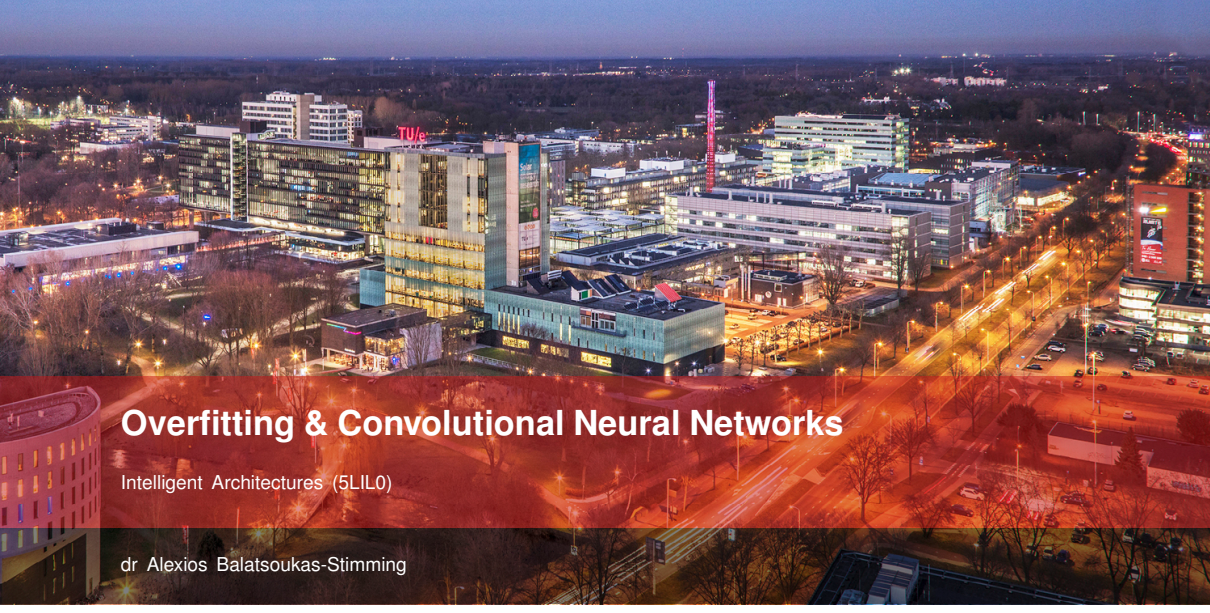


Overfitting & Convolutional Neural Networks

Intelligent Architectures (5LIL0)

dr Alexios Balatsoukas-Stimming

Department of Electrical Engineering, Electronic Systems Group

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Last Time

- **Loss functions**, e.g., squared error:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

Last Time

- **Loss functions**, e.g., squared error:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

- Training using **gradient descent**:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$$

Last Time

- **Loss functions**, e.g., squared error:

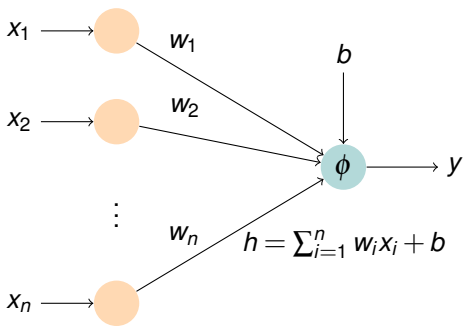
$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

- Training using **gradient descent**:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$$

- **Backpropagation** to calculate gradients:

$$\frac{\partial L}{\partial w_j} = 2(y - \hat{y}) \cdot \phi'(h) \cdot x_j$$



Last Time

- **Loss functions**, e.g., squared error:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = |\mathbf{y} - \hat{\mathbf{y}}|_2^2 = \sum_{i=1}^{M} (y_i - \hat{y}_i)^2$$

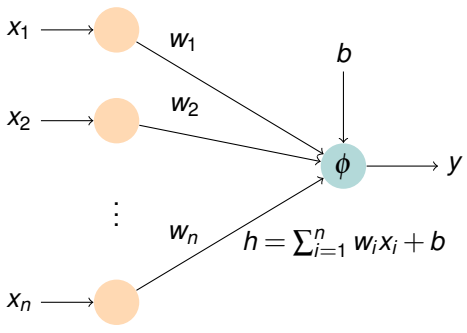
- Training using **gradient descent**:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \eta \cdot \frac{\partial L}{\partial \mathbf{w}_{k-1}}$$

- **Backpropagation** to calculate gradients:

$$\frac{\partial L}{\partial w_j} = 2(y - \hat{y}) \cdot \phi'(h) \cdot x_j$$

- **Weight initialization**: Xavier and He

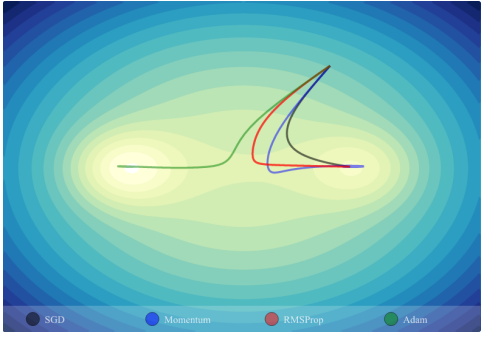


Last Time

- Improved gradient descent algorithms
  1. SGD with Momentum
  2. RMSProp
  3. Adam

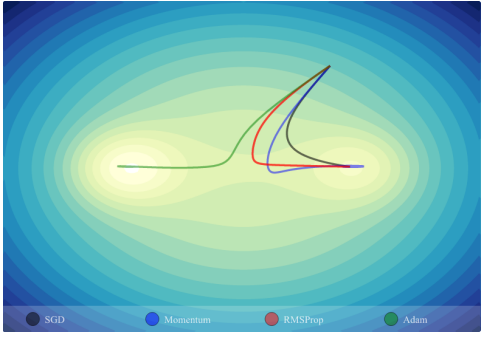
Last Time

- Improved gradient descent algorithms
  1. SGD with Momentum
  2. RMSProp
  3. Adam
- Data normalization



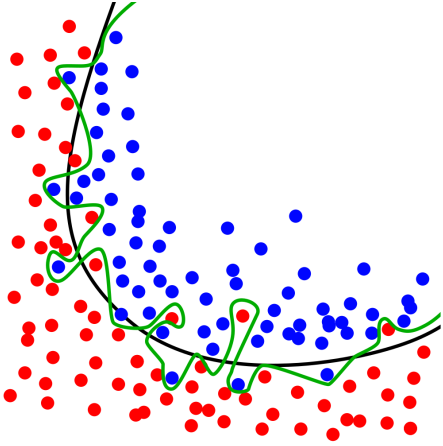
Last Time

- Improved gradient descent algorithms
  1. SGD with Momentum
  2. RMSProp
  3. Adam
- Data normalization
- Deep learning frameworks



Today

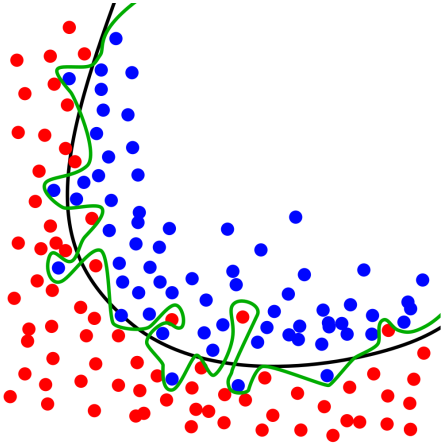
- Overfitting and how to deal with it



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Today

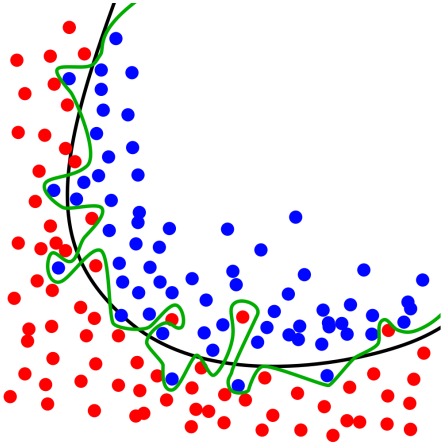
- Overfitting and how to deal with it
- Hyperparameter tuning



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Today

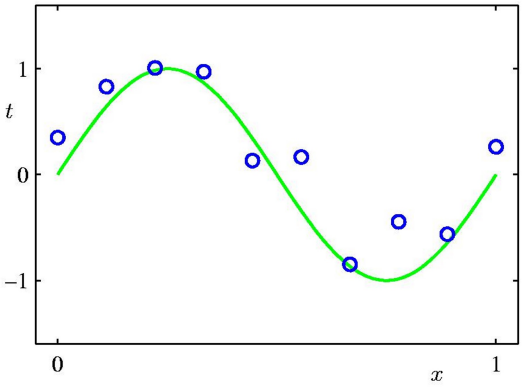
- Overfitting and how to deal with it
- Hyperparameter tuning
- Convolutional neural networks



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Learning a Curve

- Dataset from [1]: $\left\{x^{(i)}, t^{(i)}\right\}_{i=1}^{10}$
- Generated by: $t = sin(x) + \text{noise}$

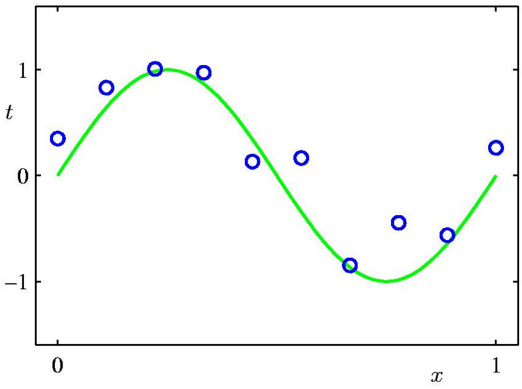


[1] C. M. Bishop, "Pattern Recognition and Machine Learning," 2006.

Learning a Curve

- Dataset from [1]: $\left\{x^{(i)}, t^{(i)}\right\}_{i=1}^{10}$
- Generated by: $t = sin(x) + \text{noise}$

- Can we learn this curve from the measurements?



