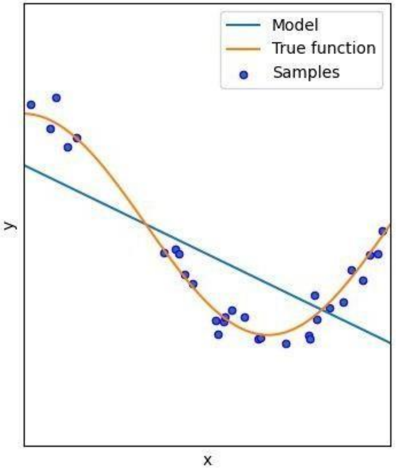
Recitation 8-11/04/2022

Overfitting & Regularization

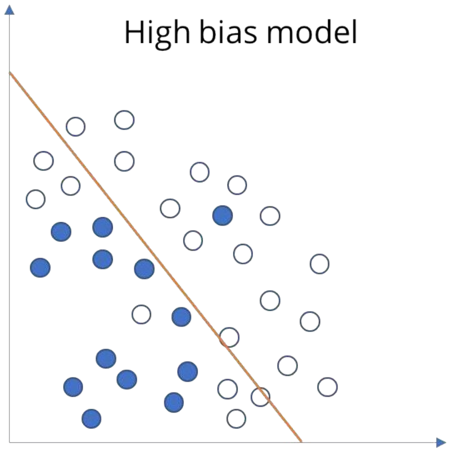
TA: Akshay Antony

Underfitting

- Underfitting is when the model is **unable to fit well** to the training data.
  - This happens when the model is very simple
  - We lose accuracy.
  - Also if training data is very lov
  - Known as Bias



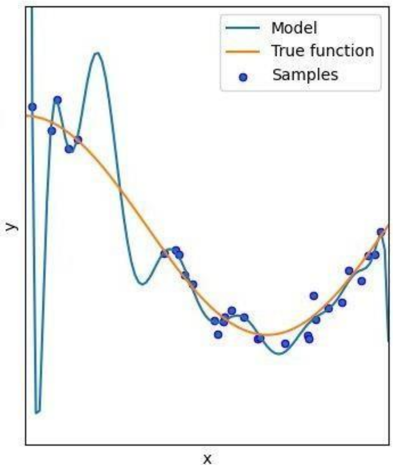
Regression



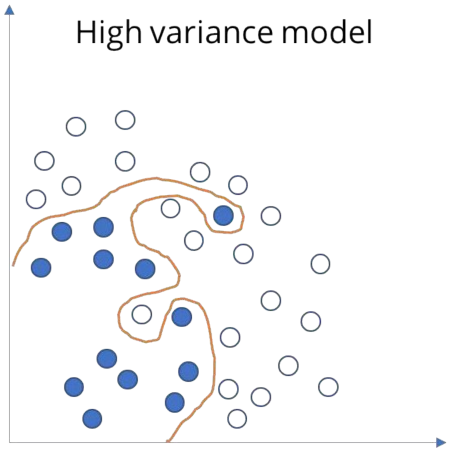
Classification

Overfitting

- Overfitting is when the model **over-fits** to the training data.
  - This happens when we choose a complex model even when a simpler model can infer the ground truth well
  - We lose generalizability.
  - Also if training data is very specific.
  - Known as Variance



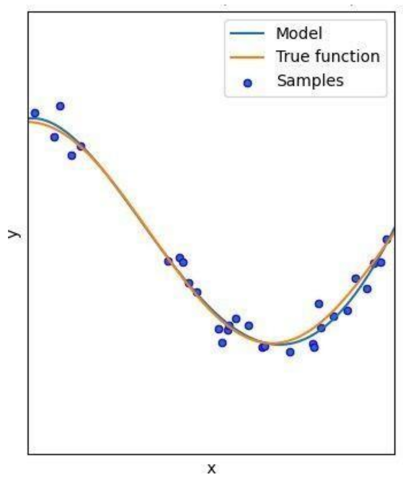
Regression



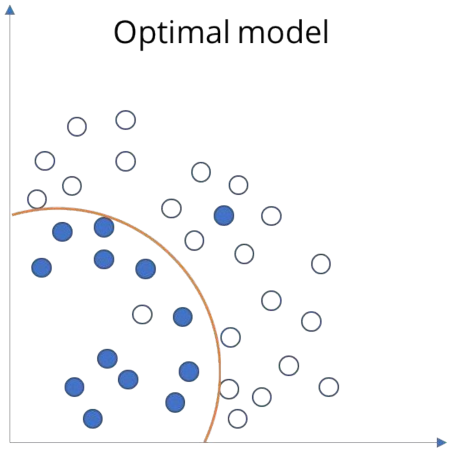
Classification

Just Right

- Just right/optimal is when the model neither **over** nor **under** fits.

