15. Oct. 2024 Wpl.2

# II. Element of Hypothesis. Test.

## B. Beyesian Hypothes Testing.

$H_0: Y \sim P_0$ vs $H_1: Y \sim P_1$

$Y$ has distribution $P$.

$H_0$: Null , $H_1$: alternative.

Desision rule $\delta$ on $\Gamma$

$$\delta(y) = \begin{cases} 1 & \text{if } y \in \Gamma_1 \\ 0 & \text{if } y \in \Gamma_1^c \end{cases}$$

$C_{ij}$, $i=0,1$ and $j=0,1$

$C_{ij}$ : the cost of chosing $H_i$ when $H_j$ is true.

$$R_j(\delta) = C_{1j} P_j(\Gamma_1) + C_{0j} P_j(\Gamma_0), \quad j=0,1$$

$R_j(\delta)$ =

cost of chuos $H_1$ when $H_j$ is true

$\times$

The probability of to do so.

$+$

cost of choosing $H_0$ when $H_j$ is true.

$\times$

The probability of to do so.

$\pi_0$: the prior probabilite of occurrences of $H_0$

$\pi_i$: $\sim$ of $H_i$

Average of Bayes risk:

$$r(\delta) = \pi_0 R_0(\delta) + \pi_1 R_1(\delta)$$

$$r(\delta) = \sum_{j=0}^{1} \pi_j C_{0j} + \int_{\Gamma_1} \left[ \sum_{j=0}^{1} \pi_j (C_{1j} - C_{0j}) p_j(y) \right] \mu(dy)$$

thus. minimum over $\Gamma_1$ if we choose.

$$\Gamma_1 = \{ y \in \Gamma \mid \pi_1 (C_{11} - C_{01}) p_1(y) \le \pi_0 (C_{00} - C_{10}) p_0(y) \}$$

Assume $C_{11} < C_{01}$

$$\Gamma_1 = \{ y \in \Gamma \mid p_1(y) \ge \tau p_0(y) \}, \quad \tau = \frac{\pi_0 (C_{10} - C_{00})}{\pi_1 (C_{01} - C_{11})}$$

likelyhood-ratio test.

$$L(y) = \frac{p_1(y)}{p_0(y)} \quad \text{(likelyhood ratio)}$$

$$\delta_B(y) = \begin{cases} 1 & \text{if } L(y) \ge \tau \\ 0 & \text{if } L(y) < \tau \end{cases}$$

---

$\pi_0(y)$, $\pi_1(y)$ are posterior of two hypotheses.

$$\Gamma_1 = \{ y \in \Gamma \mid C_{10} \pi_0(y) + C_{11} \pi_1(y) \le C_{00} \pi_0(y) + C_{01} \pi_1(y) \}$$

$C_{i0} \pi_0(y) + C_{i1} \pi_1(y)$ posterior cost of choosing $H_i$ when $y$ is observed.

---

Example:

Y (output of channel) can be 1, 0.

| True msg | received | chance |
|---|---|---|
| zero | 1 | $\lambda_0$ |
| zero | 0 | $1 - \lambda_0$ |
| one | 0 | $\lambda_1$ |
| one | 1 | $1 - \lambda_1$ |

$1-\lambda_j$, $j$ was transmitted

$Y$ is $\{0,1\}$ observation set.

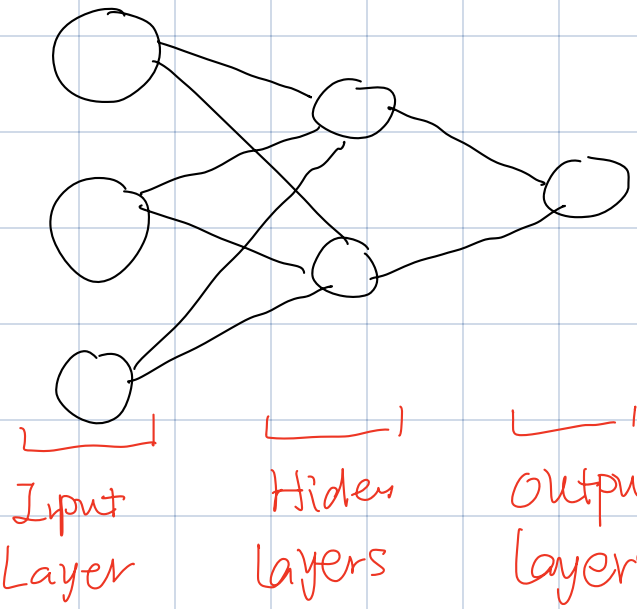
$Y$ has densities $p_j(y) = \begin{cases} \lambda_j & \text{if } y \neq j \\ 1-\lambda_j & \text{if } y = j. \end{cases}$

24, Oct, 2024 WP 1.2

Deep Learning Crash Course for Beginners. (YouTube Tutorial)

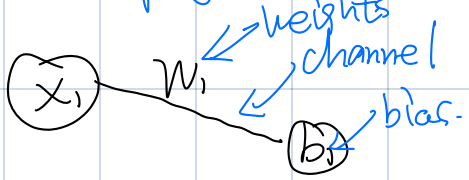
DL can learn features from raw data.

A simple Neural Network




Learning process

Forward Propagation.




$\hat{y} = \sigma\left(\sum_{i=1}^{n} x_i w_i + b_i\right)$

$\sigma$: activation function.

## Back Propagation.

$\hat{y}$ — Right

Wrong → Loss function quantify the deviation from expected output.

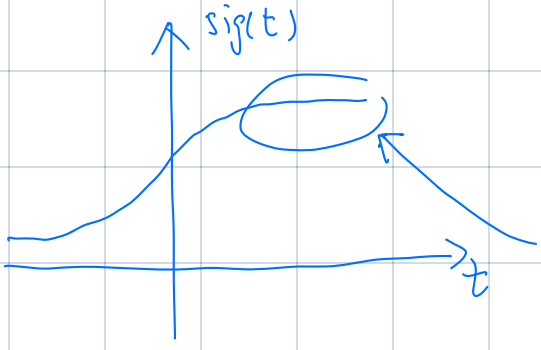
→ Go backwards & adjust initial weights and bias.

## Terminologys.

### Activation Functions:

- Introduce Non-Linearity.
- Decide whether a neuron can contribute to next layer.

Sigmoid Function. ( $sig(t) = \frac{1}{1+e^{-t}}$ )

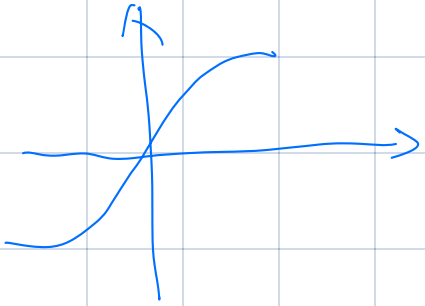



Non-linear, Analog outputs.

$sig(t) \in (0, 1)$

gradient vanish. problem.

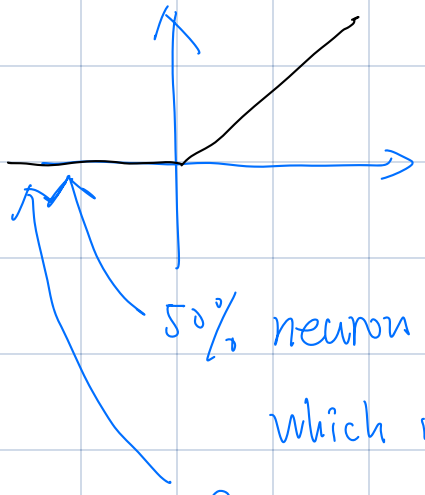
Tanh Function. $\left(\frac{2}{1+e^{-2x}} - 1\right) = \tanh(x) = 2\,\text{sigmoid}(2x) - 1$



Derivative steeper than sigmoid.

still gradient vanish problem.

# ReLU Function. (Rectified Linear Unit)



sparse Activations.

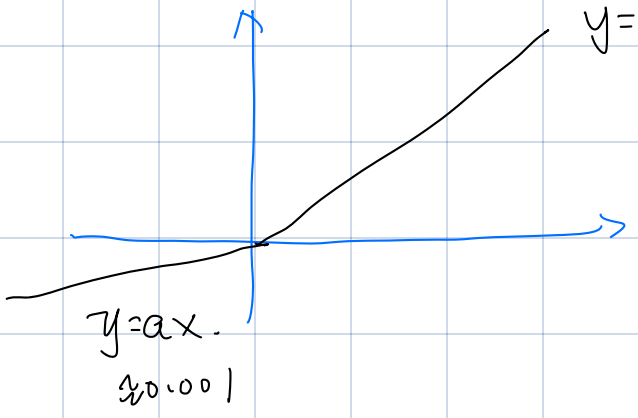
50% neuron initialized at negtive axis.

which make network lighter.

Gradent = 0, Dying ReLU Problem.

Not get adjusted in descent.

# Leaky ReLU Function.




ReLU is less computationally expensive than Tanh or sigmoid.

# Loss Function.

Regression, Binary Classification, Multi-Class Classification.

# Optimizers.

minimize loss function.

by updating the network based on the output of Loss function.

Loss function guide optimizers.

## Gradient Descent.

$$\nabla f(x,y) = \left(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\right)$$

Gradient always points in the steepest direction.

Choose proper Learning Rate to avoid local minimum.

- too larger — overshoot
- too small — local minimum.

## Stochastic Gradient Descent.

using subset of sample "batch"

- less computation.

using momentum.

- get rid of local minimum

$$\begin{cases} \theta_{t+1} = \theta_t - \eta \nabla J(\theta_t) \\ v_{t+1} = \beta v_t + \nabla J(\theta_t) \end{cases}$$

$$\Rightarrow \theta_{t+1} = \theta_t - \eta v_{t+1}$$

## Adagrad:

adapts learning Rate to individual features.