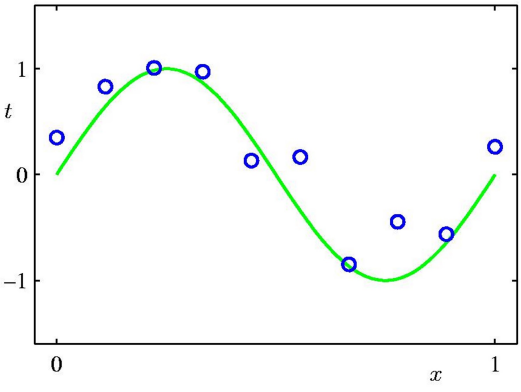
[1] C. M. Bishop, "Pattern Recognition and Machine Learning," 2006.

Learning a Curve

- Dataset from [1]: $\left\{x^{(i)}, t^{(i)}\right\}_{i=1}^{10}$
- Generated by: $t = sin(x) + \text{noise}$

- Can we learn this curve from the measurements?
- We use a **polynomial model**:

$$\hat{t}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$



[1] C. M. Bishop, "Pattern Recognition and Machine Learning," 2006.

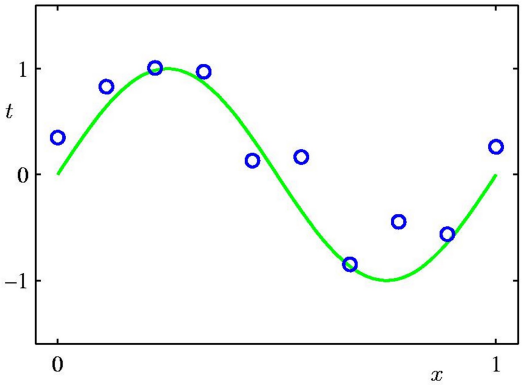
Learning a Curve

- Dataset from [1]: $\left\{x^{(i)}, t^{(i)}\right\}_{i=1}^{10}$
- Generated by: $t = sin(x) + \text{noise}$

- Can we learn this curve from the measurements?
- We use a **polynomial model**:

$$\hat{t}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

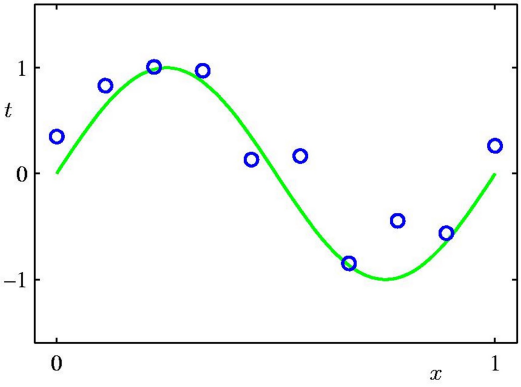
- Loss function $L = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)}\right)^2$



[1] C. M. Bishop, "Pattern Recognition and Machine Learning," 2006.

Learning a Curve

- Dataset from [1]: $\left\{x^{(i)}, t^{(i)}\right\}_{i=1}^{10}$
- Generated by: $t = sin(x) + \text{noise}$



- Can we learn this curve from the measurements?
- We use a **polynomial model**:

$$\hat{t}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

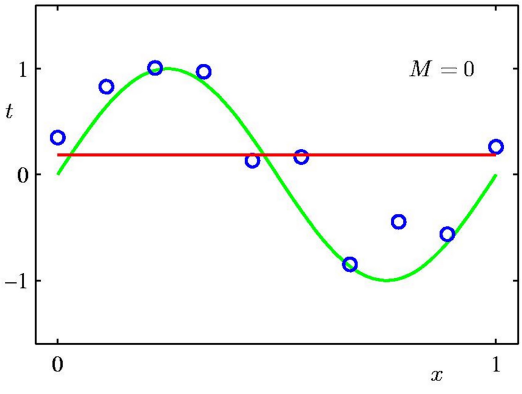
- Loss function $L = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)}\right)^2$

- Can be trained with gradient descent, but closed-form solution exists (**least squares**).

[1] C. M. Bishop, "Pattern Recognition and Machine Learning," 2006.

Learning a Curve ($M = 0$)

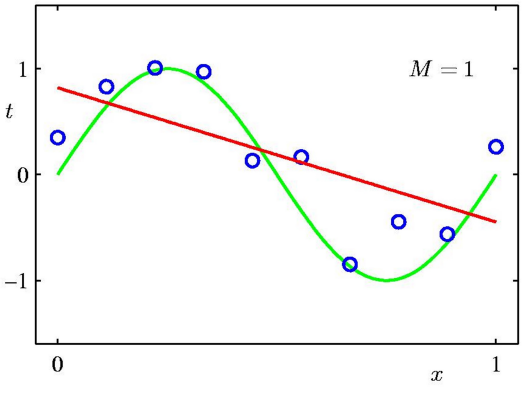
Parameter	Value
w_0	0.19



- **Model:** $\hat{t}(x, \mathbf{w}) = 0.19$

Learning a Curve ($M = 1$)

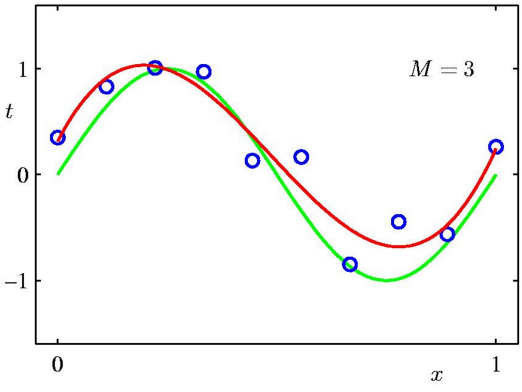
Parameter	Value
w_0	0.82
w_1	-1.27



- **Model:** $\hat{t}(x, \mathbf{w}) = 0.82 - 1.27x$

Learning a Curve ($M = 3$)

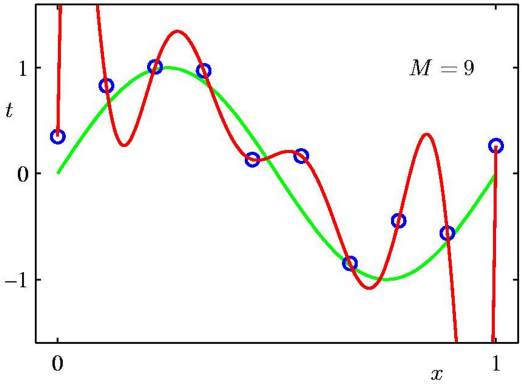
Parameter	Value
w_0	0.31
w_1	−7.99
w_2	−25.43
w_3	17.37



- **Model:** $\hat{t}(x, \mathbf{w}) = 0.31 - 7.99x - 25.43x^2 + 17.37x^3$

Learning a Curve ($M = 9$)

Parameter	Value
w_0	0.35
w_1	232
w_2	−5321
w_3	45688
w_4	−231630
w_5	640042
w_6	−1061800
w_7	1042400
w_8	−557682
w_9	125201

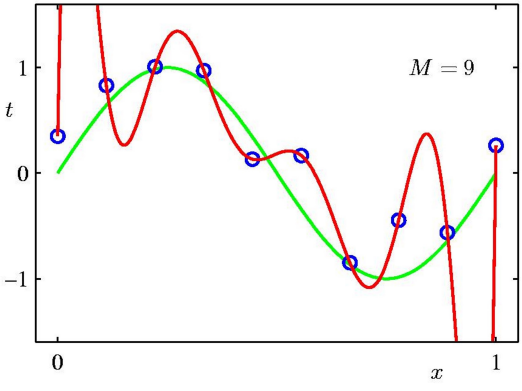


- **Model:** $\hat{t}(x, \mathbf{w}) = 0.35 + 232x - 5321x^2 + 45688x^3 - 231630x^4 + 640042x^5 - \ldots$

Overfitting

The overfitting problem:

1. The model fits dataset perfectly.
2. **BUT** its predictions are horrible almost everywhere else!



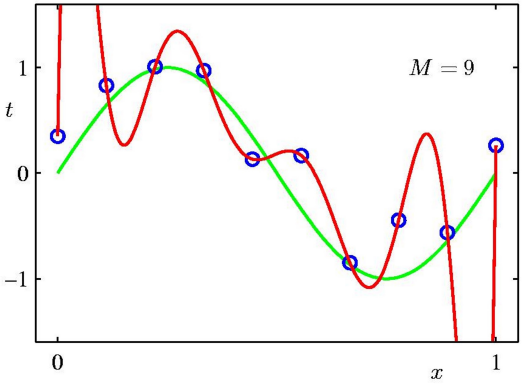
Overfitting

The overfitting problem:

1. The model fits dataset perfectly.
2. **BUT** its predictions are horrible almost everywhere else!

The root cause:

- The model has **too much freedom**.



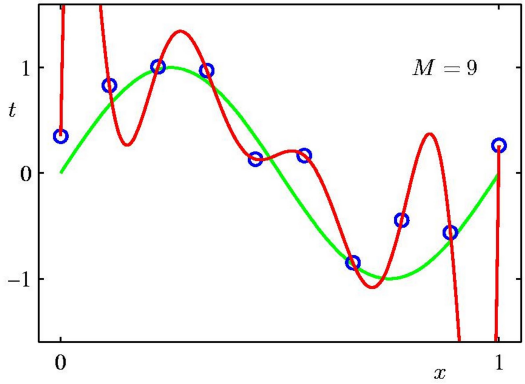
Overfitting

The overfitting problem:

1. The model fits dataset perfectly.
2. **BUT** its predictions are horrible almost everywhere else!

The root cause:

- The model has **too much freedom**.



A word of caution

Identifying and dealing with overfitting is potentially more art than science!

