Week9 Learning Reflection

Name: Xinyu Chang
Time: 2022.05.23-05.29
Class: CSE 416 Intro. Machine Learning
Lectures: 17 and 18

Summary

1. Hyperparameter optimization: recent work attempts to speed up hyperparameter evaluation by stopping poor-performing settings before they are fully trained.
2. very few evaluations: use random search
3. moderate number of evaluations: use Bayesian optimization

Concepts

(Lecture 17) Neural Networks, Deep Learning

Pre-class material

- **Neural Network**
  - When people talk about “deep learning” they are generally talking about a class of models called neural networks that are a loose approximation of how our brains work
- **Perceptron / Neuron**
  - Inputs -> weighted sum -> activation function -> output
- **Activation Function**
  - **XOR**
    - The perceptron can learn most boolean functions, but XOR not
    - data is not linearly separable, therefore can’t be learned with the perceptron
    - Idea: combine these perceptrons in layers to learn more complex functions.
- **Perceptrons for Boolean functions**
  - **Why XOR needs a hidden layers**
    - because the data is not linearly separable, can’t be learned with the perceptron.
    - Need hidden layers to come up with output with complex functions.

In-class Material

Activation functions - Sigmoid Functions

- **Sigmoid**
  - historically popular, but fallen out of favor
  - neuron's activation saturates (weights get very large -> gradients get small)
  - Not zero-centered -> other issues in the gradient steps
  - When put on the output layer, called "softmax" because interpreted as the class probability (soft assignment)
- **Hyperbolic tangent**
  - g(x) = tanh(x)
  - saturates like sigmoid unit, but zero-centered
- **Rectified Linear unit (ReLU)**
  - g(x) = x^+ = max(0, x)
  - Most popular choice these days
  - Fragile during training and neurons can "die off", be careful about learning rates
  - Noisy or leaky variants
- **Softplus**
  - g(x) = log(1+exp(x))
  - smooth approximation to rectifier activation

NN and overfitting

- How to avoid it?
  - Get more training data
  - Few hidden nodes/ better topology
    - Rule of thumb: 3-layer NNs outperform 2-layer NNs, but going deeper only helps if you are very careful (different story next time with convolutional neural networks)
  - Regularization
    - dropout
  - Early Stopping

Intuition for what NN learn in layers

NN for classification and regression

- Regression
    - The output layer will generally have one node that is the output (outputs a single number)
    - quality matric/cost function
        - RSS or RMSE
- Classification
    - The output layer will have one node per class. Usually take the node with the highest score as the prediction for an example. Can also use the logistic function to turn scores into probabilities. (softmax)
    - quality matric/cost function
        - Cross-Entropy loss

Backpropagation

- to do gradient descent with neural networks, we generally use a backpropagation algorithm
    - do a forward pass of the data through the network to get predictions
    - compare predictions to true values
    - backpropagate errors so the weights make better predictions

Batched learning

- it's pretty expensive to do this update for the entire dataset at once, so it's common to break it up into small batches to process individually.
- You generally have to repeatedly update the model parameters. We call an iteration that goes over every batch once an epoch.

```
for i in range(num_epochs):

    for batch in batches(training_data):

        preds = model.predict(batch.data) # forward pass

        diffs = compare(preds, batch.labels) # compare

        model.backprop(diffs) # backpropagation
```

NN Hyperparameters (not exhaustive)

- **Hidden layers, nodes per hidden layer**

    - **Activation function**
    - **Learning rate for gradient descent**
    - **Batch size**
    - **Number of epochs to train**
    - **More**

- **Hyperparameter optimization**
    - **Grid search**
        - code to loop over all possible combinations.
        - hyperparameters on 2d uniform grid
    - **Random search**
        - hyperparameters randomly chosen
    - **Bayesian optimization**
        - hyperparameters adaptively chosen
- **NN convergence**
    - In general, loss functions with neural networks are not convex.
    - This means the backprop algorithm for gradient descent will only converge to a local optimum.
    - Like with k-means, this means that how you initialize the weights is really important and can impact the final result.
    - How should you initialize weights?
        - usually, people do random initialization

