Regularization

And other ways to improve test performance

DL4DS – Spring 2025

Regularization

- Why is there a generalization gap between training and test data?
  - Overfitting (model describes statistical peculiarities)
  - Model unconstrained in areas where there are no training examples
- Regularization = methods to reduce the generalization gap
- Technically means adding terms to loss function
- But colloquially means any method (hack) to reduce gap between training and test data

Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Explicit regularization

- Standard loss function:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\big[\mathrm{L}[\boldsymbol{\phi}]\big] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i]\right]\end{aligned}$$

Explicit regularization

- Standard loss function:

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\big[\mathrm{L}[\boldsymbol{\phi}]\big] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}\ell_i[\mathbf{x}_i,\mathbf{y}_i]\right]\end{aligned}$$

- Regularization adds an extra term

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I}\ell_i[\mathbf{x}_i,\mathbf{y}_i] + \lambda\cdot\mathrm{g}[\boldsymbol{\phi}]\right]$$

Explicit regularization

- Standard loss function:

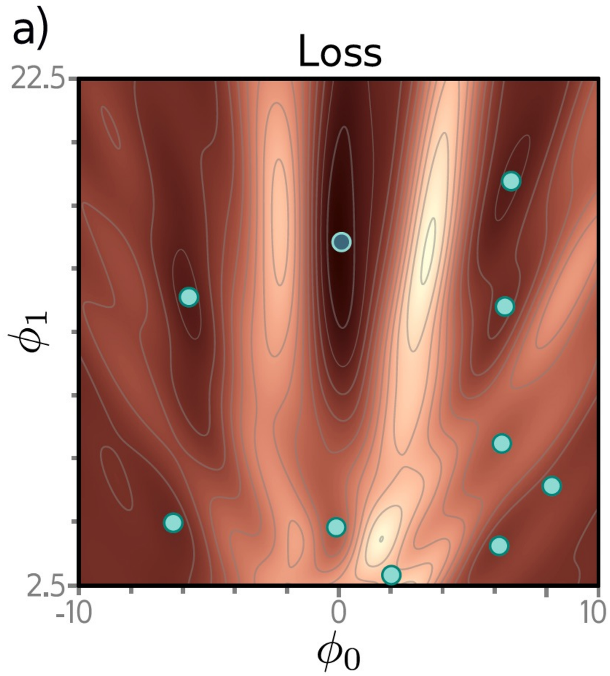
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\big[\mathrm{L}[\boldsymbol{\phi}]\big] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i]\right]$$

- Regularization adds an extra term

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i] + \lambda \cdot \mathrm{g}[\boldsymbol{\phi}]\right]$$

- Where $\mathrm{g}[\phi]$ is smaller for preferred parameters
- $\lambda > 0$ controls the strength of influence

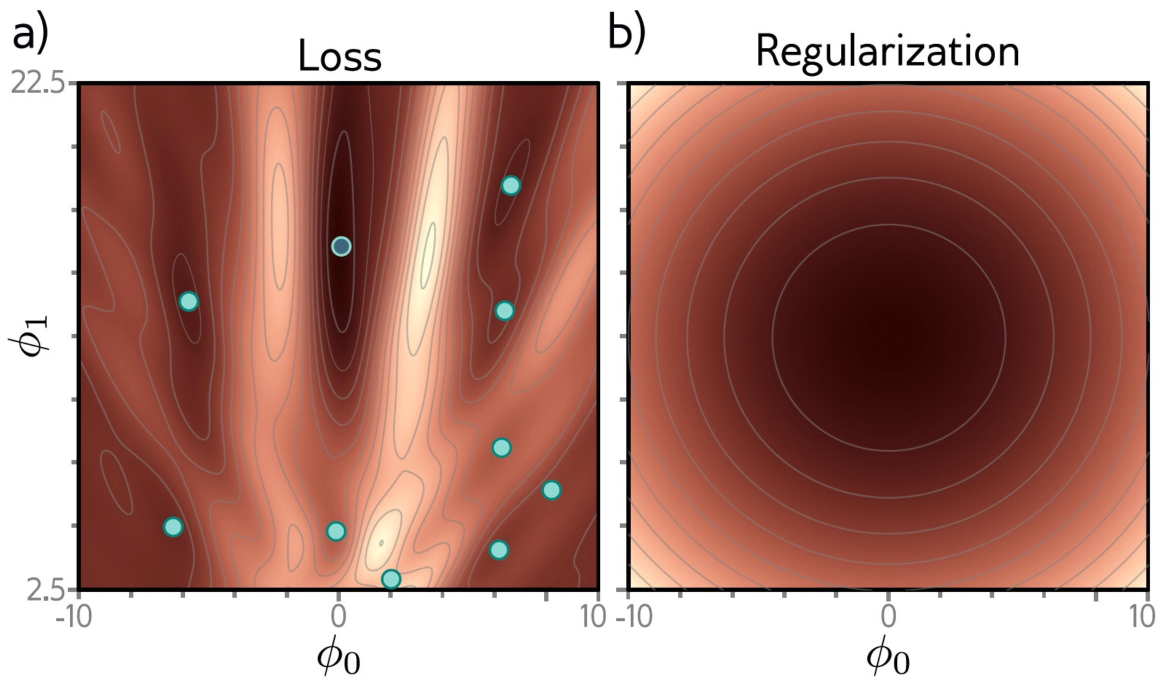
Explicit regularization



Loss function for Gabor model of Lecture 6 and Chapter 6.

○ denotes local minima

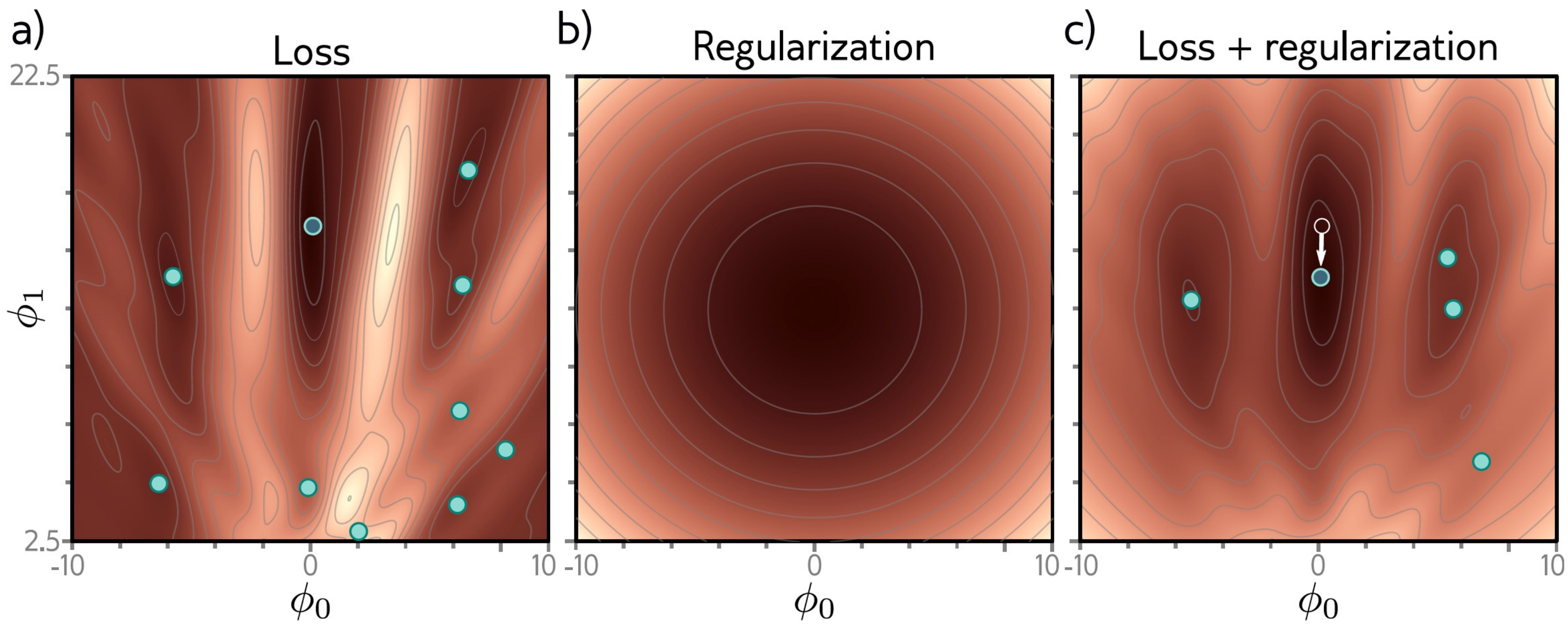
Explicit regularization



Example of a regularization function that prefers parameters close to 0.

