Regularization for Deep Learning

Deep Learning, Ian Goodfellow

Jinhwan Suk, Department of Mathematical Science, KAIST

Noise Robustness

Input noise = data augmentation

Adding noise to the weights : used in RNN

- This can be interpreted as a Bayesian inference over the weights.

Under some assumptions, noise applied to weights can be interpreted as a more traditional form of regularization.

In Bayesian View Point

$$P(w \mid D) = \frac{P(D \mid w)P(w)}{P(D)}$$

Prior for distribution of weight : $w \sim \mathcal{N}(0, \sigma^2 I)$

$$\begin{aligned} w^{MAP} &= \arg\max_w \log P(D \mid w) + \log P(w) \\ &= \arg\max_w \log P(D \mid w) - \frac{\|w\|^2}{2\sigma^2} \\ &= \arg\min_w - \log P(D \mid w) + \frac{\|w\|^2}{2\sigma^2} \end{aligned}$$

Consider the regression setting, where we wish to train a function $\hat{y}(x)$.

$$J = \mathbb{E}_{p(x,y)}[(\hat{y}(x) - y)^2]$$

We now assume that with each input presentation we also include a random perturbation $\epsilon_W \sim N(\epsilon; 0, \eta I)$ of the network weights.

$$\tilde{J}_W = \mathbb{E}_{p(x,y,\epsilon_W)}[(\hat{y}_{\epsilon_W} - y)^2]$$

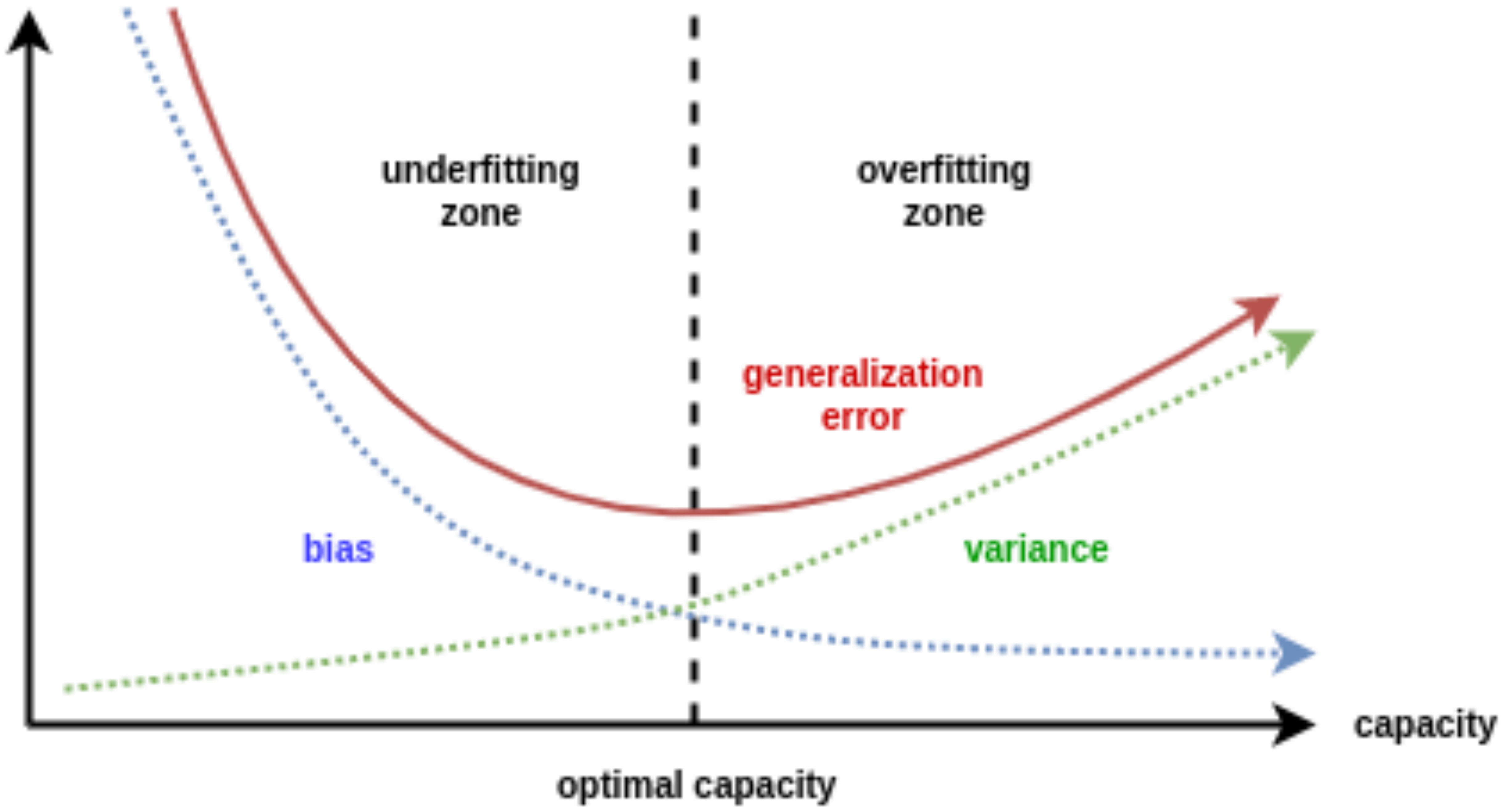
For small η, using Taylor's theorem,

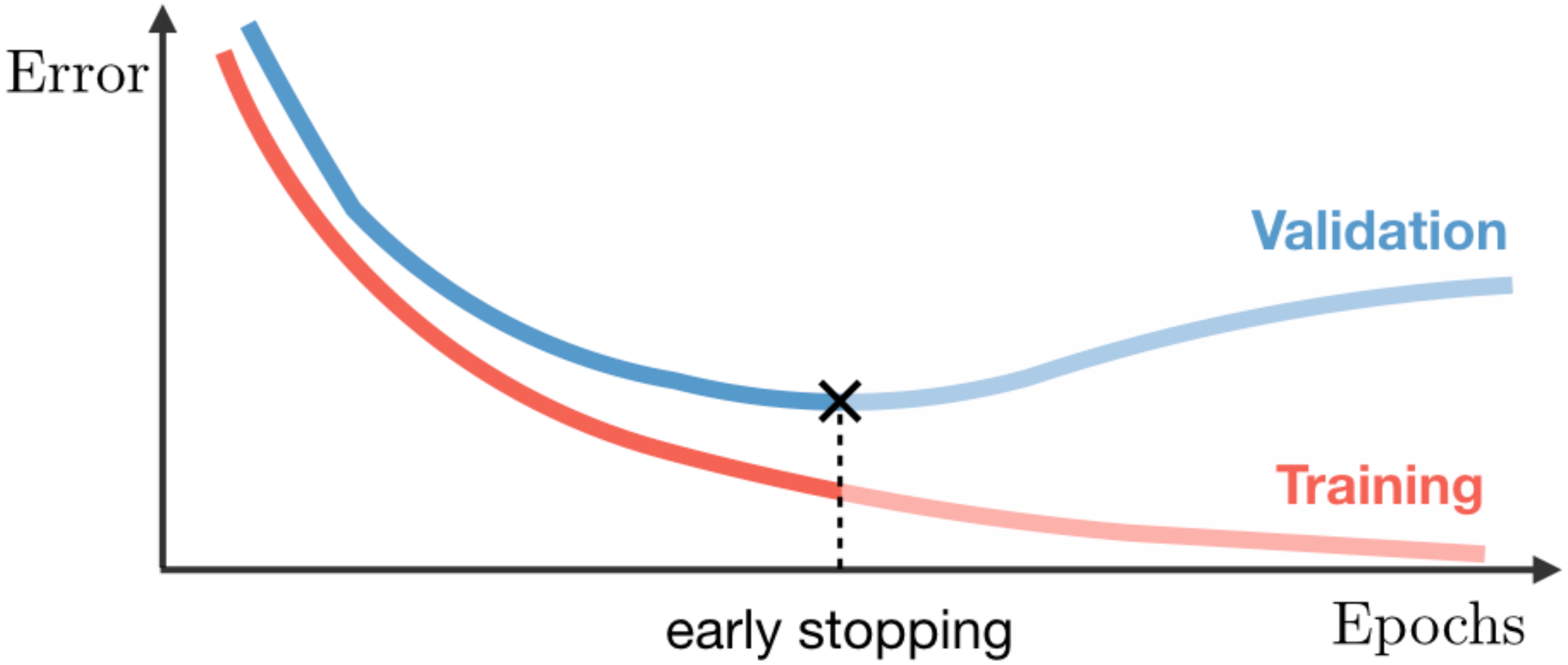
$\hat{y}_{\varepsilon_W}(x) = \hat{y}(x) + \nabla_W \hat{y}(x)\varepsilon + o(\|\varepsilon\|^2)$

