An overview on ML/DL model development

AICTE Training and Learning (ATAL) Academy Sponsored FDP on **DL for Audio and Speech Processing**

Dr. Sishir Kalita
Data Scientist
Armsoftech.air
Chennai

Outline

1. Building a DL model: steps
2. Bias/variance problem
3. Regularizations and batch norm
4. ML Strategies
5. Evaluation metrics

Different Speech Tech projects

1. Voice bot (Speech enhancement, ASR, TTS, Speech analytics, SLU)
2. Speaker recognition
3. Spoken Language Idenfication
4. Speech to speech translation

Key steps in building an ML/DL model

1. Problem definition
2. Gathering data
3. Data pre-processing
4. Define the evaluation metric
5. Model training / testing
   - Iterate till you get the good results in the test set
   - Pump more labeled data
6. Model deployment

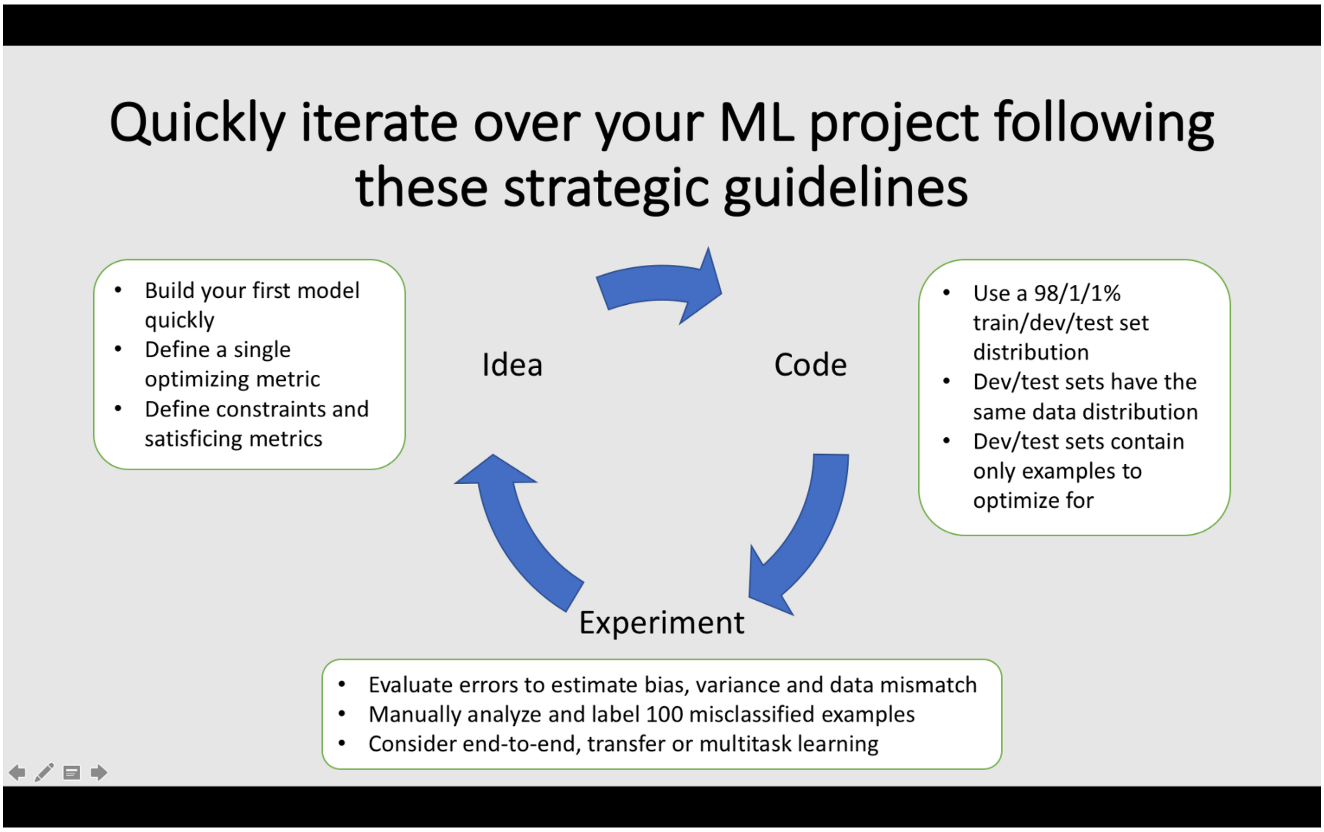


Image source: [Coursera | Deep Learning Specialization (https://www.coursera.org/learn/machine-learning-projects)](https://www.coursera.org/learn/machine-learning-projects)

Where will I get the speech data?

- Your own customized data
- [Google dataset search (https://datasetsearch.research.google.com/)](https://datasetsearch.research.google.com/)
- [Kaggle (https://www.kaggle.com/datasets)](https://www.kaggle.com/datasets)
- [Commonvoice Mozilla (https://commonvoice.mozilla.org/en/datasets)](https://commonvoice.mozilla.org/en/datasets)
- [Openslr (https://openslr.org/resources.php)](https://openslr.org/resources.php)
- [National Platform for Language Technology (https://nplt.in/demo/resources/speech-corpus)](https://nplt.in/demo/resources/speech-corpus)
- [Linguistic Data Consortium (https://catalog.ldc.upenn.ed)](https://catalog.ldc.upenn.ed)

Prepare your train, development (dev) / validation (val) and test (evaluation (eval)) sets

1. How to divide:
   - Before deep learning era: 80% (train: 8000), 10% (Dev: 1000), and 10% (Test: 1000) (**if you have 10,000 examples**)
   - Present days : 98% (train), 1% (Dev), and 1% (Test) (**if you have 10,00,0000 examples***)
2. Should have a more diverse train set
3. Distribution of dev and test sets should be same
4. Cleaning up mislabeled dev and test sets examples

