Model Compression: The OBD-SD Technique

HILLARY

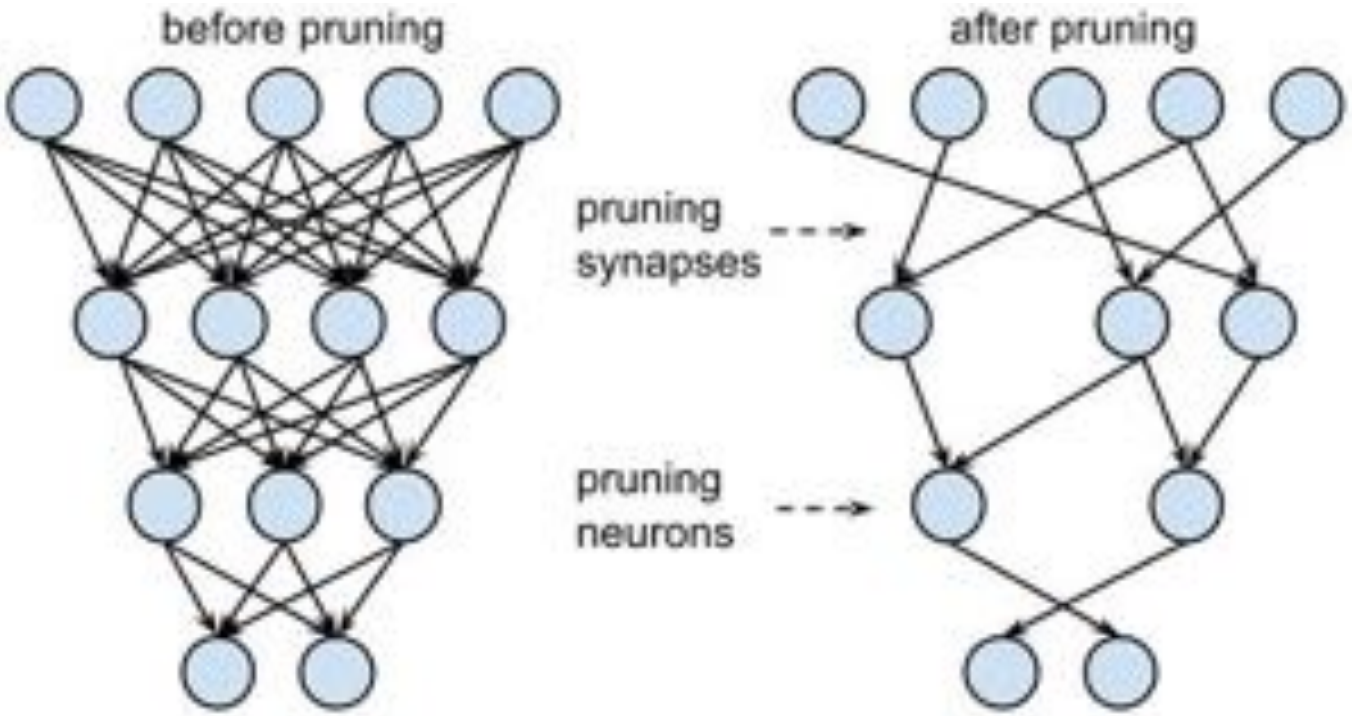
Making models smaller without reducing their accuracy

- Pruning: legit removing parameters, neurons in the model
- Quantization: clever tricks like bundling weights together or rounding them off to save on memory-per-parameter.
  - Other weird stuff involving how to store large parameter matrices in memory more efficiently
- Knowledge distillation, like training a smaller model to predict the scores of the larger

Making models smaller without reducing their accuracy

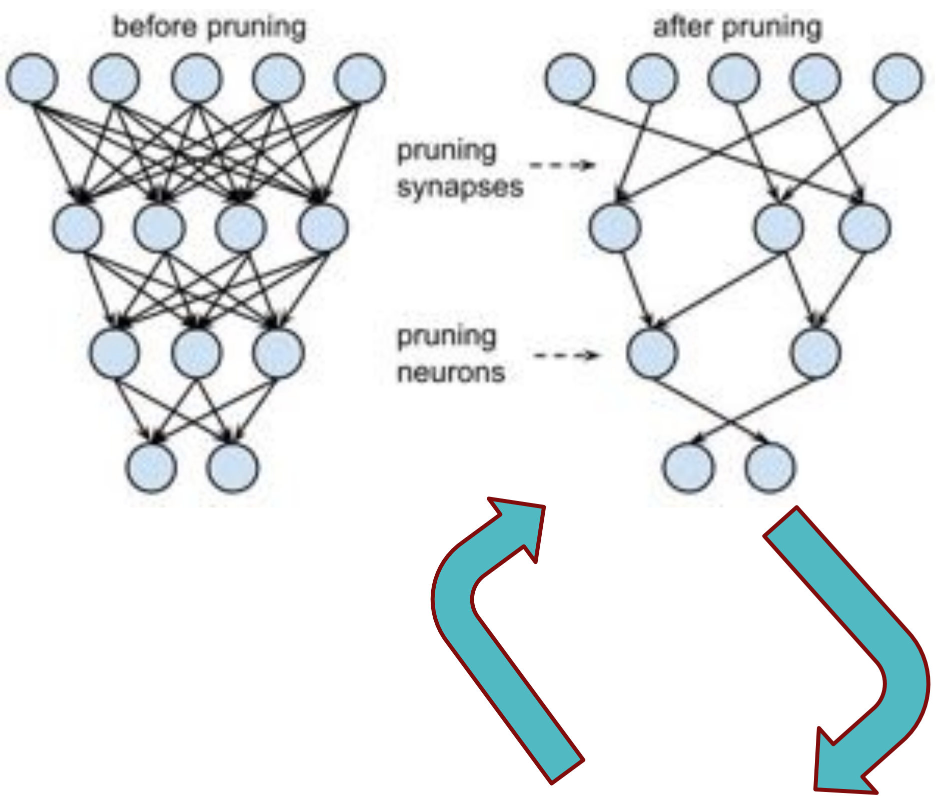
- **Pruning: legit removing parameters, neurons in the model**
- Quantization: clever tricks like bundling weights together or rounding them off to save on memory-per-parameter.
  - Other weird stuff involving how to store large parameter matrices in memory more efficiently
- Knowledge distillation, like training a smaller model to predict the scores of the larger

Pruning



before pruning
after pruning
pruning synapses
pruning neurons

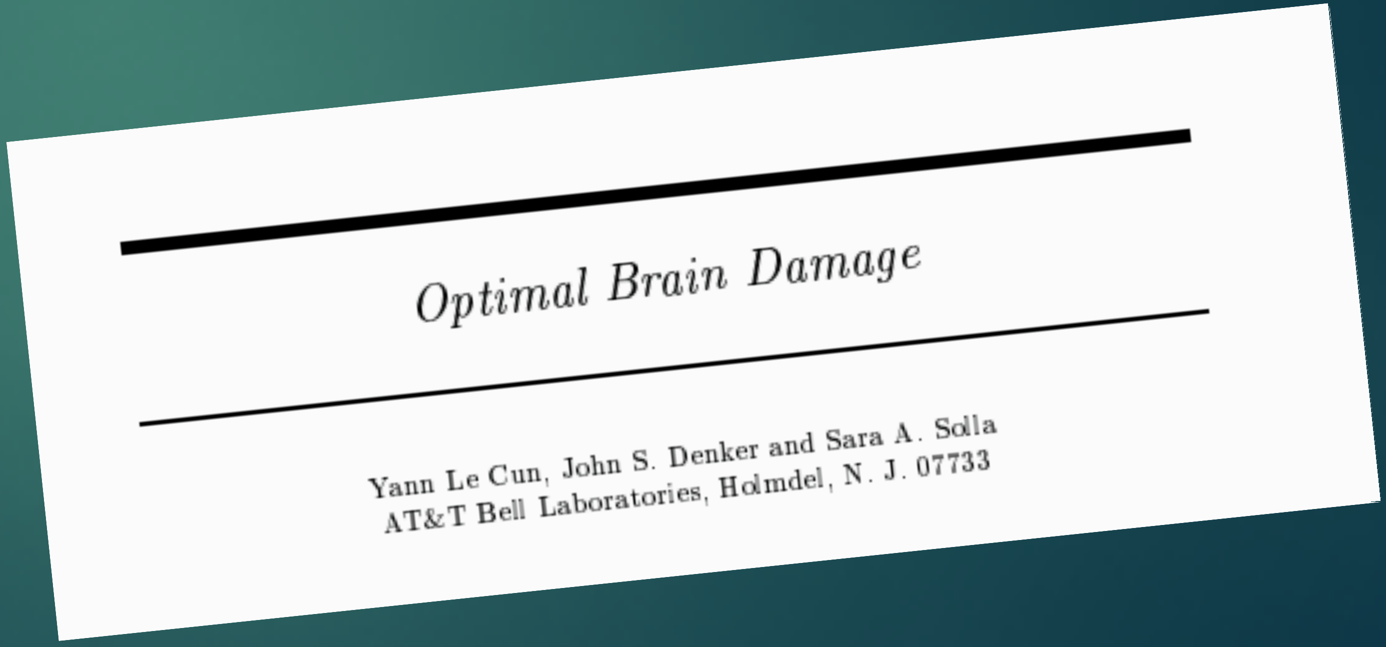
Pruning



Fine-tune

How to choose what to prune?

