Signal Processing Systems

Machine learning for signal processing *[5LSL0]*

Ruud van Sloun, Rik Vullings

May 11, 2019

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Generalization

Central challenge in machine learning: algorithms must perform well on *unseen* inputs

Generalization

Central challenge in machine learning: algorithms must perform well on *unseen* inputs

General approach:

- Use two datasets: training and test set, generated by the same process
- Examples in datasets are assumed independent and identically distributed
- Train learning algorithm on training set to minimize training error
- Evaluate learning algorithm on test set

Machine learning goals

Training error

$$\frac{1}{m^{(train)}} \left\| \mathbf{X}^{(train)} \mathbf{w} - \mathbf{y}^{(train)} \right\|_2^2$$

Machine learning goals

Training error

$$\frac{1}{m^{(train)}} \left\| \mathbf{X}^{(train)}\mathbf{w} - \mathbf{y}^{(train)} \right\|_2^2$$

Test error

$$\frac{1}{m^{(test)}} \left\| \mathbf{X}^{(test)}\mathbf{w} - \mathbf{y}^{(test)} \right\|_2^2$$

Machine learning goals

Training error

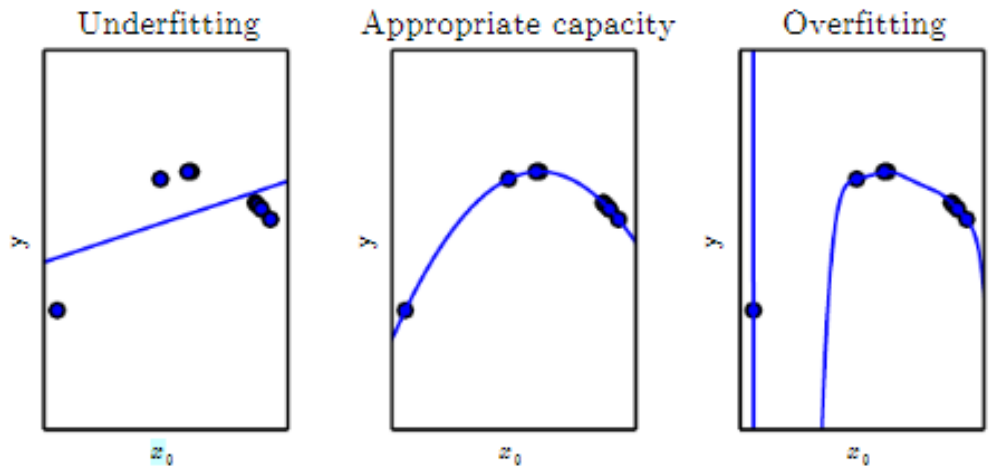
$$\frac{1}{m^{(train)}} \left\| \mathbf{X}^{(train)}\mathbf{w} - \mathbf{y}^{(train)} \right\|_2^2$$

Test error

$$\frac{1}{m^{(test)}} \left\| \mathbf{X}^{(test)}\mathbf{w} - \mathbf{y}^{(test)} \right\|_2^2$$

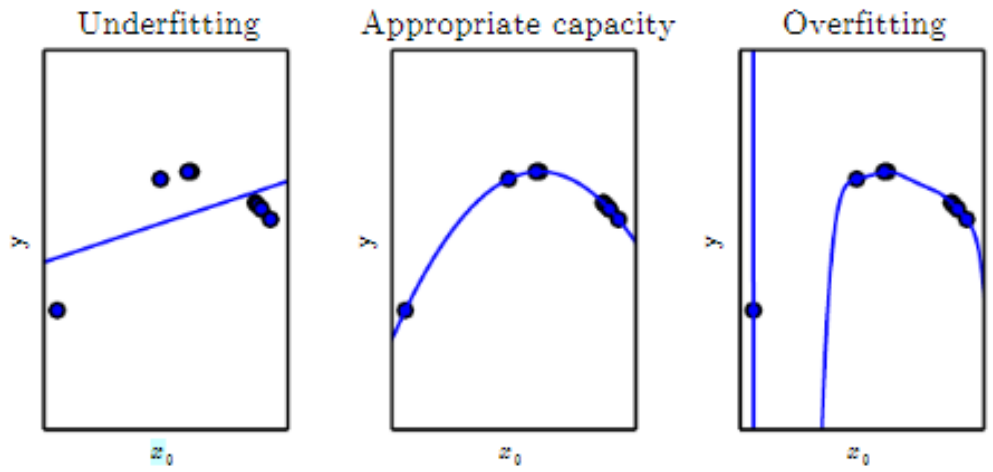
Optimization: make training error as small as possible

Underfitting and overfitting



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Underfitting and overfitting