Explicit regularization

Fewer local minima and the absolute minimum has moved.



denotes local minima

Probabilistic interpretation

- Maximum likelihood:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\phi}) \right]$$

- Regularization is equivalent to adding a prior over parameters

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\phi}) Pr(\boldsymbol{\phi}) \right]$$

Maximum a posteriori or MAP criterion

… what you know about parameters *before* seeing the data

Equivalence

- Explicit regularization:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\sum_{i=1}^{I} \ell_i[\mathbf{x}_i, \mathbf{y}_i] + \lambda \cdot \mathrm{g}[\boldsymbol{\phi}] \right]$$

- Probabilistic interpretation:

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\phi}) Pr(\boldsymbol{\phi}) \right]$$

- Converting to Negative Log Likelihood (e.g. $-\log(\cdot)$):

$$\lambda \cdot \mathrm{g}[\boldsymbol{\phi}] = -\log[Pr(\boldsymbol{\phi})]$$

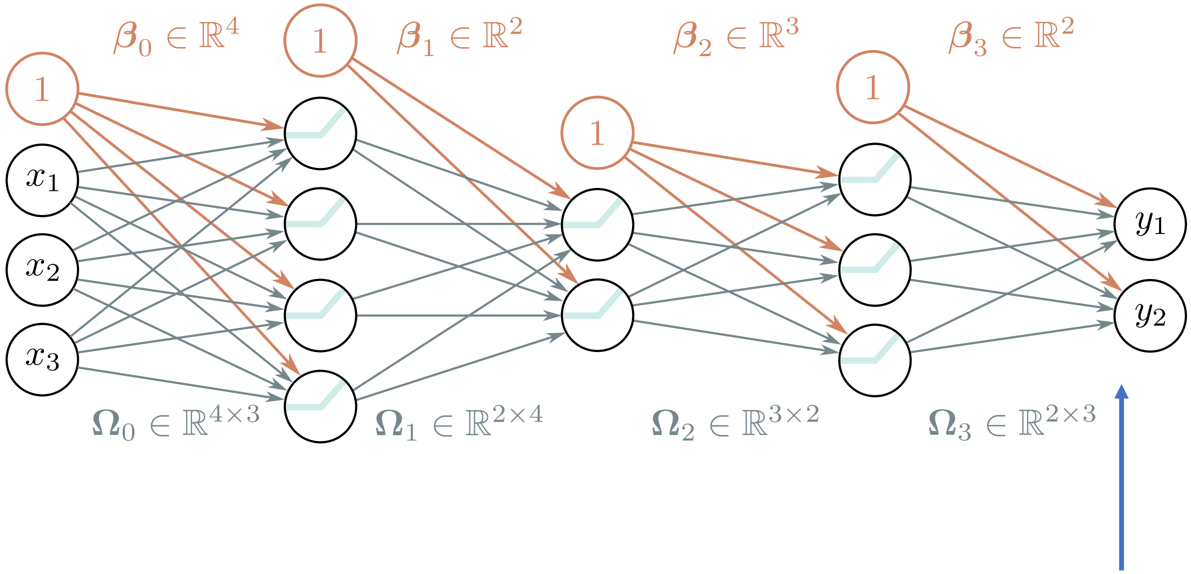
L2 Regularization

- Most common regularizer is L2 regularization
- Favors smaller parameters (like in previous example)

$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[\mathrm{L}[\boldsymbol{\phi}, \{\mathbf{x}_i, \mathbf{y}_i\}] + \lambda \sum_j \phi_j^2 \right]$$

- Also called Tikhonov regularization, ridge regression
- In neural networks, usually just for weights, and called weight decay

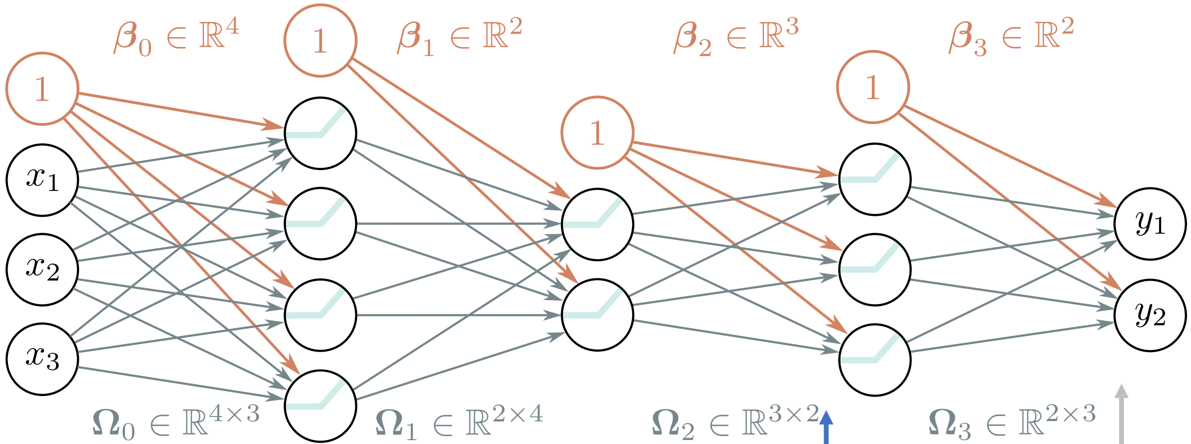
Why does L2 regularization help?



Outputs are weighted linear combination of last layer activations.

Smaller weights attenuate changes.

Why does L2 regularization help?

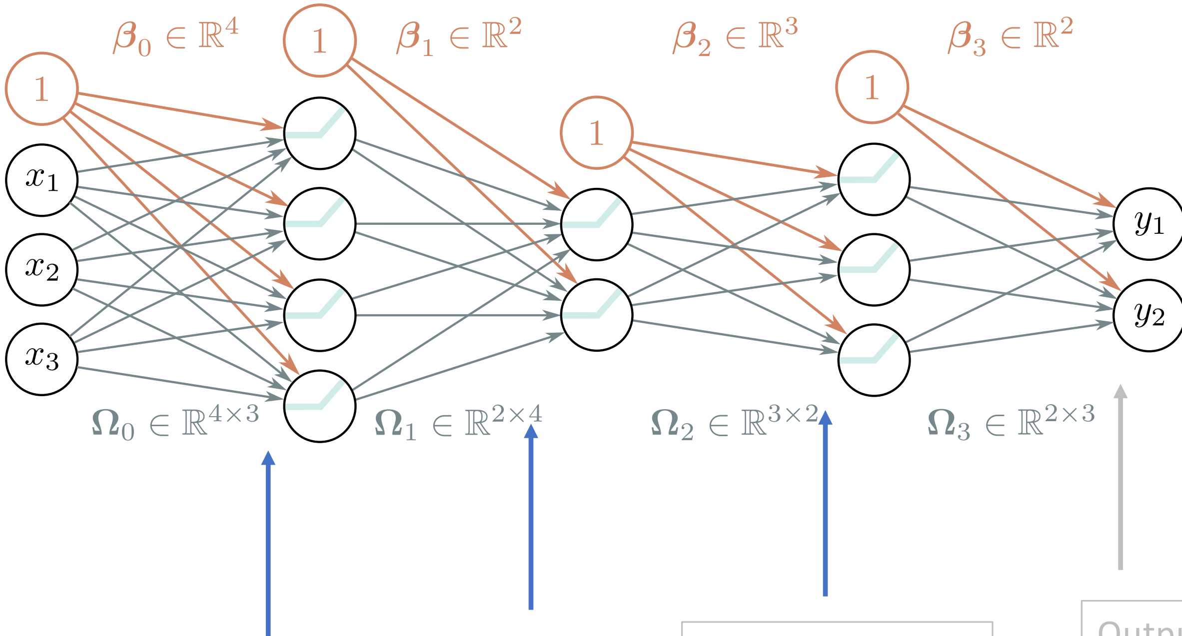


Same for the pre-activations into the last layer

Outputs are weighted linear combination of last layer activations.

Smaller weights attenuate changes.

Why does L2 regularization help?



And so on…

All the way back.

