Regularization and Optimization

DEEP LEARNING

E. FATEMIZADEH FALL 2023

Contents

The Bias-Variance Tradeoff - Recall

- Definition
- Given dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{i=1}^{N}$ drawn *i.i.d.* from some distribution $P(\boldsymbol{X}, Y)$
- Machine learning algorithm $\mathcal{A}$ trained on dataset $\mathcal{D}$, $h_{\mathcal{D}} = \mathcal{A}(\mathcal{D})$ to learn a hypothesis
- $\boldsymbol{x} \in \mathcal{R}^{\boldsymbol{d}}$ and $\boldsymbol{y} \in \mathcal{R}$
- Expected Label (given $\boldsymbol{x}$):

$$\bar{y}(\boldsymbol{x}) = E_{y|\boldsymbol{x}}[Y] = \int_y yp\,(y|\boldsymbol{x})\partial y$$

- Expected Test Error (given $h_{\mathcal{D}}$):

$$E_{(\mathbf{x},y)\sim P}[(h_D(\boldsymbol{x}) - y)^2] = \iint_x (h_D(\mathbf{x}) - y)^2\, Pr(\boldsymbol{x}, y)\, \partial y \partial \boldsymbol{x}.$$

- remember that $\mathcal{D}$ itself is drawn from $\mathcal{P}^N$, and is therefore a *random variable*. Further, $h_{\mathcal{D}}$ is a function of $\mathcal{D}$, and is therefore also a *random variable*. And we can of course compute its expectation.

The Bias-Variance Tradeoff - Recall

- Note that $\mathcal{D}$ itself is drawn from $\mathcal{P}^N$, and is therefore a *random variable*. Further, $h_{\mathcal{D}}$ is a function of $\mathcal{D}$, and is therefore also a *random variable*. And we can of course compute its expectation.
- Expected Classifier (given $\mathcal{A}$):

$$\bar{h} = E_{\mathcal{D}\sim P^N}[h_{\mathcal{D}}] = \int_{\mathcal{D}} h_{\mathcal{D}} Pr(\mathcal{D}) \partial\mathcal{D}$$

- where $Pr(\mathcal{D})$ is the probability of drawing $\mathcal{D}$ from $\mathcal{P}^N$. Here, $\bar{h}$ is a weighted average over functions.
- We can also use the fact that $h_{\mathcal{D}}$ is a random variable to compute the expected test error only given $\mathcal{A}$, taking the expectation also over $\mathcal{D}$.
- Expected Test Error (given $\mathcal{A}$):

$$E_{\substack{(\boldsymbol{x},y)\sim P \\ \mathcal{D}\sim P^N}}[(h_{\mathcal{D}}(\boldsymbol{x}) - y)^2] = \int_{\mathcal{D}} \int_{\boldsymbol{x}} \int_{y} (h_{\mathcal{D}}(\boldsymbol{x}) - y)^2 P(\boldsymbol{x}, y) Pr(\mathcal{D}) \partial\boldsymbol{x} \partial y \partial\mathcal{D}$$

- This shows the quality of machine learning algorithm $\mathcal{A}$ with respect to a data distribution, $P(\boldsymbol{X}, Y)$.

The Bias-Variance Tradeoff - Recall

- Expected Test Error:

$$E_{\boldsymbol{x},\,y,\,\mathcal{D}}[(h_{\mathcal{D}}(\boldsymbol{x})-y)^2] \overset{\text{def}}{=} E_{\substack{(\boldsymbol{x},y)\sim P \\ \mathcal{D}\sim P^n}}[(h_{\mathcal{D}}(\boldsymbol{x})-y)^2]$$

- It can be shown:

$$\underbrace{E_{\boldsymbol{x},\,y,\,\mathcal{D}}[(h_{\mathcal{D}}(\boldsymbol{x})-y)^2]}_{\text{Expected Test Error}} = \underbrace{E_{\boldsymbol{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\boldsymbol{x})-\bar{h}(\boldsymbol{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\boldsymbol{x},y}[(\bar{y}(\boldsymbol{x})-y)^2]}_{\text{irreducible Noise}} + \underbrace{E_{\boldsymbol{x}}\left[(\bar{h}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))^2\right]}_{\text{Bias}^2}$$

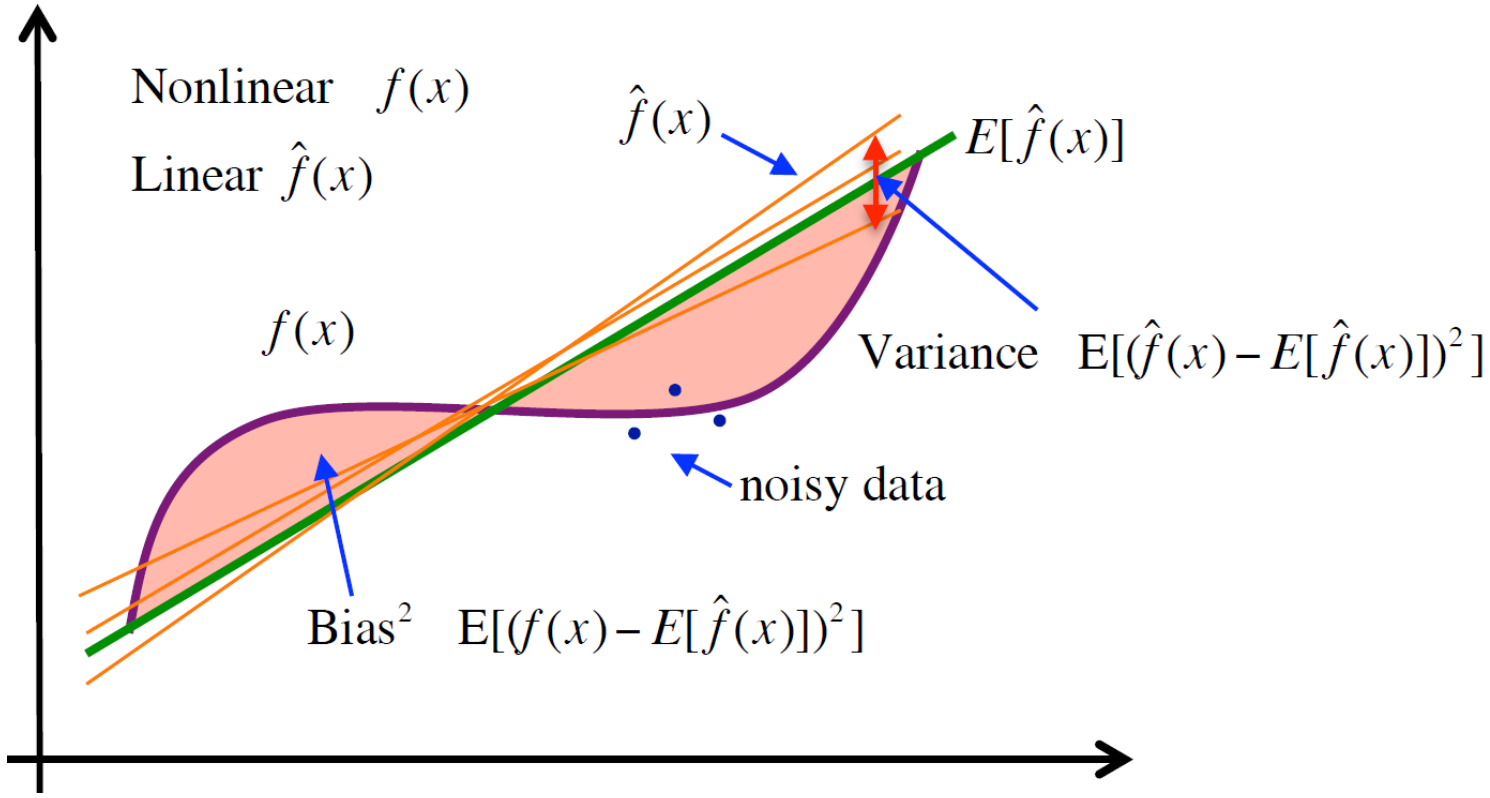
- There are three meaningful terms.
  - Variance
  - Noise
  - Bias²

The Bias-Variance Tradeoff - Recall

- Variance, $E_{\boldsymbol{x},\mathcal{D}}\left[\left(h_{\mathcal{D}}(\boldsymbol{x})-\bar{h}(\boldsymbol{x})\right)^2\right]$:
- Captures how much your classifier changes if you train on a different *training* set. How "over-specialized" is your classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?
- Bias, $E_{\boldsymbol{x}}\left[(\bar{h}(\boldsymbol{x})-\bar{y}(\boldsymbol{x}))^2\right]$:
- What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model.
- Irreducible Noise, $E_{\boldsymbol{x},y}[(\bar{y}(\boldsymbol{x})-y)^2]$:
- How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data.

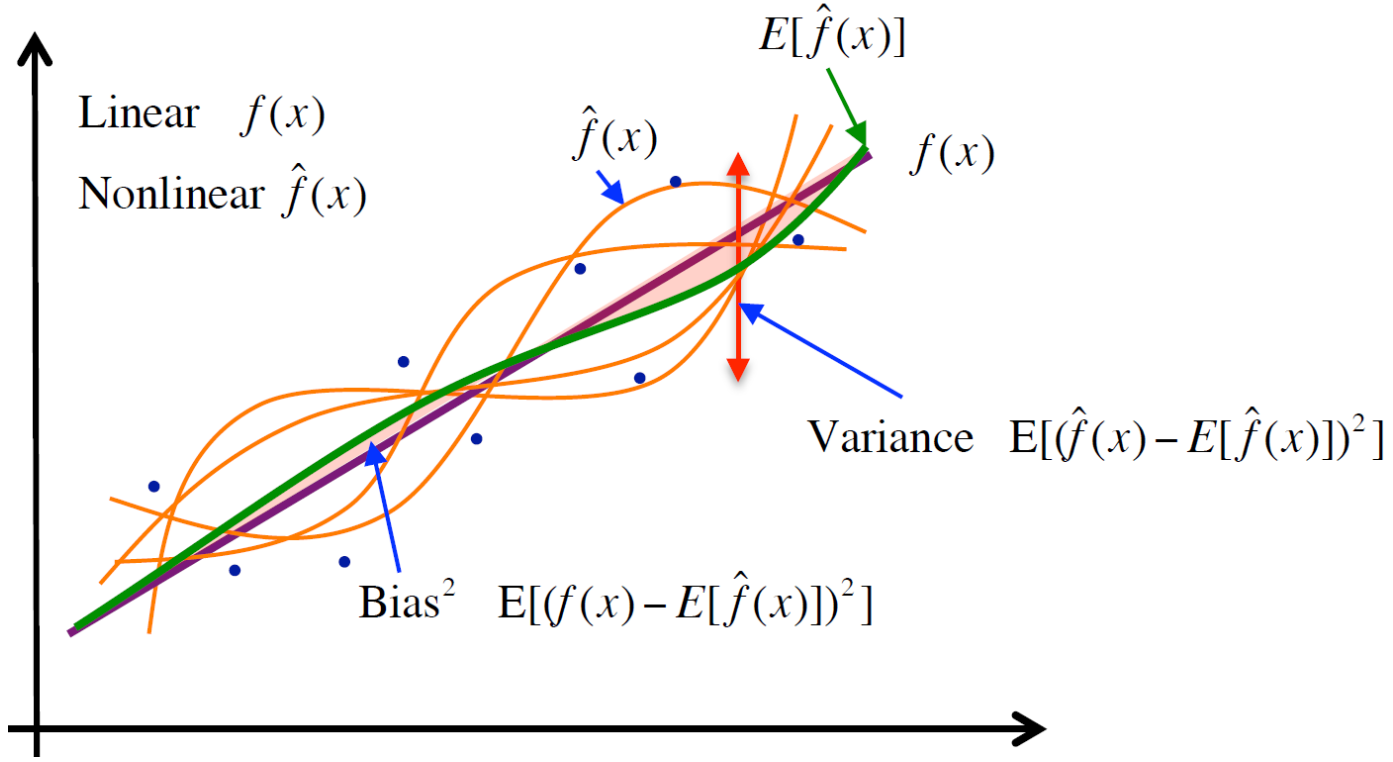
The Bias-Variance Tradeoff - Recall

- Scenario #1 (too simple machine)
- High Bias
- Low Variance



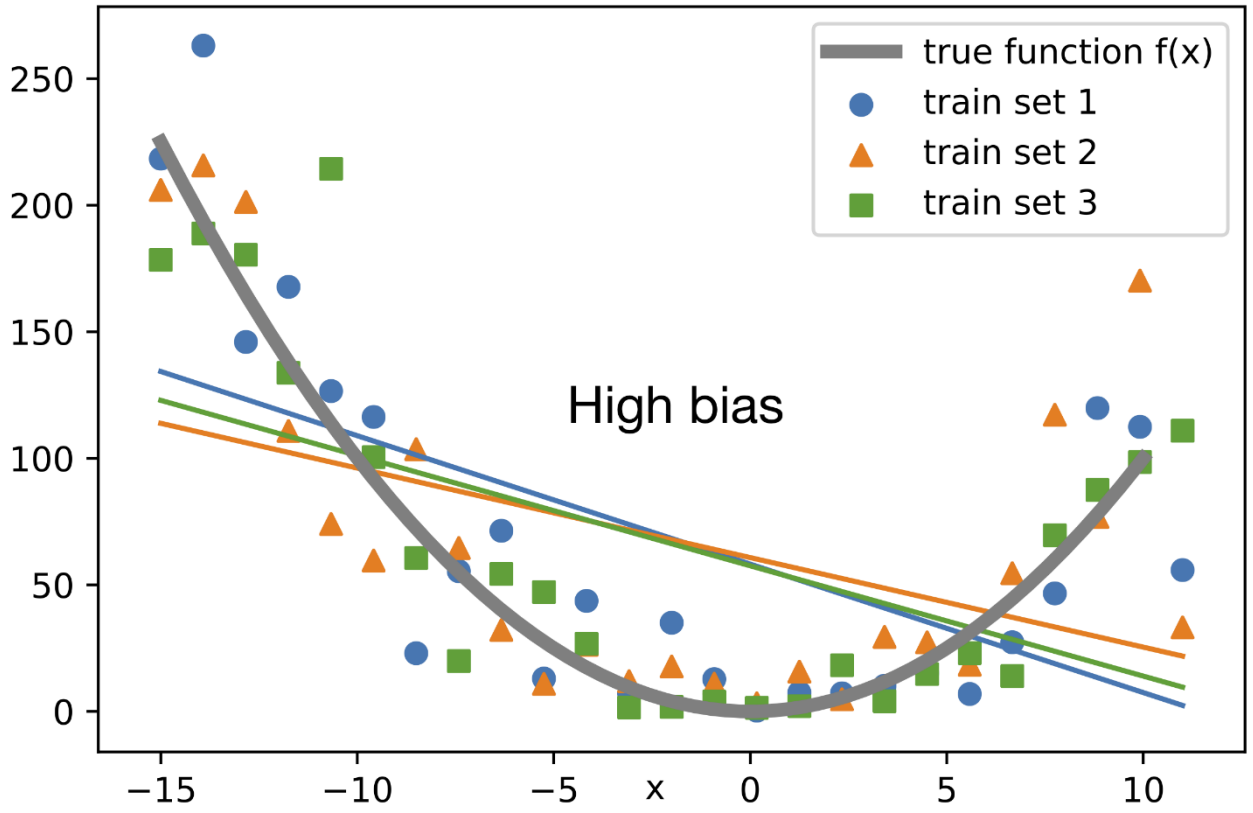
The Bias-Variance Tradeoff - Recall

- Scenario #2 (too complex machine)
- Low Bias
- High Variance



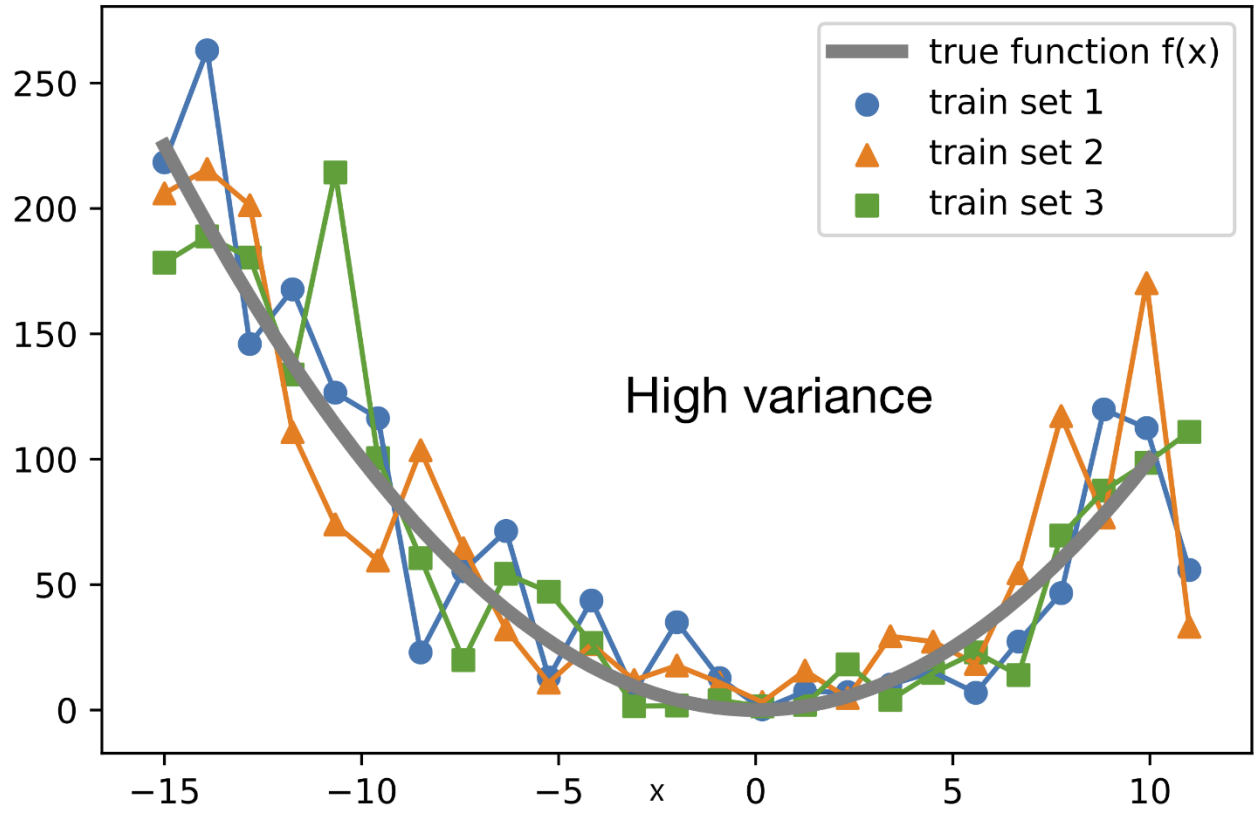
The Bias-Variance Tradeoff - Recall

- Scenario #1 (too simple machine)
- High Bias
- Low Variance



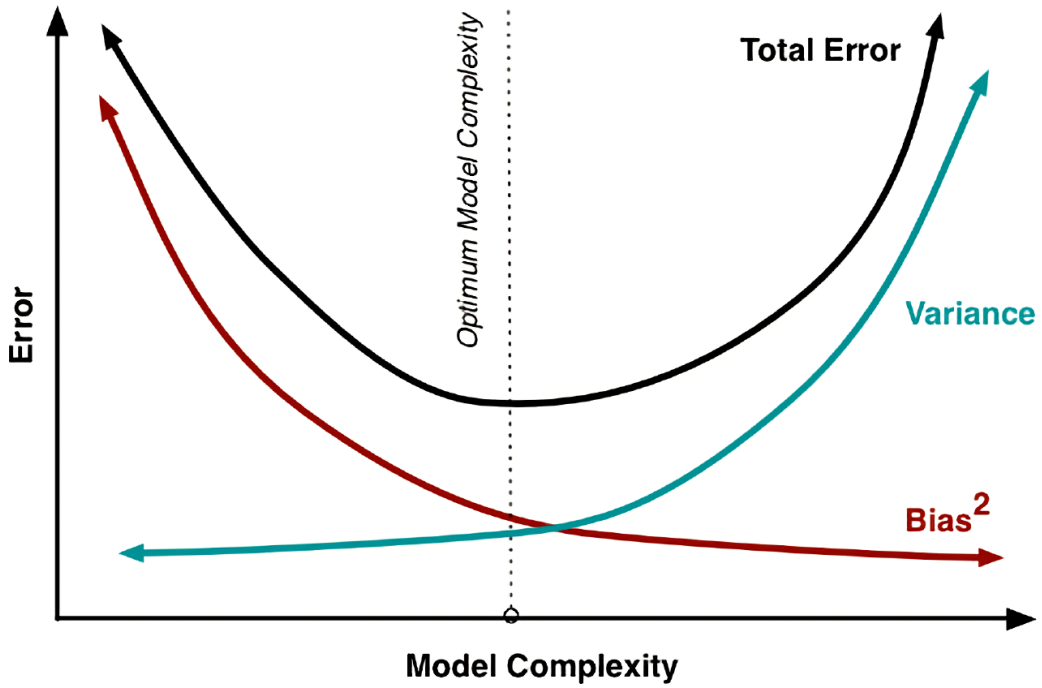
The Bias-Variance Tradeoff - Recall

- Scenario #2 (too complex machine)
- Low Bias
- High variance



The Bias-Variance Tradeoff - Recall

- Bias-Variance vs Complexity (Traditional View)

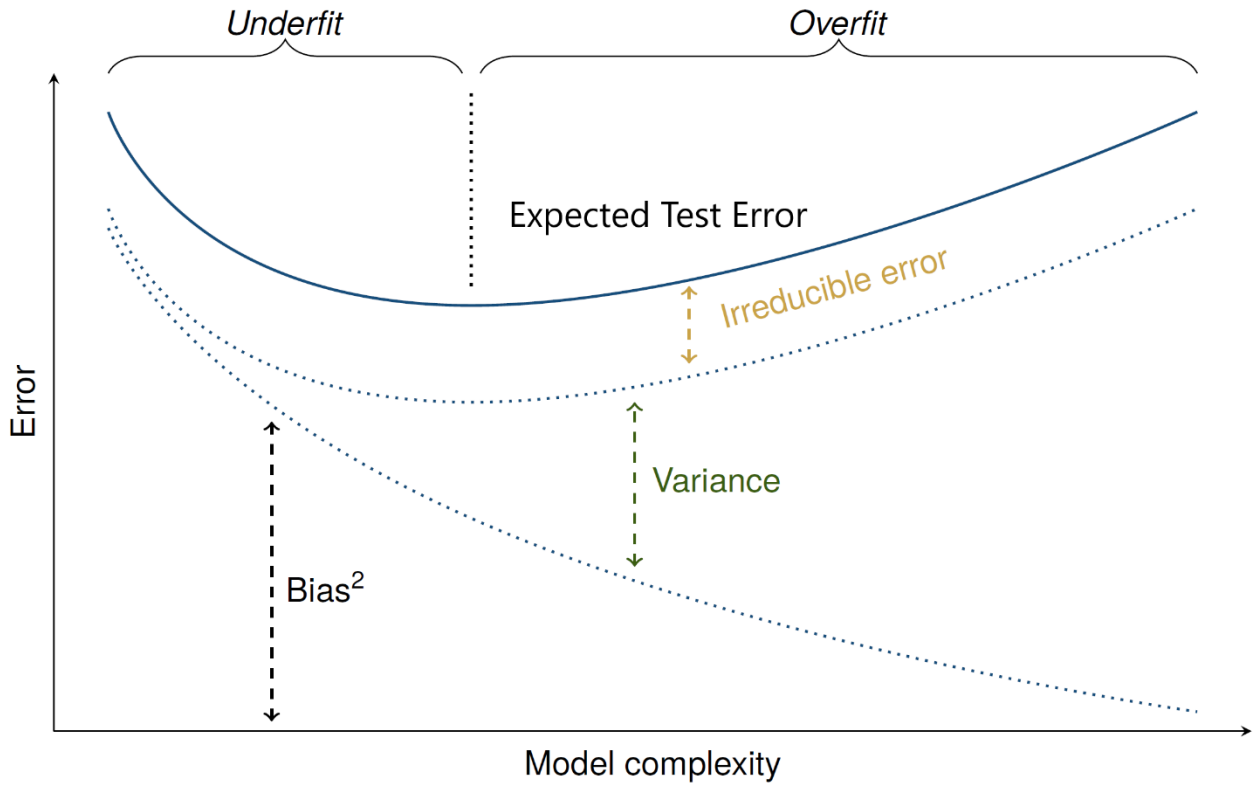


$$sign\left(\frac{\partial Bias}{\partial Modle\_Complexity}\right) = -sign\left(\frac{\partial Variance}{\partial Modle\_Complexity}\right)$$

$$Expected\ Test\ Error = Variance + Bias^2 + Irreducible\ Error$$

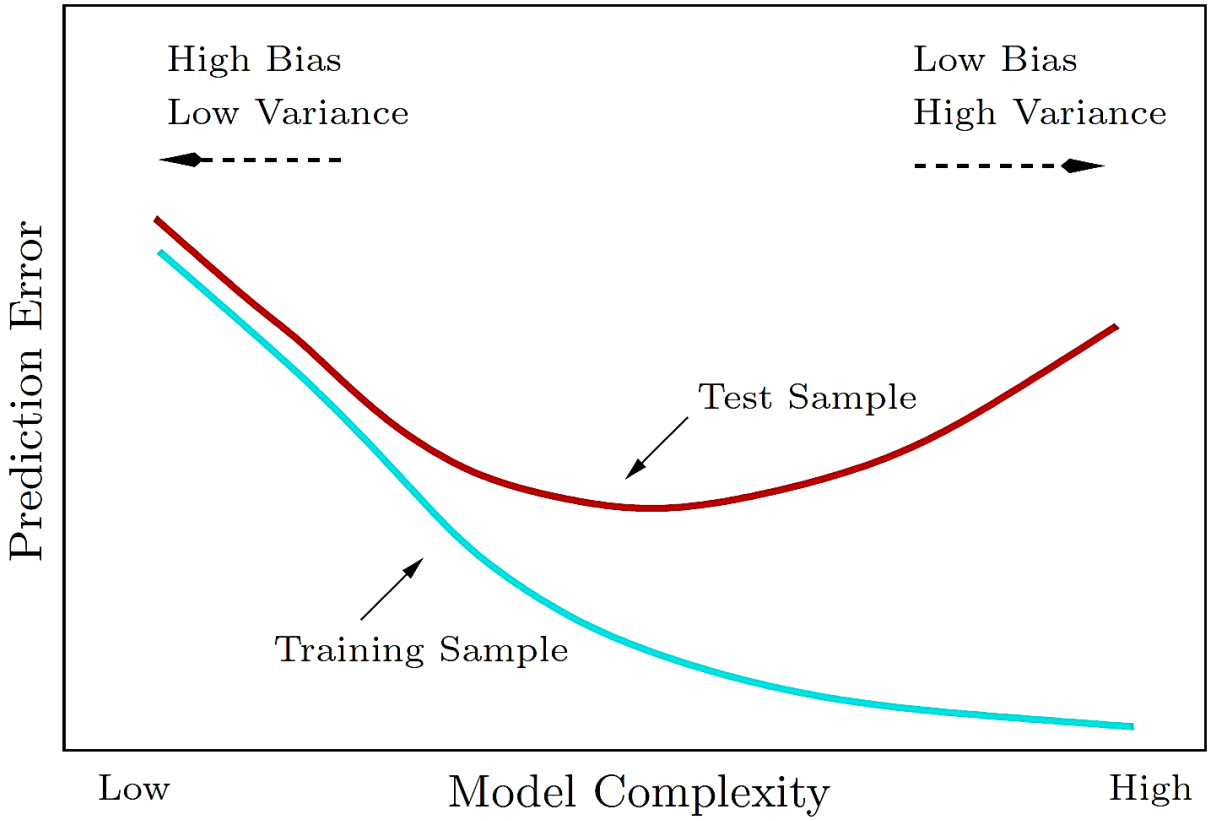
The Bias-Variance Tradeoff - Recall

- A more realistic (including irreducible error):