(Lecture 18) Convolutional Neural Networks

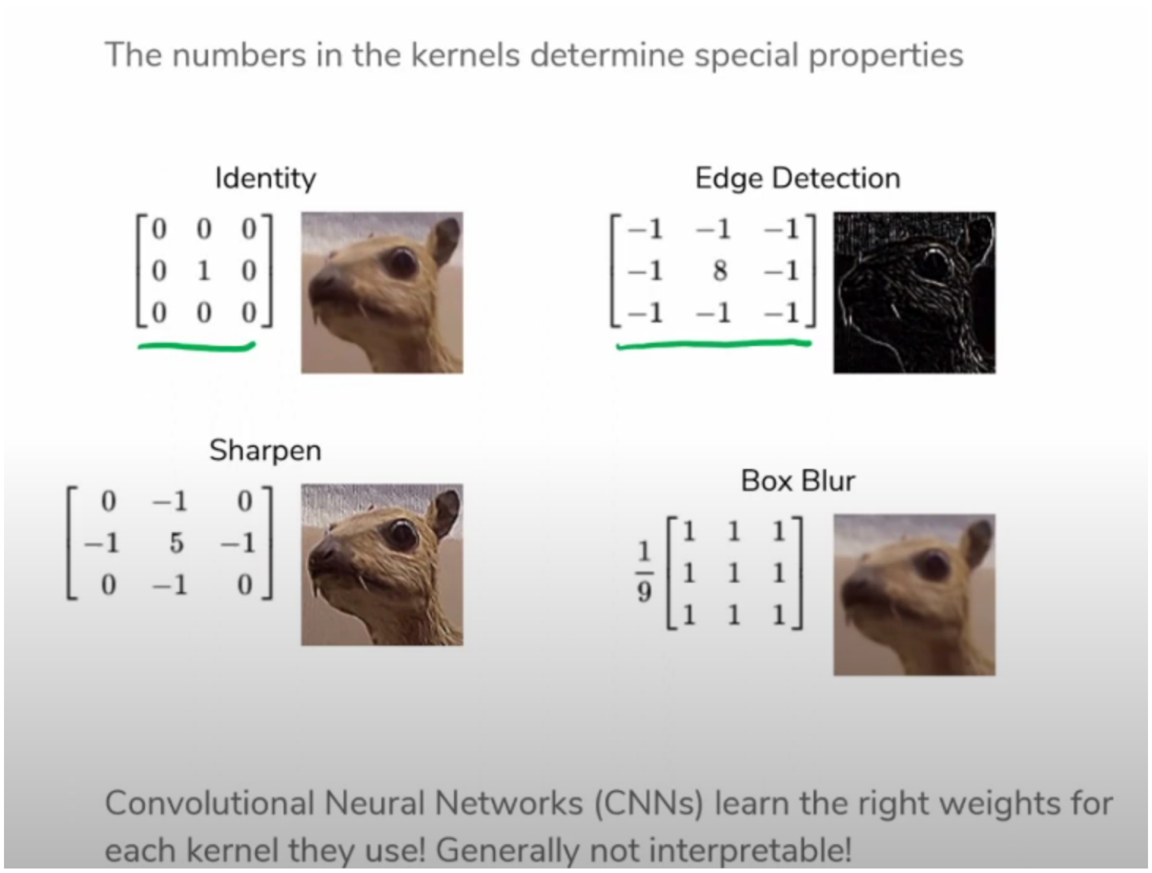
Pre-class material

- **Image dimensions** `channels @ height x width`
- **Convolution - reduce the number of weights that need to be learned by looking at local neighborhoods of image**

Use the idea of convolution to reduce the number of inputs by combing information about local pixels.

    - **Kernel -** use a kernel that slides across the image, computing the sum of the element-wise product between the kernel and the overlapping part of the image.
    - **Padding -** edge zeros

- **Stride -** how far to jump



The numbers in the kernels determine special properties

Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Box Blur

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Convolutional Neural Networks (CNNs) learn the right weights for each kernel they use! Generally not interpretable!

In-class Material

- **Pooling**

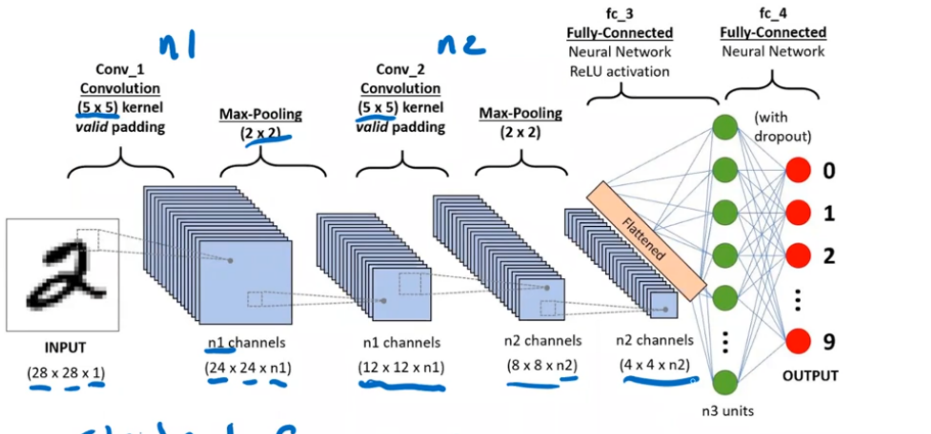
  idea is to downsample an image using some operation

  combine local pixels using some operation

  - **Max pool**
    - typical to use max pool with 2x2 filter and stride 2
    - Tends to work better than the average pool
- **Convolutional Neural Network (CNN)**