Same for the pre-activations into the last layer

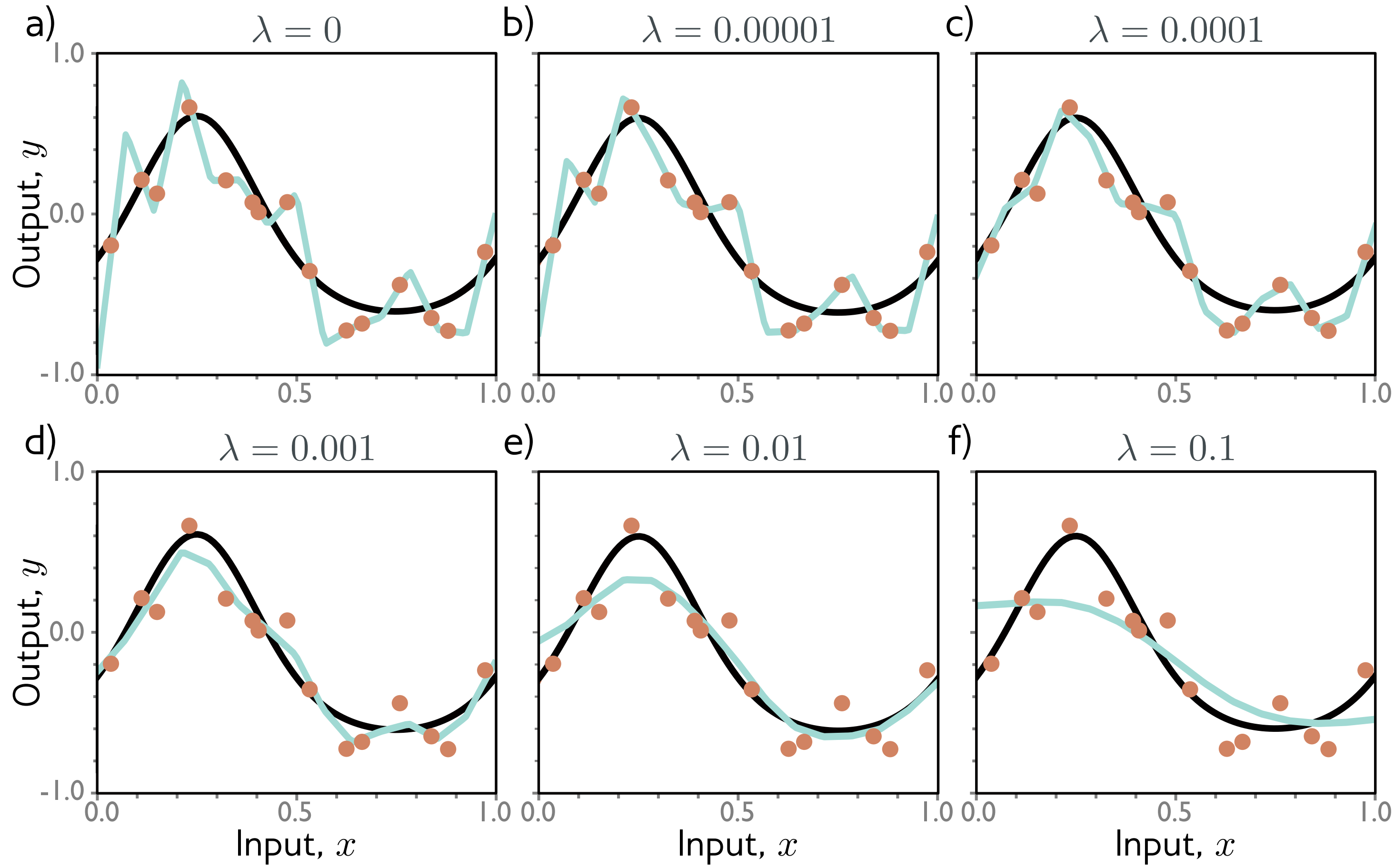
Outputs are weighted linear combination of last layer activations.

Smaller weights attenuate changes.

Why does L2 regularization help?

- Discourages fitting excessively to the training data (overfitting)
- Encourages smoothness between datapoints

L2 regularization (simple net from last lecture)



PyTorch Explicit L2 Regularizer

SGD

CLASS torch.optim.SGD(*params*, *lr=0.001*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*, ***, *maximize=False*, *foreach=None*, *differentiable=False*) [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*, *optional*) – learning rate (default: 1e-3)
- **momentum** (*float*, *optional*) – momentum factor (default: 0)
- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)

https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

ADAM

CLASS torch.optim.Adam(*params*, *lr=0.001*, *betas=(0.9, 0.999)*, *eps=1e-08*, *weight_decay=0*, *amsgrad=False*, ***, *foreach=None*, *maximize=False*, *capturable=False*, *differentiable=False*, *fused=None*) [SOURCE]

Implements Adam algorithm.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*, *Tensor*, *optional*) – learning rate (default: 1e-3). A tensor LR is not yet supported for all our implementations. Please use a float LR if you are not also specifying fused=True or capturable=True.
- **betas** (*Tuple*[*float*, *float*], *optional*) – coefficients used for computing running averages of gradient and its square (default: (0.9, 0.999))
- **eps** (*float*, *optional*) – term added to the denominator to improve numerical stability (default: 1e-8)
- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)

https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

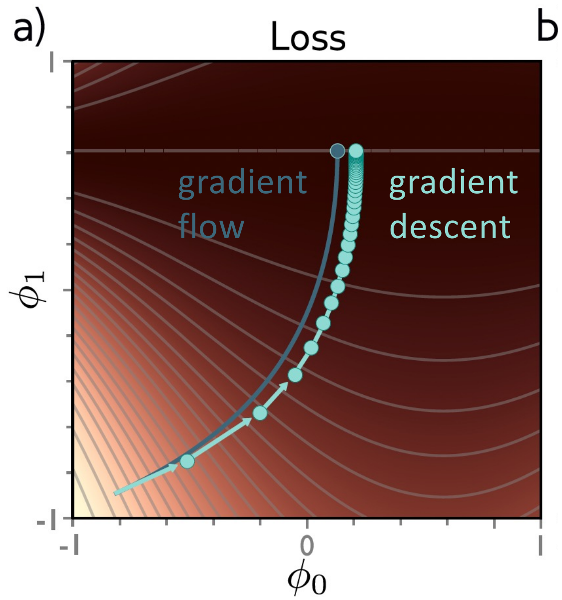
Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Implicit regularization

$$\phi_{t+1} = \phi_t - \alpha \frac{\partial L[\phi_t]}{\partial \phi} \qquad \lim_{\alpha \to 0} \qquad \frac{d\phi}{dt} = -\frac{\partial L}{\partial \phi}$$

- In the limit, as $\alpha \to 0$, the gradient descent equation becomes the gradient flow differential equation.
- Doesn't converge to the same place



Implicit regularization

$$\phi_{t+1} = \phi_t - \alpha \frac{\partial L[\phi_t]}{\partial \phi} \qquad \lim_{\alpha \to 0} \qquad \frac{d\phi}{dt} = -\frac{\partial L}{\partial \phi}$$

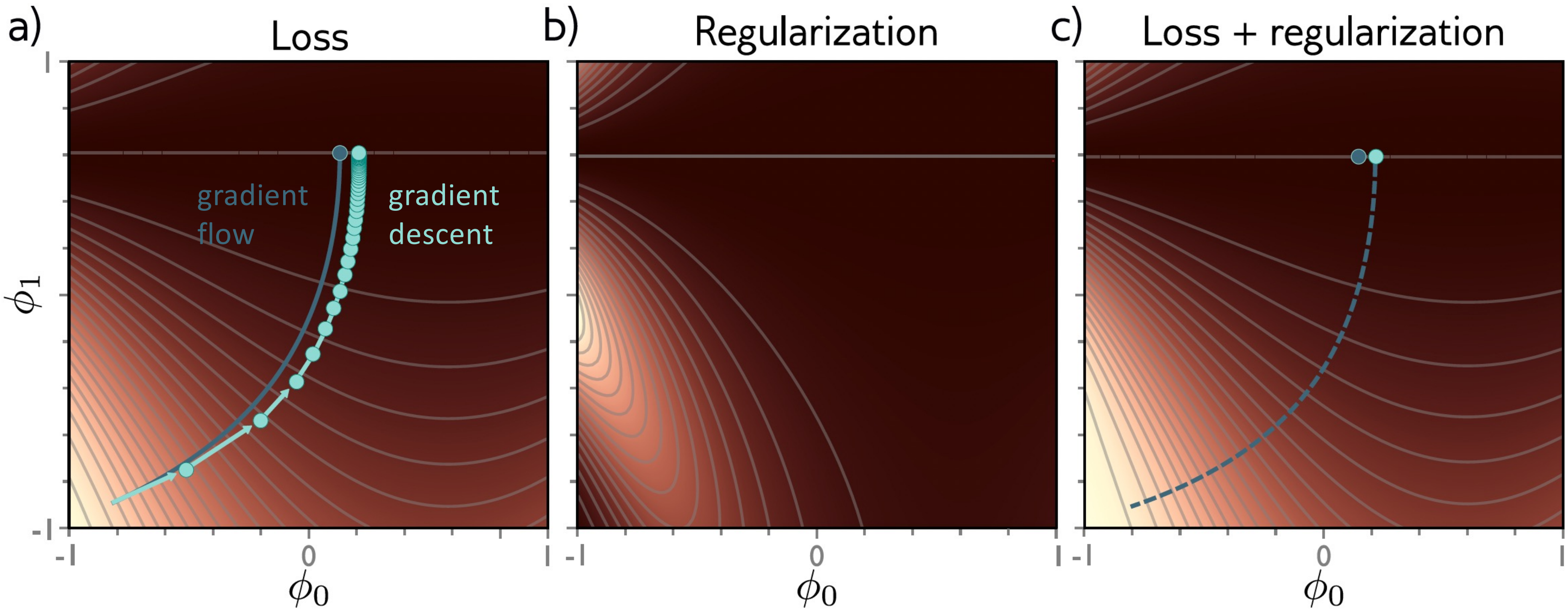
- The implicit regularization can be derived:

$$\tilde{\mathrm{L}}_{GD}[\phi] = \mathrm{L}[\phi] + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \phi}\right\|^2$$

Penalty for large gradients.

