

Deep Learning Frameworks

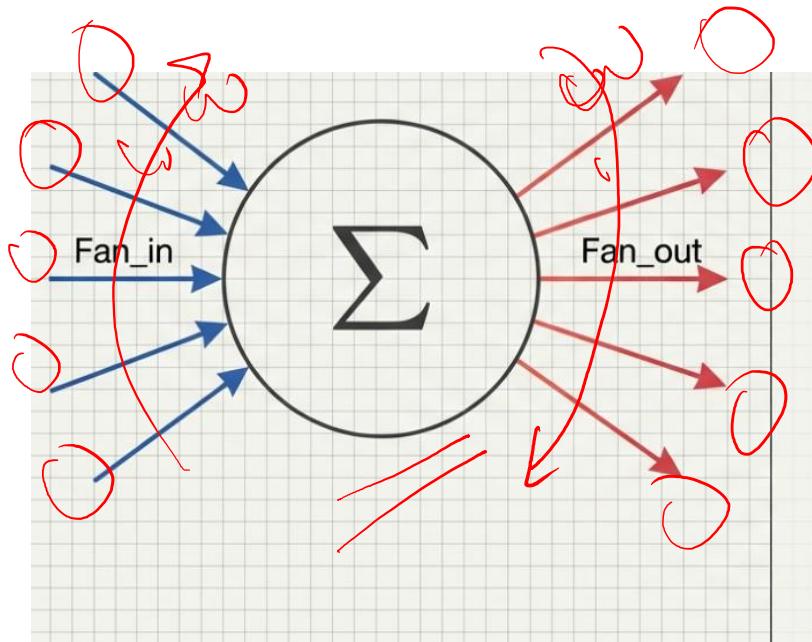
Model Initialization, Tensorflow, Keras, Dataset, Model Creation,
Training, Knowledge Distillation

<https://tinyurl.com/dlframeworks>

<https://github.com/sakharamg/DeepLearningFrameworks>

A Good Start

Linear Layers

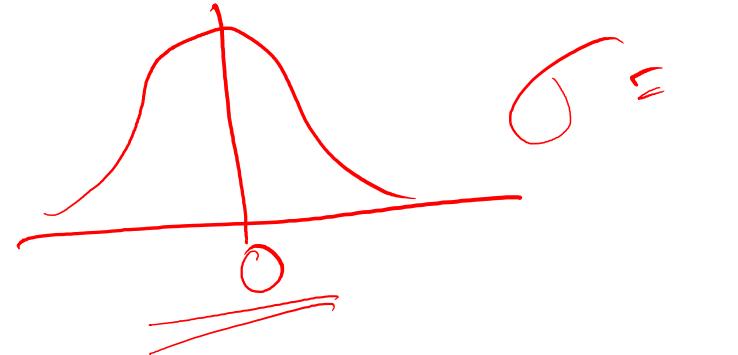


$$y_j = \sum_{i=1}^{\text{fan_in}} W_{ji}x_i$$

Consider a single layer (ignoring bias). The output y is the weighted sum of inputs x . As we sum more inputs (higher fan_in), the natural tendency is for the output variance to grow.

- **Fan_in:** The number of input connections coming *into* the neuron.
- **Fan_out:** The number of output connections going *out* to the next layer.

$$\text{Var}(y) \approx \text{fan_in} \cdot \text{Var}(W) \cdot \text{Var}(x)$$

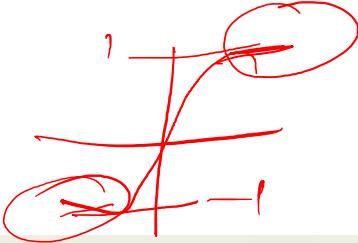


Derivation Logic:

1. Assuming inputs and weights are independent with mean 0.
2. If `fan_in` is large, the output variance explodes.
3. To keep Output Variance \approx Input Variance, we must force:

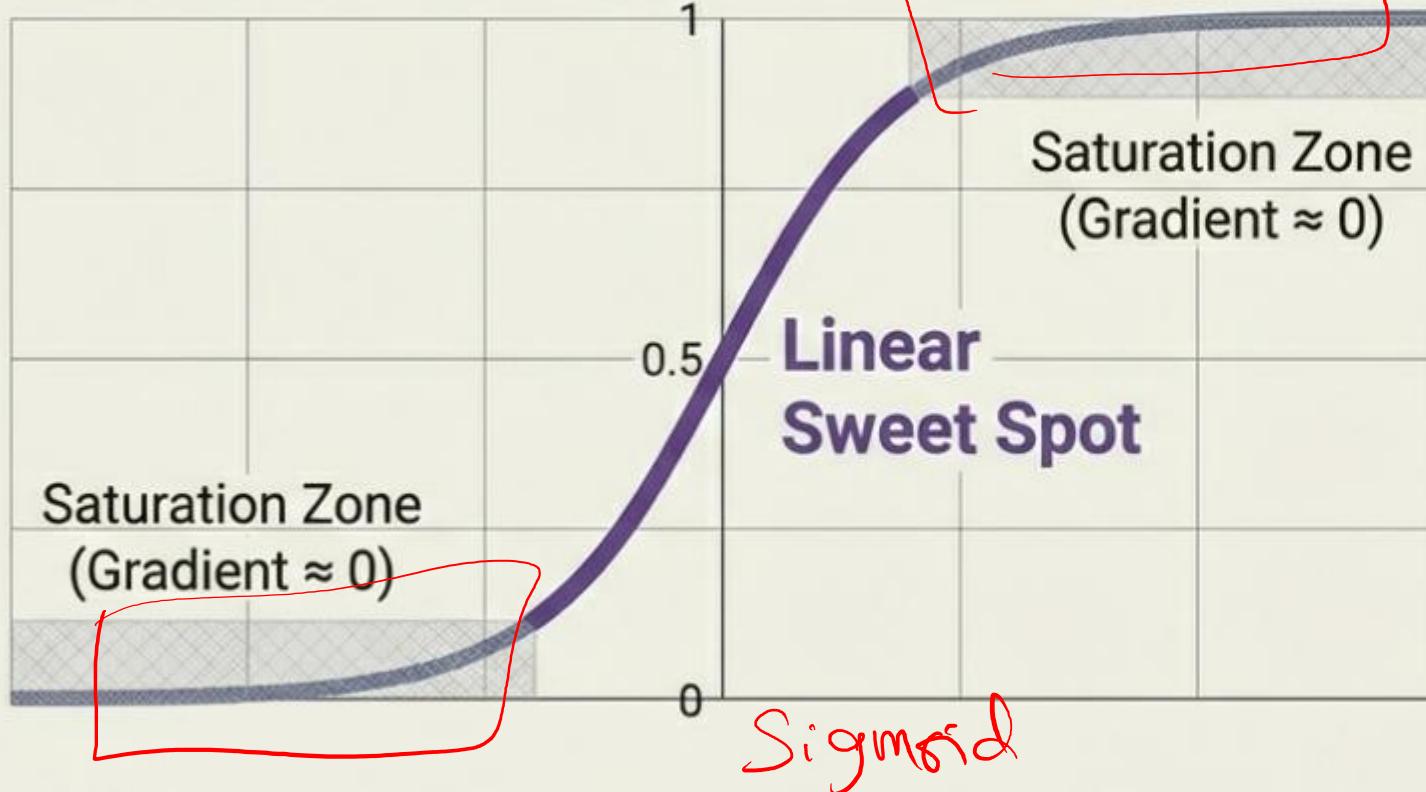
$$\text{Var}(W) \approx \frac{1}{\text{fan_in}}$$

Sigmoid and Tanh



`conv1 = torch.nn.Conv2d(...)`
`torch.nn.init.xavier_uniform(conv1.weight)`

Solution: Xavier / Glorot Initialization



The Trap: These functions saturate at the tails. If the signal is too large, gradients vanish. We must keep the signal in the “linear sweet spot”.

