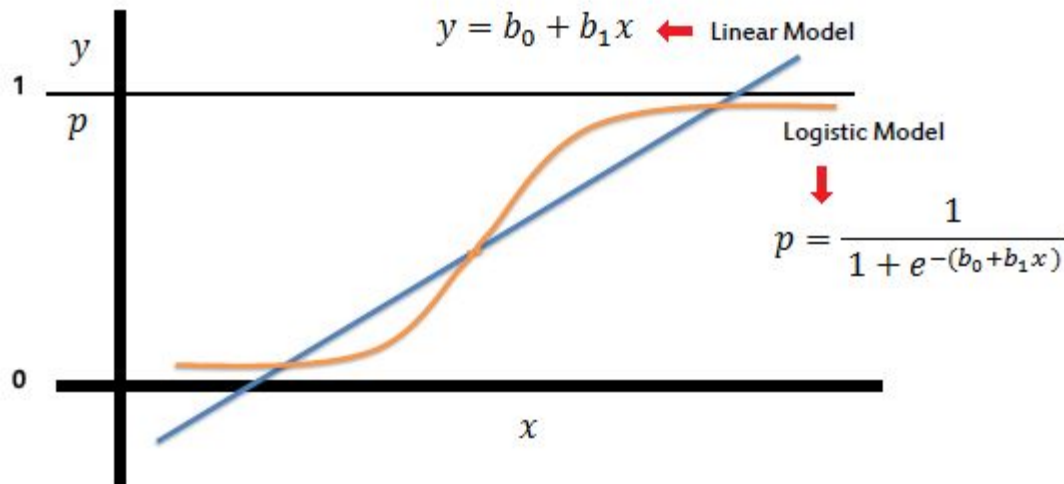
11/4/23

<https://tinyurl.com/DSCDeepLearning>

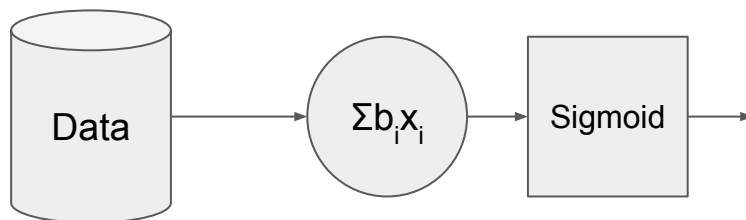
Deep Learning Review

First steps: Logistic Regression

Remember how logistic models work: They have a set of parameters (b_0, b_1) and output a probability corresponding to a certain data input (x).



Neural networks take this idea and extend it.