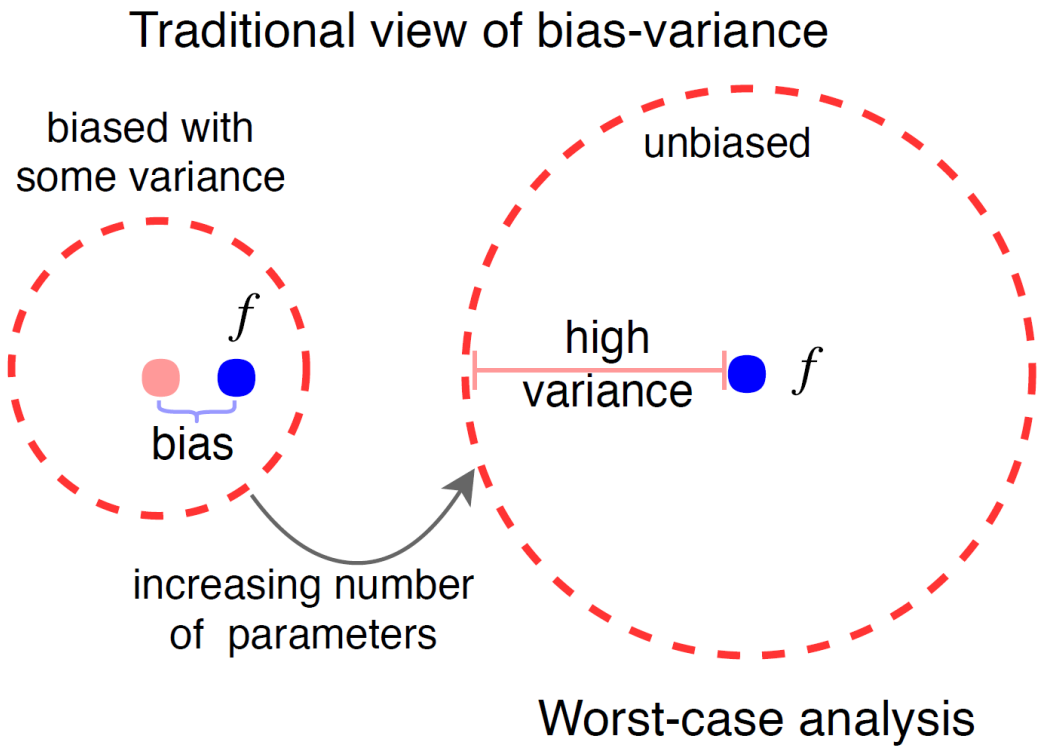
The Bias-Variance Tradeoff - Recall

- Traditional View:



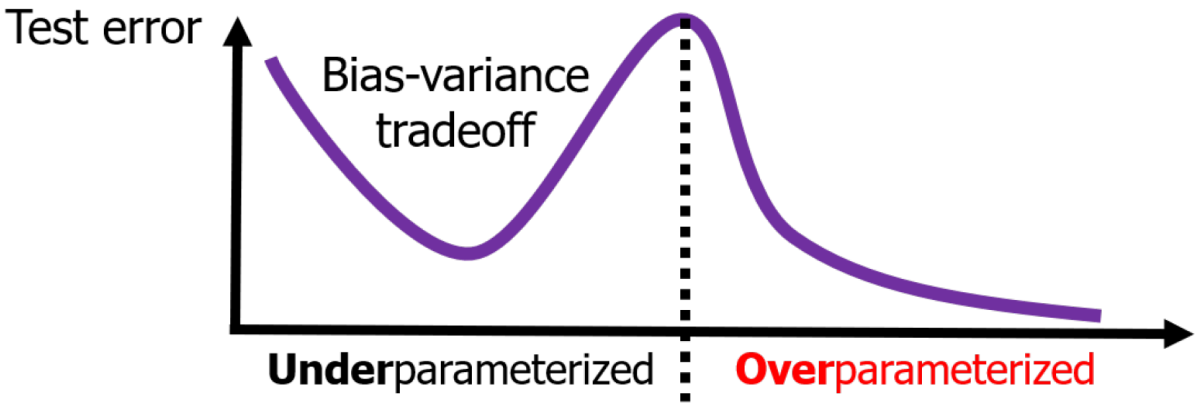
The Bias-Variance Tradeoff - Recall

- Traditional View:



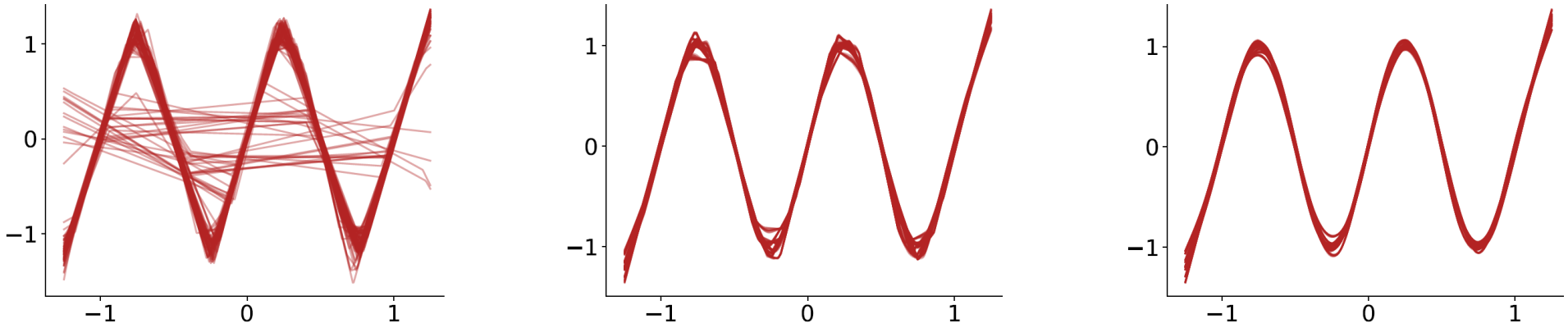
The Bias-Variance Rethinking (starting 2019)

- Historically, it was believed that the trade-off applies to neural networks as well.
- To address the trade-off, early stopping and dropping are techniques to avoid overfitting.
- In the 2010s, neural networks challenged the classical bias-variance trade-off.
- The classical U-shaped curve was replaced with a double-descent curve.
- While bias decreases monotonically, variance first increases and then decreases after a point called the interpolation threshold.



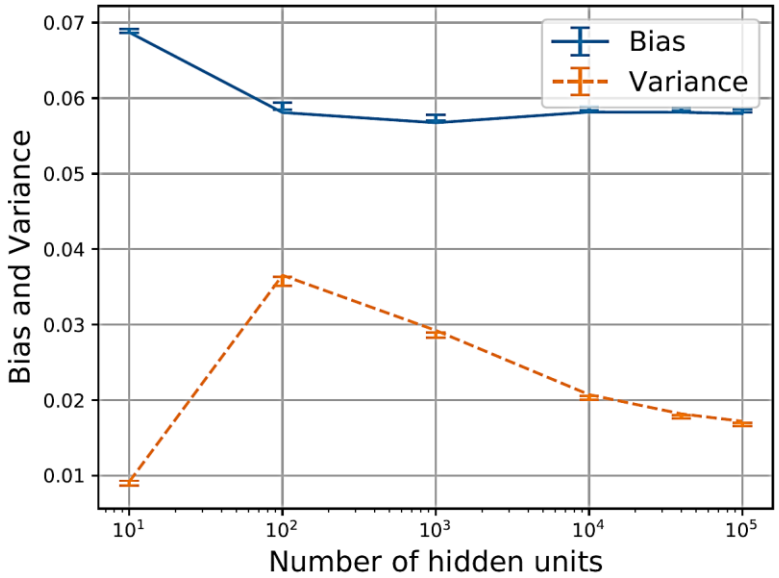
The Bias-Variance Rethinking (starting 2019)

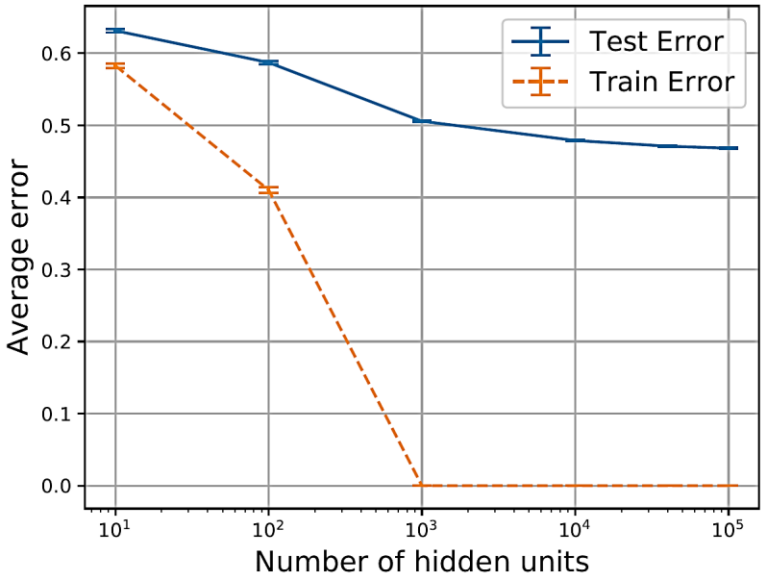
- A simple experiment:
  - 100 different learned sine functions of SLP of widths (hidden veurons) 15, 1000, and 10000 (from left to right)
  - The learned functions are increasingly similar with width, suggesting decreasing variance.



The Bias-Variance Rethinking (starting 2019)

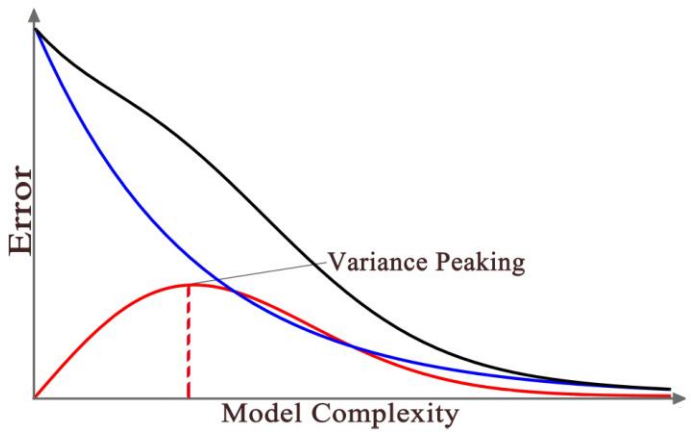
- A complex experiment:
  - CIFAR 10

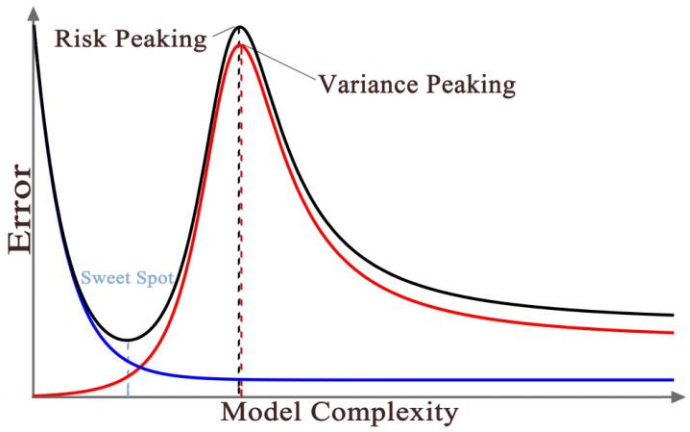


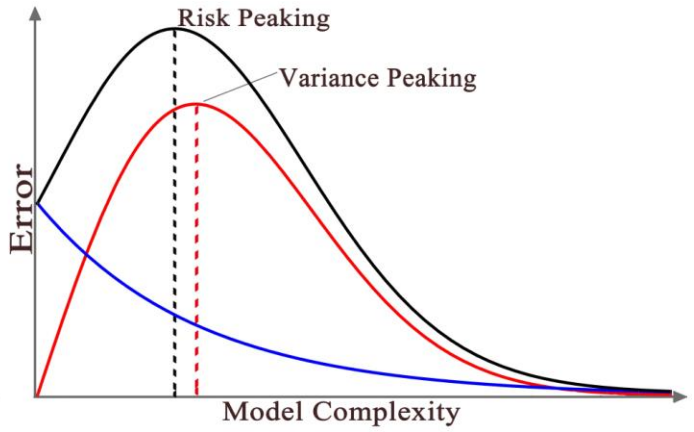


The Bias-Variance Rethinking (starting 2019)

- More Rethinking (three observable cases)

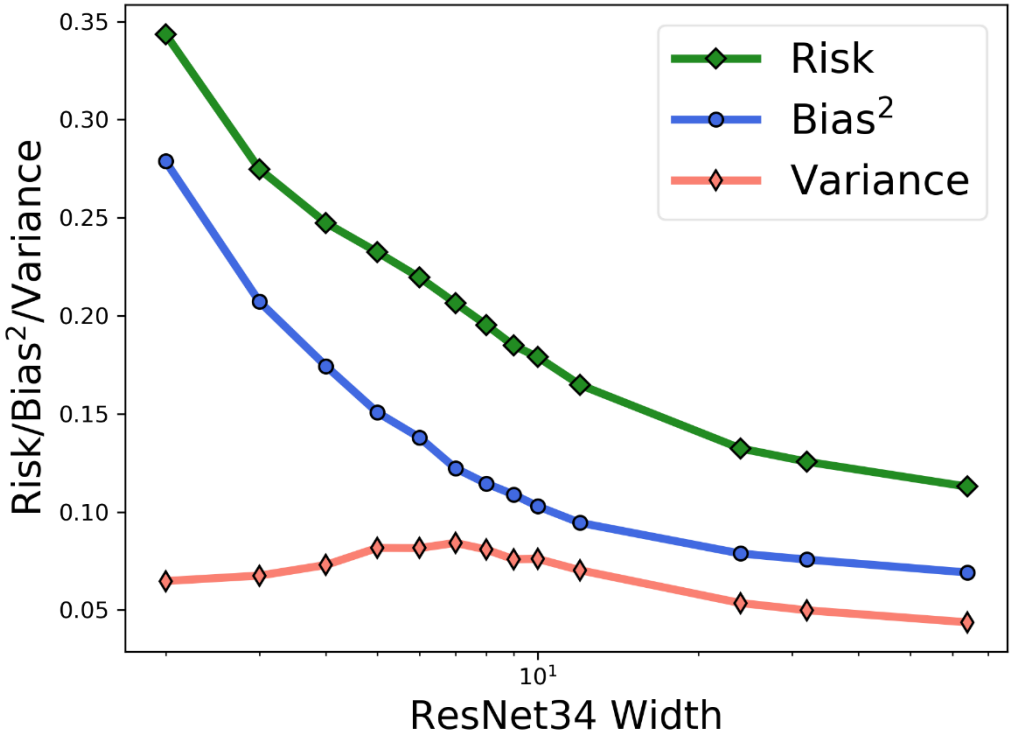






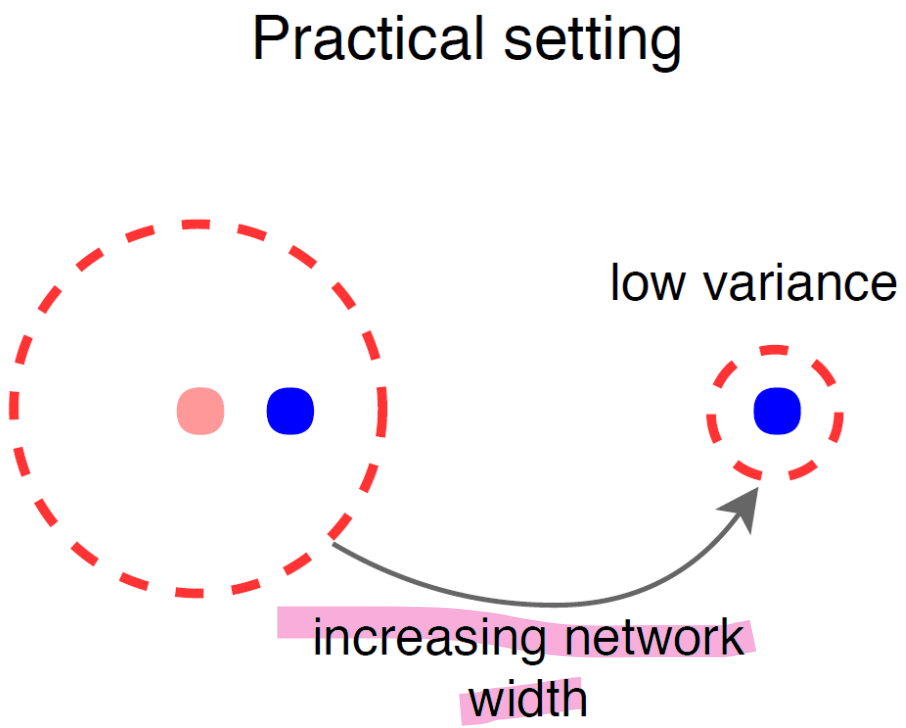
The Bias-Variance Rethinking (starting 2019)

- Example (CIFAR 10 on ResNet34):



The Bias-Variance Rethinking (starting 2019)

- Practical View:



The Bias-Variance Rethinking (starting 2019)

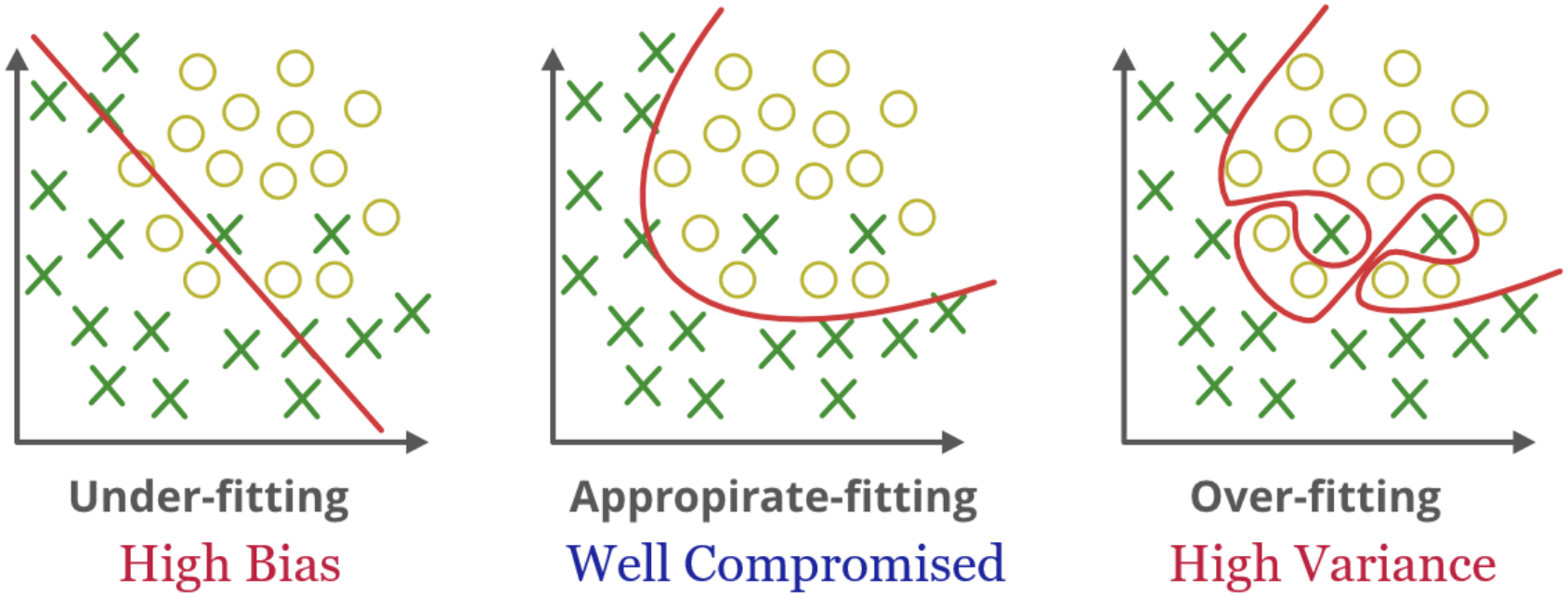
- References:
  - B. Neal, “A Modern Take on the Bias-Variance Tradeoff in Neural Networks”, arXiv:1810.08591v4 [cs.LG] 18 Dec 2019.
  - B. Neal, “On the Bias-Variance Tradeoff: Textbooks Need an Update”, arXiv:1912.08286v1 [cs.LG] 17 Dec 2019.
  - Z. Yang, “Rethinking Bias-Variance Trade-off for Generalization of Neural Networks”, arXiv:2002.11328v3 [cs.LG] 8 Dec 2020.
  - Y. Dar, “A Farewell to the Bias-Variance Tradeoff? An Overview of the Theory of Overparameterized Machine Learning”, arXiv:2109.02355v1 [stat.ML] 6 Sep 2021.

