IDC410
A course on Image Processing and Machine Learning
(Lecture 14)

Shashikant Dugad,

IISER Mohali

Reading Material

Suggested Books:

1. **Neural Networks and Deep Learning by Michael Nielsen**
2. **Fundamentals of Deep Learning by Nikhil Buduma**

Source for this presentation:

Neural Networks and Deep Learning by Michael Nielsen

https://www.scaler.com/topics/deep-learning/introduction-to-feed-forward-neural-network/

https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network

http://machine-learning-for-physicists.org. by Florian Marquardt

3Blue1Brown (Youtube Videos)

https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Source:

https://medium.com/towards-data-science/deriving-backpropagation-with-cross-entropy-loss-d24811edeaf9

https://medium.com/towards-data-science/backpropagation-the-natural-proof-946c5abf63b1

Backpropagation Algorithm for Multi-Binary Network

Backpropagation: Multi-Class Categorical Classification

- **Backpropagation equations for Categorical Cross Entropy Loss Function with Softmax as activation function in the output layer; the *FIVE forward and backpropagation equations* can be written as:**

$$\Rightarrow z^L = (W^L)^t a^{L-1} + b^L \quad unless\ L = 0\ then\ z^L = (W^L)^t x + b^L$$

$$\Rightarrow a^L = h(z^L)$$

- $$\delta^L = h'(z^L) \odot W^{L+1}\delta^{L+1} \quad unless\ L = H\ then\ \delta^L = (a^L - y)$$

- $$\frac{\partial J}{\partial b^L} = \delta^L$$

- $$\frac{\partial J}{\partial W^L} = a^{L-1} \cdot (\delta^L)^t \quad unless\ L = 0\ then\ \frac{\partial J}{\partial W^L} = x \cdot (\delta^L)^t$$

Backpropagation: Multi-Label Classification

- Until now we evolved a formalism for a ***one-hot encoded output*** multi-binary classification in which ***ONLY*** one of the objects is present in the output.
- Let us now develop a formalism when ***more than one*** objects are present in the output. This can happen when ***more than one*** objects are present in a given image. This is referred as ***multi-label binary classification*** where one can have several outputs in the ***TRUE*** state for a given input data!
- In such cases, ***sum of TRUTH*** at the output can be ***>1.***
- Therefore, in such case, ***Softmax activation function CANNOT*** be used for the outermost layer since the ***sum of predicted output for the Softmax is always ONE***.
- Instead, a **Sigmoid** function is used as an activation function for ***the outermost layer*** in which any of the class is considered to be present if its Sigmoid activation value is greater than some threshold (say 0.5 for instance)
- Also, to handle multi-label classification, we need to modify cross-entropy function appropriately and it is referred as ***multi-label cross entropy function***.

Backpropagation: Multi-Label Classification

- The multi-class cross-entropy loss function is defined as:

$$J_{(x,y)} = -\sum_{m} y_m \cdot \ln(a_m^H) - \sum_{m} (1 - y_m) \cdot \ln(1 - a_m^H)$$

- For sigmoid as an activation function for the outermost layer,

$$a_n = h(z_n) = \frac{1}{1 + e^{-z_n}}$$

- *Unlike Sigmoid, a_n of Sigmoid function is only a function of z_n;* thus, to find δ for the outermost layer, all we need to consider is that:

$$\delta_n = \frac{\partial J}{\partial z_n} = \frac{\partial J}{\partial a_n} \cdot \frac{\partial a_n}{\partial z_n} \qquad \textbf{Eqn. 1}$$

Backpropagation: Multi-Label Classification

- **Derive the first term ($\partial J/\partial a_n$) in the Eqn. 1 on previous slide:**

Step 1:
$$\frac{\partial J}{\partial a_n} = \frac{\partial(-\sum_m y_m \cdot \ln(a_m) - \sum_m (1-y_m) \cdot \ln(1-a_m))}{\partial a_n}$$

Step 2:
$$\frac{\partial J}{\partial a_n} = -\left(\frac{\partial \sum_m y_m \cdot \ln(a_m)}{\partial a_n} + \frac{\partial \sum_m (1-y_m) \cdot \ln(1-a_m)}{\partial a_n}\right)$$

Step 3:
$$\frac{\partial J}{\partial a_n} = -\left(\frac{y_n}{a_n} - \frac{1-y_n}{1-a_n}\right) = \frac{a_n - y_n}{a_n(1-a_n)}$$ **Eqn. 2**

Backpropagation: Multi-Label Classification

- Derive the second term ($\partial a_n/\partial z_n$) in Eqn. 1 on the second last slide!