$$
\begin{aligned}
\mathbb{E}_{p(x,y,\varepsilon_W)}[(\hat{y}_{\varepsilon_W}(x) - y)^2] &= \mathbb{E}_{p(x,y,\varepsilon_W)}[(\hat{y}_{\varepsilon_W}(x) - \hat{y}(x) + \hat{y}(x) - y)^2] \\
&= \mathbb{E}_{p(x,y,\varepsilon_W)}[(\hat{y}_{\varepsilon_W}(x) - \hat{y}(x))^2] + \mathbb{E}_{p(x,y,\varepsilon_W)}[(\hat{y}_{\varepsilon_W}(x) - \hat{y}(x))(\hat{y}(x) - y)] + \mathbb{E}_{p(x,y)}[(\hat{y}(x) - y)^2] \\
&= \mathbb{E}_{p(x,y,\varepsilon_W)}[\|\nabla_W \hat{y}(x)\varepsilon\|^2] + \mathbb{E}_{p(x,y,\varepsilon_W)}[\nabla_W \hat{y}(x)\varepsilon(\hat{y}(x) - y)] + J \\
&= \mathbb{E}_{p(x,y,\varepsilon_W)}[\sum_j \|(\nabla_W \hat{y}(x))_j \varepsilon_j\|^2] + J \\
&= \sum_j \mathbb{E}_{p(x,y,\varepsilon_W)}[\|(\nabla_W \hat{y}(x))_j\|^2 \|\varepsilon_j\|^2] + J \\
&= \sum_j \mathbb{E}_{p(x,y)}[\|(\nabla_W \hat{y}(x))_j\|^2] \mathbb{E}_{p(\varepsilon)}[\|\varepsilon_j\|^2] + J \\
&= \eta \mathbb{E}_{p(x,y)}[\sum_j \|\nabla_W \hat{y}(x)_j\|^2] + J \\
&= \eta \mathbb{E}_{p(x,y)}[\|\nabla_W \hat{y}(x)\|^2] + J
\end{aligned}
$$

