CO2865
Deep Learning
August 17, 2017

Logistics:-

① Monday August 11, 2017:- | Thursday August 7, 2017
CNNs (advanced) | CNN: Basics
By Chetan Arora.

4.5 weeks:- Basics | Foundation

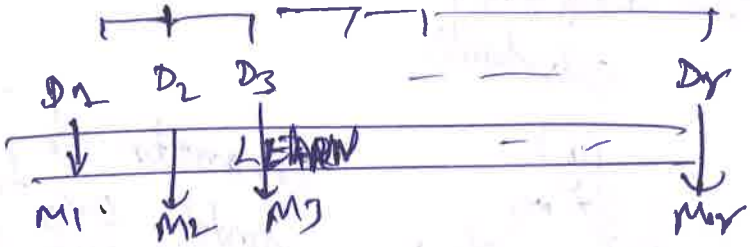
5.5 weeks
- 1.5 weeks:- CNNs
- 1.5 weeks:- RNNs
- 1.5 weeks:- GANs | Autoencoders
- 1 week:- Deep RL

4 weeks: Student Presentations.

Ensemble Methods:-
Combine multiple different models & their outputs.

Example:-
Bagging

Data: $\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$



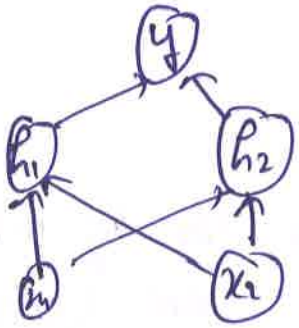
M_i:- $P(y|x; \theta_i)$

Then, final prediction:-

$$\frac{1}{r}\sum_{i=1}^{r}\left[P(y|x, \theta_i)\right]$$

In general can be-
① Bagging
② Random Initialization
③ Random selection of mini-batches
④ Dropout

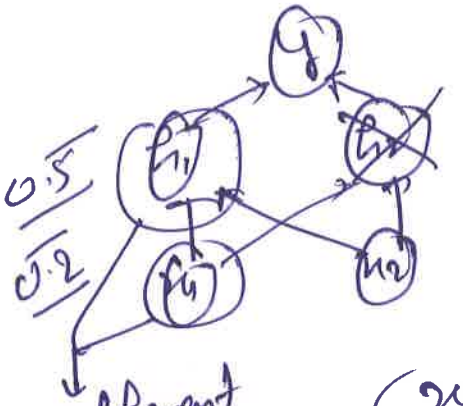
Next



4 non-output units:-
For each such unit, consider dropping it out independently.

16 possible networks

Example:-



Different probabilities of being dropped out

Note:- Explicitly constructing all such sub-networks may be expensive.

Idea:- Use in conjunction with mini batch algorithm

$(x^{(1)}, y^{(1)})$ - - $x^{(t)}$ - $(x^{(m)}, y^{(m)})$

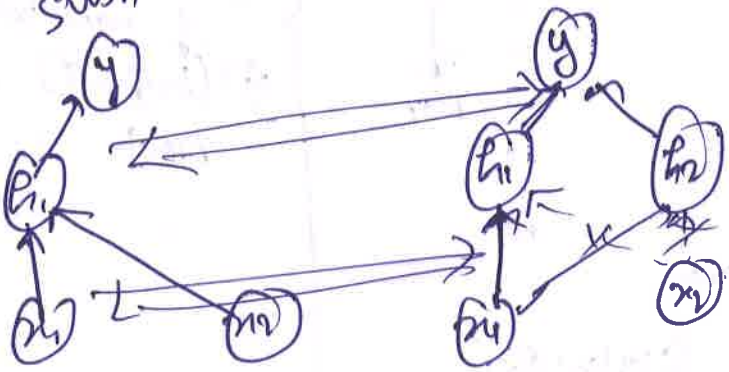
t-k sized minibatch (independent examples) Batch-wise independent of each other

① construct a sub network using random sampling
② compute gradient, ∏[Forward pass, Backward pass]
③ Proceed to next minibatch

] stochastic gradient descent

Difference:- (from Bagging)

Subnetworks



Subnetworks

Parameters of each model are tied with each other.

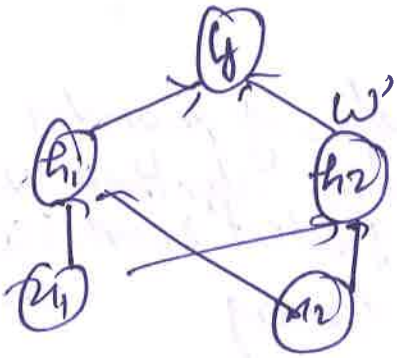
Therefore running single iteration (each minibatch) with a subnetwork is also etc.

(2)

Why does it / should it work?
What is the form of regularization that it is introducing:- It is reducing over dependence on a specific architecture.

