Lecture 10

Regularization

STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/

Overview: Regularization / Regularizing Effects

- Early stopping
- L_1/L_2 regularization (norm penalties)
- Dropout
- BatchNorm

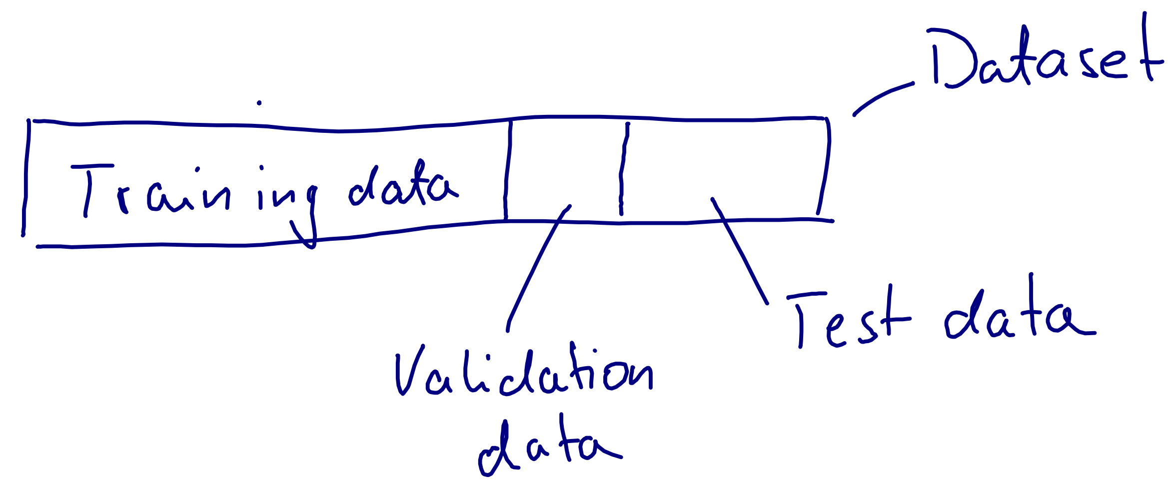
Goal: reduce overfitting

usually achieved by reducing model capacity and/or reduction of the variance of the predictions (as explained last lecture)

Early Stopping

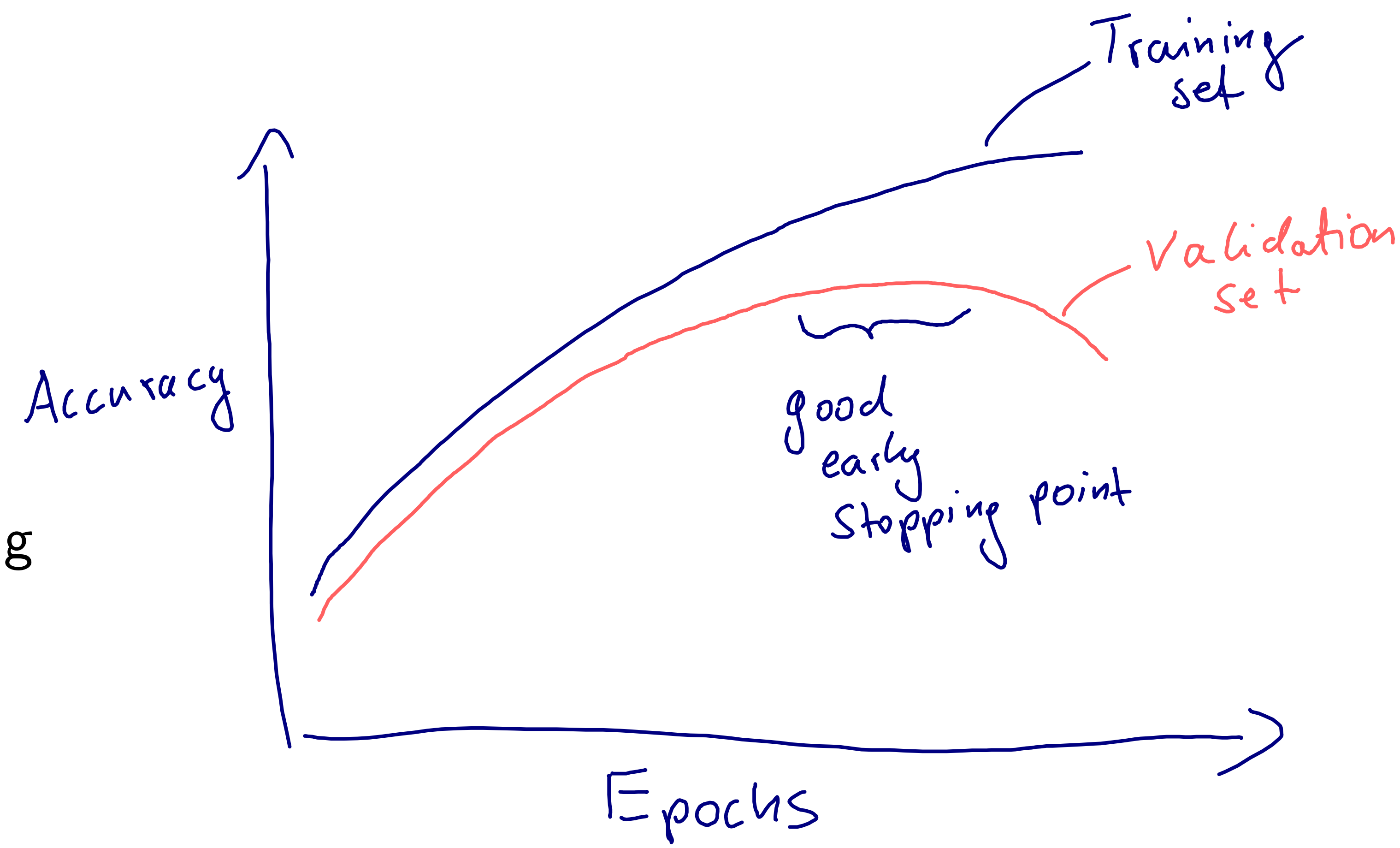
Step 1: Split your dataset into 3 parts (always recommended)

