**Part – 3 DL Concepts**

**1. Convolution Neural Network. nn.Sequential , nn.Conv2d, nn.ReLU, nn.MaxPool2d layers.**

convolutional neural network (CNN/ConvNet) is a class of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network), most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

Bottom line is that the role of the ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

**CNN = element wise matrix multiplication and addition**

**torch.nn.Sequential(\*args)**

A sequential container. Modules will be added to it in the order they are passed in the constructor.

Alternatively, an OrderedDict of modules can be passed in. The forward() method of Sequential accepts any input and forwards it to the first module it contains.

It then “chains” outputs to inputs sequentially for each subsequent module, finally returning the output of the last module.

**# Using Sequential to create a small model. When `model` is run,**

# input will first be passed to `Conv2d(1,20,5)`. The output of

# `Conv2d(1,20,5)` will be used as the input to the first

# `ReLU`; the output of the first `ReLU` will become the input

# for `Conv2d(20,64,5)`. Finally, the output of

# `Conv2d(20,64,5)` will be used as input to the second `ReLU`

model = nn.Sequential(

nn.Conv2d(1,20,5),

nn.ReLU(),

nn.Conv2d(20,64,5),

nn.ReLU()

)

**torch.nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding\_mode='zeros', device=None, dtype=None)**

# With square kernels and equal stride

m = nn.Conv2d(16, 33, 3, stride=2)

# non-square kernels and unequal stride and with padding

m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))

# non-square kernels and unequal stride and with padding and dilation

m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))

input = torch.randn(20, 16, 50, 100)

output = m(input)

**Pooling 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 by reducing the dimensions. There are two types of pooling average pooling and max pooling.

**torch.nn.MaxPool2d(kernel\_size, stride=None, padding=0, dilation=1, return\_indices=False, ceil\_mode=False)**

# pool of square window of size=3, stride=2

>>> m = nn.MaxPool2d(3, stride=2)

# pool of non-square window

>>> m = nn.MaxPool2d((3, 2), stride=(2, 1))

>>> input = torch.randn(20, 16, 50, 32)

>>> output = m(input

**torch.nn.ReLU(inplace=False)**

Applies the rectified linear unit function element-wise:

ReLU(x)=(x)+=max(0,x)

replaces the negative values with zero.

>>> m = nn.ReLU()

>>> input = torch.randn(2)

>>> output = m(input)

**2. Overfitting, How to avoid it? Learning rate scheduling? Weight decay. Gradient clipping?**

**Bias –** Simple assumption on model. Assume linear model can solve data in non linear.

**Variance –** Change of results while changing the data in train model

**Overfitting : low bias and high variance**

low bias --> low training error

high variance --> high test error

**Underfitting : high bias and high variance**

both train error and test is high

**Techniques to avoid overfitting**

**i) Cross Validation** – Split the dataset into k-sets and consider one set as test set and remaining as train set continue this until each individual set as testset.

**ii) Feature selection** – we should only choose most important feature rather than all.

**iii) Data Augumentation** – artificially increase the size of the dataset. For example, if we are training for an image classification task, we can perform various image transformations to our image dataset (e.g., flipping, rotating, rescaling, shifting).

**iv) Early Stopping –** In the error plot if validation loss degrading we need to stop the training and save the model.

**v) Dropouts** – randomly selected nuerons are ignored during training.

**vi) Regularization –** used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.

L1 Regularization(Lasso regularization)

L2 Regularization(Ridge regularization)

**Learning rate Scheduling**

in SGD optimizers Adagrad(Adaptive Gradients), Adadelta, RMSProb, Adam(Adaptive Momentum Estimation) for the fast convergence the learning rate is updated for every weight updation.

There are various learning rate schedulers in PyTorch that are made available in the **torch. optim.lr\_scheduler package.**

lr\_scheduler.LambdaLR

lr\_scheduler.MultiplicativeLR

lr\_scheduler.StepLR

lr\_scheduler.MultistepLR

lr\_scheduler.ConstantLR

lr\_scheduler.LinearLR

lr\_scheduler.ExpotentialLR

**Weight Decay**

To penalize the complexity we can add all parameters(weights) to loss, but it is not good because some parameters are positive and some are negative.

Then what if add the square of weights to loss but the loss become huge. To handle the above multiply sum of squares(w^2) with another small number. It is called weight decay

**Weight Decay VS L2 regularization**

While weight decay is added directly to the update rule, L2 regularization is added to the loss.

**Gradient Clipping**

computing gradients are the very important part in backpropagation. So we need to monitor this gradients whether it is vanishing gradients(very small gradients are computed) or exploding gradients(very large gradients are computed).

One technique to avoid the exploding gradient is gradient clipping.

Gnew = (G / ||G||2 ) \* threshold

G / ||G||2 < 1 always

||G||2 --> sqrroot(G12 + G22  + G32 + Gn2)

new gradient is always less than the threshold.

# Gradient Norm Clipping

nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=2.0, norm\_type=2)

**3. Regularization of data? Data Normalization and Augmentation?**

**Regularization:** techniquies to avoid the overfiiting from the model.

**i) L1 or Lasso regularization**

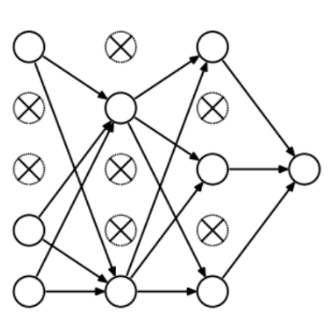


**ii) L2 or Ridge regularization**



**ii) Dropouts**

At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.



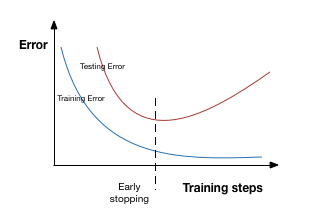
**iii) Data Augumentation**

The simplest way to reduce overfitting is to increase the size of the training data. In machine learning, we were not able to increase the size of training data as the labeled data was too costly.

But, now let’s consider we are dealing with images. In this case, there are a few ways of increasing the size of the training data – rotating the image, flipping, scaling, shifting, etc.

**iv) Early Stopping**

Early stopping is a kind of cross-validation strategy where we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model.



**Data Normalization:**

The goal of normalization is to transform features to be on a similar scale. This improves the performance and training stability of the model.

Four common normalization techniques may be useful:

* scaling to a range
* log scaling
* z-score

|  |  |  |
| --- | --- | --- |
| Normalization Technique | Formula | When to Use |
| Linear Scaling |  | When the feature is more-or-less uniformly distributed across a fixed range.  **[scaling](https://developers.google.com/machine-learning/glossary" \l "scaling)** means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range—usually 0 and 1 |
| Log Scaling | x’ = log(x) | When the feature conforms to the power law.  when a handful of your values have many points, while most other values have few points. This data distribution is known as the power law distribution. Movie ratings are a good example. |
| Z-score |  | When the feature distribution does not contain extreme outliers.  feature distributions (normal or gaussian) have mean = 0 and std = 1. |

**Data Augmentation:**

Data augmentation is a process of artificially increasing the amount of data by generating new data points from existing data.

Advanced models for data augmentation are

**Adversarial training/Adversarial machine learning**: It generates adversarial examples which disrupt a machine learning model and injects them into a dataset to train.

**Generative adversarial networks (GANs)**: GAN algorithms can learn patterns from input datasets and automatically create new examples which resemble training data.

**Neural style transfer:** Neural style transfer models can blend content image and style image and separate style from content.

**Reinforcement learning**: Reinforcement learning models train software agents to attain their goals and make decisions in a virtual environment.