Early Stopping

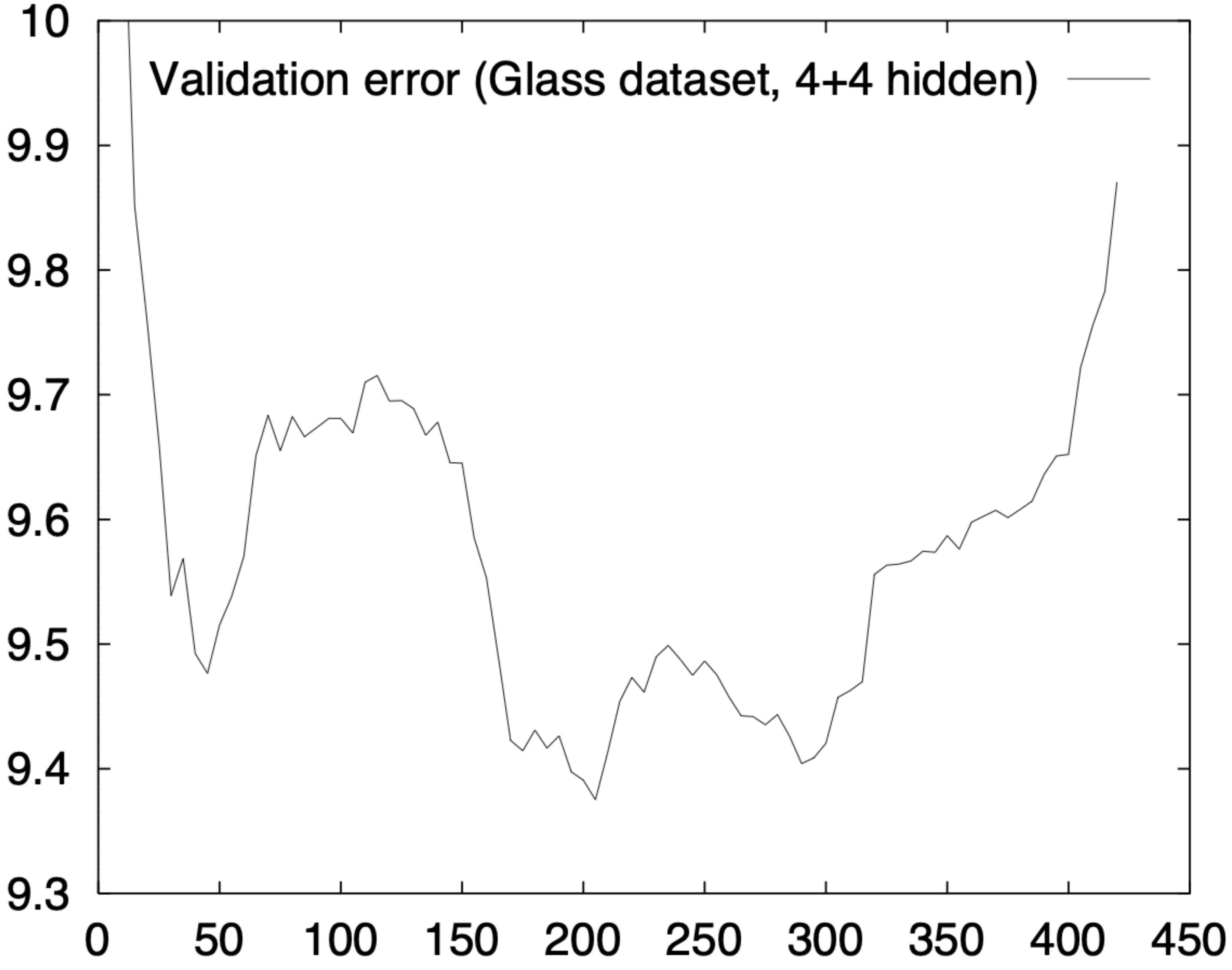
- When do we stop training??
    - Run the validation set evaluation periodically during training
    - Computational cost
- Early stopping requires a validation set





Early Stopping

But When??



Early Stopping — but when?

Lutz Prechelt (prechelt@ira.uka.de)

Fakultät für Informatik; Universität Karlsruhe
D-76128 Karlsruhe; Germany

Abstract. Validation can be used to detect when overfitting starts during supervised training of a neural network; training is then stopped before convergence to avoid the overfitting ("early stopping"). The exact criterion used for validation-based early stopping, however, is usually chosen in an ad-hoc fashion or training is stopped interactively. This trick describes how to select a stopping criterion in a systematic fashion; it is a trick for either speeding learning procedures or improving generalization, whichever is more important in the particular situation. An empirical investigation on multi-layer perceptrons shows that there exists a tradeoff between training time and generalization: From the given mix of 1296 training runs using different 12 problems and 24 different network architectures I conclude slower stopping criteria allow for small improvements in generalization (here: about 4% on average), but cost *much* more training time (here: about factor 4 longer on average).

1 Early stopping is not quite as simple

1.1 Why early stopping?

When training a neural network, one is usually interested in obtaining a network with optimal generalization performance. However, all standard neural network architectures such as the fully connected multi-layer perceptron are prone to overfitting [10]: While the network *seems* to get better and better, i.e., the error on the training set decreases, at some point during training it actually begins to get worse again, i.e., the error on unseen examples increases. The idealized expectation is that during training the generalization error of the network evolves as shown in Figure 1. Typically the generalization error is estimated by a validation error, i.e., the average error on a *validation set*, a fixed set of examples not from the training set.

There are basically two ways to fight overfitting: reducing the number of dimensions of the parameter space or reducing the effective size of each dimension. Techniques for reducing the number of parameters are greedy constructive learning [7], pruning [5, 12, 14], or weight sharing [18]. Techniques for reducing the size of each parameter dimension are regularization, such as weight decay [13] and others [25], or early stopping [17]. See also [8, 20] for an overview and [9] for an experimental comparison.

Early stopping is widely used because it is simple to understand and implement and has been reported to be superior to regularization methods in many cases, e.g. in [9].