≈ 0

$$\text{Var}(W) = \frac{2}{\text{fan_in} + \text{fan_out}}$$

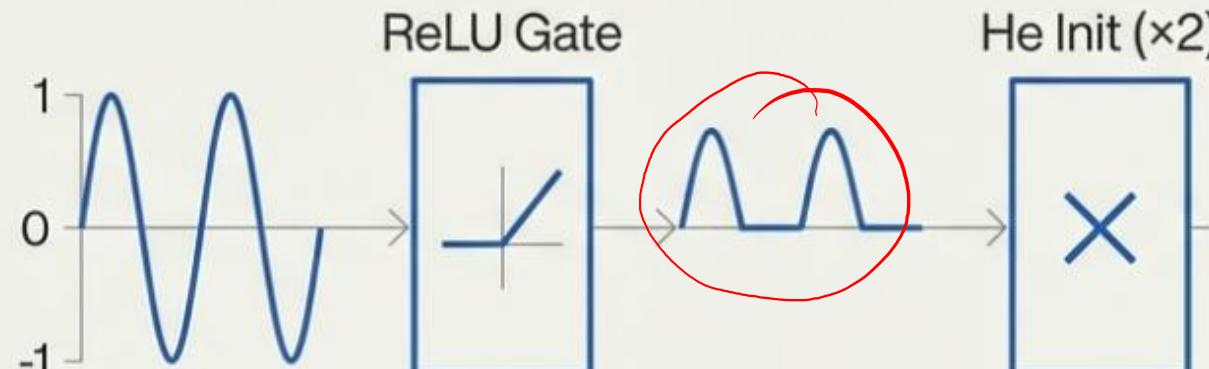
The Harmonic Mean. Since we cannot favor fan_in or fan_out, **Xavier** takes the average to maintain a compromise between forward and backward stability.

$$N(0, \frac{2}{\text{fan_in} + \text{fan_out}})$$

ReLU

$$-\infty \rightarrow \infty \Rightarrow 0 \rightarrow \infty$$

Solution: He / Kaiming Initialization



Signal

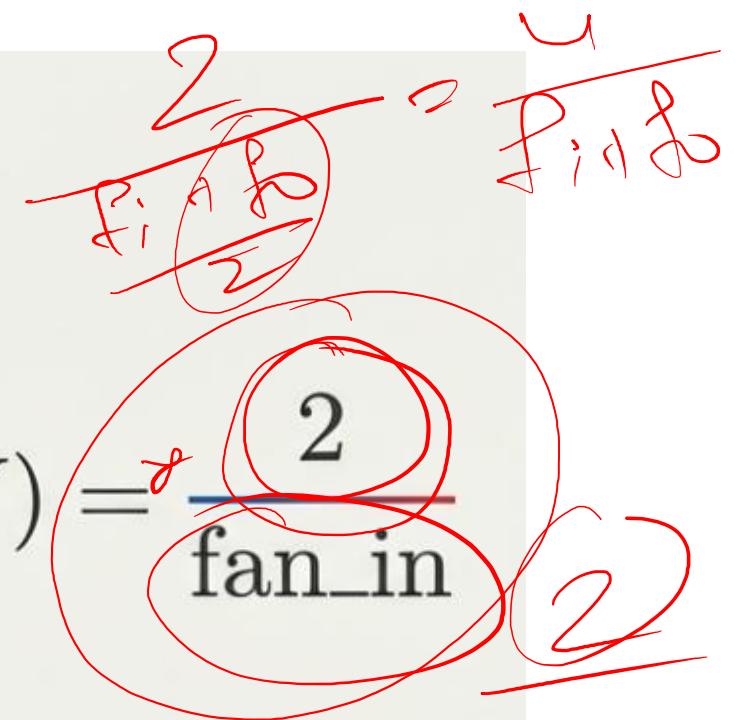
Variance
Halved
($\approx 0.5x$)

He Init
($\times 2$)

