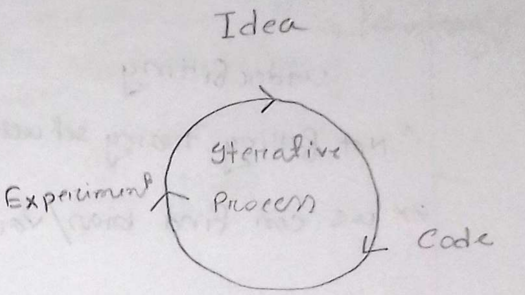
Course 2 : Improving Deep Neural Networks. Start date. 29.12.2018 10.30 pm.

week 1 : Practical Aspects of Deep Learning.

P1 : Setting up your Machine Learning Application.

L1 : Train / Dev / Test sets. [11.00 AM 30-12-2018]

For a machine learning algorithm. one have to define,



i) Number of layers.

ii) Number of hidden units.

iii) learning rates.

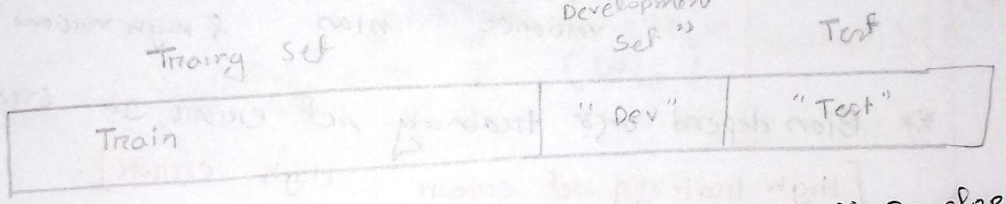
iv) Activation functions.

e.t.c and othe hyperparameters.

Iterative process

[যেকোন problem এর জন্য code এর মাধ্যমে কোন value গুলো output better হয়]

Application :- NLP, Computer vision, speech, structured Data, [Ads, search, security]



→ Training এ Parameter combination check করার জন্য Development set Evaluate করা হয়.
ব্যবহার করা হয়। আর model Test set

v.v.g [make sure that development and test set come from same distribution. (otherwise Model may fail in test set)]

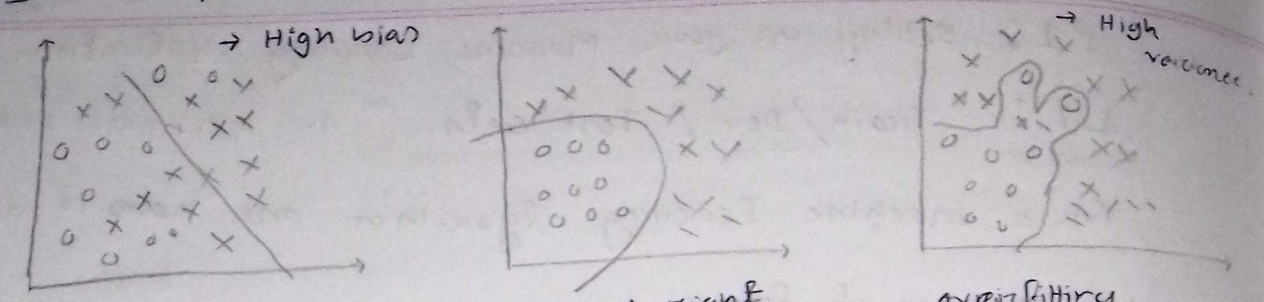
** [Test set gives us the unbiased estimate of a model]

** Not having a test set might be okay.

[Train / Dev / Test sets helps us to iterate more quickly and gives us good parameter estimation].

Scanned by CamScanner

L2: Bias/ Variance. [11:20 AM 30-12-18]



Underfitting — "Not fitting training set well"

Just right

Overfitting — "Not fitting test set well"

** we can find bias/variance by taking train/Dev set error.

Train set error :	1 %	15 %	15 %	0.5 %
Dev set error :	11 %	16 %	30 %	1 %
	High variance	High bias	High bias & high variance	Low bias & low variance.

** Bias depend করে training set error এর উপর.
[High training set error, High error].

** variance depend করে training set error এর সাপেক্ষে Dev set এ কত বেশি error হচ্ছে.

v.v.g [High difference in train and dev set error, High variance]

L3 : Basic Recipe for Machine Learning [11:40 AM 30-12-18]

practical application

High bias (training data performance) → Bigger Network. Train longer. (NN architecture, search)

↓ N

High variance (dev set performance) → More data. Regularization. (NN architecture, search)

↓ N

Done

P2: Regularizing your neural network. [Theoretical concept but important. গুরুত্বপূর্ণ]

L1 :- Regularization. [1:01 pm 30.12.18]

Logistic regression :- $w \in R^{n_x}$, $b \in R$

$$\min_{w,b} J(w,b).$$

Regularization parameter (λ)

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} ||w||_2^2$$

v.v.g.

