IDC410
A course on Image Processing and Machine Learning
(Lecture 11)

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

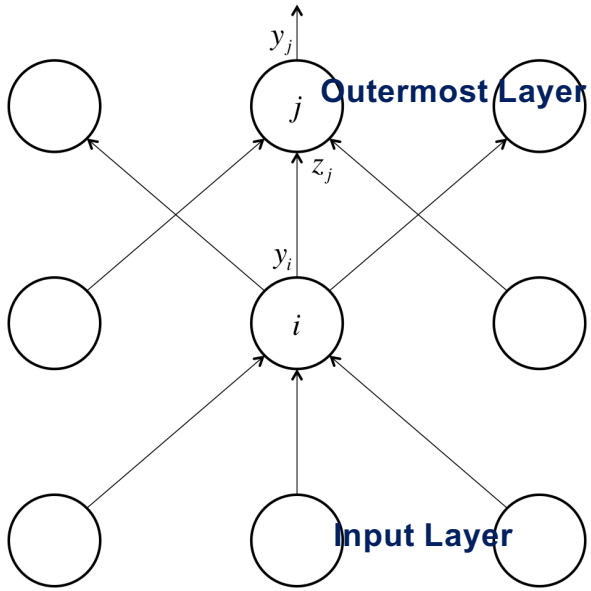
https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Network Training

Backpropagation Algorithm (Training Multi Layer Network)

- Now, we consider the case of multi-neuron output layer with a multi-neuron hidden layer between the input and the output layer
- Backpropagation algorithm for muti-layer FFN was developed by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in 1986
  - Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by backpropagating errors." Cognitive Modeling 5.3 (1988)
- Backpropagation has to deal with extremely high dimensional space.
- ***j*** and ***i*** refers to the neuron number of current and the preceding layer #, ***y*** to the activation output and z_j to the logit of the neuron ***j***
- Assuming we know the error derivatives of layer ***j***, we can use it compute the same for the previous layer ***i***
- The error function derivatives at the output layer:

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2 \Rightarrow \frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



Backpropagation Algorithm (Training Multi Layer Network)

- We can now determine how the error changes w. r. t. the weights. This gives us how to modify the weights after each training example:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial y_j}{\partial z_j} \frac{\partial E}{\partial y_j} \qquad \frac{\partial y_j}{\partial z_j} = y_j(1 - y_j) \qquad \frac{\partial z_j}{\partial w_{ij}} = y_i$$

- Therefore,

$$\therefore \frac{\partial E}{\partial w_{ij}} = y_i y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

- Finally, to complete the algorithm, just as before, we merely sum up the partial derivatives over all the training examples in our designated training dataset. This gives us the following modification formula:

$$\Delta w_{ij} = -\sum_{k \in Dataset} \epsilon y_i^k y_j^k \left(1 - y_j^k\right) \frac{\partial E^k}{\partial y_j^k}$$

Backpropagation Algorithm (Training Multi Layer Network)

- Note: $z_j = w_{ij} y_i + b_j \quad \Rightarrow \quad \frac{\partial z_j}{\partial y_i} = w_{ij}$
- **Note:** Change in a single neuron output (y_i) in the middle layer impacts logits of every neuron in the outermost layer.
- Impact of change in y_i on E can be obtained, using the fact that the partial derivative of the logit w.r.t. the output data (y_i) from the layer i, is merely the weight of the connection w_{ij}:

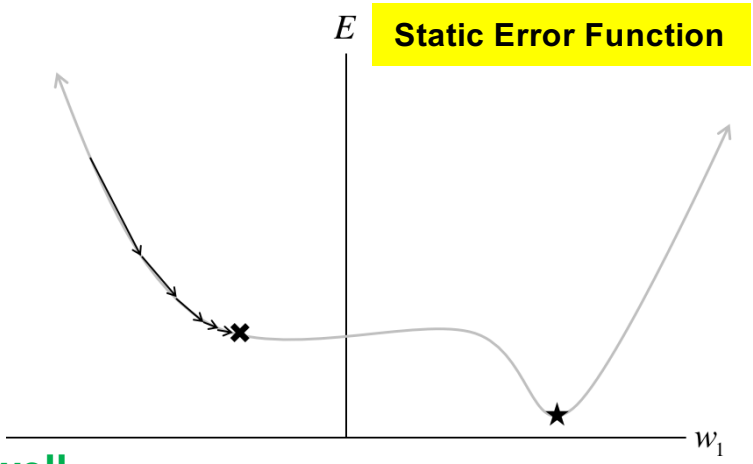
$$\frac{\partial E}{\partial y_i} = \sum_{j \in output} \frac{\partial E(y_j)}{\partial y_i} = \sum_{j \in output} \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial y_i} = \sum_{j \in output} w_{ij} y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

- **Note:** Layer h precedes Layer i and Layer i is precedes Layer j

$$\Delta w_{hi} = -\sum_{k \in Dataset} \epsilon y_h^k y_i^k (1 - y_i^k) \frac{\partial E^k}{\partial y_i^k} = -\sum_{k \in Dataset} \epsilon y_h^k y_i^k (1 - y_i^k) \sum_{j \in output} w_{ij} y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

