NN Final 2018 & 2019

1. The machine learning Recipe

1. Train & evaluate network on training data ← Bad Perf. on training data
   If (underfitting) # High Statistical Bias
   - Consider a bigger network (more hidden nodes/layers) #most of the time works
   - Train longer #works sometimes
   - Check for bias again & keep changing until good bias is reached.

2. Evaluate on validation data ← Bad Perf. on validation data
   If (overfitting) # High Statistical Variance
   - Consider more data
   - Regularization
   - Check for bias again, then for variance & so on till good bias & variance are reached.

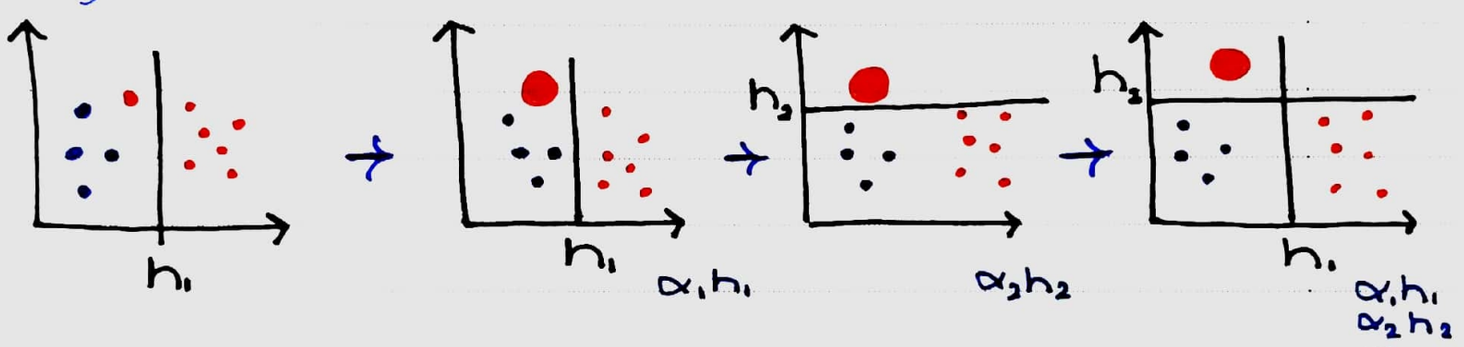
- Search for a better NN architecture #works sometimes

2. The Key Idea of Adaboost + Diagram

- Sequentially combine many weak classifiers in order to realize a strong classifier that approximates Bayes Classifier (optimal Decision).

- Key Idea (Slides)

Like Rec. but just 2 h's



→ each classifier emphasizes the mistakes of the previous one and in the end a weighted combination of them decides the output.

⟨Bonus⟩

1. $w_m = \frac{1}{M}$ $1 \le m \le M$

2. For $1 \le t \le T$:
3. $h_t = \text{train}(\text{weights}, \text{data})$
4. $err_t = \sum_{m=1}^{M} w_m \cdot I(h_t(x_m) \neq c_m) \,/\, \sum_{m=1}^{M} w_m$
5. $\alpha_t = \log\left(\frac{1 - err_t}{err_t}\right)$
6. For $m = 1$ to $m = M$: $w_m = w_m \cdot e^{\alpha_t I(h_t(x_m) \neq c_m)}$
7. $w_m = w_m / \sum_{m=1}^{M} w_m$ $1 \le m \le M$
8. Output $C(x) = \underset{K}{\text{argmax}} \sum_{t=1}^{T} \alpha_t I(h_t(x) = K)$

</Bonus>

3 - Why Can't Adaboost work for 2 classes

- α_t, which signifies the influence of each classifier should be zero (no influence) when the classifier is as bad as random guessing & +ve otherwise.

→ For K classes, the accuracy by random guessing is $\frac{1}{K}$ $\left(err = 1 - \frac{1}{K} = \frac{K-1}{K}\right)$

Hence,

$$\alpha_t = \log\left(\frac{1-err_t}{err_t}\right) = \left(\frac{1-(K-1)/K}{(K-1)/K}\right)$$

$$= \log\left(\frac{1}{K-1}\right)$$

$$= -\log(K-1)$$

- This only evaluates to zero if $K=2$

Thus, we can generalize the formula to be

$$\alpha_t = \log\left(\frac{1-err_t}{err_t}\right) + \log(K-1)$$

- and now $\alpha_t = 0$ for random guessing and $\alpha_t > 0$ otherwise.