Ideal to sparse dataset.

but:

$\eta$ get small overtime.

$$G_{t,i} = G_{t-1,i} + (\nabla J(\theta_t)_i)^2$$ ← accumulating gradient²

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \nabla J(\theta_t)_i.$$

RMSprop.

accumulates Gradients in a fixed window.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\nabla J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla J(\theta_t)$$

27 Oct. 2024 WP1.2

Adam. ( Adaptive Moment Estimation)

Using momentum too

Pretty widespread today.

4. Parameters & Hyperparameters.

Model Parameter.

value define the skill of the model

estimated directly from data.

like: weight, bias.

Model Hyperpara...

set manually to controll the behavior of model

like: learning rate, Layer and neuron numer

Epochs, Batches, Batch size and Iterations.

epochs: entire dataset pass to network;

too many epochs lead to overfitting.

batches: subset of training dataset used to model in one iteration.

batch size: total numer of training examples in a batch.

iterations: number of batches need to complete one epoch.

## Introduce to learning:

### Supervied learning:

learn by exple.

train model on well-labelled. data.

- Predict the correct label for unseen data.

sub category.

Classification:

creating mapping function.

i Handwriting Digits Dataset

Algorithms:

Linear classifer, Support vector machines,

K-Nearest Neighbour, Random forest.

### Regression:

find a relationship between dependent & independent variables.

Algorithms:

linear regression, lasso regression, Multivariat regression.

Unsupervised learning:

find features of data; no label.

hidden patter

subcategory:

clustering:

grouping data into different clusters of groups

Partitional clusting:

a data point can belong to a single cluster

hierarchical clusting:

clusters within clusters

data point may belong to many clusters.

Algorithms:

K-Means, expectation maximization, NCA.

28. Oct. 2024. WPL 2.

Association:

find relationships between different entities.

Reinfocement learing: (goal-oriented)

enable an agent to learn in interactive environment by trial & error.

rewards & punishment used to guide agent

Maximize total cumulative reward.

# Regularization.

core problem : overfitting.

performing well on training set

but not on testing set.

## tacking Overfitting.

Dropout : randomly removes some nodes.

Augmentation : Fake data ; More Data , Better model.

early stopping : stop when validation error increases.

# Neural Network Arch.

## Fully-Connected Feed Forward Neural Networks. (Vanilla)

each neuron is connected to every subsequent layer with no backward connections.

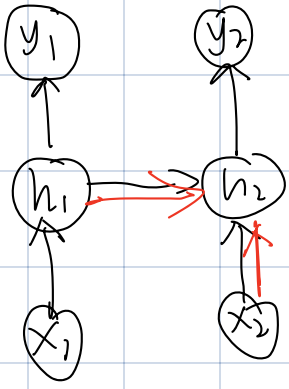
[Inputs, outputs, Hidden Layers, Neurons per hidden layer, activation functions]

⇒ wider array of problem.

- can't handle sequential data. (sentence ...)
- don't share parameters. across time.

# Recurrent Neural Net. (RNN)

use feedback loop in hidden layers.




... use previous states as input for its current prediction.

use backpropagation algorithm.

## Vanishing Gradient Problem.

gradient will shrink over network due to the nature of backpropagation, leading slow-learning of front layers. ⇒ short-term memory.

## Solutions:

Long short term memory (LSTM)

Gated RNN (GRNN)

GRNN has two gates:

Update gates.

Reset gates.

LSTM has three gates:

update gates. Reset Gate, Forget gate

# Convolutional NNs.

good for processing data like:
images, audio and video

## Hidden Layers in a CNN

Convolution layer. convolutional and pooling function are used.
Pooling ~
Fully-Connected ~
Normalization ~

## Convolution

extract visual features in chunks (filter or kernel)

## Pooling.

reduce the number of neurons necessary in subsequent layer.

Max pooling: pick biggest number in selected region.

Min pooling: ~

# Creating a deep learning model

## Gathering Data.

right data is key.

size:

Amount of data ≈ 10× No. of model parameters.

Quality:

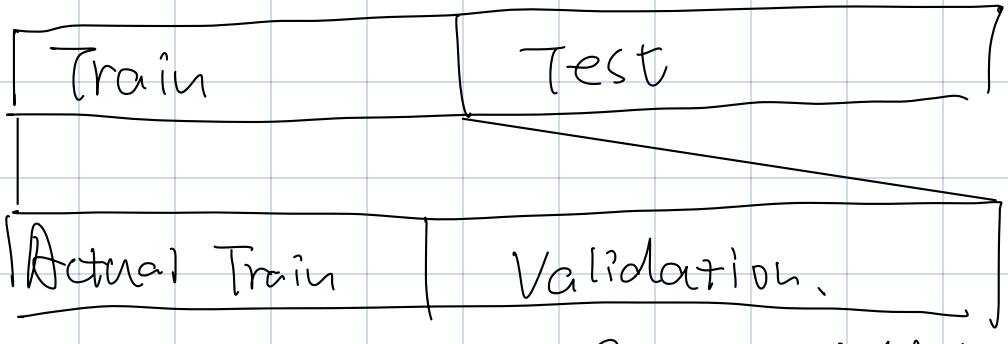
archive.ics.uci.edu.

Kaggle.com/datasets.

Google Dataset Seach.

## Preprocessing the Data.

- splitting dataset into subsets.
  : training, validation, testing.




Cross-Validation.

- formatting:
  might not in the format we need.

- Missing Data.
  represented as "NaN" or "Null"
  1. eliminating features with missing values.
     but risk losing relevent info.
  2. inputing the missing values.
- Sampling.
  use smaller sample of large dataset.
  imblancing data:
  learning majority class > learning minority class
  ⇒ bias to majority.
  Example weight = original weight × Downsampling Factor.
  ⇒ increase weight of majority to compensate decrease in number of sample.
- Feature Scaling.
  1. Normalization.
  2. standardization. ✳

Train.

Evaluation.

test model on the validation set.

Optimizing -

tuning Hyperparameter -

No. epochs ↑

adjust learning rate -

addressing Overfitting -

- get more data.
- Reduce model size.
- Regularization.

  $L_1$ Reg. $\Rightarrow$ add cost to limit complexity

  $L_2$ Reg of model.
- Data augmentation -

  in case of image - [flipping, blurring, zooming]
- Drop out.

14. Nov. 2024.

--model., --channel, --src, --dst, --snr, --number of set

① training set [0, 1] or some other value

② AWGN - SNR

Rician fading - $K$ : ratio between power in direct path and that in other

$\Omega$ : total power from both paths.

Rayleigh fading -

Generate nioseed signal [selecting channel type]

seleting model:, channel type, size of DateSet.

generating training set: output a files

train 0.01 ~ 0.001

Adam

| channel_type | Model_type | BER | |
|---|---|---|---|
| AWGN | DNN | 0.0883 | 1.000 |
| | | 0.0387 | |
| | CNN | 0.1010 | |
| | | 0.0733 | |
| | | 0.1258 | |
| | | 0.1107 | |

SGD

# of Layers 0.0248

Optimize, lossFunction, activation 0.0230

Epoch = 20? Does matrics matter?

0.0250

What channel I record for log

0.002 Rayleigh

0.0097 Rician

## Meeting 20 Nov.

- using sigmoid at output layer.
- noised signal should be complex number.
- try to implement first-order mate-learn.

## Outcome after solve complex and sigmoid.

Epochs = 50, batch size = 32, data size = 10k

test size = 0.2 x 10k.

512, leaky_relu.

256, ~

128, ~

2, sigmoid.

| Channel | BER |
|---|---|
| AWGN | 0.03187 |
| Rayleigh | 0.14497 |
| Rician. | 0.13906 |

Jan. 30, 2025.

Cognition refreshing.

train DL1 with AWGN normally.
Test DL1 with AWGN test set.

train DL2 with Rayleigh ...

train DL3 with Rician.

train model4 with 3 noise types.

11/2/2025.

Failed to learn Rayleigh.
Overfit the AWGN

D1 AWGN,
D2 Rician,
D3 Rayleigh,
D4 Random Mixed channel,
D5 Sequential Mixed channel,

} Convential DNN

D6 Random Mixed channel,
D7 Sequential Mixed channel,

} Meta-learing DNN

## Generalization Testing

$D_{1\sim7}$ 3GPP with limited data.

Control Both # of Train Data.
and # of Validating Data.

Train Date analogizes data collected from enviroment for training

Validating Date analogizes data inputed after

2025. Feb. 21

Project flow built:

Train Phase:

D1 AWGN,
D2 Rician,
D3 Rayleigh,
D4 Random Mixed channel,
D5 Sequential Mixed channel,

} Convential DNN

D6 Task { Random_mixed, Sequential_mixed } → Meta_learning DNN with Retike Algorithm.