Backpropagation: Batch Gradient Descent

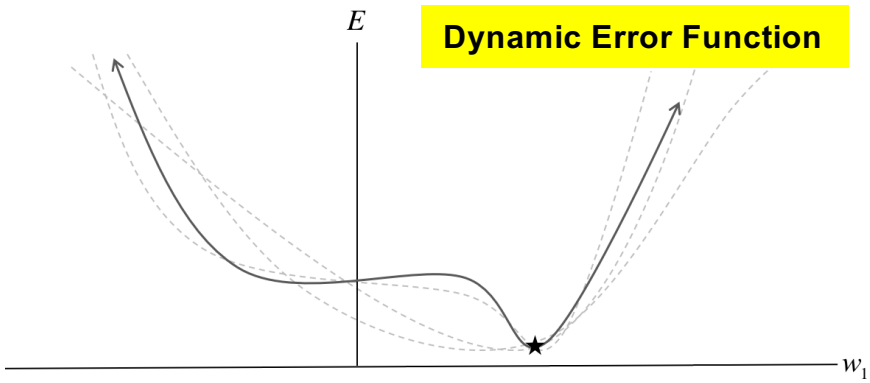
- In batch gradient descent entire dataset is used to compute the error surface
- Error surface created by the entire dataset is static in nature
- Follow the path of steepest descent to reach to minima of the error surface
  - For a simple quadratic error surface, this works quite well.
  - But what if error surface is complicated



- Consider a ONLY ONE weight scenario. Error function obtained using entire dataset can be drawn as a plane
- Batch gradient descent is sensitive to saddle points, which can lead to premature convergence

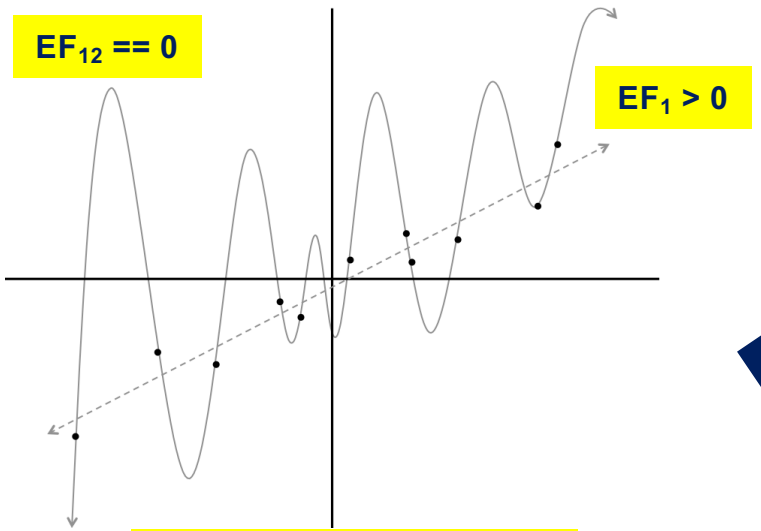
Backpropagation: Mini Batch Gradient Descent

- **Stochastic Gradient Descent (SGD)**: At each iteration, the error surface is estimated only with respect to a single example. This makes the error surface dynamic. As a result, descending on this stochastic surface significantly improves our ability to navigate stable minima and avoid convergence ay saddle point .
- **Drawback:** The error function obtained with just one example at a time, may not be a good approximation of the stable error surface. This can prolong the learning time,
- **Minibatch Gradient Descent**: The error surface is computed w. r. t. some subset of the total dataset (instead of just a single example). The stochastic error surface fluctuates w. r. t. the batch error surface, enabling saddle point avoidance
- This subset is called a **minibatch**. The Minibatches strike a balance between the efficiency of batch gradient descent and the local-minima avoidance afforded by stochastic gradient descent.



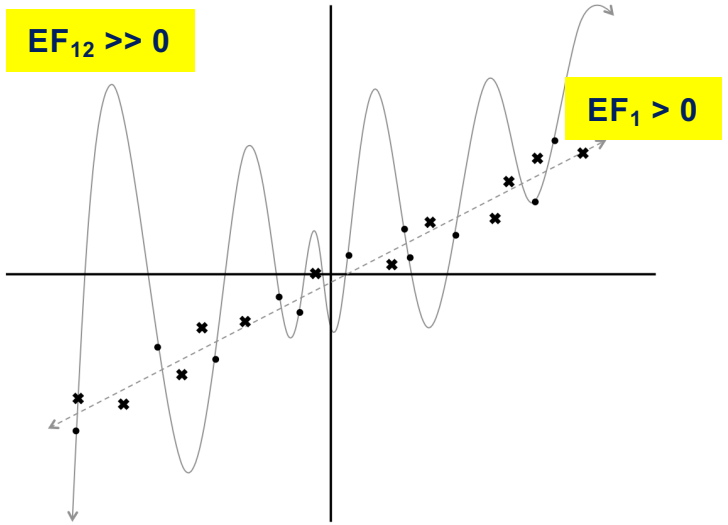
Backpropagation: Challenges

- A simple MNIST database (28 x 28 pixels) with two hidden layers with 16 neurons, and an output layers with 10 neurons has >10000 unknown weights! To determine such large number of weights in *faithful* manner can be quite challenging.
- Consider a case of single weight example with 12 data points. Error function (EF) is described using approaches: a) A polynomial model of 12 degrees and b) A linear model
- During training: $EF_{12} == 0$, $EF_1 > 0$ ➔ $EF_1 > EF_{12}$ During evaluation: $EF_{12} >> 0$, $EF_1 > 0$ ➔ $EF_1 << EF_{12}$!!!