Q4) Wrapper VS. Filter Method

Wrapper Method

- takes classifier into account while seeking the optimal feature set
- Accurate but slow & lacks generality
- e.g., Sequential Forward Selection

Filter Method

- Does not take specific classifier into account & rather relies on info content, correlation, class means distance & other statistics
- Fast & more general but tends to select large subsets
- e.g., apriori

Q5) AI VS. ML VS. DL

AI: Any technique that enables computers to mimic human intelligence using logic, if then rules, decision trees, ML and DL algos

ML: Subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. Includes DL

DL: Subset of ML that includes algorithms that enable software to train itself on tasks like speech and image recognition by exposing multi-layered Neural Networks to vast amounts of data.

<Bonus>

- Note that deeplearning algorithms can directly train on raw data without prior feature extraction (can be end-to-end) but require more data volume & computational power.

Classical AI: AI excluding machine & deep learning used when no learning/data are needed (simple applications).

</Bonus>

6- Next step after training an NN and getting 54% training accuracy and 51% validation accuracy.

→ The neural network is severely underfit (huge bias) so we should consider.

- larger network (layers/neurons)
- longer training time

- If neither that nor changing the architecture helps we should consider substantial changes in terms of the features / choice of model.

- Note that with such bias, it was pointless to check performance on validation set in the first place.

7- Linear Perceptron Update Rule

desired output (0 or 1)

actual output

augmented inputs

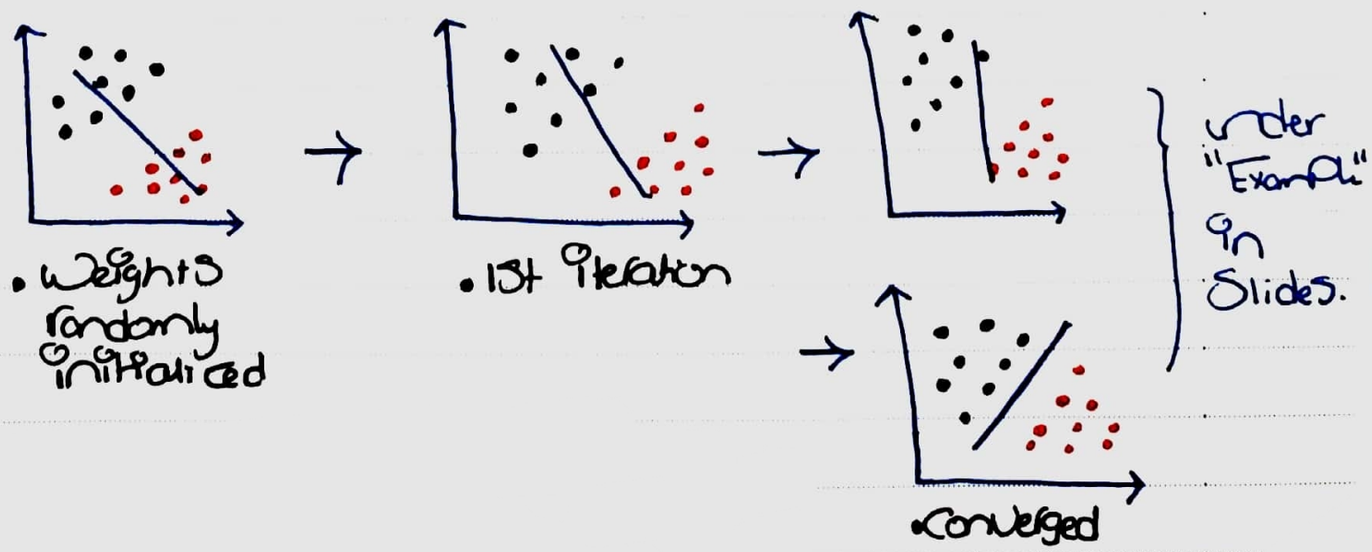
$$\underline{w}_{new} = \underline{w}_{old} + \alpha\,(d(n) - y(n))\,\underline{u}(n)$$

- The goal is to find $\underline{w} = (w_0\ w_1\ \dots\ w_N)$ such that

$$w_0 + w_1 x_1 + \dots + w_N x_N = 0$$

Perfectly seperates the two classes.