- Can we just… directly estimate how loss will be affected when a parameter is set to 0 through the power of magical math?


Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

$L :=$ LOSS FUNCTION

GOAL: Expected Δ in L when a param p_i is set to $\underline{0}$, for any $i \in 1 \dots P$

$$= \underbrace{\mathbb{E}(L(p_i = p)) - \mathbb{E}(L(p_i = 0))}_{\text{ARGMAX over all } i \text{ to find params to delete!}}$$

L := LOSS FUNCTION

GOAL: Expected Δ in L when a param p_i is set to $\underline{0}$, for any $i \in 1 \ldots P$

$$= \underbrace{\mathbb{E}(L(p_i = p)) - \mathbb{E}(L(p_i = 0))}_{\text{ARGMAX over all } i \text{ to find params to delete!}}$$

e.g.: $10 - 10 = 0$

$10 - 15 = -5$

$10 - 8 = +2$

- Can we calculate this?

$L :=$ LOSS FUNCTION

GOAL: Expected Δ in L when a param p_i is set to $\underline{0}$, for any $i \in 1 \ldots P$

$$= \underbrace{\mathbb{E}(L(p_i = p)) - \mathbb{E}(L(p_i = 0))}_{\text{ARGMAX over all } i \text{ to find params to delete!}}$$

e.g.:

$10 - 10 = 0$

$10 - 15 = -5$

$10 - 8 = +2$

TAYLOR SERIES

$\mathbb{E}$: average over many samples

→ GOAL: $L(p_i = p) - L(p_i = 0)$

EASY! HARD!

$$= \cancel{L(p_i = p)} - \left[\cancel{L(p_i = p)} + \frac{L'(p_i = p)}{1!}(0 - p)^1 + \frac{L''(p_i = p)}{2!}(0 - p)^2 + \dots \right]$$

- The first term is just the loss calculated with current weights across a bunch of test samples.
- The second term is harder: we want to be able to calculate this for al parameters p, and don't want to have to evaluate the loss a different time for every sample for every p: computationally, that's a LOT. → Use taylor series to estimate instead!

- **This can be calculated in tensorflow :D**

TAYLOR SERIES

$\mathbb{E}$: average over many samples

→ GOAL: $L(p_i = p) - L(p_i = 0)$

EASY!

HARD!

$$= \cancel{L(p_i = p)} - \left[\cancel{L(p_i = p)} + \frac{L'(p_i = p)}{1!}(0-p)^1 + \frac{L''(p_i = p)}{2!}(0-p)^2 + \ldots \right]$$

$$\approx \frac{L''(p_i = p)}{2!} \cdot p^2$$

← ARGMAX THIS OVER A SAMPLE OF OBSERVATIONS.

$$\approx \frac{L''(p_i = p) \cdot p^2}{2!}$$

- This is can be calculated in tensorflow :D

```python
with tf.GradientTape(persistent=True) as g1:
    with tf.GradientTape(persistent=True) as g2:
        prediction = self.model(inp)[0]
        loss = tf.losses.binary_crossentropy(label, prediction)
    dy_dx = g2.gradient(loss, self.model.trainable_weights)

# a second gradient can't be operated on None's so we have to temporarily remove them:
idx = [i for i in range(len(dy_dx)) if dy_dx[i] is not None]
gradients = [g1.gradient(dy_dx[i], self.model.trainable_weights[i]) for i in idx]
```

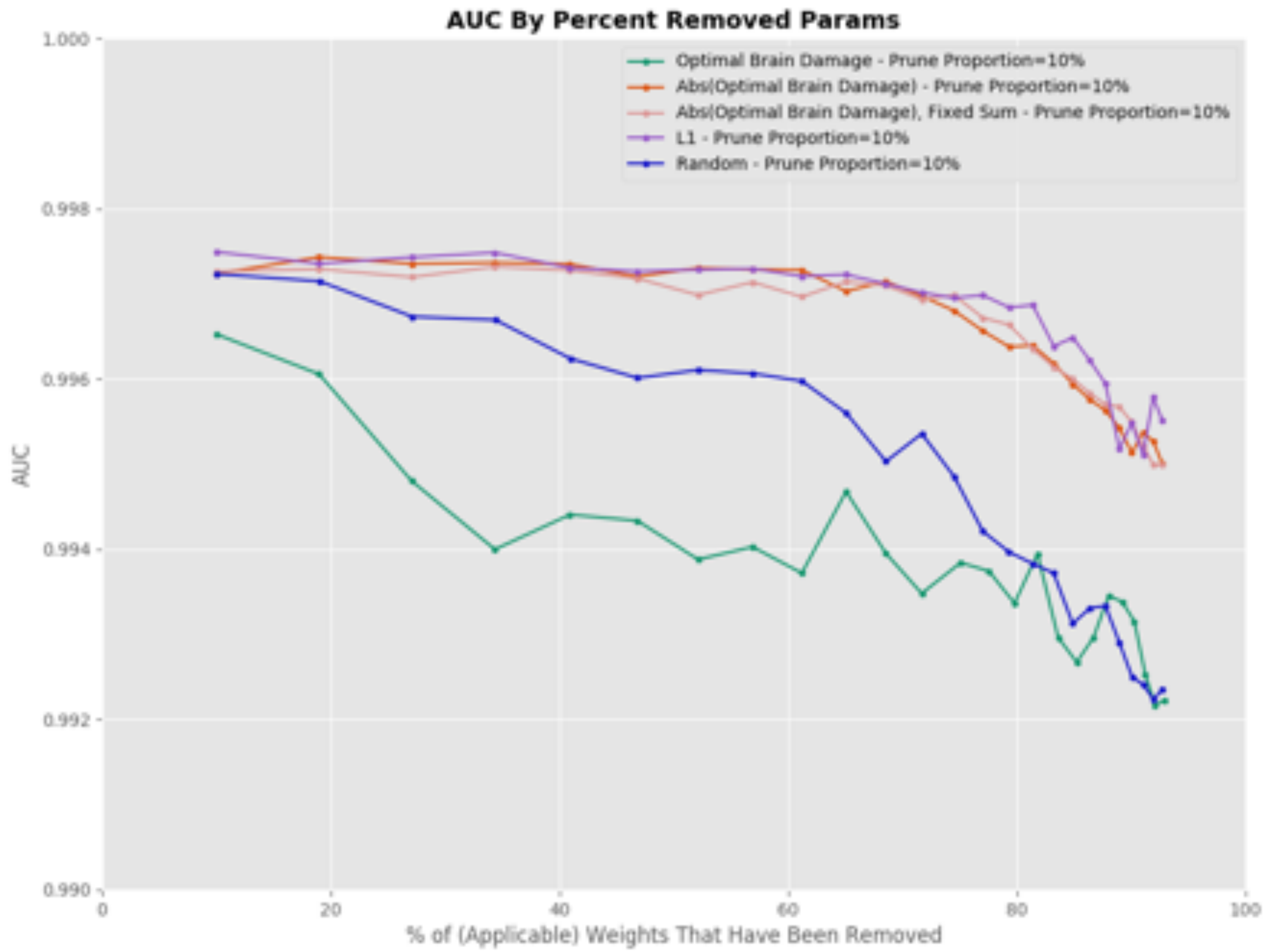
- Run this ^^ over a bunch of samples, average the results, and you have an estimate of our original goal:

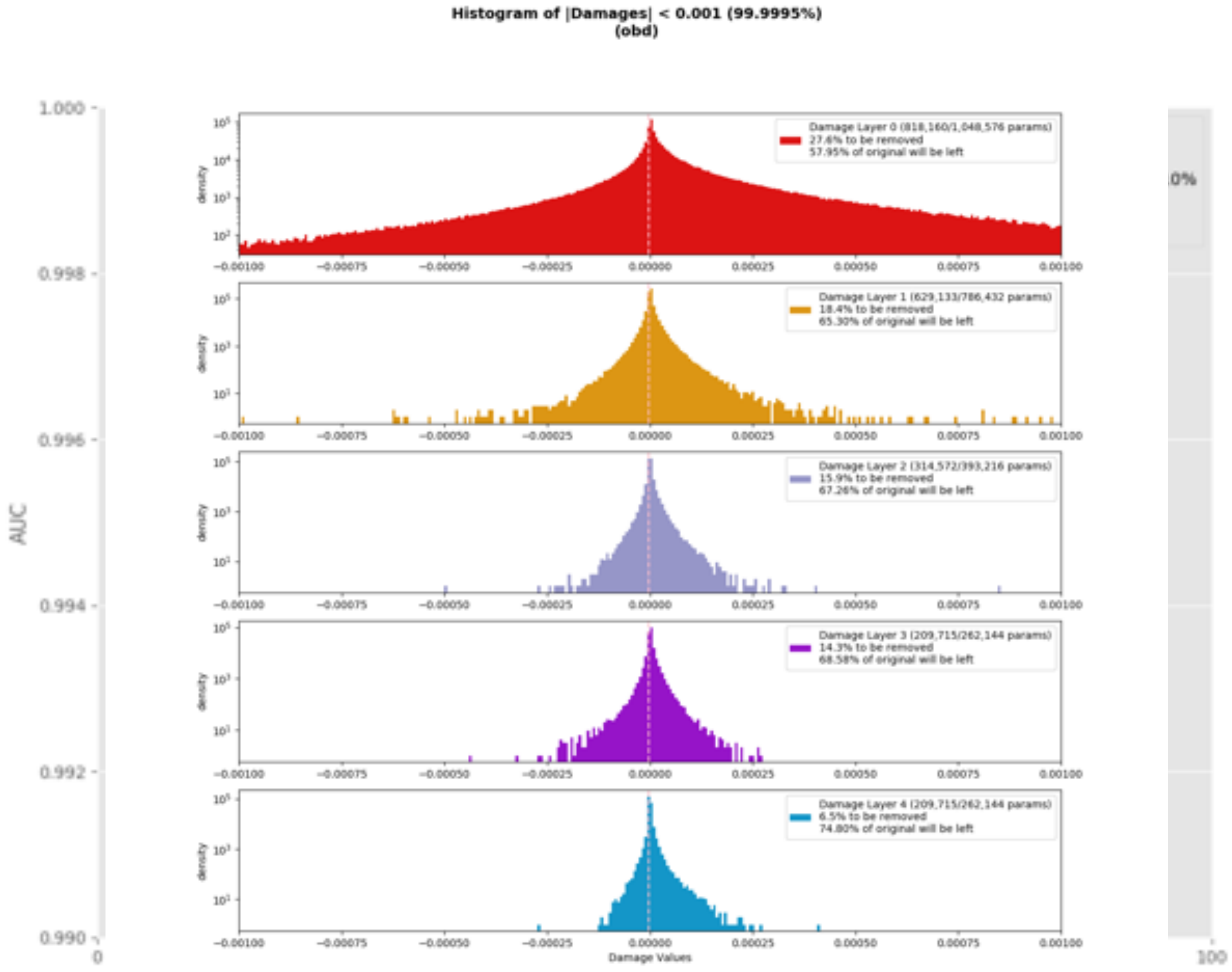
$$\mathbb{E}(L(p_i = p)) - \mathbb{E}(L(p_i = 0))$$

- → Use this to iteratively 1) remove parameters, 2) fine-tune, repeat

Optimal Brain Damage Results?!??!?!

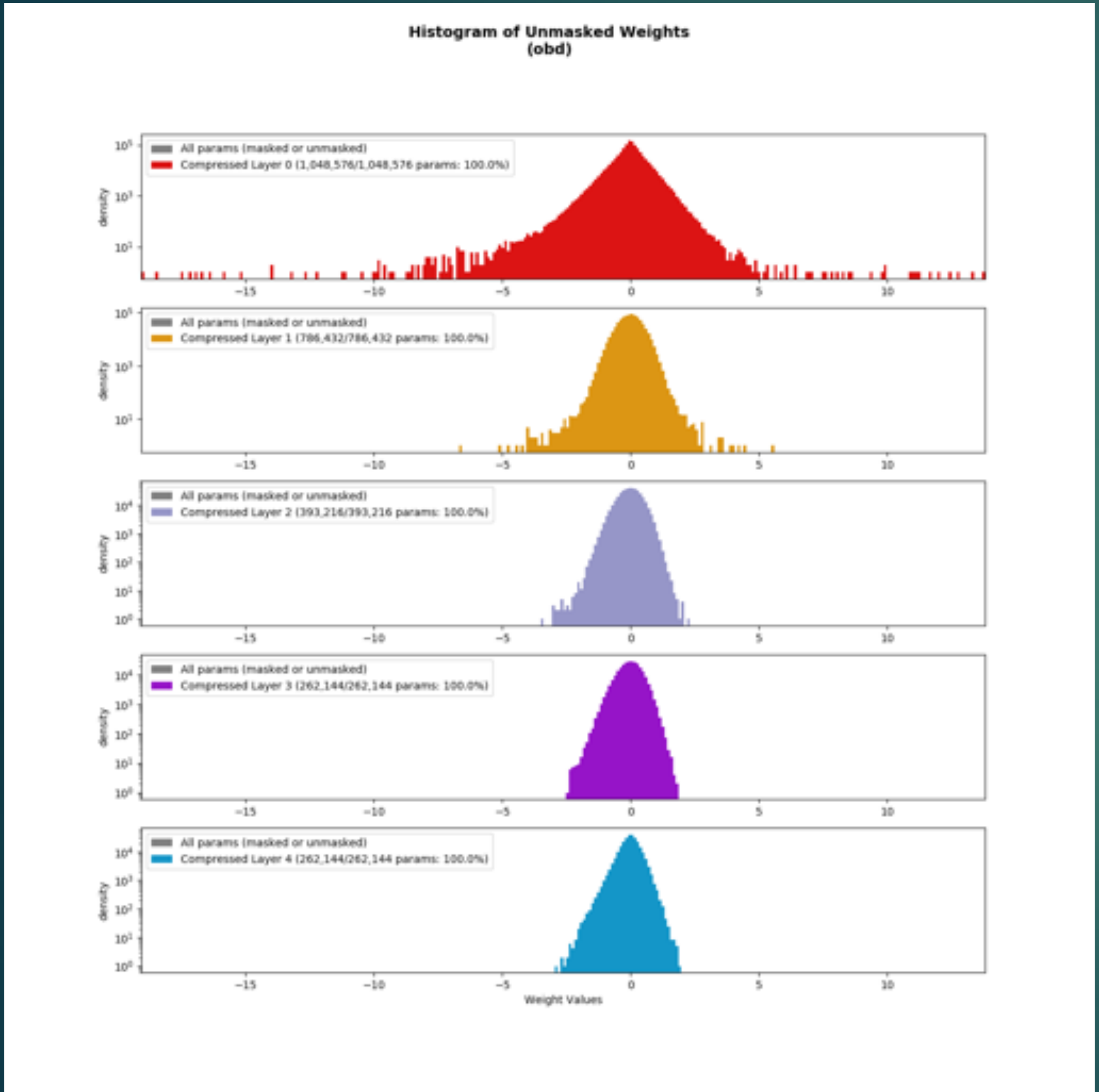
Optimal Brain Damage Results?!??!?!