→ L_2 regularization $||w||_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w.$ } Commonly used

L_1 regularization $\frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} ||w||_1$

Neural network :-

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} ||w^{[l]}||_F^2$$

"Frobenius norm" $||w^{[l]}||_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$ [w.shape = $(n^{[l]}, n^{[l-1]})$]

Backpropagation.

$$dw^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} w^{[l]}$$

$$w^{[l]} = w^{[l]} - \alpha \, dw^{[l]}.$$

} Code implement korte

After expanding.

$$dw^{[l]} = w^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} w^{[l]} \right]$$

} Code implement korte

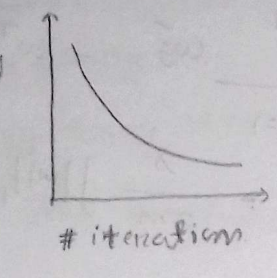
$$= w^{[l]} - \frac{\alpha \lambda}{m} w^{[l]} - \alpha (\text{from backprop}).$$

→ Here weight is decreasing in each step. That is why L2-regularization is called "weight decay"

L2: why regularization reduces overfitting ? [1.25 pm 30.12.18].

** Regularization parameter penalizes weight matrices.

→ If regularization parameter (λ) is very large then weights will be nearly zero (0) and our network become roughly linear.



[If we use regularization cost (J) will decrease monotonically]

L3: Dropout Regularization [1.35 pm 30.12.18]

[A copy of our network is made. We will set a probability threshold. Neurons under this threshold will be droped off from our network. [

Implementing dropout :- ("inverted dropout")