Steps to train a model

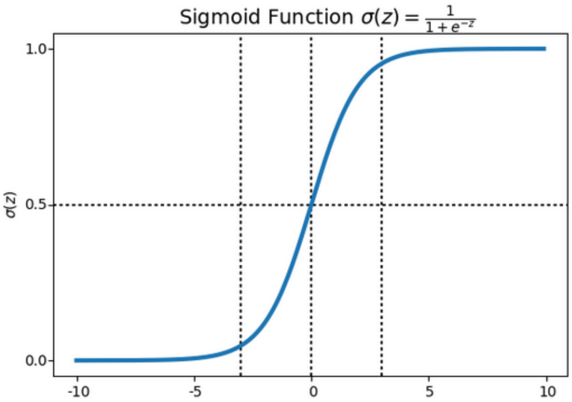
1. Decide the model and define its structure
2. Normalizing the inputs
3. Initialize the parameters of the model
4. Learn the parameters for the model by minimizing the cost
    - Calculate current loss (forward propagation)
    - Calculate current gradient (backward propagation)
    - Update parameters (gradient descent)
5. Use the learned parameters to make predictions (on the test set)

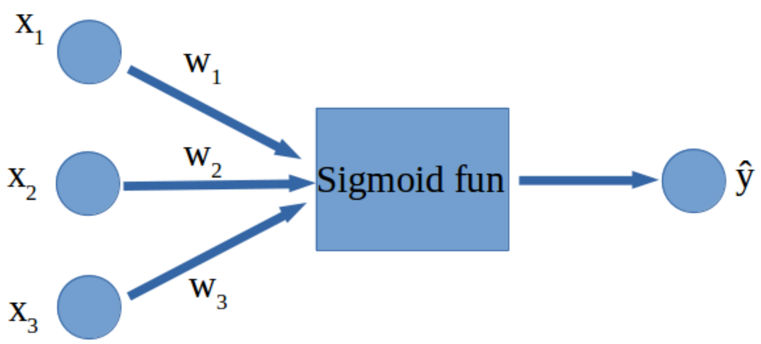
Let's consider a logistic regression model

- Binary classification model
- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots (x^{(m)}, y^{(m)})\}$

$$\hat{y} = h_{\mathrm{w}}(x) = \frac{1}{1 + e^{-\mathrm{w}^T x}}$$

$$\mathrm{w}^T x = [\mathrm{w}_1, \mathrm{w}_2, \mathrm{w}_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$





Compute the loss function: ('m' is the number of training example)

- **Binary cross-entropy loss**

$$J(\mathrm{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{y}^{(i)}) + (1 - y^{(i)})log(1 - \hat{y}^{(i)})]$$

  If y = 1 (first part) and y = 0 (second part)

Now, we need to find the **w** and **b** which minimize the $J(\mathrm{w}, b)$ [Requires one optimization algorithm]

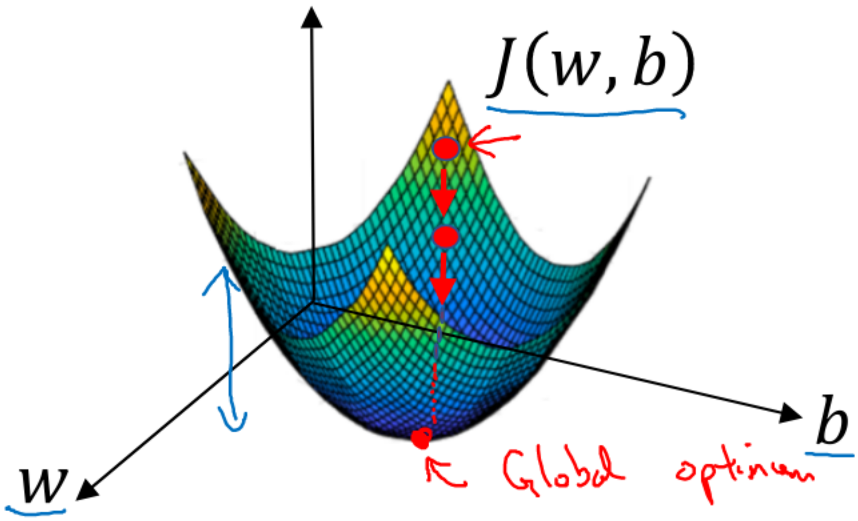


Image source: [cs230.stanford.edu (https://cs230.stanford.edu/files/C1M2.pdf)](https://cs230.stanford.edu/files/C1M2.pdf)

Optimization algorithms (I hope it is already discussed)

Gradient descent

1. Compute the gradient w.r.t **w** and **b**
2. Update the **w** and **b**

$$w_{j+1} = w_j - \alpha \frac{dJ(w,b)}{dw}$$
$$b_{j+1} = b_j - \alpha \frac{dJ(w,b)}{db}$$

3. Iterate till you reach local minima

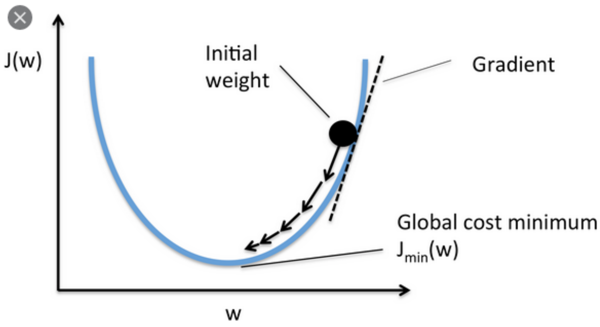
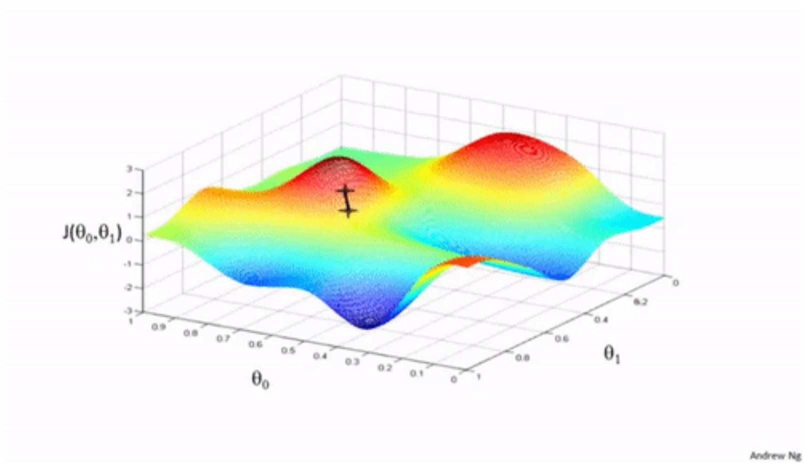


