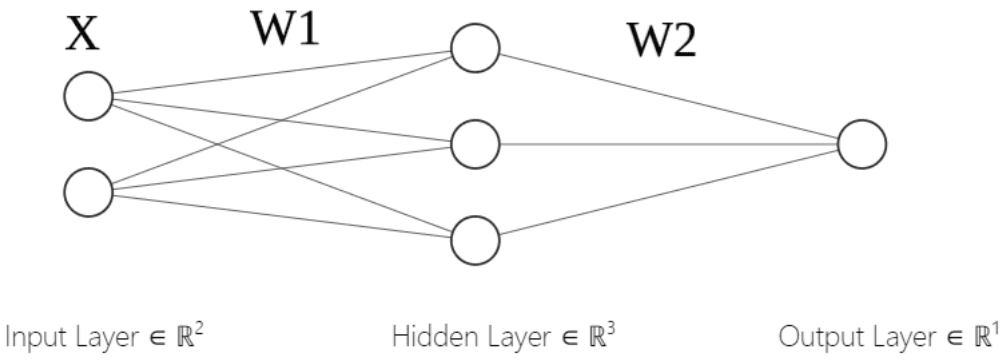
CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Transfer Learning and Landscape of CNNs

Prepared by: Mehul Gupta (20171156), Anupam Misra (2019201082)

1 RECAP

1.1 Multi-Neuron Network :: Backpropagation



Let W_1, W_2 denote the weight matrix at Layer 1, 2 respectively.

Let $Z = W_1^T X$

Let $P = \phi(W_1^T X)$

$\hat{Y} = W_2^T P$

Loss function:

$$\mathcal{L}(\hat{Y}, Y_i) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y} - y_i)^2$$

Weights Updation rule

$$W_1 = W_1 - \alpha\frac{\partial \mathcal{L}}{\partial W_1}$$

$$W_2 = W_2 - \alpha\frac{\partial \mathcal{L}}{\partial W_2}$$

Here alpha is learning rate

Computing Grad of Cost function wrt W_2

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{1}{2}\frac{\partial\Big(W_2^T P - Y\Big)^2}{\partial W_2}$$

$$\frac{\partial \mathcal{L}}{\partial W_2} = (W_2^T P - Y)P$$

Computing Grad of Cost function wrt W_1

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial P}\frac{\partial P}{\partial Z}\frac{\partial Z}{\partial W_1}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = (\hat{Y} - Y)W_2^T \phi(W_1^T X)X$$

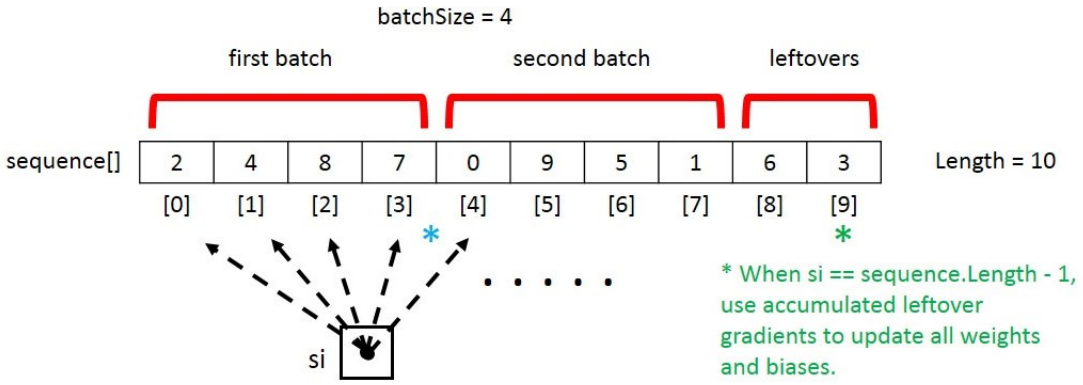
1.2 Epoch, Mini-batch, iteration

Epoch : One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Why we use more than one Epoch?
As we are using a limited dataset and to optimise the learning we are using Gradient Descent or mini-batch gradient descent which is an iterative process. So, updating the weights with single pass or one epoch is not enough.
As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

Batch : We can't pass the entire dataset into the neural net at once due to many reasons. So, we divide dataset into Number of Batches or sets or parts.

Batch size : Total number of training examples present in a single batch.