Illustrate with layer $l=3$, keep prob = 0.8

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keepprob.
[ want random matrix for e.g. with 80% 1's and 20% 0's ]   [ Activation function for 2nd layer ]
a3 = np.multiply(a3, d3).
a3 /= keepprob.]   inverted dropout
```

→ making predictions in test time.

No drop out.

$a^{[0]} = X$

$z^{[1]} = w^{[1]} \cdot a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

⋮

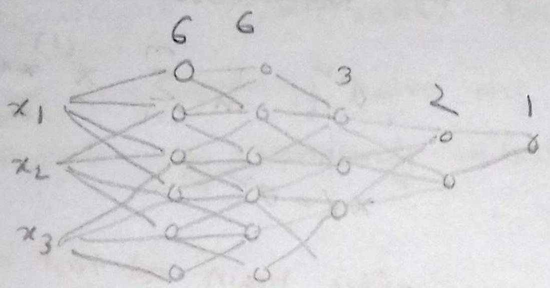
$\hat{y}$

[illegible]

L4: understanding Dropout [1.55 pm 30.12.18]

* Dropout randomly knocks out neurons of a network.

** Can not rely on any one feature, so have to spread out weight. [shrink weight] similar as L2-regularization.



** we can vary keepprobs in every layer

** Input layer এর feature Normally থাকে।

** High dropout of neuron with lower keepprob.

** Low dropout of neuron with higher keepprob.

[Mainly used in computer vision] [overfit না হলে use করার প্রয়োজন নাই।

Problem [Cost function ক্রমান্বয়ে কমে কিনা বুঝা যায় না.]

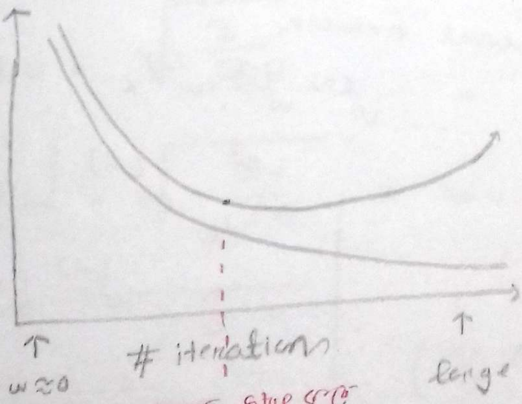
L5:- Other Regularization methods. [2.05 pm 30.12.18]

[picture flip করে training size বাড়ানো যায়]

i) Data Agumentation [বিভিন্নভাবে data change করে দেখা]

ii) Early stopping. (দুটো Problem হয়)

orthogonilization.
↳ optimization cost function.
↳ Not overfit.



** network অনেক বড় হলে দুটো technique ব্যবহার করা হয়

P3: Setting up your optimization Problem. [বিভিন্ন optimization technique নিয়ে আলোচনা করেছে]

L1: Normalizing Inputs [2.15 pm 30.12.18]

** Normalizing inputs speeds up our training Process. [put features in similar scales.]

Normalization can be done in two ways.

i) Subtract mean:

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

$$x := x - \mu$$

ii) Normalize variance.

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2$$

↑ elementwise

$$x /= \sigma^2$$

** use same μ and σ^2 to normalize train and test set.

[সবগুলো Feature কে একই range এ নিয়ে আসলে ভালো কাজ করে]

L2: Vanishing or Exploding Gradients [7.00 pm 05-01-19]

when training a very deep network your derivatives or slopes can sometimes get either very, very big or very very small this is called (vanishing or exploding gradients.)

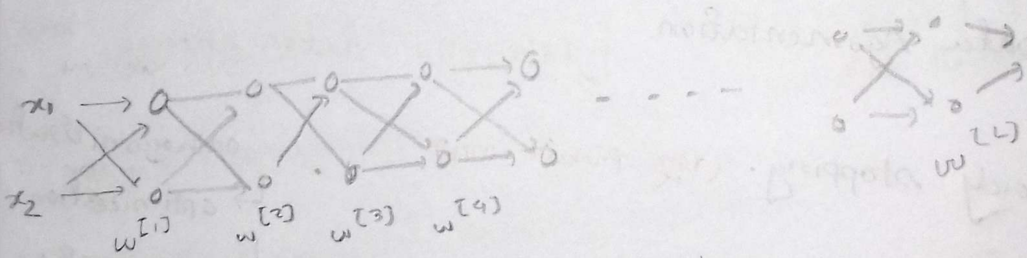


Fig:- Deep neural network

$$\hat{y} = w^{[L]} \cdot w^{[L-1]} \cdot w^{[L-2]} \cdots\cdots w^{[3]} w^{[2]} w^{[1]} x$$

$$z^{[1]} = w^{[1]} x$$

$$a^{[1]} = g(z^{[1]})$$

$$a^{[2]} = g(z^{[2]}) = g(w^{[2]} \cdot a^{[1]})$$

Let $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

$$\therefore \hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} x$$

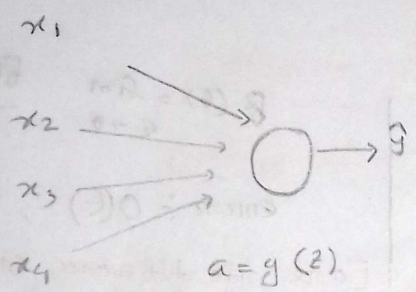
** In this case value of $\hat{y}$ will be very large and gradients will explode.

** If $w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$ then value of $\hat{y}$ will be very small or vanish.

** It makes training very difficult.

L3: weight initialization for Deep networks. [7.10 pm 05.01.19]

→ we can partially solve the vanishing / exploding gradients problem by randomly initialize the weights.



Let,

$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n$.

n = number of features.

large n → smaller w_i

$Var(w) = \frac{2}{n}$ [For ReLU it gives better result].

$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$

→ It set the weight matrix not to much greater than 1 or ~~not~~ to much smaller than 1. So weight matrix will not explode or vanish too quickly.

** Different initialization depending on variance

[Variance is another hyperparameter that can be used to tune our Algorithm]

1. ReLU = $\sqrt{\frac{2}{n^{[l-1]}}}$

2. tanh = $\sqrt{\frac{1}{n^{[l-1]}}}$ * Xavier initialization.

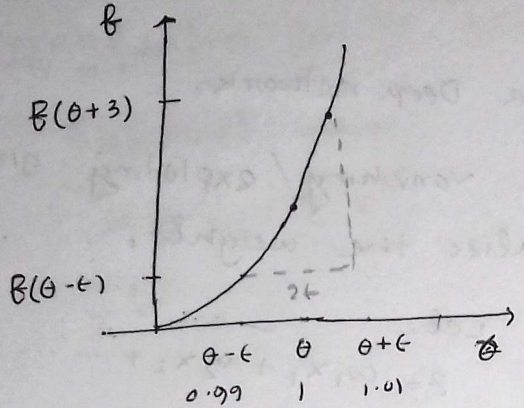
3. $\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$.

v.v.g.
** এই ধরণের অনেক Papers আছে, এগুলো নিয়ে আরো এই Papers এ reference দেয়া আছে।

* "He" initialization.

L4: Numerical approximation of gradients [7.35 pm 05-01-19]

checking your derivative computation.



$f(\theta) = \theta^3$.

$$\frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$$