→ Each weight update changes the hyperplanes shift/orientation until it classifies all points correctly

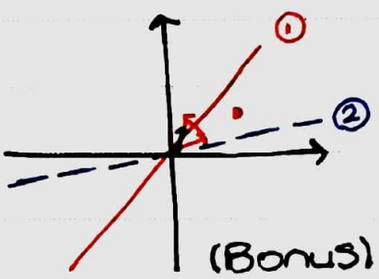


- In case $d(n) = y(n)$

$$\underline{w}_{new} = \underline{w}_{old}$$ (Correctly classified, no update)

- $d(n) = 1$, $y(n) = 0$ $\quad (d(n) - y(n)) = 1$
  - Should be above
  - actually below

$$\underline{w}_{new} = \underline{w}_{old} + \alpha \underline{u}(n)$$



- $d(n) = 0$, $y(n) = 1$

$$\underline{w}_{new} = \underline{w}_{old} - \alpha \underline{u}(n)$$

- We get a better hyperplane

<Bonus>

1. Initialize weights $w_0, w_1, \ldots, w_N$ randomly
2. While training accuracy < 100% :
3. For $n = 1$ to $n = N$:
4. Present $u(n)$ and corresponding $d(n)$
5. $y(n) = \underline{w}^T \underline{u}(n)$
6. $\underline{w}_{new} = \underline{w}_{old} + \alpha (d(n) - y(n)) \underline{u}(n)$

</Bonus>

- I doubt if the derivation for "takes the right step each update" is needed but you can write that.

8. Naïve Estimator

1. Associate with each training Pattern a bin of width Δ
2. Associate with each bin the Probability

$$P(X=x) = \frac{n_x}{\Delta M}$$

(n_x: No. of examples falling into bin)

and add any overlaps.

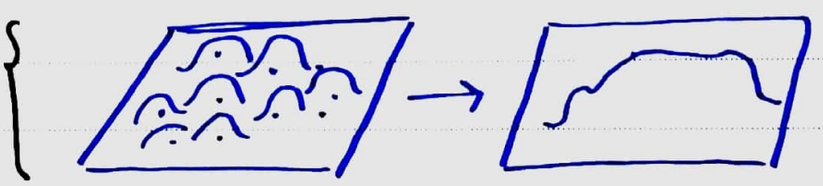
- Histogram analysis applies the same formula but uniformly Partitions the range of the dataset to Produce the bins first.
  $\Rightarrow$ Naïve estimator guarantees that the bins are centered at training Pattern and hence results in less bias.
- Both correspond to a discontinuous density estimate although the underlying one is smooth.

9. Kernel Density Estimate

1. $$\hat{P}(\underline{X}) = \frac{1}{MH^N} \cdot \sum_{m=1}^{M} g\left(\frac{\underline{X} - \underline{X}(m)}{h}\right)$$

$\cdot \varphi_n(\underline{x}) = \frac{1}{h^N} g\left(\frac{\underline{x}}{h}\right)$

- Places a bump function $g(x)$ at each Point
  → Need $\int_{-\infty}^{\infty} g(\underline{x})\, d\underline{x} = 1$



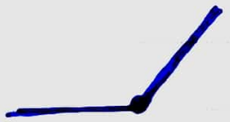
Transforms them all

- Typical Choice for $g(\underline{x})$:

$$g(\underline{x}) = \frac{e^{-x^T x}}{(2\pi)^{N/2}}, \quad \hat{p}_n(\underline{x}) = \frac{e^{-\frac{x^T x}{h^2}}}{(2\pi)^{N/2} h^N}$$

- The BW h acts as the variance.