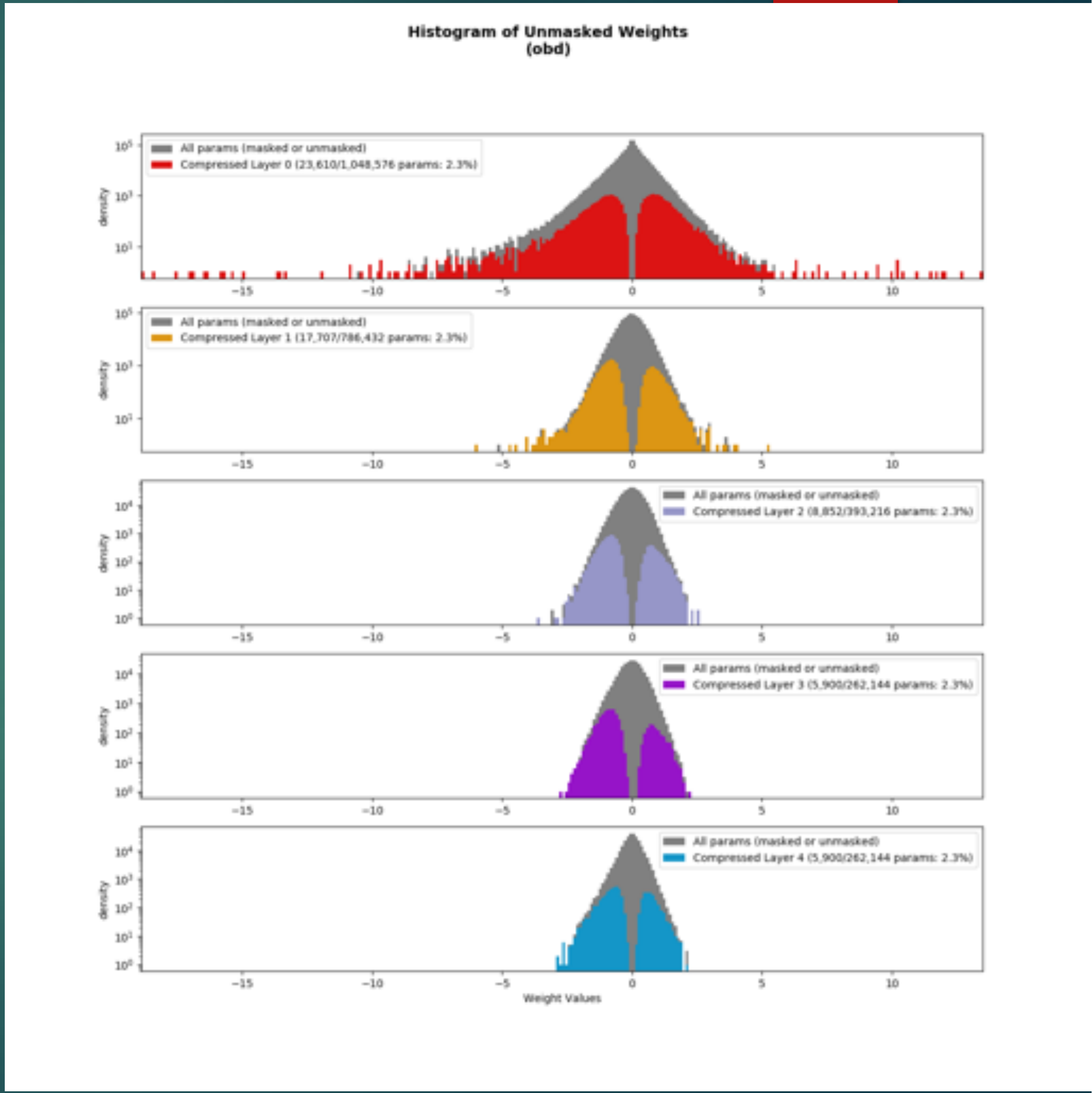

AUC By Percent Removed Params
Optimal Brain Damage - Prune Proportion=10%
Abs(Optimal Brain Damage) - Prune Proportion=10%
Abs(Optimal Brain Damage), Fixed Sum - Prune Proportion=10%
L1 - Prune Proportion=10%
Random - Prune Proportion=10%
AUC
1.000
0.998
0.996
0.994
0.992
0.990
0
20
40
60
80
100
% of (Applicable) Weights That Have Been Removed


Histogram of |Damages| < 0.001 (99.9995%)
(obd)
Damage Layer 0 (818,160/1,048,576 params)
27.6% to be removed
57.95% of original will be left
Damage Layer 1 (629,133/786,432 params)
18.4% to be removed
65.30% of original will be left
Damage Layer 2 (314,572/393,216 params)
15.9% to be removed
67.26% of original will be left
Damage Layer 3 (209,715/262,144 params)
14.3% to be removed
68.58% of original will be left
Damage Layer 4 (209,715/262,144 params)
6.5% to be removed
74.80% of original will be left
density
Damage Values
AUC

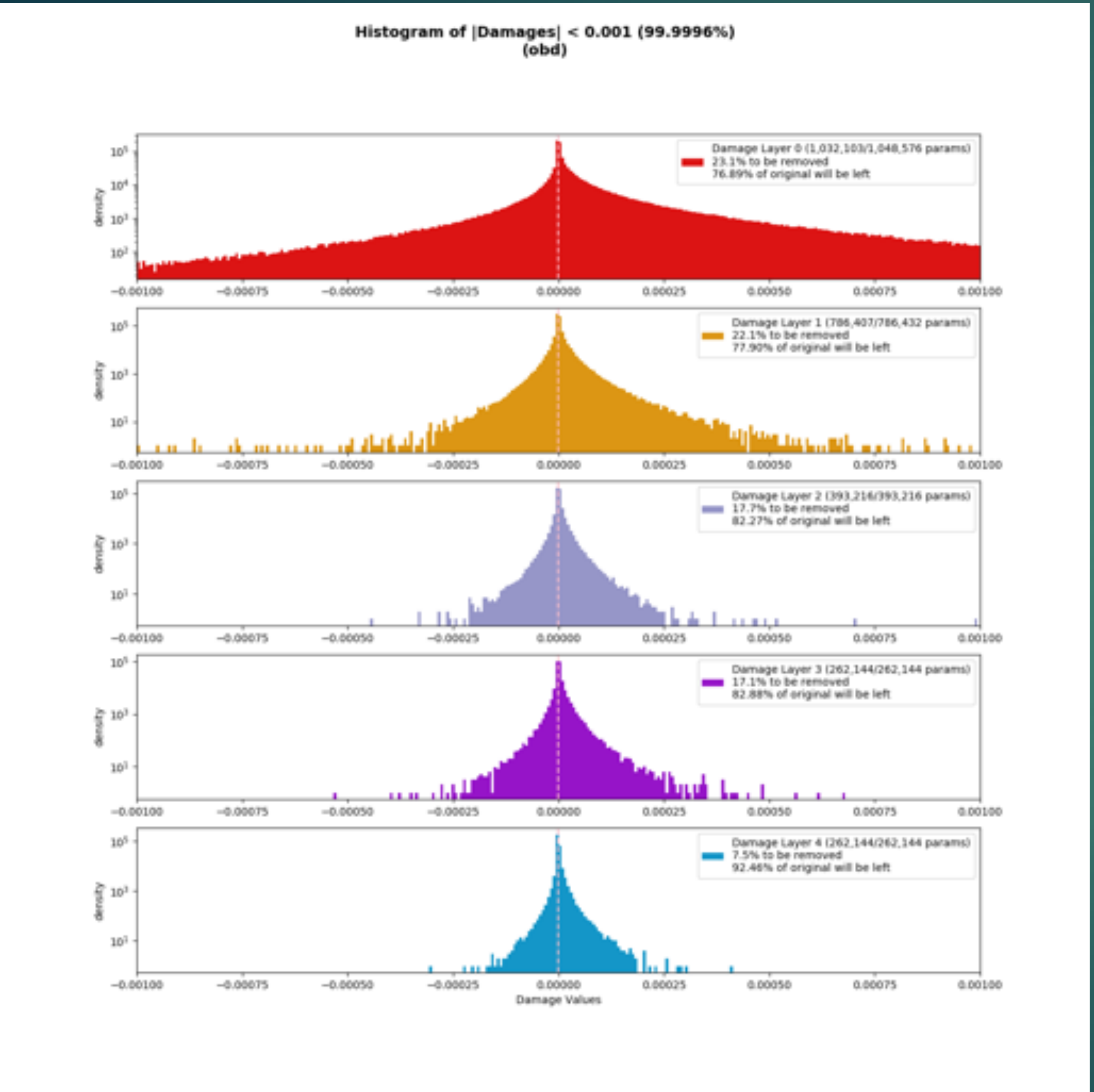
- Optimal brain damage and glorious math did WORSE THAN REMOVING RANDOM WEIGHTS!!!
- → What's going on?