Fine Tune Phase:

D1-6 3GPP [100, 500, 1000]

Generlizing Testing Phase:

Validated after each Fine-tune

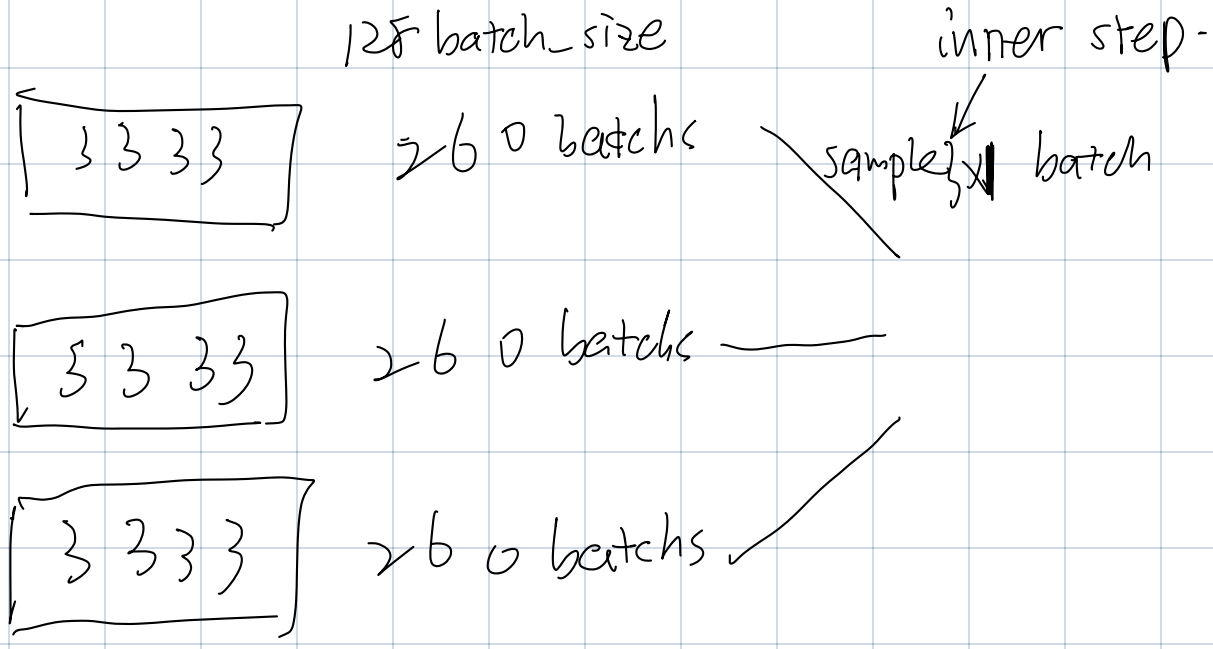
Loss and BER of $D_1$ (AWGN) get lowest point at 2nd, 3rd Epich but bounce back in following Epoch.

→ learn rate too large?

D2-5 present decreasing trend in both loss and BER.

→ learn rate too small / more epoch needed.

In generalization test, meta learning appears learning nothing (BER ≈ 0.5)




260 × 128 × 3 × 1 ← one task
128 samples
3 step / batch


100000 ≈ 86 epoch × 3 task × 3 inner step × 128 minibatch

total sample into training aligned.

2025/2/27

Reptile from OpenAI.

introduce meta-batch

与其每一个 epoch 都能在 3 个 task 学习，

meta-batch 允许 model 在不同数量 task 上学习

Meta_learning descent too slow.

meta epoch : 10 => 2000

inner learn rate : 0.001 => 0.003

reptile meta learning rate : 0.01 => 0.25

self_adjust new_meta_lr = $(1 - \frac{\text{epoch}}{\text{total epoch}}) \times$ meta_lr.

The BER of meta DNN decresed on the scale of 2000 iteration.

But still wrose than Traditional DNN

Meta iteration ≠ DNN epoch.

=> number of times parameter updated.

For 100k sample to Traditional DNN

$$\#\text{update} = \frac{\text{Total Sample}}{\text{batch\_size}} \times \text{Number of Epoch.}$$

Reptile

#update = meta_iteration.

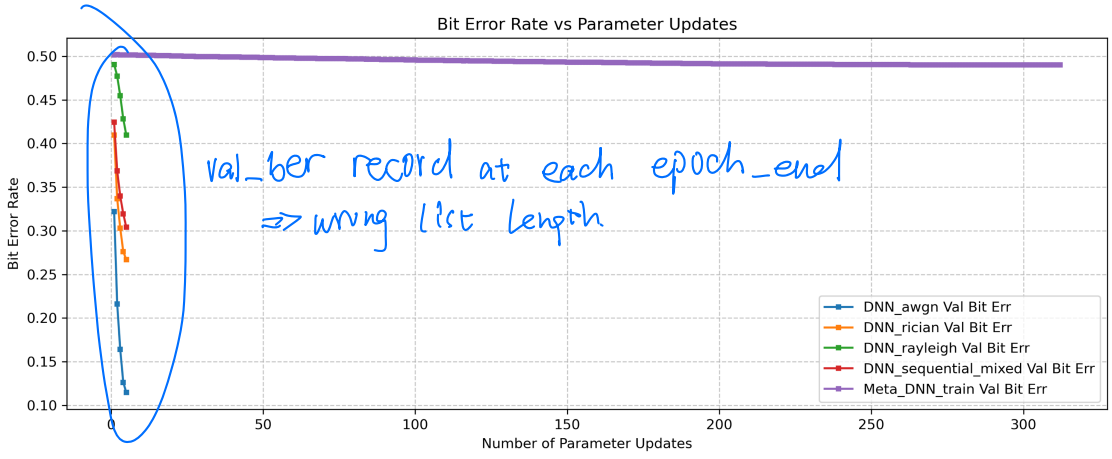
passed a list into a list ⇒ failed to output figure.

2025/3/4

100k total

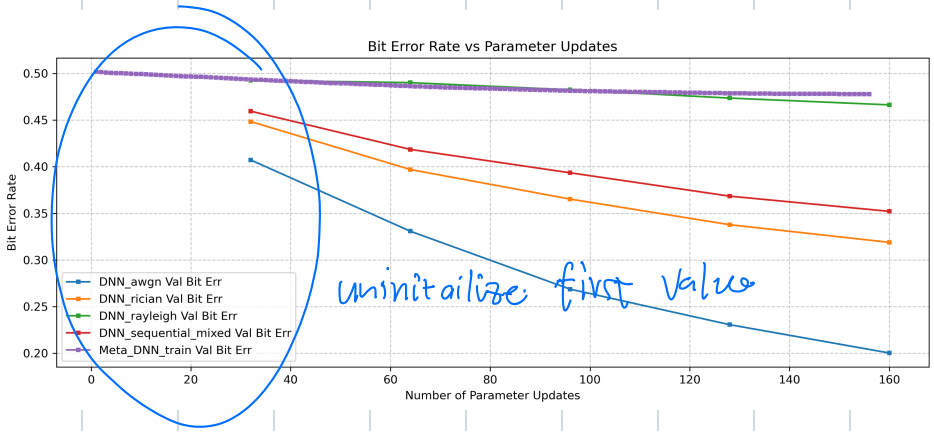
$$\frac{100k}{128} \times 20$$

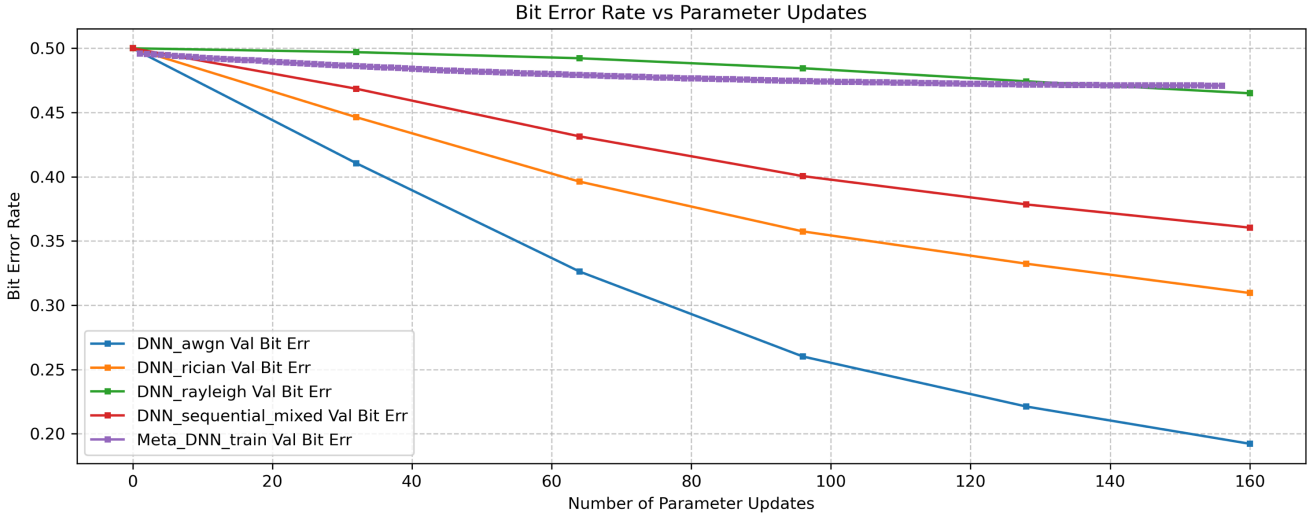
mini_batch sampled from task, put into inner_update.




val_ber record at each epoch_end
⇒ wrong list length

trace where it write down the val_ber by adding val_update_counts[]




set first value = 0.5.

2025/3/5

In fine tune phase, model get pouluted

=> get cloned before fine tune

---

Current meta_lr. using Linea decay

$$\text{meta\_lr} = \left(1 - \frac{\text{current\_iteration}}{\text{total}}\right) \times \text{meta\_lr}$$

=> hyper gradient?

calculate the gradient of $(\phi' - \phi)$ and update meta_lr based on grad.

⇒ Cosine Annealing

$$y = y_{min} + \frac{1}{2}(y_{max} - y_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right)$$

⇒ use: ConsineDecayRestarts

first_decay = ~~1000~~, 4k0 get converge...

t_mul = 1.3, m_mul = 0.9, alpha = 0.001

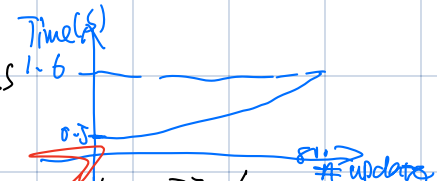
further work: inner_lr update optimize to avoid overfit or local_min.