→ height

← width

$$\frac{(1.01)^3 - (0.99)^3}{2.(0.01)} = 3.0001 \approx 3.$$

Theory:-

$$f'(\theta) = \lim_{\epsilon \to 0} \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{2\epsilon}$$

error = $O(\epsilon^2)$, 0.0001

[two side difference, given more accurate].

$$f(\theta) = \lim_{\epsilon \to \theta} \frac{f(\theta+\epsilon) - f(\theta-\epsilon)}{\epsilon}$$

error : $O(\epsilon)$, 0.01

[one side difference, given less accurate.

** L5: Gradient checking [7.45 pm 05.01.19]

v.v.g

[It is useful to check whether our backpropagation implementation is correct or not].

Gradient check for a neural network,

1. Take $W^{[1]}, b^{[1]}, \ldots W^{[L]}, b^{[L]}$ and reshape into a big vector θ. $J(W^{[1]}, b^{[1]}, \ldots W^{[L]}, b^{[L]}) = J(\theta)$.

2. Take $dW^{[1]}, db^{[1]}, \ldots dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

** $d\theta$ is gradient of cost function (J)

Gradient checking. (Grad check).

$J(\theta) = J(\theta_1, \theta_2, \cdots\cdots)$.

for each i:

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \cdots \theta_i + \epsilon, \cdots) - J(\theta_1, \theta_2, \cdots \theta_i - \epsilon, \cdots)}{2\epsilon}$$

[Taking two sided difference]

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

→ যদি approximately same না হয় তাহলে error আছে

→ Now check $\boxed{d\theta_{approx} \overset{?}{\approx} d\theta}$.

$\epsilon = 10^{-7}$.

Check :-
$$\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2}$$

[If value of check is less than 10^{-7} then it is good otherwise check]

[8:00 pm 05.01.19].

L6: Gradient checking implementation notes.

- Don't use grad check in training - only to debug. [takes more time].
- If algorithm fails grad check, look at components to try to identify bug.

→ Remember regularization in grad check.

→ Grad check doesn't work with dropout.

→ Run at random initialization; perhaps again after some training.

Programming Assignment

→ 1. বিভিন্ন ধরনের initialization নিয়ে আলোচনা করা হয়েছে.

Neural network zero initialization করলে লাভ হবে না.

[Zero initialization, Random initialization, "He" initialization].

→ ** Different initialization leads to different result.

He initialization works well for networks with ReLU activations

নিচের ভালো ভাবে দেখো

v.v.g [Decision Boundary plot করার track রাখা ভালো কেননা সহজে দেখা যায়

2. Regularization helps to improve algorithm accuracy.

3. Gradient checking ensures that backpropagation is actually working.

[1 no course এ যে Backpropagation টা শিখেছি তার ঠিক আছে)

Gradient checking is costly we will not use it in every iteration during training.

week-2 optimization algorithm.

L-1 Mini-batch gradient descent [4.15 pm 08.01.19]

Batch Vs. mini-batch gradient descent.

→ Faster computation of: First we split whole training set into definite number of partitions. These partitions is called mini-batches.

$$X = \left[x^{(1)} \; x^{(2)} \; \cdots \; x^{(1000)} \mid x^{(1001)} \cdots x^{(2000)} \mid \cdots \mid \cdots \; x^{(m)} \right].$$

(n_x, m) $\quad x^{\{1\}}$ — mini batch number $\quad x^{\{2\}} \quad x^{\{5000\}}$.

$$Y = \left[y^{(1)} \; y^{(2)} \; \cdots \; y^{(1000)} \mid y^{(1001)} \cdots y^{(2000)} \mid \cdots \mid \cdots \; y^{(m)} \right]$$

$Y^{\{1\}} \quad Y^{\{2\}} \quad Y^{\{5000\}}$

** number of mini-batches → $\frac{\text{\# number of traing example}}{\text{\# number of example in a mini-batch}}$

Mini-batch gradient descent. (implementation)

for $t = 1, \cdots, 5000$ — number of mini-batches. (1 step of gradient descent using $x^{\{t\}}, Y^{\{t\}}$)

{ forward of on $X^{\{t\}}$

$$z^{[1]} = w^{[1]} x^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{(1)}(z^{[1]})$$
$$A^{[l]} = g^{(l)}(z^{[l]})$$

} we can vectorized it (1000 example).

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|w^{[l]}\|_F^2$ (Number of example in a mini-batch.)

Backprop to compute gradients of $J^{\{t\}}$ (using $x^{\{t\}}, Y^{\{t\}}$)

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}, \quad b^{[l]} = b^{[l]} - \alpha db^{[l]}.$$

}

