PyTorch and Neural Networks

CIS 545 Recitation 10

Friday, April 7
Spring 2023

Penn Engineering

PyTorch

- Open source Machine Learning based on the Torch Library developed by Meta AI
- Allows for easier development of ML models (in particular Neural Networks) with simple APIs
- Very modular with high degree of customization

Tensors, GPU, Autograd

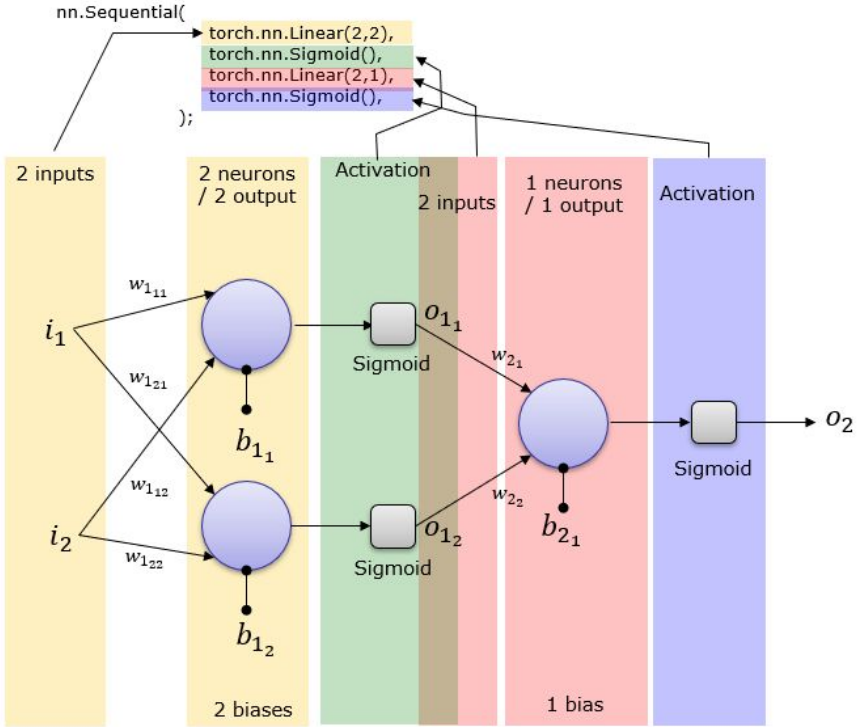
- Tensors
  - PyTorch's version of numpy ndarray
  - Many numpy array operations replicated + common matrix operations supported
- GPU
  - PyTorch supports computations on GPUs (faster computation)
  - tensor.to('cuda') or tensor.cuda()
- Autograd
  - Automatically compute the derivative of a computation w/ respect to a parameter
  - computation_result.backward()

Building Blocks

- torch.nn.Module
  - Basically a class that lets you define a set of parameters and how to take an input and do some operations with those parameters
  - forward(self, input) method actually defines how an input will interact with those parameters
- torch.optim
  - The optimizer component of PyTorch
  - Consists of various parameters that need to be optimized, usually a learning rate
  - Applies a step of optimization based on calculated gradients

torch.nn

- Various modules (you can think of them as "layers") that can be combined to produce an output
- Examples:
  - torch.nn.Linear
  - torch.nn.Conv2d
  - torch.nn.ReLU
  - torch.nn.MaxPool2d