Mini-batch Gradient Descent : When all training samples are used to create one batch, the learning algorithm is called gradient descent. When the batch is the size of one sample, the learning algorithm is

called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

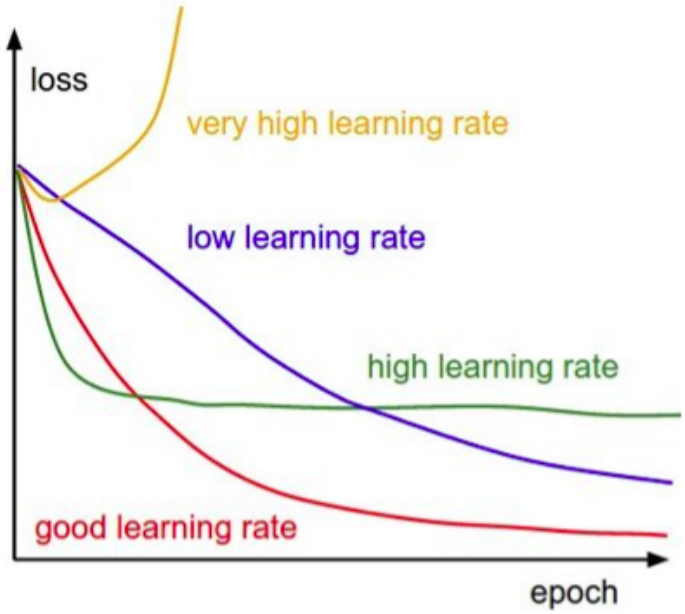
- **Gradient Descent**: Batch Size = Size of Training Set
- **Stochastic Gradient Descent** : Batch Size = 1
- **Mini-Batch Gradient Descent** : 1 < Batch Size < Size of Training Set

Iterations : Iterations is the number of batches needed to complete one epoch.
Example - Let's say we have 2000 training examples then we can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

In short

1. Think of a batch as a for-loop iterating over more than one sample and making predictions.
2. At the end of the batch, the predictions are compared to the expected output variables.
3. For given prediction an error is calculated.
4. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

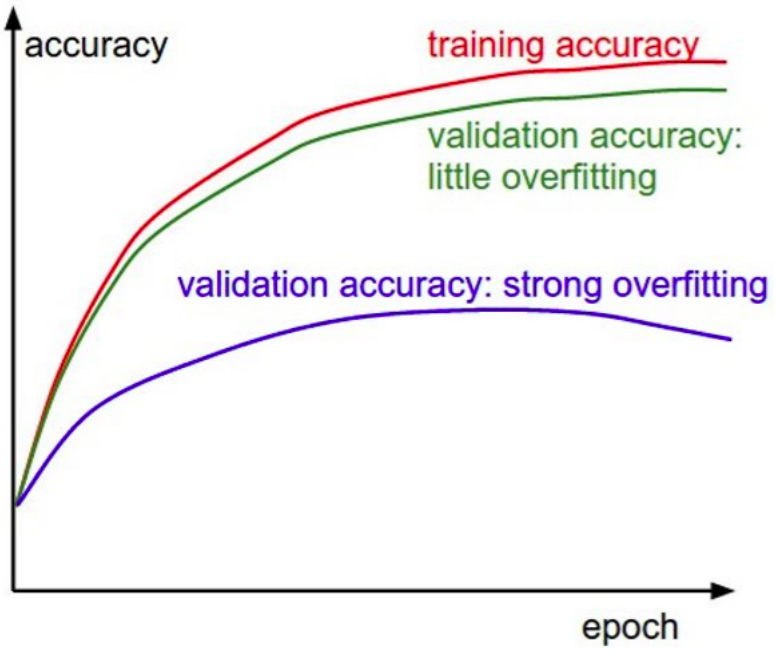
1.3 Determining learning rate



Loss value vs Number of epochs for different learning rates

Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.
The above plot shows changes in loss value as the number of epochs increases for different learning rate.If on increasing the number of epochs if loss value goes close to zero then it is a good learning rate (shown by red curve). If on increasing epochs loss increases then it is a very high learning rate (shown in orange) and such learning rate should be avoided as it shows network is not learning well.
Keras provides a number of different popular variations of stochastic gradient descent with adaptive learning rates, such as AdaGrad, Adam etc.

