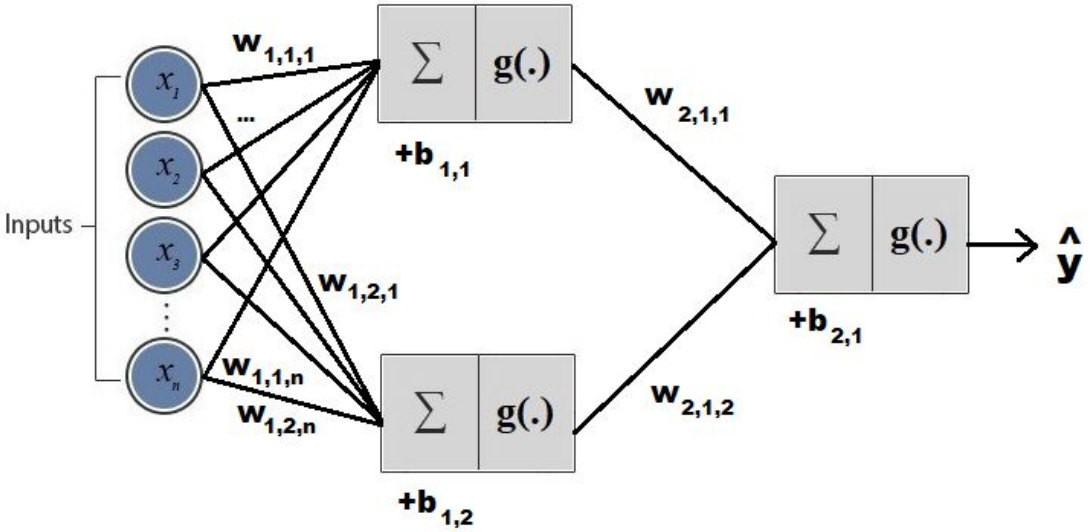
Deep Network Development

Lecture #7

Viktor Varga
Department of Artificial Intelligence, ELTE IK

Last week - Multilayer Perceptron (MLP)



$$W_1 \in \mathbb{R}^{2\times n}$$
$$b_1 \in \mathbb{R}^2$$

$$W_2 \in \mathbb{R}^{1\times 2}$$
$$b_2 \in \mathbb{R}^1$$

New notation: Theta is the set of all weight matrices and bias vectors.
(i.e., the parameters)

$$\Theta = \{W_1, b_1, W_2, b_2\}$$

Last week - Multilayer Perceptron (MLP)

The hypothesis function of a two-layer MLP neural network:

$$h(x) = g_2(W_2\, g_1(W_1 x + b_1) + b_2) = \hat{y} \approx y$$

The output of the first layer

Loss functions:

- **We always need intermediate activation functions** (nonlinearities) **in MLPs**
- **The last activation function depends on the task**

- **Regression:** MSE
- **Classification** (binary)**:** Logistic loss / Binary Cross-entropy (BCE)

Last week - Multilayer Perceptron (MLP) expanded

The hypothesis function of a two-layer MLP neural network:

$$h(x) = g_2(W_2\ g_1(W_1 x + b_1) + b_2) = \hat{y} \approx y$$

Expanded:

$$h(x) = w_{2,1,1}\, g(x_1 w_{1,1,1} + x_2 w_{1,1,2} + \cdots + x_n w_{1,1,n} + b_{1,1}) +$$
$$w_{2,1,2}\, g(x_1 w_{1,2,1} + x_2 w_{1,2,2} + \cdots + x_n w_{1,2,n} + b_{1,2}) + b_{2,1} = \hat{y} \approx y$$

$$J(\Theta) = \frac{1}{2m} \sum_{j=1}^{m} (h(x^{(j)}) - y^{(j)})^2$$

$$\Theta = \{w_{1,1,1}, w_{1,1,2}, \ldots, b_{1,1}, \ldots\}$$

Last week - Training an MLP

We will use gradient descent…

We need to calculate the derivatives of complicated expressions…

repeat until convergence {

$$\text{for } \forall\, \theta \in \Theta \quad \{$$

$$grad_\theta = \frac{\partial}{\partial \theta} J(\Theta)$$

}

$$\text{for } \forall\, \theta \in \Theta \quad \{$$

$$\theta = \theta - \alpha\, grad_\theta$$

}

}

Last week - Chain rule

Reminder: Derivatives of composite functions

$$(f \circ g)' = (f' \circ g) \cdot g'$$

The same with **Leibniz-notation**:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial z} \cdot \frac{\partial z}{\partial x} \quad \text{where} \quad z := g(x),\ u := f(z)$$

The derivative of the expression **u** with respect to **x**

The rule for differentiating composite functions is also known as the **chain rule**.

Last week - Chain rule

$$f(x) = \frac{1}{1+e^{-x}} \qquad \frac{\partial f(x)}{\partial x} = \; ?$$

Chain rule: We calculate the derivatives of the intermediate variables with respect to the chained variable, then multiply the results together.