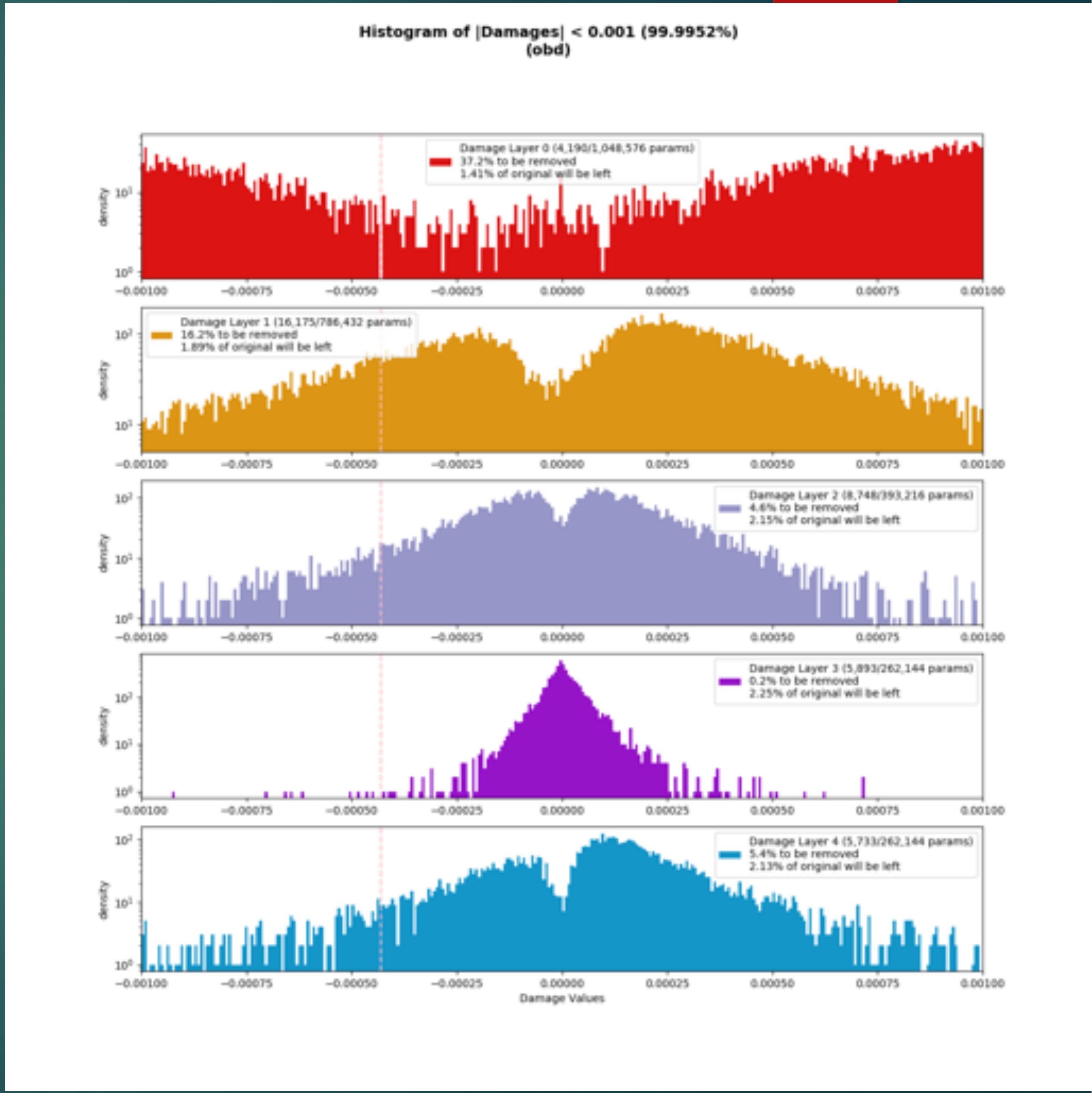
Prune round 1 … Prune round 17



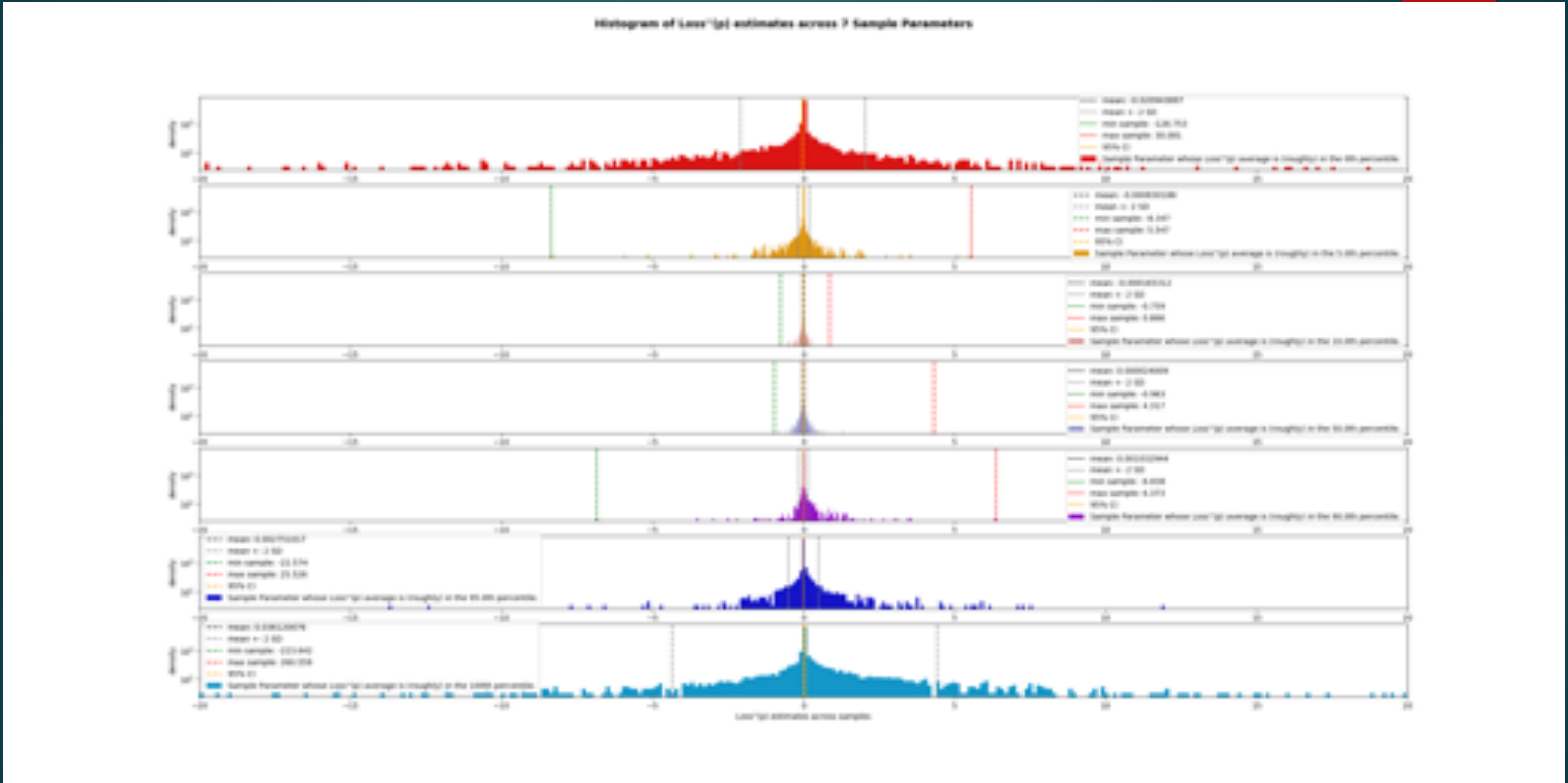


Prune round 1 … Prune round 17



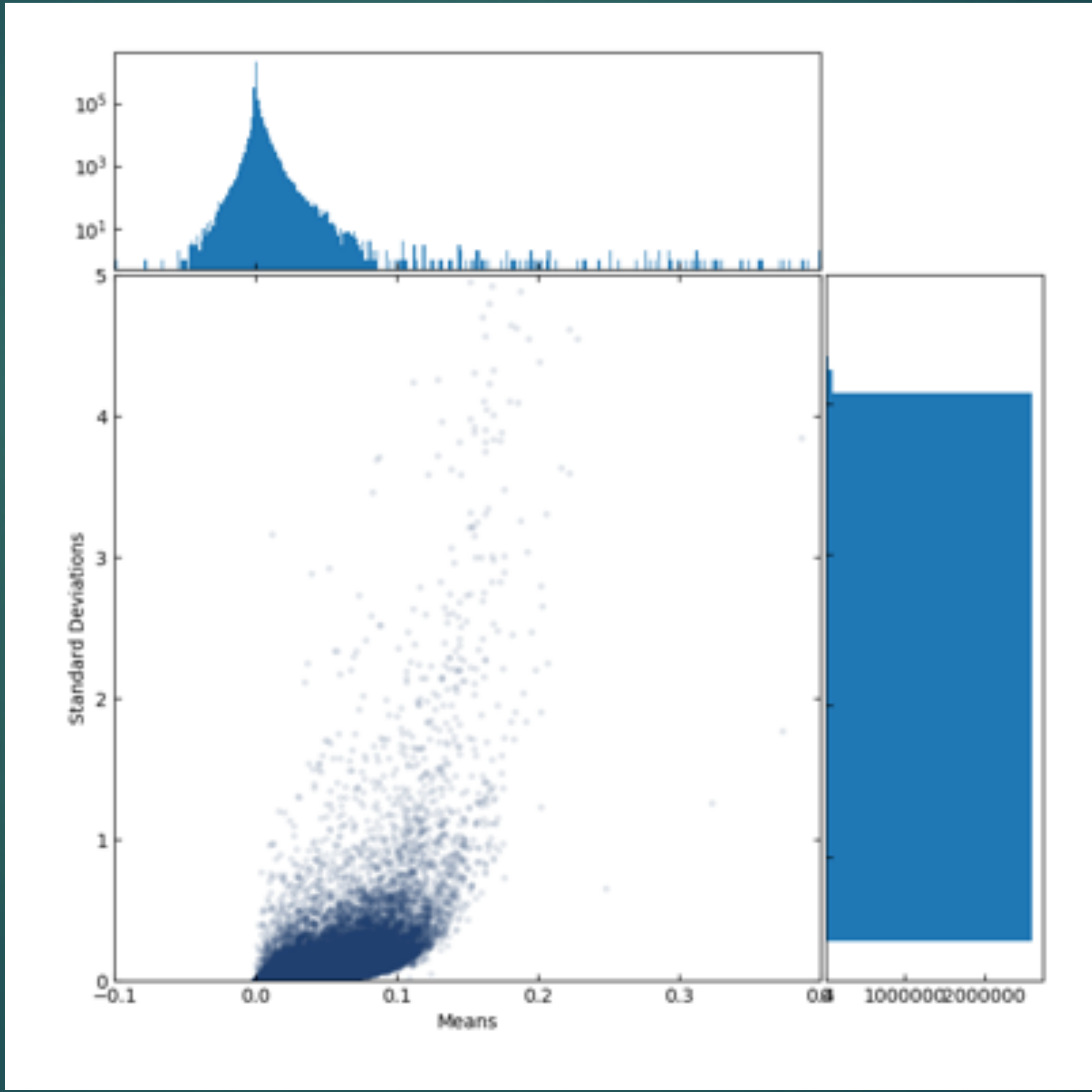


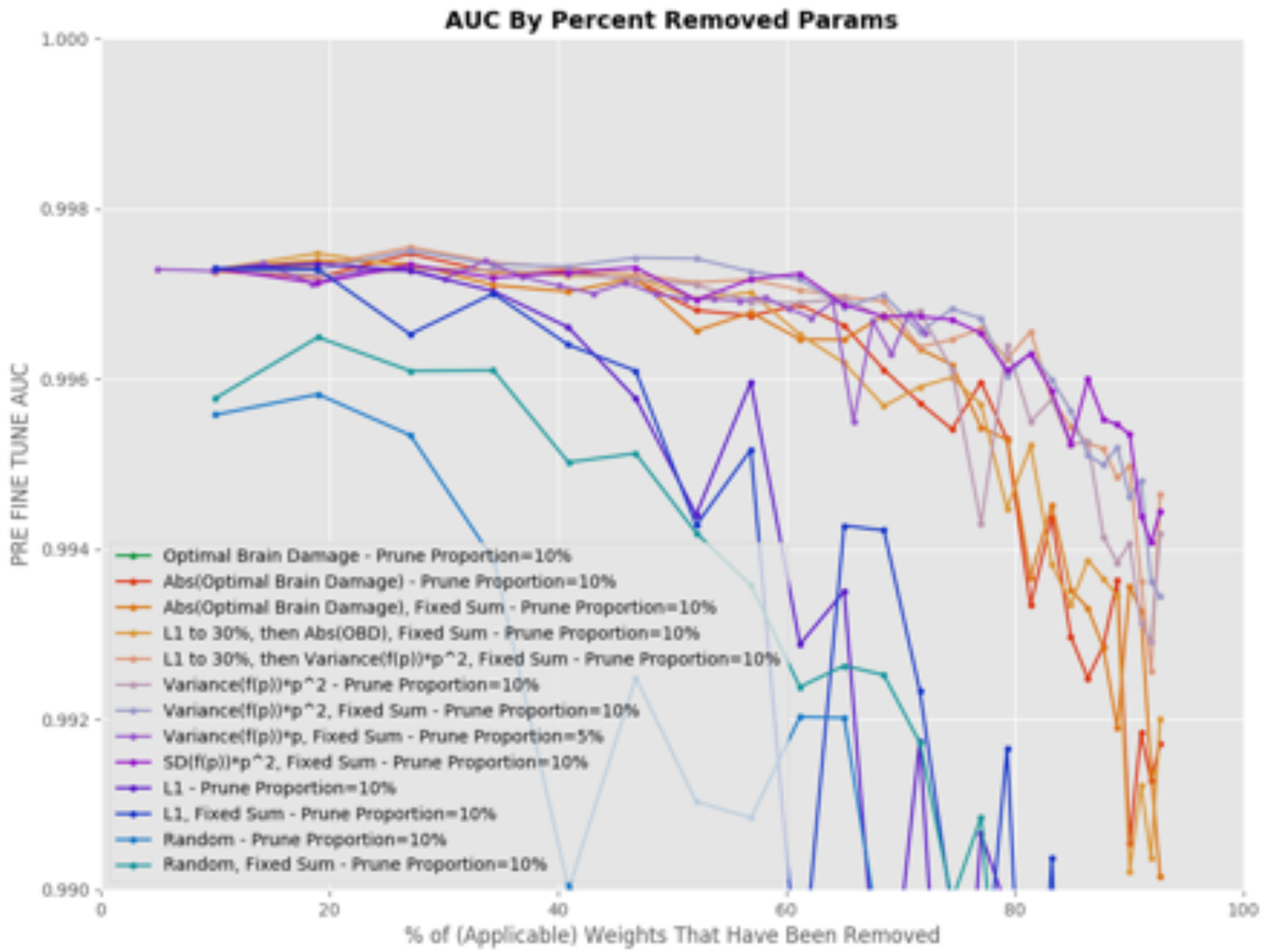
Prune round 1



- → What's going on?
- → Taylor series estimation isn't perfect, and parameter effects aren't independent.
- → In essence, we're using math to take a HUGE step-size during training, i.e. many p_i → 0, and that's just not a good idea. That's what small step-sizes during training are for!
- → As a result, we end up actually selecting for parameters that affect the model most, because E(loss-change) is a high negative (false). This is why removing random weights is better! (in complex, big models, I think)
- → Instead, what if we selected for the opposite: parameters that barely affect the loss? Instead of E(change-loss), can we look at variance(change-loss)?