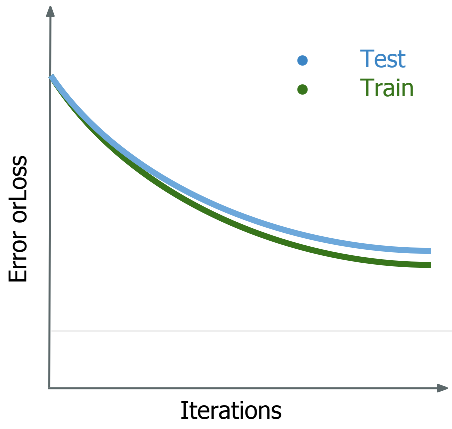


Regression

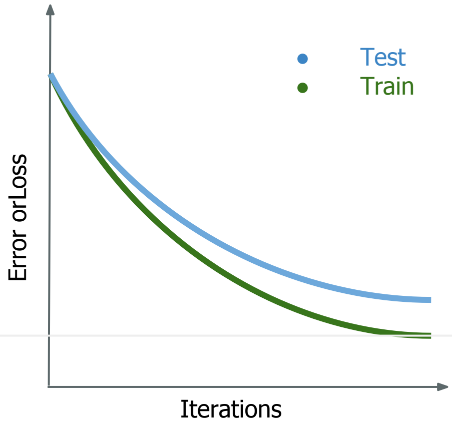


Classification

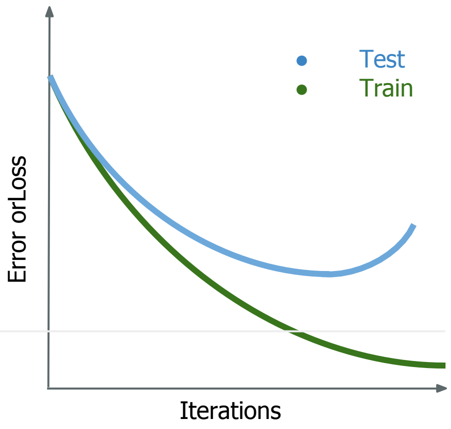
How to visualize during training/testing



Underfitting

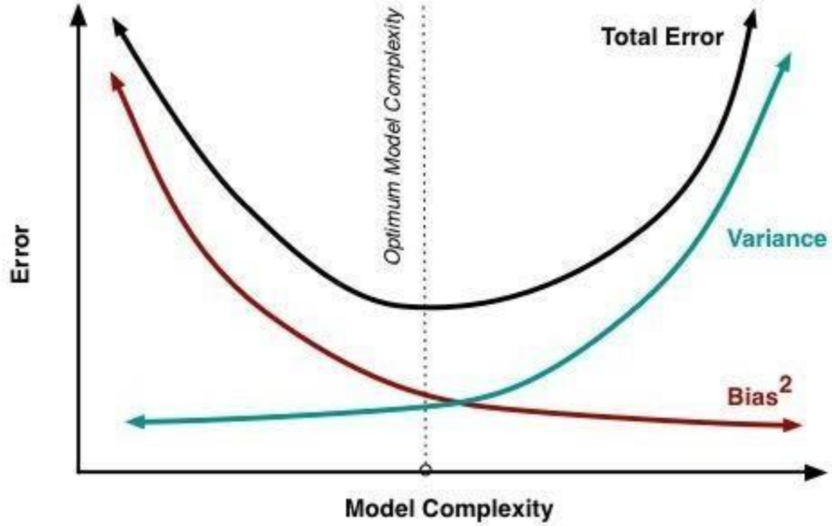


Just
Right



Overfitting

Complexity of model



Regularization (to prevent overfitting)

Let's assume:

$$H = \theta_1 * X_1 + \theta_2 * X_2$$

If we are overfitting, it is likely because the model is giving really high importance to features, some of which might not be even useful.

So if we can reduce θ (coefficients of the features) we can reduce the effect each feature has on the output.

Thus regularization is a way of capping the weights so they don't grow too much.

L1 Regularization

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term to the loss function.

- Expression: $\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$
- if λ=0 reduces to unregularized case
- Can make weights go to 0 (derivative does not contain the weight term)
- More expensive
- Not closed form solution

L2 Regularization

Ridge regression adds "squared magnitude" of coefficient as penalty term to th $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$.