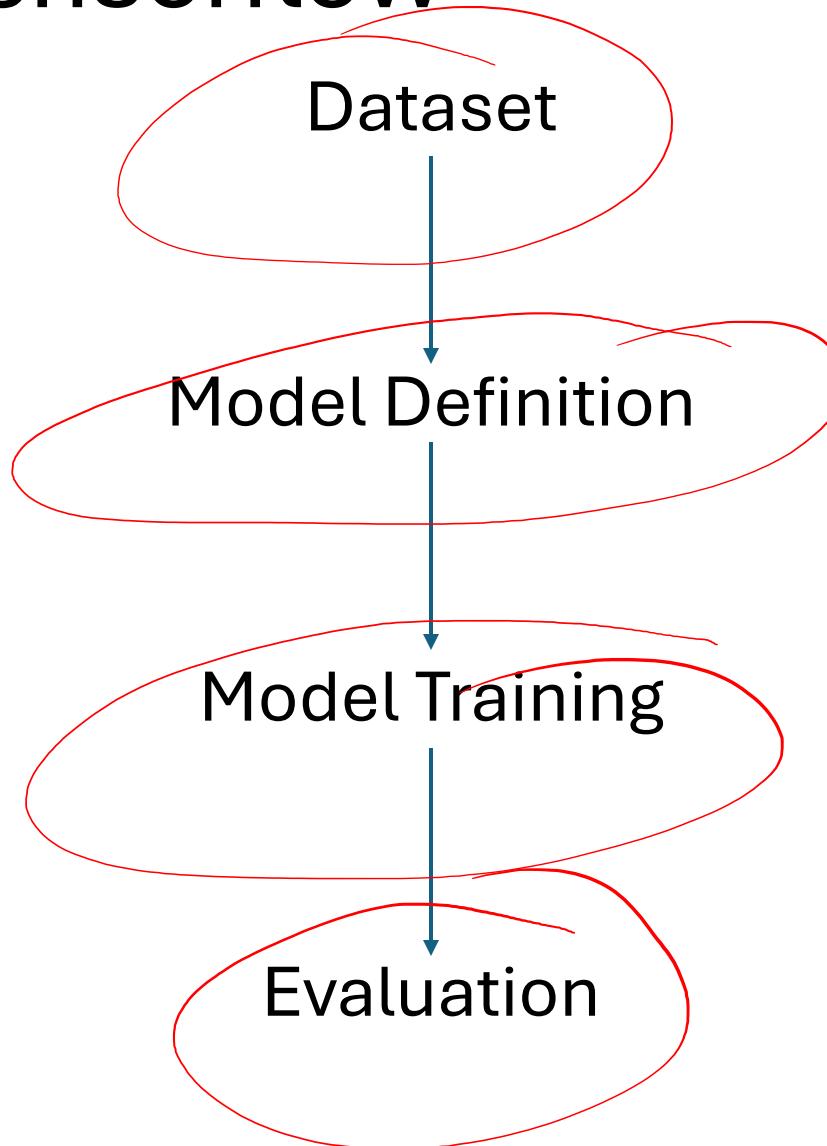
$$\text{Var}(W)$$

$$\text{Var}(W) = \frac{2}{\text{fan_in}}$$

The Trap: ReLU kills half the signal. If the network destroys half the variance, the initialization must double the variance to compensate. This preserves the energy of the signal through deep layers.

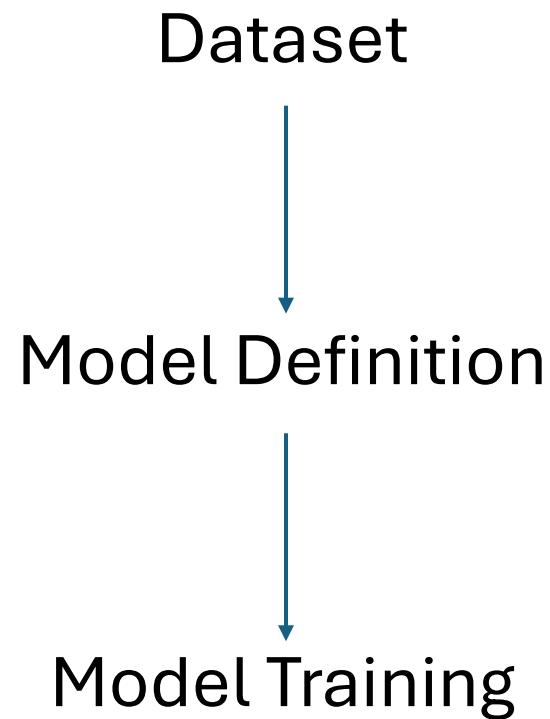


PyTorch to Tensorflow



PyTorch to Tensorflow

- from torch.utils.data import Dataset, DataLoader
- import torchvision.transforms as T
- import torch.nn as nn
- import torch.nn.functional as F
- torch.optim
- Training Loop (fixed skeleton)