10. Why is Relu better than Sigmoid

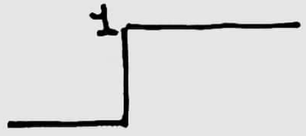
$$ReLU(x) = \text{  }$$

$$= \max(0, x)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$= \text{  }$$

$\left(\frac{d}{dx}\right.$



$\frac{d}{dx}\left(\right.$

- Sigmoid suffers from slow learning because it has weak gradients (that even vanish for large values) meanwhile Relu has steady derivatives and thus exhibits much faster learning
- Besides, Relu is easier to implement (involves no exponentiation) and empirically works well (esp. learning from images.

11. The Least Square Classifier

→ Goal is to find $w_0, w_1, \dots, w_n$ such that $y = \underline{w}^T \underline{u} = w_0 + w_1 x + \dots + w_n x$ is a hyperplane that predicts the continuous target variable y so that classification can be performed by setting on it a threshold (e.g., $y > 0 \rightarrow C_1$ and $y < 0 \rightarrow C_2$).

- This is done by minimizing the loss function

$$J = \frac{1}{M} \sum_{m=1}^{M} (\underbrace{y(m)}_{\underline{u}^T \underline{w}} - d(m))^2$$

$$= \frac{1}{M} (U\underline{w} - \underline{d})^T (U\underline{w} - d)$$

$$\rightarrow \begin{pmatrix} \underline{u}^T(1) \\ \underline{u}^T(2) \\ \vdots \end{pmatrix} \quad \rightarrow \begin{pmatrix} d(1) \\ d(2) \\ \vdots \end{pmatrix}$$

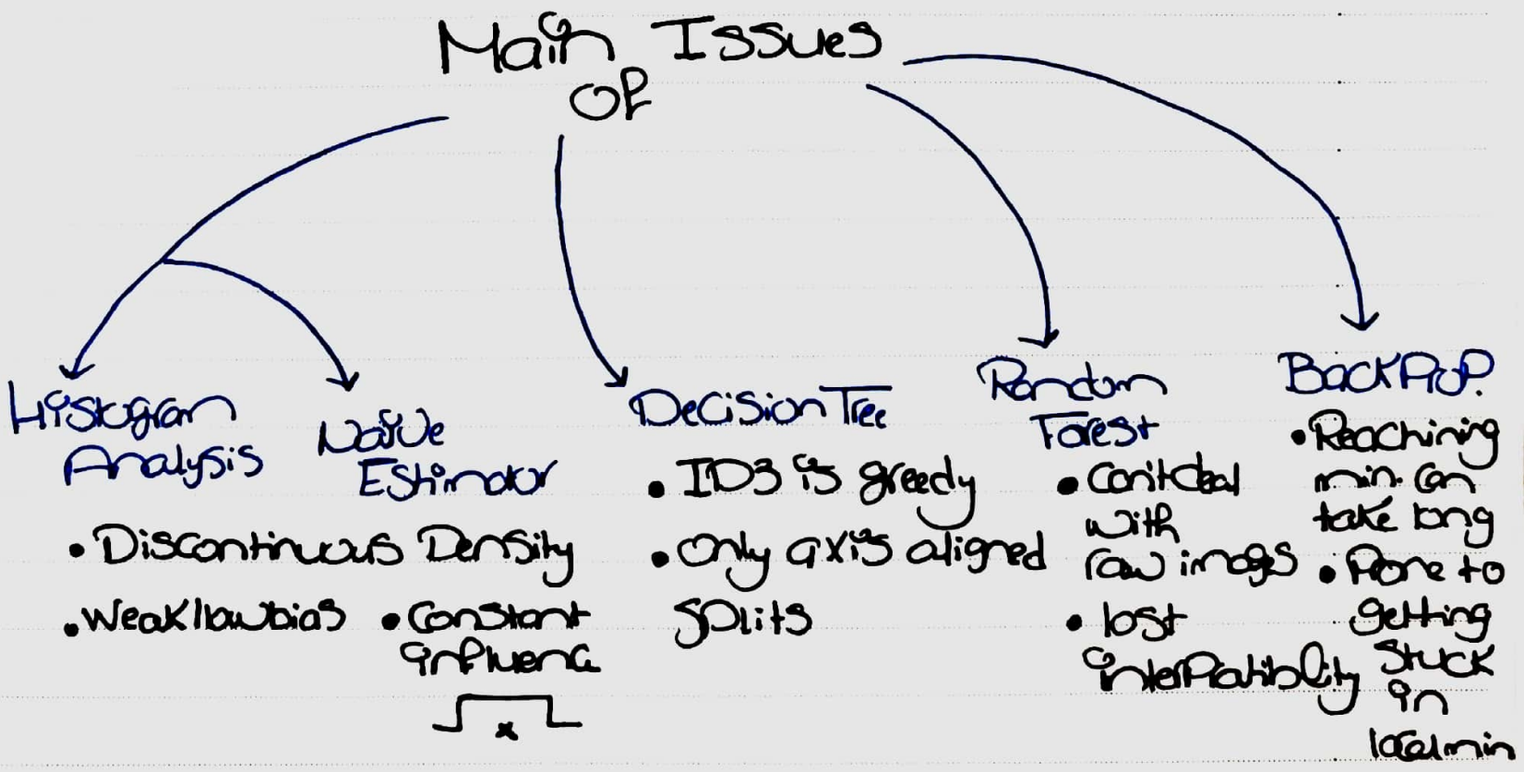
→ By $\frac{\partial J}{\partial w} = 0$ we get