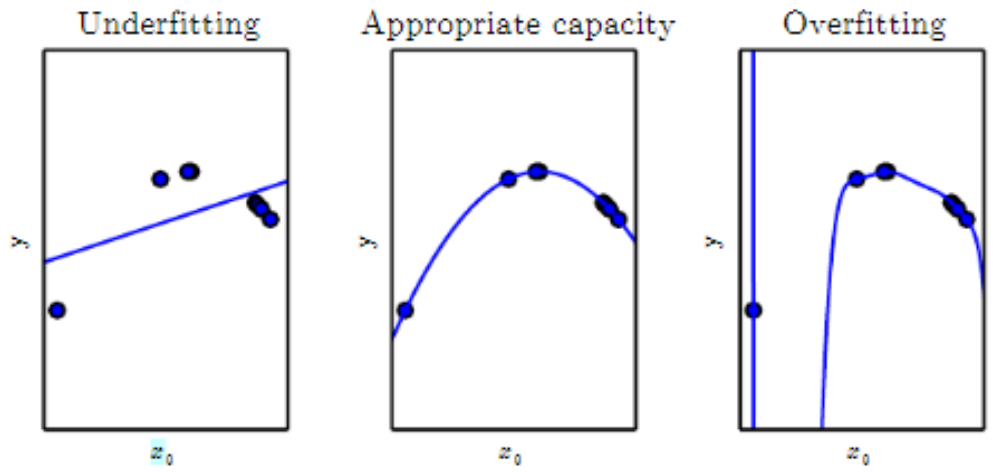


Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Underfitting: model is not able to achieve low training error

Overfitting: gap between training error and test error is too large

Underfitting and overfitting



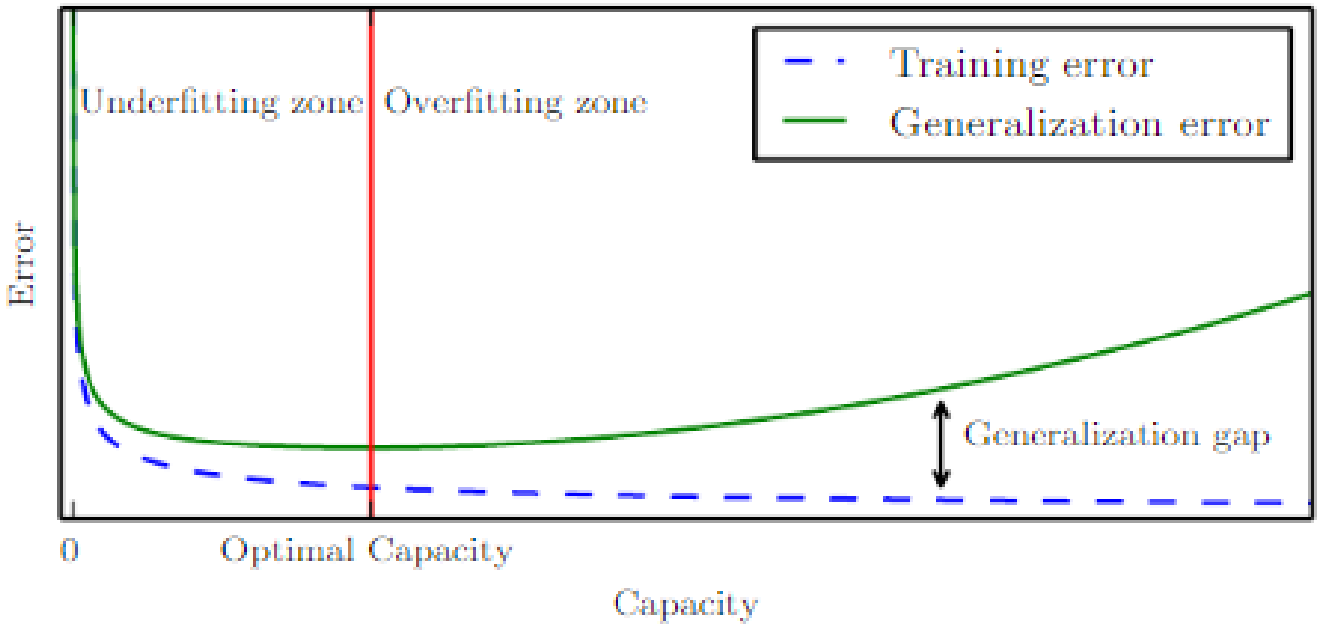
Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Underfitting: model is not able to achieve low training error