Step 1: $$\frac{\partial a_n}{\partial z_n} = \frac{\partial h(z_n)}{\partial z_n} = \frac{\partial (1/(1+e^{-z_n}))}{\partial z_n} = \frac{1}{1+e^{-z_n}}\left(0 - \frac{\partial(1+e^{-z_n})/\partial z_n}{1+e^{-z_n}}\right)$$

Step 2: $$\frac{\partial a_n}{\partial z_n} = \frac{1}{1+e^{-z_n}}\left(\frac{e^{-z_n}}{1+e^{-z_n}}\right) = \frac{1}{1+e^{-z_n}}\left(\frac{1+e^{-z_n}-1}{1+e^{-z_n}}\right) = \frac{1}{1+e^{-z_n}}\left(1 - \frac{1}{1+e^{-z_n}}\right)$$

Step 3: $$\frac{\partial a_n}{\partial z_n} = a_n(1-a_n)$$ **Eqn. 3**

- Substituting Eqn. 2 and Eqn. 3 into Eqn. 1, we get,

$$\delta_n = \frac{\partial J}{\partial z_n} = \frac{\partial J}{\partial a_n} \cdot \frac{\partial a_n}{\partial z_n} = \frac{a_n - y_n}{a_n(1-a_n)} \cdot a_n(1-a_n) \qquad \delta_n = a_n - y_n$$

$\delta^H = a^H - y$ → **Vector Form Representation**

Note that δ_n is same for both type of cross entropy function

Model Performance

Evaluation of Model Performance

- Accurate and easy-to-interpret evaluation methodologies are a necessary in ensuring the reliability of trained models
- **Confusion Matrix:** A confusion matrix is a very interpretable visualization that is widely used to describe the performance of a classification model on a set of test data.
- It displays the counts of ***True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN)***, successfully accounting for all the possible cases in the visualization. The confusion matrix is a N x N matrix, where N is the number of classes or categories.

Actual (Input)	Predicted (Output)					Actual (Input)
	A	**B**	**C**	**D**	**E**	
A	TN	FP	TN	TN	TN	
B	FN	**TP**	FN	FN	FN	
C	TN	FP	TN	TN	TN	
D	TN	FP	TN	TN	TN	
E	TN	FP	TN	TN	TN	
	Predicted (Output)					

Task: To evaluate model performance for say the Input value B

True Positive: Input value is B and Predicted value is also B

False Negative: Input value B but Predicted value is A/C/D/E

False Positive: Input value A/C/D/E but Predicted value is B

True Negative: All other combinations of input and output

Evaluation of Model Performance

- **Accuracy:** Accuracy is a measure of the fraction of correctly classified cases out of the total cases available in the dataset:

$$Accuracy\ A = \frac{TP}{TP + TN + FP + FN}$$

- Accuracy provides a general overview of the predictive power of the model. However, one must be cautious in using it with imbalanced datasets
- **Precision:** It's a measure of the fraction of true positive predictions among all the positive predictions made by the model:

$$Precision\ P_i = \frac{TP}{TP + FP} \qquad < P > = \sum_{i=1}^{C} w_i P_i$$

- w_i is fractional sample of class i. In simple terms, precision is accuracy of positive predictions. It's useful when the cost of false positive predictions is high, since its focus is on minimizing the false positive rate.

Evaluation of Model Performance

- **Recall (Sensitivity):** Recall is a measure of the ratio of true positive prediction to all the actual positive cases in the dataset (w_i is fractional sample of class i):

$$Recall \;\; R_i = \frac{TP}{TP + FN}. \qquad < R > = \sum_{i=1}^{C} w_i R_i$$

- **F1 Score:** The *F1 score* provides a balanced measure of a model's performance by combining two metrics: *Precision (P)* and *Recall (R)*. F1 score is the harmonic mean of precision and recall, making it useful when there is an uneven class distribution.