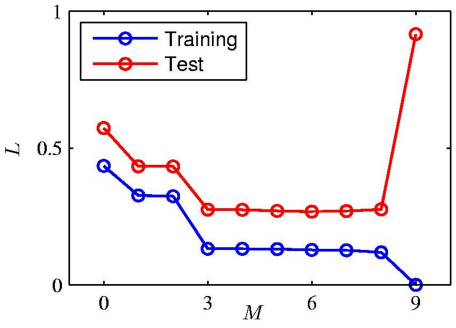
Identifying Overfitting

- Split dataset:
  1. **Training set**: only used for training model
  2. **Test set**: only used for testing performance

Identifying Overfitting

- Split dataset:
  1. **Training set**: only used for training model
  2. **Test set**: only used for testing performance

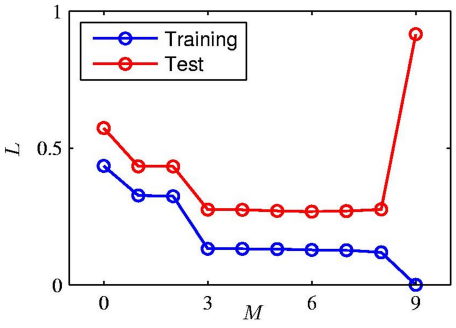
- If the test set performance is “a lot” worse than the training set performance, we have overfitting.



Identifying Overfitting

- Split dataset:
  1. **Training set**: only used for training model
  2. **Test set**: only used for testing performance

- If the test set performance is “a lot” worse than the training set performance, we have overfitting.



How to deal with overfitting?

The general idea is to **constrain** the model.

Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

Pros:

Cons:

Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

Pros: simple.

Cons:

Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

Pros: simple.

Cons: model might be too weak.

Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

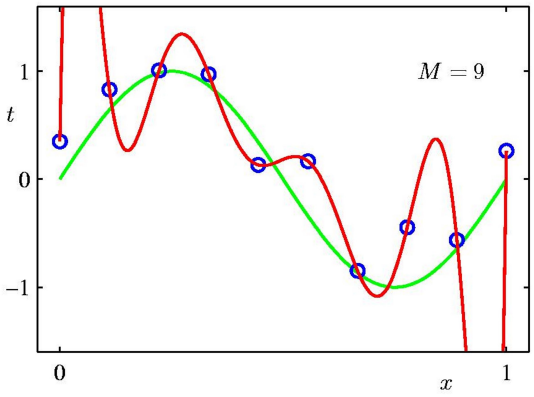
Pros: simple.

Cons: model might be too weak.

Solution 2: Use a larger training set

Pros:

Cons:



Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

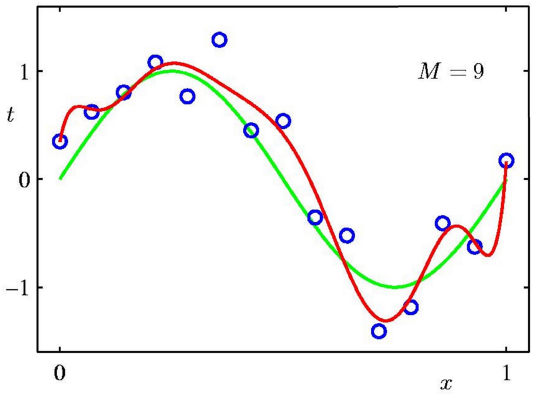
Pros: simple.

Cons: model might be too weak.

Solution 2: Use a larger training set

Pros:

Cons:



Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

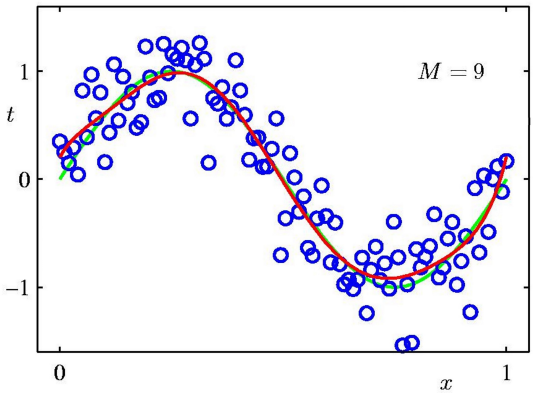
Pros: simple.

Cons: model might be too weak.

Solution 2: Use a larger training set

Pros:

Cons:



Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

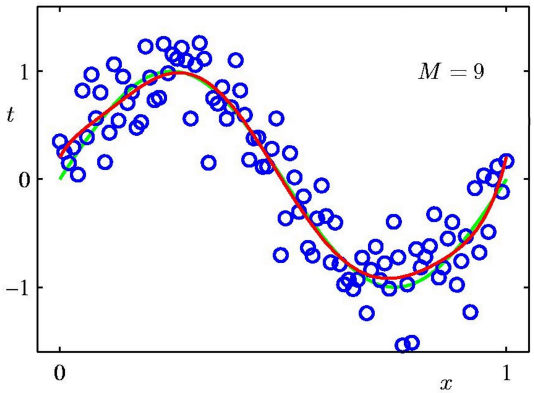
Pros: simple.

Cons: model might be too weak.

Solution 2: Use a larger training set

Pros: no need to reduce model strength.

Cons:



Dealing with Overfitting: Models & Training Set

Solution 1: Use a smaller model

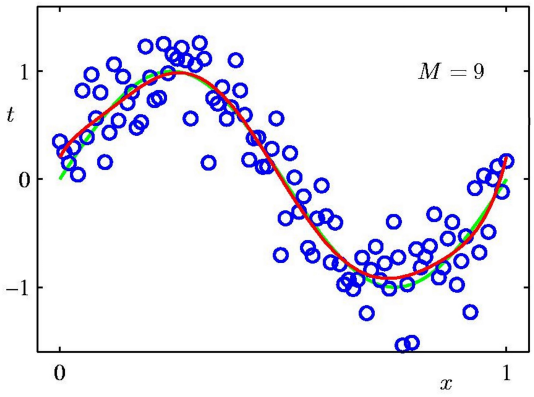
Pros: simple.

Cons: model might be too weak.

Solution 2: Use a larger training set

Pros: no need to reduce model strength.

Cons: increased training complexity, can be difficult to find more training data.



Dataset Augmentation

- Collecting more data can be difficult.

Dataset Augmentation

- Collecting more data can be difficult.
- **Solution:** transform existing dataset!

Dataset Augmentation

- Collecting more data can be difficult.
- **Solution:** transform existing dataset!

Original

Dataset Augmentation

- Collecting more data can be difficult.
- **Solution:** transform existing dataset!

Original

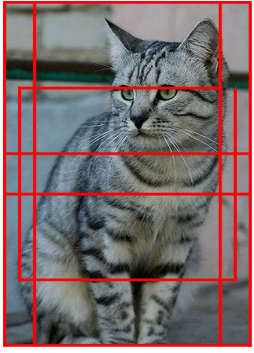
Flip

Dataset Augmentation

- Collecting more data can be difficult.
- **Solution:** transform existing dataset!

Original

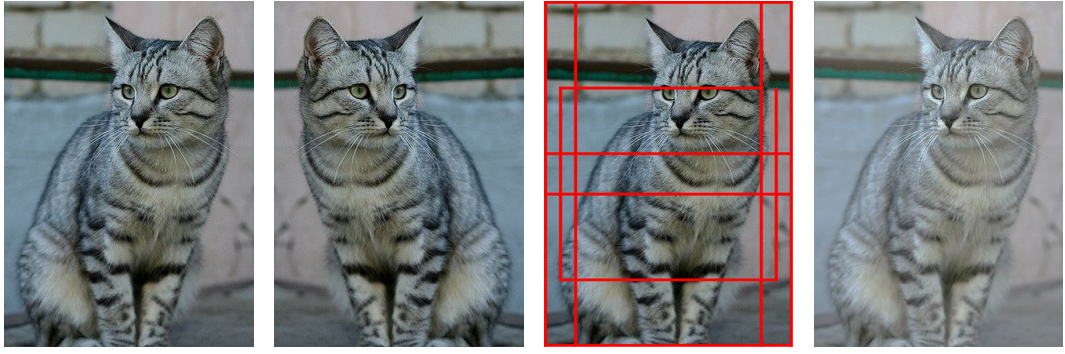
Flip



Crop & Resize

Dataset Augmentation

- Collecting more data can be difficult.
- **Solution:** transform existing dataset!



Original | Flip | Crop & Resize | Brightness & Contrast

Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

Solution 3: Encourage small weights

Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda \left| \mathbf{w} \right|_2^2$$

Pros:

Cons:

Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

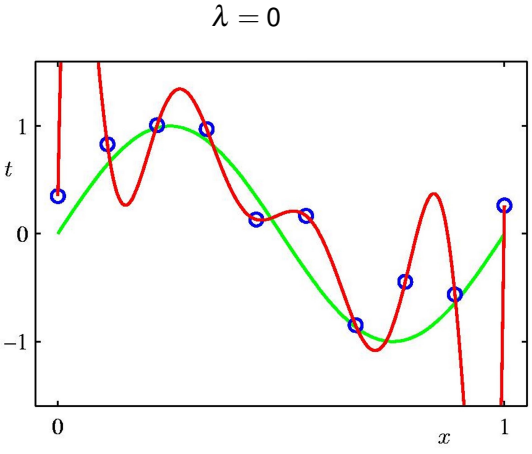
Solution 3: Encourage small weights

Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda |\mathbf{w}|_2^2$$

Pros:

Cons:



Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

Solution 3: Encourage small weights

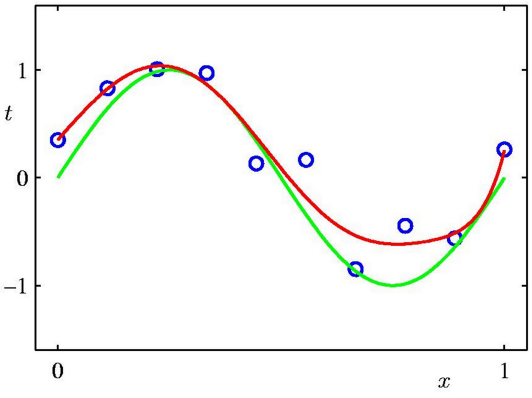
Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda |\mathbf{w}|_2^2$$

Pros:

Cons:

$\lambda = e^{-18}$



Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

Solution 3: Encourage small weights

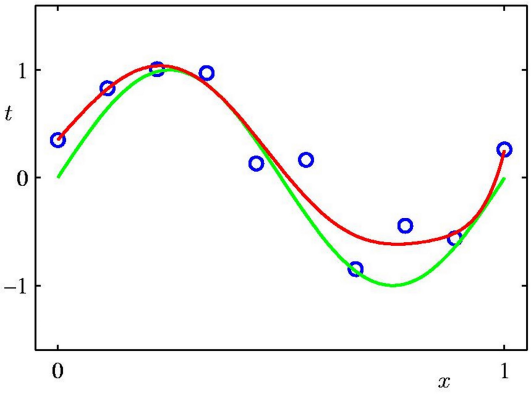
Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda |\mathbf{w}|_2^2$$

Pros: effective, same model & dataset.

