# Foundations of Deep Learning

Geralyn Chong

March 16, 2025

## Contents

Note: This is a brief set of notes that I took during the day. Feel free to add to the latex and update me so I can update the master doc!

## 1 Useful Links:

1. Nvidia Container to running Fundamentals of Deep Learning
2. PDFs of the Workshop's Slides:
   - Slide 1
   - Slide 2
   - Slide 3

- Slide 4
- Slide 5
- Slide 6

# 2 History of AI

## 2.1 Early Neural Networks

- Inspired by biology → Von Neumann Architecture
- **Expert Systems** aids the ”intelligence” of old computer systems; Built to mimic a human expert;
  - But we don’t want to hard code these complex conditions of real world problems!
  - Training models like training children to recognize important relationships and patterns on their own by exposing them to data etc.

## 2.2 The Deep Learning Revolution

1. **Data** Networks need a lot of data via the internet
2. **Processing Power**
3. **Importance of the GPU** Matrix multiplication applies to graphic rendering (animation, simulation, etc.) Parallel processing - help to execute these operations millions of times.

# 3 What is Deep Learning?

## 3.1 Traditional Programming

1. Define a set of rules for classification
2. Program those rules into a computer
3. Feeback

## 3.2 Deep Learning steps...

If the rules are nuanced, complex, use deep learning. Taking in real world conditions that are hard to describe.

1. Show modle the examples with the answer of how to classify
2. Takes guesses
3. Model learns to correctly cateogirze as its training. System **LEARNS** these rules on their own

Deep Learning has a large depth ond complexity of layers and networks.

# 4 Hello Neural Networks

- Train a network to correctly classify handwritten digits
- Try learning like a neural network
- Cool sandbox link

# 5 Lab 1: Mnist Dataset

## 5.1 Validation vs Test

1. Validation and Test sets are held out from the training process because evaluating models based on the same data that we train them on would result in bias

2. Training set refers to a set of examples that we want to pass through the model in order to learn specific rules. However, we can use the verification set to *fine-tune* the model (e.g choosing the number of hidden units in a neural network). Test set used to evaluate the performance of the model. Pseudocode:

   ```
   # split data
   data = ...
   train, validation, test = split(data)

   # tune model hyperparameters
   parameters = ...
   for params in parameters:
    model = fit(train, params)
    skill = evaluate(model, validation)

   # evaluate final model for comparison with other models
   model = fit(train)
   skill = evaluate(model, test)
   ```

## 5.2 However, just a validation set is not enough!

Another method that can be used is the **k-fold cross validation method**. This method aims to mitigate against biases when dividing large datasets randomly. With smaller datasets, we might be overfitting towards a specific division of points and failing the test set as a result. Pseudocode:

```
# split data
data = ...
train, test = split(data)

# tune model hyperparameters
parameters = ...
k = ...
for params in parameters:
 skills = list()
 for i in k:
  fold_train, fold_val = cv_split(i, k, train)
  model = fit(fold_train, params)
  skill_estimate = evaluate(model, fold_val)
  skills.append(skill_estimate)
 skill = summarize(skills)

# evaluate final model for comparison with other models
model = fit(train)
skill = evaluate(model, test)
# split data
data = ...
train, test = split(data)

# tune model hyperparameters
parameters = ...
```

```
k = ...
for params in parameters:
 skills = list()
 for i in k:
  fold_train, fold_val = cv_split(i, k, train)
  model = fit(fold_train, params)
  skill_estimate = evaluate(model, fold_val)
  skills.append(skill_estimate)
 skill = summarize(skills)

# evaluate final model for comparison with other models
model = fit(train)
skill = evaluate(model, test)
```

### 5.3 What is the tech stack?

- Sequential API - useful built-in functions for designing and construction neural networks

### 5.4 Walk through of Lab 1

Regression is when you take in a continuous input to predict a continuous output. Optimizers:

- Adam - error is high, the marble will speed up as it goes down the 'mountain' avoiding the local minimums to try to land at a global minimum.

1. We begin by converting our images into a **tensor** ($n$-dimensional representation of our data) and verify the minimum and maximum values of our tensor data. Transforms are a group of torchvision functions that can be used to transform a dataset. For example:

   (a) `transform.Compose()` is used to combine a list of transforms like so: `transform.Compose([transforms.toTensor()])`

   (b) After setting up the set of transform functions we want to apply, we set them to the dataset's transform variable:

   ```
   train_set.transform = trans
   valid_set.transform = trans
   ```

   (c) Using **Dataloaders** as our approach to how we pull out our flashcards of input data, we can set the batch size $= 32$ so that we shuffle and pull out a random set of 32 sample instances / cards to train:

   ```
   batch_size = 32
   train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
   valid_loader = DataLoader(valid_set, batch_size=batch_size)
   ```

   (d) Now, we can build the **model!** but first we must consider some prerequisites and layers:

      i. `Flatten()` will be used to convert our tensors of $n$ dimensional representation to a vector as an input. This is important because our input tensor of our data was $(C \times H \times W)$ and we need the data to be a 1-dimensional array to be able to parse it through our connected network