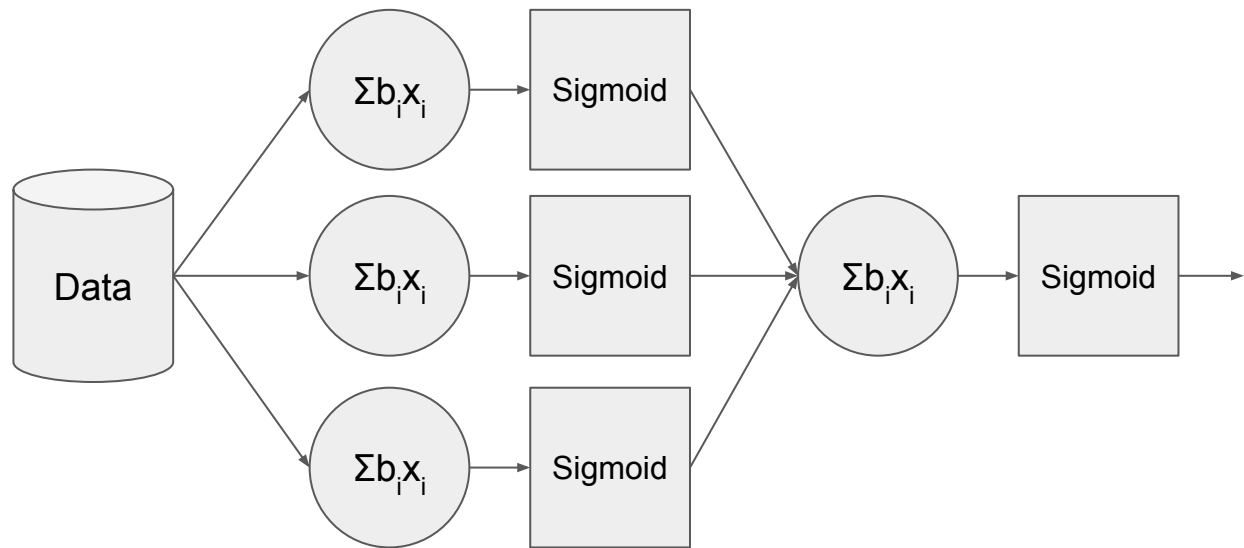


Next step: A Neural Network

We can instead have multiple logistic regressions in parallel that each have different values for b .

Now each one of these outputs can be used as the data input for a next logistic regression.

During the training process the model learns to diversify each set of b values in a way that gives more useful information.