Cons:

$\lambda = e^{-18}$



Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

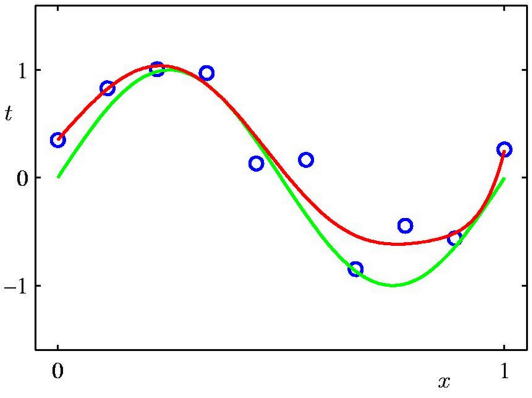
Solution 3: Encourage small weights

Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda |\mathbf{w}|_2^2$$

Pros: effective, same model & dataset.
Cons: more complex training, λ tuning.

$\lambda = e^{-18}$



Dealing with Overfitting: Regularization/Weight Decay

- Model values fluctuate wildly due to **very large weights**.

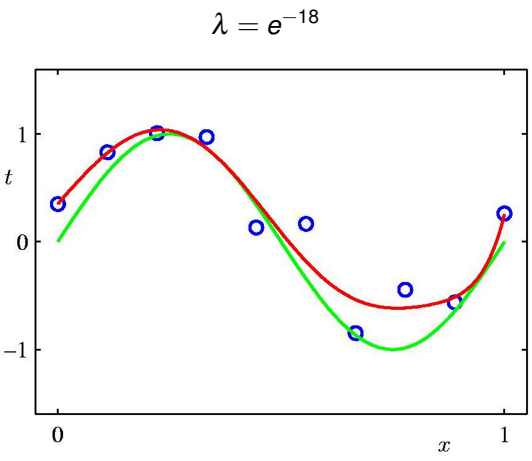
Solution 3: Encourage small weights

Modify loss function ($\lambda \geq 0$):

$$L(\mathbf{t}, \hat{\mathbf{t}}, \mathbf{w}) = \sum_{i=1}^{N} \left(t^{(i)} - \hat{t}^{(i)} \right)^2 + \lambda |\mathbf{w}|_2^2$$

Pros: effective, same model & dataset.
Cons: more complex training, λ tuning.

Alternative: add $\lambda |\mathbf{w}|_1$ to encourage sparse weights (see "Pruning" lecture).



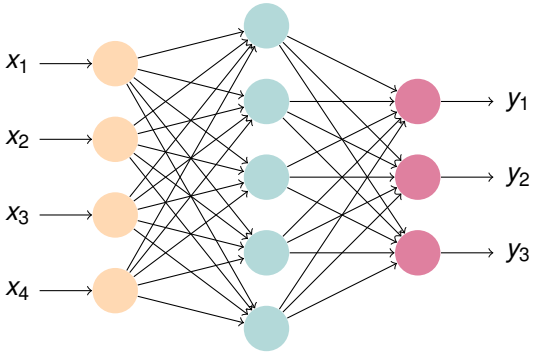
Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros:

Cons:



[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

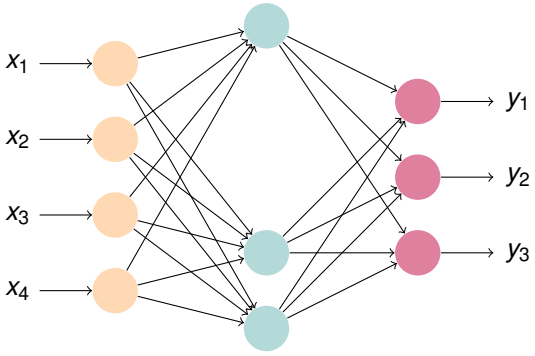
Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros:

Cons:



[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

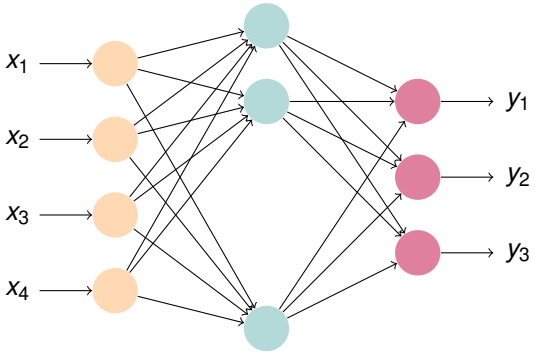
Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros:

Cons:



[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

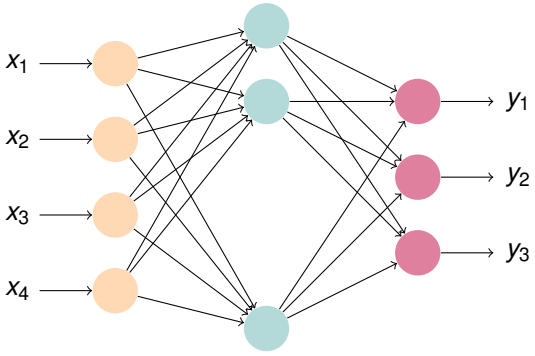
Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros: simple, effective.

Cons:



[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

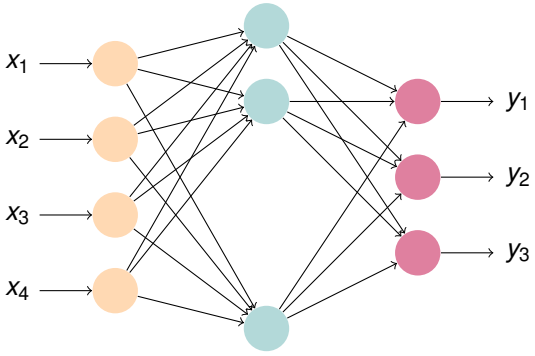
Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros: simple, effective.

Cons: more complex training, patented.



[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

Dealing with Overfitting: Dropout

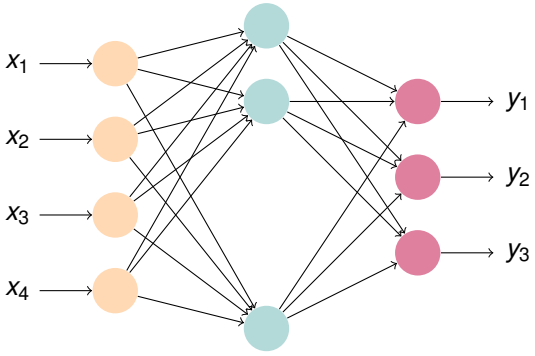
Solution 4: Dropout [1]

Remove neurons with probability p during the training process.

Pros: simple, effective.

Cons: more complex training, patented.

Variation: randomly drop weights.



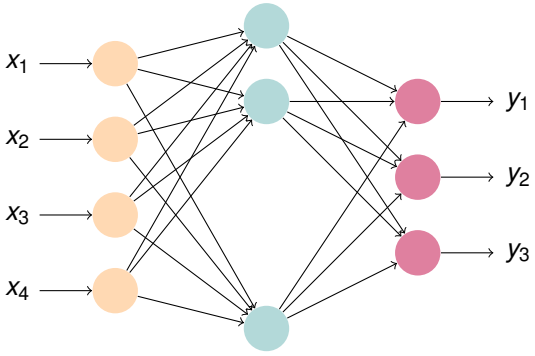
[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

Dealing with Overfitting: Dropout

Solution 4: Dropout [1]

Remove neurons with probability p during the training process.
Pros: simple, effective.
Cons: more complex training, patented.

Variation: randomly drop weights.



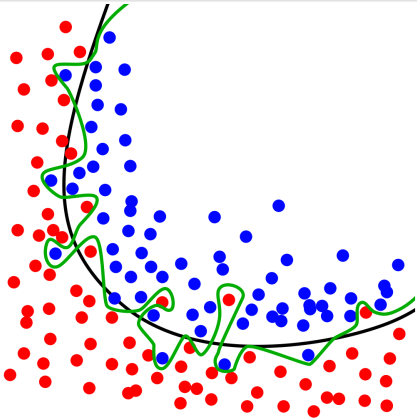
- Previous methods are general, this method is specific to neural networks!

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," 2014.

Overfitting: Summary

Problem

Overfitting occurs when the model fits the training data **too perfectly**.



Overfitting: Summary

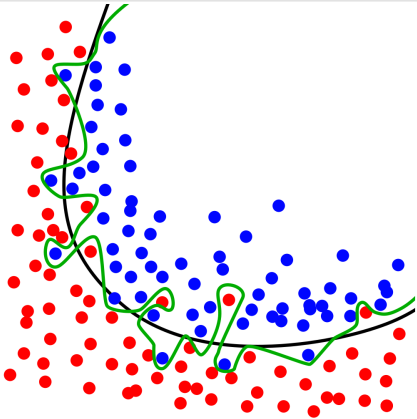
Problem

Overfitting occurs when the model fits the training data **too perfectly**.