import tensorflow.keras as keras

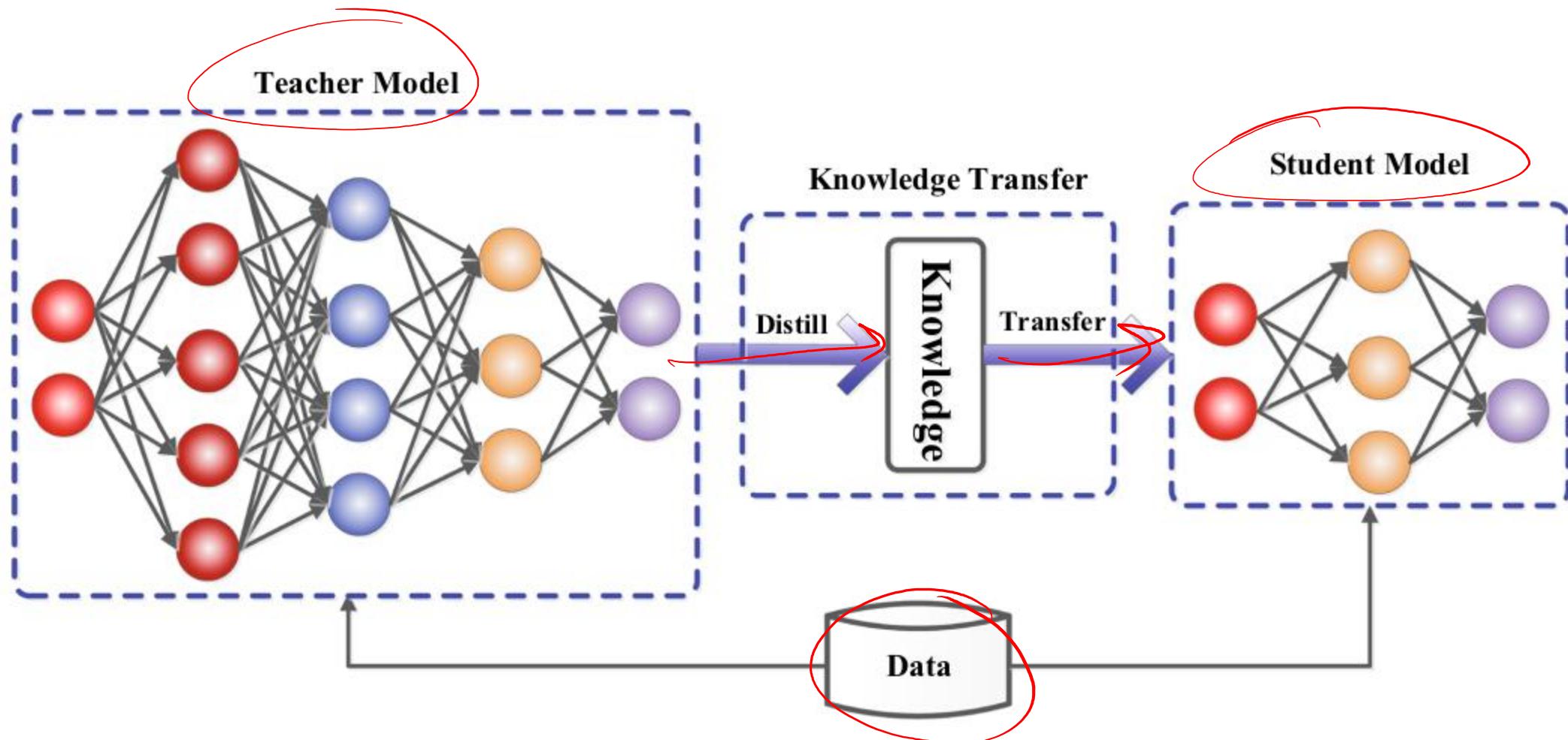
- tf.keras.optimizers
- model.compile
- model.fit

Lab: Fully Connected Network for predicting Fashion Item (FashionMNIST)