$$F1 = \frac{2PR}{P + R}$$

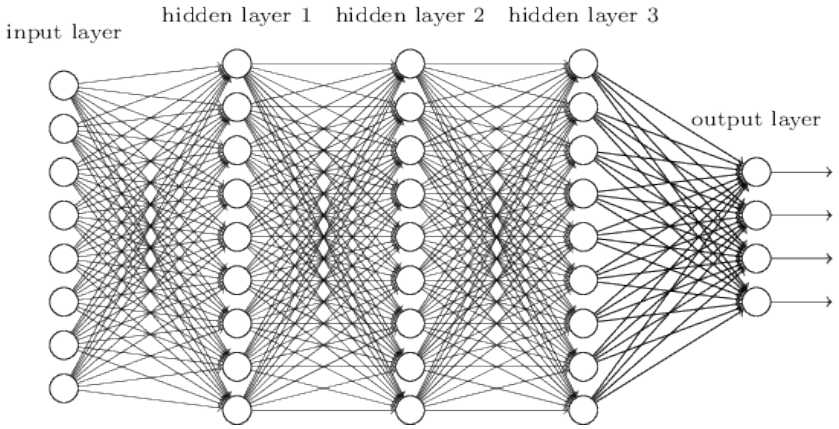
- **Specificity:** It's a measure of the ratio between true negative predictions and all actual negative cases in the dataset. Specificity focuses on accurately identifying negative cases

$$Specificity \; S = \frac{TN}{TN + FP}$$

Convolutional Neural Networks (CNN)

Convolutional Neural Networks

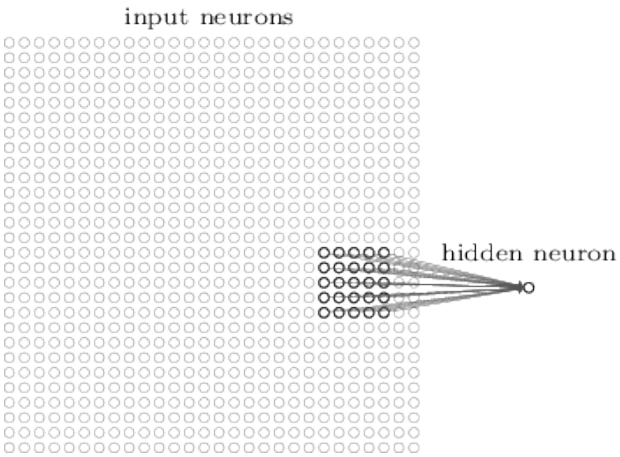
- **Fully connected network (FCN):**
  - **The *FCN* does not take into account the *spatial structure (features)* of the images that may appear in *ANY* region of the image.**
  - **It treats input pixels which are far apart and close together on exactly the same footing!**
  - **It is rather a brut-force approach to learn the image without paying attention to the spatial features.**
  - **This results in large number of weights and biases making it computational challenging**
  - **Not a suitable approach for complex inputs**



- **Instead, build an architecture that will exploit built-in features present in the image**
- ***Convolutional filters* are best to extract spatial structure *(features)* in the image during training**
- **Use of filters significantly reduce the number of free parameters *(weights and biases)* making training faster**

Convolutional Neural Networks

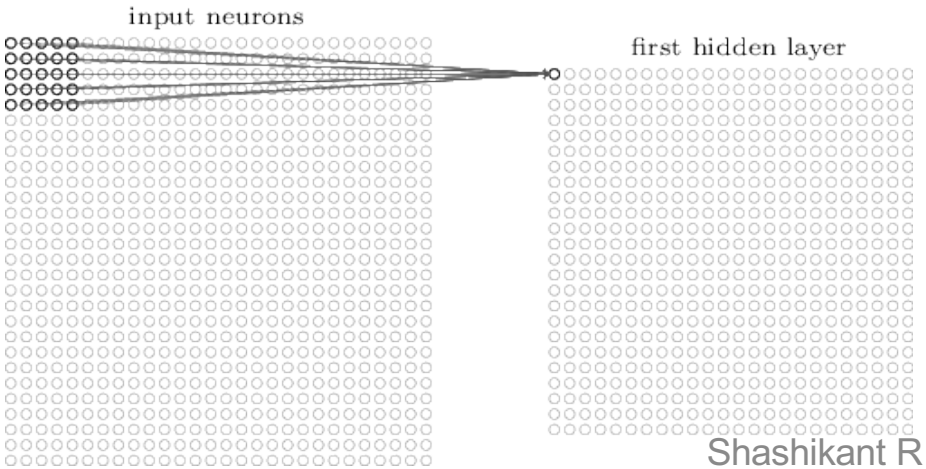
- **Convolutional Neural Network (CNN):** They are used in the most neural networks for image recognition and many other applications
- CNN uses three basic ideas: *a) Local receptive fields, b) shared weights/biases, and c) pooling*
- In the *FCN*, the inputs were depicted as a vertical line of neurons (1-dimensional input). In *CNN*, inputs are depicted *28×28 squared array* for a grey scale image *(2-dimensional input)* of neurons
- Unlike FCN, pixels in the input image are connected to only few neurons of an hidden layer representing localized regions of the input image.