Solution

Constrain the model:

1. Smaller model



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Overfitting: Summary

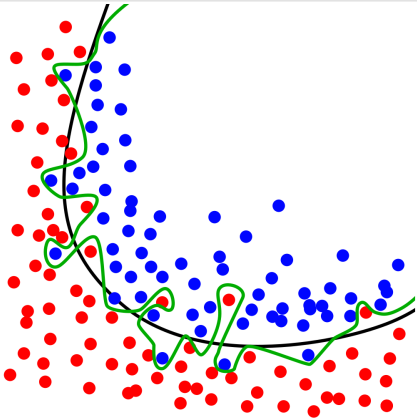
Problem

Overfitting occurs when the model fits the training data **too perfectly**.

Solution

Constrain the model:

1. Smaller model
2. Larger training set (data augmentation)



Overfitting: Summary

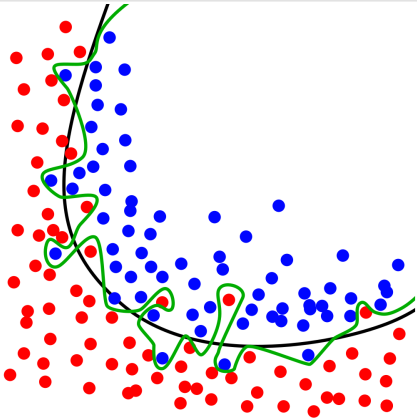
Problem

Overfitting occurs when the model fits the training data **too perfectly**.

Solution

Constrain the model:

1. Smaller model
2. Larger training set (data augmentation)
3. Regularization/weight decay



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Overfitting: Summary

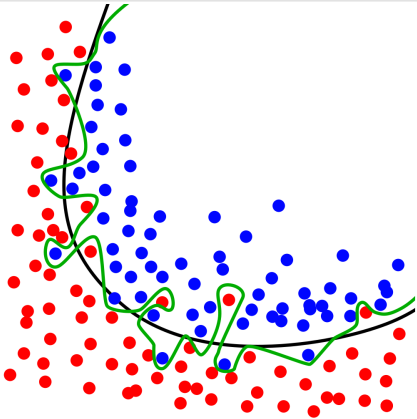
Problem

Overfitting occurs when the model fits the training data **too perfectly**.

Solution

Constrain the model:

1. Smaller model
2. Larger training set (data augmentation)
3. Regularization/weight decay
4. Dropout



Overfitting: Summary

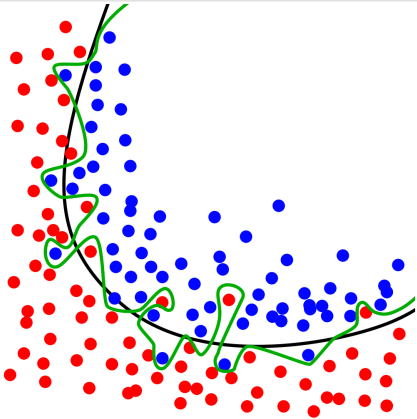
Problem

Overfitting occurs when the model fits the training data **too perfectly**.

Solution

Constrain the model:

1. Smaller model
2. Larger training set (data augmentation)
3. Regularization/weight decay
4. Dropout
5. ...and many more (ensembling, remove features, early stopping)!



Chabacano, CC BY-SA 4.0, via Wikimedia Commons

Hyperparameters

Definition

Parameters that are not derived via training are called **hyperparameters**.

Hyperparameters

Examples:

Definition

Parameters that are not derived via training are called **hyperparameters**.

Hyperparameters

Definition

Parameters that are not derived via training are called **hyperparameters**.

Examples:

1. Structure of the neural network
2. Activation function
3. Learning rate η
4. Mini-batch size B
5. Type of weight initialization
6. Type of data normalization
7. Type of optimization algorithm
   - Momentum weight α
   - RMSProp weight γ
   - Adam weights β_1 and β_2
8. Regularization weight λ
9. Dropout probability p

Hyperparameters

Definition

Parameters that are not derived via training are called **hyperparameters**.

Selecting good hyperparameter values is **critical**!

Problem

The search space is **huge**. With 10 hyperparameters and 5 options for each: **almost 10 million trainings**!

Examples:

1. Structure of the neural network
2. Activation function
3. Learning rate η
4. Mini-batch size B
5. Type of weight initialization
6. Type of data normalization
7. Type of optimization algorithm
   - Momentum weight α
   - RMSProp weight γ
   - Adam weights β_1 and β_2
8. Regularization weight λ
9. Dropout probability p

Hyperparameter Tuning

- Part of dataset is used as a **validation set**: training/validation/test sets are **isolated**.
- This avoids overfitting of hyperparameter choices to test set.

Hyperparameter Tuning

- Part of dataset is used as a **validation set**: training/validation/test sets are **isolated**.
- This avoids overfitting of hyperparameter choices to test set.

Strategy 1: Exhaustive search/grid search

Very simple, can be parallelized, can be feasible in certain cases with few hyperparameters.

Hyperparameter Tuning

- Part of dataset is used as a **validation set**: training/validation/test sets are **isolated**.
- This avoids overfitting of hyperparameter choices to test set.

Strategy 1: Exhaustive search/grid search

Very simple, can be parallelized, can be feasible in certain cases with few hyperparameters.

Strategy 2: Random search

Very simple, can be parallelized, feasible for larger sets, but may miss “best” solution.

Hyperparameter Tuning

- Part of dataset is used as a **validation set**: training/validation/test sets are **isolated**.
- This avoids overfitting of hyperparameter choices to test set.

Strategy 1: Exhaustive search/grid search

Very simple, can be parallelized, can be feasible in certain cases with few hyperparameters.

Strategy 2: Random search

Very simple, can be parallelized, feasible for larger sets, but may miss “best” solution.

Strategy 3: Guided search

Based on, e.g., Bayesian learning or genetic algorithms. More complex, but more effective.

Hyperparameter Tuning

- Part of dataset is used as a **validation set**: training/validation/test sets are **isolated**.
- This avoids overfitting of hyperparameter choices to test set.

Strategy 1: Exhaustive search/grid search

Very simple, can be parallelized, can be feasible in certain cases with few hyperparameters.

Strategy 2: Random search

Very simple, can be parallelized, feasible for larger sets, but may miss “best” solution.

Strategy 3: Guided search

Based on, e.g., Bayesian learning or genetic algorithms. More complex, but more effective.

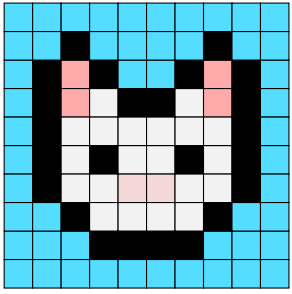
More details: “Neural architecture search” lecture.

Time For A Break

"*Running the same system harder or faster will not change the pattern as long as the structure is not revised.*"

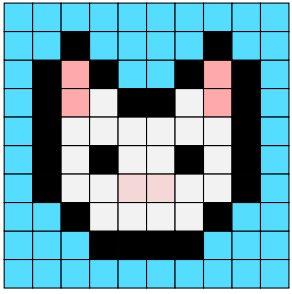
– Dennis Meadows, "The Limits to Growth: The 30-Year Update," 2004.

Issues With Feedforward Neural Networks



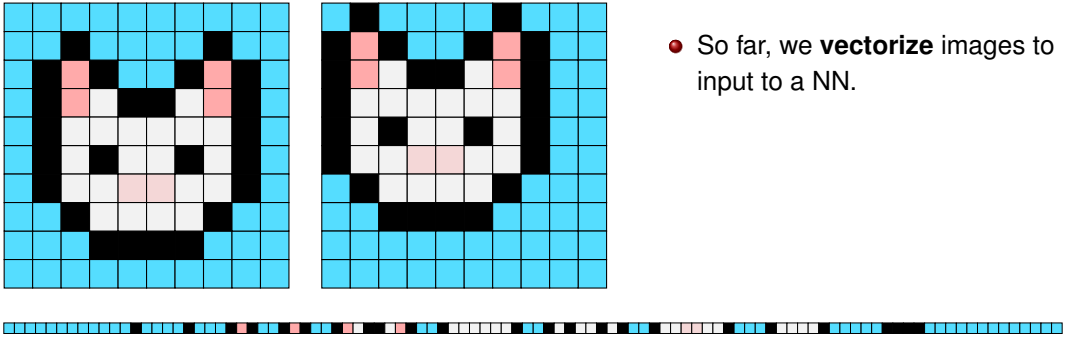
- So far, we **vectorize** images to input to a NN.

Issues With Feedforward Neural Networks



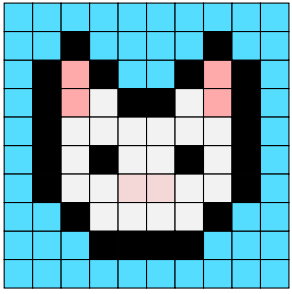
- So far, we **vectorize** images to input to a NN.

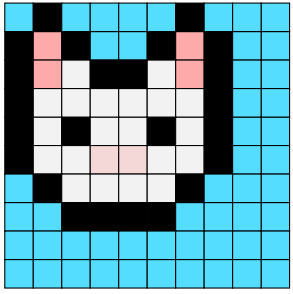
Issues With Feedforward Neural Networks



- So far, we **vectorize** images to input to a NN.

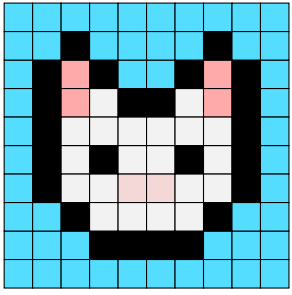
Issues With Feedforward Neural Networks

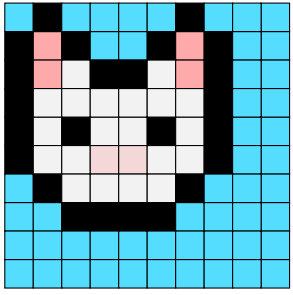




- So far, we **vectorize** images to input to a NN.