- use test set only once at the end (for unbiased estimate of generalization performance)
- use validation accuracy for tuning (always recommended)



Step 2: Early stopping (not very common anymore)

- reduce overfitting by observing the training/validation accuracy gap during training



L_1/L_2 Regularization

As I am sure you already know it from various statistics classes, we will keep it short:

- L_1-regularization => LASSO regression
- L_2-regularization => Ridge regression (Thikonov regularization)

Basically, a "weight shrinkage" or a "penalty against complexity"

L_1/L_2 Regularization

$$\text{Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]})$$

$$\text{L2-Regularized-Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]}) + \frac{\lambda}{n}\sum_{j} w_j^2$$

where: $\sum_{j} w_j^2 = ||\mathbf{w}||_2^2$

and λ is a hyperparameter

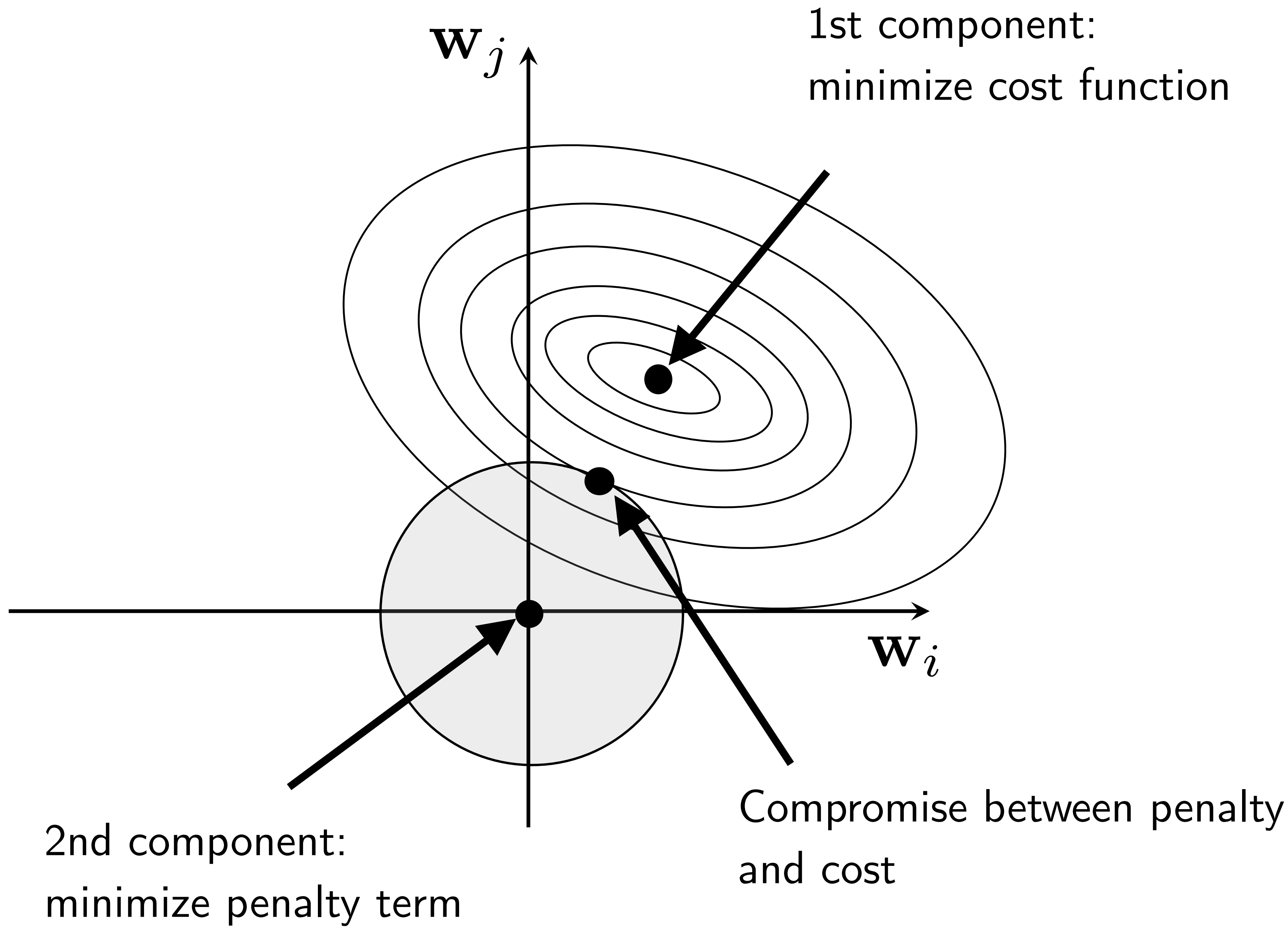
$\mathbf{L_1/L_2}$ Regularization

$$\text{L1-Regularized-Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y^{[i]}, \hat{y}^{[i]}) + \frac{\lambda}{n}\sum_{j} |w_j|$$

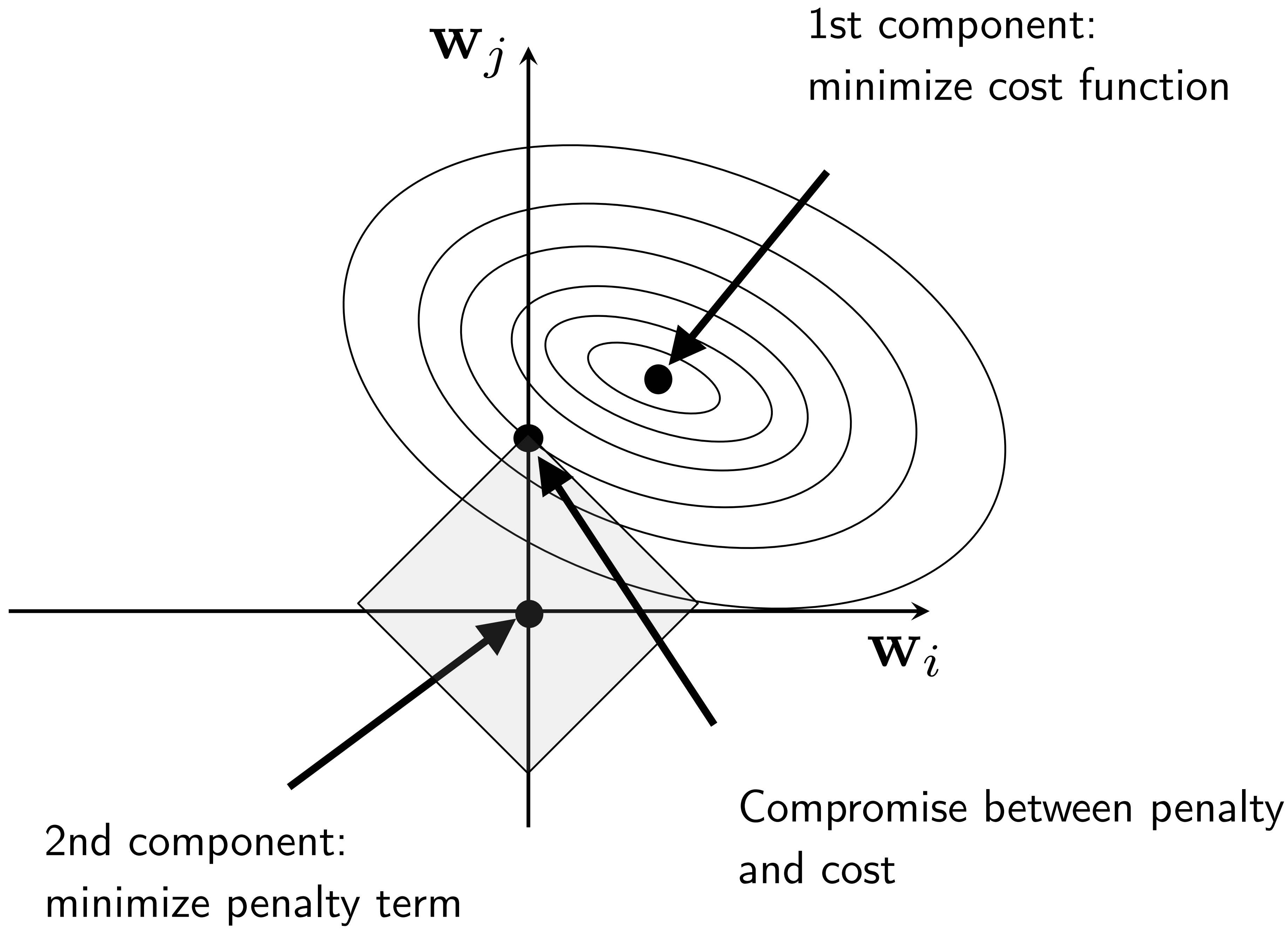
where: $\sum_{j} |w_j| = ||\mathbf{w}||_1$