Overfitting: gap between training error and test error is too large

Regularization: make gap between training and test error as small as possible

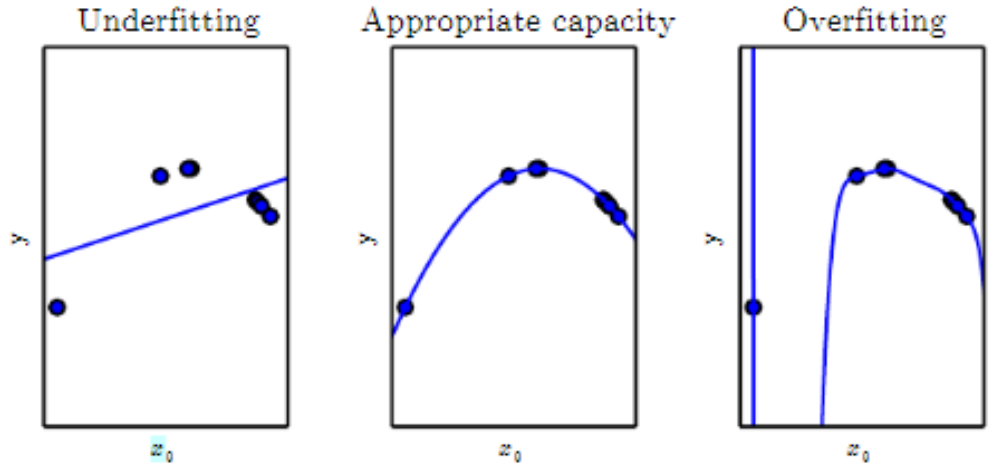
Underfitting and overfitting



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

For capacities higher than the optimal one, the size of the generalization gap outweighs the decrease in training error.

Regularization

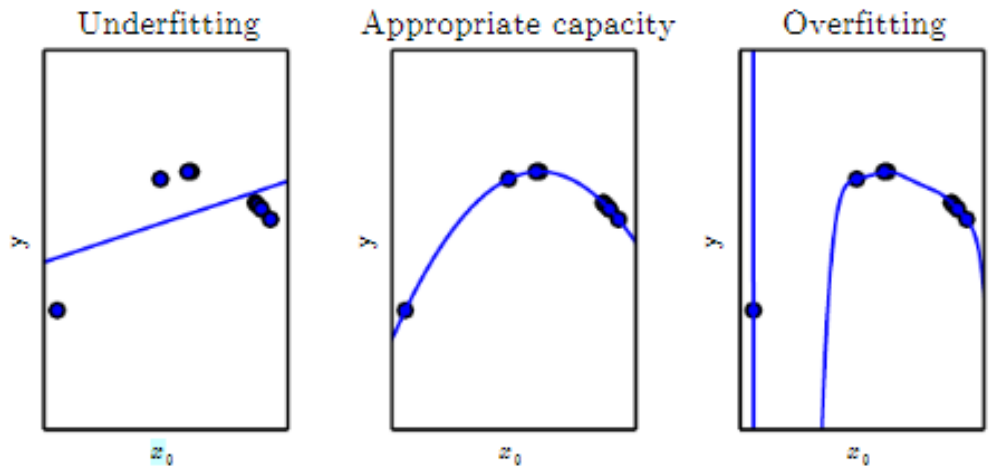


Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Optimize a modified cost function:

$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^T \mathbf{w}$$

Regularization



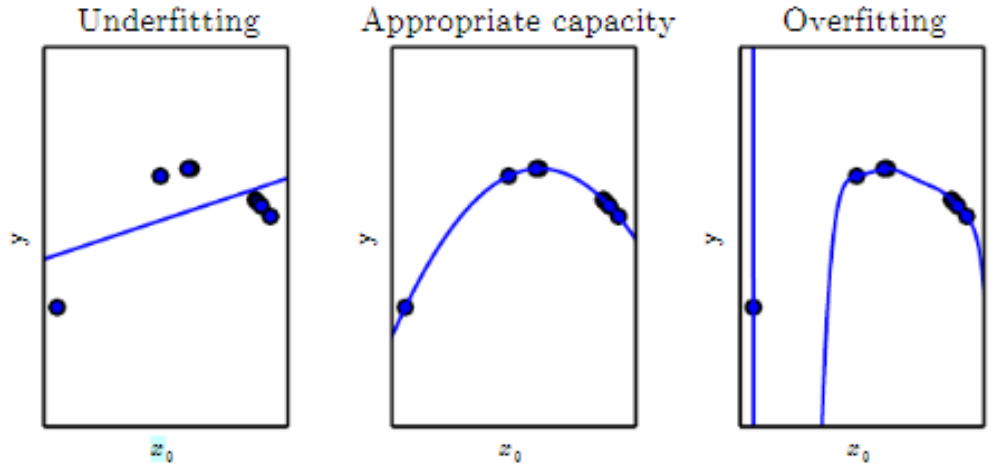
Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Optimize a modified cost function:

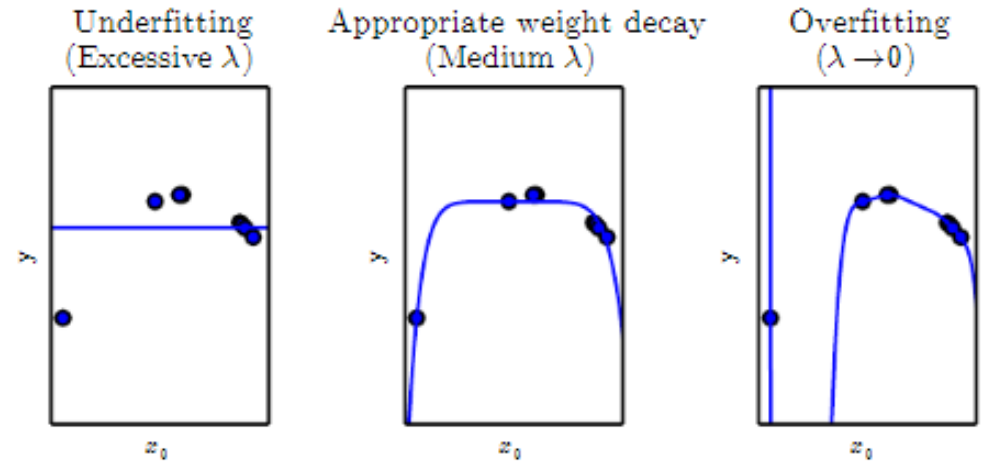
$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^T \mathbf{w}$$