- Expression:
- Cannot make weights to zero (derivative depends on the weight)
- Less expensive
- Closed form (differentiable at every point)

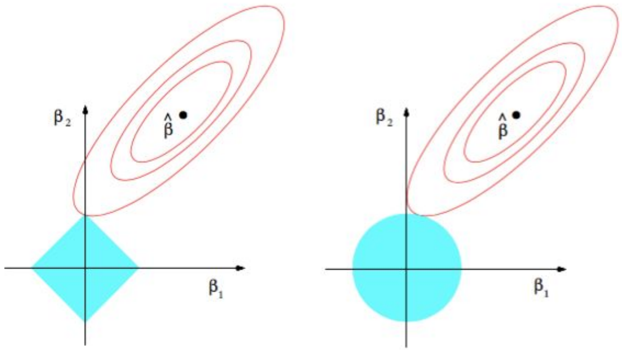
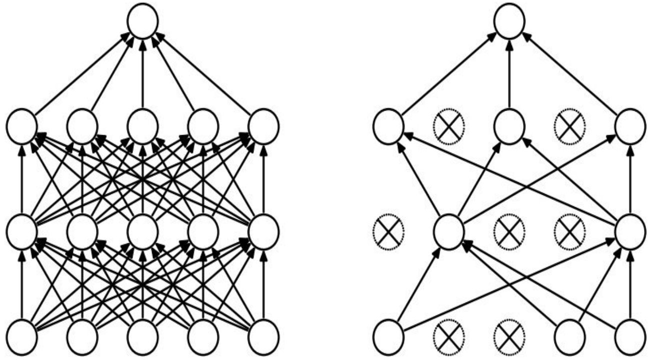


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

Dropout

1. Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.
2. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.
3. Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect



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

Pytorch nn.Dropout

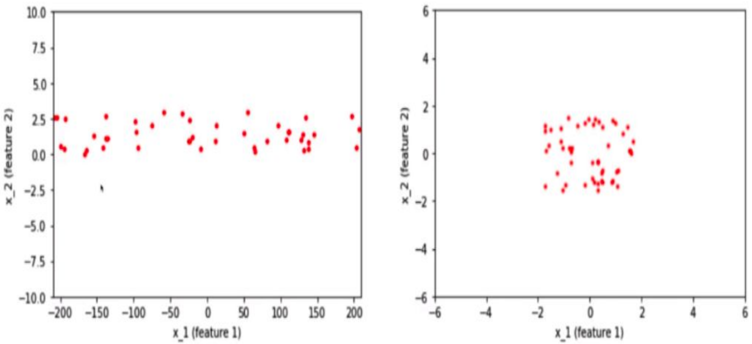
- torch.nn.Dropout(p=0.5, inplace=False)
- p (float) – probability of an element to be zeroed. Default: 0.5
- Scaled by during training factor of 1/(1-p)
- If model.eval() dropout is deactivated

Where Not to use:

- Just before final linear layer
- For small networks
- Not training for large number of iterations

Batchnorm as regularizer

- Normalizing the inputs to the layer has an effect on the training of the model, dramatically reducing the number of epochs required.
- In pytorch BatchNorm learns a mean and std for each BatchNorm layer
- During training this mean and std acts as noise and hence a regularizer
- In eval mode the learned mean and std is used



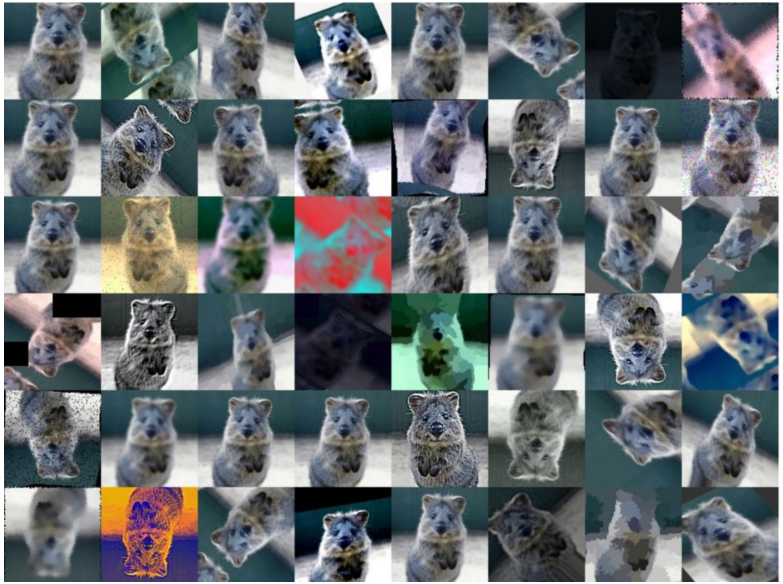
Before standardization

After standardization

Data Augmentation

Images (torchvision.transforms)