See derivation at end of UDL Ch. 9

Implicit regularization



Gradient descent doesn't converge to same location as (continuous) gradient flow.

Plot of the Implicit regularization ($\sim\left\|\partial L/\partial\phi\right\|^2$) to be added to loss

With regularization, continuous descent converges to same place

See derivation at end of UDL Ch. 9

Implicit regularization of SGD

- Gradient descent disfavors areas where gradients are steep

$$\tilde{\mathrm{L}}_{GD}[\boldsymbol{\phi}] = \mathrm{L}[\boldsymbol{\phi}] + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2$$

- SGD likes all batches to have similar gradients

$$\tilde{\mathrm{L}}_{SGD}[\boldsymbol{\phi}] = \tilde{\mathrm{L}}_{GD}[\boldsymbol{\phi}] + \frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \boldsymbol{\phi}} - \frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2$$

Want the batch variance to be small, rather than some batches fitting well and others not well…

Where $L = \frac{1}{I}\sum_{i=1}^{I}\ell_i[\mathbf{x}_i, y_i]$ and $L_b = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}_b}\ell_i[\mathbf{x}_i, y_i].$

Implicit regularization of SGD

- Gradient descent disfavors areas where gradients are steep

$$\tilde{\mathrm{L}}_{GD}[\boldsymbol{\phi}] = \mathrm{L}[\boldsymbol{\phi}] + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2$$

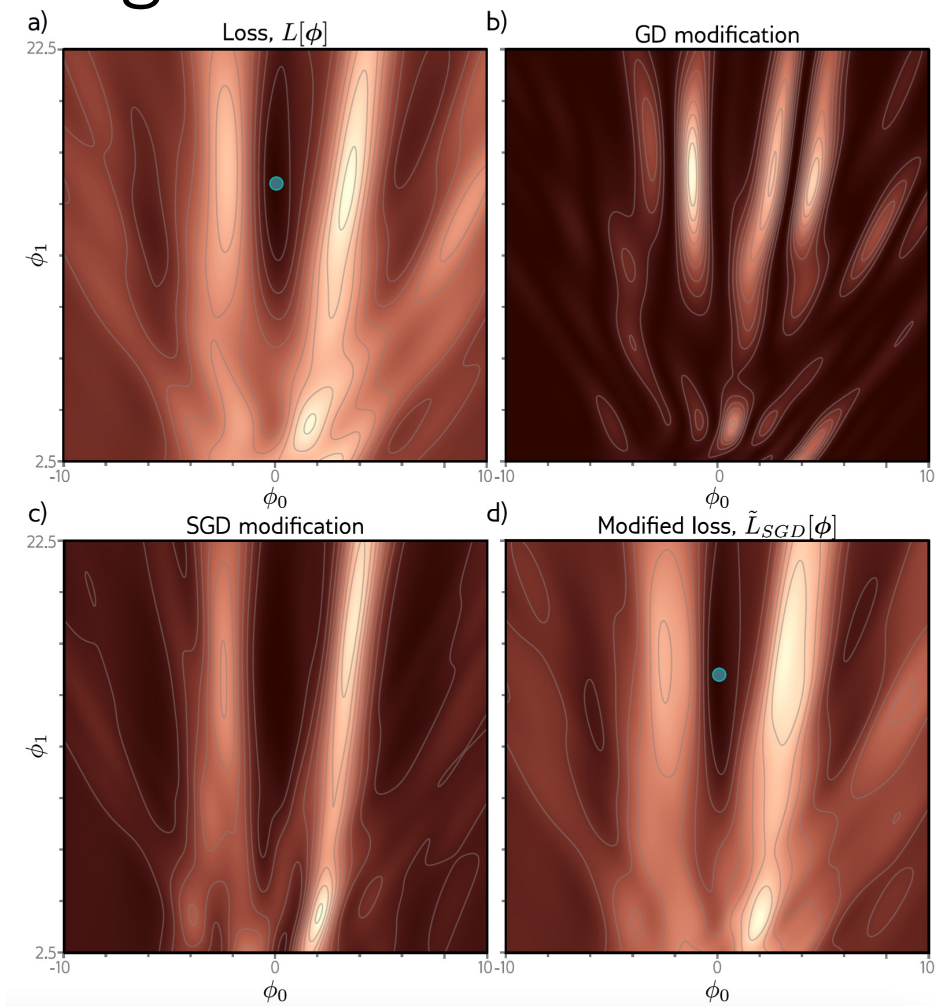
- SGD likes all batches to have similar gradients

$$\begin{aligned}\tilde{\mathrm{L}}_{SGD}[\boldsymbol{\phi}] &= \tilde{\mathrm{L}}_{GD}[\boldsymbol{\phi}] + \frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \boldsymbol{\phi}} - \frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2 \\ &= \mathrm{L}[\boldsymbol{\phi}] + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2 + \frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \boldsymbol{\phi}} - \frac{\partial L}{\partial \boldsymbol{\phi}}\right\|^2\end{aligned}$$

- Depends on learning rate – perhaps why larger learning rates generalize better.

Loss and Regularization Surfaces

Original Gabor Model Loss

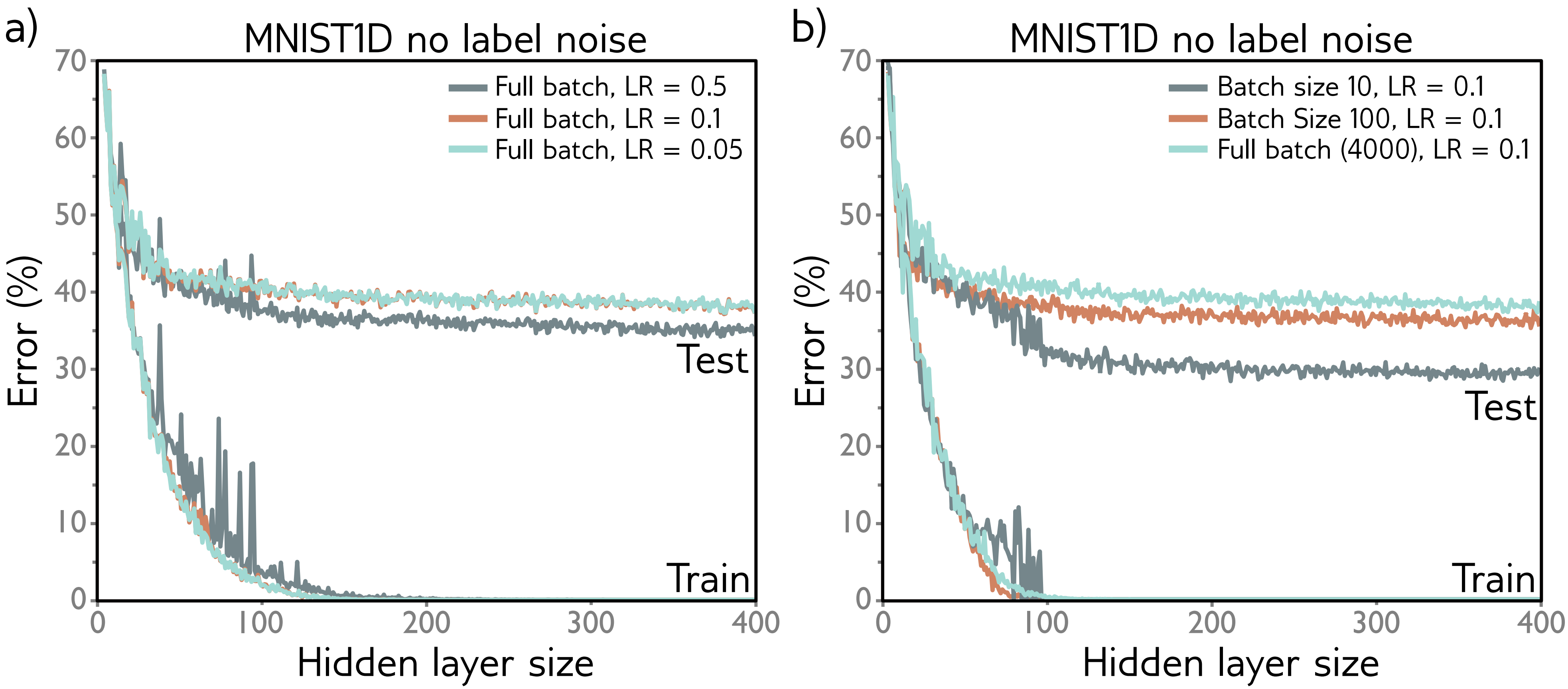


$$\frac{\alpha}{4}\left\|\frac{\partial L}{\partial \phi}\right\|^2$$

$$\frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \phi}-\frac{\partial L}{\partial \phi}\right\|^2$$

$$\tilde{\mathrm{L}}_{SGD}[\phi]$$

$$= \mathrm{L}[\phi] + \frac{\alpha}{4}\left\|\frac{\partial L}{\partial \phi}\right\|^2 + \frac{\alpha}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_b}{\partial \phi}-\frac{\partial L}{\partial \phi}\right\|^2$$