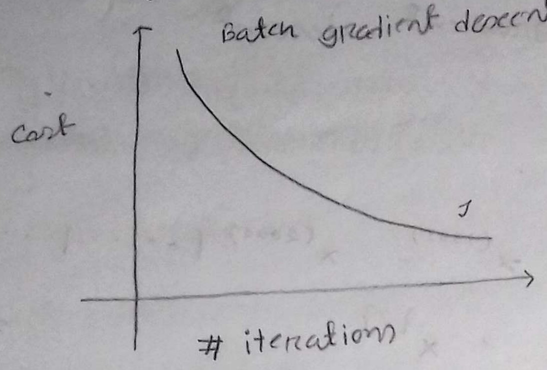
1-epoch

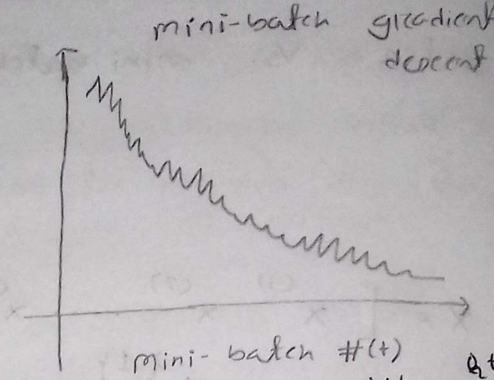
mini-batch

** For large traing set n runs, much faster than batch gradient descent.

L-2 understanding mini-batch gradient descent. [4.35 pm 08.01.19]

Training with mini-batch gradient descent.





plot $J^{\{t\}}$ using $X^{\{t\}}, Y^{\{t\}}$.

choosing your mini-batch size. (one of hyperparameter)

If mini-batch size = m : Batch gradient descent $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

If mini-batch size = 1 : Stochastic gradient descent | Every example is its own $(x^{(1)}, y^{(1)})$ mini-batch.

$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$

v.v.g.

⇒ In practice we chose mini-batch size between (1 to m)

Stochastic gradient descent (mini batch size = 1)	(good) mini-batch size not too big not too small	Batch gradient descent (mini batch size = m)
↓	↓	↓
lose speed up from vectorization	Fastest learning. • vectorization • make program without process entire set.	Too long for each iteration. (take long time)

1. If small training set : use batch gradient descent ($m \leq 2000$)
2. Typical minibatch size : 64, 128, 256, 512 ← "minibatch size 2 এর power এতে computationally advantage পাওয়া যায়"
3. make sure that minibatch $(X^{\{t\}}, Y^{\{t\}})$ fits in your CPU/GPU memory.

L3: Exponentially weighted averages. [4.57 pm 08.01.19]

Temparature in London.

$\theta_1 = 40^\circ F$

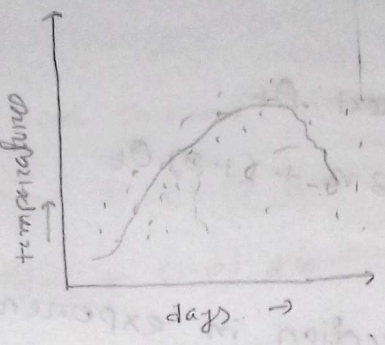
$\theta_2 = 49^\circ F$

$\theta_3 = 45^\circ F$.

$\vdots$

$\theta_{150} = 60^\circ F$.

$\vdots$



$v_0 = 0$.

$v_1 = 0.9\, v_0 + 0.1\, \theta_1$

$v_2 = 0.9\, v_1 + 0.1\, \theta_2$

$\vdots$

$v_t = 0.9\, v_{t-1} + 0.1\, \theta_t$.

Exponentially weighted averages.

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

(β → hyperparameter.)

[formula for computing exponentially weighted moving averages]

[v_t an approximately average over $\simeq \frac{1}{1-\beta}$ day temperature].

$\beta = 0.9$: 10 day temp

$\beta = 0.98$: 50 days temp.

$\beta = 0.5$: 2 day temp.

{ Higher values of β in averages over larger values of days. }

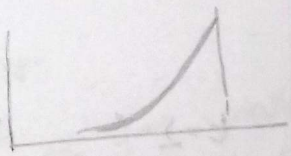
L4: Understanding exponentially weighted averages. [5.10 pm 08.01.19]

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$$\begin{cases} v_{100} = 0.9\, v_{99} + 0.1\, \theta_{100} \\ v_{99} = 0.9\, v_{98} + 0.1\, \theta_{99} \\ v_{98} = 0.9\, v_{97} + 0.1\, \theta_{98}. \end{cases}$$

$\rightarrow$ $v_{100} = 0.1\,\theta_{100} + 0.9\, v_{99}$.

(θ_{100} ↓ current value. v_{99} ↓ previous value)



→ Here we have a weight decay function.

→ Before weight decays to zero it has effect on our weighted average equation.

→ we can find how many days we are averaging by using $\left(\frac{1}{1-\beta}\right)$ equation.

Implementation.

$v_\theta = 0$

Repeat {

get next θ_t

$v_\theta := \beta v_\theta + (1-\beta)\theta_t$

}

[It takes really small amount of memory]

L-5 Bias Correction in exponentiall wei. Av. [5.25 pm 08.01.2019]

(graph নিয়ে আলোচনা করা হয়েছে)

slide 9.

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t.$$

$v_0 = 0$

$v_1 = 0.98 v_0 + 0.02\theta_1$

- If we take $v_0 = 0$ then $v_1 = 0.02\theta_1$. So we don't correctly estimates v_1.

→ we can correct this bias by using a simple equation,

$$\frac{v_t}{1-\beta^t}$$

$t=2$: $1-\beta^t = 1-(0.98)^2 = 0.0396.$

$$\frac{v_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396.}$$

→ when value of t is high ~~we can~~ the $\beta^t \approx 0$ Hence we correctly remove biases.

[It is used to build some other better optimization algorithm].

L-6 Gradient descent with momentum [1.12 Am 09.01.19]

[Faster than traditional gradient descent] (slide ও graph নিয়ে first আইডিয়া দেবে)

Momentum :-

on iteration t :

Compute dw, db on current mini-batch.

$$v_{dw} = \beta v_{dw} + (1-\beta)\, dw.$$

$$v_{db} = \beta v_{db} + (1-\beta)\, db.$$

$$w = w - \alpha v_{dw}, \quad b = b - \alpha v_{db}.$$

* It smoths the steps of gradient descent. (faster).

Here: Hyperparameter : α, β — এই দুইটা parameter change করলে অনেক বেশি different output পাব.