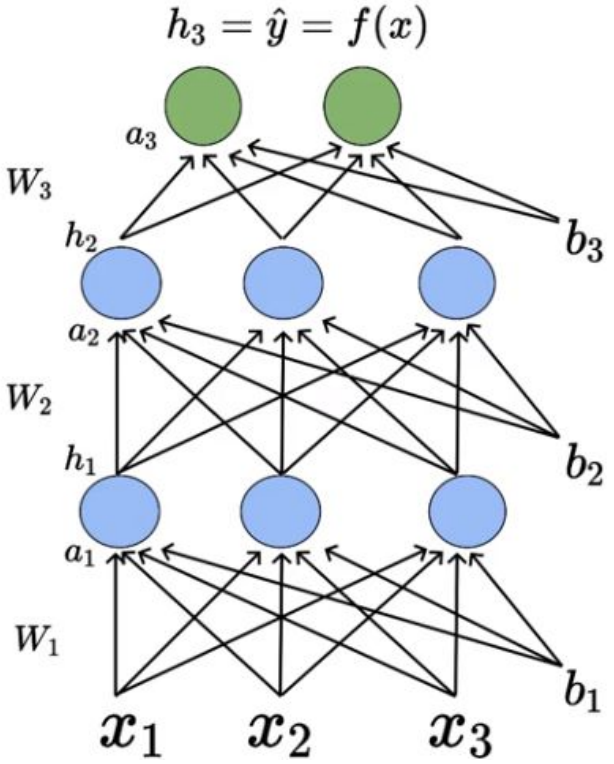


Neural Networks

- Machine Learning model consisting of multiple layers of nodes connected via weights and biases (high level)
- Multiple types for a variety of applications; we will be focusing on Feedforward and Convolutional Neural Networks
- Takes in an input (a tensor/matrix/vector) and applies a series of (nonlinear and linear transformations) to the input to produce an output
- Weights and biases updated using backpropagation, optimizer, and loss function

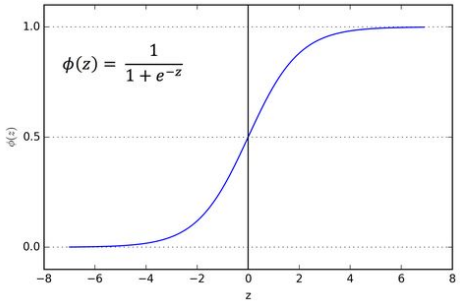
Feedforward Neural Networks

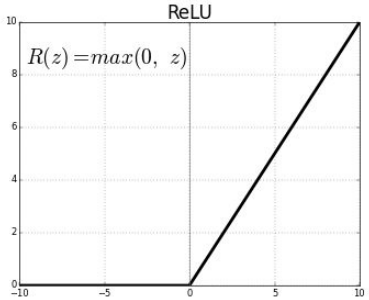
- Each neuron computes a linear sum of inputs and weights which is then passed to a non-linear activation function.
- Network consists of one or more hidden layers and each layer takes the previous layer as input.
- Network output can be a scalar or vector, depending on task
  - Binary classification: scalar probability
  - Multiclass classification: vector of probabilities



Nonlinear Activation Functions

- Common Activation Functions Used in Neural Networks:
  - Sigmoid: (generally used for output layers in case of Binary classification)
  - ReLU: (generally used for hidden layers because of faster convergence)
  - Softmax: (generally used for output layers in case of Multiclass classification)





$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Training Neural Networks

- **Set a neural network architecture:** This involves determining: the number of input nodes (based on the number of input features), number of hidden layers, number of nodes in each of the hidden layers, number of output nodes (based on the number of output classes)
- **Random Initialization of Weights:** The weights are randomly initialized to value in between 0 and 1, or rather, very close to zero.
- **Train in batches**: Faster! Batch Size = how many samples must be processed before the internal model parameters are updated)
- **Implementation of forward propagation algorithm**: calculate hypothesis function for a set on input vector for any of the hidden layer.

Training Neural Networks (continued…)

- **Implementation of cost/loss function:** The cost function would help determine how well the neural network fits the training data.
- **Implementation of back propagation algorithm:** Computes the error vector related with each of the nodes by partial differentiation of loss with respect to model parameters.
- **Implement optimization methods** (such as gradient descent or any other advanced technique): to try and minimize the cost function as a function of parameters or weights.
- **Repeat** for many iterations until loss converges!