Inference:- Need to average over models (implicitly represented)
↳ A very large # of them

Idea:-



$w' = p * w$
p:- probability of dropping w.
Weight scaling inference rule

$$P_{\text{ensemble}}(y|x) \doteq \left(\prod_{\mu} P(y|x;\mu) \right)^{1/2^d}$$

μ ↓ possible submodels

d # units to be dropped

$\equiv$ weight scaling inference.

Why does it work?:-

Robust set of features

Drop inputs or hidden units

Provide additional enhancements compared to independently trained ensembles.

Adversarial Training

Read from Book

Optimization:- (Done with Modeling + Regularization)

$J(\theta) + \Omega(\theta)$ ↳ Regularizer

↓ cost function (-ve log-likelihood)

↳ ① Why different from pure optimization
② Challenges
③ Specific Algorithms
↳ ① Initialization of parameters
② Optimization algorithms (variations) ↳ gradient based / second order
③ ↳ Adaptive learning rate

$J(\theta) = E_{(x,y) \sim \text{Data}} L[f(x,y);\theta)]$

Data → $\hat{p}_{data}$ ↓ per-example loss

$\frac{1}{m} \sum_{i=1}^{m}$ ↑ m

Proxy for actual performance metric that you care about (generalization error)

$J^*(\theta) = E_{(x,y) \sim p^*_{data}} [L(f(x;\theta); y)]$ ↓ If we knew this, then pure optimization problem

③

$\Rightarrow \arg\min_{\theta} J(\theta) \rightarrow$ Training Loss $= \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)})$

↓

Empirical Risk Minimization

How is it related to true risk / generalization error

PAC Learning

↳ Problem of Overfitting

Issues:- May care about 0-1 Loss
↳ Not differentiable

Optimize Surrogate Loss :-

Negative log-likelihood

↳ Another Idea:-

Early Stopping

↳ [convergence criteria
↳ Based on a validation set]

Batch / Mini Batch Algorithms :-

Batch gradient Descent

$\frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)})$.

$\Leftrightarrow$

Minibatch / Stochastic gradient Descent

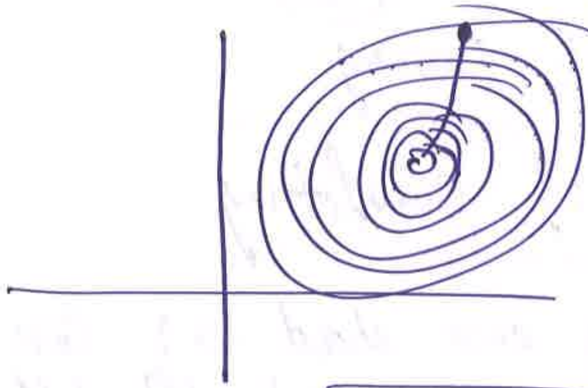
$\frac{1}{\gamma}\sum_{i=1}^{\gamma} L(f(x^{(i)};\theta), y^{(i)})$

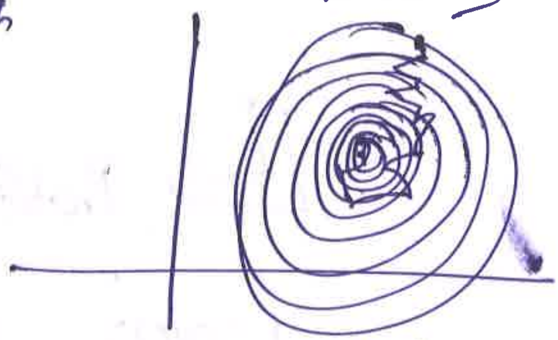
[① Divide into m/γ batches
② Could be randomly chosen]

Significant computational advantages

→ ① Each iteration $O(m)$ time
(Batch gradient Descent)

② Each iteration takes $O(r)$ time
(approximate gradient)





Stochastic
Online

⇓ Offers regularization

Not practical
for most applications

Most Algorithms

$[1 < r < m]$
Mini Batch

Gradient → estimate $E[x]$

$\hat{\mu} = 1/m \sum_{i=1}^{m} x^{(i)}$ i.i.d.