1. Simple transformations
   - Resize: Resize(size)
   - Gray Scale: Grayscale(), to_grayscale()
   - Normalize: Normalize(mean, std)
   - Random Rotation: RandomRotation(degree)
   - Center Crop: CenterCrop(size)
   - Random Crop: RandomCrop(size)



Random Rotation

- torchvision.transforms.RandomRotation(degrees)
- degrees (sequence or number) – Range of degrees to select from

GrayScale

- torchvision.transforms.Grayscale

Normalize

- torchvision.transforms.Normalize(mean, std)
- mean: mean of size (C, 1) C number of channels in input image
- std: std of size (C, 1) C number of channels in input image
- mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] for imagenet, use this while doing transfer learning

RandomResize

- torchvision.transforms.RandomResizedCrop
- Crop a random portion of image and resize it to a given size.

Flips

- RandomHorizontalFlip: argument probability to flip
- RandomVerticalFlip: argument probability to flip

Custom Augmentation

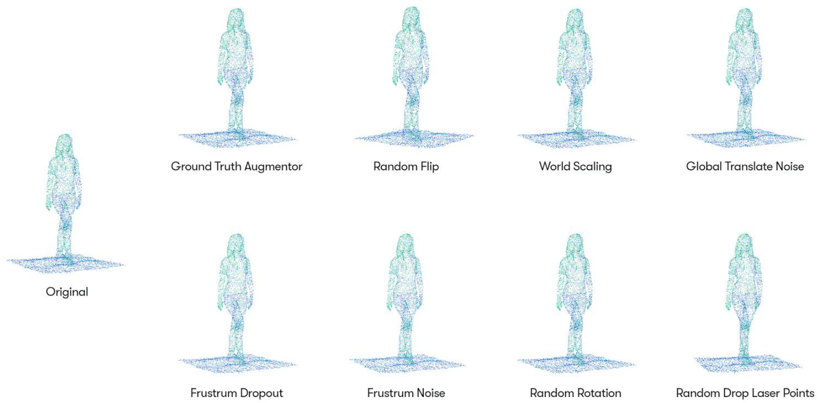
- Write a class capable of using transforms.Compose
- Define a function named __call__ (object() is shorthand for object.__call__())

```
class RandomHorizontalFlip(object):
    def __init__(self,p=0.5):
        self.p=p
    def __call__(self,sample):
        image, target=sample['image'],
        sample['target']
        if random.random()>self.p:
            trans=tt.RandomHorizontalFlip(1)
            image=trans(image)
            if target!=0:
                target=-1*target
        return {'image':image,'target':target}
```

Data Augmentation

Point Clouds:

- Write custom transforms
- Random Rotation
- Random flipping
- Adding Noise
- Random scaling
- Drop Points

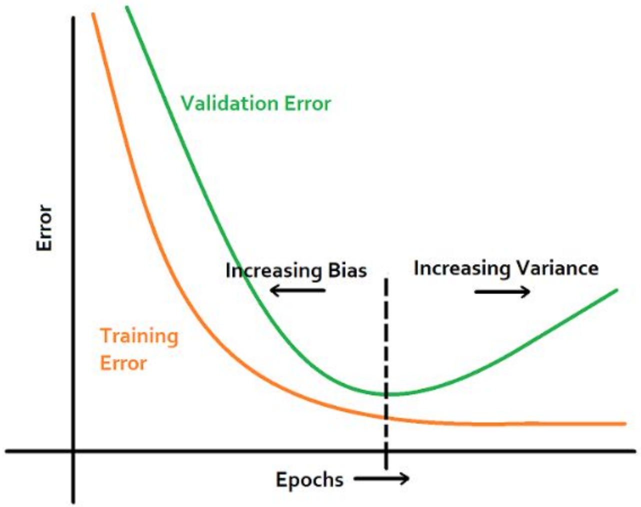


Early Stopping

- When training a large network, there will be a point during training when the model will stop generalizing and start learning the statistical noise in the training dataset.
- If the performance of the model on the validation dataset starts to degrade (e.g. loss begins to increase or accuracy begins to decrease), then the training process is stopped.
- Early stopping may be thought of as a type of "implicit" regularization, much like using a smaller network that has less capacity.

Criteria:

- val_loss - train_loss > threshold
- val_loss does not improve



Available in pytorch-lightning

```
EarlyStopping(monitor="val_accuracy", min_delta=0.00, patience=3, verbose=False, mode="max")
```

1. monitor:loss needed to focus on
2. min_delta: amount of change to consider as an improvement
3. patience: number iteration to wait until stopping