Generally, performance is

- best for larger learning rates
- best with smaller batches

Recap: Implicit regularization of GD and SGD

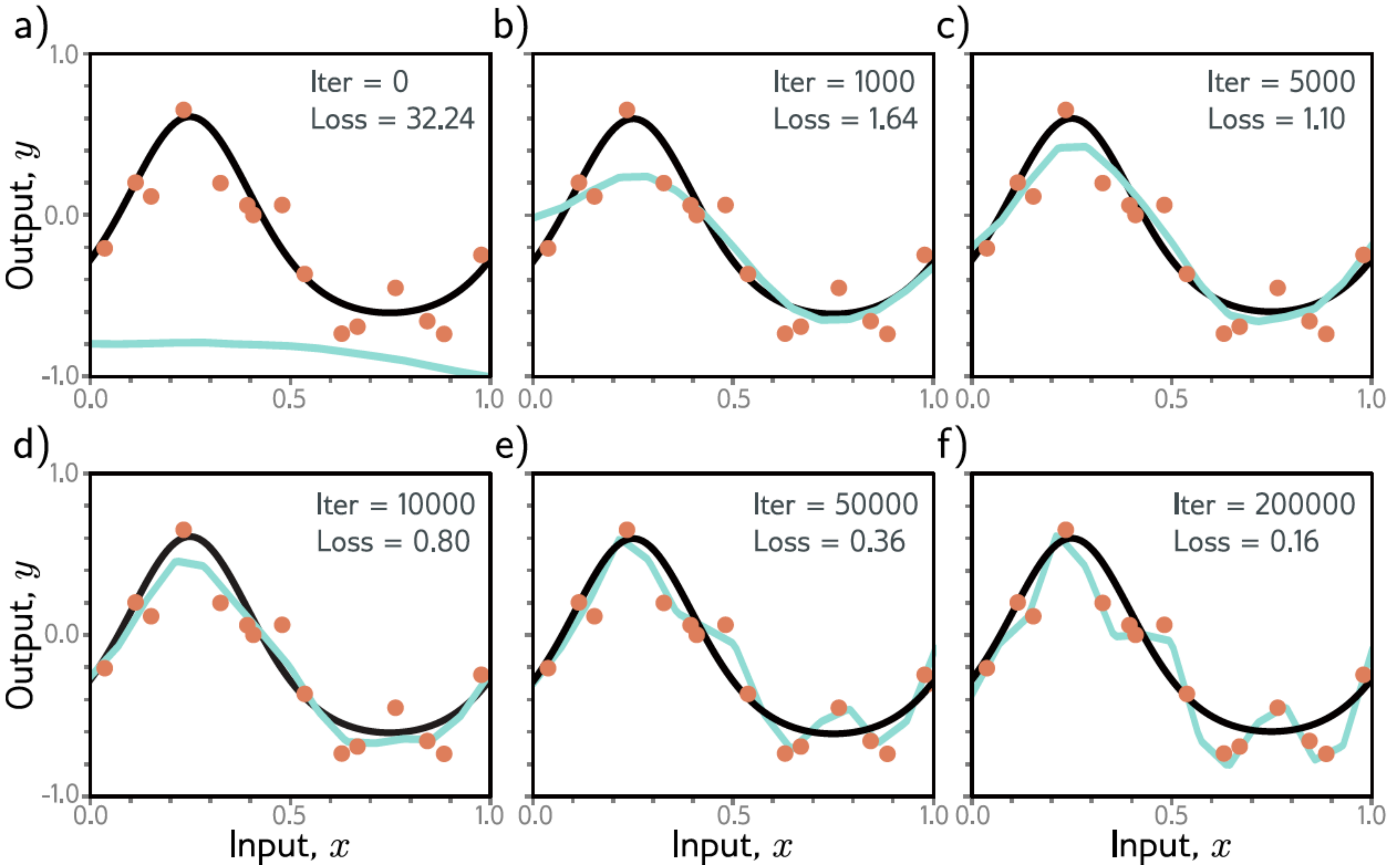
- Larger learning rates may lead to better generalization
- SGD seems to favor places where gradients are stable (all batches aggre on slope)
- SGD generalizes better than GD
- Smaller batches in SGD generally perform better than larger ones

Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Early stopping

- If we stop training early, weights don't have time to overfit to noise
- Weights start small, don't have time to get large
- Reduces effective model complexity
- Known as early stopping
- Don't have to re-train with different hyper-parameters – just "checkpoint" regularly and pick the model with lowest validation loss



Simplified shallow network model with 14 linear regions initialized randomly (cyan curve in (a)) and trained with SGD using a batch size of five and a learning rate of 0.05.

Regularization

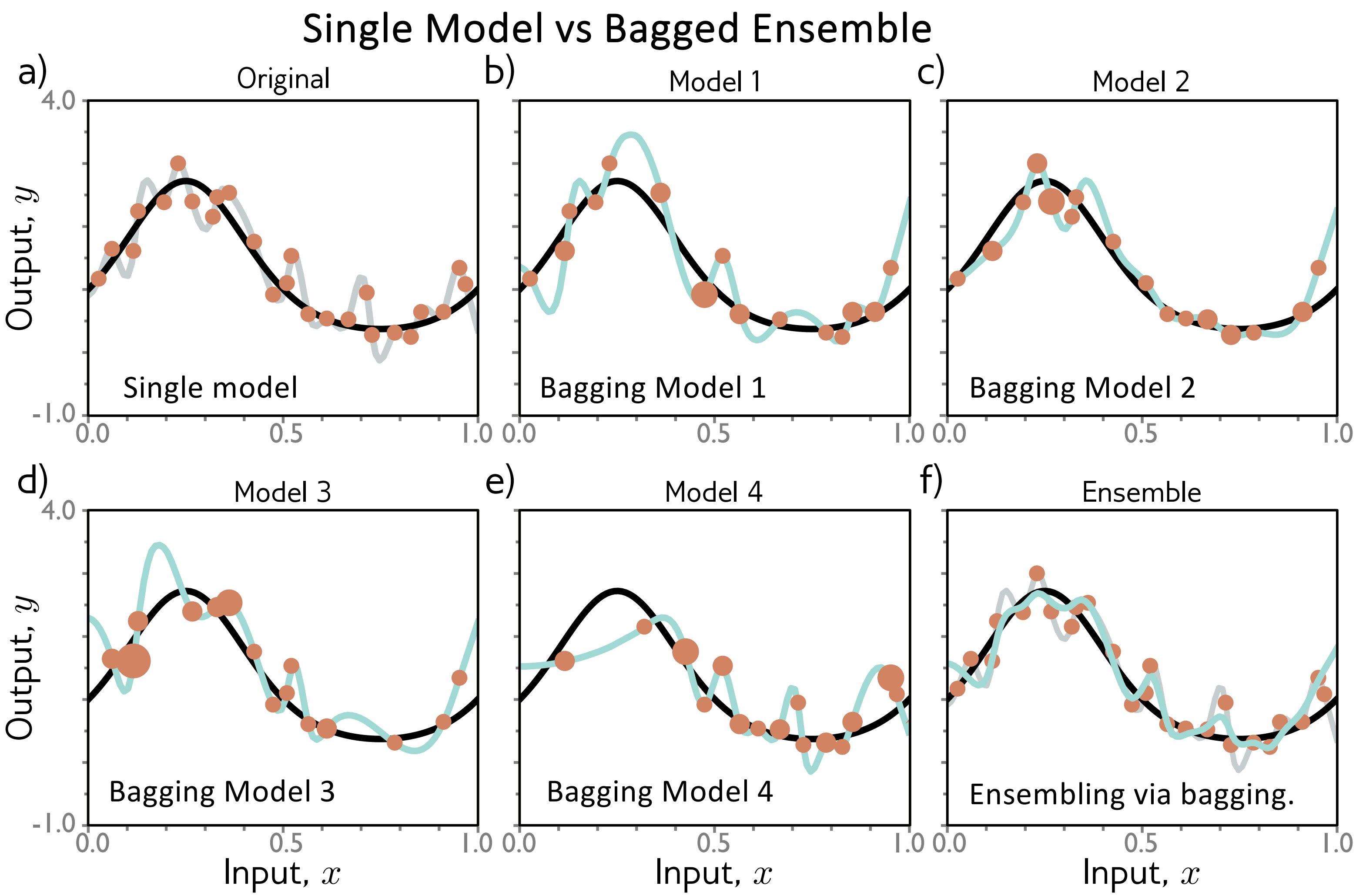
- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Ensembling