○ try to decrease train time of Meta-DNN

900 update,

$$\text{inner iteration} = \frac{\text{DNN samples}}{\text{minibatch\_size} \times \text{Number of task}} = 27$$

| Type | Time |
|---|---|
| plain | 609.43s 1.01[s]/meta_update |
| ~~@tf.function~~ → future work | |
| ~~parrallel~~ → future work | |
| CosineRestar | 667.63s |
| @tf.function (on inner_update) | 345.78s 1.057[s]/update |




Tensorflow repeatly recompiling

legend issue in 3gpp phase, take average or place it outside.

let reduce_retracing = True

allow tensor to trace graph

version control: DNN function well → 0.1.1

update result naming 0.1.2

add mixed 0.1.3

Multi BCP 0.2.3

basic metaDNN 0.3.3

Train phase 0.4.3

mini_batch 0.4.4

training figure (plot_by_update) 0.5.4

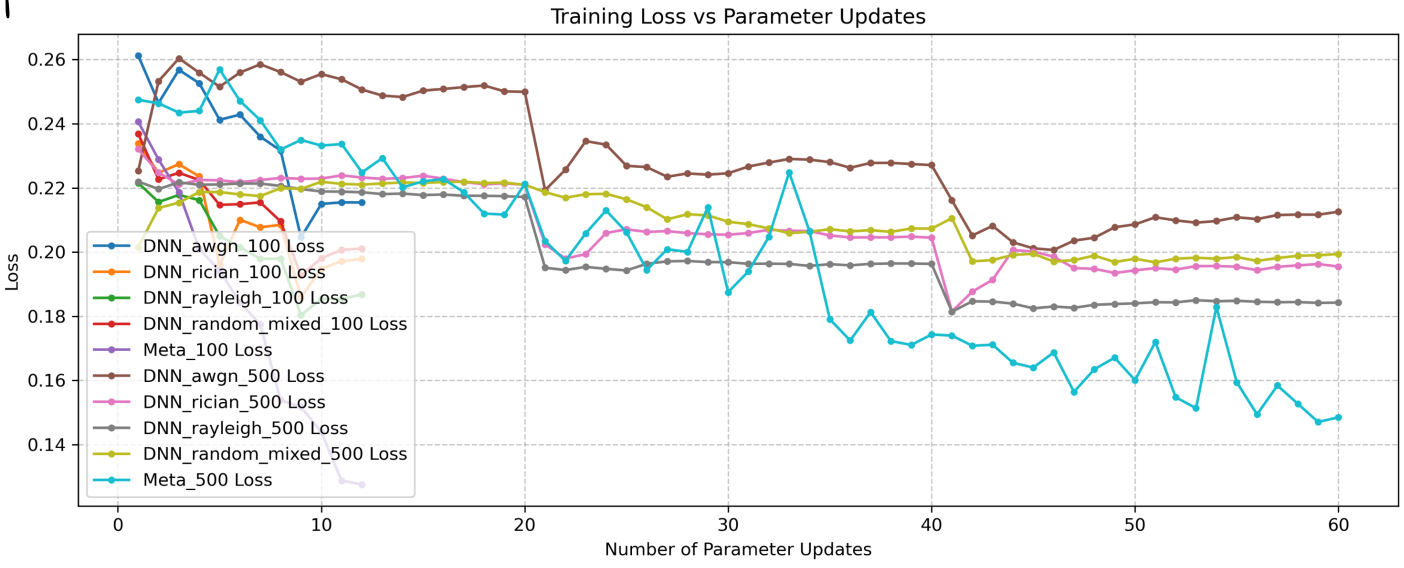
Generate test, colorrestart, 0.6.5
Time trace -

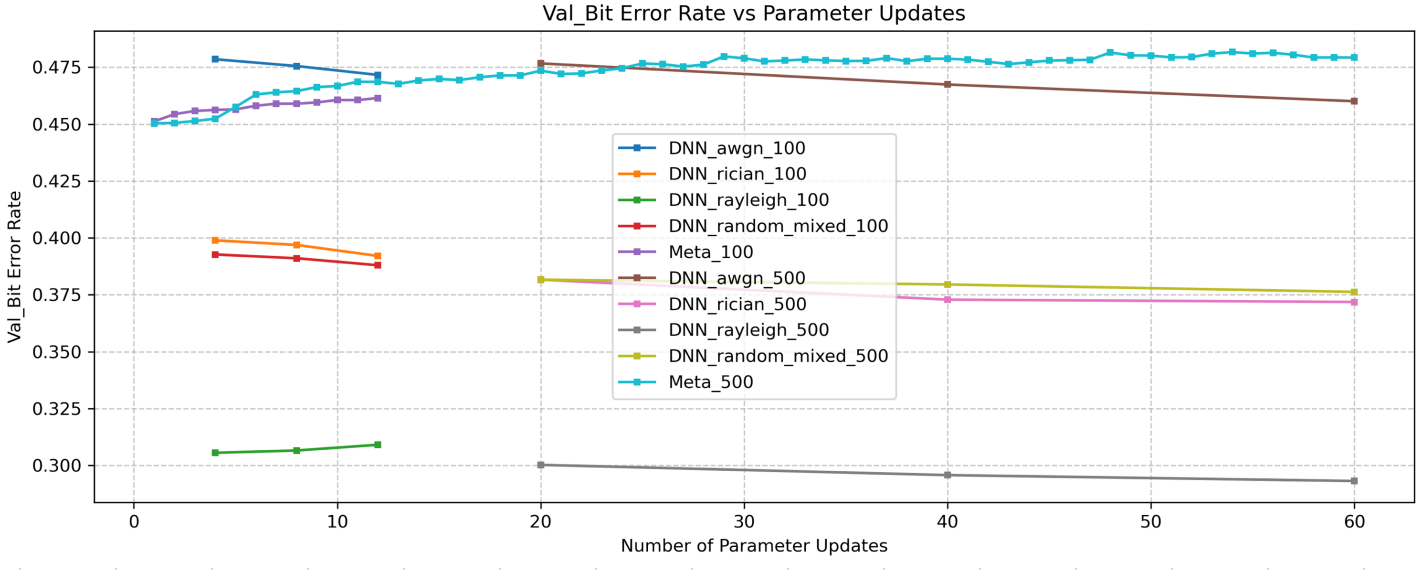
reuse meta_grads

to optimize memory management

Initialize the inner train function with model object and decorated with tf.function to utilize graph

update_time. 800, Meta_ro_end BER=0.408828



update 600 → ber trapped at 0.4

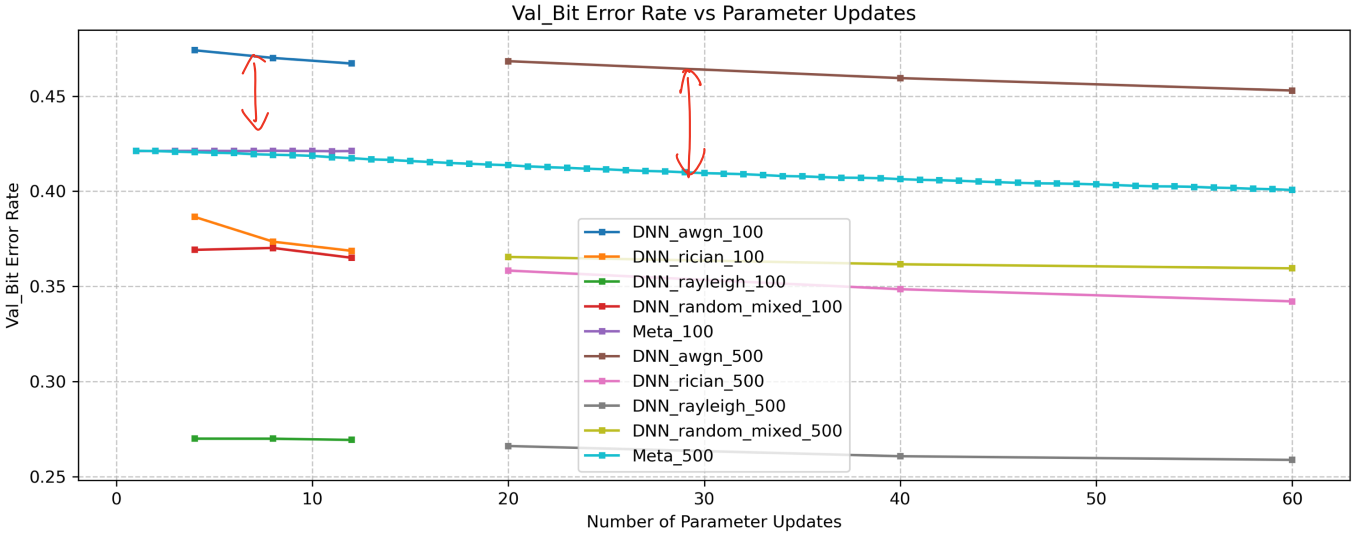
remove best_weith keeper, enumerate update

→ trapped at 0.4

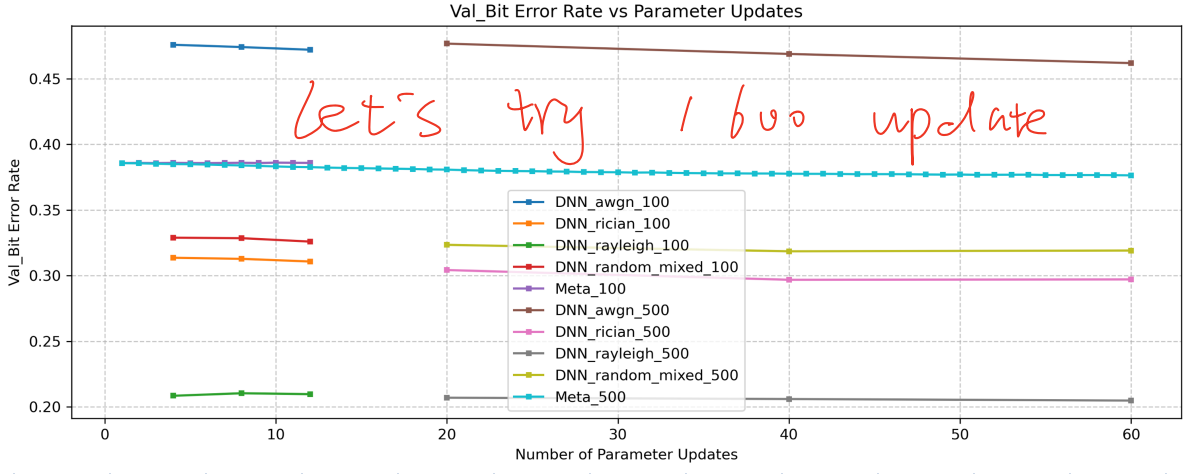
remove tf.function ✓

final keep : Performance tracker, CosineRestarter.