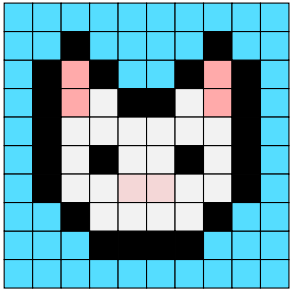
Issues With Feedforward Neural Networks

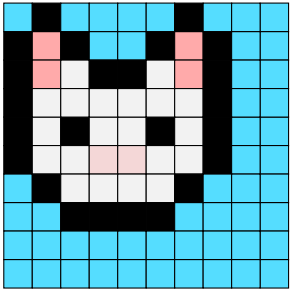




- So far, we **vectorize** images to input to a NN.
- A **shift** completely changes the input vector, even though it's **effectively the same image**!

Issues With Feedforward Neural Networks





- So far, we **vectorize** images to input to a NN.
- A **shift** completely changes the input vector, even though it's **effectively the same image**!

- Using RGB, the first hidden layer alone has $10 \times 10 \times 3 = 300$ **weights per neuron!**

The Convolution Operation

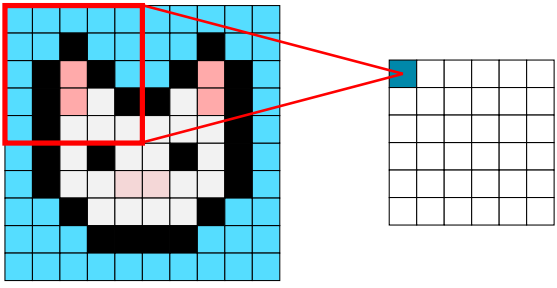
Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image

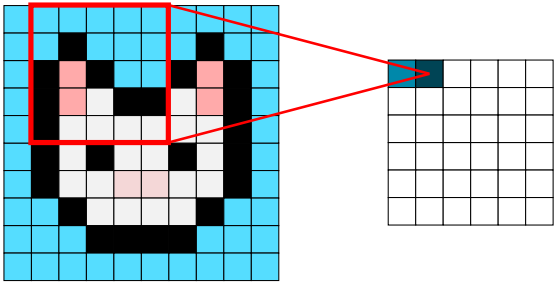


- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image

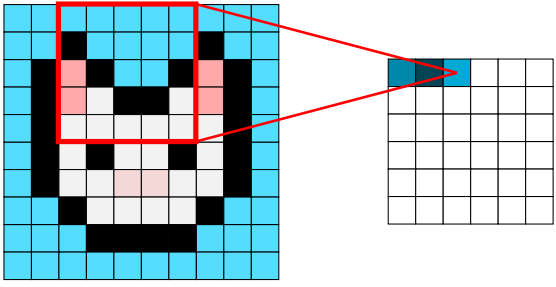


- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image

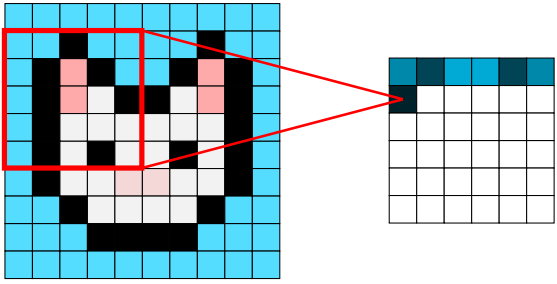


- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image

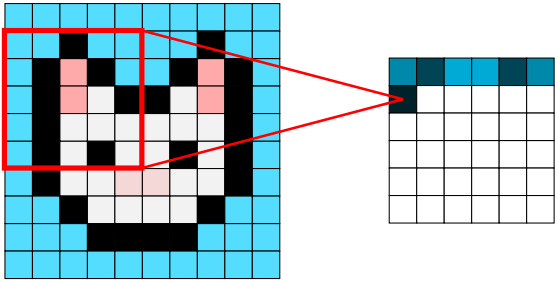


- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image



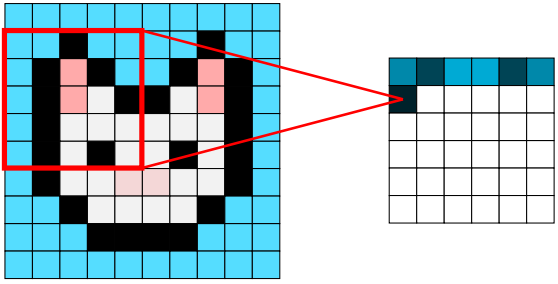
- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

- **Input:** $W \times H$
- **Filter:** $S \times R$
- **Output:** $(W-S+1) \times (H-R+1)$

The Convolution Operation

Natural images have certain properties:

1. **Locality:** nearby pixels are correlated
2. **Translation invariance:** meaningful patterns can occur anywhere in the image



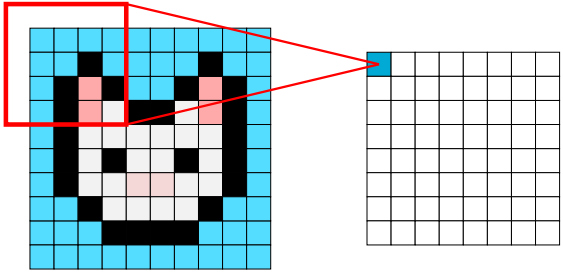
- **Convolutional neural networks** exploit these properties by using localized **filters** (aka **kernels**).

- **Input:** $W \times H$
- **Filter:** $S \times R$
- **Output:** $(W-S+1) \times (H-R+1)$

Advantages

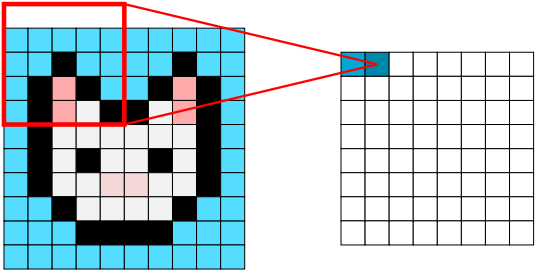
1) **Fewer weights** due to sparse connectivity and 2) **translation equivariance**

Padding



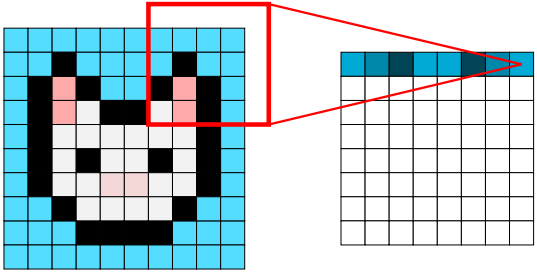
- **Padding:** use P pixels outside input

Padding



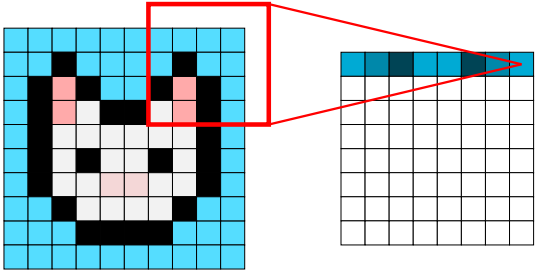
- **Padding:** use P pixels outside input

Padding



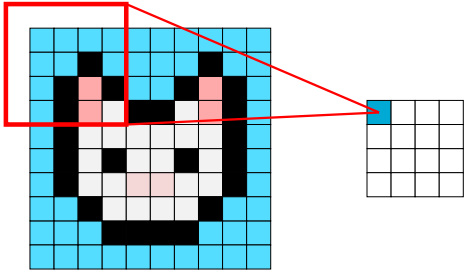
- **Padding:** use P pixels outside input

Padding



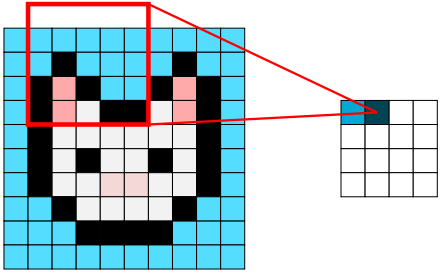
- **Padding:** use P pixels outside input
- **Input:** $W \times H$
- **Filter:** $S \times R$
- **Output:** $(W-S+2P+1) \times (H-R+2P+1)$

Stride



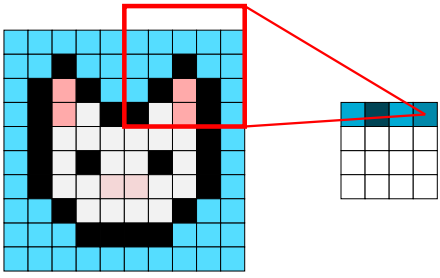
- **Stride:** steps of T pixels

Stride



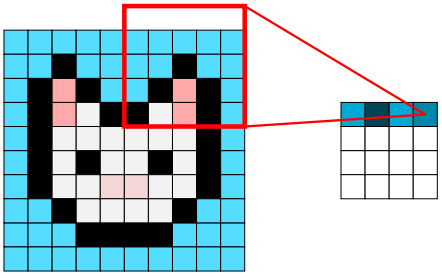
- **Stride:** steps of T pixels

Stride



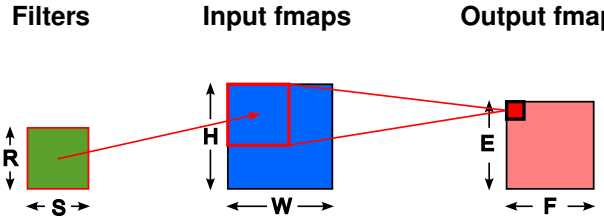
- **Stride:** steps of T pixels

Stride



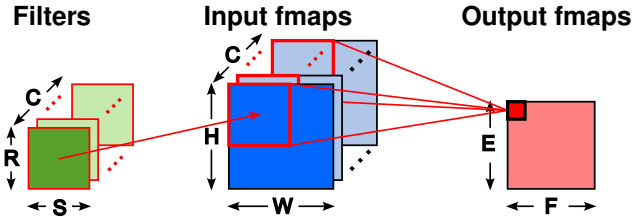
- **Stride:** steps of T pixels
- **Input:** $W \times H$
- **Filter:** $S \times R$
- **Output:** $(\lfloor \frac{W-S+2P}{T} \rfloor + 1) \times (\lfloor \frac{H-R+2P}{T} \rfloor + 1)$

