Multi-layer Perceptron

SYDE 599 Deep Learning F23

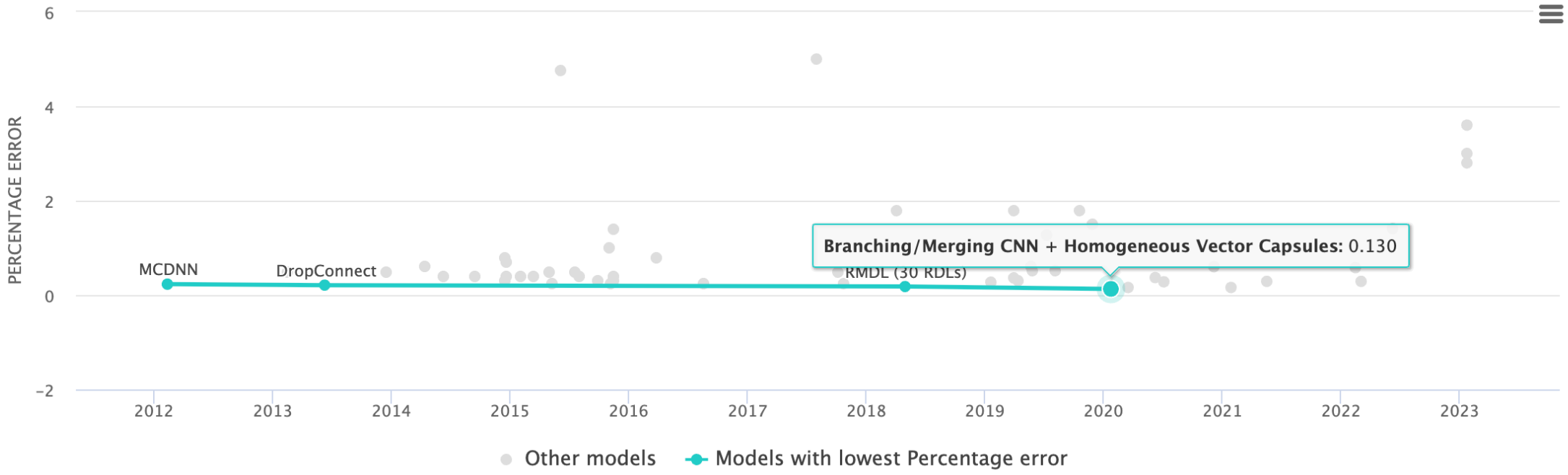
September 26, 2023

MNIST dataset

- Modified National Institute of Standards and Technology database
- A widely used dataset of images of handwritten digits
- Relevant to reading US ZIP codes for automatic mail sorting
- Ten categories (digits 0-9)
- 60,000 training examples
- 10,000 testing examples