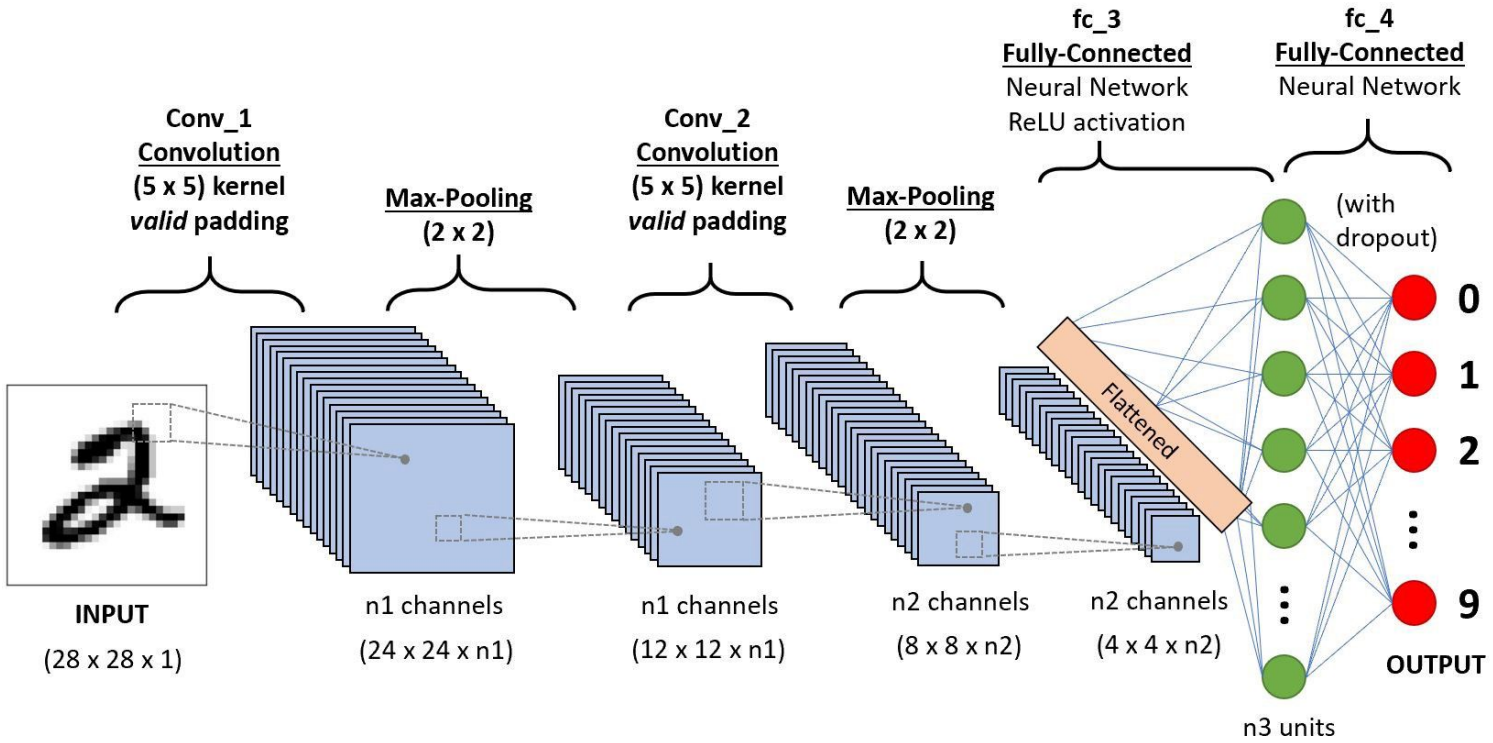
Dealing with Overfitting

- **Overfitting:** It occurs when a model learns the training dataset too well, performing well on the training dataset but does not perform well on a hold out sample.
- **Some methods to address overfitting:**
  - **Regularization:**
    - **L1:-** It adds sum of the absolute values of all weights in the model to cost function. It shrinks the coefficient of a less important feature to zero thus, removing some feature and hence providing a sparse solution
    - **L2:-** It adds sum of squares of all weights in the model to cost function. It is able to learn complex data patterns and gives non-sparse solutions unlike L1 regularization.

Dealing with Overfitting (continued…)

- **DropOut:** Dropout refers to dropping out units in a neural network (remove it temporarily from the network). The choice of which units to drop is random (a fixed probability p independent of other units). This procedure effectively generates slightly different models with different neuron topologies at each iteration, thus giving neurons in the model, less chance to coordinate in the memorisation process that happens during overfitting.
- **Early Stopping:** It implies to stop training of the model early before it reaches overfitting stage. Performance metrics (eg. accuracy, loss) can monitored for train and validation sets to implement this.
- Other methods: Data Augmentation, Batch Normalization, etc.