The Convolutional Layer



- **Filters**: $S \times R$
- **Input fmaps**: $W \times H$
- **Output fmaps**: $F \times E$

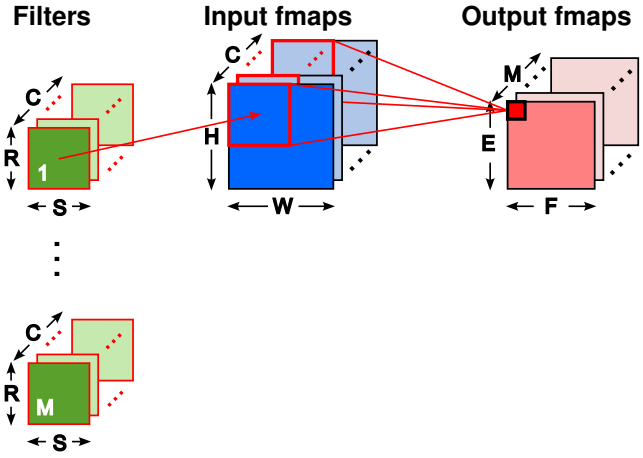
The Convolutional Layer



- **Filters**: $S \times R \times C$
- **Input fmaps**: $W \times H \times C$
- **Output fmaps**: $F \times E$

- **Input channels:** C

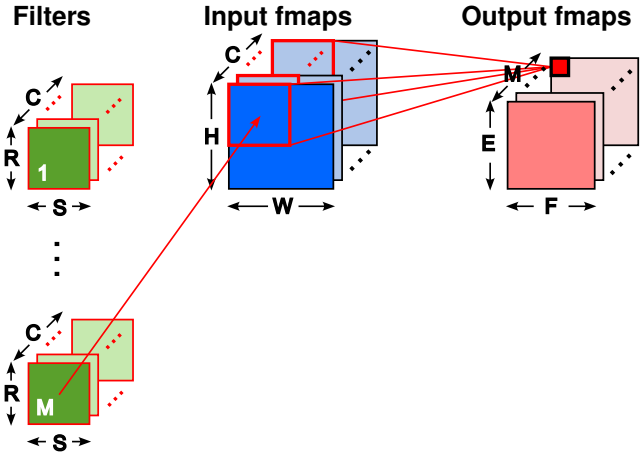
The Convolutional Layer



- **Filters**: $S \times R \times C \times M$
- **Input fmaps**: $W \times H \times C$
- **Output fmaps**: $F \times E \times M$

- **Input channels:** C
- **Output channels:** M

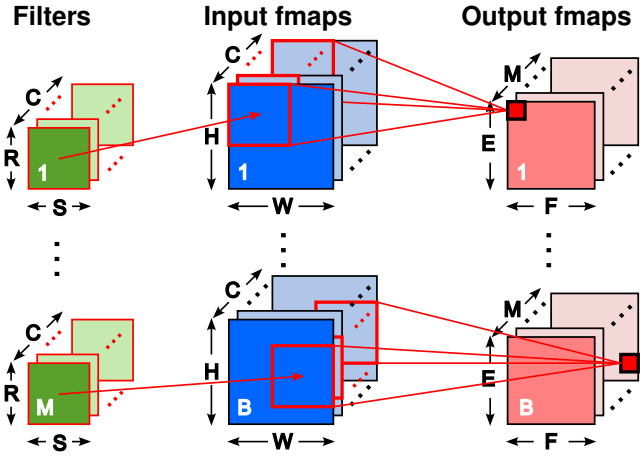
The Convolutional Layer



- **Filters**: $S \times R \times C \times M$
- **Input fmaps**: $W \times H \times C$
- **Output fmaps**: $F \times E \times M$

- **Input channels:** C
- **Output channels:** M

The Convolutional Layer



- **Filters**: $S \times R \times C \times M$
- **Input fmaps**: $W \times H \times C \times B$
- **Output fmaps**: $F \times E \times M \times B$

- **Input channels:** C
- **Output channels:** M
- **Batch size:** B

Pooling & Unpooling

Pooling is used between convolutional layers and **reduces the dimensions** of data.

Pooling & Unpooling

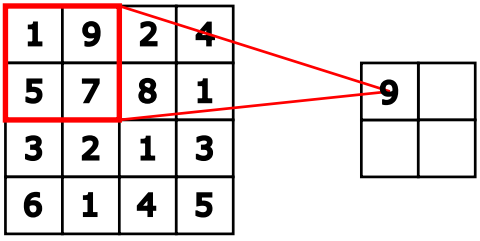
Pooling is used between convolutional layers and **reduces the dimensions** of data.

1	9	2	4
5	7	8	1
3	2	1	3
6	1	4	5

- **Max pooling** keeps the maximum value of each block.

Pooling & Unpooling

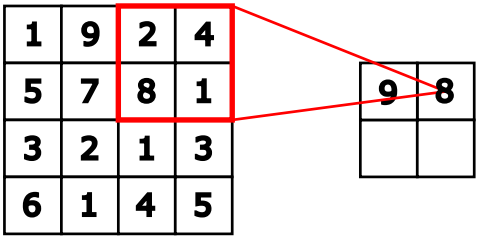
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.

Pooling & Unpooling

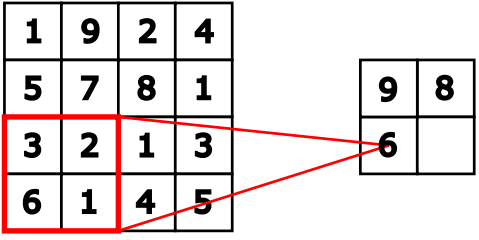
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.

Pooling & Unpooling

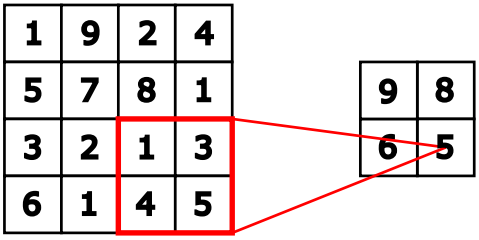
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.

Pooling & Unpooling

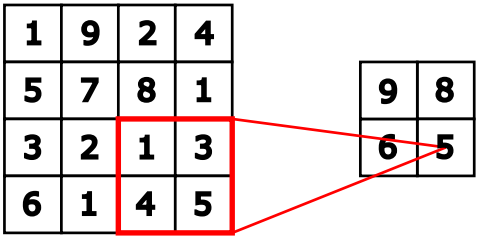
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.

Pooling & Unpooling

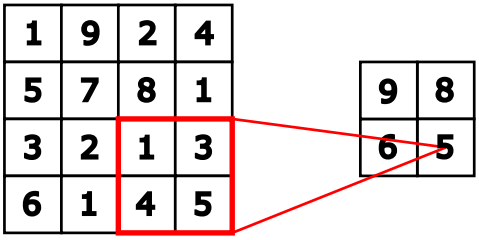
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.
- **Average pooling** keeps the average value of each block.

Pooling & Unpooling

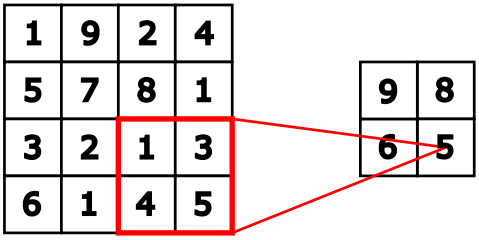
Pooling is used between convolutional layers and **reduces the dimensions** of data.



- **Max pooling** keeps the maximum value of each block.
- **Average pooling** keeps the average value of each block.
- **Padding** and **stride** are used.

Pooling & Unpooling

Pooling is used between convolutional layers and **reduces the dimensions** of data.

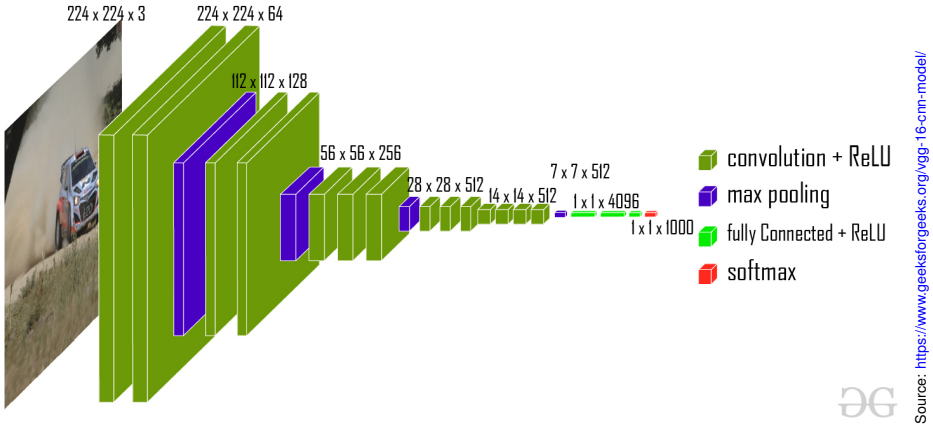


- **Max pooling** keeps the maximum value of each block.
- **Average pooling** keeps the average value of each block.
- **Padding** and **stride** are used.

