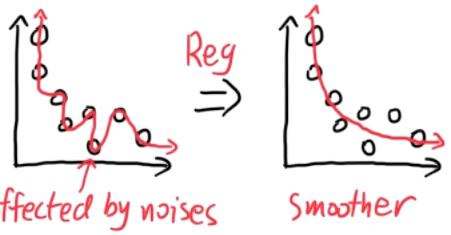
9) Regularization

Families of Regularization

- It penalizes model from overlearning specific examples
  - Useful for larger dataset & deeper methods
  - U can regularize node, weight, or the data



train() & eval() modes

- Workflow: epoch loop { model.train(); batch_loop: {...};
  model.eval(); with torch.no_grad(): yHat = model(x); }

① model.train(): Regularization on
② model.eval(): Regularization off
} switch model b/t these modes, only for batch & dropout regularization

③ torch.no_grad(): deactivate unnecessary gradient computations

Dropout Regularization

- Process: "zeroing output"

① During training, randomly drop-out the nodes from the model by chance P

② During testing (NO Dropout!), so scale-down each node weight $W_i := W_i * (1-p)$

- OR: u can skip scale-down but scale-up leftover node weight in ①

$W_i := W_i * g$, $g = \frac{1}{1-p}$

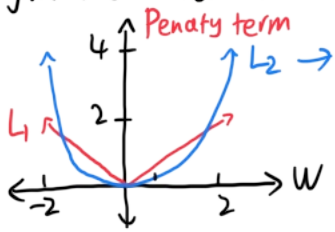
- Interpretation: stabilizes the model bc it is less reliant on individual nodes
  unnecessary/detrimental for small model/dataset

Weight Regularization

- L1 Regularization: $W = \underset{w}{\operatorname{argmin}} J$ but now $J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i) + \frac{\alpha}{2m}\lambda |W_i|$ ← usually 0.01, ← # weights

- L2 Regularization: $W = \operatorname{argmin} J$ but now $J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i) + \lambda \|W_i\|_2^2$ ($W^T W$)

- Compare L_1 vs. L_2:
  model reduces w regaldless of its position. So this will create sparse weights (=0)



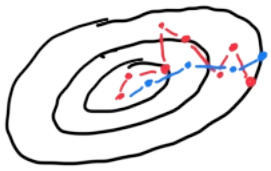
L_2 → Large w pressures model to reduce it
model compresses w to range close to 0

- Interpretation: avoids numerical instability (for large w, similar input → very different output)

Mini-Batching

- Implicity regularize weight. Usually, pick batch_size = 2^k over [2, 512] ↙ GPU processing efficiency

- Visual:



- True SGD (batch_size=1)
- Mini-Batching
- One-Batch (batch_size = len(dataset))

Train speed ↓ +

Generalization ↑ +

- Ensure all batches have the same size.