- Combine several models – an ensemble
- Combining outputs

	Mean	Median/Frequent (Robust)
Regression	Mean of outputs	Median of outputs
Classification	Mean before softmax	Most frequent predicted class

- Can be simply different initializations or even different models
- Or train with different subsets of the data resampled with replacements – bootstrap aggregating (bagging)



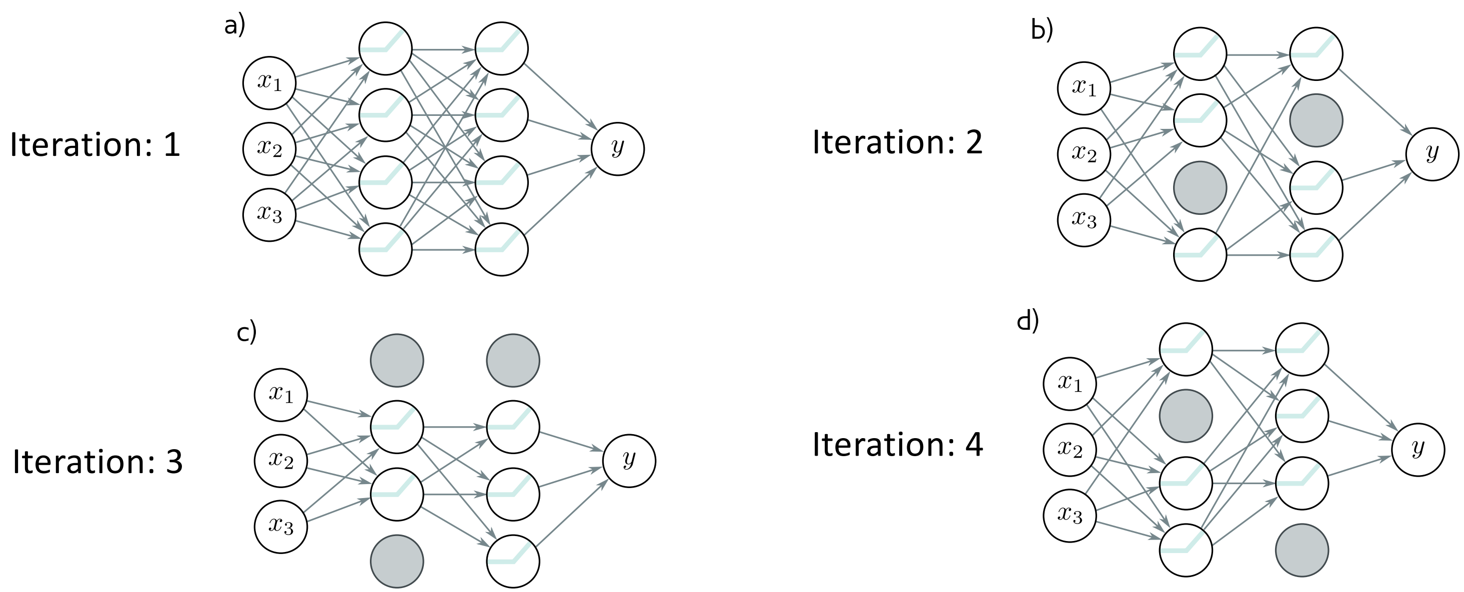
Size of orange point indicates number of times the data point was re-sampled.

Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

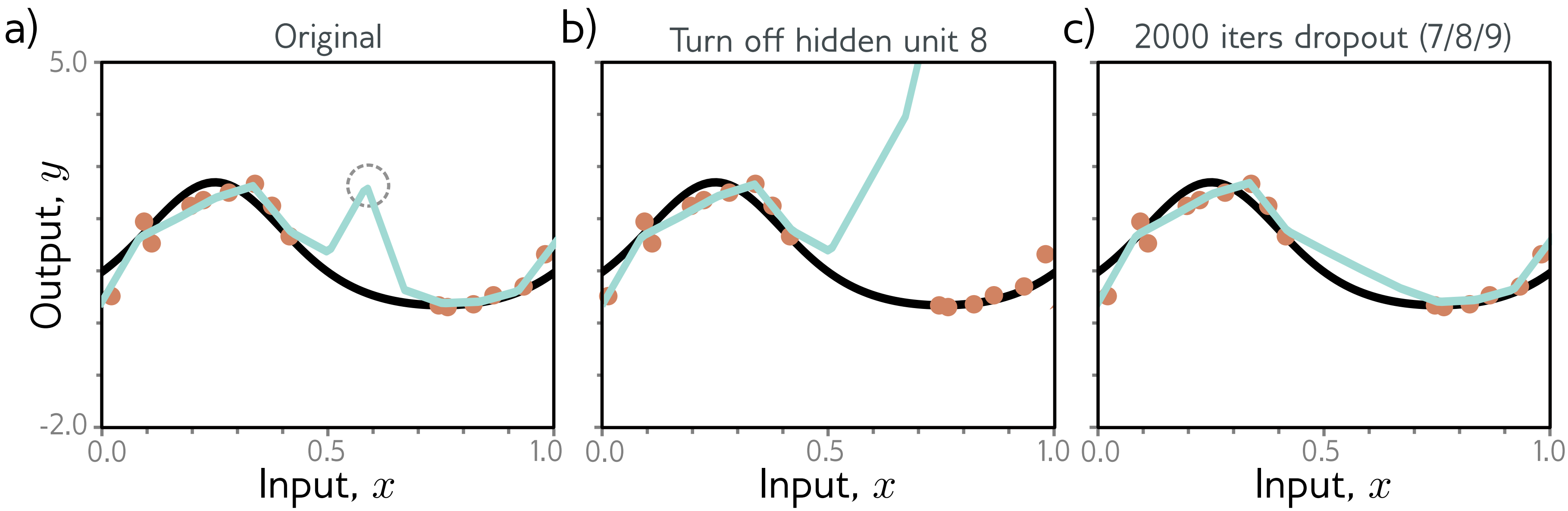
Dropout

Randomly clamp ~50% of hidden units to 0 on each iteration.



- Makes the network less dependent on any given hidden unit.
- At test time, all hidden units are active, which was not the case during training
  - Must rescale using *weight scaling inference rule* – multiple weights by (1 – dropout probability)

Dropout



- Prevents situations where subsequent hidden units correct for excessive swings from earlier hidden units
- Can eliminate kinks in function that are far from data and don't contribute to training loss

Monte Carlo Dropout for Inference

- Run the network multiple times with different random subsets of units clamped to zero (as in training)
- Combine the results using an ensembling method

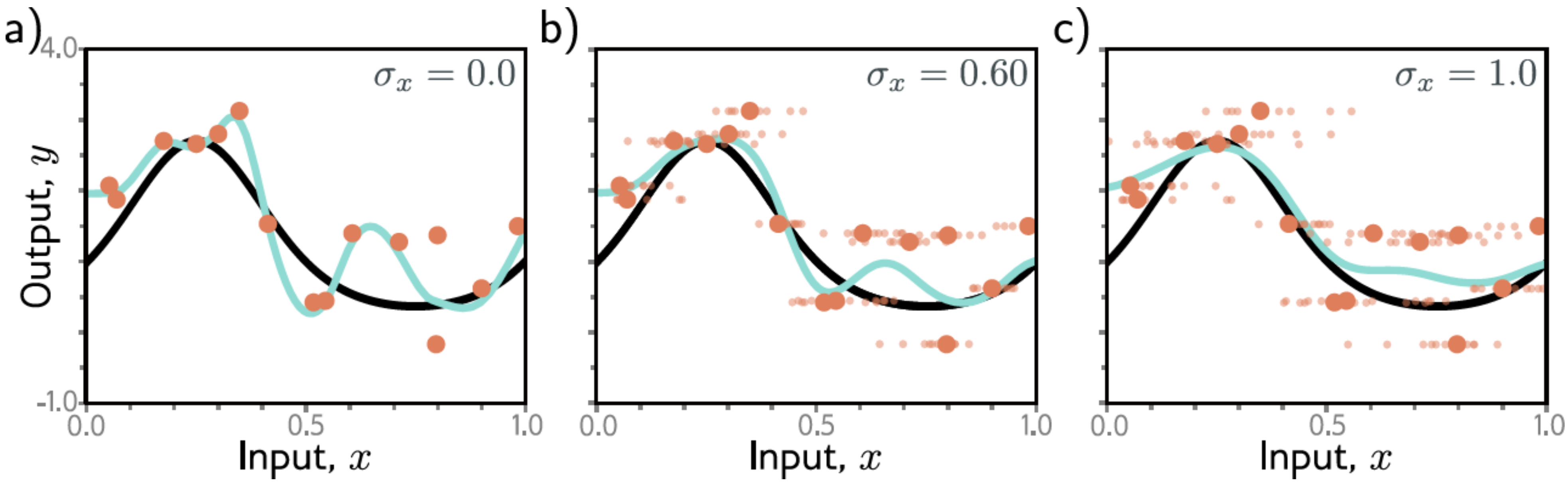
- This is closely related to ensembling in that every random version of the network is a different model; however, we do not have to train or store multiple networks here.

Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Adding noise

Adding noise to input with different variances.



- to inputs – induces weight regularization (see Exercise 9.3 in UDL)
- to weights – makes robust to small weight perturbations
- to outputs (labels) – reduces “overconfident” probability for target class

Regularization

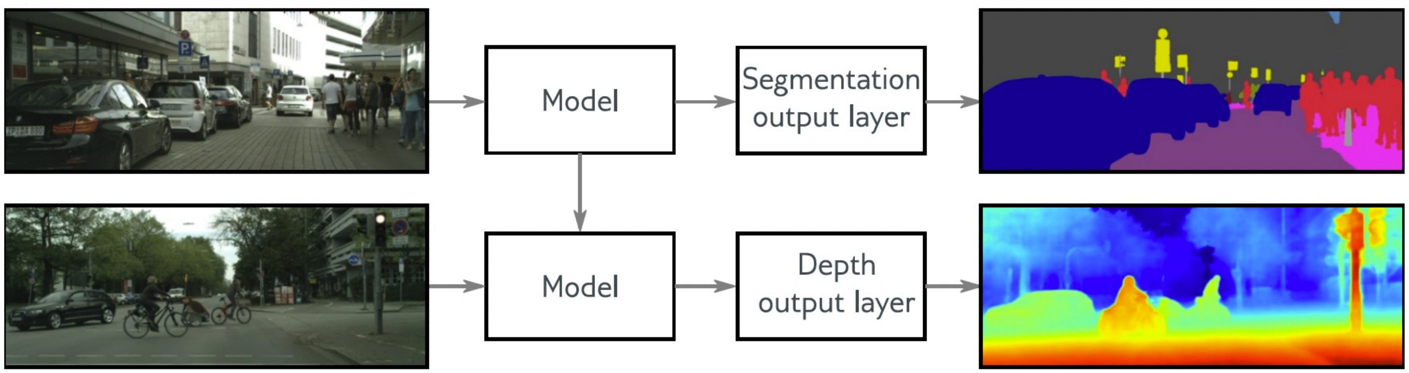
- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Transfer & Multitask Learning, Augmentation

- Strictly speaking not regularization, but can help improve generalization when dataset sizes are limited

Transfer Learning

(1) Train the model for segmentation



Assume we have lots of segmentation training data

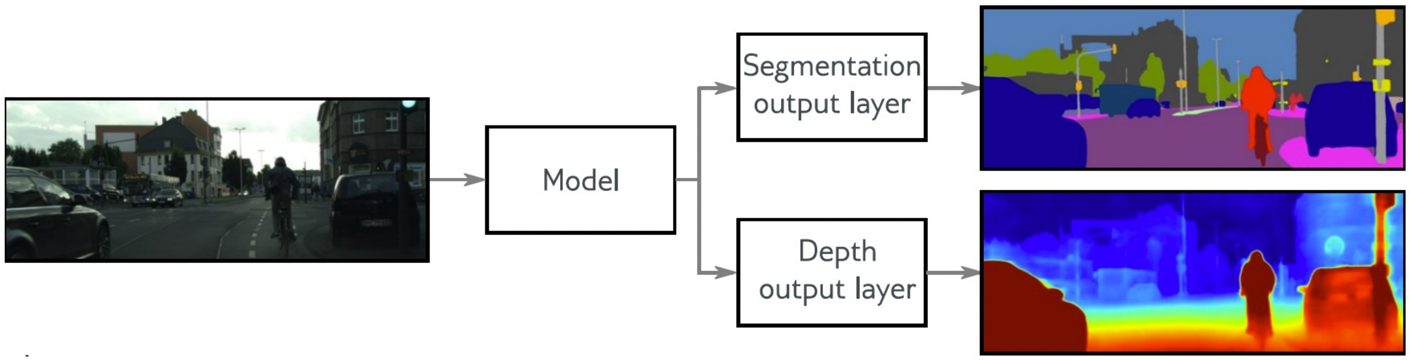
Assume we have limited depth training data

(2) Replace the final layers to match the new task and

(3) Either:

a) Freeze the rest of the layers and train the final layers
b) Fine tune the entire model

Multi-Task Learning



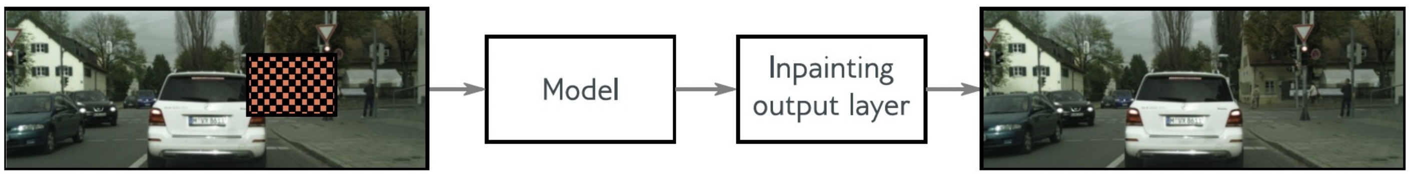
- Train the model for 2 or more tasks simultaneously
  - Weighted combo of loss fncs

$$L_{total} = \alpha \cdot L_{segmentaiton} + \beta \cdot L_{depth}$$

- Less likely to overfit to training data of one task

- Can be harder to get training to converge. Might have to vary the individual task loss weightings, α and β.

Self-Supervised Learning



The animal didn't cross the because it was too tired.

- Mask out part of the training data
- Train model to try to infer missing data
  - masked data is the target
- ➔ Model learns characteristics of the data

- Then apply transfer learning

Regularization

- Explicit regularization
- Implicit regularization
- Early stopping
- Ensembling
- Dropout
- Adding noise
- Transfer learning, multi-task learning, self-supervised learning
- Data augmentation