Image source: [rasbt.github.io (http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)]



Other loss functions

- **Focal loss:** It reshapes the cross entropy loss in such a way that it down weighs the loss assigned to well classified examples
- **Negative log likelihood loss (NLLL):** takes class weights as input
- **Constrastive loss**
- **Connectionist Temporal Classification Loss (CTC Loss):** where we need alignment between sequences

Focal Loss (https://medium.com/adventures-with-deep-learning/focal-loss-demystified-c529277052de)
CTC Loss (https://distill.pub/2017/ctc/)

Stochastic gradient descent vs **mini batch gradient descent** vs **Batch gradient descent**

1. Let's say, your train set size 'm'
2. Stochastic gradient descent: calculate error for each example and update the model for each example
3. Mini batch gradient descent: take a mini batch ($M < m$), compute the error, and update the model for each mini-batch
4. Batch gradient descent: calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated

Other advanced optimization algorithms

1. Gradient descent with momentum
2. Adam
3. RMSprop

What is epoch?

1. One cycle through the entire training dataset is called a training epoch
2. Number of passes (1 pass : one forward pass + one backward pass in one batch)
3. Let $m = 1000$, and $M = 10$
4. For SGD: there will be 1000 iterations/epoch
5. For Mini batch gradient descent: there will be 1000/10 (100) iterations/epoch
6. For Batch gradient descent: there will be 1 iteration/epoch

```python
for epoch in range(n_epochs):
    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(device)
        y_batch = y_batch.to(device)

        # Sets model to TRAIN mode
        model.train()
        #################
        # Forward pass
        # Makes predictions
        yhat = model(x_batch)
        # Computes loss
        loss = loss_fn(y_batch, yhat)

        #################
        # Backward pass
        # Computes gradients
        loss.backward()

        #################
        # Updates parameters
        optimizer.step()

    with torch.no_grad():
        for x_eval, y_eval in eval_loader:
            x_eval = x_eval.to(device)
            y_eval = y_eval.to(device)

            model.eval()

            yhat = model(x_eval)
            eval_loss = loss_fn(y_eval, yhat)
            eval_losses.append(eval_loss.item())
```

Normalization of the inputs

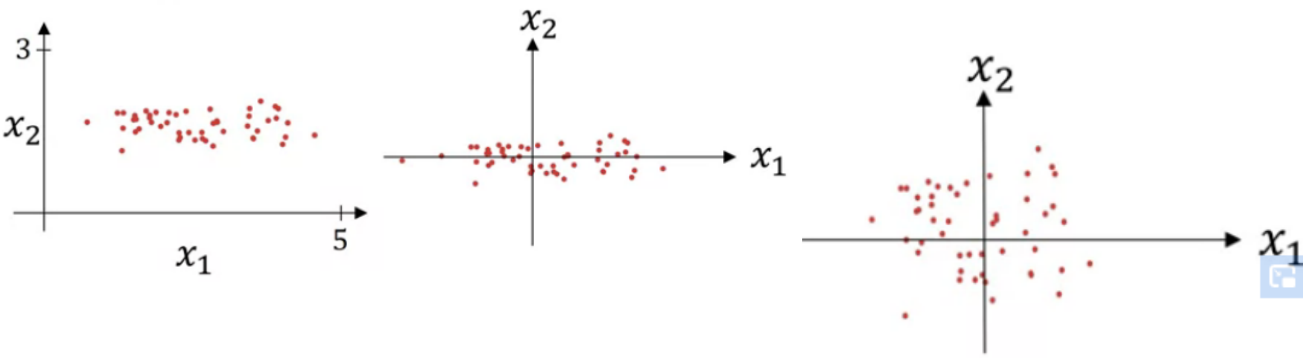


Image source: [Andrew Ng (deeplearning.ai)]()

For train set:

$$X_{norm}^{train} = \frac{X^{train} - \mu^{train}}{\sigma^{train}}$$

For test set:

$$X_{norm}^{test} = \frac{X^{test} - \mu^{train}}{\sigma^{train}}$$

Why we should normalize the input

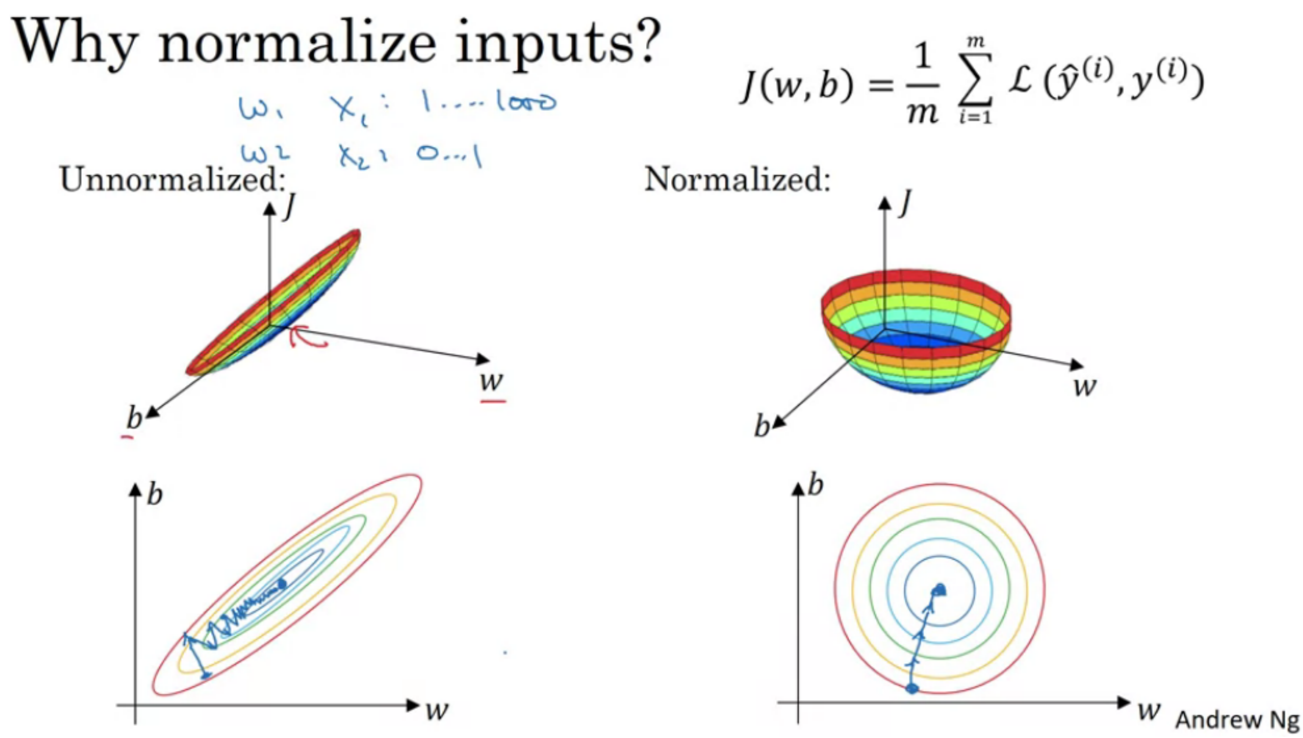


Image source: [Andrew Ng (deeplearning.ai)]()

Bias/Variance problem

1. We can get some idea for improving the model by knowing the bias/variance problem

	Train set error (%)	Test set error (%)	Conclusion
High bias problem	25	40	Underfitting
High variance problem	1	12	Overfitting
Good model	1	2	

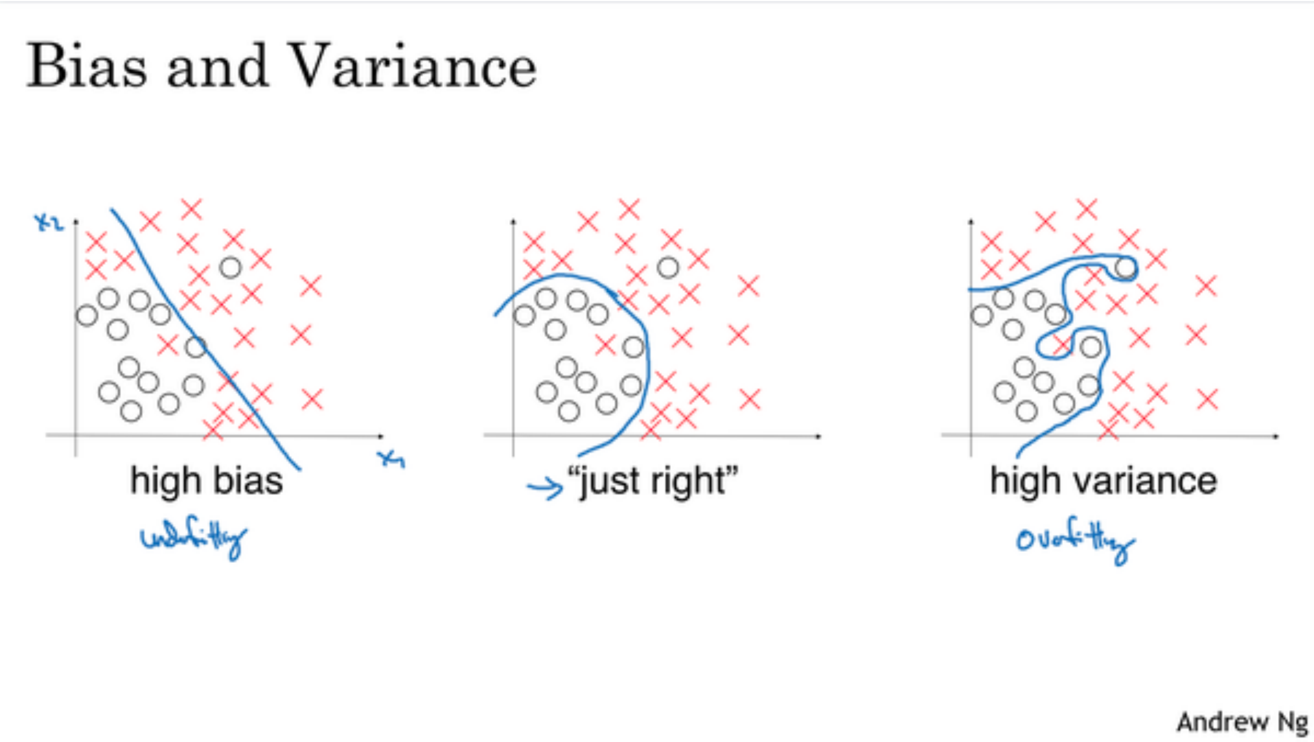
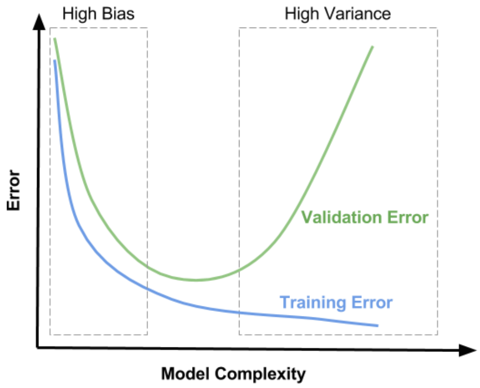


Image source: [deeplearning.ai (deeplearning.ai)](deeplearning.ai)

Addressing bias/variance problem

1. **Reducing the high variance**
    - Add more training data
    - Add regularization
        - L2 regularization
        - Dropout
    - Add early stopping
    - Decrease the model size
2. **Reducing the high bias**
    - Increase the model size (# of neurons/layers)
    - Try to run it longer
    - Different (advanced) optimization algorithms
    - Reduce or eliminate regularization
    - Modify model architecture

Try until you get better results on both train and test sets



Iamge source: [dziganto.github.io (https://dziganto.github.io/cross-validation/data%20science/machine%20learning/model%20tuning/python/Model-Tuning-with-Validation-and-Cross-Validation/)](https://dziganto.github.io/cross-validation/data%20science/machine%20learning/model%20tuning/python/Model-Tuning-with-Validation-and-Cross-Validation/)

L2 Regularization

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

1. Here, λ is the regularization parameter (hyperparameter)
2. Penalizes large weights and effectively limits the freedom the model
3. Causes the weight to decay in proportion to its size
4. If lambda is too large - a lot of **w**'s will be close to zeros which will make the NN simpler (you can think of it as it would behave closer to logistic regression).