$\hat{\mu} = 1/m \sum_{i=1}^{m} x^{(i)}$

$E[\hat{\mu}] = \mu$

~~$\sigma[\hat{\mu}] =$~~

$Var[\hat{\mu}] = 1/m^2 \sum_{i=1}^{m} Var(x^{(i)})$

$= \frac{1}{m^2} \sigma^2 m = \frac{\sigma^2}{m}$

⇒ Standard Deviation in estimate of $\hat{\mu}$ $= \frac{\sigma}{\sqrt{m}}$ → goes down inversely quadratic

But m times more computation

(4)

Considerations:-

① Lager Batch size ⇒ more accurate estimate of gradient

② Parallelization within batch?
↳ prefer Lorger batches
↳ certain batch sizes may be preferable (powers of two)

③ Regularizing effect: small batch sizes

④ Kind of algorithms-
① Gradient Band ✓
② second order $H^{-1}g$
↳ small errors in estimating g can magnify ⇒ larger batch

⑤ Example in each batch should be uncorrelated (independent):-

① Similarly examples across batches be independent from each other

⑦ When examples are not repeated.
stochastic gradient descent is a unbiased estimate of true gradient
way?

$$\hat{J}(\theta) = (x,y) \sim \hat{p}_{data} \left[\nabla_\theta L(x, \theta, y) \right]$$

Variance may be high

Challenges in Neural Network Optimization :-

Non-Convex optimization

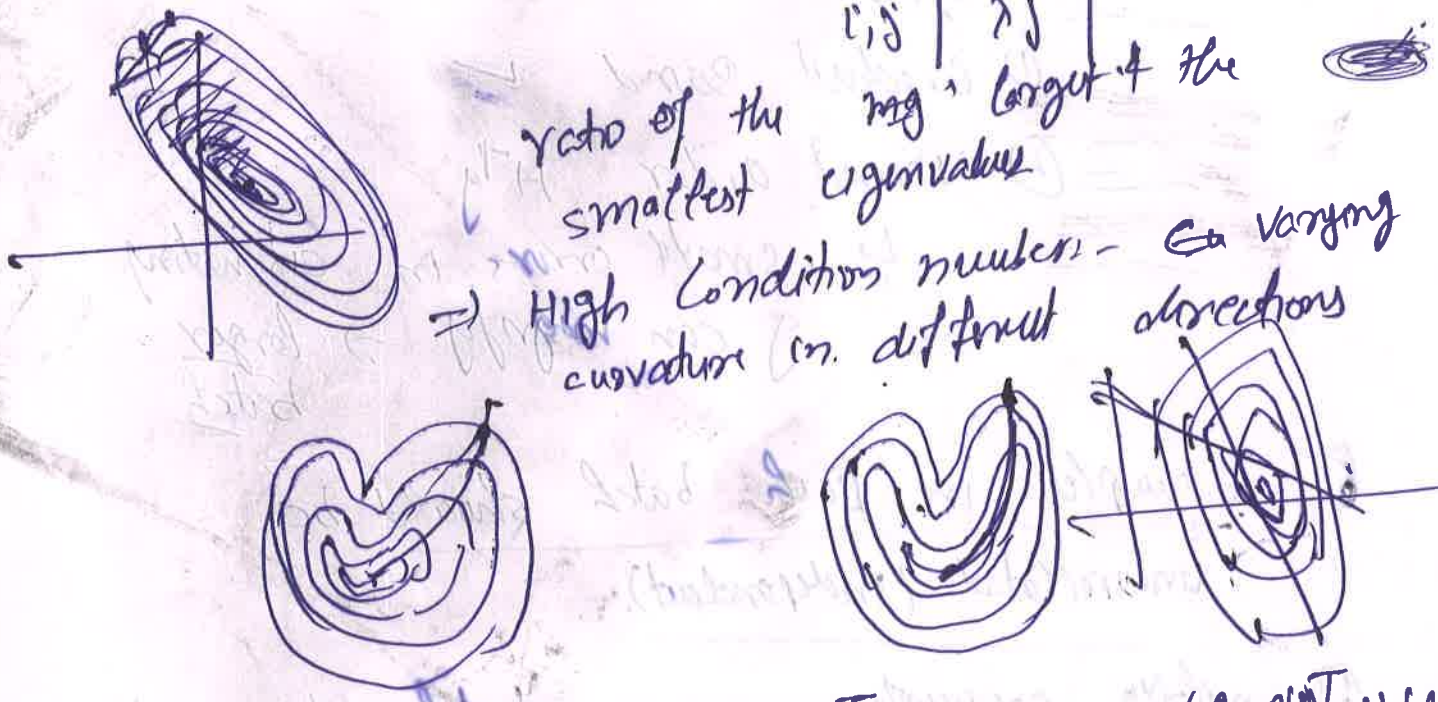
① ill-conditioning of Hessian Matrix :-

Conditioning Number :-

Matrix :- H (Hessian Matrix)

Condition Number :- $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$

ratio of the mg. largest & the smallest eigenvalue



⇒ High Condition number :- ∈ varying curvature in different directions

$$J(\theta) = J(\theta^{(0)}) + (\theta-\theta^{(0)})^T \underbrace{\nabla_\theta J(\theta)}_{g} + (\theta-\theta^{(0)})^T H (\theta-\theta^{(0)})$$