L-7 RMS prop. [1.25 Am 09.01.19].

[Root mean square prop speeds up gradient descent].

on iteration t :

Compute dw, db on current mini batch.

$$S_{dw} = \beta_2 v_{dw} + (1-\beta_2)\, dw^2 \quad \text{(element wise)}$$

small

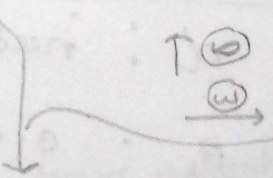
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2$$

large.

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}+\varepsilon}}, \quad b := b - \alpha \frac{db}{\sqrt{S_{db}+\varepsilon}}$$

→ updates in vertical direction divided by a large number. so it slows down oscillations.

updates in horizontal direction divided by a small number so it help to converge faster.



Ensures that it never becomes divided by zero.

L-8 Adam optimization algorithm [1.40 Am 09.01.19]

[Combination of momentum and RMS prop and a generalize optimization algorithm]

Adam optimization algorithm.

~~Comp on iteration t:~~

$V_{dw} = 0$, $S_{dw} = 0$, $V_{db} = 0$, $S_{db} = 0$

on iteration t:

Compute dw, db, using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dw$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db$$

} momentum part. "β_1"

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dw^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2$$

} RMS prop part "β_2"

$$V_{dw}^{\text{Corrected}} = V_{dw}/(1-\beta_1^t), \quad V_{db}^{\text{Corrected}} = V_{db}/(1-\beta_1^t)$$
$$S_{dw}^{\text{Corrected}} = S_{dw}/(1-\beta_2^t), \quad S_{db}^{\text{Corrected}} = S_{db}/(1-\beta_2^t)$$

} Bias Correction part.

$$w = w - \alpha \cdot \frac{V_{dw}^{\text{Corrected}}}{\sqrt{S_{dw}^{\text{Corrected}} + \epsilon}}, \quad b = b - \alpha \cdot \frac{V_{db}^{\text{Corrected}}}{\sqrt{S_{db}^{\text{Corrected}} + \epsilon}}$$

} update of gradient descent

v.v.g. → **Hyperparameter choice :-**

α : needs to be tuned.

β_1 : 0.9 (common choice) (dw) [momentum]

β_2 : 0.999 (Adam's paper) (dw^2) [RMS prop].

ϵ : 10^{-8} (Adam's paper).

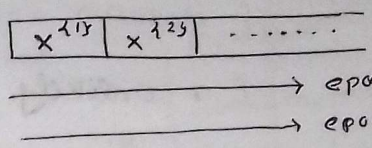
Adam : Adaptive moment estimation

L-9 Learning rate decay [1.00 PM 09.01.19]

[slowly reduce learning rate α]

Learning rate decay.

1 epoch = 1 pass through data



$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

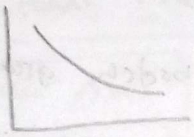
α_0 → learning rate

decay-rate → Hyperparameter we want to tune.

Epoch	α
1	0.1
2	0.67
3	0.5
⋮	⋮

$\alpha_0 = 0.2$

decay rate = 1

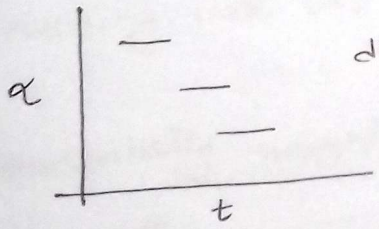


Other learning rate decay methods.

$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ → exponentially decay.

$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

t → minibatch number.



discrete staircase

manual decay.

[Learning rate decay helps to train neural network faster].

L-10 The problem of local optima [1.10 pm 09.01.19]

[Saddle point

[plateaus are the points where derivatives are zero for a long time]

[• unlikely to get stuck in bad local optima.

• plateaus can make learning slow

Programming Assignment :-

1. with a well-tuned mini-batch size, usually outperforms either gradient descent or stochastic gradient descent (when training set is high).

2. learn to implement mini-batch gradient descent.

3. Adam is the most effective optimization algorithm for training a neural network. It ~~describe~~ combines idea of RMSprop and momentum.

[Adam outperforms mini-batch gradient descent and momentum].

[Random algorithm is best]

week 3 : Hyperparameter tuning, Batch normalization and Programming frameworks.

P1 : Hyperparameter tuning.

L1 : Tuning Process [2.20 pm 10.01.19]

v.v.g.

Hyperparameters for a neural network.

→ 1 Learning rate (α) → most important

→ 2 Momentum (β) 0.9

→ Adam optimization parameters (β_1, β_2, ϵ) 0.999, 10^{-8}

→ 3 number of layers.

→ 2 Hidden units.

→ 3 Learning rate decay.

→ 2 mini-batch size.

Theory type assignment, implementation ও লাগবে না।

→ In hyperparameter tuning of deep learning we use random values.

L2 : using an appropriate scale to pick hyperparameters [2.30 pm 10.01.19]