Data augmentation

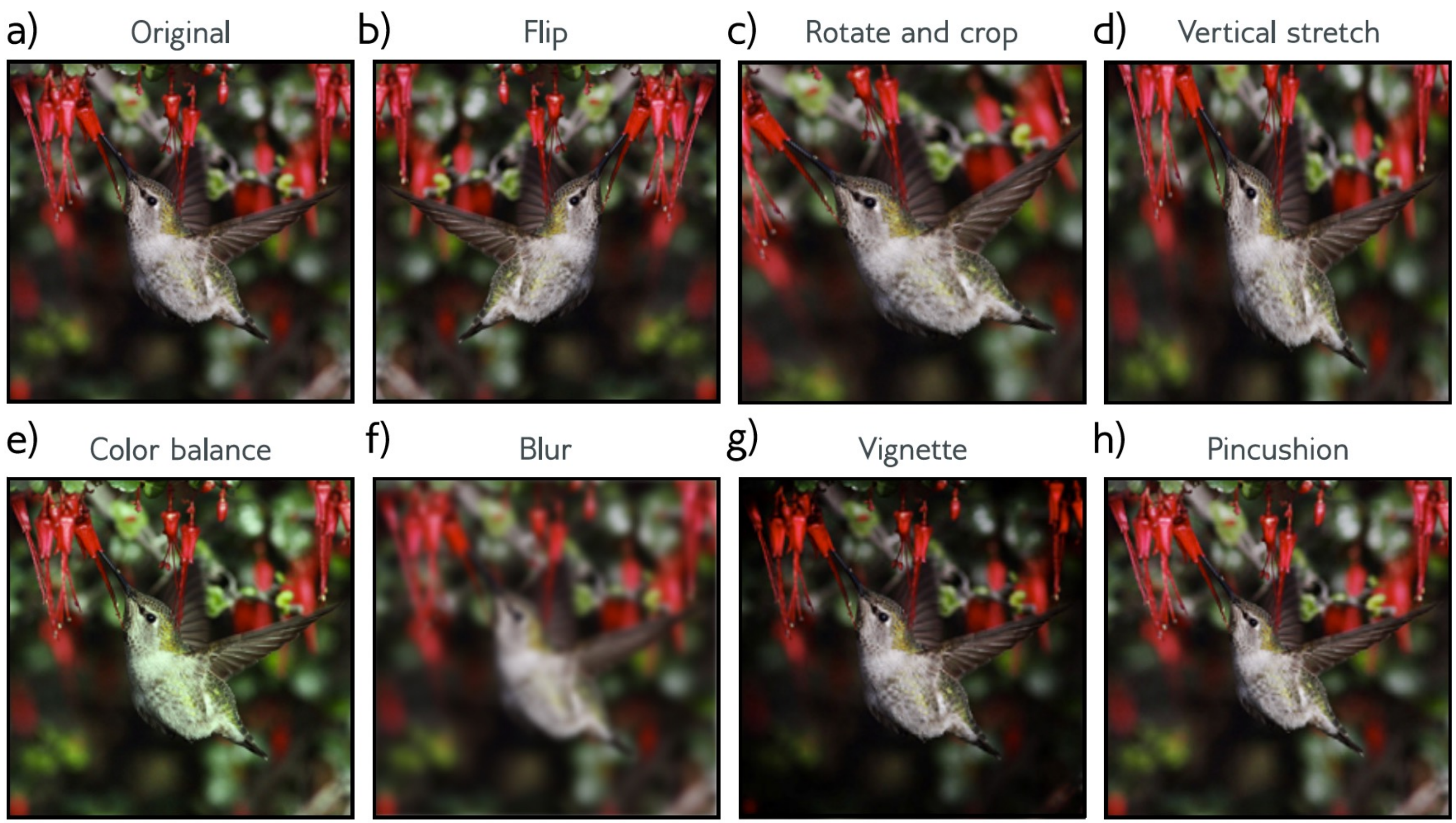


Image augmentation in PyTorch

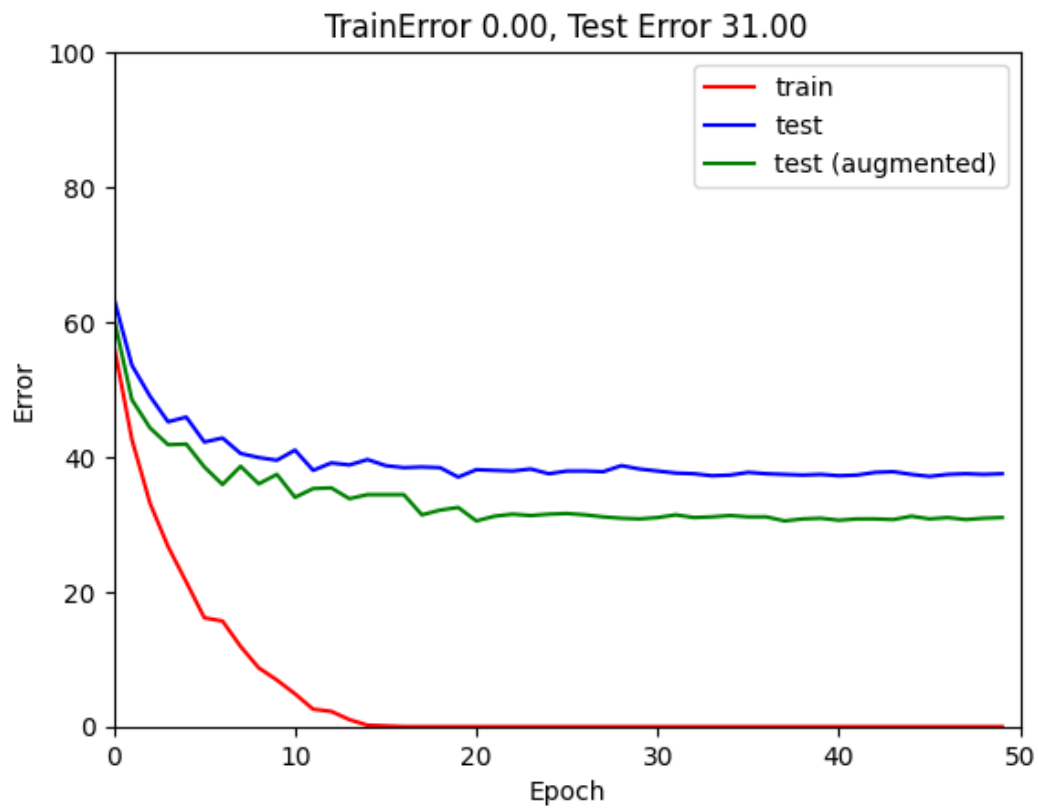
```
import torch
import torchvision.transforms as transforms


# Define augmentation pipeline
transform = transforms.Compose([
transforms.RandomHorizontalFlip(p=0.5),
transforms.RandomVerticalFlip(p=0.3),
transforms.RandomRotation(degrees=30),
transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5),
transforms.RandomAffine(degrees=20, translate=(0.2, 0.2), shear=10),
transforms.RandomPerspective(distortion_scale=0.5, p=0.5),
transforms.ToTensor(), # Convert image to tensor
])

# Apply transformations multiple times to visualize augmentation
augmented_image = transform(image)
```

https://pytorch.org/vision/main/transforms.html

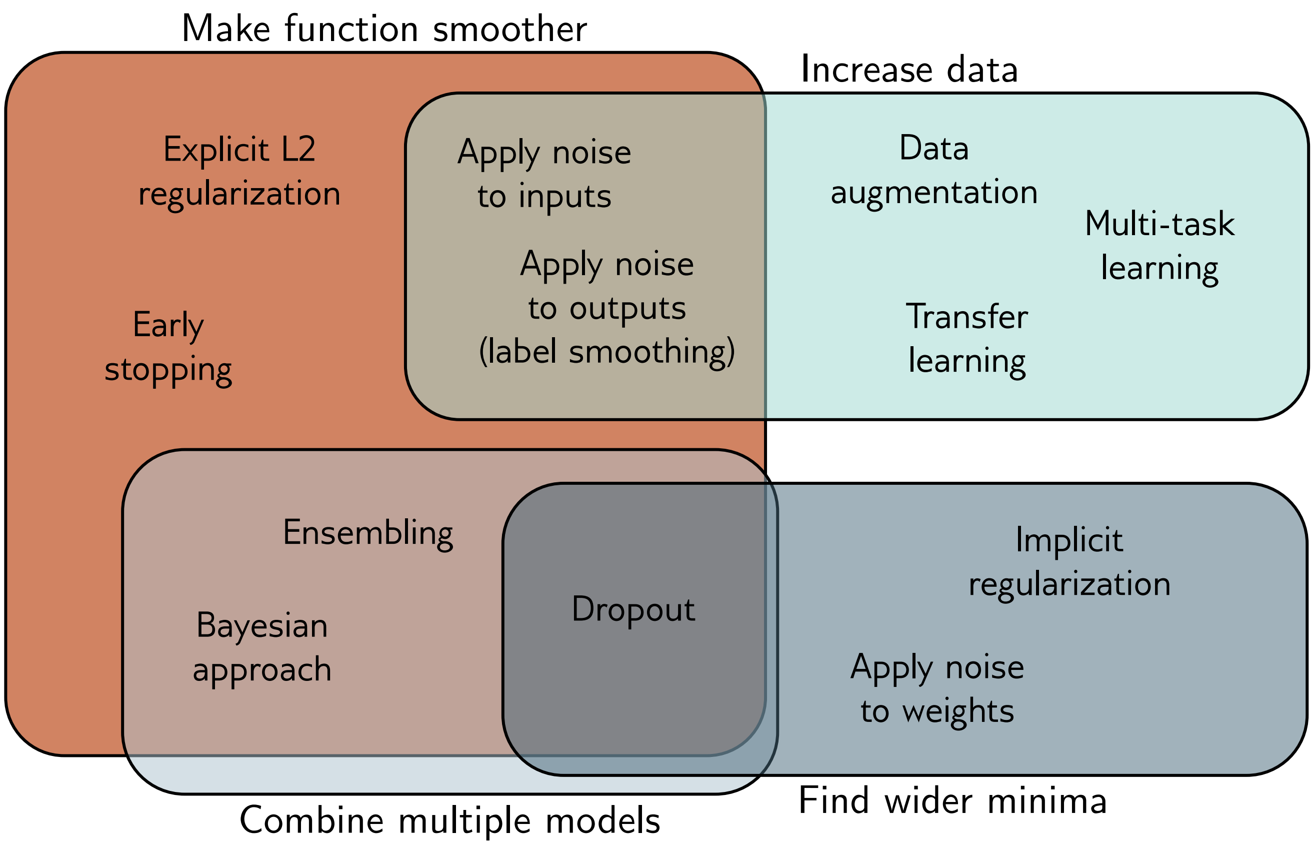
Data Augmentation: MNIST1D



Examples in training set: 4000
Examples in test set: 1000
Length of each example: 40

- Randomly circularly rotate
- randomly scale between 0.8 and 1.2

Regularization overview



Feedback?

Link