The Bias-Variance Tradeoff in Brief

- Simple models trained on different samples of data do not differ much from each other. However, far from the true (underfitting).
- On the other hand, complex models trained on different subsets of data are very different from each other (overfitting).
- Simple model: high bias, low variance
- Complex model: low bias, high variance

Regularization Motivation

- More Generalization Power
- Overcome Overfitting



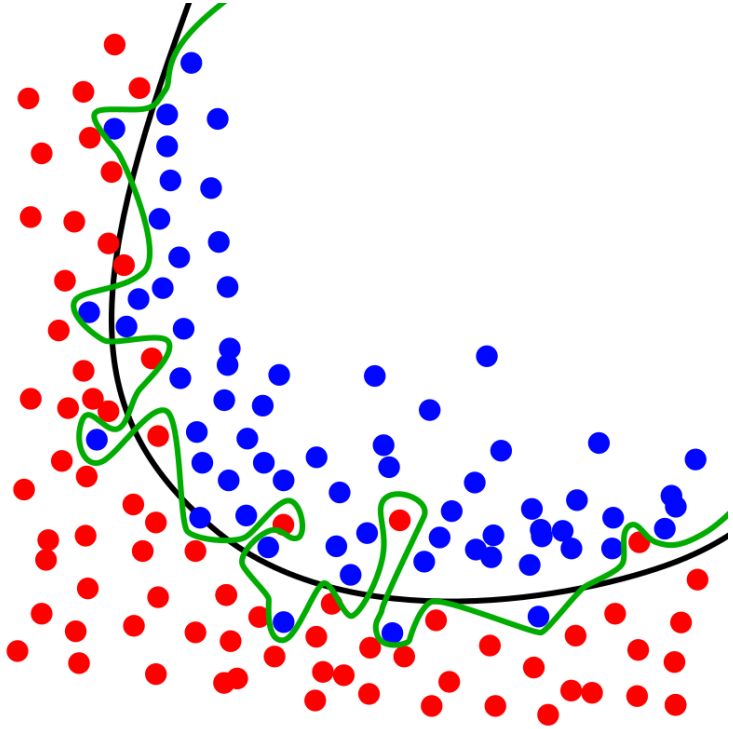
Regularization

- Overfitting Challenge (What to do?)

Symptoms		Cause	Solution
Training Error	Validation Error		
High	High	High Bias	• Increase model complexity • Train for more epochs • Add Features • Boosting
Low	High	High Variance	• Add more training data (Augmentation) • Reduce model complexity • Regularization • Bagging
Low	Low	Perfecto	Good job!

Regularization Definition

- "Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."
- Solution:
  - Parameter (weights) Penalties
  - Dataset Augmentation
  - Noise Robustness (input/output)
  - Early Stopping
  - Bagging and Other Ensemble Methods
  - Dropout



Parameter Penalties

- Instead empirical risk minimization:

$$minimize\big(Loss(Data|Model)\big)$$

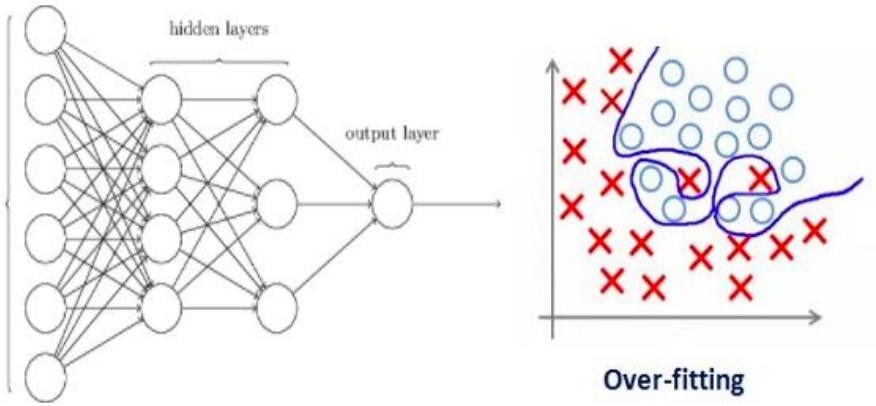
- Try structural risk minimization:

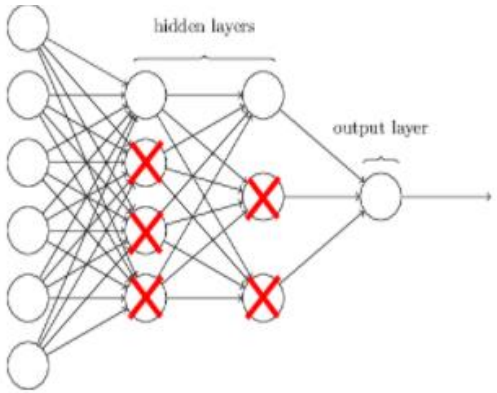
$$minimize\,\{Loss(Data|Model) + \lambda Complexity(Model)\}$$

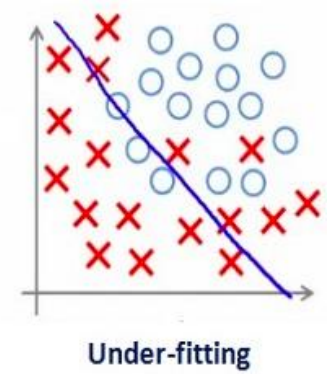
- Two major policies for quantify model complexity:
  - A function of the weights of all the features in the model.
  - A function of the total number of features with nonzero weights.

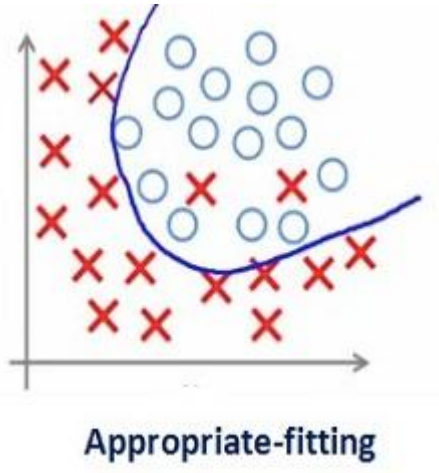
Parameter Penalties

- How it works:









Regularization by Parameters Penalties

- General Formulation:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- Typically not applied on Bias terms (why)?

L2 Regularization (Ridge Regression)

- An old fashion technique!
- Idea is weight decay
- Known as:
  - Ridge Regression
  - Tikhonov Regularization
- General formulations:

$$J_{L_2} = (MSE/CE) + \lambda \sum_{Weight \backslash Bias} \|w\|_2^2$$

$$J_{L_2} = (MSE/CE) + \sum_{Layers} \lambda_{Layers} \sum_{Weights \backslash \text{Bias}} \|w\|_2^2$$

L2 Regularization (Ridge)

- Formulation:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2} \|\boldsymbol{w}\|_2^2 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- It can be shown:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}))$$

- or

$$\boldsymbol{w} \leftarrow (1 - \alpha\epsilon)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

L2 Regularization - Analysis

- If we know exact solution:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}), \qquad \nabla J(\boldsymbol{w}^*) = 0$$

- 2nd order approximation around $\boldsymbol{w}^*$:

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^\top \underbrace{\nabla J(\boldsymbol{w}^*)}_{0} + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- It can be shown L2 regularization solution of $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is:

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H} \boldsymbol{w}^*$$

- For convex approximation $\boldsymbol{H}$ is pdm, now decompose it:

$$\boldsymbol{H} = \boldsymbol{Q} \boldsymbol{\Lambda} \boldsymbol{Q}^{-1}$$

- $\boldsymbol{Q}$ is unitary ($\boldsymbol{Q}^{\boldsymbol{T}} = \boldsymbol{Q}^{-1}$) and $\boldsymbol{\Lambda}$ is diagonal and positive.

L2 Regularization - Analysis

- With some matrix manipulation:

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* = \left(\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}} + \alpha \boldsymbol{I}\right)^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = [\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{Q}^{\boldsymbol{T}}]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^*$$

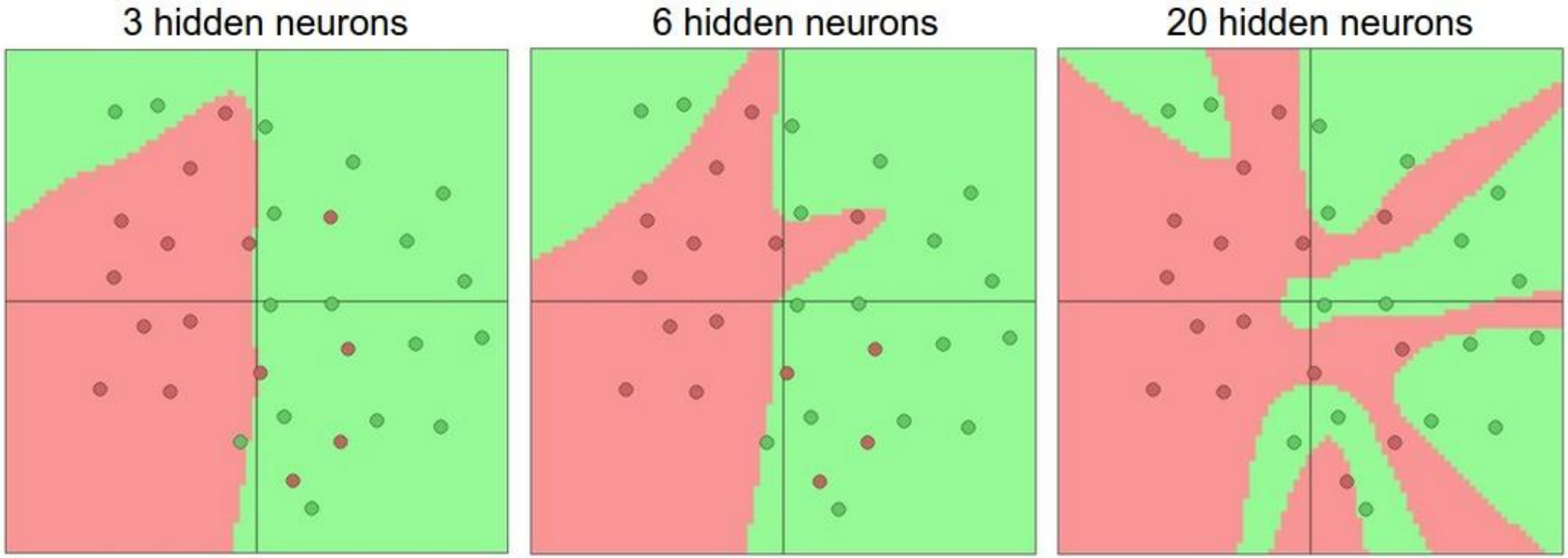
- Now, note that:

$$(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda} = diag\left(\frac{\lambda_i}{\lambda_i + \alpha}\right)$$

- It penalized eignevectors of $\boldsymbol{H}$, preserve large eigenvalue and dimmish small one.

L2 Regularization - Example

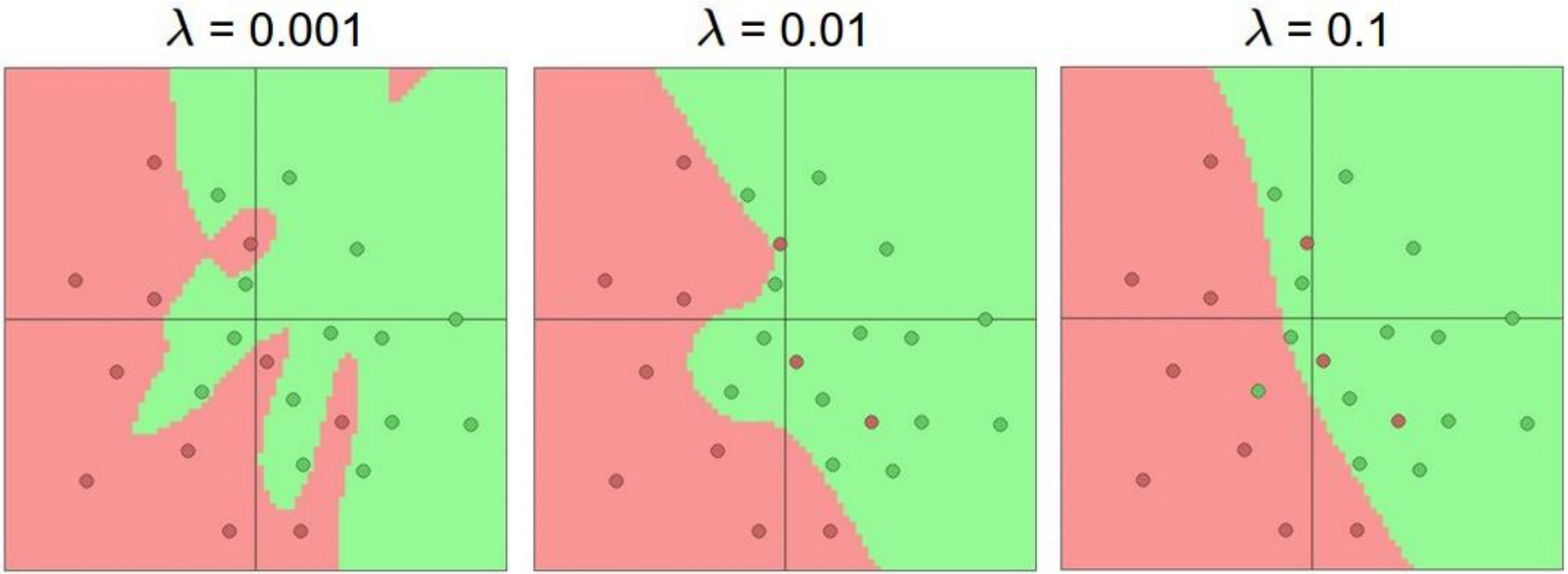
- Consider a two-layers MLP with different hidden neurons:



Run more example here: https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

L2 Regularization - Example

- Now add L2 regularization term (20 hidden neurons), here λ is regularization weight:



Run more example here: https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

L2 Regularization - Analysis

- Numerical Examples:

$$\boldsymbol{H} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \Rightarrow \boldsymbol{Q\Lambda^{-1}\Lambda Q^T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \tilde{\boldsymbol{w}} = \boldsymbol{w}^*$$

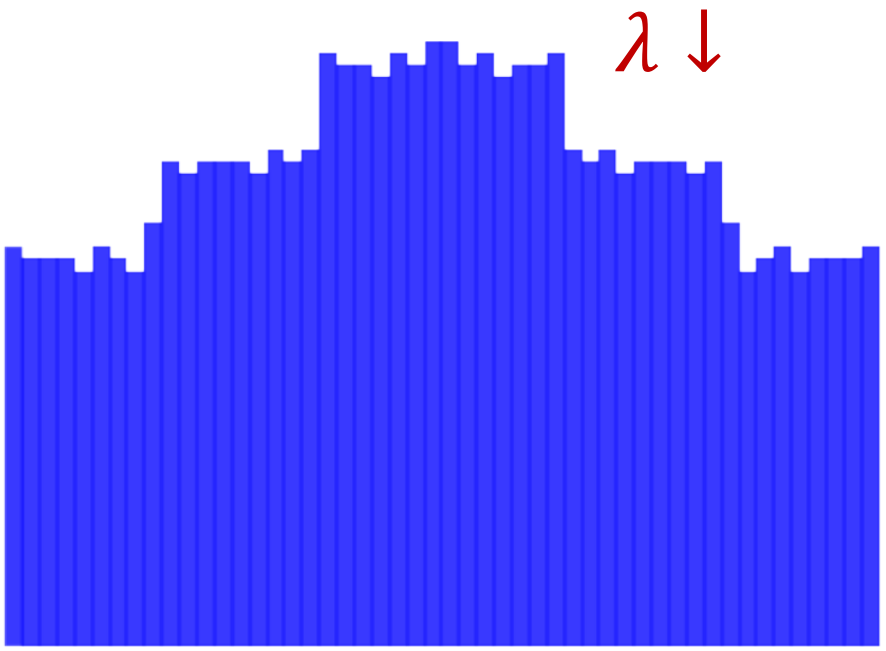
$$\boldsymbol{\Lambda} = \begin{bmatrix} 0.5858 & 0 & 0 \\ 0 & 2.000 & 0 \\ 0 & 0 & 3.4142 \end{bmatrix} \Rightarrow \boldsymbol{\Lambda^{-1}\Lambda} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, (\boldsymbol{\Lambda} + \mathbf{2}\boldsymbol{I})^{-1}\boldsymbol{\Lambda} = \begin{bmatrix} 0.2265 & 0 & 0 \\ 0 & 0.5000 & 0 \\ 0 & 0 & 0.6306 \end{bmatrix}$$

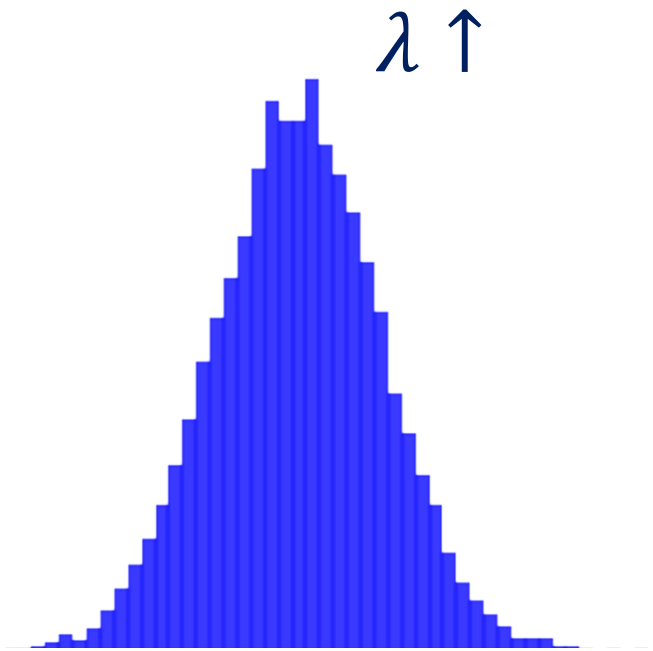
$$\boldsymbol{Q}(\boldsymbol{\Lambda} + \mathbf{2}\boldsymbol{I})^{-1}\boldsymbol{\Lambda Q^T} = \begin{bmatrix} 0.46 & -0.14 & -0.04 \\ -0.14 & 0.43 & -0.14 \\ -0.04 & -0.14 & 0.46 \end{bmatrix} \Rightarrow \tilde{\boldsymbol{w}} = \begin{bmatrix} 0.46 & -0.14 & -0.04 \\ -0.14 & 0.43 & -0.14 \\ -0.04 & -0.14 & 0.46 \end{bmatrix} \boldsymbol{w}^*$$

$$\boldsymbol{w}^* = [0.81 \quad 0.91 \quad 0.13]^T \Longrightarrow \tilde{\boldsymbol{w}} = [0.24 \quad 0.2597 \quad -0.1]^T, \|\boldsymbol{w}^*\|_2 = 1.2252, \|\tilde{\boldsymbol{w}}\|_2 = 0.3675$$

L2 Regularization – Lambda Effect

- Weights Histogram





L2 Regularization – Regression Analysis

- In Linear Regression:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2$$

- It is easy to show (without regularization):

$$\boldsymbol{w}^* = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

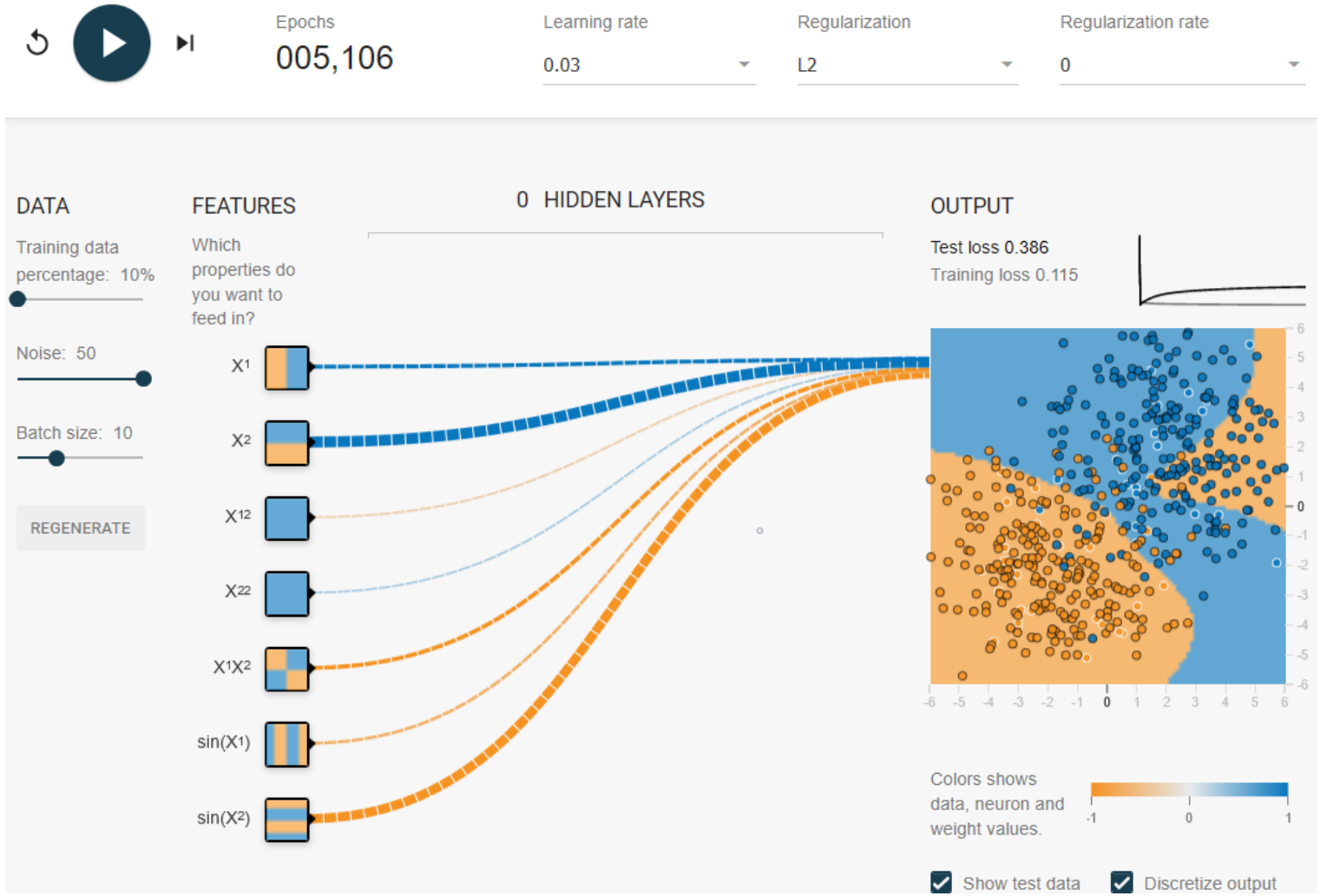
- And (with regularization):

$$\tilde{\boldsymbol{w}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

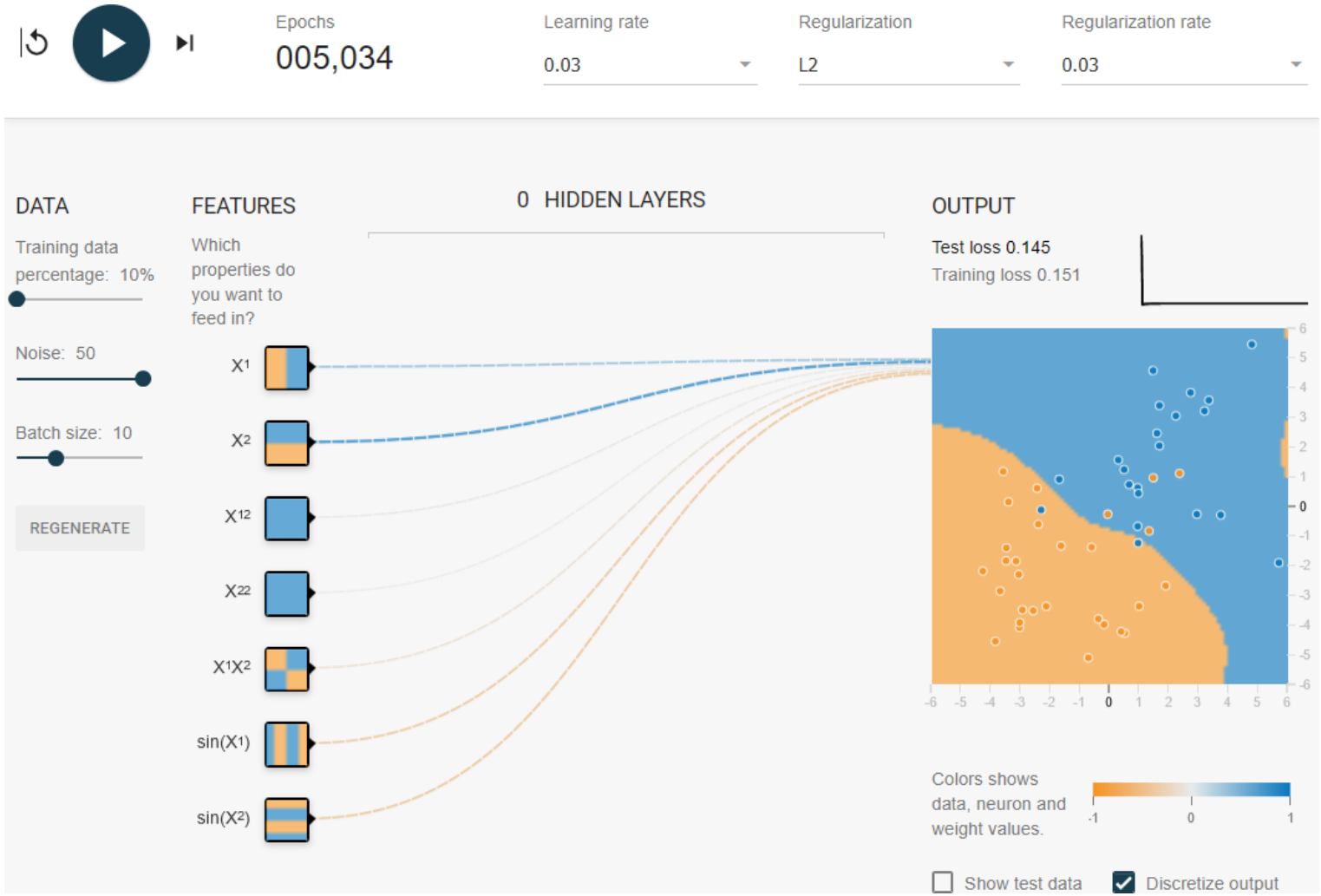
- Solve *multicollinearity* issue ($\boldsymbol{X}^T\boldsymbol{X}$ is near-singular matrix).