$$\text{Regularizer} \rightarrow \Omega(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$$

Regularization



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Regularization

- Parameter Norm Penalties
- Label smoothing
- Early stopping
- Dropout
- Parameter Tying and Sharing

Parameter Norm Penalties

Regularization by limiting the capacity of models by adding a parameter norm penalty:

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

Parameter Norm Penalties

Regularization by limiting the capacity of models by adding a parameter norm penalty:

$$\boxed{\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \Omega(\theta)}$$

L^2 regularization: $\Omega(\theta) = \frac{1}{2} \|\theta\|_2^2$

L^1 regularization: $\Omega(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$

L^2 Regularization: per update step

Cost function:

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \frac{\lambda}{2}\theta^T\theta$$

L^2 Regularization: per update step

Cost function:

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \frac{\lambda}{2}\theta^T\theta$$

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta$$

Gradient update step:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha\left(\nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta\right) \\ &= (1 - \alpha\lambda)\,\theta - \alpha\nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) \end{aligned}$$

L^2 Regularization: per update step

Cost function:

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \frac{\lambda}{2}\theta^T\theta$$

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta$$

Gradient update step:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha\left(\nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta\right) \\ &= (1 - \alpha\lambda)\,\theta - \alpha\nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) \end{aligned}$$

On every update step, the weight is shrunk by a factor $1 - \alpha\lambda$.

L^2 Regularization: entire training

Quadratic approximation of unregularized $J(\theta; \mathbf{X}, \mathbf{y})$ around $\theta = \theta^*$ gives:

$$\hat{J}(\theta) = J(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T \mathbf{H} (\theta - \theta^*)$$

Note the similarity to the quadratic cost function for the MMSE/Wiener; the Hessian $\mathbf{H}$ replacing the autocorrelation $\mathbf{R}$.

L^2 Regularization: entire training

Quadratic approximation of unregularized $J(\theta; \mathbf{X}, \mathbf{y})$ around $\theta = \theta^*$ gives:

$$\hat{J}(\theta) = J(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T \mathbf{H} (\theta - \theta^*)$$

Note the similarity to the quadratic cost function for the MMSE/Wiener; the Hessian $\mathbf{H}$ replacing the autocorrelation $\mathbf{R}$.

For regularized cost function

$$\begin{aligned} \nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) &= \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta \\ &= \mathbf{H}(\theta - \theta^*) + \lambda\theta \end{aligned}$$

L^2 Regularization: entire training

Quadratic approximation of unregularized $J(\theta; \mathbf{X}, \mathbf{y})$ around $\theta = \theta^*$ gives:

$$\hat{J}(\theta) = J(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T \mathbf{H}(\theta - \theta^*)$$

Note the similarity to the quadratic cost function for the MMSE/Wiener; the Hessian $\mathbf{H}$ replacing the autocorrelation $\mathbf{R}$.

For regularized cost function

$$\begin{aligned} \nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) &= \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta \\ &= \mathbf{H}(\theta - \theta^*) + \lambda\theta \end{aligned}$$

Equating to zero yields

$$\tilde{\theta} = (\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}\theta^*$$

L^2 Regularization: entire training

$$\tilde{\theta} = (\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H} \theta^*$$

$\lambda = 0$ yields $\tilde{\theta} = \theta^*$. In other words: the minimum for the regularized cost function (with zero regularization) is identical to that of the unregularized cost function.

L^2 Regularization: entire training

$$\tilde{\theta} = (\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H} \theta^*$$

$\lambda = 0$ yields $\tilde{\theta} = \theta^*$. In other words: the minimum for the regularized cost function (with zero regularization) is identical to that of the unregularized cost function.