update : 800



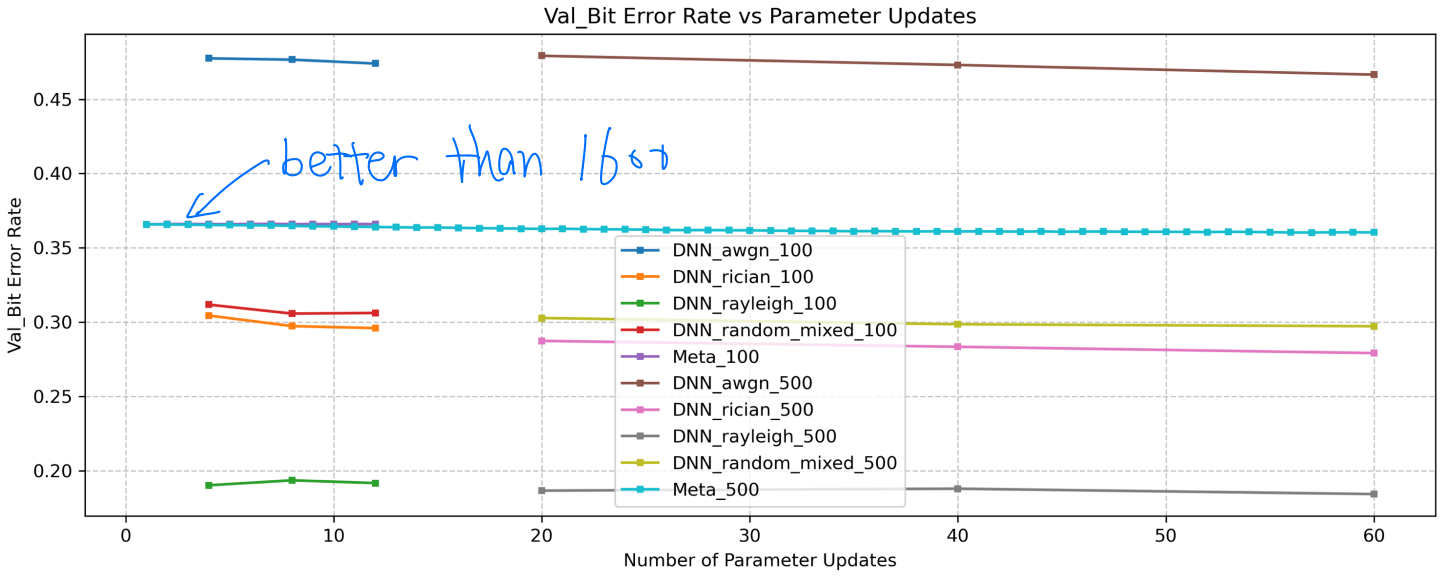

DNN is not fit enough to the respective channels



2024.3.10

Let's try 2400 updates

What expected is meta_DNN performs better than DNN. on 3gpp channel.