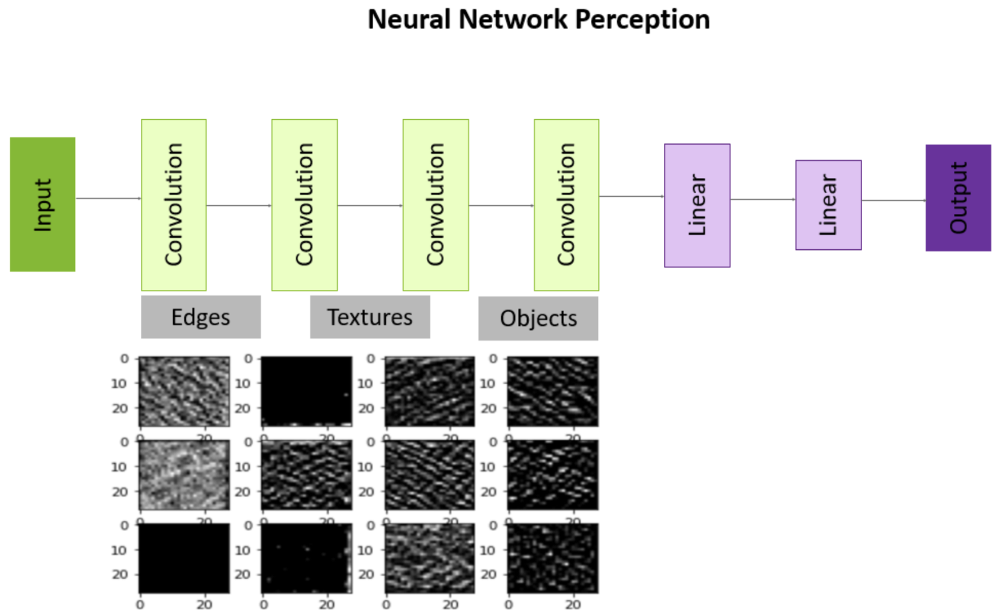
      ii.

# 6 Lab 2: ASL

### 6.1 Kernels and Convolution

Star in the image is a convolution operator. If we want the input image to be the same size as the output or if we want more data to inform our convolution, we can add some zero padding to the image. When the image is small, padding can impact our convolution heavily.

Mirror padding refers to adding duplicated width and height wise. Convolutional neural networks map each neuron to a kernel of trainable weights. An edge is a place where the change in color of that image is changing rapidly. Hence, we can use these kernels to effectively detect shapes in images!



Max pooling is when you take the window to identify the maximum to represent the entire window when reducing the dimensions of an image.

Dropout training refers to when you shutoff neurons during the training process to eliminate the possibility of one neuron overcontributing to the neural network.

Batch normalization: scales weights / values of hidden layers to improve training.

### 6.2 Lab 3: Convolutional networks to avoid overfitting

1. From the previous representation of our input data, we saw tensors that were flattened to represent data in a 1-dimensional array. However, for the first convolutional layer of our model, we want to retain spacial information about which pixels are next to each other. We reshape our pixel representations such that it mimics some picturee representation.

2. This differs from the previous when we define our `DATASET` class because we include a reshaping of the input data representation.

3. Dataloader is created so that we set the batch size and shuffle for the training data so that the subset of the input data is randomized.

Training accuracy was still higher than the validation accuracy. But sometimes it's not the model that needs work, it's the data. Require better examples.

## 7 Lab 4: Data Augmentation

Data Augmentation: changing the color hue, rotation, and flipping images to increasing our sample space. (Homography) There are various random augmentation functions that we can deploy:

- `RandomResizeCrop((IMG_WIDTH, IMG_HEIGHT), scale=(min, max), ratio=(1, 1))`
  *ratio being aspect ratio*

- `RandomHorizontalFlip()`

- `RandomRotation()`

- `ColorJitter(brightness, contrast, saturation, hue)`

Remembering that we can use the `Compose()` function to group transform functions together, we can implement these data augmentation techniques as such:

```
random_transforms = transforms.Compose([
    transforms.RandomRotation(5),
    transforms.RandomResizedCrop((IMG_WIDTH, IMG_HEIGHT), scale=(.9, 1), ratio=(1, 1)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=.2, contrast=.5)
])
```

So before we implement our training function we need to change the input to the model we built using this transformation process: `output = model(random_transforms(x))`

### 7.1 Deploying the model:

Because we have trained our model on a specific set of requirements, more spcifically, grayscale and 28 × 28 images, we need to ensure that any new images that we pass through comply with these requirements of our model- otherwise, the outputs mean nothing much. Here is an example of ensuring that images read into the model are grayscale:

```
IMG_WIDTH = 28
IMG_HEIGHT = 28

preprocess_trans = transforms.Compose([
    transforms.ToDtype(torch.float32, scale=True), # Converts [0, 255] to [0, 1]
    transforms.Resize((IMG_WIDTH, IMG_HEIGHT)),
    transforms.Grayscale()  # From Color to Gray
])
```

After checking that our image is correctly reformated, we can start making predictions!