Early Stopping

How early stopping acts as a regularizer :

Early stopping is equivalent to L^2 *regularization.*

$$\theta = w; \quad w* = argmin\, J(w)$$

$$\hat{J}(\theta) = J(w*) + \frac{1}{2}(w - w*)^T H_{w*}(w - w*) + o(\|w - w*\|^2)$$

$$\nabla_w \hat{J}(w) = H_{w*}(w - w*)$$

Set $w^{(0)} = 0$; $\tau =$ optimization step ; $\epsilon =$ learning rate.

$$\begin{aligned} w^{(\tau)} &= w^{(\tau-1)} - \epsilon \nabla_w \hat{J}(w^{(\tau-1)}) \\ &= w^{(\tau-1)} - \epsilon H_{w*}(w^{(\tau-1)} - w*) \end{aligned}$$

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

$$w^{(\tau)} - w^* = (I - \epsilon Q \Lambda Q^T)(w^{(\tau-1)} - w^*)$$
$$Q^T(w^{(\tau)} - w^*) = (I - \epsilon \Lambda) Q^T (w^{(\tau-1)} - w^*)$$

Assume that ϵ is chosen to be small enough to guarantee $|1 - \epsilon \lambda_i| < 1$.

Then,

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

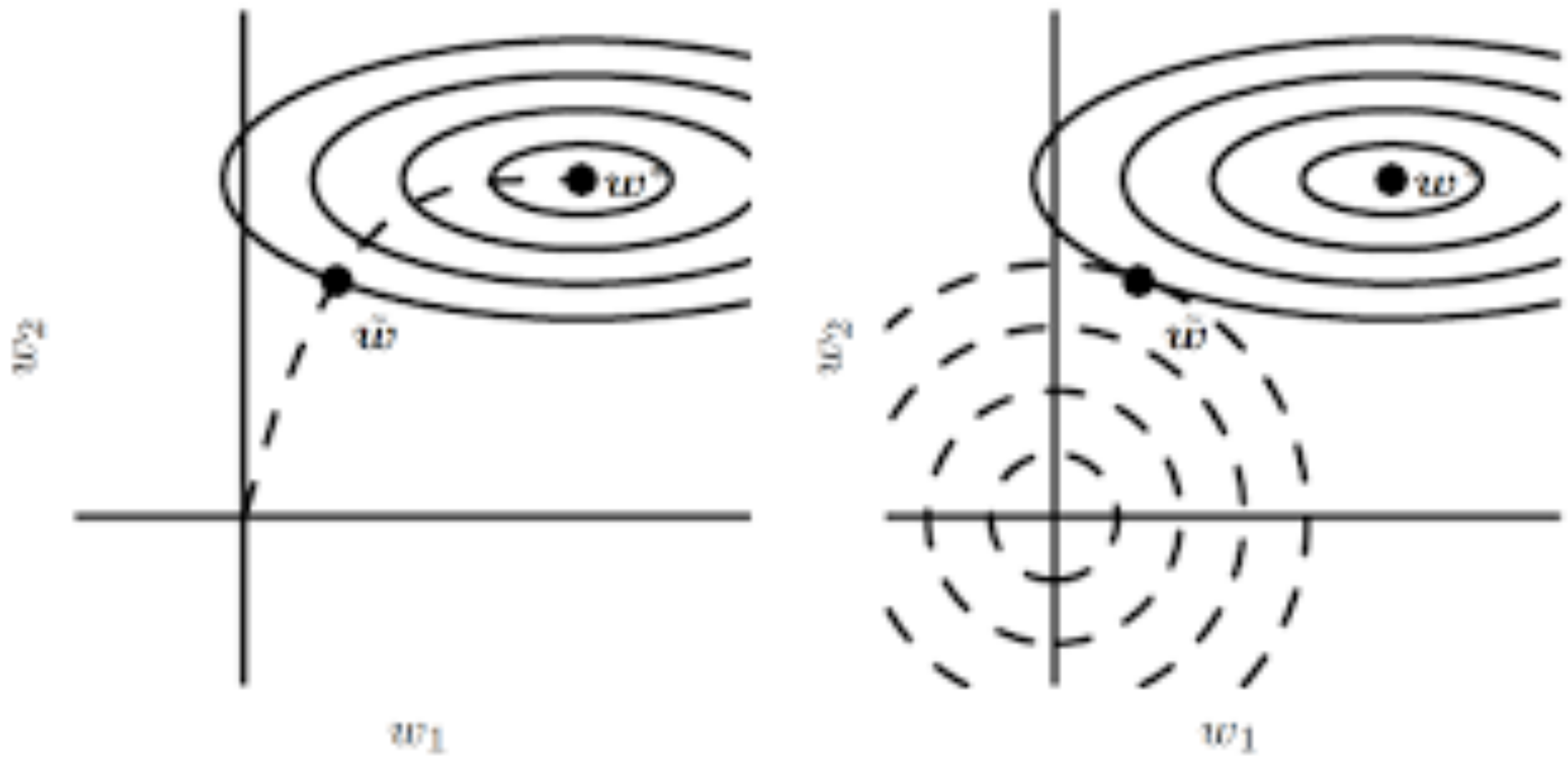
Recall the expression for $\tilde{w}$ for L^2 regularization.