L2 Regularization - Example

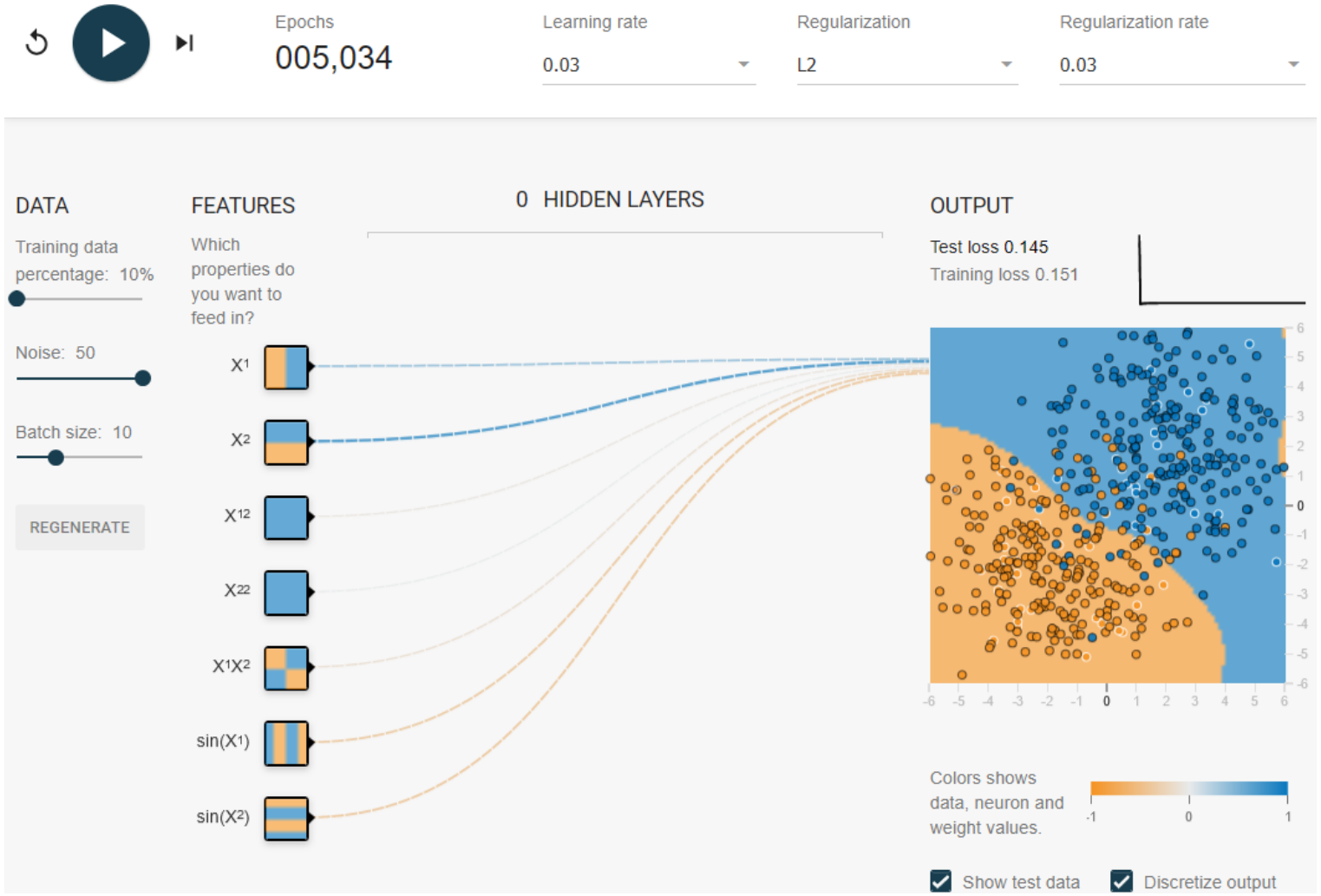
L2 Regularization - Example



L2 Regularization - Example

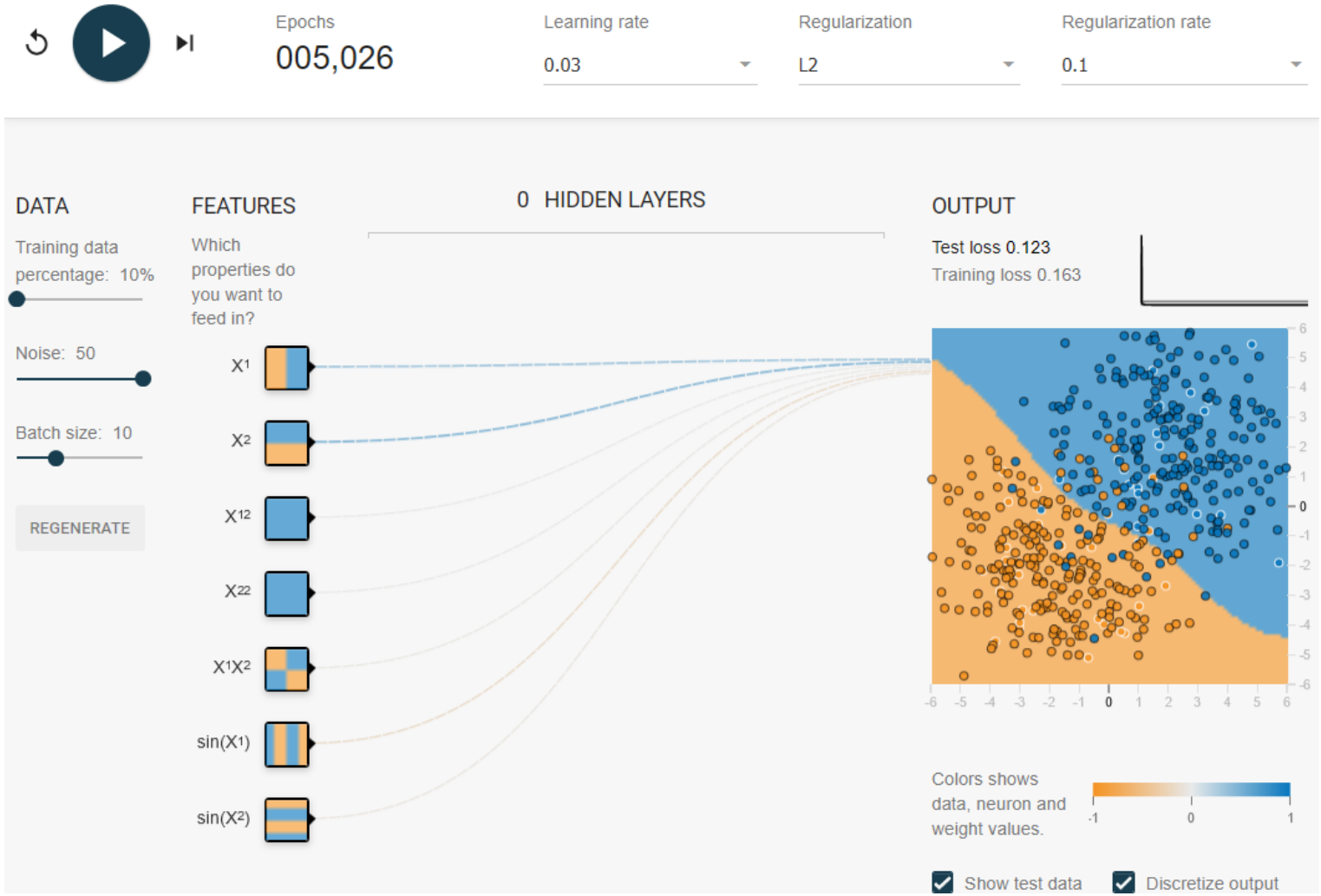


L2 Regularization - Example



L2 Regularization - Example

L2 Regularization - Example



L1 Regularization (Lasso or Basis Pursuit)

- Formulation:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Then:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha sign(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}))$$

- Assume *diagonal* Hessian matrix, $\boldsymbol{H}$, 2nd order approximation around $\boldsymbol{w}^*$:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + (\boldsymbol{w} - \boldsymbol{w}^*)^\top \underbrace{\nabla J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y})}_{0} + \sum_i \left[\frac{1}{2}\mathbf{H}_{i,i}(w_i - w_i^*)^2 + \alpha |w_i|\right]$$

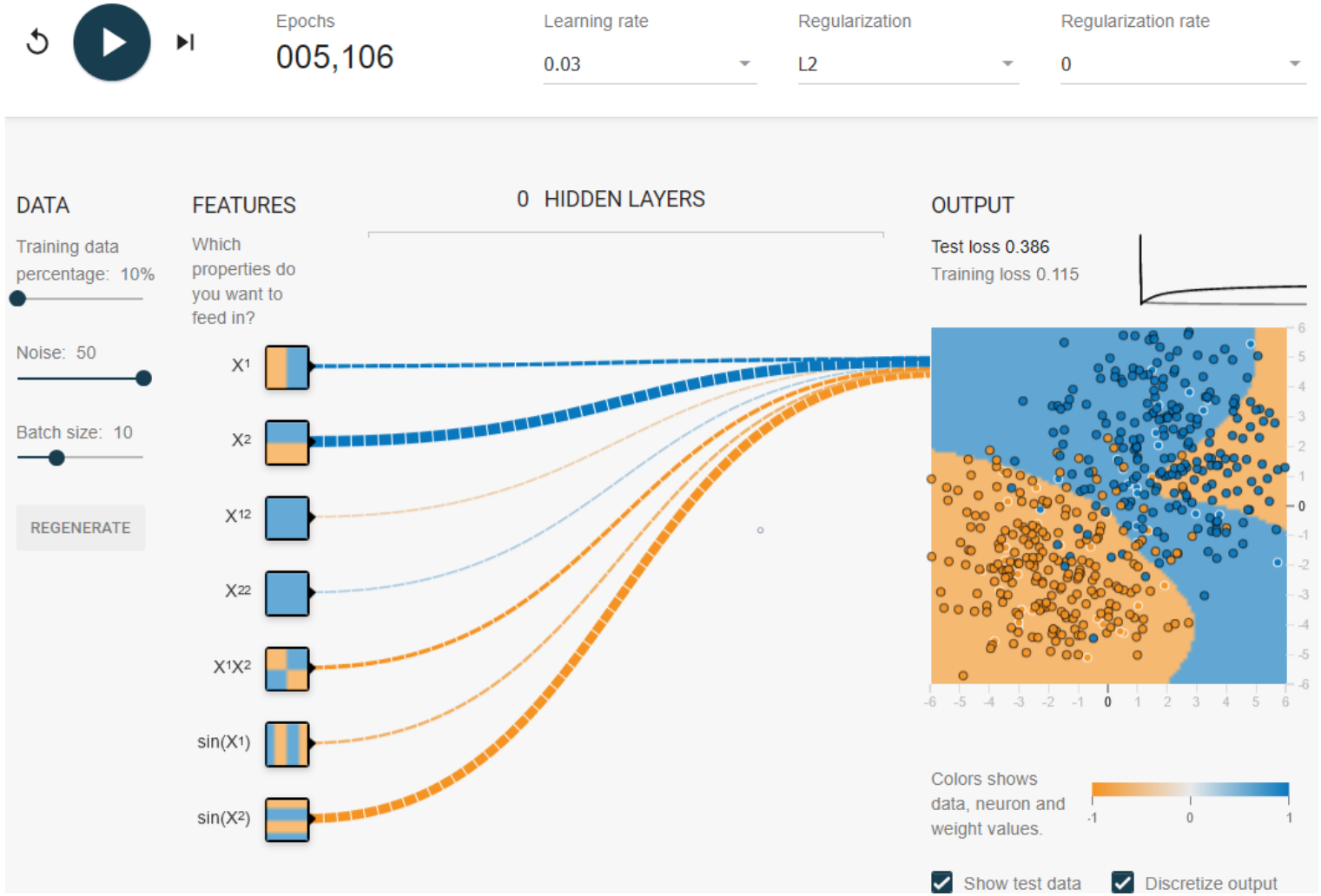
- Exact solution is:

$$w_i = sign(w_i^*) max\left\{|w_i^*| - \frac{\alpha}{\mathbf{H}_{i,i}}, 0\right\}$$

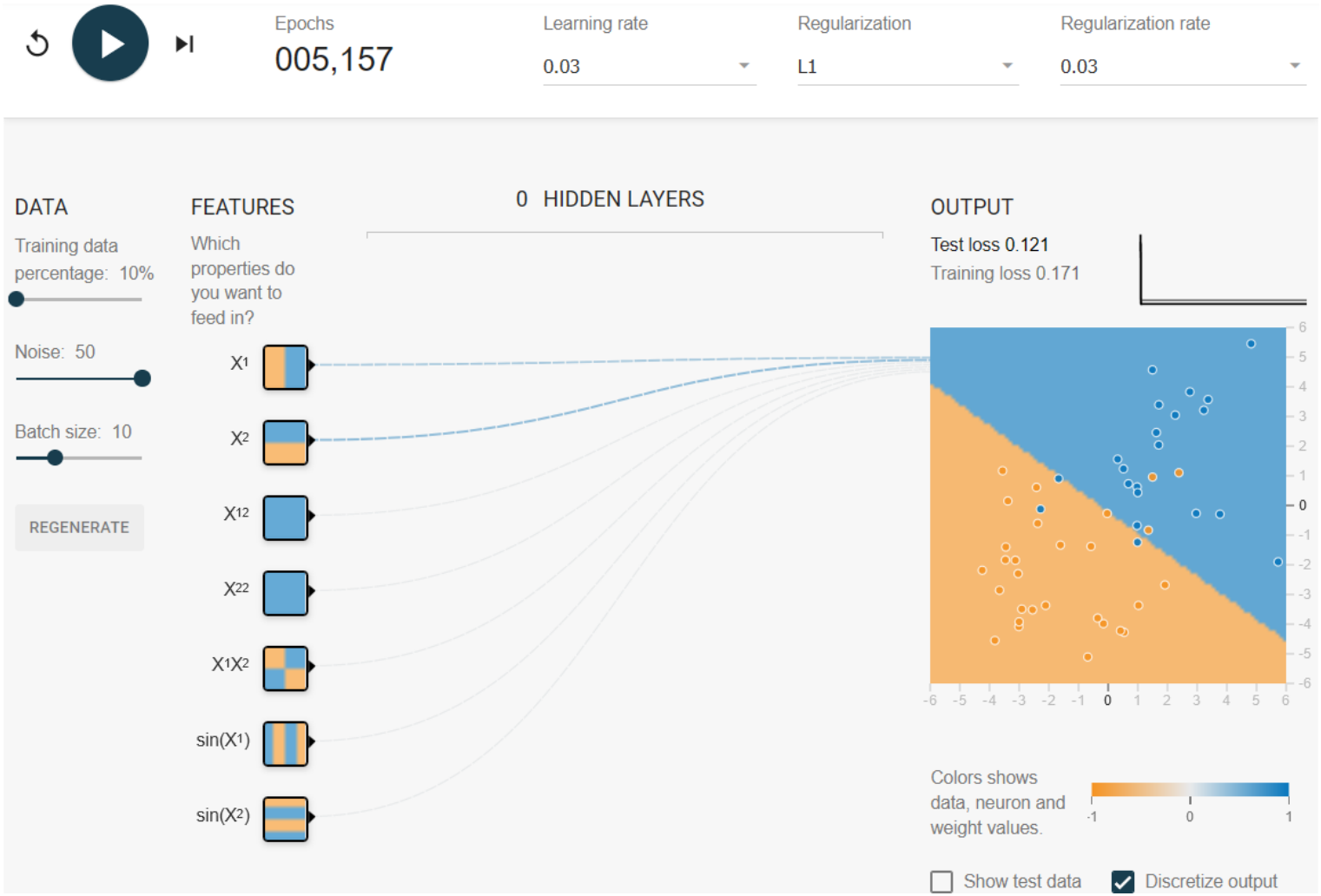
- This is weight sparsifier!