1. Ensure that our reformatted image is `.unsqueeze(index)` to add a dimension of 1 to the index of our "list / vector" representation of the image.

Once we ensure that our dimensions match up, we can push it through the `model(batched_image)`. Batched image refers to how our model expects a batched input so placing our one image into a batch by itself allows us to generate a prediction for it alone.

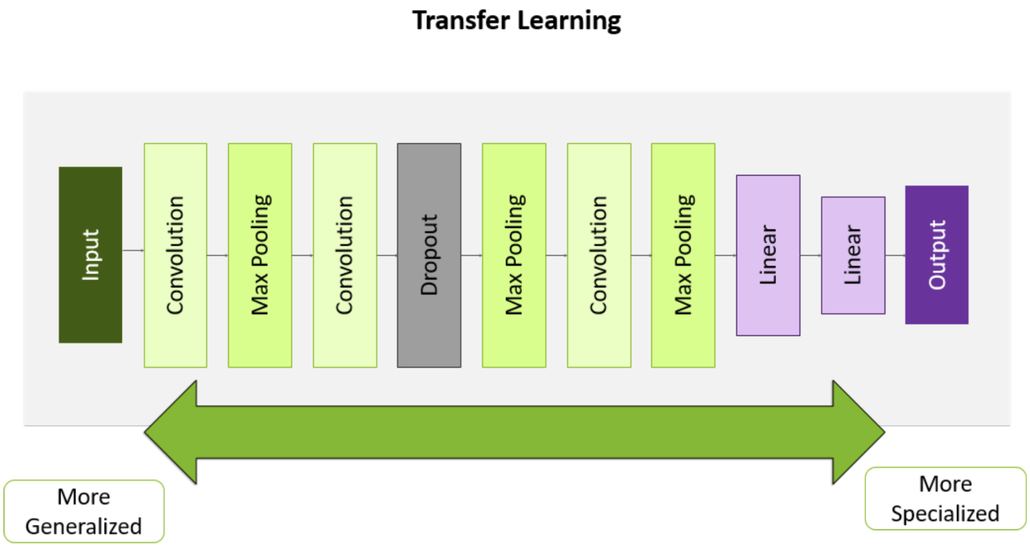
Once we do this, we recieve a prediction that is not like our labels but instead like this:

```
tensor([[-25.0550,  12.9947,  -9.9380, -14.2813,  -3.0655,  -1.5400,  -1.0154,
        -14.3424,  -8.6212, -27.2385, -16.8412, -19.3476, -17.7453, -25.7918,
        -10.6322, -22.6966, -20.0060, -22.4649, -13.0284, -11.5978, -30.6815,
        -11.5446,  -5.6419, -28.9596]], device='cuda:0',
    grad_fn=<AddmmBackward0>)
```

We see that this prediction output corresponds to a 24-length array that represents each letter of ASL that our input data might represent. So the `argmax()` can help us identify which letter the input most likely represents.

## 8 Pre-trained Models: Very Deep Convolutional Networks for Large-scale image recognition

Lab 5a and Lab 5b



Freezing the model to avoid overfitting on the much smaller dataset. Data bias: when a subset of data that is over-represented in the dataset.

Dreaming: feeding an image through layers of a network and utilizing gradient ascent instead of gradient descent. It exaggerates the features it detects!

Using a pre-trained model, we want to ensure that we understand the pre-trained weights in order to match our $n$-dimensional representations and to freeze the base model when training our additional layers. Fine-tuning occurs after the initial training of the new layers and we can then unfreeze the base model to fine-tune by small learning increments.

## 9 Advanced Architectures

- Computer Vision: better our understanding of human perception of visual objects (color - RGB)
- NLP: generating human language but need to represent language numerically. How do we capture sentence structure and semantic subtleties?

## 10 Natural Language Processing

From our dictionary of words, we assign each input word / token to a neuron like utilizing One-hot encoding. Embedding handles our large dictionary size such as going from a layer with a higher number of neurons to a lower number of neurons. Suppose that some embedding outputs 3 encodings per input. When we take two embedding matrices and multiply them together, we can derive an attention matrix.

BERT, Bidirectional Encoder Representations for Transformers by Google was trained towards predicting a missing word from a sequence and predicting a new sentence after a sequence of sentences.

Autoencoders are used to represent an input with a lower dimensional medium in order to reconstruct the same output. An encoder allows us to storage large amounts of data with low dimensional representation. This can be applied to the detection of anomalies and security purposes.

- Variational Autoeconder: generative AI for images
- Diffusion Models: Taking a noisy image and slowly remove small amounts of noise until we generate a somewhat noise free image (Stable Diffusion!)
- Reinforcement Learning; Put off a reward for now in hopes of attaining a greater reward later. Applications in Robotics and Finance