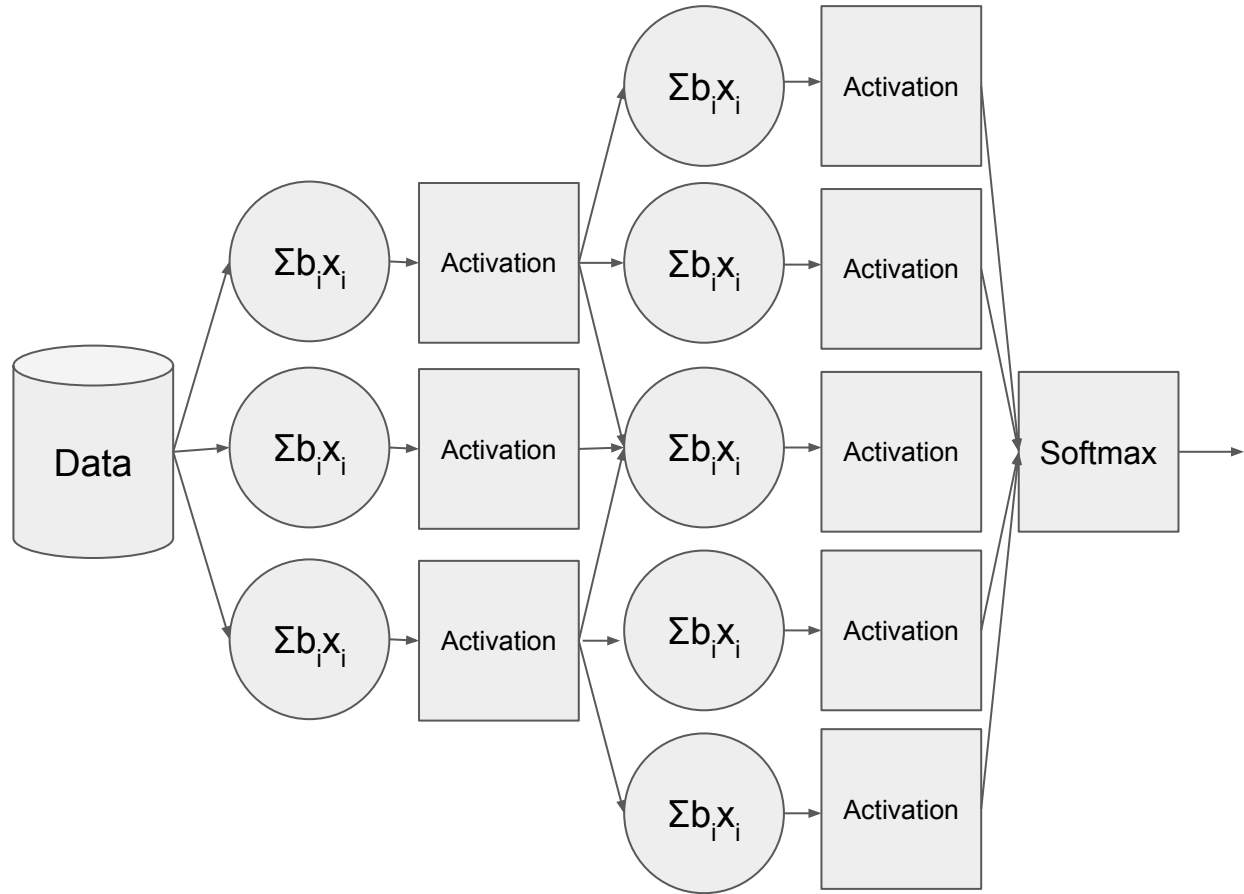
1 layer with 3 neurons

Deep Neural Network

We call this a deep neural network when we have multiple layers stacked after each other.


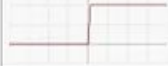







Also note that we have many choices for activation functions besides Sigmoid.

```
Layer1 = Linear(data.shape, 3)
fn1 = Activation(3)
Layer2 = Linear(3, 5)
fn2 = Activation(5)
Output = Softmax()
```



Activation Functions

Activation functions are essential for deep learning. They make our functions nonlinear, which gives our model the ability to fit to any shape function. Otherwise, our model would always reduce to linear regression.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Convolutional Neural Networks

CNNs are powerful in domains where it is applicable.

Each convolutional layer effectively scans patches of the input and applies several different filters to them.

For an interesting demonstration of CNNs, visit this site:

[ConvNetJS MNIST demo \(stanford.edu\)](https://cs.stanford.edu/people/karlenet/convnetjs/mnist_demo.html)

Later on in the notebook, you might find that the CNN model you design reaches a much higher training accuracy despite having only 1/10 of the model size.

Workshop Plan

The goal of this workshop is to help you ease into using PyTorch for training deep neural networks, and letting you challenge yourself as much as you'd like. We also try to encourage you to explore many things by looking through the documentation.

In this workshop, we will focus on the most critical concepts needed for training sizable neural networks in PyTorch, as well as the code necessary to run it. For any details or to build deeper understanding on particular things, feel free to ask us!

Regularization

If you get far enough, you will notice that a well-designed CNN model can easily reach near 100% accuracy on the training dataset, while still performing relatively poorly on the held out testing dataset. This is a result of overfitting, and regularization is a way to combat that.

In this workshop we can try

- Dropout
- BatchNorm
- Weight decay
- Data augmentation

Schedulers, Optimizers, Loss functions

Optimizers and loss functions are necessary in any training run. You have a number of options for these, but some are much better than others. Cross Entropy Loss is standard for multi class classification, for example. We use SGD for our optimizer, but feel free to experiment with others such as Adam.

Schedulers are not an absolute necessity, however, but make a major difference when training for many epochs, and typically for massive models over large datasets. We implemented one for you here, but it's likely that the trainings will never be long enough for it to be useful (less than 15 epochs).

PyTorch

PyTorch has everything we need to train deep learning models.

One important feature is the ability to send PyTorch Tensors to the GPU for faster calculations, which we make use of in this workshop.

Setup

We have worked on making the set up as easy as possible for this workshop.

We will be using Google Colab, and the dataset will be downloaded directly to Colab's storage.

Optionally, you can try out Weights & Biases by uncommenting some code we included in the training loop. This would require you to create an account and paste the api key in the notebook.

Context

We will be working on predicting road signage. Namely crosswalk signs, speed limit signs, stop signs, and traffic lights.

From this dataset, we will extract 4 channel x 256 pixel x 256 pixel images.

Note that this dataset is unbalanced.



Reference Notebook

In case you get stuck or don't have time to finish, we have a reference notebook that is ready to "Run all." You can play with both of the models here.

<https://drive.google.com/file/d/1z9X1SKIB8ORW3Jj30yAp4SjsIBqD44b5/view?usp=sharing>