For $\lambda \neq 0$, use eigen decomposition ($\mathbf{H} = \mathbf{Q} \Lambda \mathbf{Q}^T$):

$$\tilde{\theta} = \mathbf{Q} (\Lambda + \lambda \mathbf{I})^{-1} \Lambda \mathbf{Q}^T \theta^*$$

L^2 Regularization: entire training

$$\tilde{\theta} = (\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H} \theta^*$$

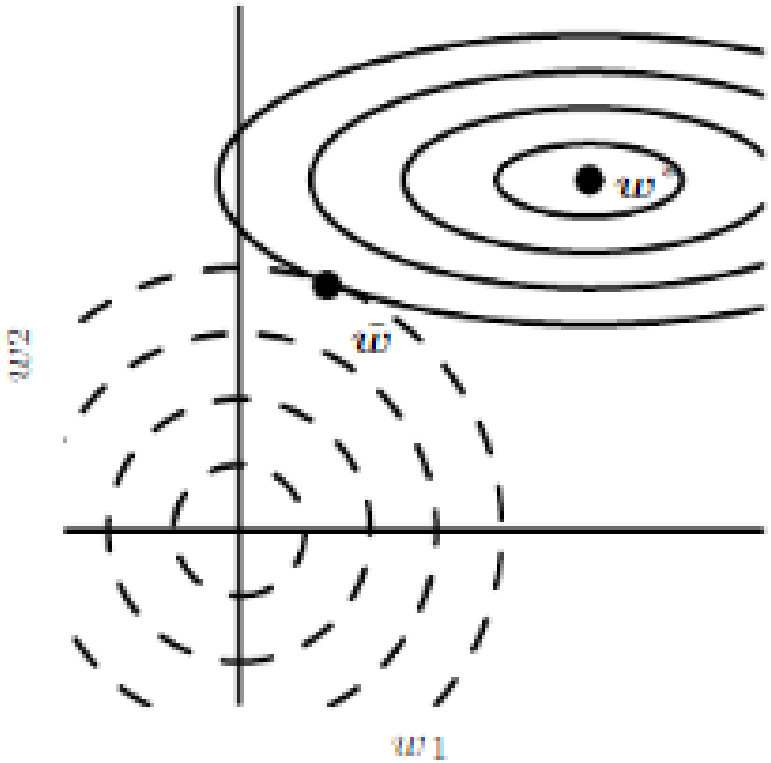
$\lambda = 0$ yields $\tilde{\theta} = \theta^*$. In other words: the minimum for the regularized cost function (with zero regularization) is identical to that of the unregularized cost function.

For $\lambda \neq 0$, use eigen decomposition ($\mathbf{H} = \mathbf{Q} \Lambda \mathbf{Q}^T$):

$$\tilde{\theta} = \mathbf{Q} (\Lambda + \lambda \mathbf{I})^{-1} \Lambda \mathbf{Q}^T \theta^*$$

The L^2 regularization scales the weights θ^* of the unregularized cost function along the axes defined by the eigenvectors of $\mathbf{H}$. For large eigenvalues, the effect of regularization is small. For small eigenvalues, the effect can be large.

L^2 Regularization: entire training



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Note that θ_1 (w_1 in plot) is relatively much different between $\tilde{\theta}$ and θ^*, whereas the difference in θ_2 is much smaller.

L^2 vs linear regression

Remember from week 2: cost function for linear regression:

$$J(\theta) = (\theta^T \mathbf{X} - \mathbf{y})(\theta^T \mathbf{X} - \mathbf{y})^T$$

where equating the derivative wrt θ to zero yielded

$$\theta = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}^T$$

L^2 vs linear regression

Remember from week 2: cost function for linear regression:

$$J(\theta) = (\theta^T \mathbf{X} - \mathbf{y})(\theta^T \mathbf{X} - \mathbf{y})^T$$

where equating the derivative wrt θ to zero yielded

$$\theta = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}^T$$

Adding L^2 regularization changes this to

$$\theta = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1}\mathbf{X}\mathbf{y}^T$$

Learning algorithm perceives higher variance of $\mathbf{X}$, causing it to shrink weights of features that have low covariance compared to the added variance ($\sim \lambda$)

L^1 Regularization

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \|\theta\|_1$$

with gradient

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \frac{\theta}{|\theta|}$$

L^1 Regularization

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \|\theta\|_1$$

with gradient

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda \frac{\theta}{|\theta|}$$

Remember L^2:

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\theta$$

Contrary to L^2 regularization, for L^1 the regularization does NOT scale linearly with θ.

L^1 Regularization

Under a quadratic approximation around θ^*, for each dimension i, the solution to the minimization of the cost function is

$$\theta_i = \frac{\theta_i^*}{|\theta_i^*|} max \left\{ |\theta_i^*| - \frac{\lambda}{H_{i,i}}, 0 \right\}$$

L^1 Regularization

Under a quadratic approximation around θ^*, for each dimension i, the solution to the minimization of the cost function is

$$\theta_i = \frac{\theta_i^*}{|\theta_i^*|} max \left\{ |\theta_i^*| - \frac{\lambda}{H_{i,i}}, 0 \right\}$$

L^1 regularization can push values of θ to zero, making it more sparse than L^2.

L^1 vs L^2

L^1:

- sparsity that can act as feature selection
- more robust to outliers

L^2:

- smoother weights θ (for L^1 a small difference in input can yield large difference in weights)
- analytical solution (L^1 only for the quadratic approximation case), making it computationally more efficient

Label smoothing

Goal of the parameter norm penalties was to avoid θ becoming too large.