L1 Regularization - Example

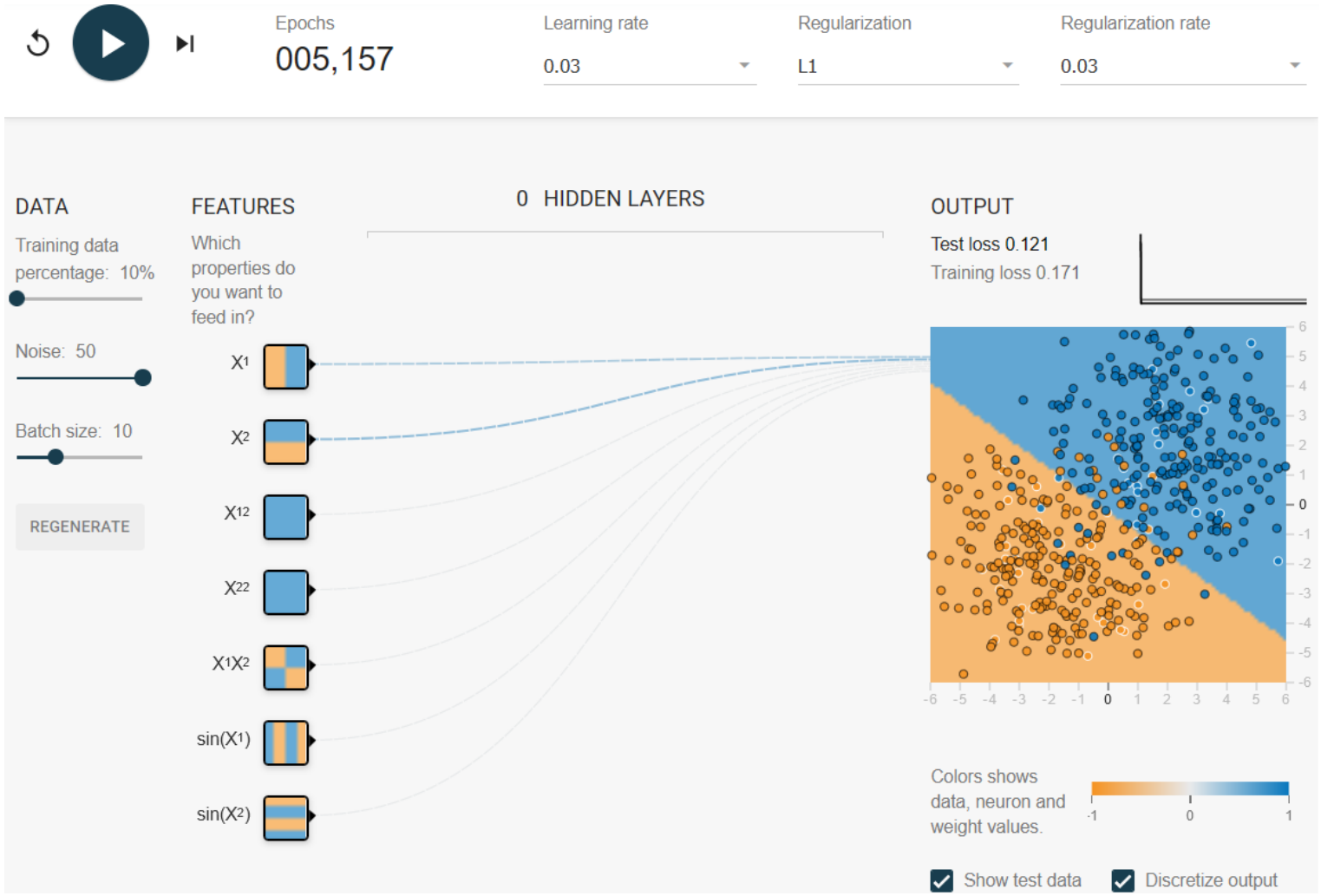
L1 Regularization - Example



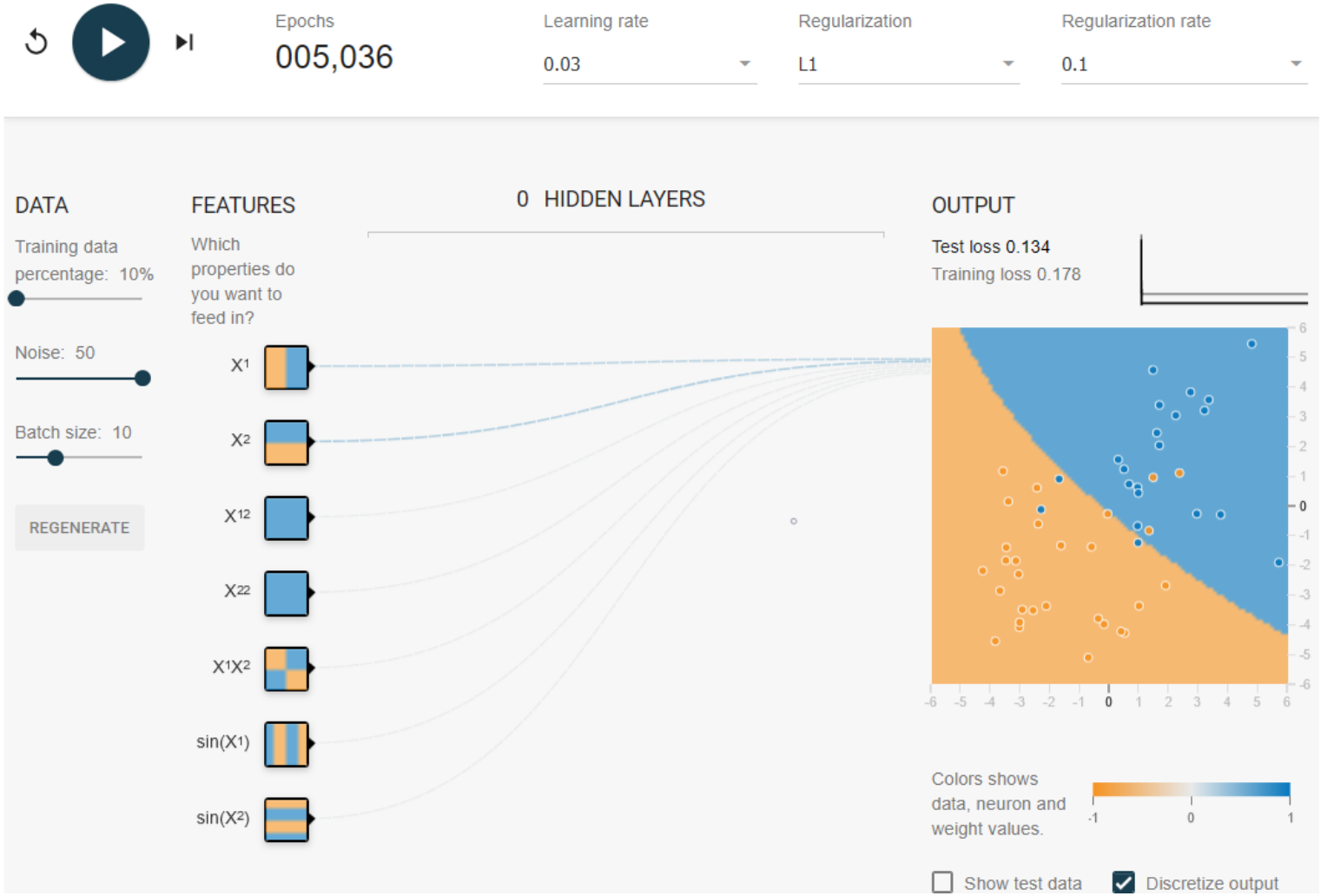
L1 Regularization - Example



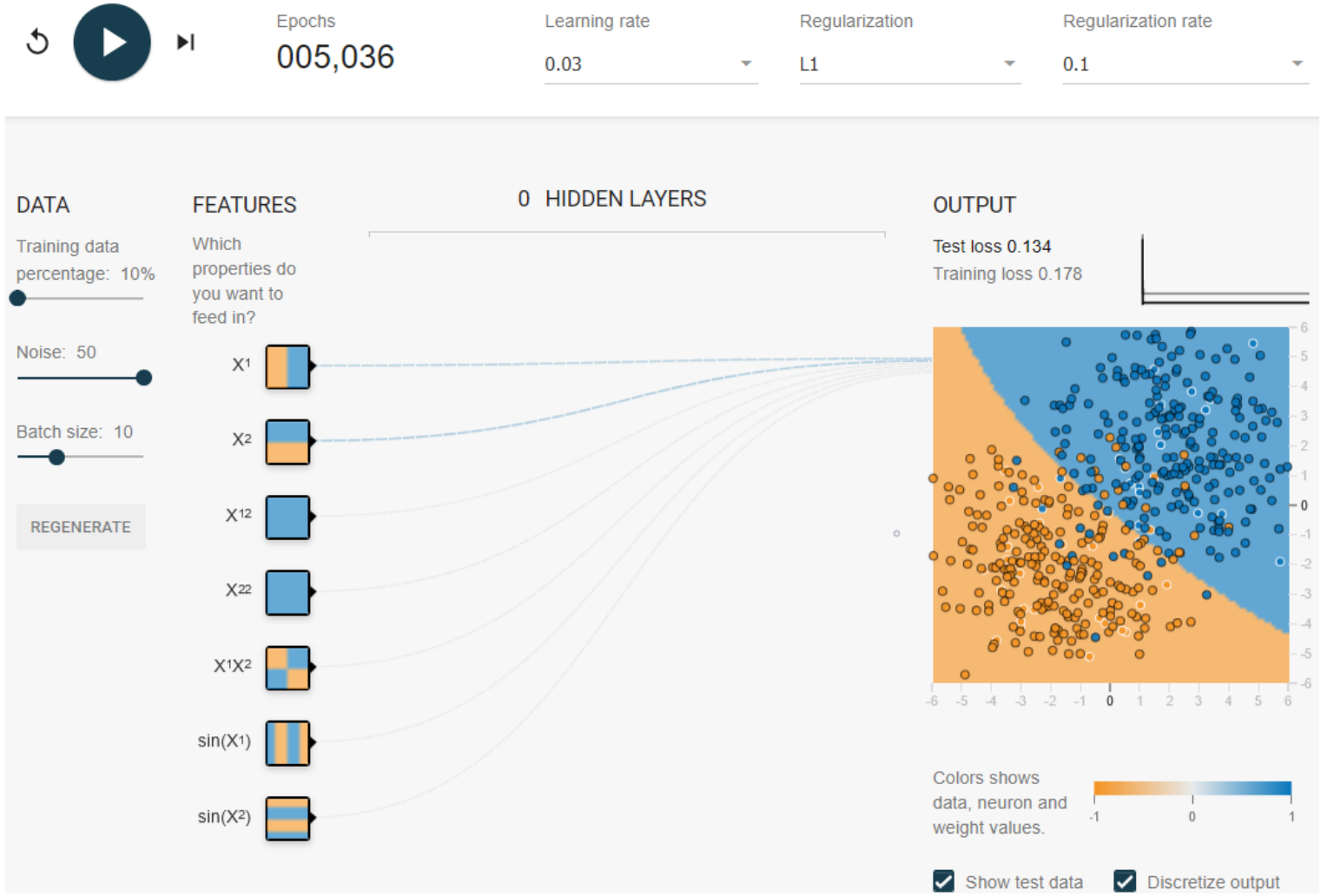
L1 Regularization - Example



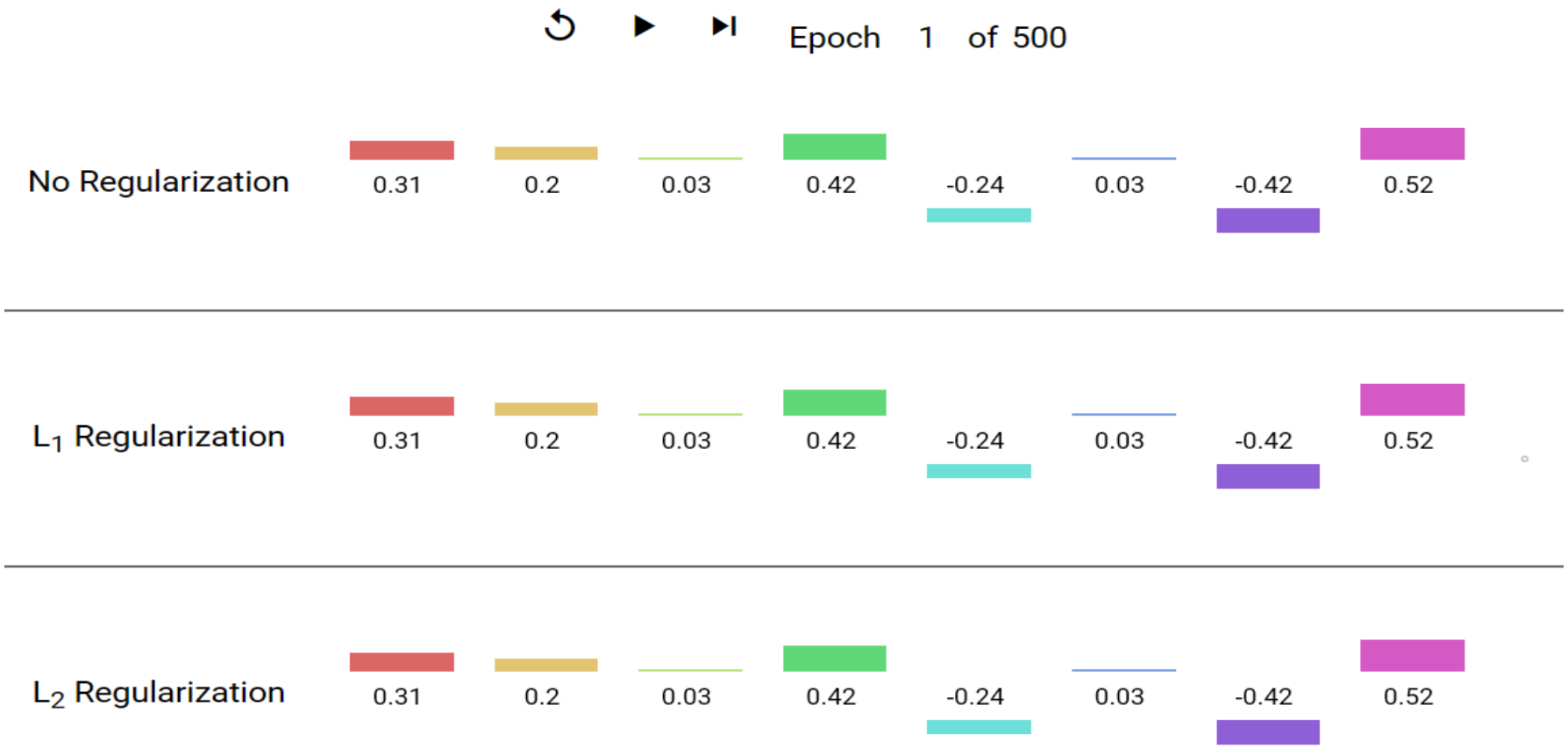
L1 Regularization - Example



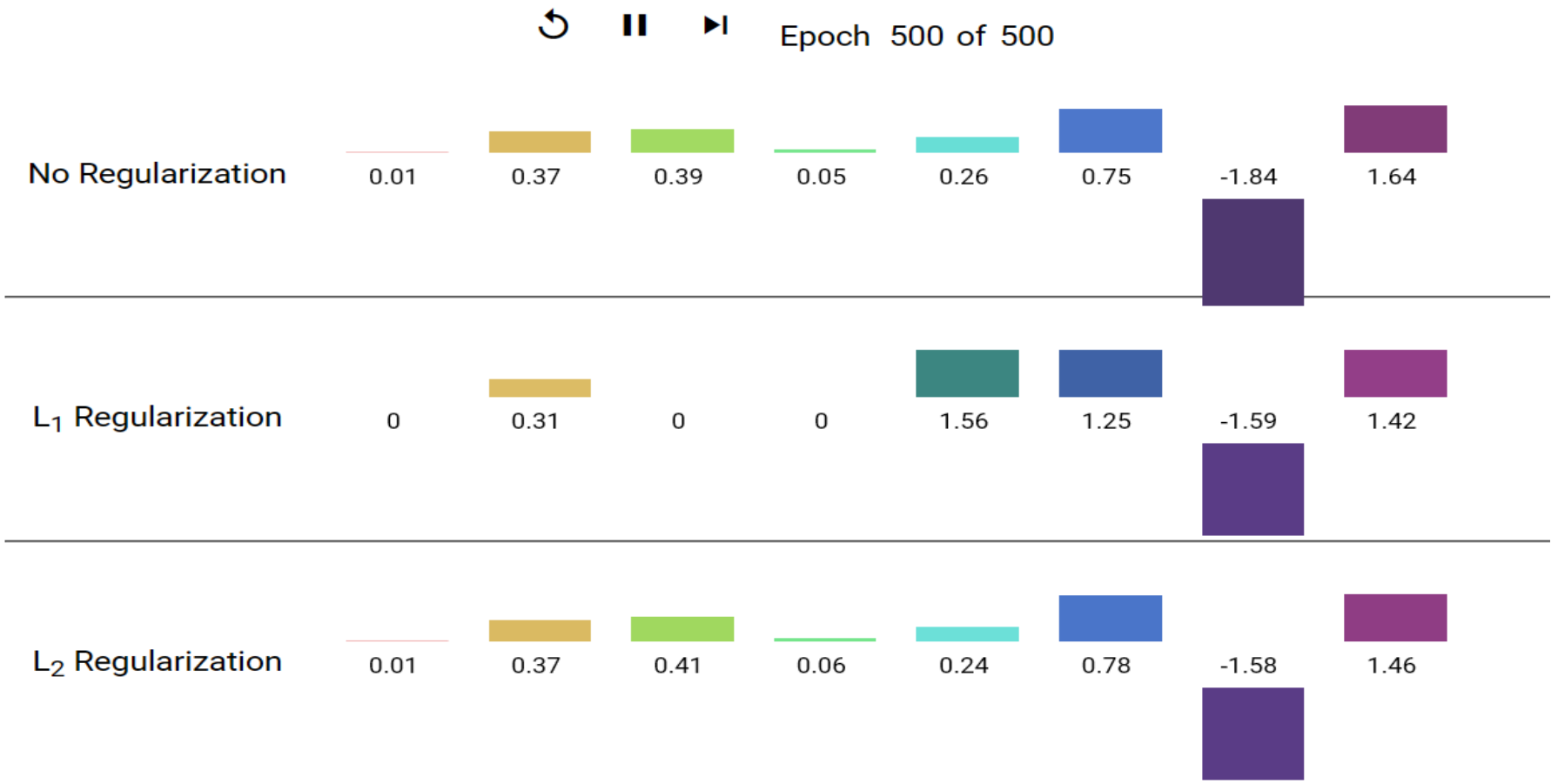
L1 Regularization - Example



Regularization Effect
https://developers.google.com/machine-learning/crash-course/



Regularization Effect
https://developers.google.com/machine-learning/crash-course/



Elastic Net Regularization

- L1 regularization can occasionally produce non-unique solutions

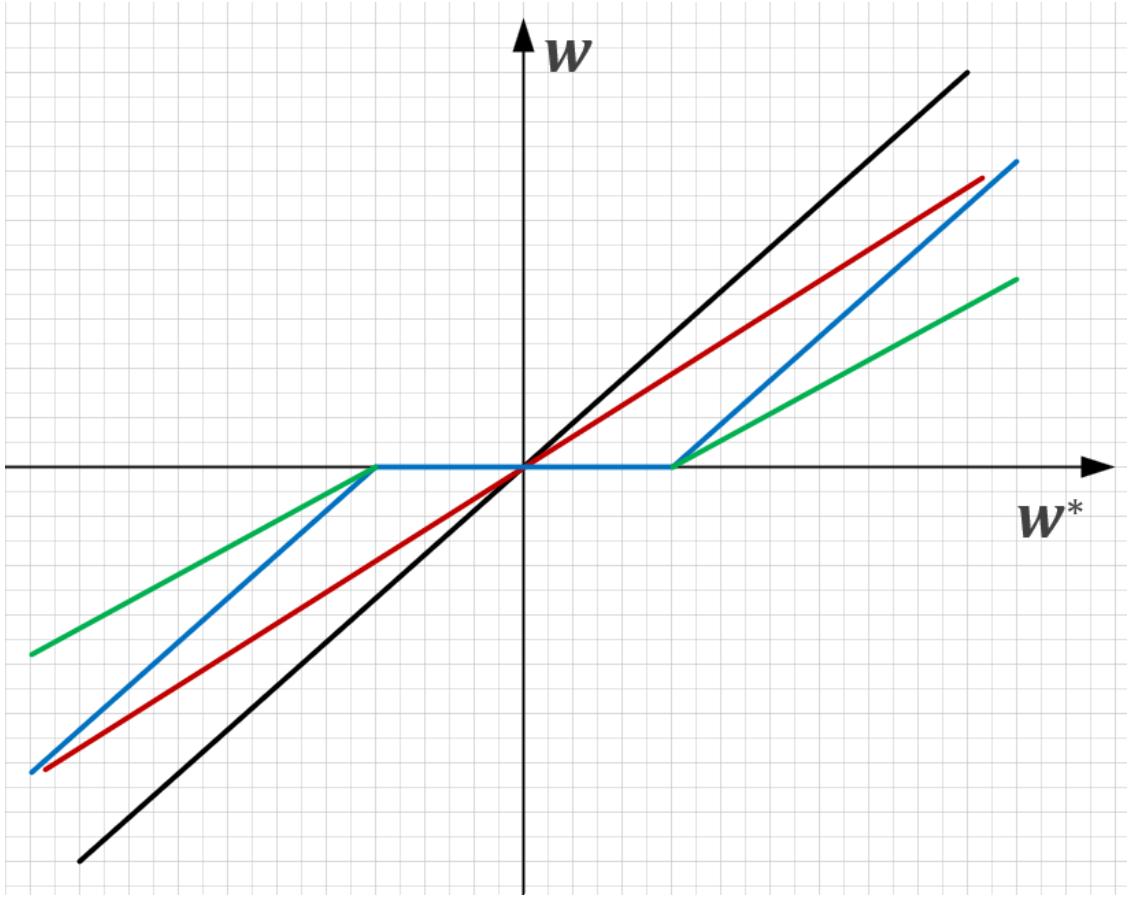
- L1+L2 Regularization:

$$J_{Elastic} = (MSE/CE) + \lambda \left(\alpha \sum_{Weights \backslash Bias} \|\boldsymbol{w}\|_2^2 + (1-\alpha) \sum_{Weights \backslash Bias} \|\boldsymbol{w}\|_1 \right)$$

- Ref: Regularization and Variable Selection via the Elastic Net, Zou and Hastie, 2004.

L1 vs L2 vs Elastic Net

- No Regularization
- Ridge (L2) Regularization
- Lasso (L1) Regularization
- Elastic Net Regularization



Geometric Perspective

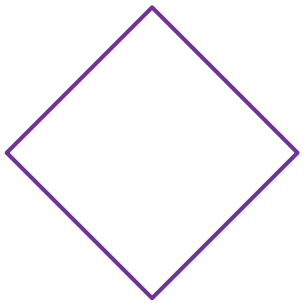
- The Problems:

$$minimize\big(Loss(Data|Model) + \lambda\|\boldsymbol{w}\|_p\big)$$

- is equivalent to:

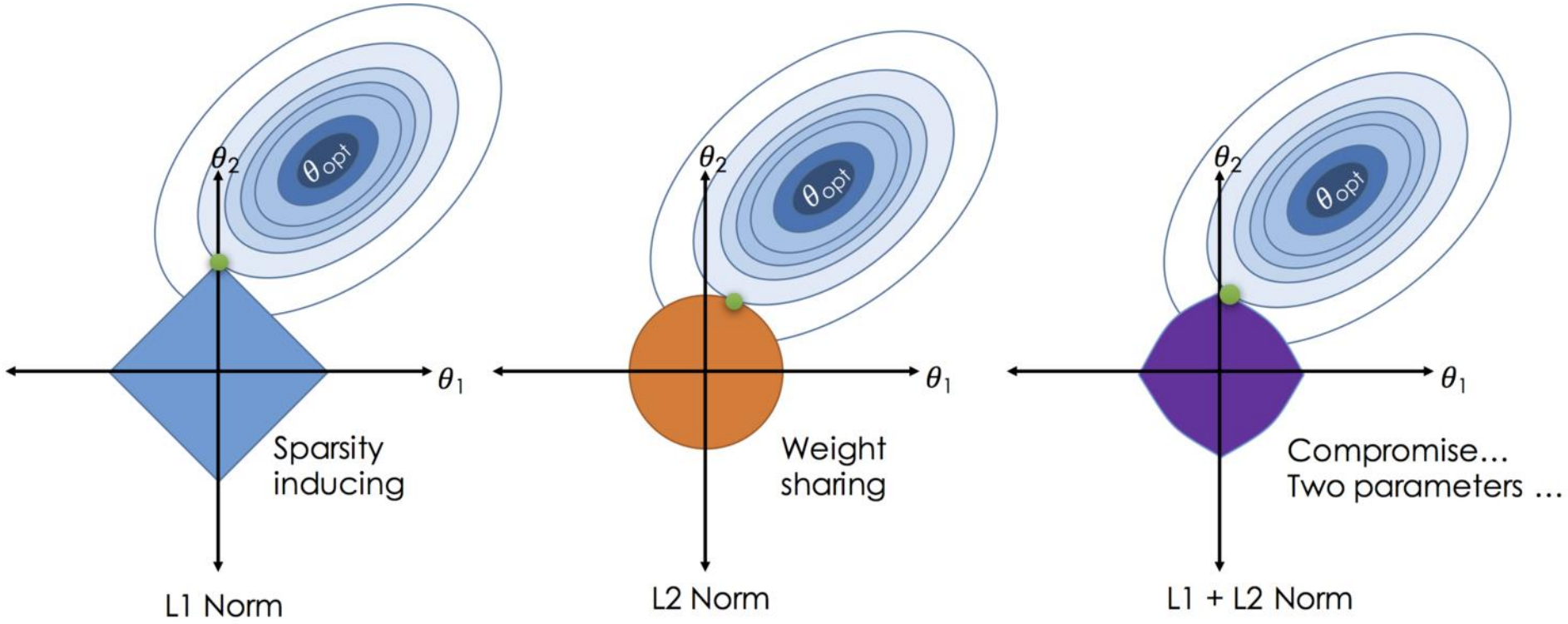
$$minimize\big(Loss(Data|Model)\big), \qquad s.t.\ \|\boldsymbol{w}\|_p \leq \eta$$

- L1 non-unique solution (rare)



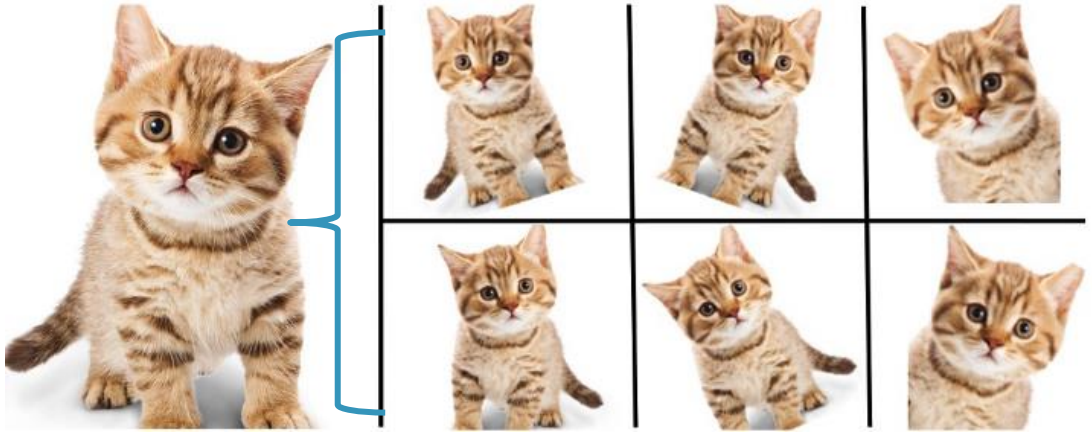
Geometric Perspective

- Illustration:



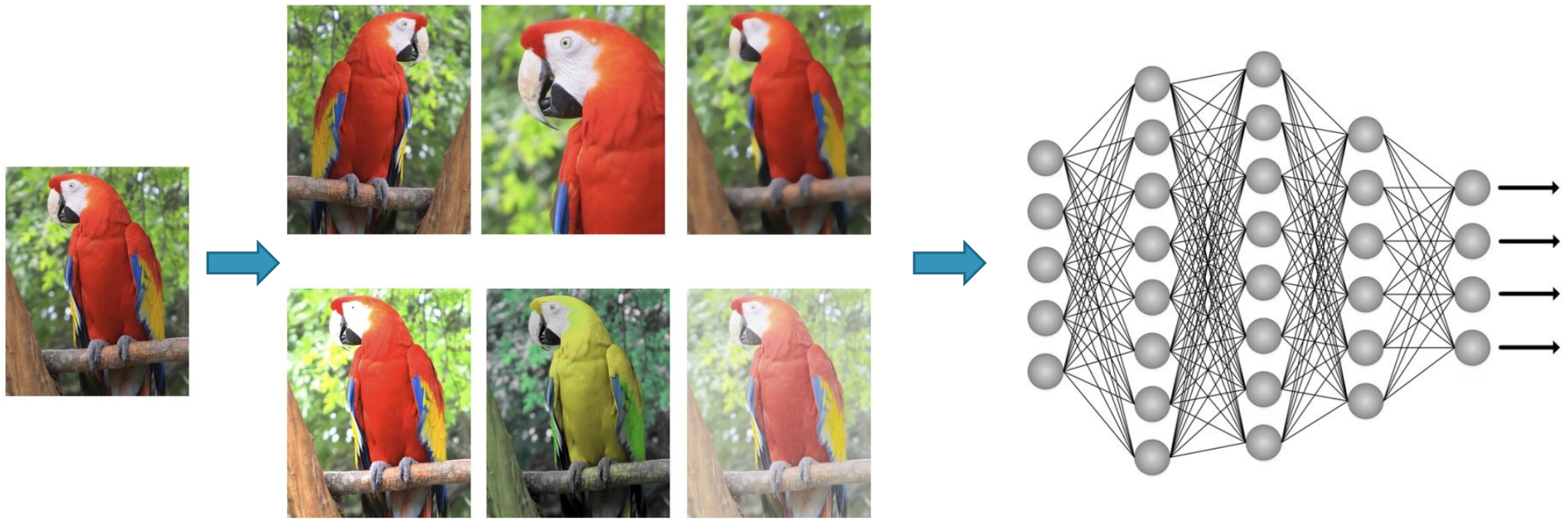
Data Augmentation

- Idea:
  - Invariance (Rotation, Scaling, Translation, Shearing, Mirroring, …) and Regularization



Data Augmentation

- How use it:

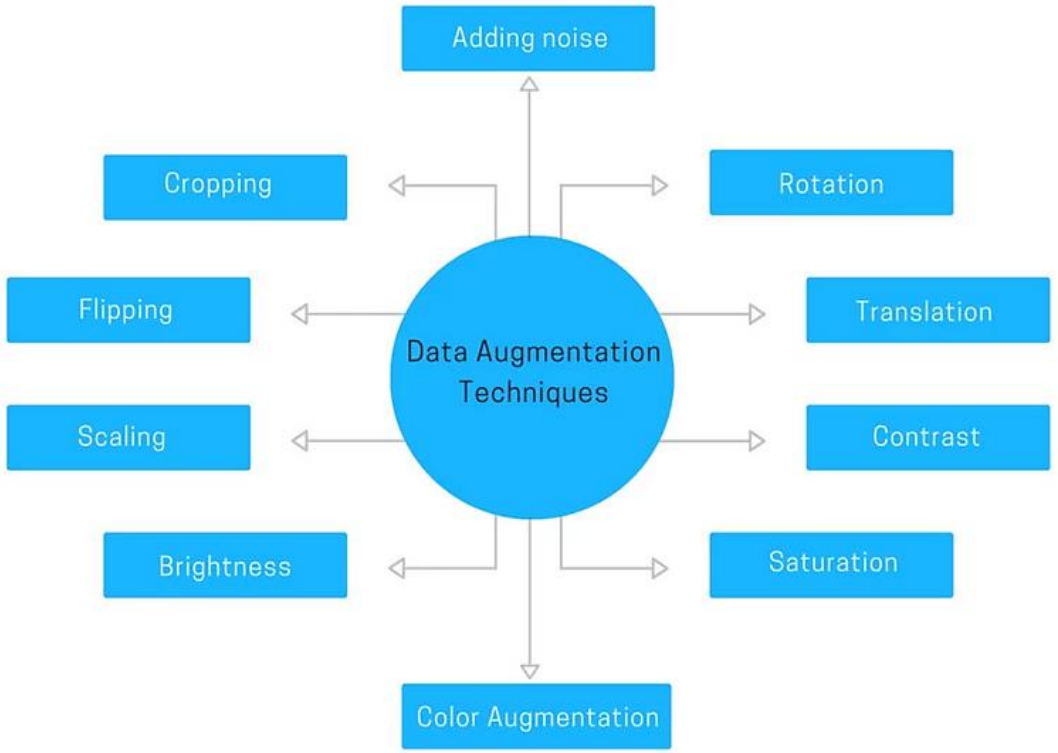


Data Augmentation - Image

- Stochastic Transform:
  - Horizontal/Vertical Flip (Mirroring), Translation, Scaling; Rotation; Shearing (Skewing);
  - Contrast/Brightness/illumination/Color/… manipulation
  - Random Crop and then Scale
  - Random Cutout or Mask (Blackout a random rectangular/square)
  - Additive/Multiplicative Noise
  - Image stitching
- Modern Approach:
  - Deep Generated Data, Augmented Network (DeepFake, …)
  - Neural style transfer