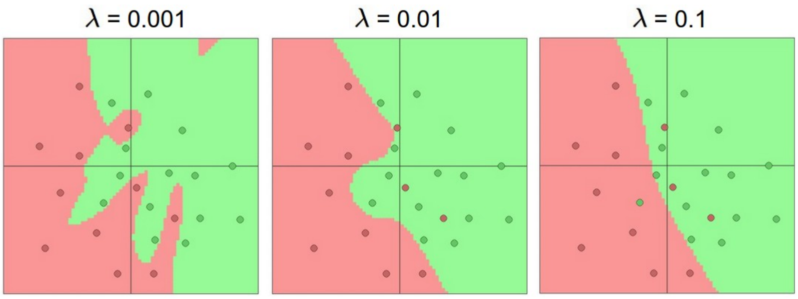


Image source: [cs231n.github.io (https://cs231n.github.io/neural-networks-1/)](https://cs231n.github.io/neural-networks-1/)

Dropout

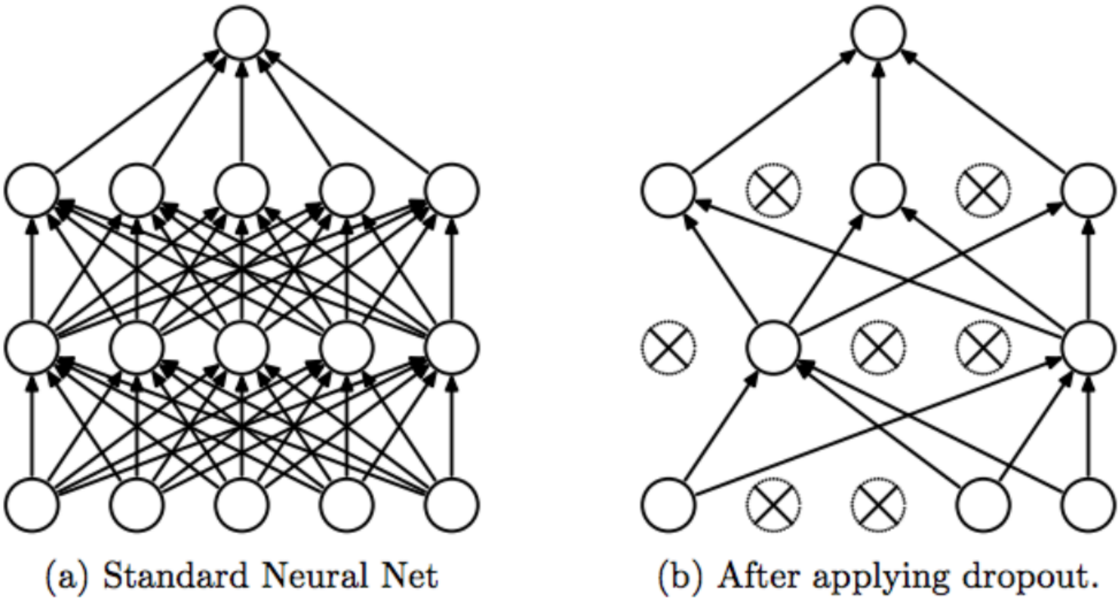


Image source: [cs231n.github.io (https://cs231n.github.io/neural-networks-2/)](https://cs231n.github.io/neural-networks-2/)

1. The dropout regularization eliminates some neurons/weights on each iteration based on a probability
2. Can’t rely on any one feature, so have to spread out weights [Andrew Ng]

Demo: [L2 & Dropout (https://qmsvpvzwwppdeofafmdkgv.coursera-apps.org/notebooks/week5/Regularization/Regularization_v2a.ipynb)](https://qmsvpvzwwppdeofafmdkgv.coursera-apps.org/notebooks/week5/Regularization/Regularization_v2a.ipynb)

Other regularization methods

1. Data augmentation
    - If data mismatch between train and test set
    - Add noise in the speech signal or perturb the speech signal
    - Use speech synthesis
    - Distorts the image (scaled, rotate)
    - Create image using graphics
2. Early stopping
    - Check the train and validation set errors

Parameters and Hyperparameters

1. Weights (w) or bias (b) is a learnable parameter
2. Hyper parameters (parameters that control the algorithm)
    - Learning rate
    - Number of iteration
    - Number of hidden layers
    - Number of hidden units
    - Choice of activation functions
    - Mini-batch size

Normalizing activations in a network

Batch normalization (BN)

1. BN allows each layer of a network to learn by itself a little bit more independently of other layers
2. Reduces the problem of input values changing (shifting)

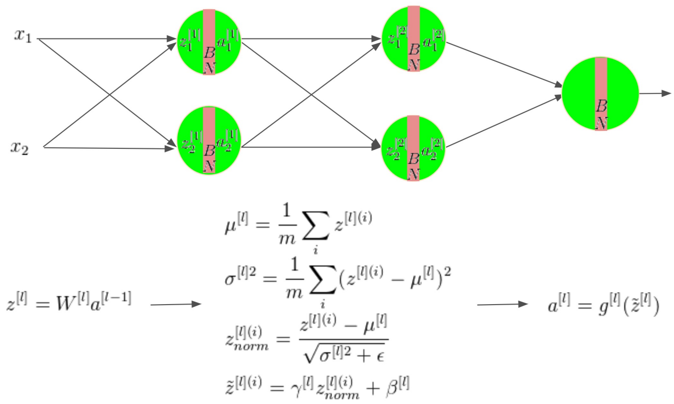
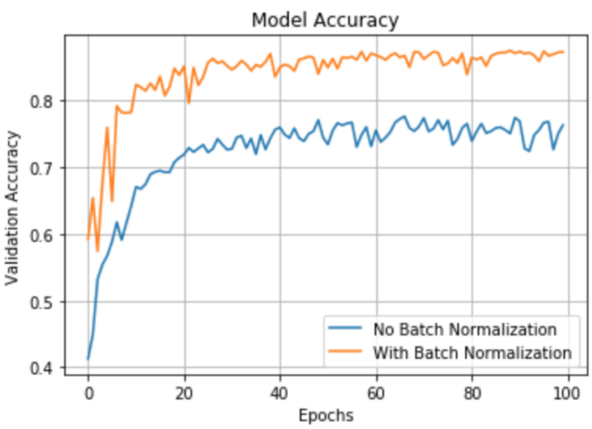