What does SD(L''(p)) vs MEAN(L''(p)) look like?



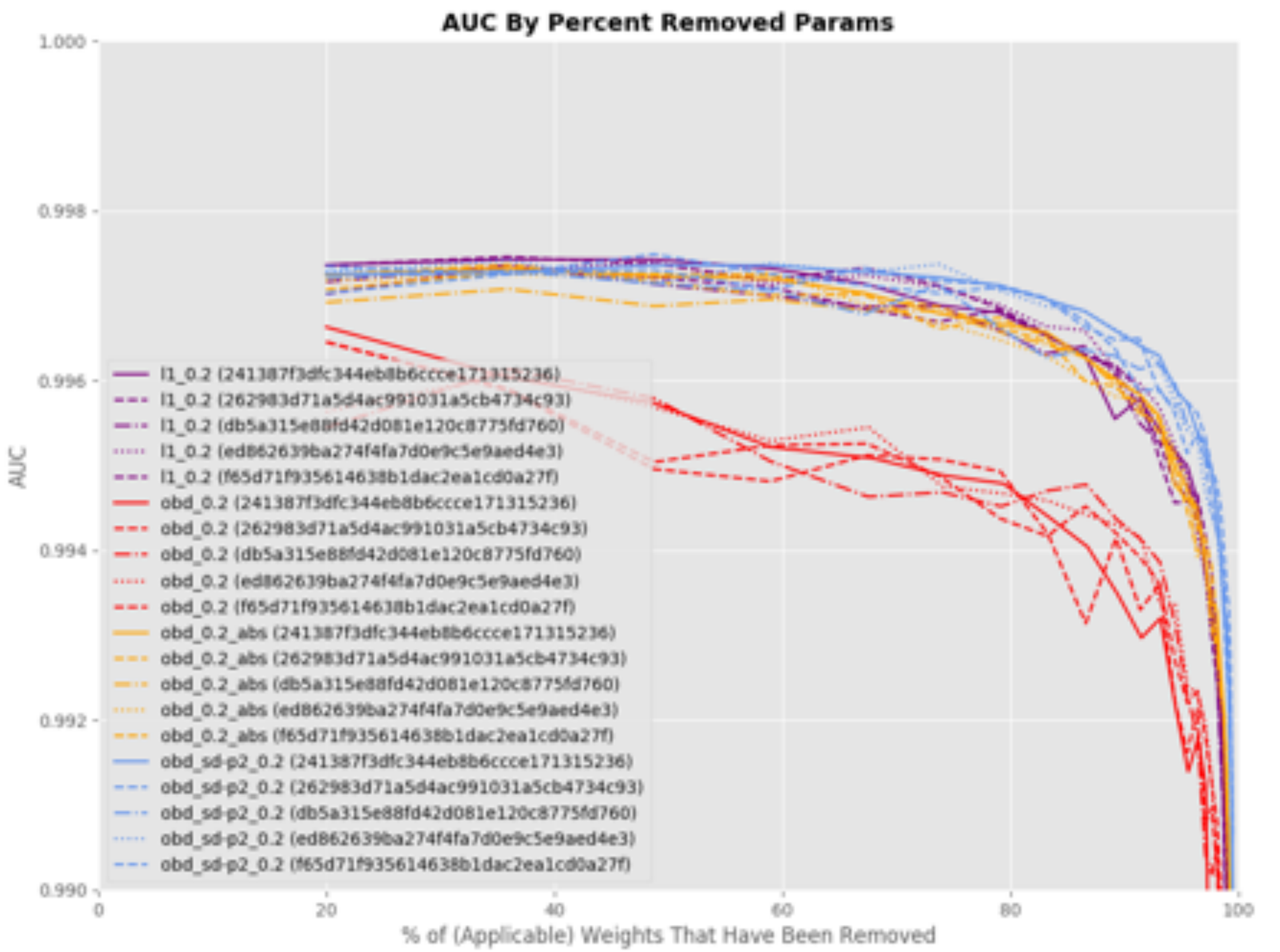

AUC By Percent Removed Params
PRE FINE TUNE AUC
% of (Applicable) Weights That Have Been Removed
1.000
0.998
0.996
0.994
0.992
0.990
0
20
40
60
80
100
Optimal Brain Damage - Prune Proportion=10%
Abs(Optimal Brain Damage) - Prune Proportion=10%
Abs(Optimal Brain Damage), Fixed Sum - Prune Proportion=10%
L1 to 30%, then Abs(OBD), Fixed Sum - Prune Proportion=10%
L1 to 30%, then Variance(f(p))*p^2, Fixed Sum - Prune Proportion=10%
Variance(f(p))*p^2 - Prune Proportion=10%
Variance(f(p))*p^2, Fixed Sum - Prune Proportion=10%
Variance(f(p))*p, Fixed Sum - Prune Proportion=5%
SD(f(p))*p^2, Fixed Sum - Prune Proportion=10%
L1 - Prune Proportion=10%
L1, Fixed Sum - Prune Proportion=10%
Random - Prune Proportion=10%
Random, Fixed Sum - Prune Proportion=10%

VARIANCE APPROACH

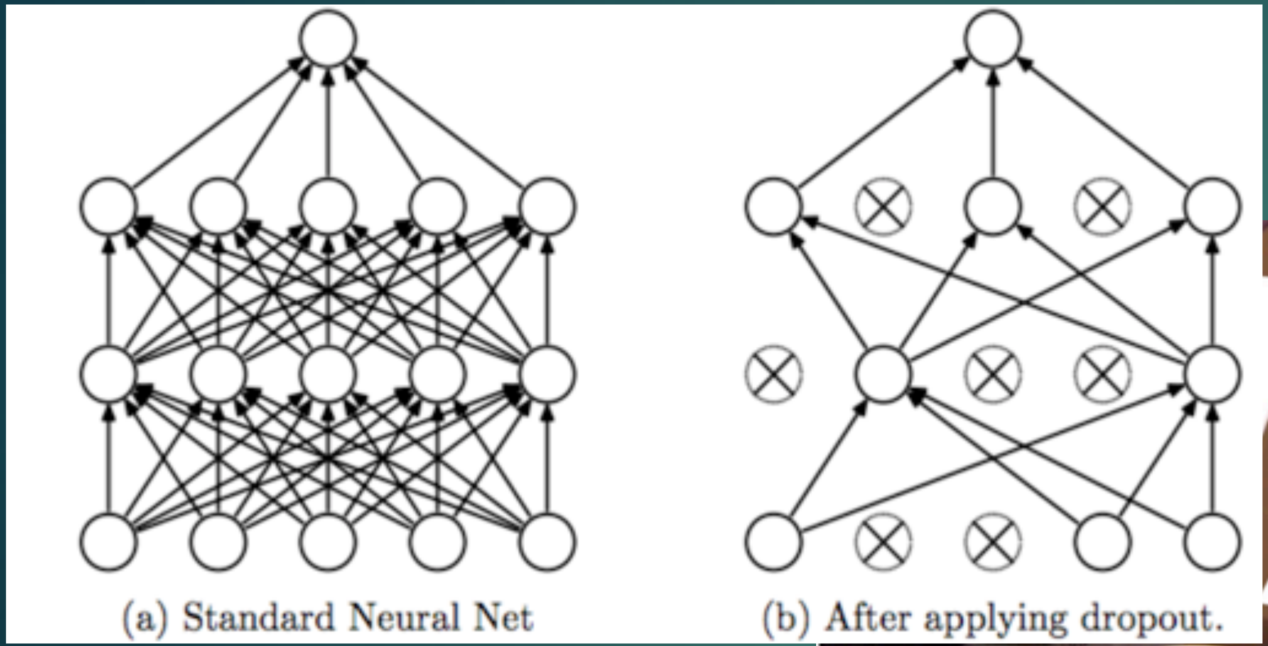
$$\text{variance}\left(\frac{L''(p_i=p)}{2}\cdot p^2\right) = \frac{p^4}{4}\,\text{variance}(L''(p_i=p))$$