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

If the hyper parameters ϵ, α, and τ are chosen such that

$$(I - \epsilon \Lambda)^\tau = (\Lambda + \alpha I)^{-1} \alpha,$$

then L^2 regularization and early stopping can be seen as equivalent.


w^*
$\tilde{w}$
w_2
w_1
w^*
$\tilde{w}$
w_2
w_1

Define the Network

Defining a simple MLP model.

In [3]:

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, 10)
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        # flatten image input
        x = x.view(-1, 28 * 28)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        # add hidden layer, with relu activation function
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        # add output layer
        x = self.fc3(x)
        return x

# initialize the NN
model = Net()
print(model)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
  (dropout): Dropout(p=0.5)
)
```

Specify Loss Function and Optimizer

In [4]:
```
# specify loss function
criterion = nn.CrossEntropyLoss()

# specify optimizer
optimizer = torch.optim.Adam(model.parameters())
```

Import the Early Stopping Class

In [5]:
```
# import EarlyStopping
from pytorchtools import EarlyStopping
```

Train the Model using Early Stopping

In [6]:
```
def train_model(model, batch_size, patience, n_epochs):

    # to track the training loss as the model trains
    train_losses = []
    # to track the validation loss as the model trains
    valid_losses = []
    # to track the average training loss per epoch as the model trains
    avg_train_losses = []
    # to track the average validation loss per epoch as the model trains
    avg_valid_losses = []

    # initialize the early_stopping object
    early_stopping = EarlyStopping(patience=patience, verbose=True)

    for epoch in range(1, n_epochs + 1):