Label smoothing

Goal of the parameter norm penalties was to avoid θ becoming too large.

Recap from classification:

$$f(x; w, b) = w^T x + b$$

Consider a situation with k classes (e.g. $k = 10$ for digit recognition). The function $f(x; w, b)$ will produce a length k output vector. How to assign probabilities to each of the classes?

Label smoothing

Goal of the parameter norm penalties was to avoid θ becoming too large.

Recap from classification:

$$f(x; w, b) = w^T x + b$$

Consider a situation with k classes (e.g. $k = 10$ for digit recognition). The function $f(x; w, b)$ will produce a length k output vector. How to assign probabilities to each of the classes?

$$S(f(x; w, b))_i = \frac{\exp(f(x; w, b)_i)}{\sum_j \exp(f(x; w, b)_j)}$$

Softmax

Softmax example

$$\text{Scores } f(x; w, b) = \begin{cases} 0.1 \\ 0.2 \\ 3.0 \end{cases}$$

The class probabilities are then

$$p = \begin{cases} \frac{\exp(0.1)}{\exp(0.1)+\exp(0.2)+\exp(3.0)} = 0.049 \\ 0.055 \\ 0.892 \end{cases}$$

Softmax example

$$\text{Scores } f(x; w, b) = \begin{cases} 0.1 \\ 0.2 \\ 3.0 \end{cases}$$

The class probabilities are then

$$p = \begin{cases} \frac{\exp(0.1)}{\exp(0.1)+\exp(0.2)+\exp(3.0)} = 0.049 \\ 0.055 \\ 0.892 \end{cases}$$

Q: What would happen with the probabilities if the scores are 10 times as high? Do they get closer to 0 and 1 or become more uniform?

Label smoothing

Goal of the parameter norm penalties was to avoid θ becoming too large.

For classification, probabilities for labels are hard (i.e. 0 or 1). Maximum likelihood learning with softmax cannot converge and weights θ become progressively large.

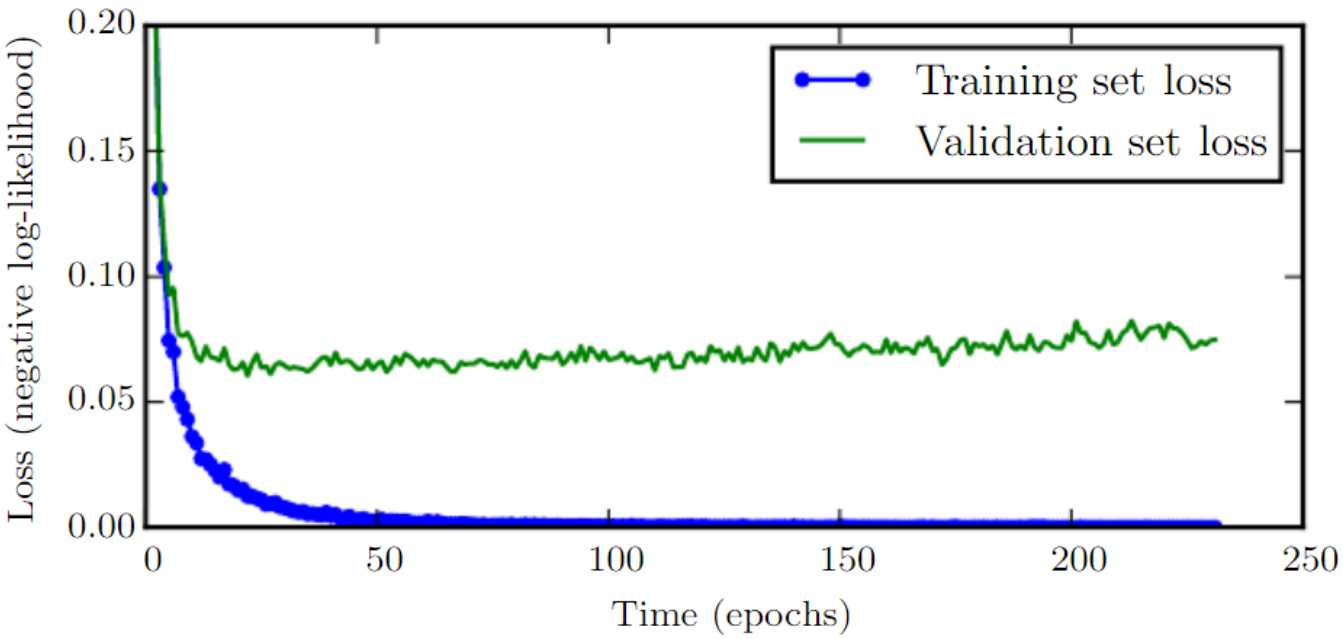
Label smoothing

Goal of the parameter norm penalties was to avoid θ becoming too large.

For classification, probabilities for labels are hard (i.e. 0 or 1). Maximum likelihood learning with softmax cannot converge and weights θ become progressively large.