$$\underline{w} = (U^T U)^{-1} (U d)$$

which is the $\underline{w}$ for the unique hyperplane that minimizes the loss.

12. Main Issues of GMMs

1. Initialization
   → Because the underlying function to be optimized is non-convex, GMMs are sensitive to initialization.
   • Bad initialization → Bad estimate
   #Can use K-means to help initialize.
2. Need to decide # components
   → Can either try different values and take best according to validation set or use information theory.



13. Types of Features to be removed

- Irrelevant Features: Don't help discriminate between the two classes (e.g., height for male vs. female)
- Correlated Features: Vary very closely with each. The presence of one of them should be enough. (e.g., area and perimeter of O)

14. Regularization & why its used.

→ Technique used to prevent overfitting which often occurs when we have a large network with many parameters.

- Can be achieved by
  - * Adding a regularization term to the loss

$$J = \sum_{m=1}^{M} J_m + \frac{\lambda}{2M} \|\underline{W}\|_2^2$$

($\underline{W}$: vector of all network weights)

  - * Dropout regularization where nodes (each) in a layer may drop out each iteration according to a set probability.

Both methods in Principle simulate the effect of training a smaller network.

15. Time Series Prediction

De Seasonalize:

1. For each season

$$a(\text{year}) = \frac{1}{12} \sum_{\text{window}} x(t)$$

For each month

$$z(i) = \frac{x(i)}{a(\text{year})}$$

For each month

$$u(i) = \frac{\sum_{\text{years}} z(i)}{\#\text{years}} \qquad \left(\sum_{j=1}^{\#\text{years}} z_j(i)\right)$$

$$x(t)|_{des} = \frac{x(t)}{u(\text{month}(t))}$$

Predict Trend: $\hat{x}(t)|_{des}$

Recover Seasonality:

$$\tilde{x}(t) = \hat{x}(t)|_{des} \cdot u(\text{month}(t))$$

16.

1. For each class C_i assume & find Parameters (e.g. Gaussian) or estimate the conditional density $P(\underline{X}|C_i)$