        ###################
        # train the model #
        ###################
        model.train() # prep model for training
        for batch, (data, target) in enumerate(train_loader, 1):
            # clear the gradients of all optimized variables
            optimizer.zero_grad()
            # forward pass: compute predicted outputs by passing inputs to the model
            output = model(data)
            # calculate the loss
            loss = criterion(output, target)
            # backward pass: compute gradient of the loss with respect to model parameters
            loss.backward()
            # perform a single optimization step (parameter update)
            optimizer.step()
            # record training loss
            train_losses.append(loss.item())
```

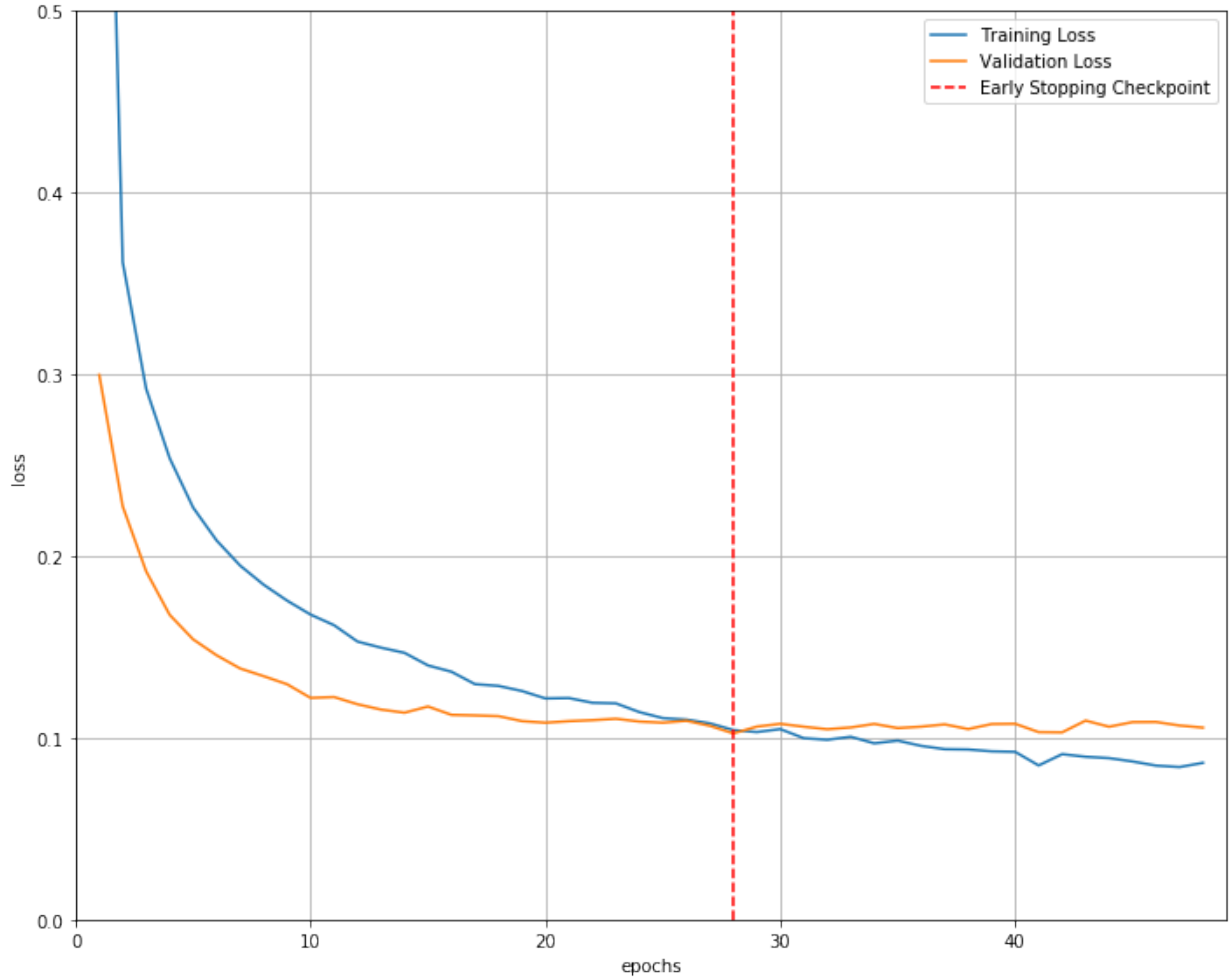
```
        ######################
        # validate the model #
        ######################
        model.eval() # prep model for evaluation
        for data, target in valid_loader:
            # forward pass: compute predicted outputs by passing inputs to the model
            output = model(data)
            # calculate the loss
            loss = criterion(output, target)
            # record validation loss
            valid_losses.append(loss.item())

        # print training/validation statistics
        # calculate average loss over an epoch
        train_loss = np.average(train_losses)
        valid_loss = np.average(valid_losses)
        avg_train_losses.append(train_loss)
        avg_valid_losses.append(valid_loss)

        epoch_len = len(str(n_epochs))

        print_msg = (f'[{epoch:>{epoch_len}}/{n_epochs:>{epoch_len}}] ' +
                     f'train_loss: {train_loss:.5f} ' +
                     f'valid_loss: {valid_loss:.5f}')

        print(print_msg)

        # clear lists to track next epoch
        train_losses = []
        valid_losses = []

        # early_stopping needs the validation loss to check if it has decresed,
        # and if it has, it will make a checkpoint of the current model
        early_stopping(valid_loss, model)

        if early_stopping.early_stop:
            print("Early stopping")
            break

    # load the last checkpoint with the best model
    model.load_state_dict(torch.load('checkpoint.pt'))

    return  model, avg_train_losses, avg_valid_losses
```

In [7]:

```python
batch_size = 256
n_epochs = 100

train_loader, test_loader, valid_loader = create_datasets(batch_size)

# early stopping patience; how long to wait after last time validation loss improved.
patience = 20

model, train_loss, valid_loss = train_model(model, batch_size, patience, n_epochs)
```