Image source: [earnopencv (https://www.learnopencv.com/batch-normalization-in-deep-networks/)](https://www.learnopencv.com/batch-normalization-in-deep-networks/)

```
model = Sequential
model.add(Dense(32))
model.add(BatchNormalization())
model.add(Activation('relu'))
```



Some strategies while developing an ML/DL model-

1. Carrying out error analysis
2. Pretraining
    - Transfer learning
    - Self-supervised learning
3. Multi-task learning
4. End-to-end modeling
5. Domain adaption
6. Self-training

Carrying out error analysis

- Error analysis - process of manually examining mistakes that your algorithm is making.
- It can give you insights into what to do next
- Cleaning up incorrectly labeled data

Pretraining - Transfer Learning (TL) and Self-supervised Learning (SSL)

- Pretraining has become a standard technique in CV, NLP
- Transfer learning (TL) uses labeled data to learn a good representation network - Supervised fashion
- Self-supervised learning does not require annotated labels

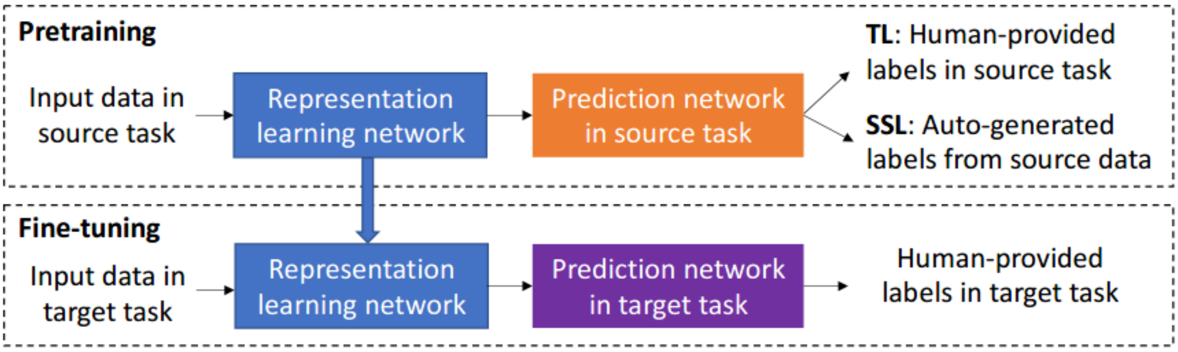


Image source: [A Tale of Two Pretraining Paradigms (https://arxiv.org/pdf/2007.04234.pdf)](https://arxiv.org/pdf/2007.04234.pdf)

Transfer learning

- Let's consider we have on ASR model for Tamil.
- Can we use that model to train an ASR model for Telugu?

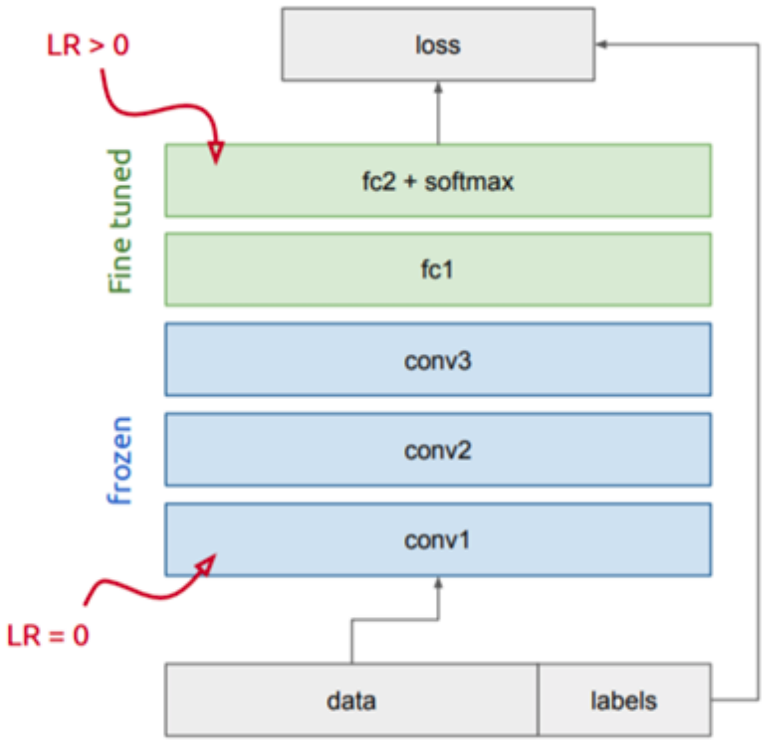
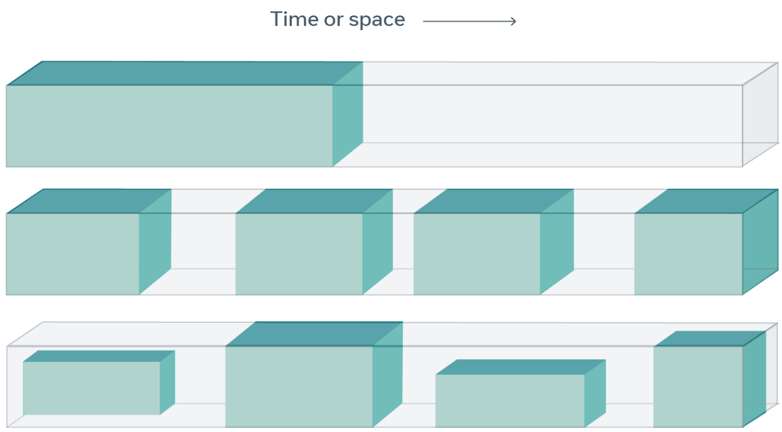


Image source: [towardsdatascience (https://arxiv.org/pdf/2007.04234.pdf)](https://arxiv.org/pdf/2007.04234.pdf)