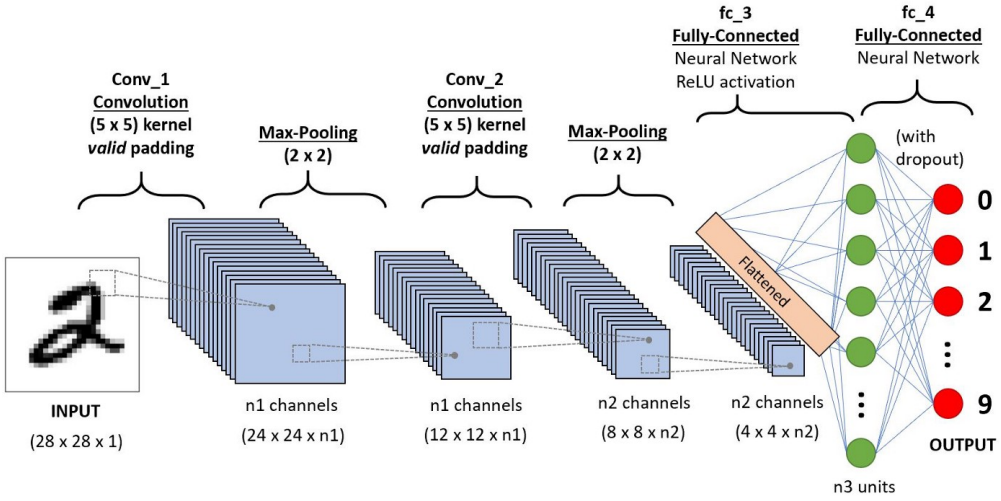
1.4 Accuracy vs epochs



Accuracy vs Number of epochs

If validation accuracy behaves same as training accuracy then it shows that in training we have done less overfitting and if validation accuracy goes down then it shows that we have overfitted the data in our training of neural network.

1.5 Recall CNN



There are mainly 3 types of layers in CNN as follows :

1. **Convolutional layer :**

- Apply filters to extract features
- Filters are composed of small kernels
- one bias per filter
- Apply activation function on every value of feature map.

2. **Pooling layer :**
   - Reduce dimensionality
   - Extract maximum average of a region
   - sliding window approach
3. **fully connected layer :**
   - Aggregate information from final feature maps
   - Generate final classification

2 Transfer Learning

It is not a good idea to train a neural network from scratch whenever we can get away with it because it takes a lot of data and time. It is possible that we may not have a lot of data. Whenever we are starting a new task in most of the cases it is not the first time someone is doing that task. Usually some one must have done that task or a similar task in the past. In such cases we perform transfer learning. Transfer learning saves time and does not require a large amount of data.

Definition 2.1. *Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.*

2.1 Intuition Behind Transfer Learning

The intuition behind transfer learning is that we copy what exists and learn only what is needed. A neural network learns hierarchy of features. The low level layers which are closer to the input learn low level features (for example vertical edges, horizontal edges etc.). As the layers get closer to the output, the layers start learning things which are semantically meaningful (for example wheels, doors etc.). The low level features are not specific to a particular task where as the high level features are specific to a particular task. So we might be able to reuse these low level layers for some other task given that we reuse them along with their weights.

2.2 Setting up Neural Network

In transfer learning we reuse some of the components of the source task network in our target task network. The architecture of target task and source task are same. So we first setup the initial part of the architecture of target task from the source task. We copy the weights from the source task (obtained after training the source task) to the target task for the copied layers. The number of layers that we copy is our design choice depends on the target task. In practice only convolutional layers are copied, fully connected layers are rarely copied because the copied layers might leak some semantics of the source task and the fully connected layers are specific to the source task. The number of layers to be copied can be determined by experimentation. Then we add new layers customised to our target task. Now in the target task the weights for copied layers are already present and the weights for the new layers have to be learned. The number of weights the network has to learn are reduced so this saves time.