- L1-regularization encourages sparsity (which may be useful)
- However, usually L1 regularization does not work well in practice and is very rarely used
- Also, it's not smooth and harder to optimize

Geometric Interpretation of L_2 Regularization



Geometric Interpretation of L_2 Regularization



L_2 Regularization for Neural Nets

$$\text{L2-Regularized-Cost}_{\mathbf{w},\mathbf{b}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y^{[i]}, \hat{y}^{[i]}) + \frac{\lambda}{n}\sum_{l=1}^{L}||\mathbf{w}^{(l)}||_F^2$$

sum over layers

where $||\mathbf{w}^{(l)}||_F^2$ is the Frobenius norm:

$$||\mathbf{w}^{(l)}||_F^2 = \sum_i \sum_j (w_{i,j}^{(l)})^2$$

L_2 Regularization for Neural Nets

Regular gradient descent update:

$$w_{i,j} := w_{i,j} - \eta \frac{\partial \mathcal{L}}{\partial w_{i,j}}$$

Gradient descent update with L2 regularization:

$$w_{i,j} := w_{i,j} - \eta \left(\frac{\partial \mathcal{L}}{\partial w_{i,j}} + \frac{2\lambda}{n} w_{i,j} \right)$$

L_2 Regularization for Logistic Regression in PyTorch

Manually:

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

for epoch in range(num_epochs):

    #### Compute outputs ####
    out = model(X_train_tensor)

    #### Compute gradients ####

    ###########################################################
    ## Apply L2 regularization (weight decay)
    cost = F.binary_cross_entropy(out, y_train_tensor, reduction='sum')
    cost = cost + 0.5 * LAMBDA * torch.mm(model.linear.weight,
                                          model.linear.weight.t())

    # note that PyTorch also regularizes the bias, hence, if we want
    # to reproduce the behavior of SGD's "weight_decay" param, we have to add
    # the bias term as well:
    cost = cost + 0.5 * LAMBDA * model.linear.bias**2
    #--------------------------------------------------------

    optimizer.zero_grad()
    cost.backward()
```

(Note that I am using 0.5 here because PyTorch does it; Could be considered "convenient " as the exponent "2" cancels in the derivative. This implementation exactly matches the one on the next slide)

L2-log-reg.ipynb

L_2 Regularization for Logistic Regression in PyTorch

Automatically:

```
#########################################################
## Apply L2 regularization
optimizer = torch.optim.SGD(model.parameters(),
                            lr=0.1,
                            weight_decay=LAMBDA)
#-------------------------------------------------------


for epoch in range(num_epochs):

    #### Compute outputs ####
    out = model(X_train_tensor)

    #### Compute gradients ####
    cost = F.binary_cross_entropy(out, y_train_tensor, reduction='sum')
    optimizer.zero_grad()
    cost.backward()
```

L2-log-reg.ipynb

L_2 Regularization for Neural Nets in PyTorch

- For all layers, same as before ("automatic approach" via **`weight_decay`**)
- Or, manually:

```
for epoch in range(NUM_EPOCHS):
    model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.view(-1, 28*28).to(DEVICE)
        targets = targets.to(DEVICE)

        ### FORWARD AND BACK PROP
        logits, probas = model(features)

        cost = F.cross_entropy(logits, targets)