Convolutional Neural Networks



*Note: Sizes are different from previous slide

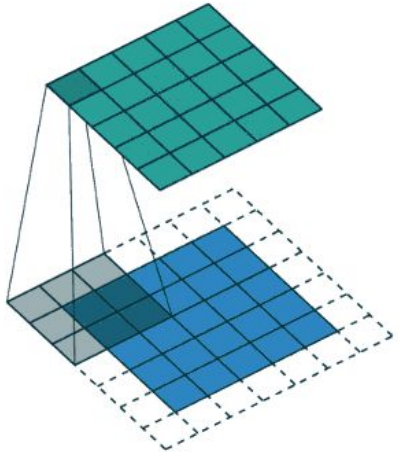
Layers of CNN

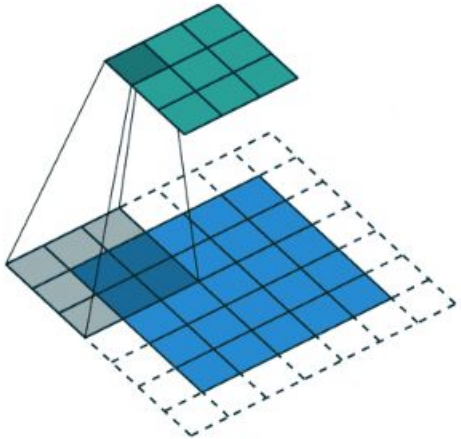
- INPUT [32x32x3] will holds raw pixel values, in this case an image of width and height 32, and color channels R,G,B.
- CONV layer will compute output of neurons connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- RELU layer will apply an elementwise activation function; the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score; each neuron in this layer will be connected to all the numbers in the previous volume.

CONV Layer

- The objective is to extract the high-level features from an image, such as the edges, corners, colors, gradient orientation, etc. which will be input for the FC layer.
- Can implement Same Padding (dimensionality >=) or Valid Padding (dimensionality decreases)

Kernel = 3x3
Pad = 1
Stride = 1

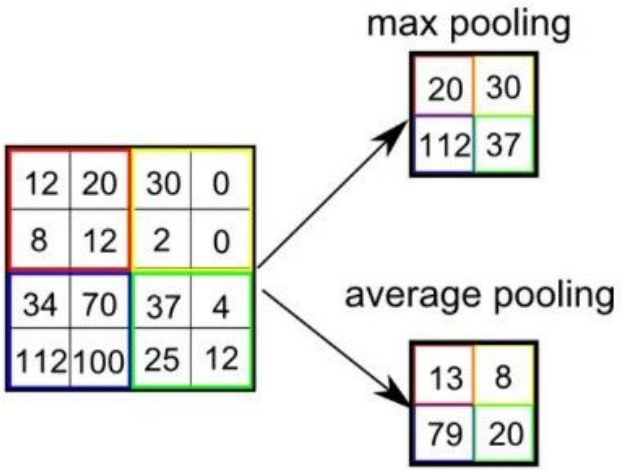


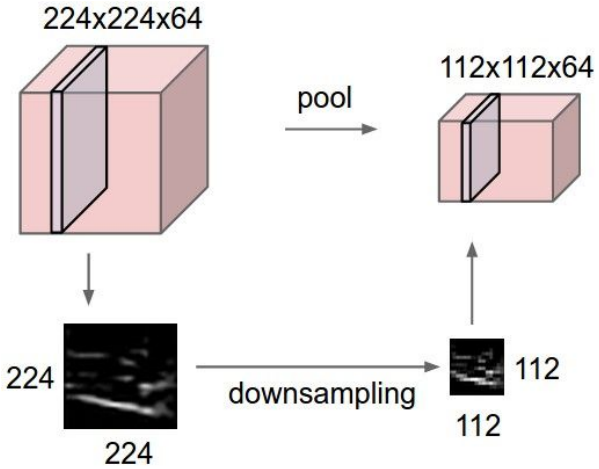


Kernel = 3x3
Pad = 1
Stride = 2

POOL Layer

- Similar to the CONV Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- This is to decrease the computational power required to process the data through dimensionality reduction.
- Can implement Max Pooling or Average Pooling.