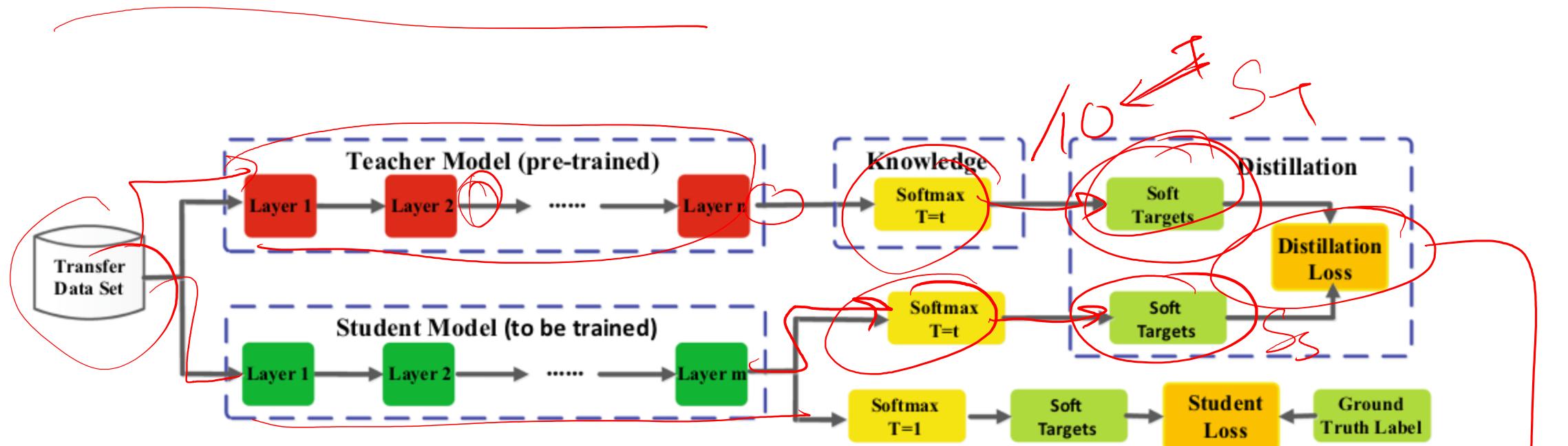
PT

TF

Knowledge Distillation



Response-based KD



Student Loss : Cross Entropy Loss

$$L_{CE} = - \sum_i y_i \log p(z_i^s, T = 1)$$

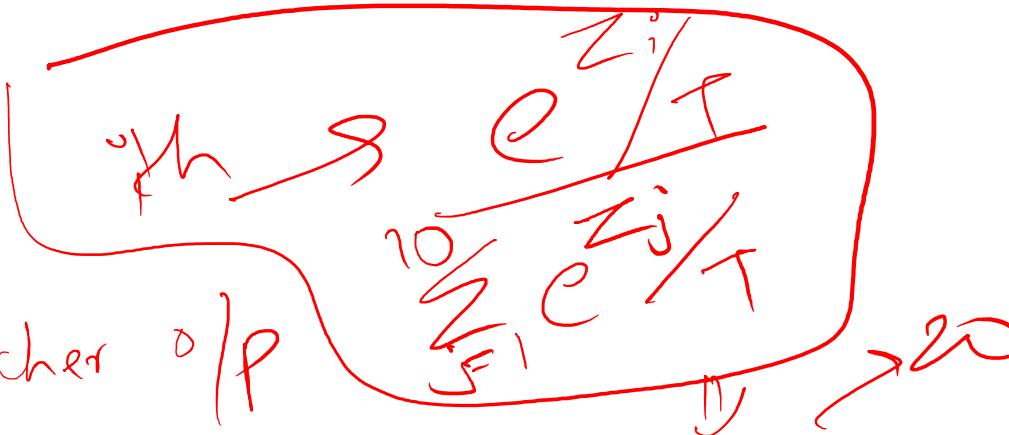
Distillation Loss : KL Divergence Loss

$$L_{distill} = \sum_i p(z_i^t, T) \ln \frac{p(z_i^t, T)}{p(z_i^s, T)}$$

$$\ln(N) + \ln(0.3)$$

0.3

DIS H
CE
OR



Teacher O/P

Output of model(T=1)

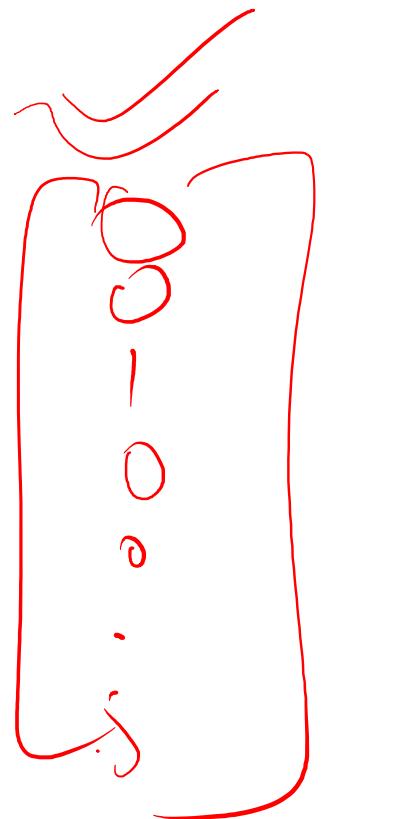
5.9049452e-14
3.8963577e-10
1.0000000e+00
1.5487779e-11
1.1989066e-22
7.1172944e-18
2.3661837e-17
2.5463203e-16
1.8852514e-15
6.4822944e-25]