1. number of hidden unit ও number of layers এর জন্য linear search ঠিক আছে।

2. learning rate (α) এর জন্য random search use করা যায়।

3. Exponentially weighted average এর জন্য random search করার দরকার নাই।

L3: Hyperparameters tuning in practice: [2:42 pm 10.01.19]

[Computational model ~~is~~ depend on (any) theoretical area]

→ Babysitting one model.

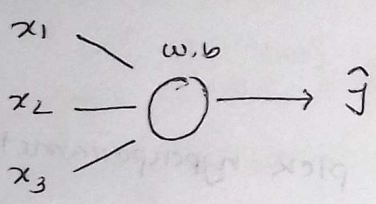
→ Training many models in parallel.

P2: Batch Normalization.

L1: Normalizing activations in a network [2:50pm 10.01.19]

Nor [Batch normalization makes hyperparameter search problem much easier, and makes neural network much robust]

Normalizing inputs to speed up learning.



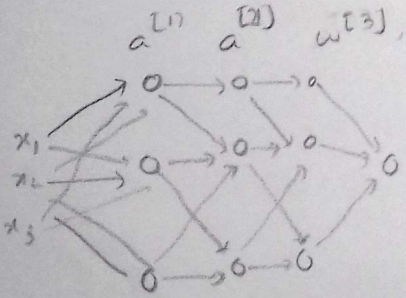
$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$x = x - \mu$$

$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)2}$$

$$x = x/\sigma^2$$

Here the question is for any hidden layer can we normalize the activations/inputs to speed up computation of next layer.



→ Here is a debate whether we normalize activation $a^{[l]}$ or $z^{[l]}$

$$[a^{[l]} = g^l(z^{[l]})$$

→ Normally we normalize $z^{[l]}$ as a default.

Implementing Batch Norm.

Given some intermediate values of a neural network $[z^{(1)} \dots z^{(m)}]$.

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z_i - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$ — used for numerical stability if σ^2 becomes zero.

z tilde

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$ (gamma)

Learnable parameters of a model. It allows us to set the mean of $\tilde{z}$ (z tilde) to be whatever you want.

→ use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$ for the computation of later layers in the neural network.

viv.y

** Batch norm not only normalize the input layer but also the values of hidden layer in an NN.

- By varying γ (gamma) and β (beta) we can control mean and variance. For example. if,

$$\gamma = \sqrt{\sigma^2 + \epsilon}$$

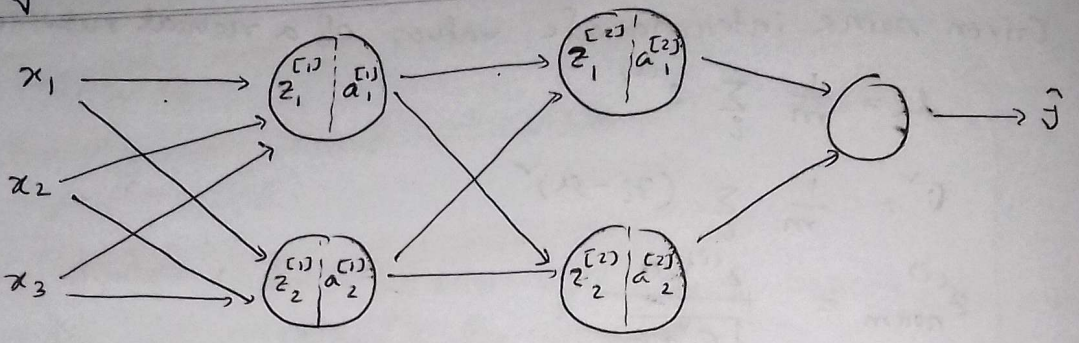
and $\beta = \mu$.

then $\tilde{z}^{(i)} = z^{(i)}$

Now our hidden units have standardize mean and variance controlled by two explicit parameters (gamma and beta).

L2: Fitting Batch Norm into a NN [3.55 pm 10.01.19]

Adding Batch Norm to a network.



Steps of NN.

$$X \xrightarrow[b^{[1]}]{w^{[1]}} z^{[1]} \xrightarrow[BN]{\beta^{[1]}, g^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow[b^{[2]}]{w^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, g^{[2]}} \tilde{z}^{[2]} = a^{[2]}$$

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \ldots w^{[L]}, b^{[L]}$

$\beta^{[1]}, g^{[1]}, \beta^{[2]}, g^{[2]}, \ldots \beta^{[L]}, g^{[L]}$

$d\beta^{[l]}$

$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$

→ we can implement batch Norm by only one line of code

working with mini-batches.

$X^{\{1\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow{\beta^{[1]}, g^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \rightarrow$

$X^{\{2\}} \rightarrow z^{[1]} \rightarrow \tilde{z}^{[1]} \ldots$

$X^{\{3\}} \ldots$

$\vdots$

On batch norm parameters $b^{[l]}$ is cancelled out in mean subtraction step. So,

Parameters: $w^{[l]}, \cancel{b^{[l]}}, \beta^{[l]}, g^{[l]}$

$(n^{[l]}, 1)\ (n^{[l]}, 1)$

$z^{[l]} = w^{[l]} a^{[l-1]} + \cancel{b^{[l]}}$

$z^{[l]} = w^{[l]} a^{[l-1]}$

$z^{[l]}_{norm} =$

$\tilde{z}^{[l]} = g^{[l]} z_{norm} + \beta^{[l]}$

Implementing gradient descent using Batch-Norm.

for $t = 1$ ---- number of mini-batches.

→ Compute forward prop on $X^{\{t\}}$.

→ In each hidden layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$.

→ Use backprop to compute $dw^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$.

→ Update parameters.

$$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$$
$$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$
$$\gamma^{[l]} = \gamma^{[l]} - \alpha d\gamma^{[l]}$$

— update.

[we can also use Momentum, RMSprop, Adam optimization]

→ আমরা Project এ আগে এগুলো implement করব.

L-3 Why does Batch Norm work? [4:20 pm 10.01.19]

[কেন কাজ করে এটা নিয়ে]