```
[  1/100] train_loss: 0.84499 valid_loss: 0.29977
[  2/100] train_loss: 0.36182 valid_loss: 0.22742
Validation loss decreased (inf --> 0.227419).  Saving model ...
[  3/100] train_loss: 0.29205 valid_loss: 0.19163
Validation loss decreased (0.227419 --> 0.191628).  Saving model ...
[  4/100] train_loss: 0.25390 valid_loss: 0.16771
Validation loss decreased (0.191628 --> 0.167714).  Saving model ...
[  5/100] train_loss: 0.22671 valid_loss: 0.15422
Validation loss decreased (0.167714 --> 0.154222).  Saving model ...
[  6/100] train_loss: 0.20862 valid_loss: 0.14546
Validation loss decreased (0.154222 --> 0.145459).  Saving model ...
[  7/100] train_loss: 0.19482 valid_loss: 0.13821
Validation loss decreased (0.145459 --> 0.138206).  Saving model ...
[  8/100] train_loss: 0.18431 valid_loss: 0.13398
Validation loss decreased (0.138206 --> 0.133979).  Saving model ...
[  9/100] train_loss: 0.17554 valid_loss: 0.12953
Validation loss decreased (0.133979 --> 0.129535).  Saving model ...
[ 10/100] train_loss: 0.16785 valid_loss: 0.12202
Validation loss decreased (0.129535 --> 0.122023).  Saving model ...
[ 11/100] train_loss: 0.16202 valid_loss: 0.12249
EarlyStopping counter: 1 out of 20
[ 12/100] train_loss: 0.15300 valid_loss: 0.11852
Validation loss decreased (0.122023 --> 0.118516).  Saving model ...
[ 13/100] train_loss: 0.14965 valid_loss: 0.11560
Validation loss decreased (0.118516 --> 0.115598).  Saving model ...
[ 14/100] train_loss: 0.14680 valid_loss: 0.11387
Validation loss decreased (0.115598 --> 0.113867).  Saving model ...
[ 15/100] train_loss: 0.13988 valid_loss: 0.11728
EarlyStopping counter: 1 out of 20
[ 16/100] train_loss: 0.13641 valid_loss: 0.11269
Validation loss decreased (0.113867 --> 0.112686).  Saving model ...
[ 17/100] train_loss: 0.12957 valid_loss: 0.11237
Validation loss decreased (0.112686 --> 0.112374).  Saving model ...
```

```
EarlyStopping counter: 5 out of 20
[ 34/100] train_loss: 0.09704 valid_loss: 0.10770
EarlyStopping counter: 6 out of 20
[ 35/100] train_loss: 0.09850 valid_loss: 0.10542
EarlyStopping counter: 7 out of 20
[ 36/100] train_loss: 0.09561 valid_loss: 0.10619
EarlyStopping counter: 8 out of 20
[ 37/100] train_loss: 0.09381 valid_loss: 0.10745
EarlyStopping counter: 9 out of 20
[ 38/100] train_loss: 0.09363 valid_loss: 0.10487
EarlyStopping counter: 10 out of 20
[ 39/100] train_loss: 0.09263 valid_loss: 0.10763
EarlyStopping counter: 11 out of 20
[ 40/100] train_loss: 0.09234 valid_loss: 0.10778
EarlyStopping counter: 12 out of 20
[ 41/100] train_loss: 0.08485 valid_loss: 0.10319
EarlyStopping counter: 13 out of 20
[ 42/100] train_loss: 0.09105 valid_loss: 0.10305
EarlyStopping counter: 14 out of 20
[ 43/100] train_loss: 0.08963 valid_loss: 0.10952
EarlyStopping counter: 15 out of 20
[ 44/100] train_loss: 0.08887 valid_loss: 0.10615
EarlyStopping counter: 16 out of 20
[ 45/100] train_loss: 0.08704 valid_loss: 0.10870
EarlyStopping counter: 17 out of 20
[ 46/100] train_loss: 0.08477 valid_loss: 0.10877
EarlyStopping counter: 18 out of 20
[ 47/100] train_loss: 0.08397 valid_loss: 0.10682
EarlyStopping counter: 19 out of 20
[ 48/100] train_loss: 0.08630 valid_loss: 0.10565
EarlyStopping counter: 20 out of 20
Early stopping
```


Training Loss
Validation Loss
Early Stopping Checkpoint
loss
epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40

Thank you.