Softened Output(T=10)

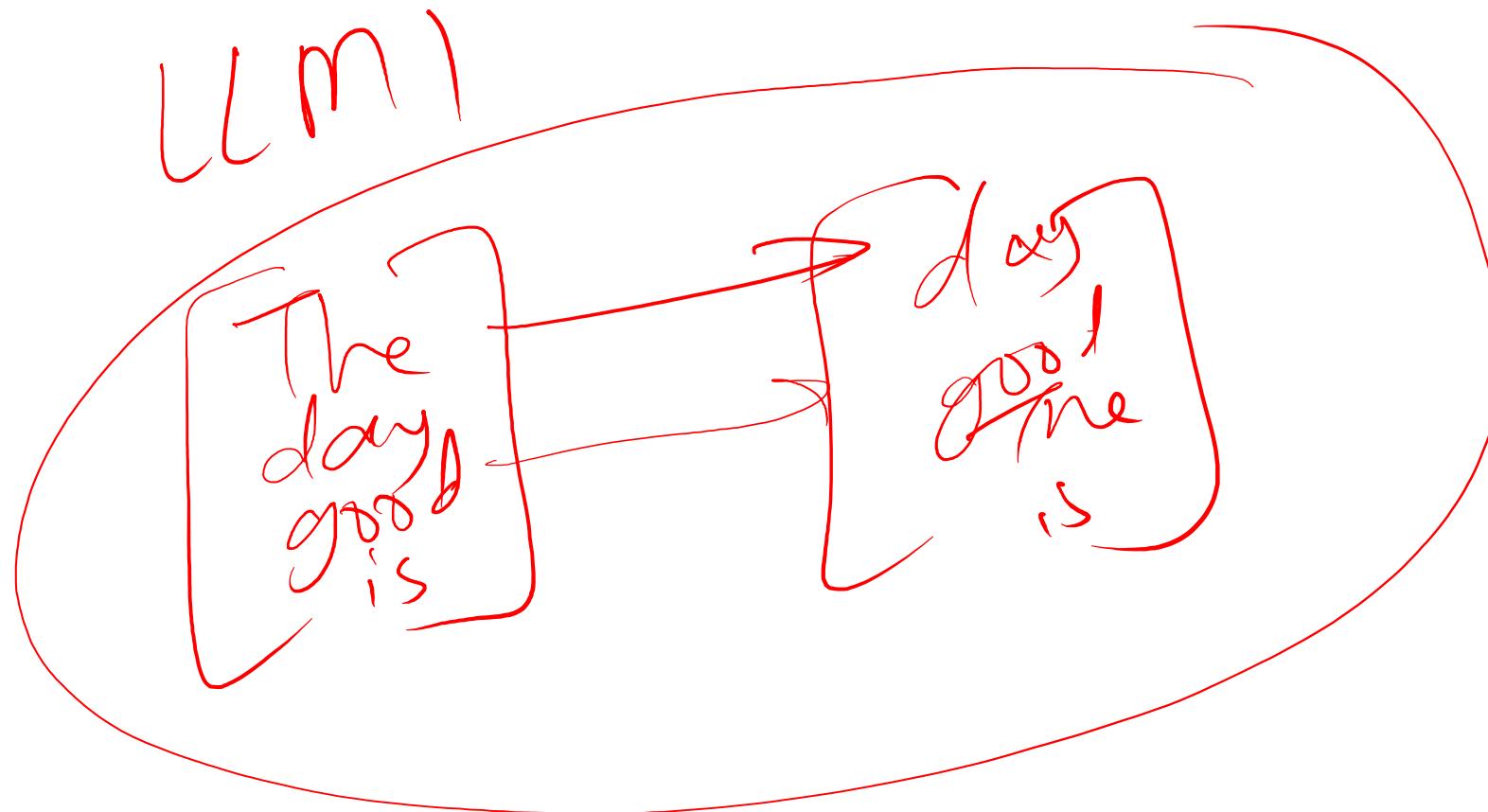
0.0350216
0.08438793
0.7365706
0.06112422
0.00473253
0.01420515
0.0160184
0.02031448
0.02481712
0.00280794

0.999 →

softmax



Lab



Thank You

Appendix