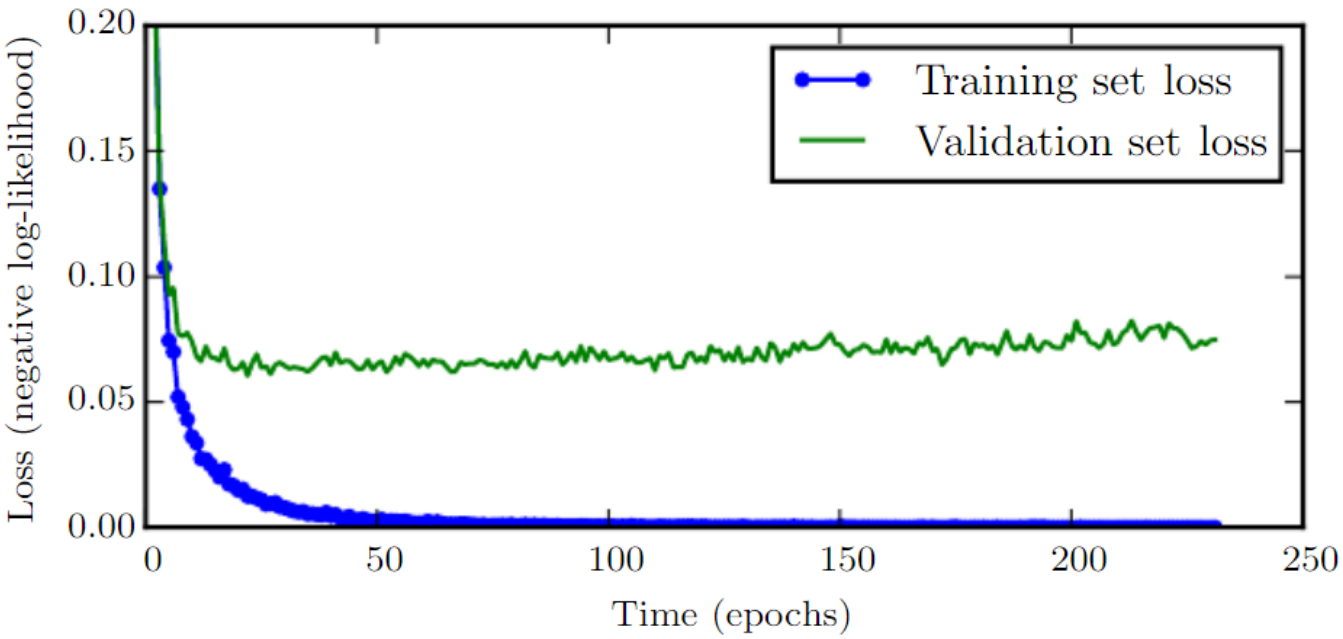
In label smoothing, instead of penalizing for large weights, we assume "noise" on the labels: e.g. the probability that label y is correct is not 1 but $1 - \epsilon$.
Subsequently, we train softmax to $\frac{\epsilon}{k-1}$ and $1 - \epsilon$ targets.