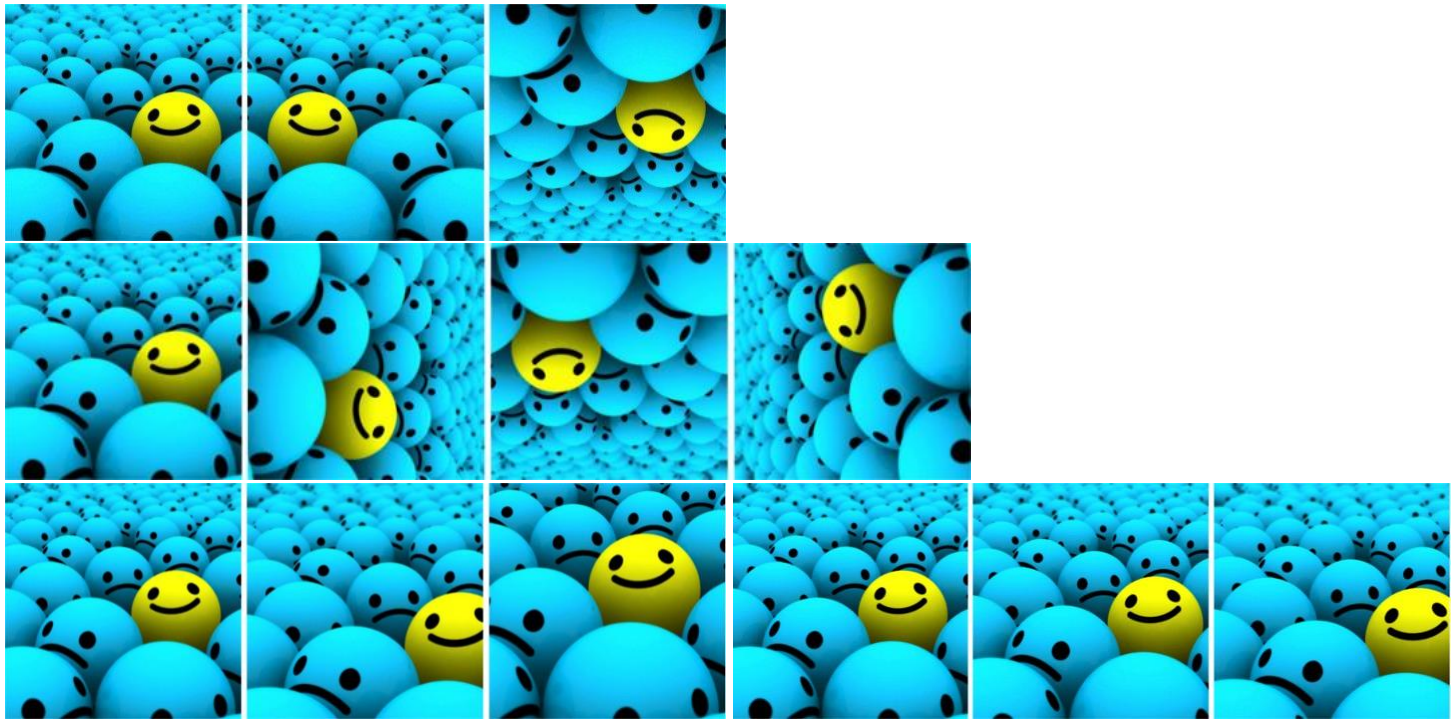
Data Augmentation - Image

- Stochastic Transform:



Data Augmentation - Image

- Examples #1:



Data Augmentation - Image

- Examples #2:

Data Augmentation - Image

- Examples #3:

Data Augmentation - Image

- Examples #4:

Data Augmentation - Image

- Examples #4:

Data Augmentation - NLP

- Easy Data Augmentation (EDA) operations:
  - Synonym replacement,
  - Word insertion,
  - word swap,
  - word deletion.
- Back translation:
  - Re-translating text from the target language back to its original language.
- Text Generation
- Shuffle Sentences Transform
- See https://neptune.ai/blog/data-augmentation-nlp for details

Data Augmentation - Speech

- Noise Injection
- Change playing speed
- Special Effects
- Speech Generation

Data Augmentation

- A Good survey for image
- Image Data Augmentation Approaches: A Comprehensive Survey and Future directions (arXiv:2301.02830v4)

Regularization – Dropout

- Main article: - Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava, 2014
- Preliminary:
  - Ensemble Methods: Learn multiple models (Weak Classifier) to solve the same problem and then combine (aggregate) them for better results
  - Combination of “Different/Same” “Model/Dataset/Features/Objective/ Optimization/Initials”
- Policies:
  - Bagging
  - Boosting
  - Stacking

Regularization – Dropout

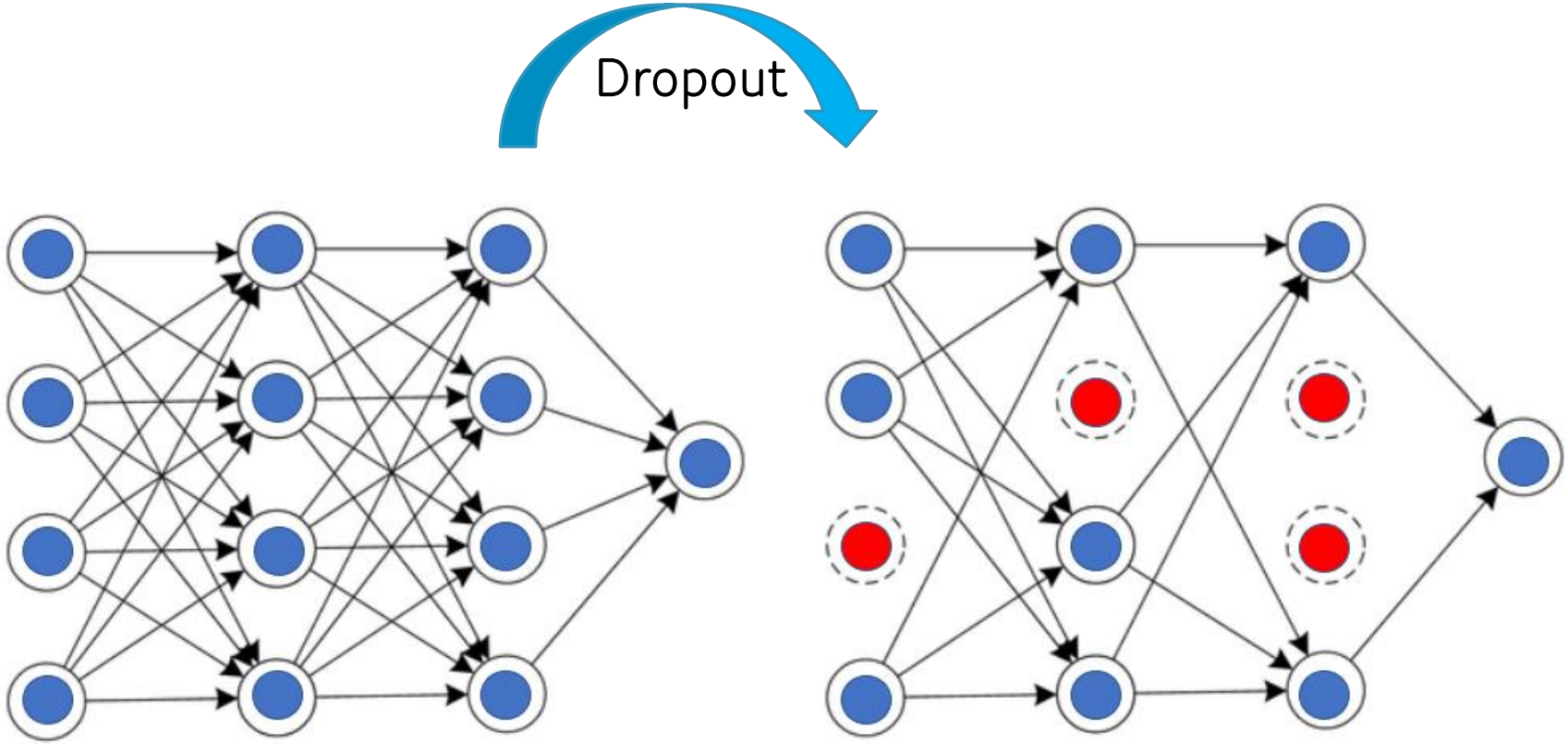
- Bagging (Bootstrap aggregating):
  - K new training datasets are generated by uniformly random sampling with replacement (Bootstrapping) from the original dataset.
  - Train K model independently
  - Combine/Aggregate the classifier/regressor using:
    - Averaging (Arithmetic, Geometric, …)
    - Voting
- Boosting: Each data and classifier has its own weight (determined with boosting procedure), AdaBoost is most known methods.
- Stacking: Train K models using all data, then train a combiner/aggregator classifier to combine K result, as best as possible.

Regularization – Dropout

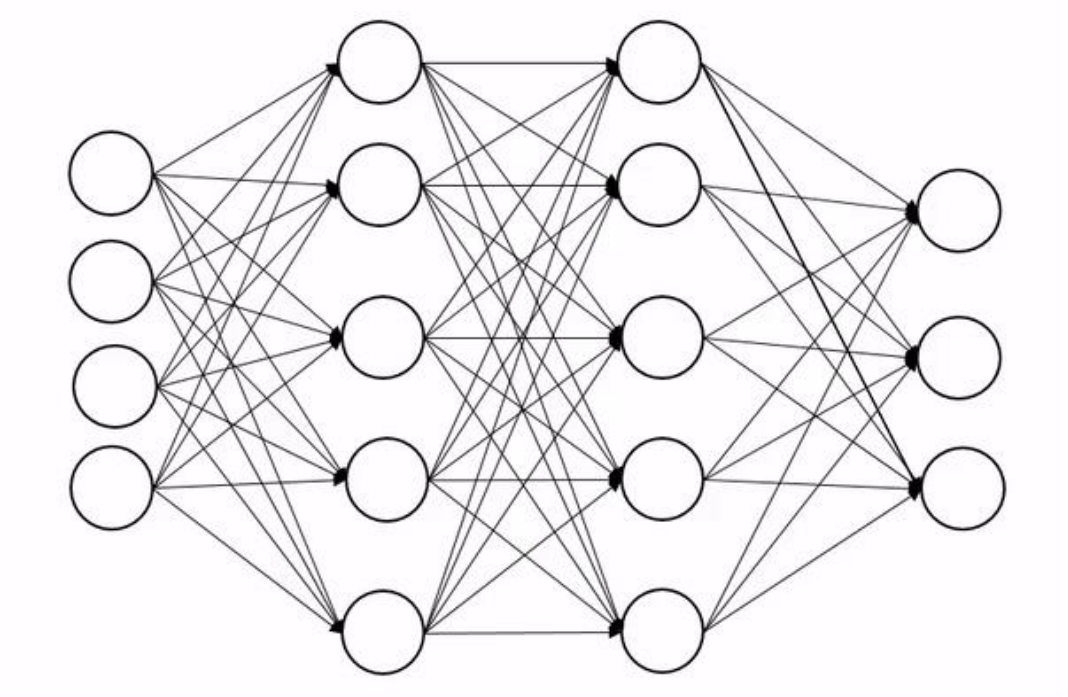
- Motivation:
  - Co-Adaptation:
  - If two or more neurons extract the same feature repeatedly or highly correlated behavior (co-adaptation), the network isn't reaching its full capability.

Regularization – Dropout

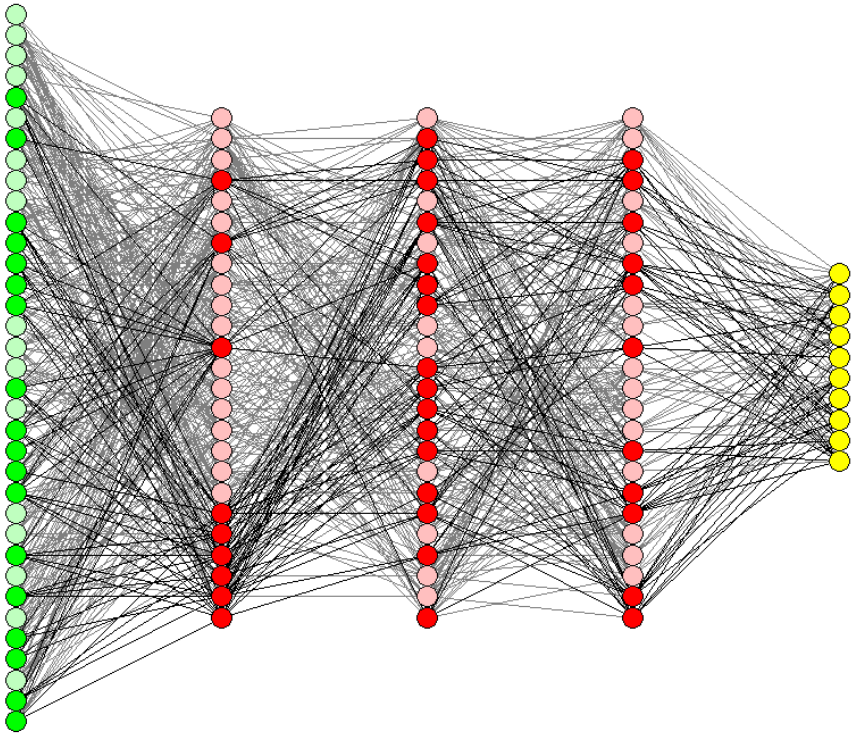
- Solution: Update (EBP) a fraction (Randomly Chosen) of all weights in each iteration!



Regularization – Dropout



Regularization – Dropout

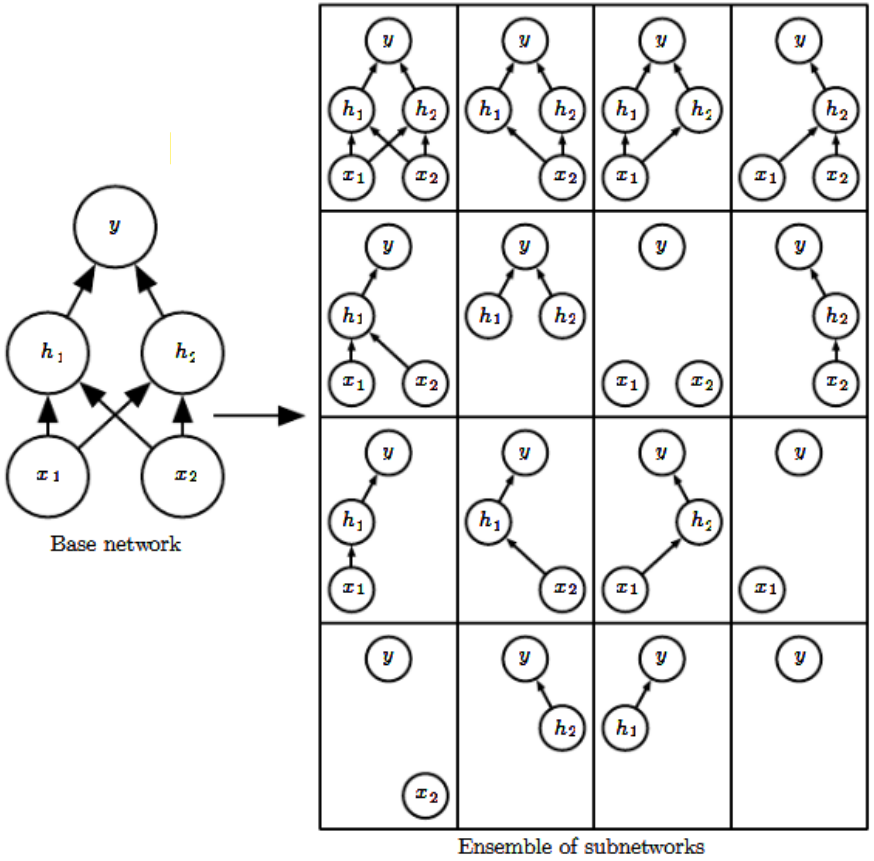


Regularization – Dropout

- Training Phase:
  - For each hidden layer, for each training sample, for each iteration, dropout (zero out) a random fraction, $(1 - p)$, of neuron (and corresponding weight).
    - Input Layer: $P > 0.8$ or $P = 1.0$
    - Hidden Layer: $P = 0.5$
    - Output Layer: $P = 1.0$
- Test Phase:
  - Use all activations, but reduce them by a factor $(1 - p)$ to account for the missing activations during training (neuron output multiplied by p).

Regularization – Dropout

- Dropout Training Scheme



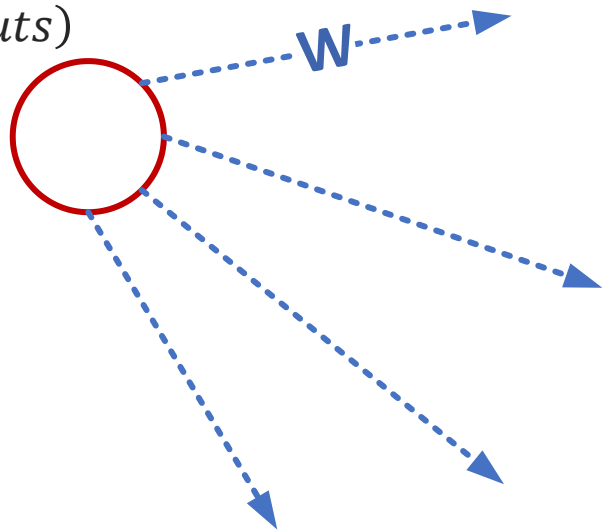
Regularization – Direct Dropout

- Dropout Training/Test Scheme:

Present with probability p

$O_i = X_i f(inputs)$

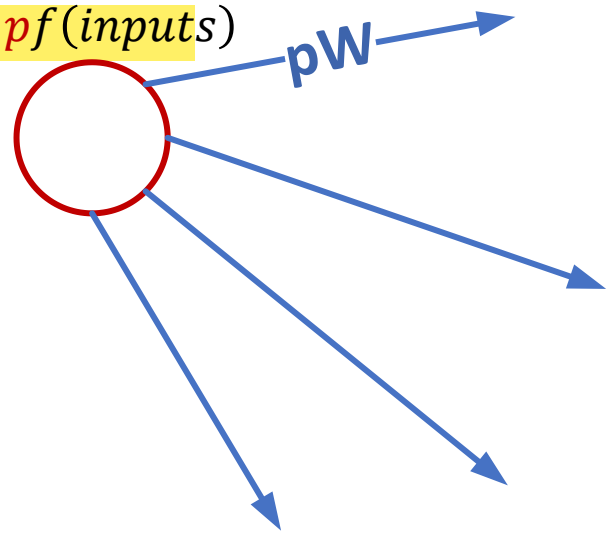
$X_i \in \{0.1\}$



Training Phase

Always Present

$O_i = pf(inputs)$



Test Phase

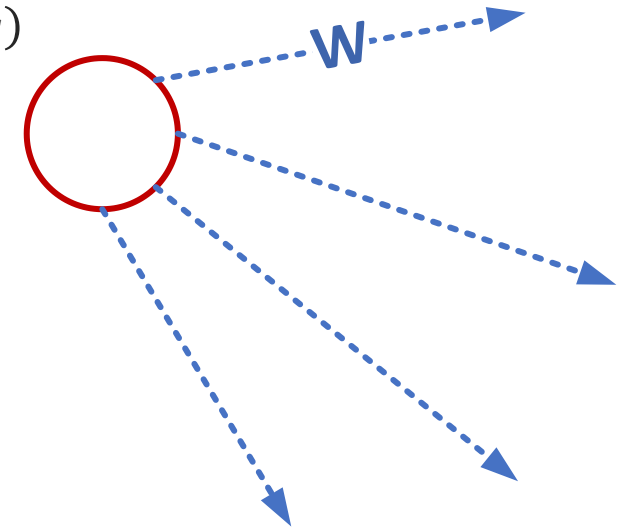
Regularization – Inverted Dropout

- In this approach a debiasing techniques is used:

Present with probability p

$$O_i = \frac{1}{p} X_i f(inputs)$$

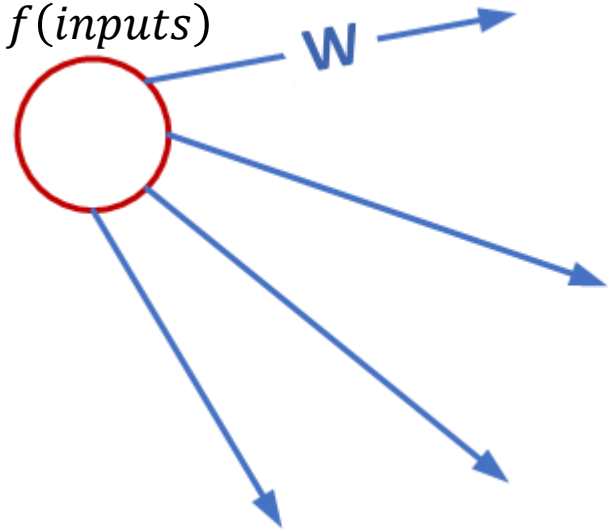
$$X_i \in \{0.1\}$$



Training Phase

Always Present

$$O_i = f(inputs)$$



Test Phase

Dropout Mathematics

- Mathematics of Dropout (Understanding Dropout, Baldi, NIPS2013)
- Dropping out is modelled by Bernoulli distribution:

$$\delta_i \sim Bernoulli(p)$$

- Complete Network Error (E_N):

$$E_N = \frac{1}{2}(t - \sum_{i=1}^{n} w_i' I_i)^2$$

- Dropout Network Error (E_D):

$$E_D = \frac{1}{2}(t - \sum_{i=1}^{n} \delta_i w_i I_i)^2$$

- Rewrite Complete Network Error (E_N):

$$E_N = \frac{1}{2}(t - \sum_{i=1}^{n} p_i w_i I_i)^2$$

Dropout Mathematics

- Gradient (Dropout Network):

$$\frac{\partial E_D}{\partial w_i} = -t\delta_i I_i + w_i \delta_i^2 I_i^2 + \sum_{j=1, j\neq i}^{n} w_j \delta_i \delta_j I_i I_j$$

- Gradient (Complete Network):

$$\frac{\partial E_N}{\partial w_i} = -t p_i I_i + w_i p_i^2 I_i^2 + \sum_{j=1, j\neq i}^{n} w_j p_i p_j I_i I_j$$

Dropout Mathematics

- Stochastic Gradient (Dropout Network):

$$E\left[\frac{\partial E_D}{\partial w_i}\right] = -tp_i I_i + w_i p_i^2 I_i^2 + w_i \mathrm{Var}(\delta_i) I_i^2 + \sum_{j=1, j\neq i}^{n} w_j p_i p_j I_i I_j$$

$$= \frac{\partial E_N}{\partial w_i} + w_i \mathrm{var}(\delta_i) I_i^2$$

$$= \frac{\partial E_N}{\partial w_i} + w_i p_i (1 - p_i) I_i^2$$

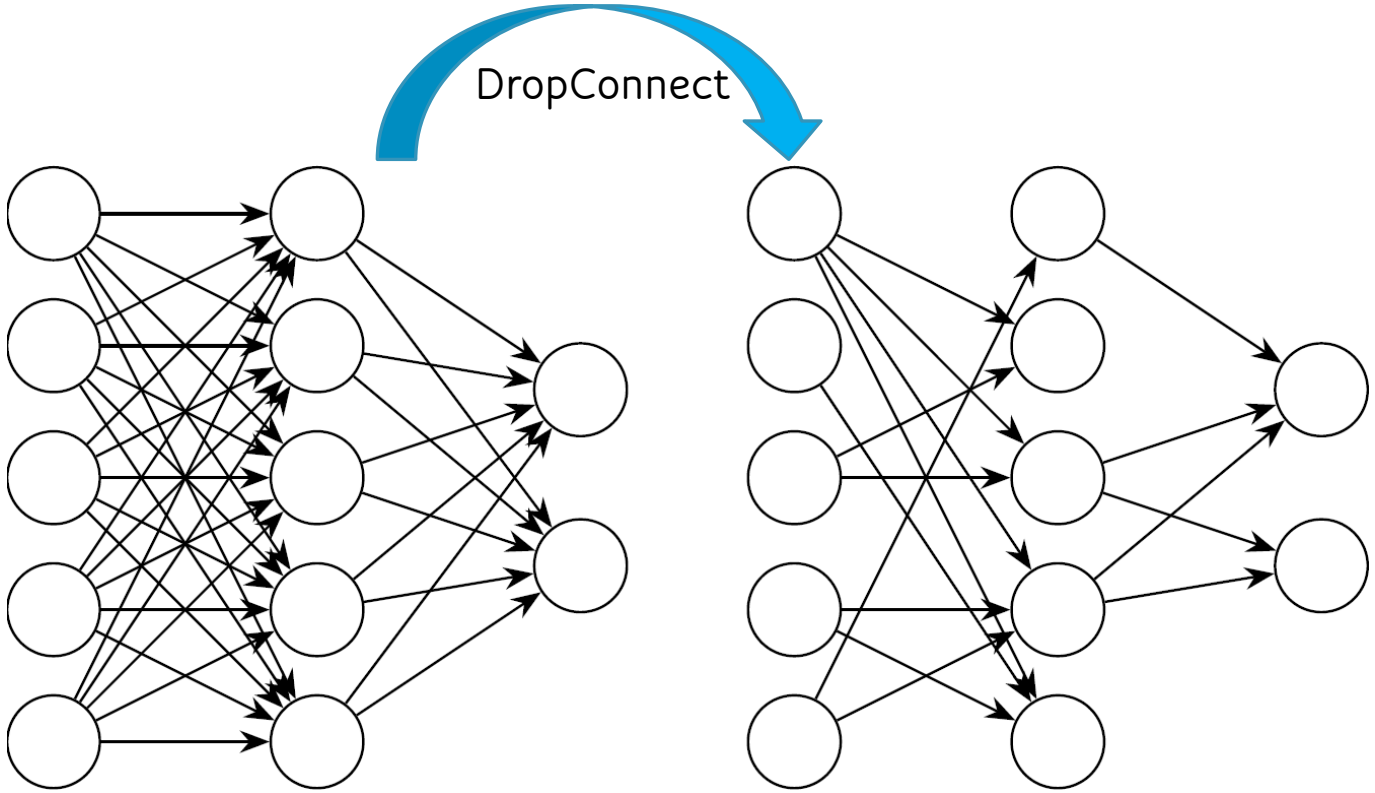
- It is like Regularization:

$$E_R = \frac{1}{2}\left(t - \sum_{i=1}^{n} p_i w_i I_i\right)^2 + \sum_{i=1}^{n} p_i (1 - p_i) w_i^2 I_i^2$$

- p=0.5 is recommended (why?)