1. Normalizing features to mean zero and variance one speed up learning. Because features are ranging from zero to one rather than ranging form different ranges.

2. Batch-norm reduce the problem of input value-changing

3. It allows each layer learn independently a little bit then other layers.

4. It makes the job of learning in the later layers much easier.

Batch Norm as regularization.

1. Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
2. This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.
3. This has a slight regularization effect.

L-4 Batch norm at test time [8.20 pm 10.01.19]

Equation of Batch Norm at train time.

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

During testing we may not have enough example to form a mini-batch or may have only one example.

In that case:

μ, σ^2: estimate using exponentially weighted average

(This keep track of latest weights average of μ, σ^2)

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}} \ldots$

$\mu^{\{1\}[l]}, \mu^{\{2\}[l]}, \mu^{\{3\}[l]} \ldots \rightarrow \mu$

$\sigma^{2\{1\}[l]}, \sigma^{2\{2\}[l]}, \sigma^{2\{3\}[l]} \ldots \rightarrow \sigma^2$

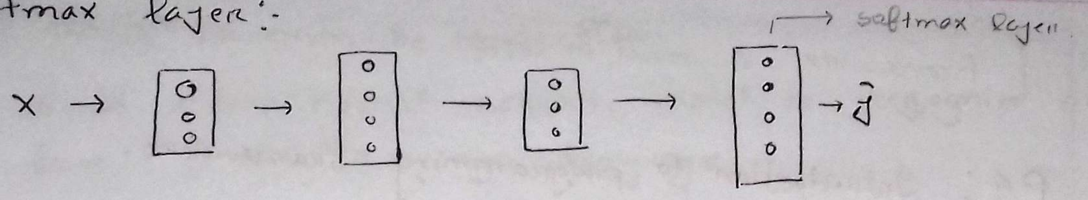
Now apply exponentially weighted average and use it in test time.

$$\left[z_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad \tilde{z} = \gamma z_{norm} + \beta \right].$$

P3 : multiclass classification

L1 :- Softmax Regression . [10.45 am 10.01.19]

Softmax layer :-



→ Final output layer of a multiclass classifier is called softmax layer.

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

Activation function for softmax layer:

$$t = e^{(z^{[l]})}$$ → ensures that probability lies between 0 to 1.

$$a^{[l]} = \frac{e^{(z^{[l]})}}{\sum_{j=1}^{n^{[l]}} t} \quad \text{or} \quad \frac{t_i}{\sum_{i=1} t_i}$$

[softmax এর জন্য example নিচে আলোচনা করা হয়েছে].

L2 : Training a softmax classifier. [11.00 am 10.01.09]

["soft max" comes from "hard max"]

Softmax regression generalizes logistic regression to C classes.

→ If C=2 softmax reduces to logistic regression.

Loss function:

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad a^{[l]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j \log \hat{y}_j$$ → Loss on one training example

$$J(w^{[1]}, b^{[1]} \ldots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$ → Loss in entire training set

input :- (4 class, m examples)

$$Y = \begin{bmatrix} 0 & 1 & 0 & \cdots & m \\ 0 & 0 & 1 & \cdots & m \\ 0 & 0 & 0 & \cdots & m \\ 1 & 1 & 0 & \cdots & m \end{bmatrix}$$

output

$$\hat{Y} = \begin{bmatrix} .33 & \cdots & m \\ .22 & \cdots & m \\ .21 & \cdots & m \\ .24 & \cdots & m \end{bmatrix}$$

Gradient descent with softmax. [slide এ আছে ভালো করে বুঝ

[Framework এ built in থাকে so এগুলো ইমপ্লিমেন্ট হয় না

P4: Introduction to programming frameworks.

L1: Deep learning frameworks [11.15 AM 10.01.19]

Choosing deep learning frameworks.

- Ease of programming (development and deployment)
- Running speed.
- Truly open (open source with good governance).

[some frameworks]

L2: Tensorflow [11.30 am 10.01.19]

[Tensorflow knows automatically how to implement backprop]

** we have to define a cost function and tensorflow automatically find the derivatives and a way to minimize cost function.

Running and writing program in tensorflow has following steps.

i) Create tensors (variables) that are not yet executed.

ii) write operations between those tensors.

iii) Initialize your tensors.

iv) Create a session.

(v) Run the session.

Programming Assignment.

→ Learn basic about tensorflow.

→ Build a deep neural network model to recognize number from 0 to 5 in sign language.

will learn
i) initialize variables.
ii) start your own session.
iii) Train algorithm.
iv) Implement a neural network.

** remember to initialize your variables, create a session and run the ~~session~~ operations inside the session.

* a placeholder is an object where value can be specified in later.

— one hot vectors. (one hot encoding)

— backpropagation automatically ~~framework~~ does it?