Early stopping



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

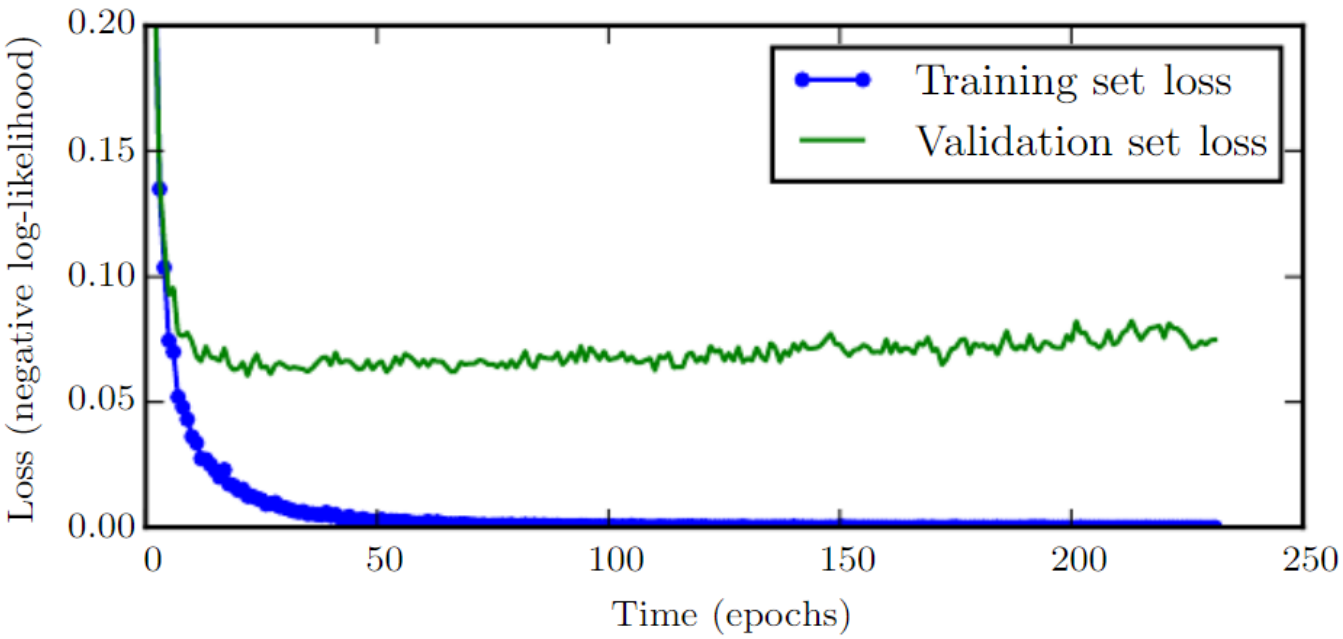
Early stopping



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Save parameters that yield the best validation loss and stop training when parameters have not improved loss for specified number of iterations.

Early stopping



Picture from Goodfellow et al. "Deep learning" (www.deeplearningbook.org)

Save parameters that yield the best validation loss and stop training when parameters have not improved loss for specified number of iterations.

Why does early stopping act as regularizer?

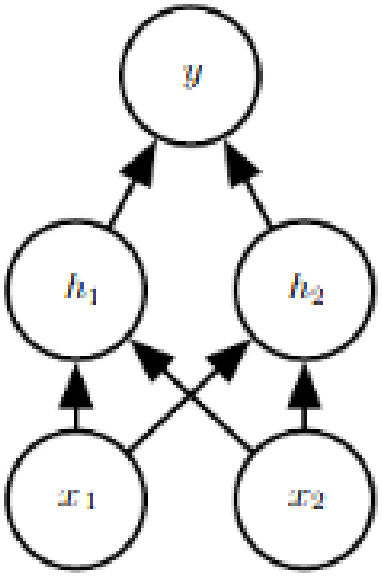
Dropout

Ensemble methodes: train several independent models and have them vote on the output. Requires *replacement sampling* and is highly effective for generalization but highly computationally expensive.

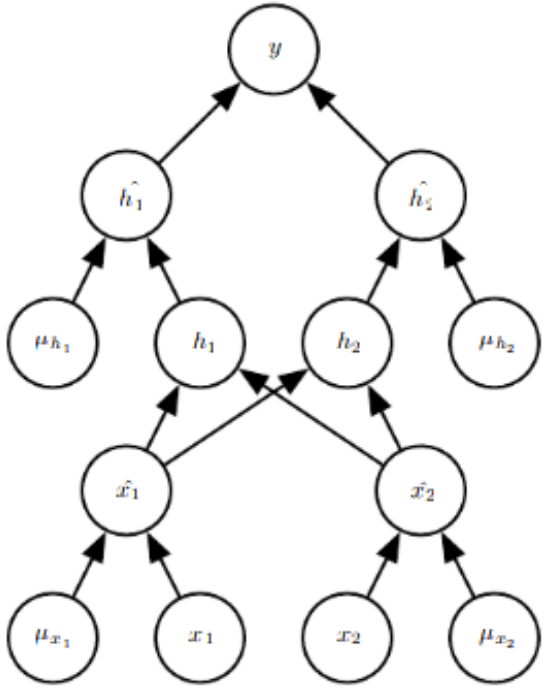
Dropout

- Practical alternative for ensemble methods
- Very powerful regularization method
- As simple as multiplying the output of variable input or hidden units in the network with binary μ that is sampled independently per unit with probability $p(\mu)$

Dropout



Picture from Goodfellow et al. "Deep learning"
(www.deeplearningbook.org)



Picture from Goodfellow et al. "Deep learning"
(www.deeplearningbook.org)

Dropout

- In each training step of the model, masks μ are newly sampled
- Model parameters are inherited from parental network (i.e. *shared* among subsets)
- Inference on output *y* by combining the weights of the ensemble of subset models
- Dropout reduces the capacity of model so necessitates more complex models

Note: dropout does not destruct raw input data, but rather intelligently destructs information.
Example: face detection in images

Parameter tying and sharing

In dropout, model parameters were inherited from parental network. In other words, they were shared between the subsets. Different models that should do the same task are likely to have similar parameters. We can enforce this via:

- Parameter tying: parameter norm penalty (e.g. L^2) of the form $\Omega(\theta^A, \theta^B)$. Parameters between model A and B are forced to be similar
- Parameter sharing: extreme case of parameter tying (parameters forced towards being identical)

Parameter tying and sharing

In dropout, model parameters were inherited from parental network. In other words, they were shared between the subsets. Different models that should do the same task are likely to have similar parameters. We can enforce this via:

- Parameter tying: parameter norm penalty (e.g. L^2) of the form $\Omega(\theta^A, \theta^B)$. Parameters between model A and B are forced to be similar
- Parameter sharing: extreme case of parameter tying (parameters forced towards being identical)

Parameter sharing:

- limits memory usage
- one of the key aspects of Convolutional Neural Networks (think of e.g. translation in picture)