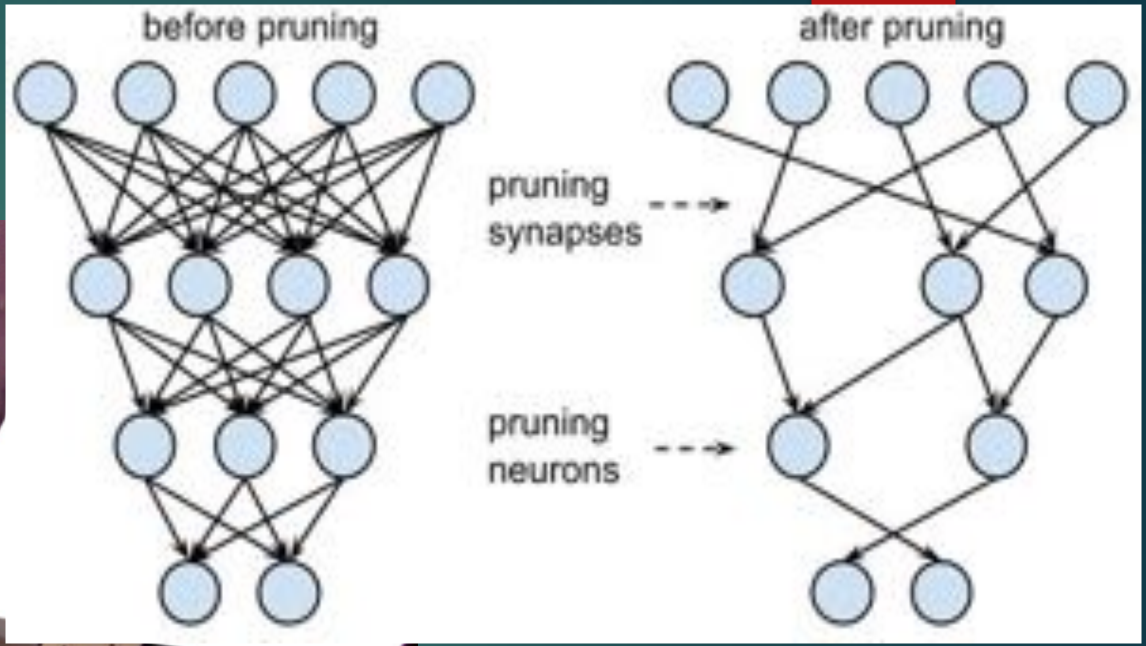
$$\mathcal{L}$$

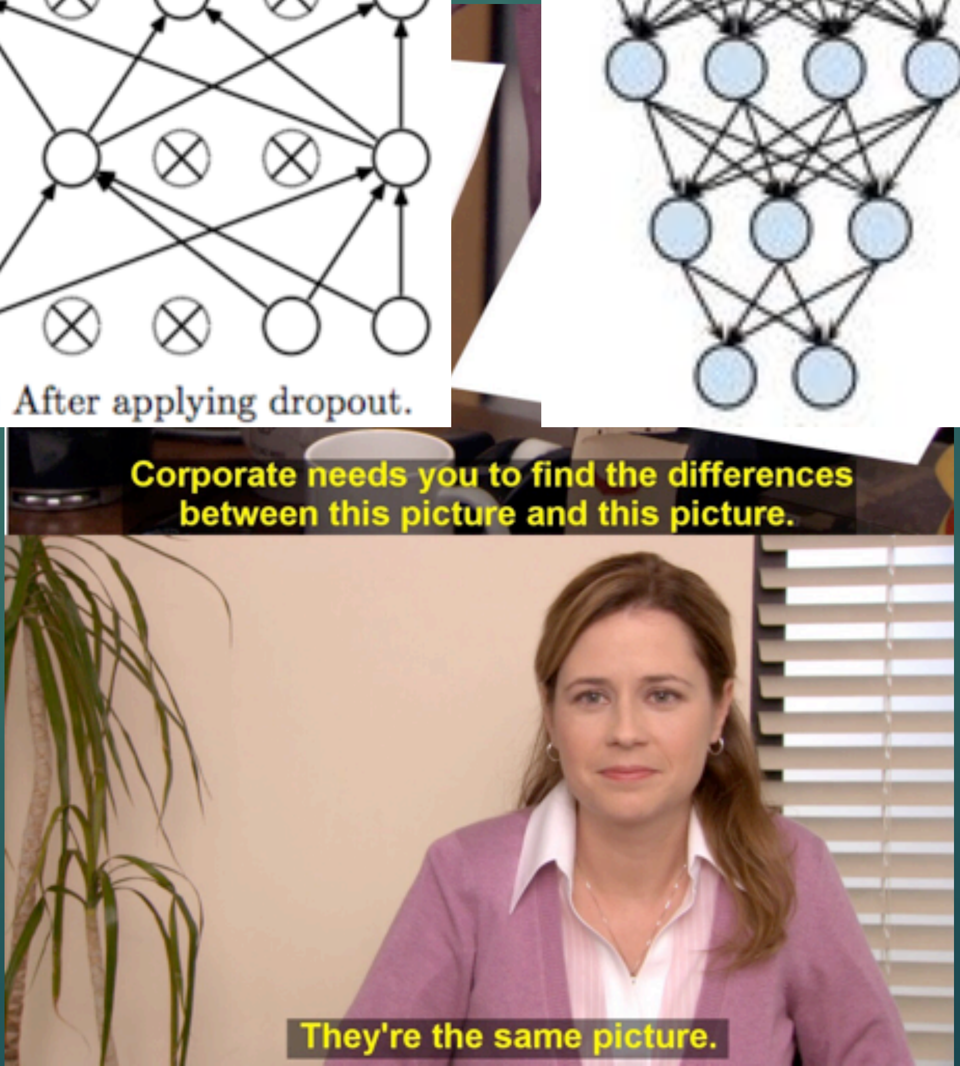
→ Same ordering as: $p^2\,\text{sd}(L''(p_i=p))$


AUC By Percent Removed Params
AUC
% of (Applicable) Weights That Have Been Removed
1.000
0.998
0.996
0.994
0.992
0.990
0
20
40
60
80
100
l1_0.2 (241387f3dfc344eb8b6ccce171315236)
l1_0.2 (262983d71a5d4ac991031a5cb4734c93)
l1_0.2 (db5a315e88fd42d081e120c8775fd760)
l1_0.2 (ed862639ba274f4fa7d0e9c5e9aed4e3)
l1_0.2 (f65d71f935614638b1dac2ea1cd0a27f)
obd_0.2 (241387f3dfc344eb8b6ccce171315236)
obd_0.2 (262983d71a5d4ac991031a5cb4734c93)
obd_0.2 (db5a315e88fd42d081e120c8775fd760)
obd_0.2 (ed862639ba274f4fa7d0e9c5e9aed4e3)
obd_0.2 (f65d71f935614638b1dac2ea1cd0a27f)
obd_0.2_abs (241387f3dfc344eb8b6ccce171315236)
obd_0.2_abs (262983d71a5d4ac991031a5cb4734c93)
obd_0.2_abs (db5a315e88fd42d081e120c8775fd760)
obd_0.2_abs (ed862639ba274f4fa7d0e9c5e9aed4e3)
obd_0.2_abs (f65d71f935614638b1dac2ea1cd0a27f)
obd_sd-p2_0.2 (241387f3dfc344eb8b6ccce171315236)
obd_sd-p2_0.2 (262983d71a5d4ac991031a5cb4734c93)
obd_sd-p2_0.2 (db5a315e88fd42d081e120c8775fd760)
obd_sd-p2_0.2 (ed862639ba274f4fa7d0e9c5e9aed4e3)
obd_sd-p2_0.2 (f65d71f935614638b1dac2ea1cd0a27f)

- →**Next Steps:**
- →**remove neurons instead of just single weights**
- →**Merge neurons instead of just deleting them**
- →**We assumed we couldn't calculate L(p=0) directly because of computational problems, but could we estimate it in other ways?**
  - → Idea: apply dropout, track which weights / nodes are being removed, and then average over many samples to estimate what elements are OK to remove! Essentially run a regression over the resulting accuracy to see which elements have high vs low impact.


(a) Standard Neural Net
(b) After applying dropout.


before pruning
after pruning
pruning synapses
pruning neurons


Corporate needs you to find the differences between this picture and this picture.
They're the same picture.