2.3 Training the Network

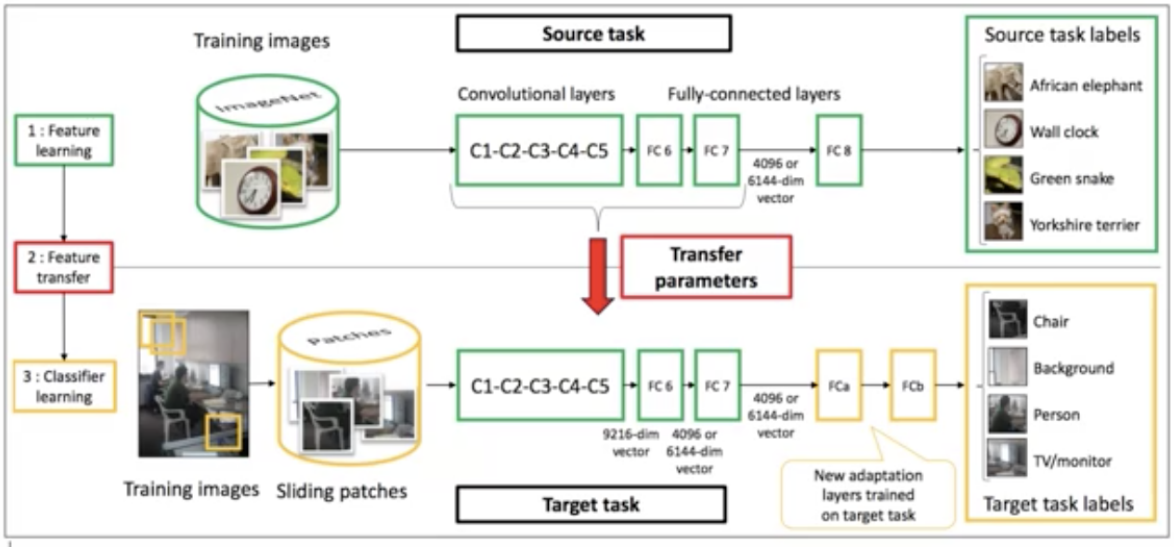
2.3.1 Approach 1

1. In this approach we freeze the weights for copied layers and only learn the weights for new layers. This can be done by preventing weight update for copied layers during back propagation.
2. This approach is ideal when the new domain data is small in quantity. It becomes tough to learn weights for the entire network with small quantity of data.
3. During the forward pass the output of copied layers is calculated. Now this output is used to learn the weights of new layers. The weights of copied layers do not change.
4. Advantage of using this approach is that as less number of weights are updated the training becomes faster.
5. In this approach we assume that the copied layers with frozen weights give us useful features that we can use in the higher layers.

2.3.2 Approach 2

1. In this approach we do not freeze any weights. Here we use different learning rates for different layers. For the copied layers we use the learning rate used in the training of the source task and for the new layers we use an increased learning rate. Learning rate is increased by a factor which is usually 10.
2. We use an increased learning rate for newly added layers because we want them to adapt faster and we want more loss to be reduced due to adaptation of new layers than copied layers. The new layers have to catch up to the copied layers. The new layers are given higher priority.
3. Here we are modifying the copied layers for them to adapt better to the target task.
4. This approach is ideal when the new domain data is of reasonably large quantity or domain shift is significant i.e. the target task is reasonably different from source task. This approach is used even if the new domain data is small in quantity, but there is a significant domain shift or sometimes neither approach is used, instead it is better to use a smaller network and train it from scratch.

Transfer learning cannot be used for data with different dimensions. For different dimensions the network can be tricked and then transfer learning can be used. For example in case of RGB images and black and white images.



Above is an example of transfer learning where the source task recognises African elephant, wall clock etc. and the target task recognises chair, background etc.

3 Landscape of CNNs: Applications and Architectures

3.1 Semantic Segmentation

1. Semantic Segmentation is given a particular picture we want to predict what class each pixel belongs to. For example, given a picture of a road we want to predict which pixels are car, which are roads, which are trees etc. and represent them using different colours.
2. Here the output is spatial.
3. For this we perform a bunch of convolution layer processing and then at the output we setup a softmax for each pixel because we want to predict what class it belongs to. The final output is called a label map which tells us which class each pixel belongs to.

3.2 Image $->$ Label

1. We can set up networks which recognise what is the class of the object in the given image.
2. We can also set up networks which recognise what kind of scene it is and what all is present in that scene.
3. We can also set up networks which recognises faces, used for faceids etc.

3.3 Image $->$ Number

1. These networks are used to get number related information. For example, given a face we want to determine its age. In case of age prediction we can use the relu function to avoid negative values and get a sensible output.
2. This can be setup as a regression problem or a classification problem if the output is an integer with limited range like in the case of age prediction.
3. Object Detection - Here we not only want to determine what objects are present, but also where they are present, their spatial locations. To determine the spatial location we find the bounding box of the object and for that we need to find the coordinates of the four points (corners) of the bounding box.

3.4 Image $->$ Label Image

1. Scene parsing - It is similar to semantic segmentation.
2. Object parsing - It is similar to scene parsing. In this we determine individual parts of an object instead of individual objects in a scene. For example, for a person we determine individual items of clothing, individual accessories etc.

3.5 Image $->$ Image

Popular examples of this network are depth estimation network, crowd counting network etc. For crowd counting the output is a 1D spatial output and for the ground truth we prepare an image known as crowd

density, which tells us how dense is the crowd in the image. We determine the crowd density estimate and then do a summation over it to obtain the crowd count estimate. In case of depth estimation we determine how far are objects in the image.

3.6 Multi Channel Input

Tasks are performed by taking input from multiple channels. For example determining neural damage using MRI input from multiple channels.

3.7 Multi Branch Input

Popular example of this is image colourization. In this case we don't have to explicitly collect any label data. Start with some collection of colour images. Convert these images to grayscale images. Now we have input output pairs. So we setup a network which takes black and white images as input and gives coloured images as output. This may have an additional network to collect auxiliary information that will improve our output by determining things like whether the image is indoor or outdoor etc.

3.8 Image CNNs for non-image data

Image CNNs can also be used for non-image data like audio because they are configured to process spatial data. In case of audio we can represent it as a spectrogram. In structure spectrogram resembles a 2D image. So we can perform convolution style processing on spectrogram. This can be used for different tasks like speaker identification, instrument identification etc.

References

[1] Lecture Slides

[2] Transfer Learning definition