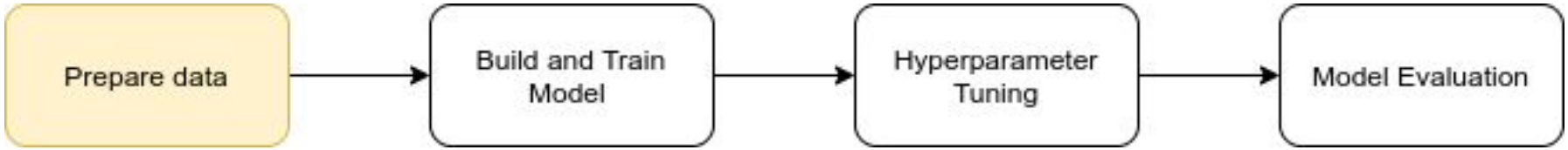




FC (Fully-Connected) Layer

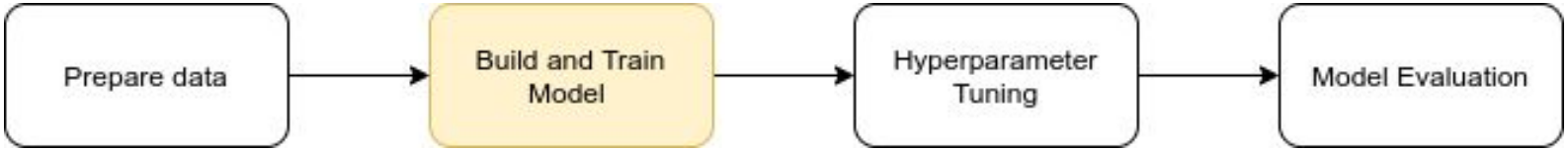
- Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer.
- Now that we have converted our input image into a suitable form, we flatten the image into a column vector -> feed it to a feed-forward neural network -> apply backpropagation to every iteration of training.
- Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them.

Model flowchart



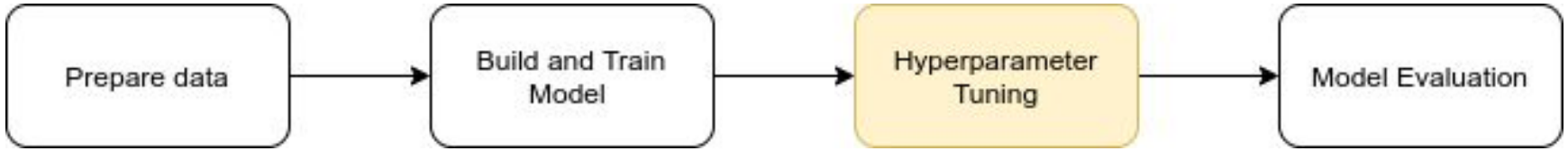
- Get data
  - PyTorch has `Dataset` object ⇒ store AND transform data
  - PyTorch has `DataLoader` object ⇒ load data in batches during training phase
- Preprocess data (images for CNN)
  - PyTorch's `torchvision.transforms` has a lot of image transformations that can be applied to the image
  - Ex. Grayscale, Reshape image to be the same size, transform from NumPy array into a PyTorch tensor

Model flowchart

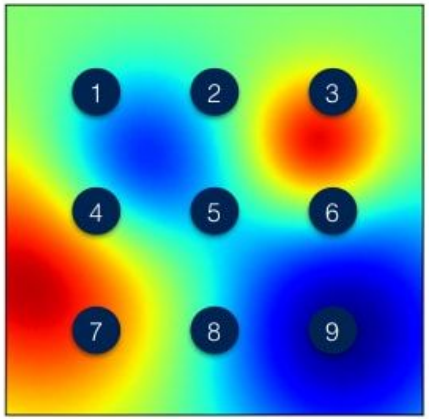


- Build model
  - In PyTorch, we use `nn.Module` to build model ⇒ define class
    - `__init__(self)`
    - `forward(self, x)`; x is the input into the model
  - **NOTE:** `model(x)` is equivalent to `model.forward(x)`
- Train model
  - Define loss function, optimizer
  - Loop through the `DataLoader`
    - Feed data to model, `zero_grad()`, evaluate loss, back propagation, update the weight via `step()`

Model flowchart

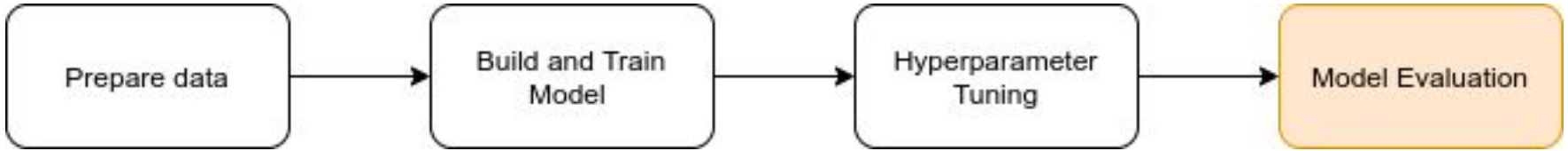


- Hyperparameters
  - Definition: parameters whose values are set before starting the model training process
  - Ex.: filter size, stride, padding, learning rate
- Implementation
  - scikit-learn ⇒ `GridSearchCV`
  - PyTorch ⇒ `from ray import tune`



Grid Search

Model flowchart



- Model Evaluation (Supervised Learning)
  - Evaluate the **generalizability** of the model ⇒ evaluate on unseen (test) data
  - loop through test `DataLoader`
    - Input data into model, see if predicted = actual or not, add that to measure
- Metrics for model evaluation
  - Accuracy
  - Confusion matrix