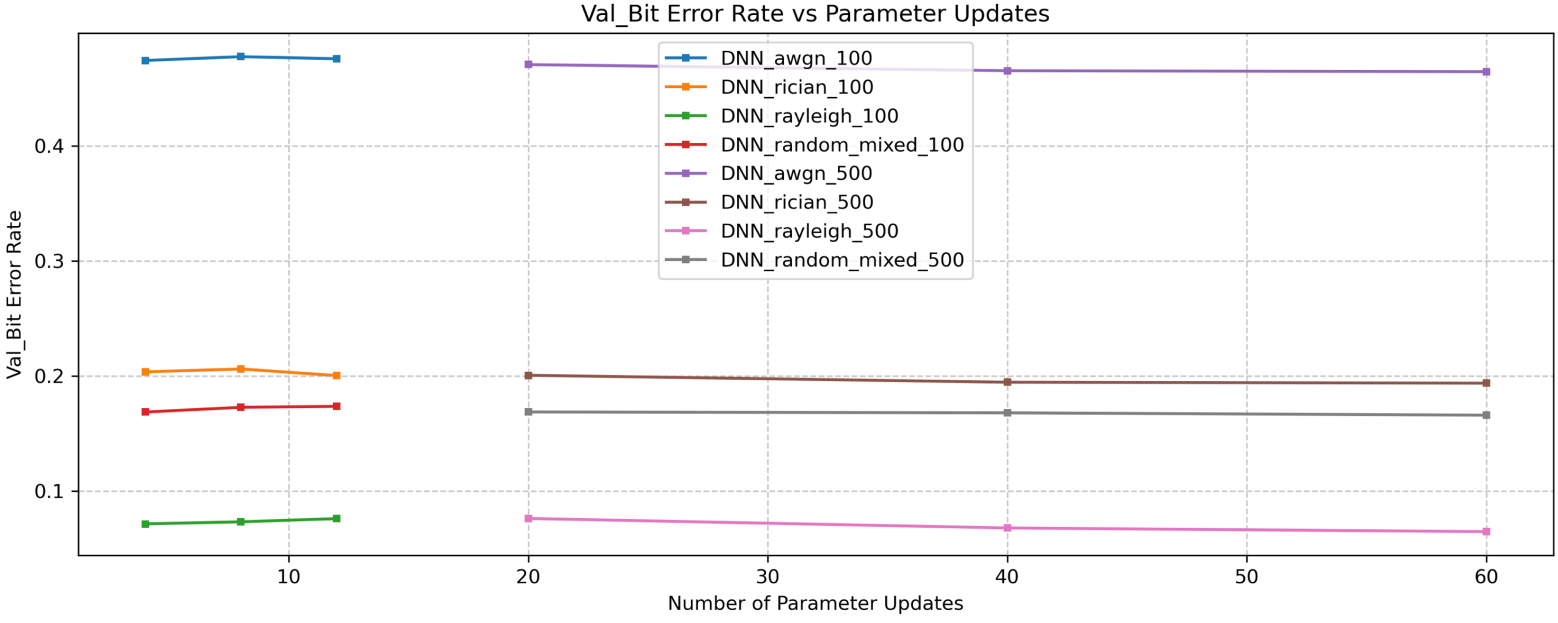


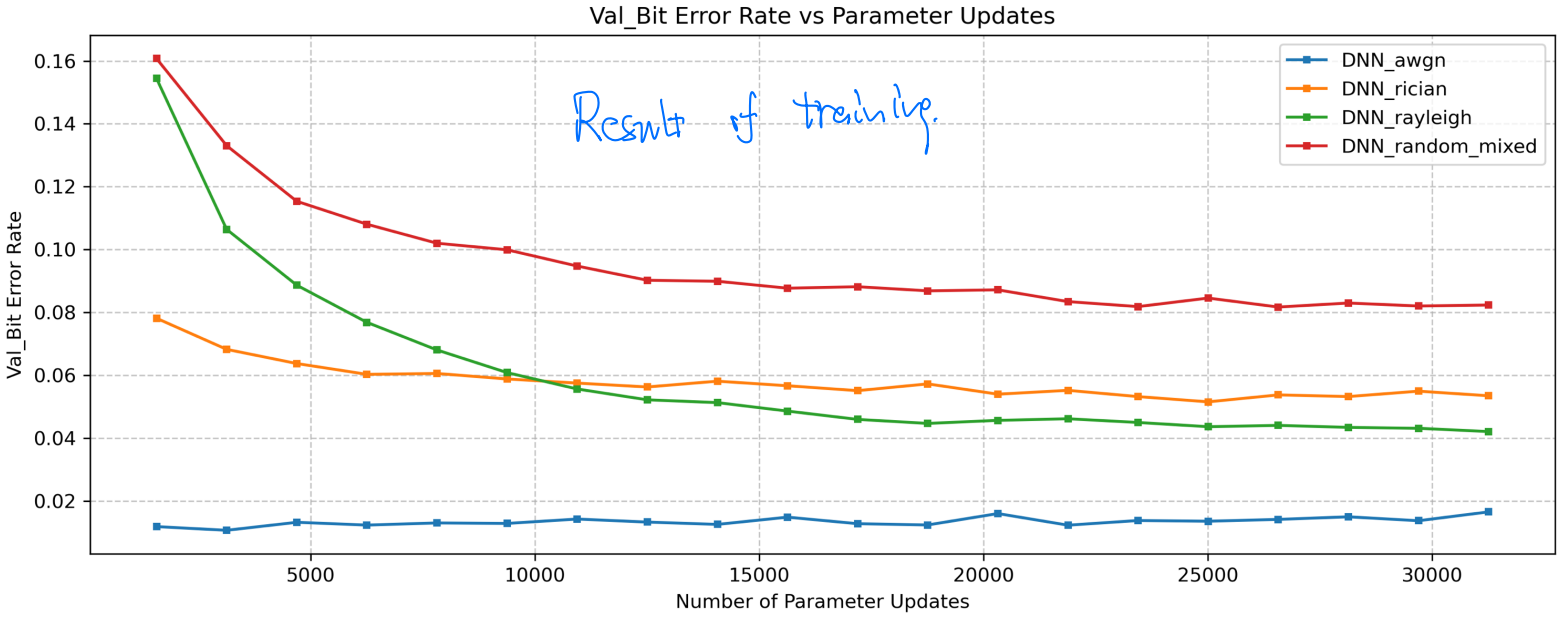

better than 16st

try 10k sample = 31250 update

(2epoch, 64batch size only DNN

What expected is well-trained DNN will perform worse on 3GPP






2025/3/11

Supervisor meeting

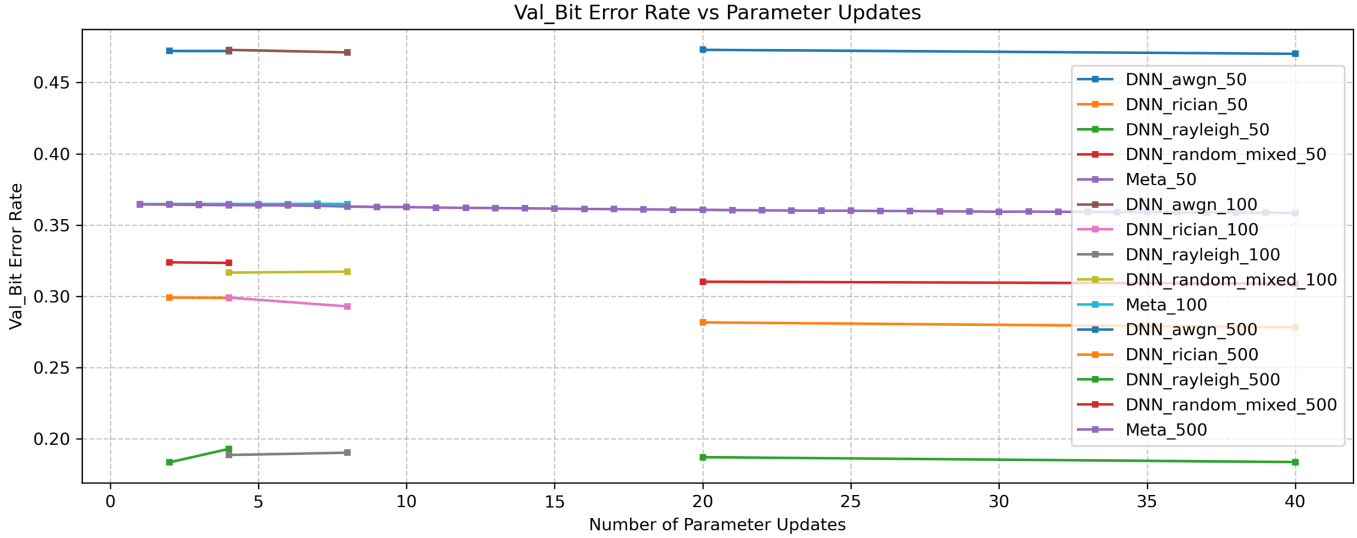
Train on 3 of them, take rest as target channel.

rician rayleigh awgn 3GPP

50 100 150 200 250

fine tune training sample.

With 2400 update, add 50 sample set into validation phase.



Meta_DNN perform worse than rician, random_mixed and rayleigh. generally.

2025/3/15

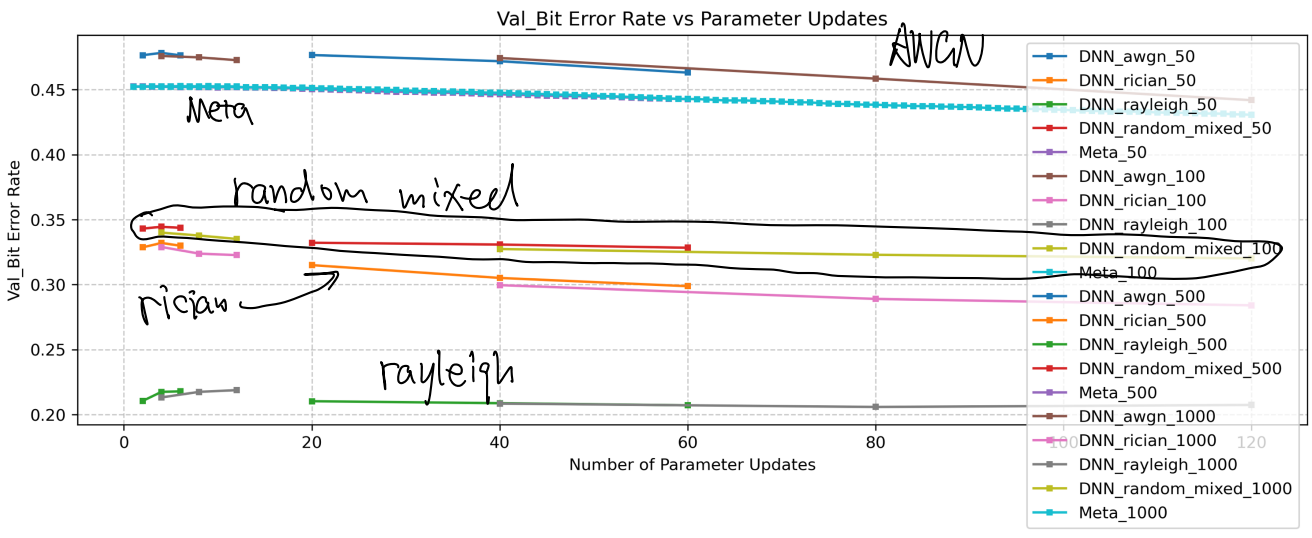
Review the "On First-order Meta Learning Algorithms"

try

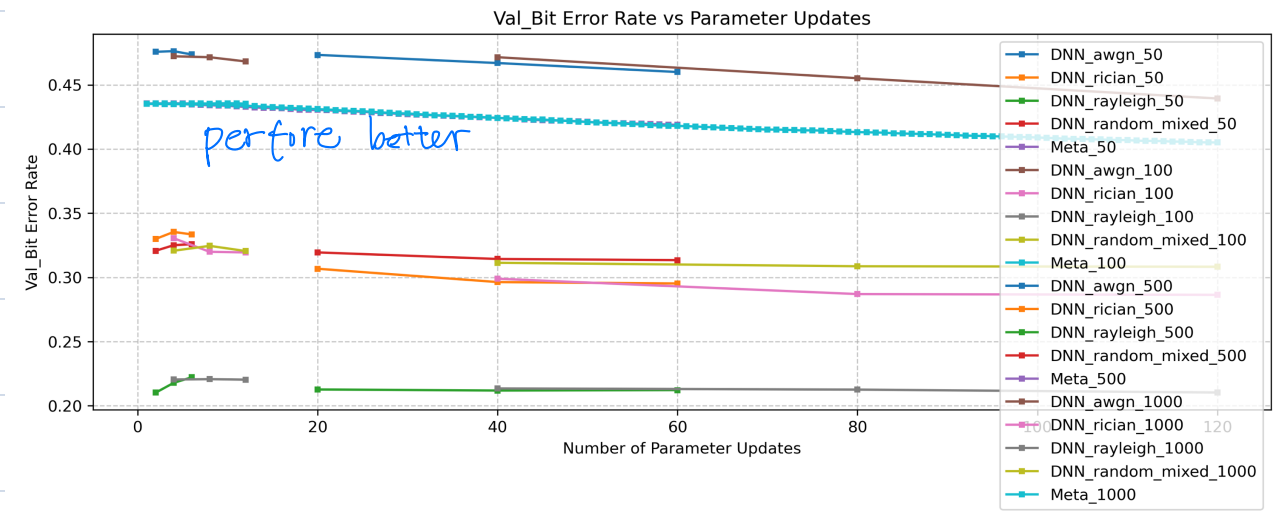
1. apply ~~linear annealing on Meta-update~~ cosine Restart applied
2. Adjust inner step from {1, 2, 5, 8, 10, 15, 20}
3. Mini batch size => {10, 20, 32}