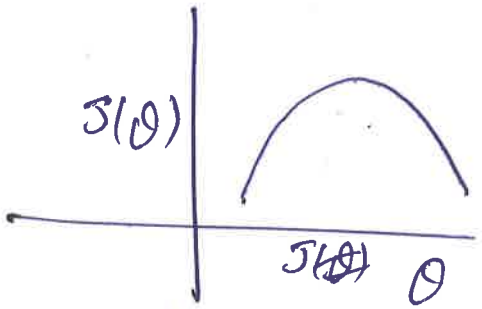
Learning rate :- η

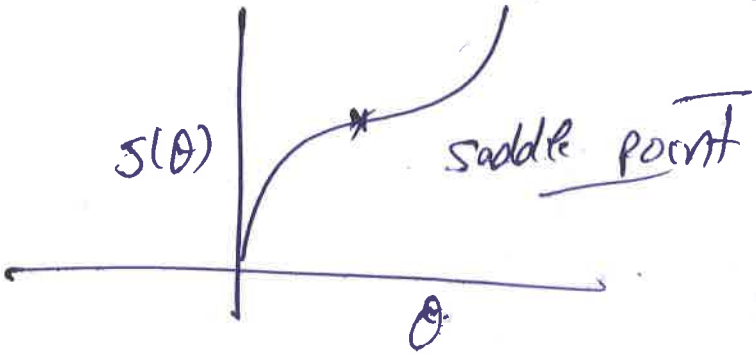
$$\theta \leftarrow \theta^{(0)} - \varepsilon g$$

$$\text{ft. } J(\theta^{(0)} - \varepsilon g) = J(\theta^{(0)}) - \varepsilon g^T g + \frac{1}{2}\varepsilon^2 g^T H g$$

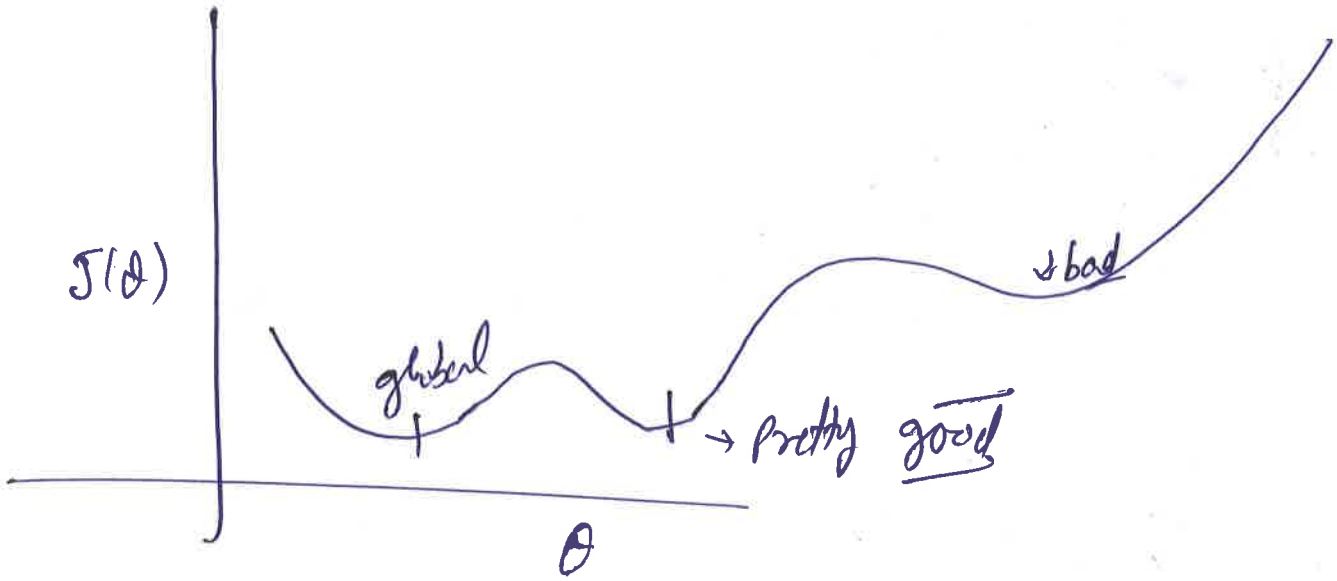
Decrease in cost :- $\boxed{\varepsilon g^T g - \frac{1}{2}\varepsilon^2 g^T H g}$

(5)





H has both +ve & -ve eigenvalues



Dropout proof:-

Does [Can work better than sampling]

Ensemble prediction:-

Average prediction $\left[\sum_{\mu} \frac{1}{2^d} p(y|x;\mu)\right]$

multiply weight by $\frac{1}{2}$ → weight scaling ⇒ Geometric Average

No non-linearity

$\left[\prod_{\mu} p(y|x;\mu)^{1/2^d}\right]$ → # of configurations

(Hinton et al., 2012)