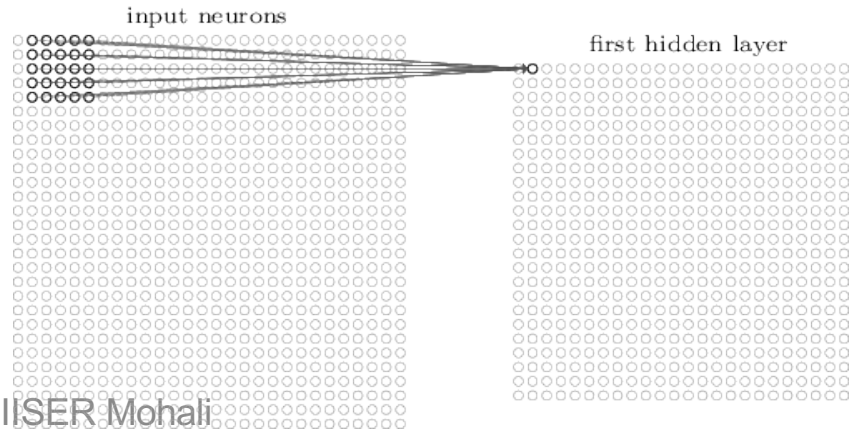


- For example, a *5×5 region*, corresponding to a *25 input pixels (NOT all input pixels)* are connected to particular hidden neuron.
- This region (*5x5 in figure*) in the input image is called the *local receptive field* for that particular connected hidden neuron.

Convolutional Neural Networks

- The value at the hidden neuron is obtained by applying ***5x5 filter*** on the inputs received from ***Local Receptive Field*** followed by the application of activation function on the ***output*** of the filter
- The ***output*** of the filter is nothing but weighted sum of inputs received from the ***Local Receptive Field plus the bias***.
- ***Weights (25) and a bias*** are stored in each hidden neuron of the hidden layer
- Slide the ***local receptive field*** across the entire input image. For each ***local receptive field***, there is a specific hidden neuron in the first hidden layer as shown below
- Size of first hidden layer with ***5x5 filter*** applied on ***28x28*** image ***(with stride=1)*** is ***24x24***





Convolutional Neural Networks

- ***Shared weights and biases:*** Each hidden neuron has a ***bias*** and ***5×5 weights*** connected to its corresponding ***local receptive field*** in the input image.
- In the CNN, ***SAME weights*** and ***bias*** are used for all the ***local receptive fields in the input image*** which means, the ***SAME weights*** and ***bias*** for each of the ***24x24*** neurons in the first hidden layer! Due to this, they are called ***Shared weights and biases***
- Such approach significantly reduces the ***number of unknown parameters*** while training
- Therefore, final value stored at the $(j,k)^{th}$ hidden neuron in the first hidden layer is given by:

$$z^L_{j,k} = \left(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{lm} a^{L-1}_{j+l,k+m} \right) \qquad a^L_{j,k} = \sigma\left(z^L_{j,k}\right)$$

- w_{lm} is matrix element (weight) in the ***5x5 filter***. ***b*** is bias. $a^{L-1}_{j+l,k+m}$ represents input to the hidden layer (the pixel intensity in the input image), **σ** is the activation function. $a^L_{j,k}$ is the value stored for the $(j,k)^{th}$ position in the hidden layer
- Note: Convolutional filter OR the weight matrix w_{lm} and the ***bias b*** is same for all the neurons in the hidden layer!

Convolutional Neural Networks

- Since all the hidden neurons in a given layer shares exactly same weights and bias, the first hidden layer detects the presence of a particular *feature* along with *its location* in the input image!
- The shared weights and bias defines *a kernel or filter*. The *filter* is often referred as the *feature detector (or extractor)*. It is useful to have the same *feature extractor* applied across the entire image.
- CNN are well adapted to the translational invariance of features embedded anywhere in the images
- Therefore, the hidden layer can also be called as a *feature map* containing information on *location* of *a particular feature* in image
- However, there can be several features in the image which needs to be extracted. Therefore, there has to be several *feature maps (hidden layers)* for *each of the features*
  - A given complete *convolutional layer* consists of several different feature maps