https://commons.wikimedia.org/wiki/File:MnistExamples.png

State-of-the-art performance



https://paperswithcode.com/sota/image-classification-on-mnist

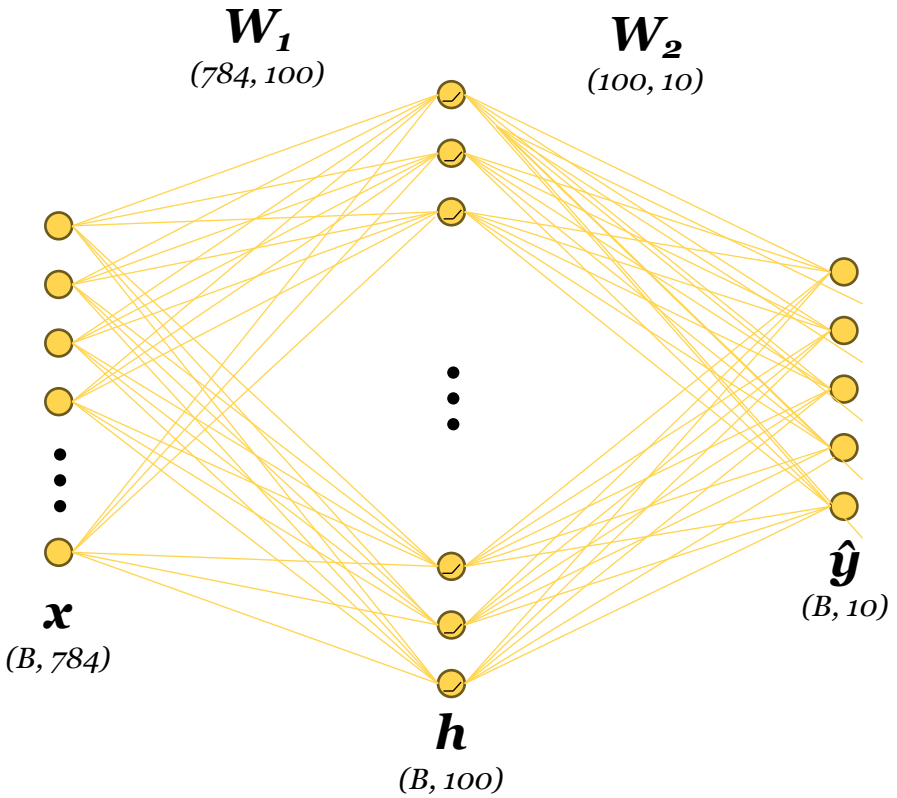
UNIVERSITY OF WATERLOO

Dataloader

- PyTorch's dataloader is an iterator that yields batches of (inputs, labels)
- We can lazily apply transformations to the data, including data augmentation
- For custom datasets:
  - Inputs should be an iterable of arrays that can be cast to tensors
  - Define a `__getitem__` method to build batches
  - https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

Multi-layer perceptron model

- In diagrams, we draw MLPs as neurons with incoming and outgoing connections
- In practice, we implement MLPs with two linear (dense) layers and a pointwise non-linearity in between
  - $h = \text{relu}(W_1 x + b_1)$
  - $\hat{y} = W_2 h + b_2$
- `linear(x) = x @ W + b`
  - Recall matmul is a linear transform on last dim of x
- $\hat{y}_i$ represents the log odds of predicting class i
- Class predictions are done by selecting the index with maximal log odds ($c = \text{argmax}(\hat{y})$)



Writing models in PyTorch

- All models are a subclass of `nn.Module` supertype
  - Any `nn.Parameter` tensors (including from sub-modules) that are assigned to class attributes track gradients (`requires_grad=True`)
  - All layers are also `nn.Module`'s and usually directly contain `nn.Parameter`'s
  - Allows parameters to be passed to optimizer easily (`model.parameters()`)
  - Allows parameters to be moved to GPU devices easily (`model.to(device)`)
- Model architecture is set in `__init__`
  - Name and assign other `nn.Module` layers to class attributes
- Model computation is defined in `forward`
  - Defines flow of data from inputs through layers defined in `__init__` to compute model output

Question

- What kind of task is MNIST? What loss function should we use?

Cross entropy and softmax

- We should be using softmax final activation with cross entropy loss
- `nn.CrossEntropyLoss()` fuses softmax and cross entropy calculations to be faster and more numerically stable when using integer labels
- Inputs are logits of shape (N, C)
- Labels are class indices as **integers** of shape (N,)
- https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Setup hyperparameters

- Model architecture
  - # of layers, # of neurons per layer
  - Layer structure
- Data preprocessing, data augmentation
- Optimizer setup
  - **Optimizer choice** (Adam, SGD, etc.) and **learning rate**
  - **Batch size, number of epochs**
  - Different hyperparameters for each optimizer (Week 5)
  - Weight decay (L2 penalty) (Week 6)
  - Learning rate schedulers

Training loop

- Combines stochastic first-order optimization and ML practices
- For each mini-batch:
  1. Fetch the next batch of data from the dataloader
  2. Reset gradients
  3. Compute model outputs from input data
  4. Compute loss function on model outputs and labels
  5. Compute gradient of loss w.r.t. parameters with backpropagation
  6. Update parameters with gradient descent step