task step (inner step) = 8. , # of updates = 1600

batch size = 32

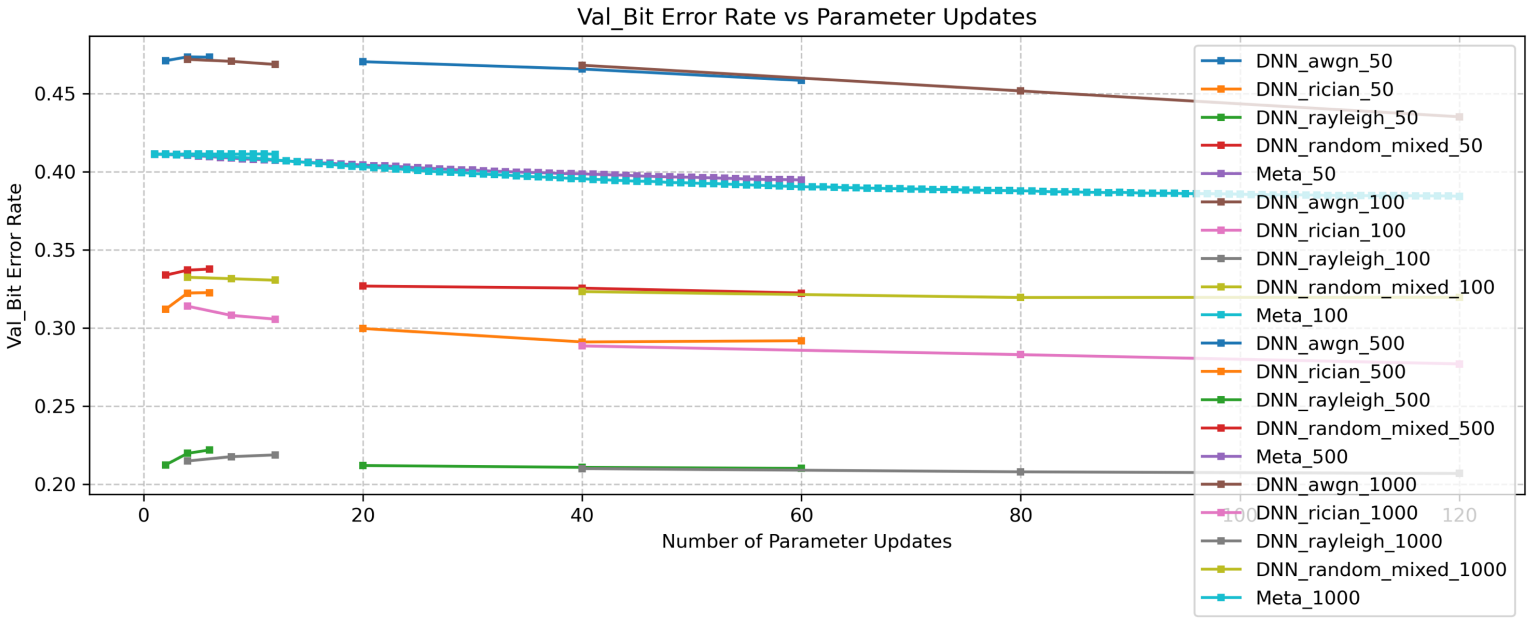



task step = 10



task

step = 15

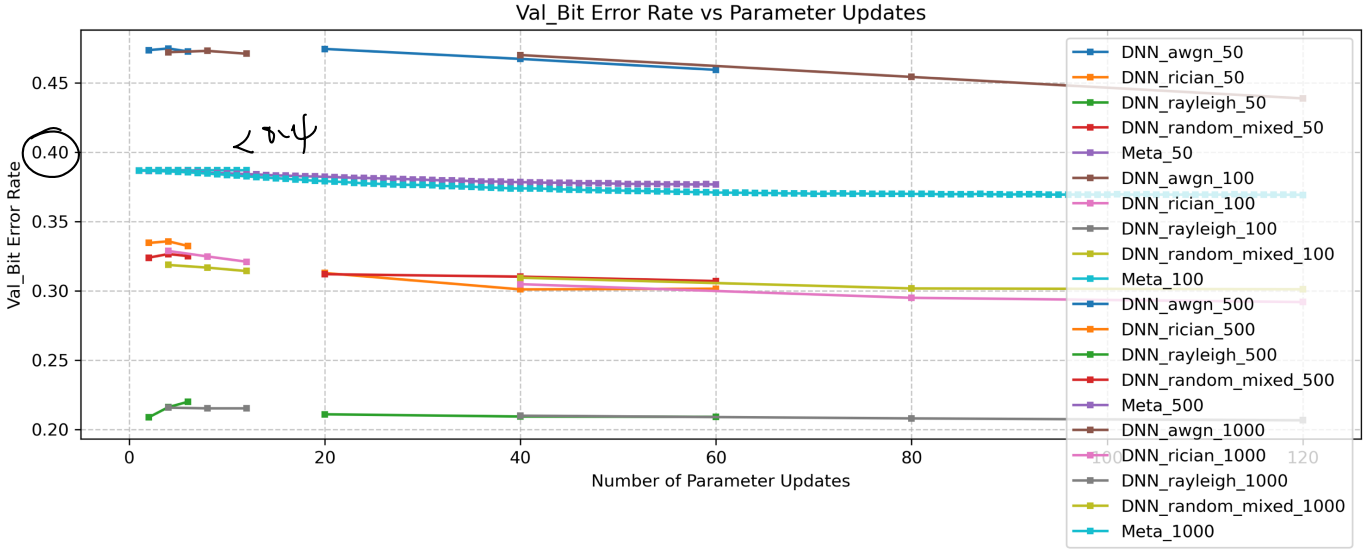



Take Adam in inner update.

task - step = 15

loss descent very slow.

task - step ≤ 50



try update time = 10000 (with 32 batch size, 10 epoch)
inner step = 50

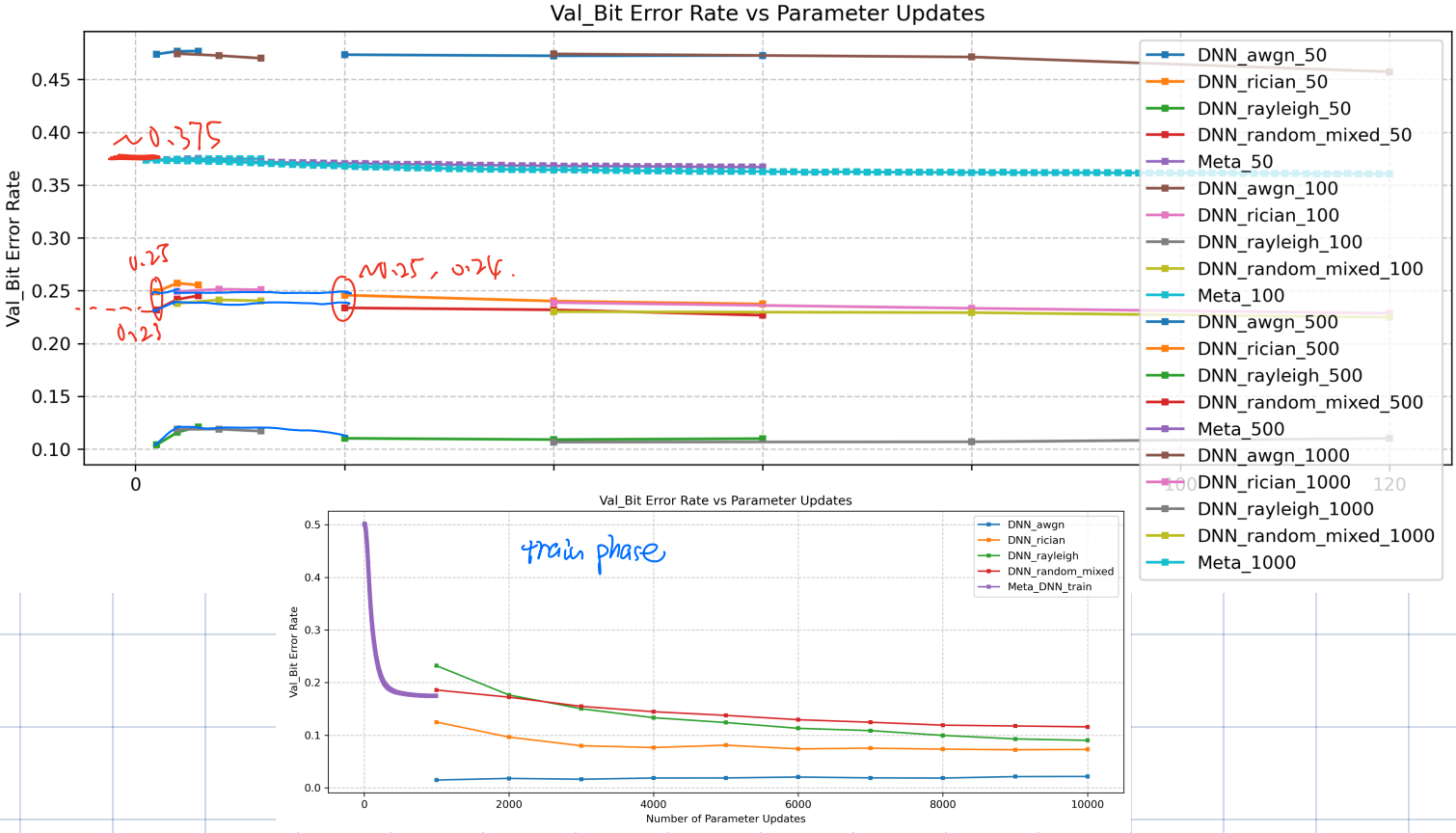
bit error rate jumping between 0.17~ when 2700~/10000

therefore aborted.

Early stop check added

patience = 10, if varience < 0.001, patience = patience − 1

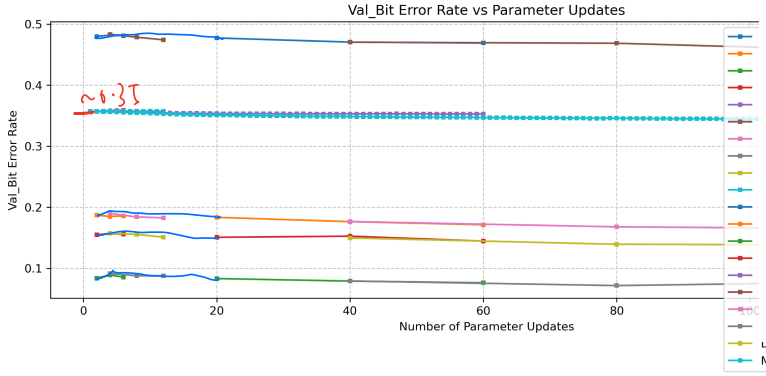
Early stop at 99b with BER = 0.17457

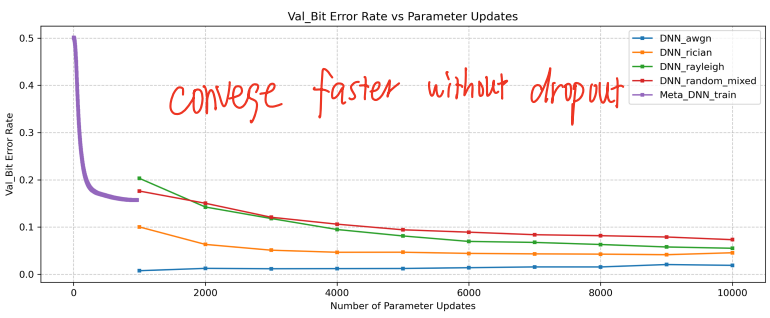






What if remove dropout layer to downgrade the generlization ability of DNN Ear stop at 963. with BER = 0.17145



2025/3/17

early stop at 2365 with BER = 0.089622

## Architecture DNN

Hidden { 256, relu
512, relu
256, relu
output = 112, sigmoid

Optimizers = Adam(0.001)
loss = MSE
Epoch = 10, batch_size = 32

## Reptile

### Meta_update

meta_lr = 0.3,

Best para tracer:

Warm-up epoch = min(50, Total iteration // 10)