Self-supervised based pretraining model

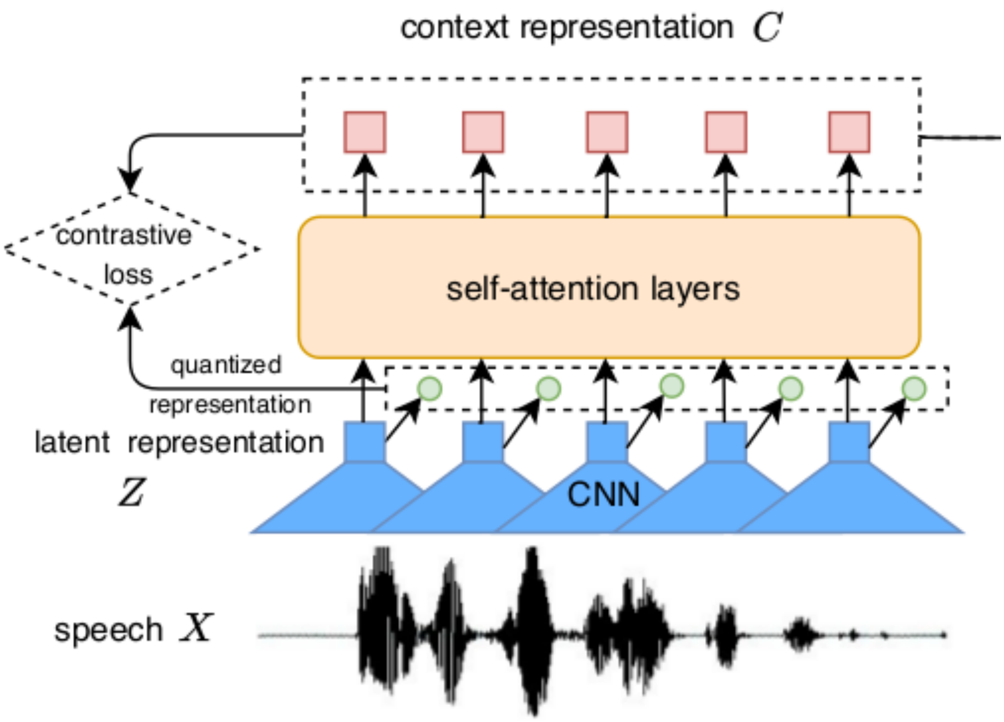
- What is self-supervised learning?
  - Self-supervised learning obtains supervisory signals from the data itself
  - Labels are naturally part of the input data
  - Learn general data representations from unlabeled examples
  - Fine tuning for your downstream tasks



[Self-supervised-learning (https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/)](https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/)

Wav2vec2

- Wav2Vec2 learns powerful speech representations from large amount of unlabeled speech
- It learns contextualized speech representations by randomly masking feature vectors before passing them to a transformer network
- **Pretraining** and **Finetuning**
- [Facebook Pretrained model (https://github.com/pytorch/fairseq/blob/master/examples/wav2vec/README.md)](https://github.com/pytorch/fairseq/blob/master/examples/wav2vec/README.md)



[wav2vec2 (https://arxiv.org/abs/2006.11477)](https://arxiv.org/abs/2006.11477)

Wav2vec2 for ASR

- Wav2Vec2 is fine-tuned using CTC loss with transcribed data

Model	Unlabeled data	LM	dev clean	dev other	test clean	test other
10 min labeled						
BASE	LS-960	None	46.1	51.5	46.9	50.9
		4-gram	8.9	15.7	9.1	15.6
		Transf.	6.6	13.2	6.9	12.9
LARGE	LS-960	None	43.0	46.3	43.5	45.3
		4-gram	8.6	12.9	8.9	13.1
		Transf.	6.6	10.6	6.8	10.8
LARGE	LV-60k	None	38.3	41.0	40.2	38.7
		4-gram	6.3	9.8	6.6	10.3
		Transf.	4.6	7.9	4.8	8.2
1h labeled						
BASE	LS-960	None	24.1	29.6	24.5	29.7
		4-gram	5.0	10.8	5.5	11.3
		Transf.	3.8	9.0	4.0	9.3
LARGE	LS-960	None	21.6	25.3	22.1	25.3
		4-gram	4.8	8.5	5.1	9.4
		Transf.	3.8	7.1	3.9	7.6
LARGE	LV-60k	None	17.3	20.6	17.2	20.3
		4-gram	3.6	6.5	3.8	7.1
		Transf.	2.9	5.4	2.9	5.8
10h labeled						
BASE	LS-960	None	10.9	17.4	11.1	17.6
		4-gram	3.8	9.1	4.3	9.5
		Transf.	2.9	7.4	3.2	7.8
LARGE	LS-960	None	8.1	12.0	8.0	12.1
		4-gram	3.4	6.9	3.8	7.3
		Transf.	2.9	5.7	3.2	6.1
LARGE	LV-60k	None	6.3	9.8	6.3	10.0
		4-gram	2.6	5.5	3.0	5.8
		Transf.	2.4	4.8	2.6	4.9
100h labeled						
BASE	LS-960	None	6.1	13.5	6.1	13.3
		4-gram	2.7	7.9	3.4	8.0
		Transf.	2.2	6.3	2.6	6.3
LARGE	LS-960	None	4.6	9.3	4.7	9.0
		4-gram	2.3	5.7	2.8	6.0
		Transf.	2.1	4.8	2.3	5.0
LARGE	LV-60k	None	3.3	6.5	3.1	6.3
		4-gram	1.8	4.5	2.3	4.6
		Transf.	1.9	4.0	2.0	4.0

[wav2vec2 (https://arxiv.org/abs/2006.11477)](https://arxiv.org/abs/2006.11477)

Wav2vec2 learned speech embeddings for other downstrem task

- **Speaker verification and language identification (https://arxiv.org/pdf/2012.06185.pdf)**
- **Emotion recognition (https://arxiv.org/abs/2104.03502)**
- **ASR model development for low-resource language (https://arxiv.org/abs/2012.12121)**

ASR Development uisng wav2vec2 for Indian language

1. Fine-tuned on three different databases provided by IITM
   - **Pretained model: XSLR-53 (https://github.com/pytorch/fairseq/blob/master/examples/wav2vec/README.md) | Trained on 56000 hours of speech data of 53 different languages**

Language	Train (Hours)	Eval (Hours)	WER	LM (kenLM)
Indian English	179.5	5.4	4.91 %	6 Gram
Hindi	178.4	4.9	4.55 %	5 Gram
Tamil	104.5	3.8	5.84 %	4 Gram