Regularization – DropConnect

- Individual weights and biases rather than neuron outputs are set to zero with some probability

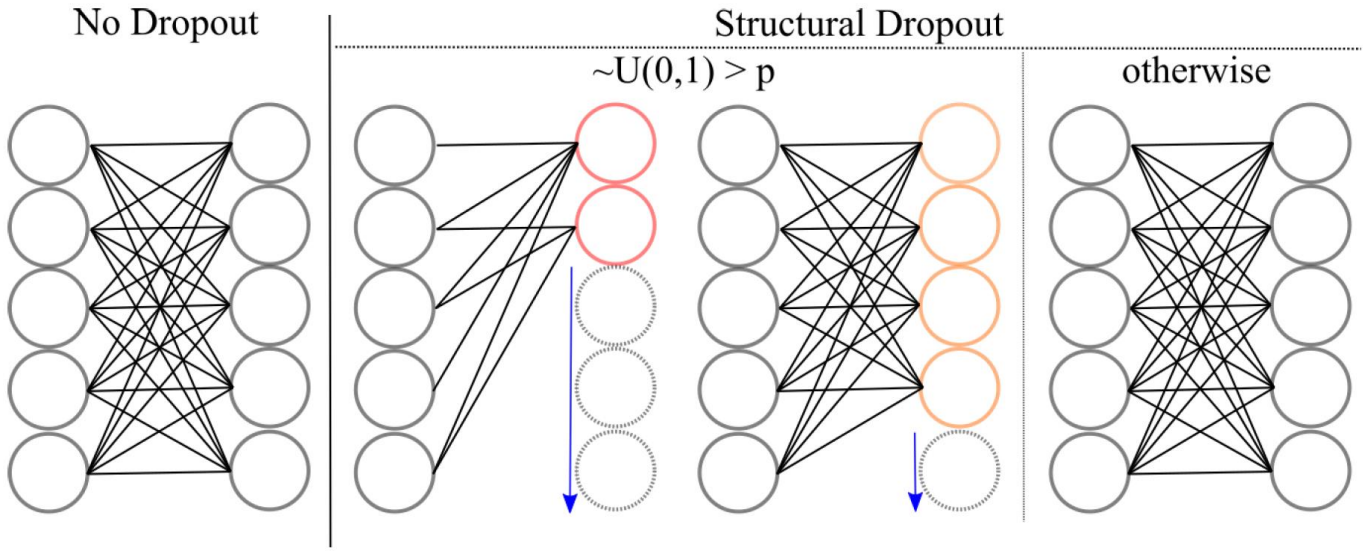


Regularization – Dropout

- Some Variations:
  - Gaussian Dropout Multiplicative: $\delta_i \sim N(1, \sigma^2)$
  - Gaussian Dropout Additive: $\delta_i \sim N(0, \sigma^2)$
  - Fast Dropout*: Dropout implemented as neuron output random sampling from an underlying distribution, which can be approximated by a Gaussian (preserving input mean and variance)
  - Alpha Dropout: Randomly set activation function input to α such that output means and variance kept to 0 and 1, respectively.
  - …

Regularization – Structural Dropout

- It is used for model width compression:
  - Rather than randomly selecting indices to prune, during training Structural Dropout prunes all nodes after a uniformly randomly selected index, and normalizes the expectation based on the number of features dropped.



Mont Carlo Dropout

- Monte Carlo dropout is a dropout method that can estimate model uncertainty.
- Implementation:
  - Train a neural network normally using standard dropout.
  - To perform inference (Test Phase) on an input sample:
    - The network is run T times with standard dropout, all with the same input but with different randomly generated dropout masks each time.
    - Estimators for the mean (model output) and variance of the implicit Bayesian model output (model uncertainty indicator) are given by:

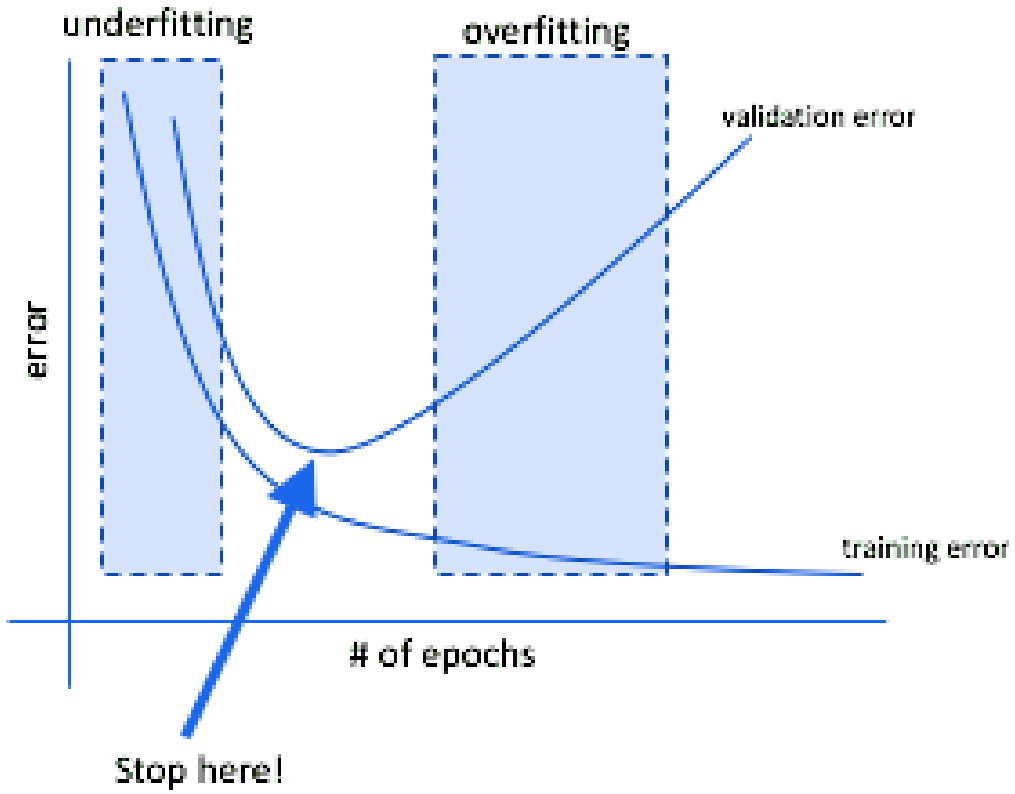
$$\mathrm{E}[\mathbf{y}] \approx \frac{1}{T}\sum_{t=1}^{T}\hat{y}_t(\mathbf{x})$$

$$\mathrm{Var}(\mathbf{y}) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{y}}_t(\mathbf{x})\hat{\mathbf{y}}_t(\mathbf{x})^T - E[\mathbf{y}]E[\mathbf{y}]^T$$

    - τ is a constant determined by the model structure.
    - Ref: Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

Regularization - Early Stopping

- Again Validation Error Curve!



Regularization - The early stopping meta-algorithm for determining the best amount of time to train.

- Let n be the number of steps between evaluations.
- Let p be the «patience» the number of times to observe worsening validation set error before giving up.
- Initialization ($\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_0}, i \leftarrow 0, j \leftarrow 0, v \leftarrow \infty, \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, and\ i^* \leftarrow i$)
- **while** $j < p$ **do**
- Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
- $i \leftarrow i + n$
- $v' \leftarrow ValidationSetError(\boldsymbol{\theta})$
- **if** $v' < v$ **then**
- $j \leftarrow 0$
- $\theta^* \leftarrow \theta$
- $i^* \leftarrow i$
- $v \leftarrow v'$
- **else**
- $j \leftarrow j + 1$
- **end if**
- **end while**
- Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Regularization - Early Stopping How to use data, Policy #1

- A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.
- Let $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ be the training set.
- Split $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ into $\{\boldsymbol{X}^{(subtrain)}, \boldsymbol{X}^{(valid)}\}$ and $\{\boldsymbol{Y}^{(subtrain)}, \boldsymbol{Y}^{(valid)}\}$ respectively.
- Run early stopping starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(subtrain)}$ and $\boldsymbol{Y}^{(subtrain)}$ for training data and $\boldsymbol{X}^{(valid)}$ and $\boldsymbol{Y}^{(valid)}$ for validation data. This returns i^*, the optimal number of steps.
- Set $\boldsymbol{\theta}$ to random values again.
- Train on $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ for i^* steps.

Regularization - Early Stopping How to use data, Policy #2

- Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.
- Let $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ be the training set.
- Split $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ into $\{\boldsymbol{X}^{(subtrain)}, \boldsymbol{X}^{(valid)}\}$ and $\{\boldsymbol{Y}^{(subtrain)}, \boldsymbol{Y}^{(valid)}\}$ respectively.
- Run early stopping starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(subtrain)}$ and $\boldsymbol{Y}^{(subtrain)}$ for training data and $\boldsymbol{X}^{(valid)}$ and $\boldsymbol{Y}^{(valid)}$ for validation data. This updates $\boldsymbol{\theta}$.
- $\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(subtrain)}, \boldsymbol{Y}^{(subtrain)})$
- **while** $J(\boldsymbol{\theta}, \boldsymbol{X}^{(valid)}, \boldsymbol{Y}^{(valid)}) > \epsilon$, **do**
- Train on $\boldsymbol{X}^{(train)}$ and $\boldsymbol{Y}^{(train)}$ for n steps
- **end while**

Regularization - Early Stopping Theory

- Why it Early Stopping acts as regulator?
- Taylor approximation around true solution ($\boldsymbol{w}^*$), and convex assumption!

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- Gradient Descent, starting from ($\boldsymbol{w}^{(0)} = 0$, t is iteration counter):

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}\left(\boldsymbol{w}^{(\tau-1)}\right) = \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right)$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right).$$

Regularization - Early Stopping Theory

- Eigen decomposition of $\boldsymbol{H}(\boldsymbol{H}=\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top})$:

$$\boldsymbol{w}^{(\tau)}-\boldsymbol{w}^{*} = (\boldsymbol{I}-\epsilon\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top})\left(\boldsymbol{w}^{(\tau-1)}-\boldsymbol{w}^{*}\right)$$
$$\boldsymbol{Q}^{\top}\left(\boldsymbol{w}^{(\tau)}-\boldsymbol{w}^{*}\right) = (\boldsymbol{I}-\epsilon\boldsymbol{\Lambda})\boldsymbol{Q}^{\top}\left(\boldsymbol{w}^{(\tau-1)}-\boldsymbol{w}^{*}\right)$$

- 1st order DE, starting $w^{(0)}=0$, and small step size ($|1-\varepsilon\lambda_i|<1$)

$$\boldsymbol{Q}^{\top}\boldsymbol{w}^{(\tau)} = [\boldsymbol{I}-(\boldsymbol{I}-\epsilon\boldsymbol{\Lambda})^{\tau}]\boldsymbol{Q}^{\top}\boldsymbol{w}^{*}$$

- Remember similar analysis in L2 Regularization:

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda}+\alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^{*} \Rightarrow \boldsymbol{Q}^{\top}\tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda}+\alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^{*}$$

- Obviously $\frac{\lambda_i}{\lambda_i+\alpha} = 1-\frac{\alpha}{\lambda_i+\alpha}$, then

$$\boldsymbol{Q}^{\top}\tilde{\boldsymbol{w}} = [\boldsymbol{I}-(\boldsymbol{\Lambda}+\alpha\boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^{*}$$

Regularization - Early Stopping Theory

- Compare two approaches:
- Early Stopping:

$$\boldsymbol{Q}^{\top}\boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^{\tau}]\boldsymbol{Q}^{\top}\boldsymbol{w}^{*}$$

- L2 Regularization:

$$\boldsymbol{Q}^{\top}\,\tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha]\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w}^{*}$$

- Gives:

$$(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^{\tau} = (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha = \left(\boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{\Lambda}\right)^{-1}$$

- Solve for small ϵ and large α (Regularization factor):

$$\boldsymbol{I} - \epsilon\tau\boldsymbol{\Lambda} \approx \boldsymbol{I} - \frac{1}{\alpha}\boldsymbol{\Lambda} \Rightarrow \tau = \frac{1}{\alpha\epsilon}, \alpha = \frac{1}{\tau\epsilon}$$

Regularization - Noise Robustness

- Add noise to input or output
- Train models to minimize noise injected loss:

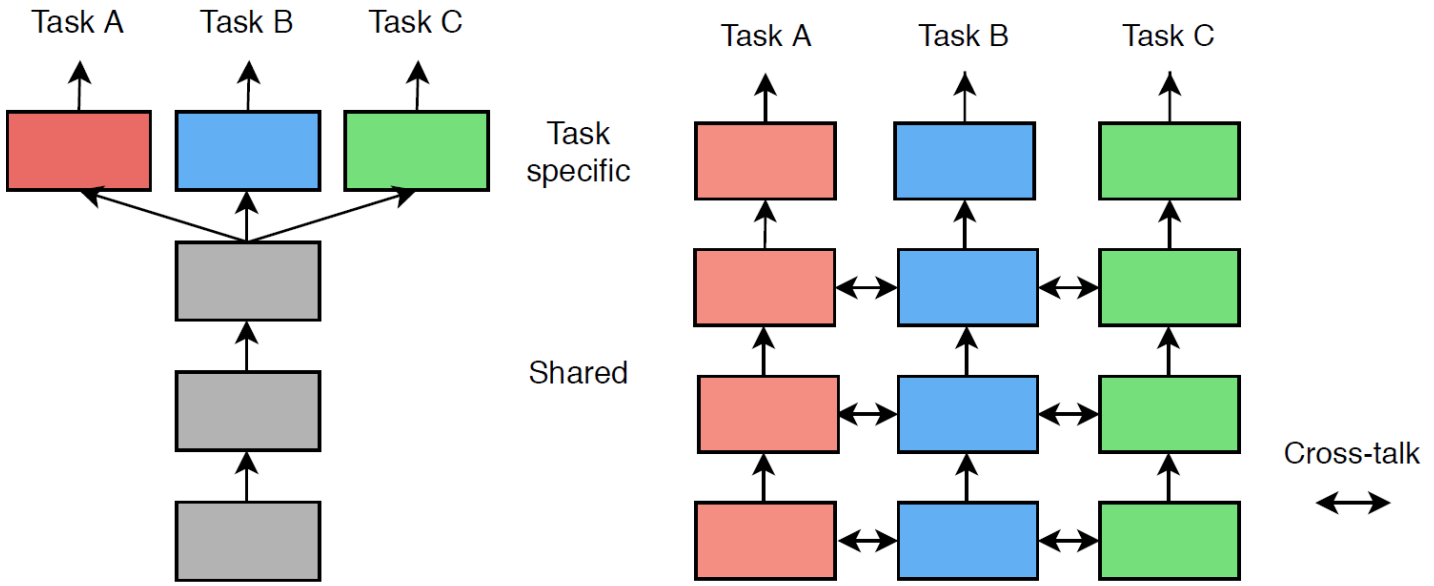
$$Loss(f(\boldsymbol{X}+\boldsymbol{\varepsilon}), \boldsymbol{Y}+\boldsymbol{\delta})$$

- Example (Label smoothing):

$$\begin{bmatrix}0\\1\\0\end{bmatrix} \Rightarrow \begin{bmatrix}0.01\\0.98\\0.01\end{bmatrix}$$

Regularization - Multitasking Learning

- Learn two or more tasks with a layer of shared parameters



(a) Hard parameter sharing (b) Soft parameter sharing

- Cross-talk example: The distance between the parameters of the model used as regularizer

Regularization - Parameter Tying

- Suppose the two models map the input to two different, but related outputs and the tasks are similar enough (perhaps with similar input and output distributions) that we believe the model parameters should be close to each other.

$$\boldsymbol{Y}^{(A)} = f\left(\boldsymbol{\omega}^{(A)}, \boldsymbol{X}\right)$$

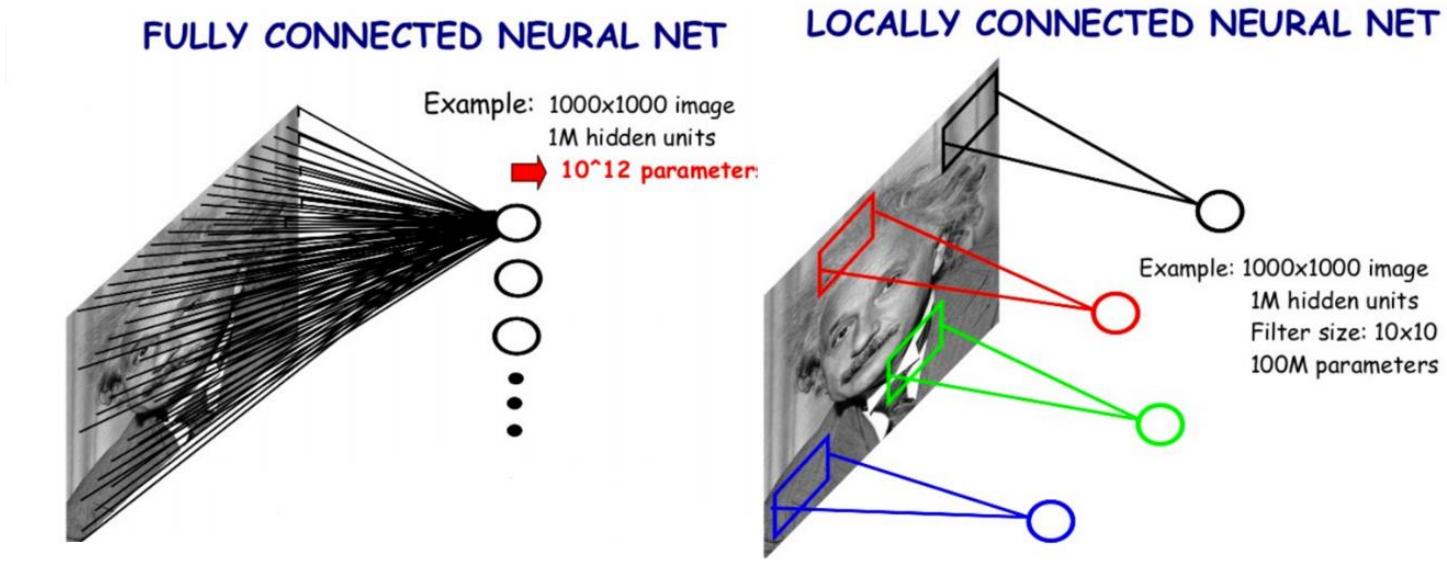
$$\boldsymbol{Y}^{(B)} = f\left(\boldsymbol{\omega}^{(B)}, \boldsymbol{X}\right)$$

- Assume tasks are similar enough (perhaps with similar input and output distributions) that we believe the model parameters should be close to each other.
- Use a parameter norm penalty:

$$\boldsymbol{\Omega}^{(A,B)} = \left\|\boldsymbol{\omega}^{(A)} - \boldsymbol{\omega}^{(B)}\right\|_2^2$$

Regularization - Others

- Parameter Sharing:
  - Force two set of weights is equals
- Sparse Parameters:
  - Force some weights equal to zero



Optimization Challenges

- Local minimum and saddle points:
  - For many high-dimensional non-convex functions (and random function). Local minimum/maximum are rare compared to saddle points. (#Saddles/#Locals grows exponentially with R^D)
  - In local minimum, eigenvalues of Hessian are Positive
  - In Saddle Points, eigenvalues of Hessian may be Positive or Negative
- Ill-Conditioning:
  - Hessian term may exceeds gradient term

$$f\left(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}\right) \approx f\left(\boldsymbol{x}^{(0)}\right) - \epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^T \boldsymbol{H} \boldsymbol{g}$$

Optimization Challenges

- Vanishing/Exploding Gradient:
  - Suppose a simple computational graph (Back Propagation) in which we repeatedly (t times) multiplying by a matrix $\boldsymbol{W}$

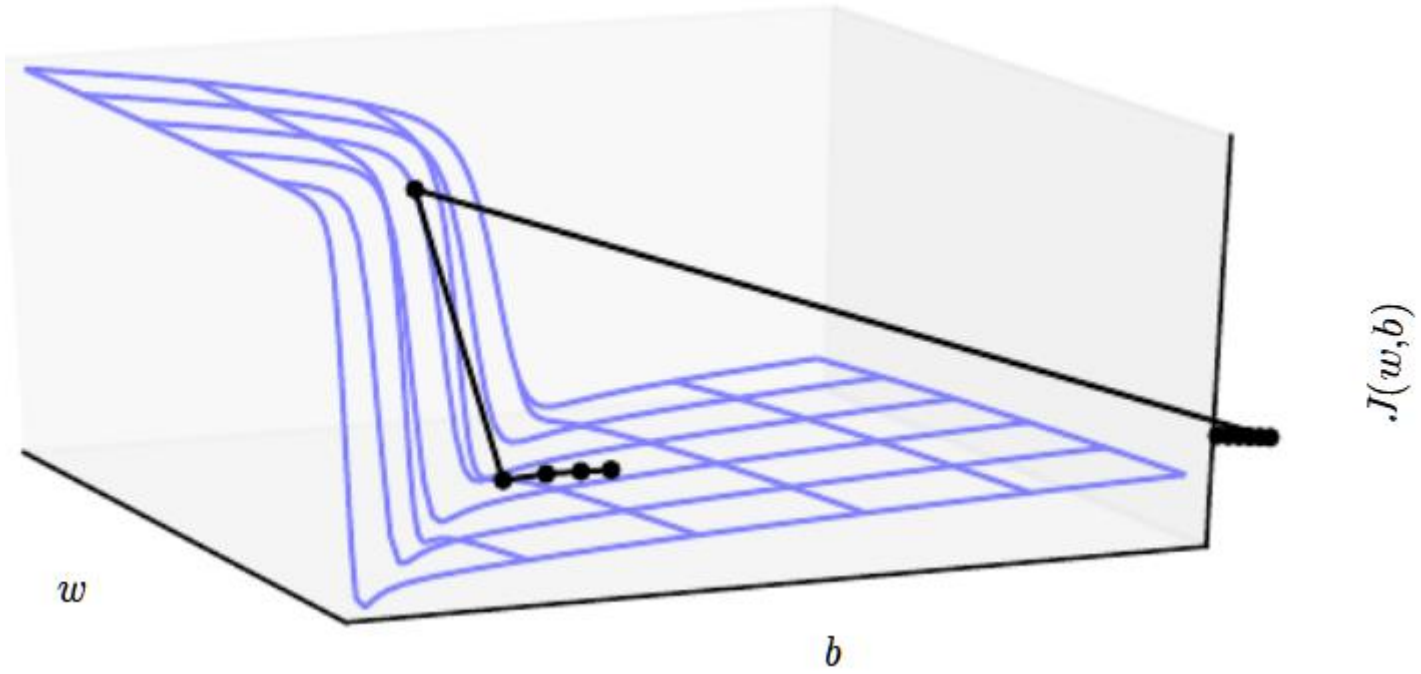
$$\boldsymbol{W} = \boldsymbol{V} diag(\boldsymbol{\lambda}) \boldsymbol{V}^{-1}$$

$$\boldsymbol{W}^{\boldsymbol{t}} = \left(\boldsymbol{V} diag(\boldsymbol{\lambda}) \boldsymbol{V}^{-1}\right)^{\boldsymbol{t}} = \boldsymbol{W} = \boldsymbol{V} diag(\boldsymbol{\lambda})^{\boldsymbol{t}} \boldsymbol{V}^{-1}$$

- Vanishing ($\lambda_i < 1$) or Exploding ($\lambda_i > 1$)
  - Vanishing: which direction is?
  - Exploding: Cliff Structure

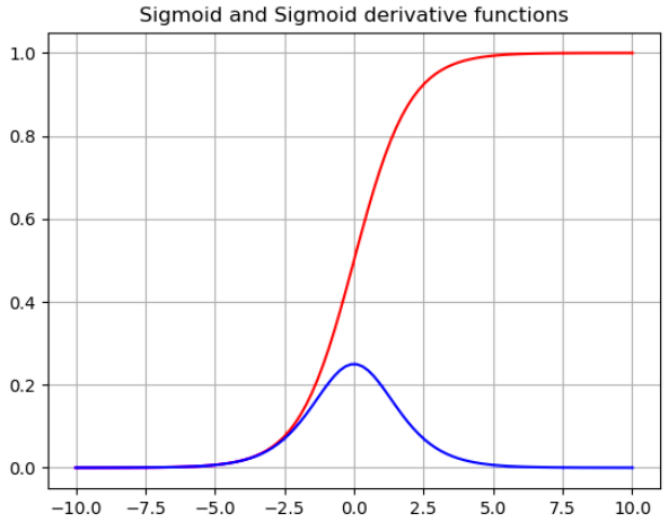
Optimization Challenges

- Exploding Gradient in Cliff Structure

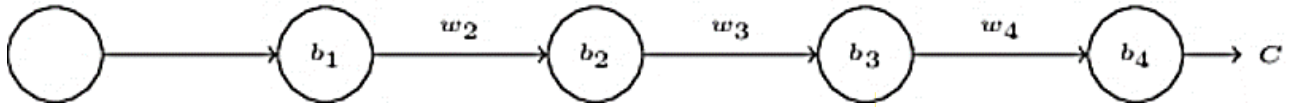


Optimization Challenges

- In Deep Neural Network:



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



- Chaotic Behavior in Nonlinear Function!
- ReLU is a good choice!
- Gradient Clipping ($\alpha \frac{g}{\|g\|}$)

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD) updates at training iteration k
- inputs: Learning rate, ϵ_k, and Initial parameter , $\boldsymbol{\theta}$.
- **while** stopping criteria not met do
- Sample a minibatch of m examples from training set $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\{\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(m)}\}$
- Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
- Apply update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
- **end while**.
- $\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha \epsilon_\tau$, with $\alpha = \frac{k}{\tau}$. After iteration τ, it is common to leave ϵ constant.
- $\epsilon_\tau \approx 0.01 \epsilon_0$

SGD+Momentum

- SGD+Momentum updates at training iteration k
- inputs: Learning rate, ϵ_k, momentum parameter, α, initial velocity, $\boldsymbol{v}$, and initial parameter , $\boldsymbol{\theta}$.
- **while** stopping criteria not met do
- Sample a minibatch of m examples from training set $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\{\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(m)}\}$
- Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
- Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$
- Apply update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
- **end while**.
- Typical α: 0.5, 0.8, 0.9.