Training with two models

Over fitting!!!



Evaluating each model with new data points

Backpropagation: Overfitting → Data Split

- The linear model is not only better subjectively but also quantitatively ($EF_{12} >> EF_1$)
- By building a very complex model, it's easy to perfectly fit the training dataset as the model provides extremely large # of degrees of freedom to forcibly fit the observations in the training set.
- In other words, such **complex models** are not well generalized. This is a phenomenon called ***OVERFITTING***, and it is one of the biggest challenges in training neural networks containing large number of connections with unknown weights associated with each connection
- In training process, there is a direct trade-off between overfitting and model complexity.
  - Complex models are needed to capture all of the useful information necessary to solve a problem. However, complex models (especially with a limited dataset at our disposal) can run the risk of overfitting.
- **Possible solution:** Do not evaluate a model using the data used for training. Instead, split the data into training set and test set. This enables us to make a fair evaluation of our model by directly measuring how well it generalizes on new data which it has not yet seen
- **Warning:** Without split in the dataset (training and test data), we won't be able draw any meaningful conclusions about our model.

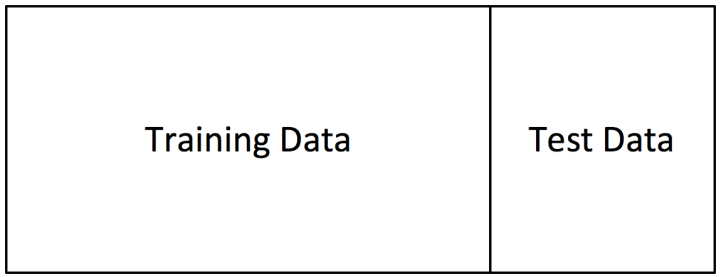
Backpropagation: Overfitting → EPOCH

- If we have very large dataset, then, at some stage of training, instead of learning useful features, we may start overfitting to the training set.
- By building a very complex model, it's easy to perfectly fit the training dataset as the model provides extremely large # of degrees of freedom to forcibly fit the observations in the training set.
  - To prevent poor generalization despite large dataset, we should have strategy to stop the training process before it starts overfitting
- To accomplish this, we introduce a concept of ***EPOCHS***. An epoch is a single iteration over the entire training set. Each time an entire dataset passes through an algorithm, it is said to have completed an epoch.
- Furthermore, if we have a training set of size D and we are doing mini-batch gradient descent with batch size B, then an epoch would be result in about D/B model updates (Weights are updated D/B times in each epoch).
- The **number of epochs** is considered as an hyperparameter which defines the number of times the entire training data set has to be worked through the learning algorithm.

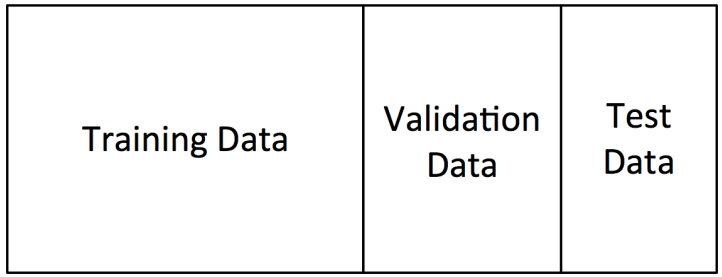
Backpropagation: Overfitting → EPOCH

- In order to quantify, how well our model is generalizing at the end of each epoch, we use an additional dataset referred as **validation dataset**. At the end of the epoch, the **validation dataset** will tell us how well the model does on the data it has yet to see.
- At the end of each epoch, if errors on the training dataset as well as validation dataset are decreasing then we continue with the next epoch
- However, if the accuracy on the training set continues to increase while the accuracy on the validation set stays the same (or decreases), then, it's a sign to stop the training because we are now entering in overfitting phase!!!
- Finally, if we are not satisfied with the final model performance, then, we need to rethink about the a) inadequate size of the data, b) architecture itself or c) whether the dataset actually contains the information required to make the prediction we're interested in making
- **Hyper parameters:** Learning Rate (0.1, 0.01, 0.001, …), Mini-batch size (16, 32, 64, 128, …), Epochs(~100-1000). It is possible to dynamically retune learning rate and mini-batch size at every epoch.

Backpropagation: Flowchart



Training without concept of EPOCH



Training with concept of EPOCH