1. Kaldi based TDNN model

Language	Train (Hours)	Eval (Hours)	WER	LM (SRILM)
Indian English	179.5	5.4	4.97 %	RNNLM Rescore
Hindi	178.4	4.9	3.73 %	5 Gram
Tamil	104.5	3.8	5.21 %	5 Gram

Wav2vec2 pretrained model Indic languages

- EkStep Models (https://github.com/Open-Speech-EkStep/vakyansh-models)
- Paper (https://arxiv.org/pdf/2107.07402.pdf)

Multi-task learning

1. One neural network learns several tasks at the same time
2. Each of these tasks helps all of the other tasks

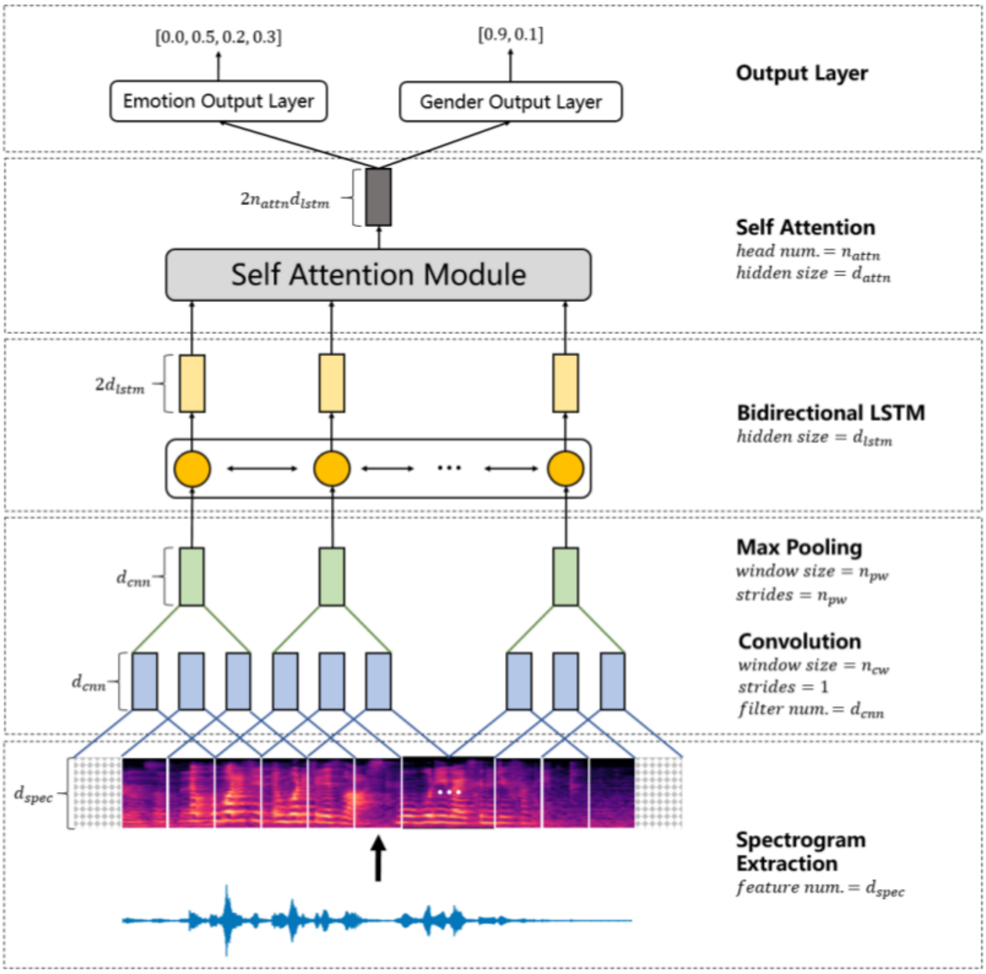


Image source: [isca (https://www.isca-speech.org/archive/Interspeech_2019/pdfs/2594.pdf)](https://www.isca-speech.org/archive/Interspeech_2019/pdfs/2594.pdf)

End-to-end modelling

- No feature engineering and no intermediate stages
- Need large amount of data

Evaluation metric:

Highly depends on your application and what you are trying to optimize for

1. Confusion matrix
2. Accuracy
3. Precision / Recall / F1 score
4. Word error rate in ASR
5. Bilingual Evaluation Understudy (BLEU) Score in machine translation
6. Equal error rate in speaker verifiction

Basic metrics you should know

H/P	+ class	- class
+ class	True positive	False negative
- class	False positive	True negative

[When FP and FN will be useful!]

1. FN should be zero: cancer diagnosis
2. FP should be zero: speaker verification

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

Precision (proportion of positive identifications which was actually correct) = $\frac{TP}{TP+FP}$

Recall = (proportion of actual positives which was identified correctly) = $\frac{TP}{TP+FN}$

F1 score = HM(Precision, Recall)

References:

1. Structuring Machine Learning Projects : [Andrew Ng (https://www.coursera.org/learn/machine-learning-projects)](https://www.coursera.org/learn/machine-learning-projects)
2. Visualization of ML techniques : [egfycat (https://gfycat.com/gifs/search/gradient+descent)](https://gfycat.com/gifs/search/gradient+descent)
3. CNN materials: [cs231n.stanford.edu (http://cs231n.stanford.edu/)](http://cs231n.stanford.edu/)
4. Andrew Ng DL notes: [cs230.stanford.edu (https://cs230.stanford.edu)](https://cs230.stanford.edu)
5. MOOCS: Coursera & EDx
6. Read ML articles in [https://medium.com (https://medium.com)](https://medium.com)
7. [Machine Learning Yearning (https://d2wvfoqc9gyqzf.cloudfront.net/content/uploads/2018/09/Ng-MLY01-13.pdf)](https://d2wvfoqc9gyqzf.cloudfront.net/content/uploads/2018/09/Ng-MLY01-13.pdf)

You can reach me @

1. email: sisiitg@gmail.com
2. mobile no.: 9435379331
3. [Linkedin (https://www.linkedin.com/in/sishir-kalita-3a006120/)](https://www.linkedin.com/in/sishir-kalita-3a006120/)

THANK YOU!

Stay safe

```
In [ ]:
```