SGD+Nesterov Momentum (NAG)

- Nesterov accelerated gradient (NAG) updates at training iteration k
- inputs: Learning rate, ϵ_k, momentum parameter, α, initial velocity, $\boldsymbol{v}$, and initial parameter , $\boldsymbol{\theta}$.
- **while** stopping criteria not met do
- Sample a minibatch of m examples from training set $\{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\{\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(m)}\}$
- Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}\right), \boldsymbol{y}^{(i)}\right)$
- Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$
- Apply update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
- **end while**.
- Typical α: 0.5, 0.8, 0.9.

Algorithms with Adaptive Learning Rates

- AdaGrad
- AdaDelta
- RMSProp
- ADAM*
- ADAMax
- NADAM
- signSGD*new*
- …

Approximate Second-Order Methods

- Newton's Method Idea:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left.\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}, \boldsymbol{H} = \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left.\frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

- Find best update direction:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\boldsymbol{g}$$

- $\boldsymbol{g}$ and $\boldsymbol{H}$ are estimated using minibatch random sampling.
- Regularized update:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1}\boldsymbol{g}$$

Approximate Second-Order Methods

- CG (Conjugate Gradients)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno)
- L-BFGS* (Limited Memory BFGS)
- Sophia^{new} (for LLM)

Additional Strategies for SGD

- Random Shuffle training data after every epoch
- Curriculum Learning: Sort data (easy to hard)
- Batch normalization: Re-normalizes every mini-batch to zero mean, unit variance
- Gradient noise (Decreasing variance)

Normalization

- In a deep neural network, there is a phenomenon called *internal covariate shift*, which is a change in the input distribution to the network's layers due to the ever-changing network parameters during training.
- The input layer may have certain features which dominate the process, due to having high numerical values. This can create a bias in the network because only those features contribute to the outcome of the training.
- For example, imagine feature one having values between 1 and 5, and feature two having values between 100 and 10000. During training, due to the difference in scale of both features, feature two would dominate the network and only that feature would have a contribution to the outcome of the model.

Why?

- Reducing the internal covariate shift to improve training
- Scaling each feature to a similar range to prevent or reduce bias in the network
- Speeding up the optimization process by preventing weights from exploding all over the place and limiting them to a specific range
- Reducing overfitting in the network by aiding in regularization

Simple Methods

- Normalization or (min-Max scaling):

$$\tilde{x}_{ki} = \frac{x_{ki} - min_k}{max_k - min_k} \in [0,1], \qquad or \;\; \tilde{x}_{ki} = \frac{2x_{ki} - (max_k + min_k)}{max_k - min_k} \in [-1,1]$$

- where x_{ki} is k^{th} feature of i^{th} sample, $max_k = max\{x_{ki}\}_i$, and $min_k = min\{x_{ki}\}_i$.
- Standardization or Z-Score normalization:

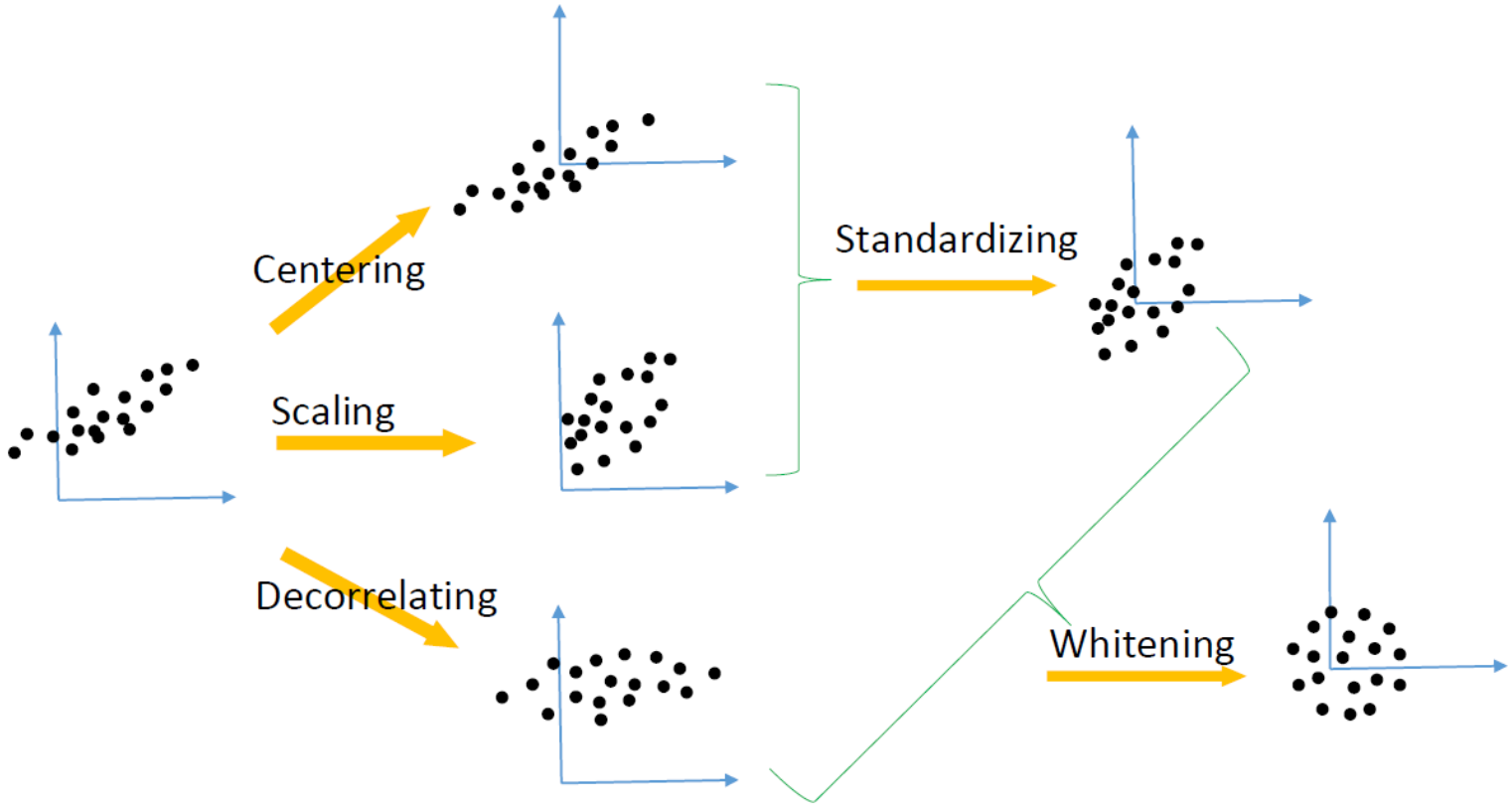
$$\tilde{x}_{ki} = \frac{x_{ki} - m_k}{\sqrt{\sigma_k^2 + \varepsilon}}, \qquad m_k = \frac{1}{N}\sum_{i=1}^{N} x_{ki}\,, \qquad \sigma_k^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ki} - m_k)^2$$

- where m_k and σ_k^2 are mean and variance of feature k over training set
- Some times additional learnable scale and shift parameters used:

$$\hat{x}_{ki} = \gamma\tilde{x}_{ki} + \beta$$

Simple Methods

- Illustration:



Idea for Deep Learning

- Need same distribution in each minibatch!
- How to Solve:
  - Input layer normalization (He/Xavier/…) and shuffling!
- What about hidden layer:
  - Uniform/Normal dist. in input layer ⇨Non-uniform/Non-Normal dist.
  - In hidden layer Hidden layer input dist. varied in each batch! (internal covariate shift)
- Original work:
  - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift by Sergey Ioffe and Christian Szegedy, 2015 (Google Team)

Idea for Deep Learning

- Proposed Methods
- Batch Normalization
- Weight Normalization
- Layer Normalization
- Group Normalization
- Weight Standardization

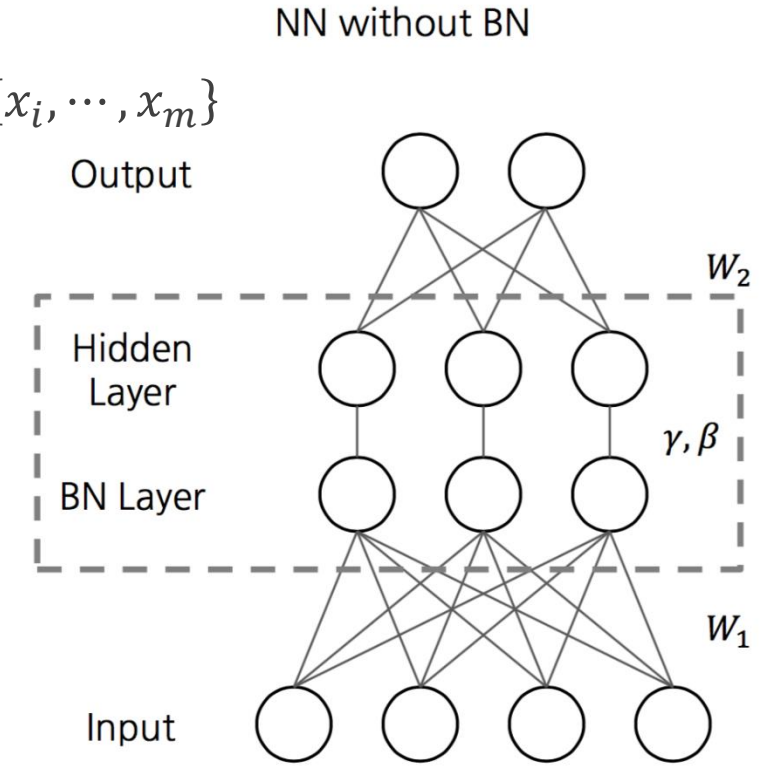
Batch Normalization

- Idea: Insert a (Batch) Normalization Layer (BN-Layer) before each layer
- Normalization need data → Batch/min or batch training
- Normalization per cell! (Whitening need too many sample per mini-batch)
- Normalize each layer output (input of next layer) in each iteration (min-batch)

Batch Normalization - Training

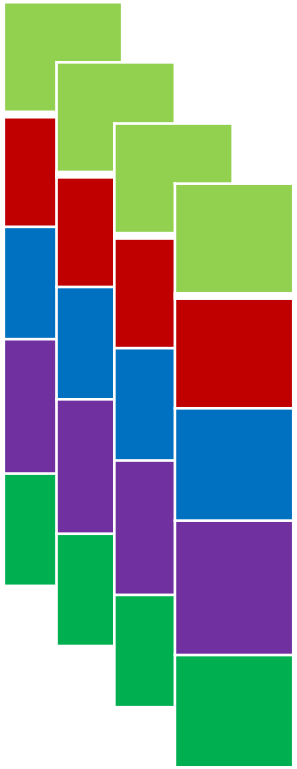
- Zero-mean, unit-variance is not ideal for nonlinear units
- Two Learnable extra-weights per cell (γ, β).
- Next Layer input is y_i

- **Input**: a minibatch of m examples from training set $\mathcal{B} = \{x_i, \cdots, x_m\}$
- **Output**: $\{y_i, \cdots, y_m\}, y_i = BN_{\gamma,\beta}(x_i)$
- $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$, minibatch means
- $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$, minibatch variance
- $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$, normalization
- $y_i = \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$, scale and shift



Batch Normalization - Training

- Graphical illustration (m=4, k=5)
- Same color units share mean and variance.



Batch Normalization – Parameter Training

- $\frac{\partial Loss}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial y_i} \frac{\partial y_i}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial y_i}$
- $\frac{\partial Loss}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial y_i} \frac{\partial y_i}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial y_i} \hat{x}_i$
- $\frac{\partial Loss}{\partial x_i} = \frac{\partial Loss}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial Loss}{\partial \sigma_{\mathcal{B}}^2} \frac{\partial \sigma_{\mathcal{B}}^2}{\partial x_i} + \frac{\partial Loss}{\partial \mu_{\mathcal{B}}} \frac{\partial \mu_{\mathcal{B}}}{\partial x_i} = \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \frac{\partial Loss}{\partial \hat{x}_i} + \frac{2(x_i - \mu_{\mathcal{B}})}{m} \frac{\partial Loss}{\partial \sigma_{\mathcal{B}}^2} + \frac{1}{m} \frac{\partial Loss}{\partial \mu_{\mathcal{B}}}$
- $\frac{\partial Loss}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \mu_{\mathcal{B}}} + \frac{\partial Loss}{\partial \sigma_{\mathcal{B}}^2} \frac{\partial \sigma_{\mathcal{B}}^2}{\partial \mu_{\mathcal{B}}} = -\frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \sum_{i=1}^{m} \frac{\partial Loss}{\partial \hat{x}_i} - \frac{2}{m} \frac{\partial Loss}{\partial \sigma_{\mathcal{B}}^2} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})$
- $\frac{\partial Loss}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial Loss}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial \sigma_{\mathcal{B}}^2} + \frac{\partial Loss}{\partial \mu_{\mathcal{B}}} \frac{\partial \mu_{\mathcal{B}}}{\partial \sigma_{\mathcal{B}}^2} = -\frac{1}{2} \left(\sigma_{\mathcal{B}}^2 + \epsilon\right)^{-3/2} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}}) \frac{\partial Loss}{\partial \hat{x}_i} + 0$
- $\frac{\partial Loss}{\partial \hat{x}_i} = \frac{\partial Loss}{\partial y_i} \frac{\partial y_i}{\partial \hat{x}_i} = \gamma \frac{\partial Loss}{\partial y_i}$

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

Batch Normalization - Inference

- Test/Inference Phase:
  - There is ONE sample in input NOT mini-batch
  - But we need $E\{x\}$ and $var\{x\}$ to normalize

$$\hat{\boldsymbol{x}} \leftarrow \frac{\hat{\boldsymbol{x}} - E\{\boldsymbol{x}\}}{\sqrt{var\{\boldsymbol{x}\} + \epsilon}}$$

  - Use expectation (unbiased estimator) of μ_B and σ_B^2 (over all minibatch in training phase) instead, as following:

$$E\{\boldsymbol{x}\} \leftarrow E_B\{\boldsymbol{\mu}_B\}$$
$$var\{x\} \leftarrow \frac{m}{m-1} E_B\{\boldsymbol{\sigma}_B^2\}$$

$$\boldsymbol{y} = \gamma \frac{\boldsymbol{x} - E\{\boldsymbol{x}\}}{\sqrt{var\{\boldsymbol{x}\} + \epsilon}} + \beta$$

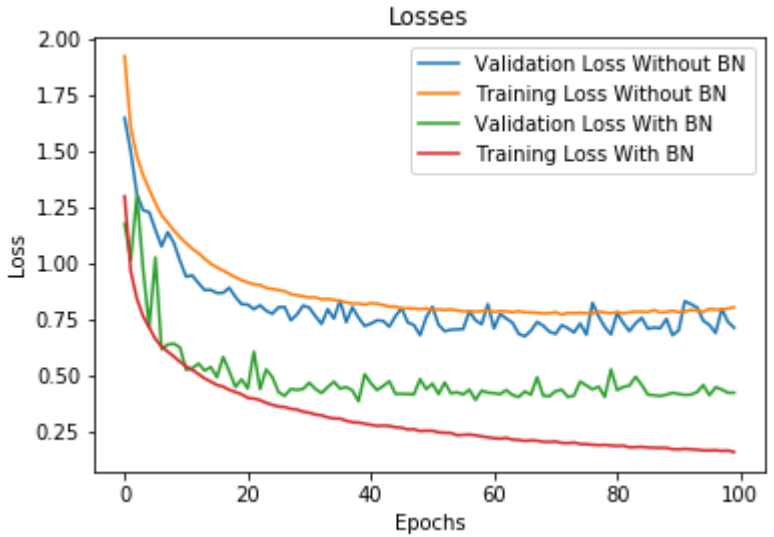
Batch Normalization - Inference

- 2. Running (moving) mean/variance during training phase
- $\hat{\mu}_B(t) = \lambda\hat{\mu}_B(t-1) + (1-\lambda)\mu_B(t-1)$,
- $\hat{\sigma}_B^2(t) = \lambda\hat{\sigma}_B^2(t-1) + (1-\lambda)\sigma_B^2(t-1)$
- where $\lambda = 1/n$, and n is the number of batches tracked (sometimes $\lambda \sim 0.95$).

$$\boldsymbol{y}_i = \gamma \frac{\boldsymbol{x}_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

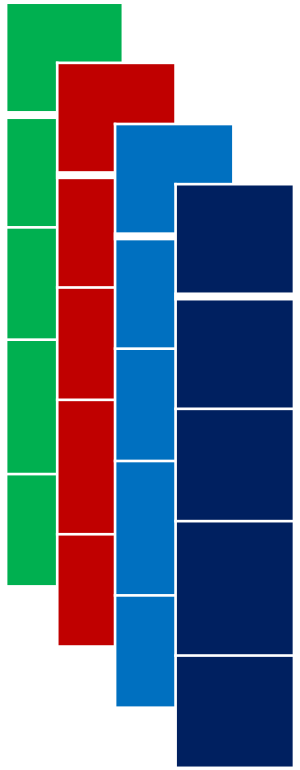
Batch Normalization

- Other benefits:
  - Higher learning rate (avoiding vanishing/exploding gradient)
  - Regularization (Batch information for normalization)
  - May use saturating active function



Layer Normalization

- Layer normalization normalizes input across the features instead of normalizing input features across the batch dimension in batch normalization.
  - Graphical illustration (m=4, k=5)
  - Same color units share mean and variance.

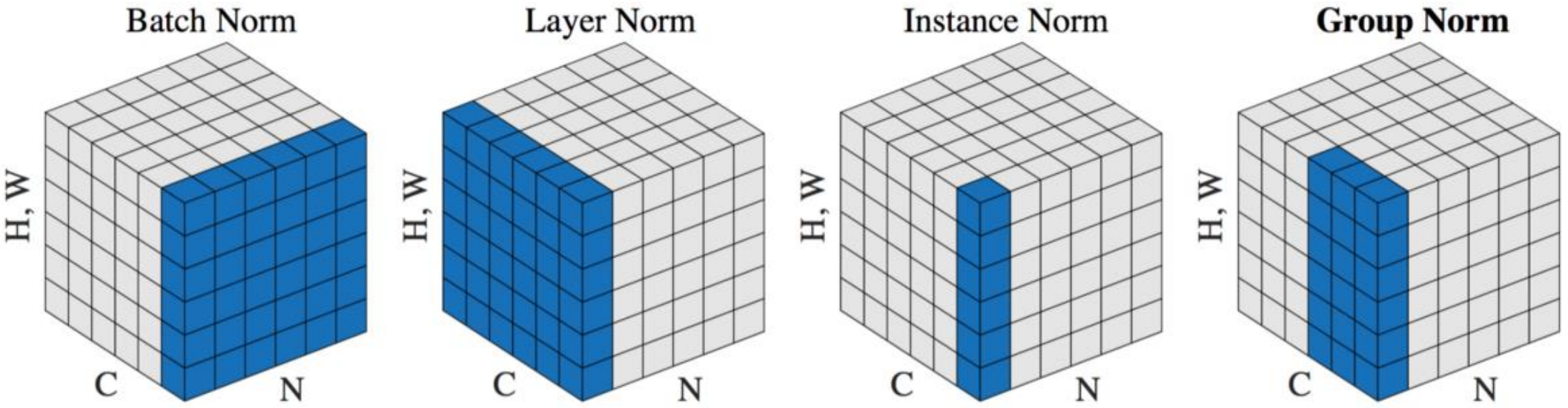


Other methods (for CNN)

- Instant Normalization
- Group Normalization

Other methods (Later)

- Graphical Illustration:
- With N,C,H,W referring to the batches, channels, height, and width respectively.



Weight Normalization, Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks.

- A simple trick (reparameterizaton)

$$\boldsymbol{W} = g\frac{\boldsymbol{V}}{\|\boldsymbol{V}\|}$$

$$\frac{\partial Loss}{\partial g} = \frac{\partial Loss}{\partial \boldsymbol{W}}\frac{\partial \boldsymbol{W}}{\partial g} = \frac{\boldsymbol{V}}{\|\boldsymbol{V}\|}\frac{\partial Loss}{\partial \boldsymbol{W}}$$

$$\frac{\partial Loss}{\partial \boldsymbol{V}} = \frac{\partial Loss}{\partial \boldsymbol{W}} = \frac{g}{\|\boldsymbol{V}\|}\frac{\partial Loss}{\partial \boldsymbol{W}} - g\frac{\boldsymbol{V}}{\|\boldsymbol{V}\|^2}\frac{\partial Loss}{\partial g}$$

- It has been shown this trick speedup learning procedure.

Weight Standardization, Micro-Batch Training with Batch-Channel Normalization and Weight Standardization.

- In Weight Standardization, instead of directly optimizing the loss function on the original weights ($\boldsymbol{W}$), we reparameterize the weights as a function of W, i.e., $\boldsymbol{V} = WS(\boldsymbol{W})$ and optimize the loss function on $\boldsymbol{V}$ by SGD.

$$\boldsymbol{V} = [V_{ij}], \qquad V_{ij} = \frac{W_{ij} - \mu_{W_{ij}}}{\sigma_{W_{ij}}}$$

- In above equation, mean and variance calculated over a hidden layer (or channel):

$$\mu_{W_{ij}} = \frac{1}{S}\sum_{i=1}^{S} W_{ij}$$

$$\sigma^2_{W_{ij}} = \frac{1}{S}\sum_{i=1}^{m}\left(W_{ij} - \mu_{W_{ij}}\right)^2 + \epsilon$$

Read More

- Normalization effects on deep neural networks, https://arxiv.org/abs/2209.01018
- Normalization Techniques in Training DNNs: Methodology, Analysis and Application, https://arxiv.org/abs/2009.12836

Regularization

- Any Question?