        # regularize loss
        L2 = 0.
        for p in model.parameters():
            L2 = L2 + (p**2).sum()
        cost = cost + 2./targets.size(0) * LAMBDA * L2

        optimizer.zero_grad()
        cost.backward()
```

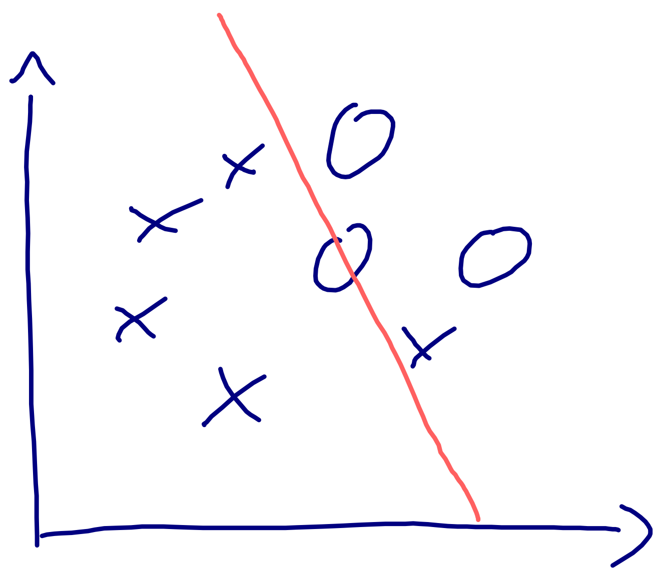
L_2 Regularization for Neural Nets in PyTorch

- Or, if you only want to regularize the weights, not the biases:

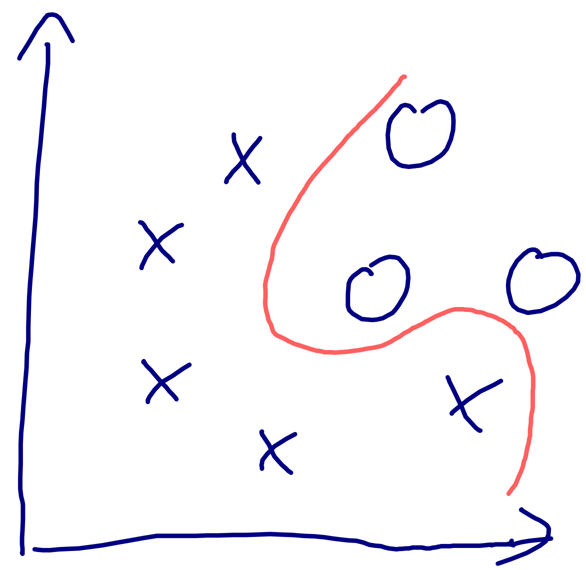
```
# regularize loss
L2 = 0.
for name, p in model.named_parameters():
    if 'weight' in name:
        L2 = L2 + (p**2).sum()

cost = cost + 2./targets.size(0) * LAMBDA * L2

optimizer.zero_grad()
cost.backward()
```

Effect of Norm Penalties on the Decision Boundary

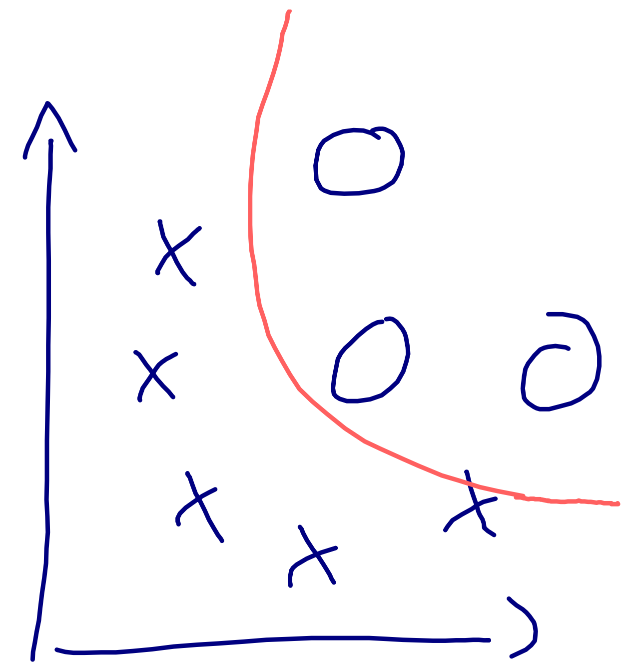
Assume a nonlinear model



Large regularization penalty
=> high bias



Low regularization
=> high variance



Good compromise

Dropout and BatchNorm continued on Friday!