$lr = lr * \frac{(epoch+1)}{Warm\text{-}up}$

Early stopping { patience = 50,
min_delta = 0.00005 }

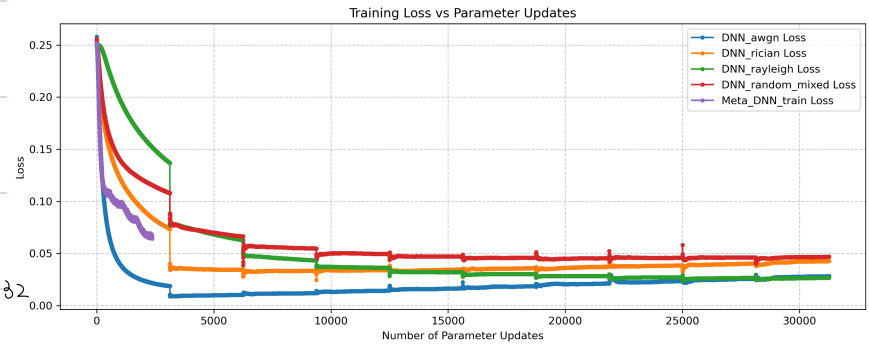
CosineDecayRestart ( first_decay_step = 500,
t_mul = 1.1 period = 1.1 × period
m_mul = 1 lr = m_mul × lr
alpha = 0.001 min_lr = alpha × initial_lr)

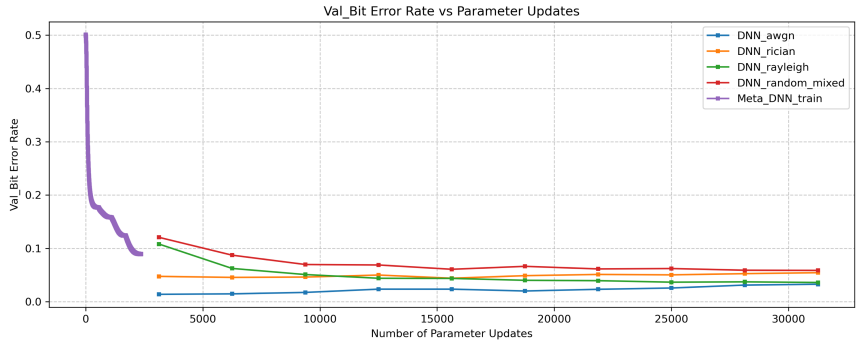
### Inner_update

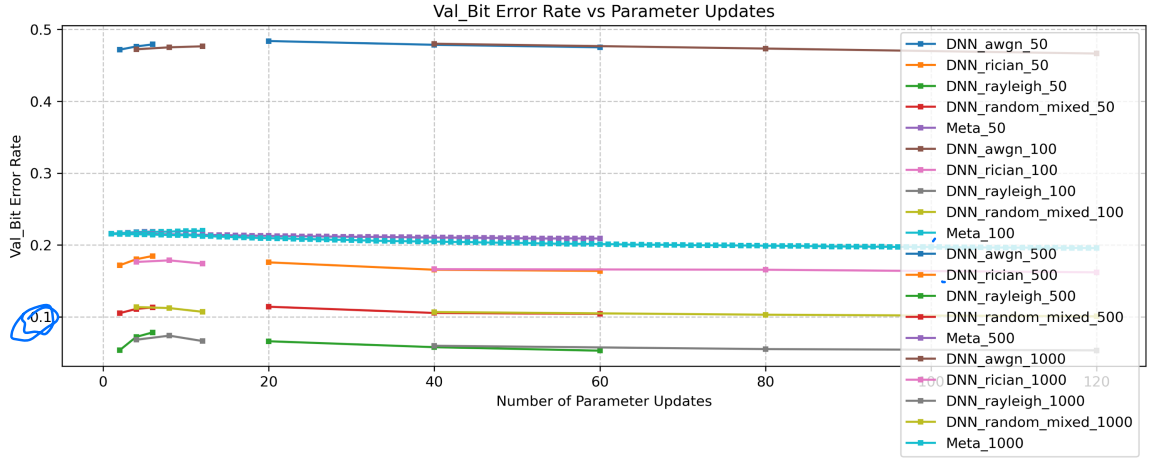
inner_lr = 0.02,

task_step = 50
mini_batch_size = 32

SGD
MSE




## Second validation

early stop at 1093 with BER = 0.156662

try remove early stop to take fully advantage of CosineDecayRestart

2025/3/19

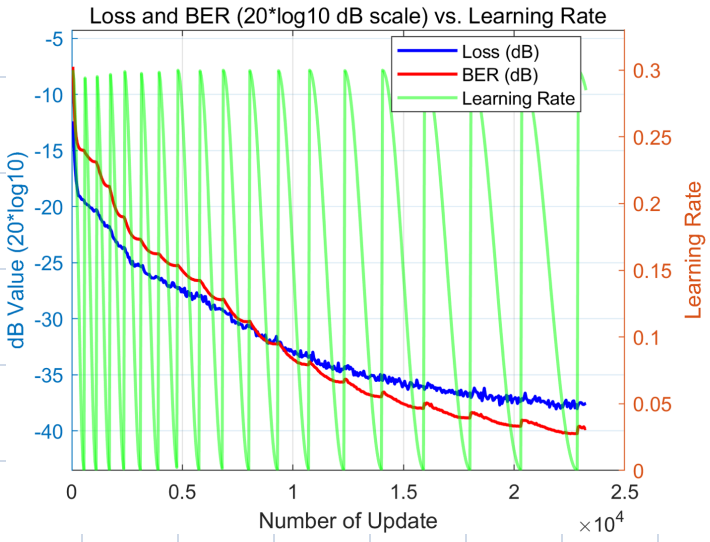
After removing early stop.

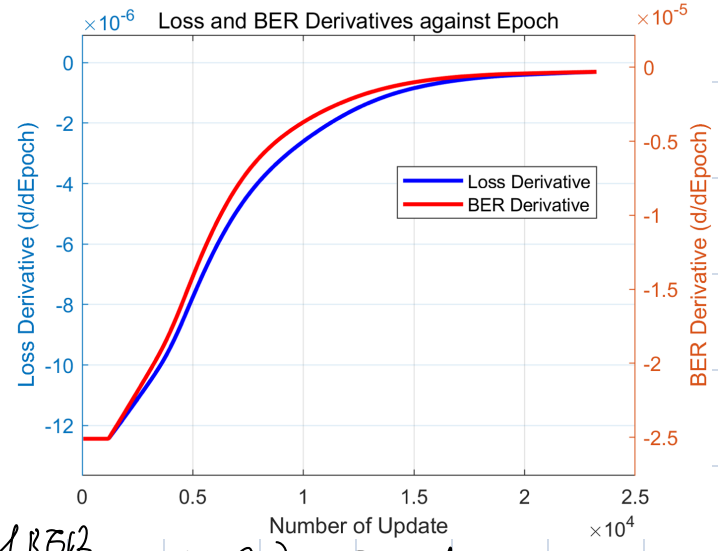
Stop by Resource exhausted at 23250/23250 with

BER = 0.01021

reveal the relationship between

Training Process Analysis



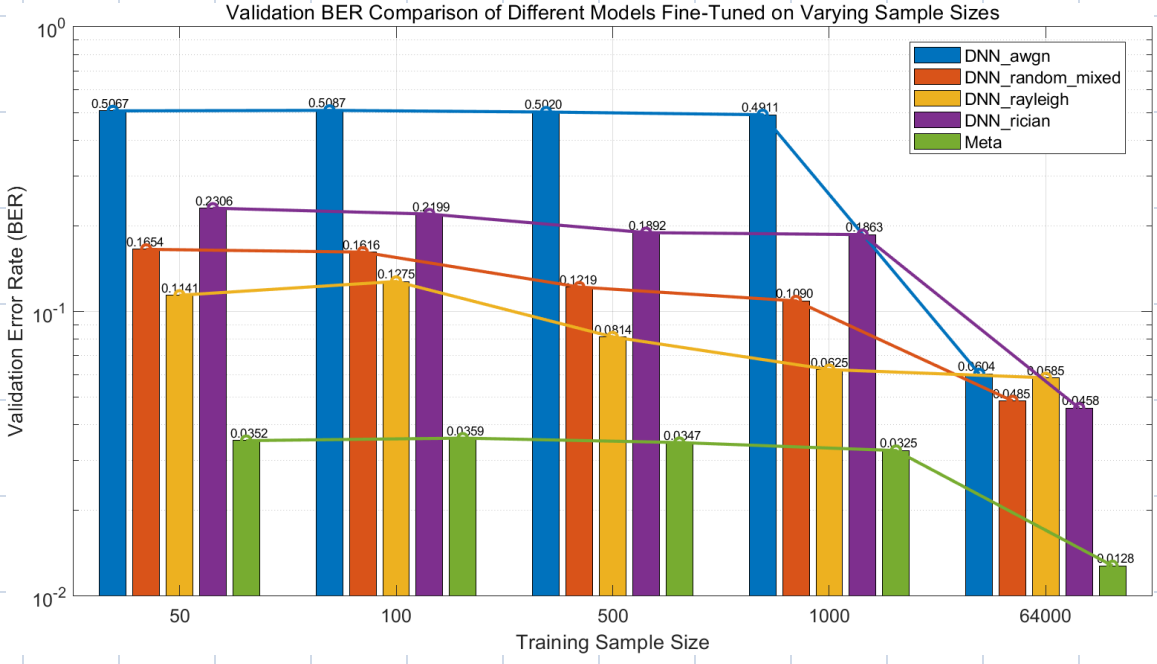




the model progress slow ($\frac{d\,BER}{d\,Epoch} \to 0$) @update $\approx 1.5 \times 10^4$

I chosed 20K updates for subsequent training.

2025/3/23

Result:



Meta-DNN outperformers than DNNs in limite data case.

(50, 100, 500)