Example architecture for hand-written digit recognition

- Each convolution section uses many different kernels (increasing depth of channels)
- Pooling layers down sample each channel separately
- Usually ends with fully connected neural network



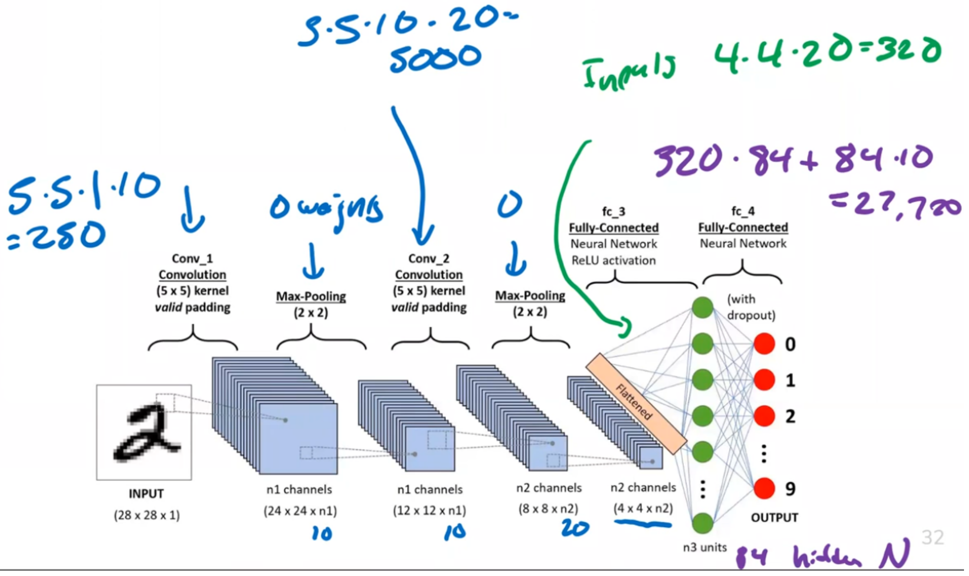
n1
n2
Conv_1 Convolution (5 x 5) kernel *valid* padding
Max-Pooling (2 x 2)
Conv_2 Convolution (5 x 5) kernel *valid* padding
Max-Pooling (2 x 2)
fc_3 Fully-Connected Neural Network ReLU activation
fc_4 Fully-Connected Neural Network
(with dropout)
Flattened
INPUT (28 x 28 x 1)
n1 channels (24 x 24 x n1)
n1 channels (12 x 12 x n1)
n2 channels (8 x 8 x n2)
n2 channels (4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

stride 1 for conv

250 + 8000 + 27,720 = 32,970

Consider solving a digit recognition task on 28x28 images.
Suppose I wanted to use a hidden layer with 84 neurons

With Convolutions (assume n1=10, n2=20)



- **General structure**

        - A series of Convolution + Activation Functions and Pooling layers. It's very common to do a pool after each convolution.
        - Then after some number of these operations, flatten the image to work with the final neural network
    - **CNN for grayscale images**
    - **CNN for color images**
- **Weight sharing**
    - Only need to learn a small number of values (kernel weights) that get applied to the entire image region by region
    - gives efficiency + shift invariance
- **Pros and Cons of Deep Learning**
    - Pros
        - no need to manually engineer features, enable automated learning of features
        - Impressive performance gains
            - Image processing
            - Natural Language Processing
            - Speech recognition
    - Cons
        - Requires a lot of data
        - computationally really expensive
        - hard to tune hyper-parameters
            - Choices of architecture (we've added even more hyperparameters)
            - learning algorithm
        - not very interpretable
- **Transfer Learning**
    - share the weights for the general part of the network
    - can give you
        - a higher start
        - a higher slope
        - a higher asymptote
- **Adversarial attacks**
- **Dataset bias**

Uncertainties

1. If training a model using Grid Search requires an extremely long time, ex. 1 whole day, will that be reasonable if I decided to not use such a model to generate the result?
2.