2. Estimate $P(C_i)$ from dataset / real world

3. Given X, Predict the class K where

$$K = \underset{i}{\operatorname{argmax}}\, P(\underline{X}|C_i)P(C_i)$$

This is Bayes classifier which comes from Bayes Rule

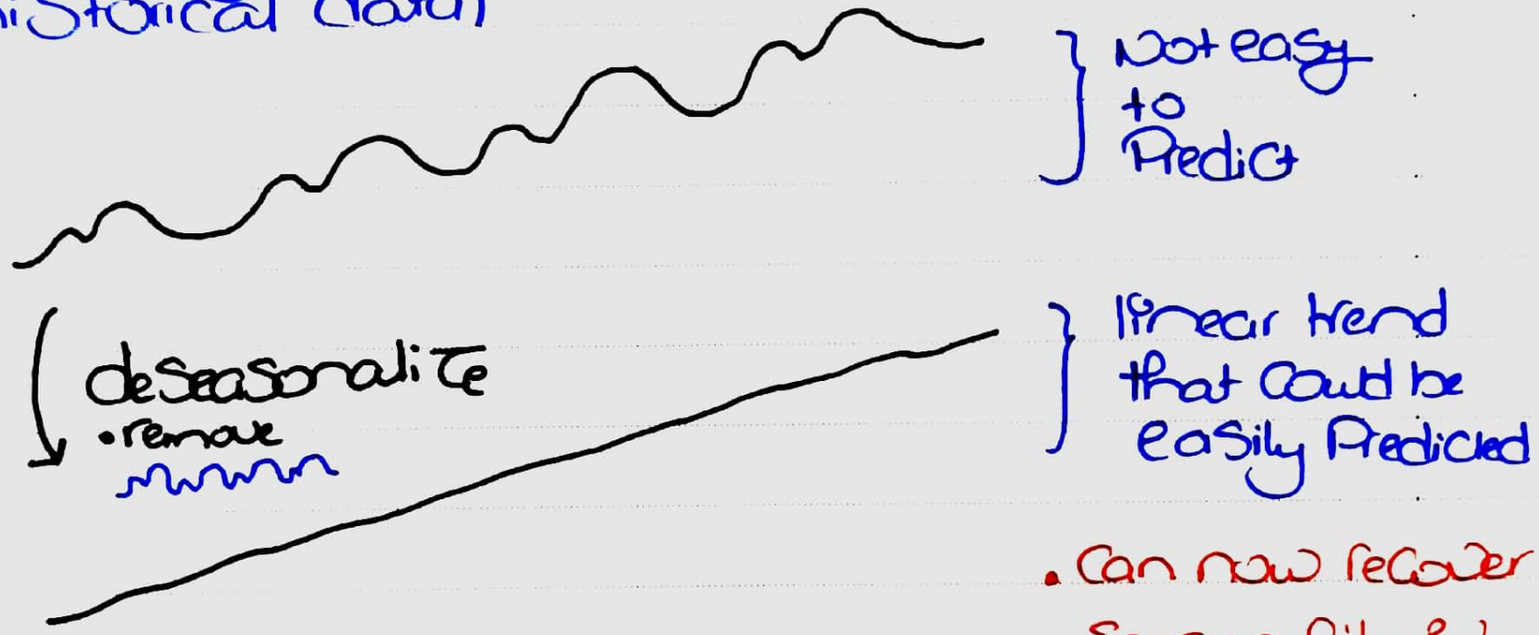
$$P(C_i|\underline{X}) = \frac{P(\underline{X}|C_i)P(C_i)}{P(\underline{X})}$$

→ Given a new sample $\underline{X}$ we compute the probability of each class being the corresponding one then taking the highest

→ We can ignore the marginal $P(\underline{X})$ since it's common to all evaluations. This corresponds to the Bayes classifier.

17. Explain the importance of deseasonalization using diagrams

- It allows us to perform time series forecasting (Predicting future values based on time-stamped historical data)



} Not easy to Predict

} linear trend that could be easily predicted

- Can now recover seasonality & make predictions

18. BackPropagation Steps & Disadvantages

1. Randomly Initialize all weights & biases (e.g., in $[-1,1]$)

2. For $m=1$ to $m=M$:

3. Forward Pass

Feed $u(m)$ & find layer outputs

$$\ell=1,\dots,L \begin{cases} X_i^{(\ell)} = \sum_{j=1}^{N(\ell-1)} w_{ij}^{(\ell)} Y_j^{(\ell-1)} + w_{i0}^{(\ell)} \\ Y_i^{(\ell)} = P(X_i^{(\ell)}) \end{cases} \quad 1 \le i \le N(\ell)$$