Unpooling can be used to **increase the dimensions** of data. Several options:

1. Nearest neighbor and max unpooling.
2. Learned transpose convolution.

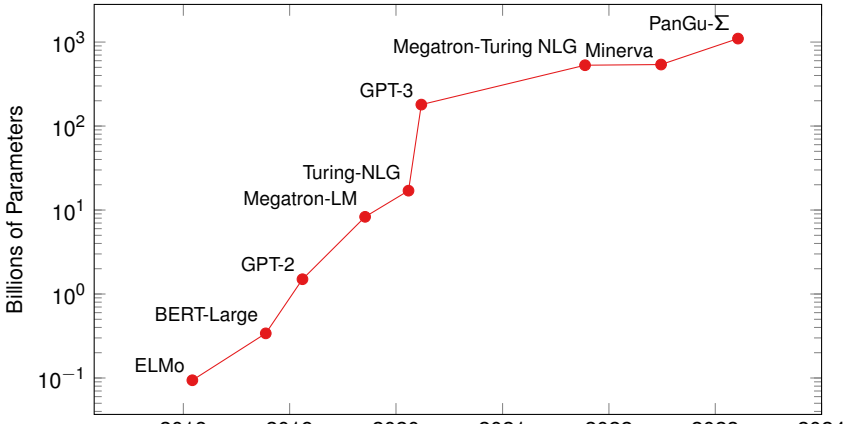
Putting It All Together: A Convolutional Neural Network (VGG-16)



Source: https://www.geeksforgeeks.org/vgg-16-cnn-model/

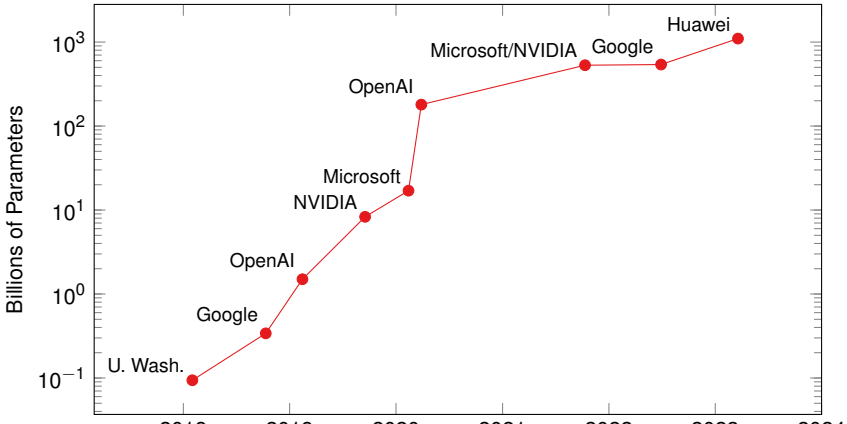
- The number of channels C equals the number of output feature maps M of the previous layer, so C is often omitted.

Model Size Is Exploding



Data source: https://epoch.ai/data/notable-ai-models

Model Size Is Exploding



Data source: https://epoch.ai/data/notable-ai-models

Model Size Problems

Huge model sizes cause a number of problems:

Model Size Problems

Huge model sizes cause a number of problems:

1. **Storage:** large amounts of memory are required.
   - GPT-3 has 175 billion parameters $\rightarrow$ 700 GB of storage

Model Size Problems

Huge model sizes cause a number of problems:

1. **Storage:** large amounts of memory are required.
   - GPT-3 has 175 billion parameters $\rightarrow$ 700 GB of storage
2. **Speed:** training can take ages.
   - GPT-3 took 15 days to train on a supercomputer with 10,000 NVIDIA V100 GPUs [1].

[1] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," 2020.

Model Size Problems

Huge model sizes cause a number of problems:

1. **Storage:** large amounts of memory are required.
   - GPT-3 has 175 billion parameters $\rightarrow$ 700 GB of storage
2. **Speed:** training can take ages.
   - GPT-3 took 15 days to train on a supercomputer with 10,000 NVIDIA V100 GPUs [1].
3. **Energy:** massive amount of computations uses immense energy.
   - GPT-3 used 1287 MWh over 15 days [1], same as **11,000 households** in NL over the same amount of time [2].

[1] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," 2020.
[2] Statistics Netherlands - Energy consumption private dwellings

Model Size Problems

Huge model sizes cause a number of problems:

1. **Storage:** large amounts of memory are required.
   - GPT-3 has 175 billion parameters $\rightarrow$ 700 GB of storage
2. **Speed:** training can take ages.
   - GPT-3 took 15 days to train on a supercomputer with 10,000 NVIDIA V100 GPUs [1].
3. **Energy:** massive amount of computations uses immense energy.
   - GPT-3 used 1287 MWh over 15 days [1], same as **11,000 households** in NL over the same amount of time [2].
4. **De-democratization:** only entities with enormous budgets can develop such models.
   - Training GPT-3 cost **2,100,000 USD**.

[1] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," 2020.
[2] Statistics Netherlands - Energy consumption private dwellings

Model Size Problems

Huge model sizes cause a number of problems:

1. **Storage:** large amounts of memory are required.
   - GPT-3 has 175 billion parameters $\rightarrow$ 700 GB of storage
2. **Speed:** training can take ages.
   - GPT-3 took 15 days to train on a supercomputer with 10,000 NVIDIA V100 GPUs [1].
3. **Energy:** massive amount of computations uses immense energy.
   - GPT-3 used 1287 MWh over 15 days [1], same as **11,000 households** in NL over the same amount of time [2].
4. **De-democratization:** only entities with enormous budgets can develop such models.
   - Training GPT-3 cost **2,100,000 USD**.
   - Training GPT-4 cost **41,100,000 USD!!!**

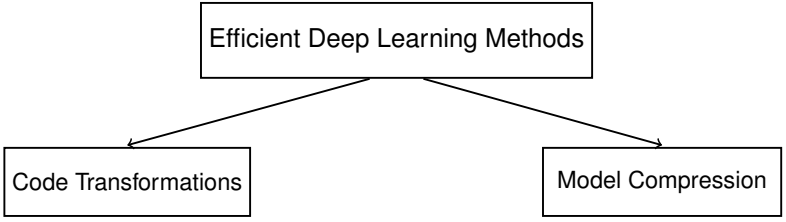
[1] D. Patterson *et al.*, "Carbon Emissions and Large Neural Network Training," 2020.
[2] Statistics Netherlands - Energy consumption private dwellings

Optimizations

- Optimizations can help mitigate these issues!

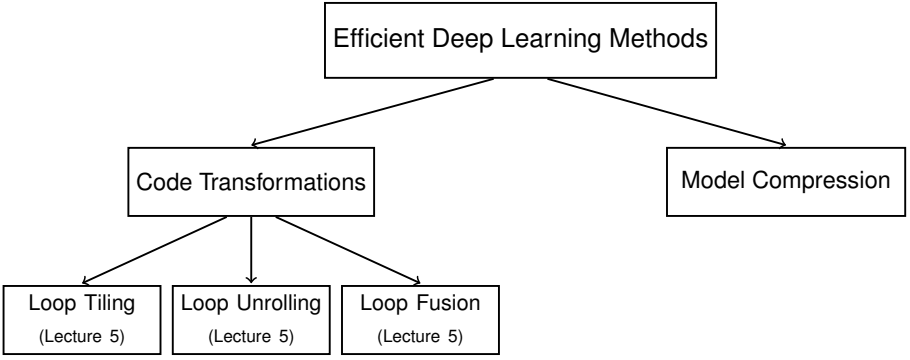
Optimizations

- Optimizations can help mitigate these issues!



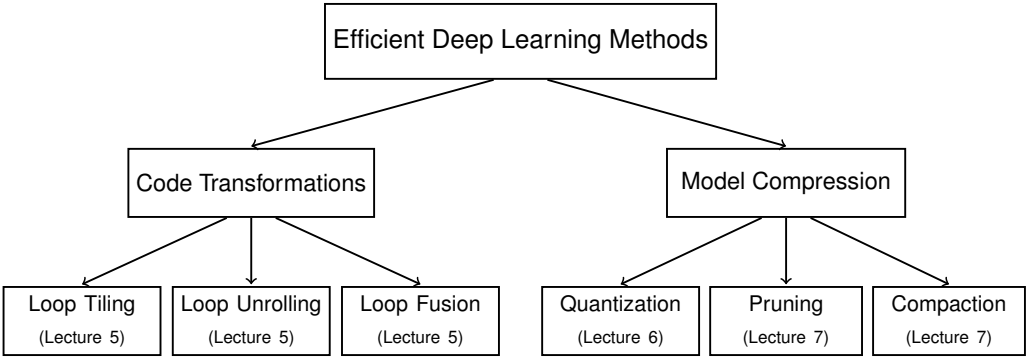
Optimizations

- Optimizations can help mitigate these issues!



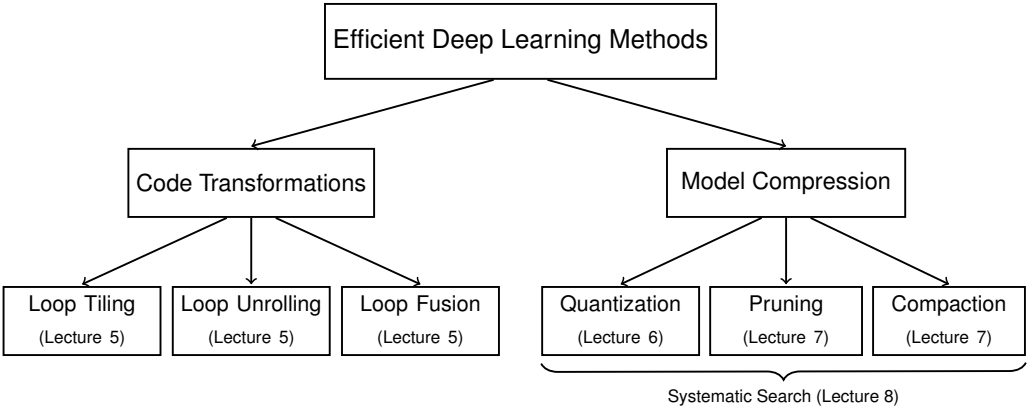
Optimizations

- Optimizations can help mitigate these issues!



Optimizations

- Optimizations can help mitigate these issues!



Time For A Break

"*Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius – and a lot of courage – to move in the opposite direction.*"

– Ernst F. Schumacher, "Small Is Beautiful: A Study of Economics As If People Mattered," 1973.

Conclusion

Summary:

1. Overfitting is an issue but can be overcome: **constrain the model**.
2. Convolutional neural networks exploit spatial image properties to reduce complexity.
3. **Optimization is crucial** to keep complexity explosion in check.

Conclusion

Summary:

1. Overfitting is an issue but can be overcome: **constrain the model**.
2. Convolutional neural networks exploit spatial image properties to reduce complexity.
3. **Optimization is crucial** to keep complexity explosion in check.

Next time:

1. Loop transformations.
2. Quantization.