4. Backward Pass

Use $d(m)$ & forward pass outputs to compute $\partial J_m / \partial w_{IJ}^{(\ell)}$ for all layers i.e.

$$\ell=L,\dots,1 \begin{cases} \cdot\ \partial J_m / \partial X_I^{(\ell)} & 1 \le I \le N(\ell) \\ \cdot\ \partial J_m / \partial w_{IJ}^{(\ell)} & \forall I,J \end{cases}$$

5. Update weights

$$w_{IJ}^{(\ell)} = w_{IJ}^{(\ell)} - \alpha\, \partial J_m / \partial w_{IJ}^{(\ell)} \quad \forall I,J,\ell$$

6. Compute total loss and stop once converged.

Disadvantages:

- Can take long to reach the minimum (esp. when close)
  - α is low → too slow
  - α is large → oscillations/ not converging

  Decaying α is a solution
- Prone to getting stuck in local minima
  - * loss is highly non-convex

  → Solution can be to repeat training many times @ different weight initializations

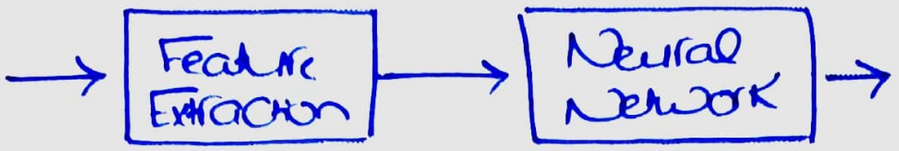
19. Three Methods to update weights

$$w_{new} = w_{old} - \alpha \sum_{n=1}^{x} \frac{\partial J_n}{\partial w_{old}}$$

x

- M: Full batch Gradient Descent
  - → Slow but consistent
  - * Consider whole dataset for a single update
- 1: Stochastic GD
  - → Fast but prone too oscillations/ bouncing around min.
  - → Doesn't make use of all parallelization possible
  - * Considers single example each step
- M': Mini-batch (Size is power of 2)
  - → Faster & more consistent
  - * Consider fraction of dataset M' each step

20.



- CNNs Precede the Feedward layers with feature extraction layers that extract features in light of minimizing the loss

Two types of layers

Convolutional of K filters each PxP

→ Objective is to learn features. Each feature has as much channels as input so each pair is convolved then added (along with bias) to produce a feature map

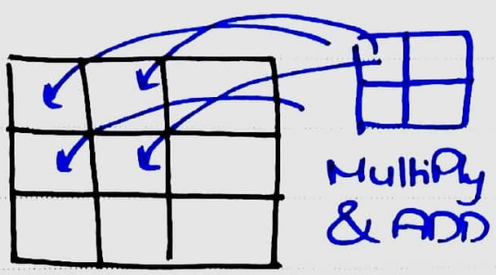
Pooling layer with QxQ window

→ Objective is to summarize the feature map by avg-ing or max-ing over a window (no learning)

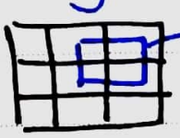
→ When window is as large as feature map it converts d-channel feature map to d-feature vector #Global Pooling

Suitable input for NN

- Convolution



- Pooling



Less Memory FootPrint

Parameter Sharing
- Same filter applied on different feature map Pixels
(not that each has its own weights)

Sparsity of Connections
- output depends on a limited no. of inPuts (below filter).
(not that it depends on all of them)

21. Three Limitations of Deep Learning

1. Unintended results from fitness function (can minimize loss in expected ways)

2. Sensitive to Standard Adversarial Attacks
( +  → wrong classification)
Img
Adversarial Perturbation

3. Oversensitive to changes in context
(dataset has limited no of them, in real world its combinatorially large)

4. Large Amount of labeled data (expensive)

23. K-Fold Cross Validation

1. Partition the dataset into K folds
2. Train the model K times on each consider K-1 folds only then validate on the last one
3. Average the K resulting performance estimates

- It's used to arrive at (statistically significant*) performance estimate of the model. In this sense it make the validation set only optional
- Can hence be used to tune hyperparameters

My thanks <3