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial q} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial x} = (-1) \cdot (e^{-x}) \cdot -\frac{1}{(1+e^{-x})^2}$$

$$z := -x$$

$$q := 1 + e^{z}$$

$$f(x) := \frac{1}{q} = q^{-1}$$

$$\frac{\partial z}{\partial x} = -1$$

$$\frac{\partial q}{\partial z} = e^{z} = e^{-x}$$

$$\frac{\partial f(x)}{\partial q} = -q^{-2} = -\frac{1}{(1+e^{-x})^2}$$

Last week - Training a Multilayer Perceptron (MLP)

The derivative of the loss must be calculated w.r.t. each parameter!

$$\frac{\partial J(\Theta)}{\partial b_{2,1}} = \boxed{\frac{\partial J(\Theta)}{\partial \hat{y}}} \cdot \frac{\partial \hat{y}}{\partial b_{2,1}}$$

$$\frac{\partial J(\Theta)}{\partial w_{2,1,1}} = \boxed{\frac{\partial J(\Theta)}{\partial \hat{y}}} \cdot \boxed{\frac{\partial \hat{y}}{\partial z_1}} \cdot \frac{\partial z_1}{\partial w_{2,1,1}}$$

$$\frac{\partial J(\Theta)}{\partial w_{1,1,1}} = \boxed{\frac{\partial J(\Theta)}{\partial \hat{y}}} \cdot \boxed{\frac{\partial \hat{y}}{\partial z_1}} \cdot \frac{\partial z_1}{\partial q_1} \cdot \frac{\partial q_1}{\partial w_{1,1,1}}$$

. . .

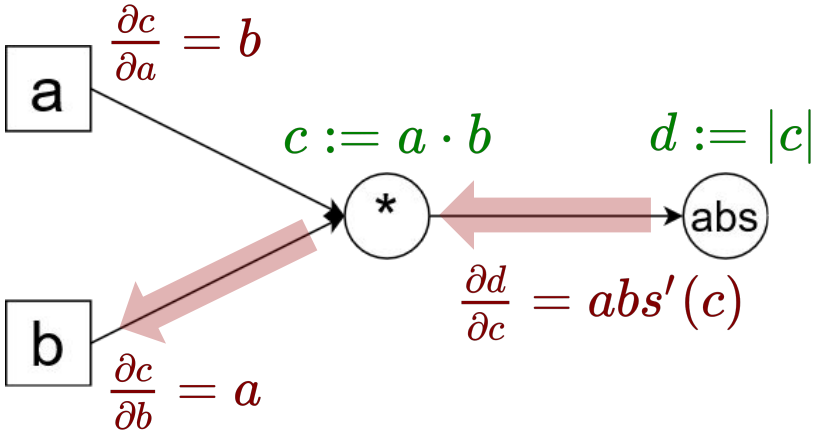
The same members appear multiple times in the derivatives. **It is useless to calculate them multiple times.**

Gradient method + keeping track of sub-results
→ **Backpropagation algorithm**

Last week - Backpropagation algorithm

Backpropagation algorithm

Computational graph: Representing complicated expressions



$$\frac{\partial d}{\partial b} = \frac{\partial d}{\partial c} \cdot \frac{\partial c}{\partial b} = abs'(c) \cdot a$$

Last week - Training an MLP

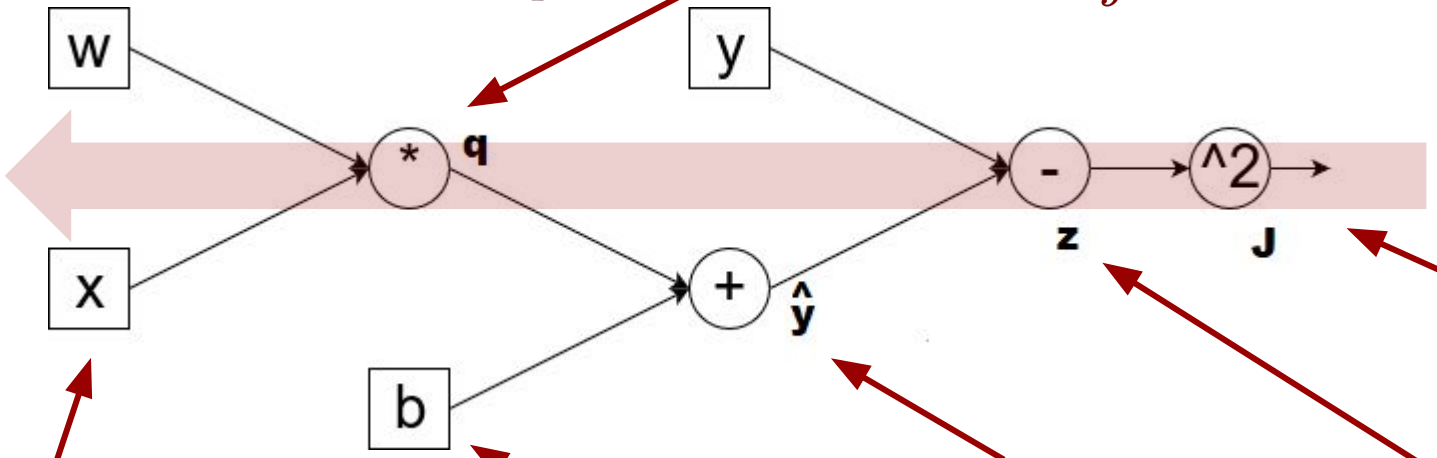
Step 1: Symbolic derivation

$$q = wx$$

$$\hat{y} = q + b$$

$$z = y - \hat{y}$$

$$J = z^2$$



$$\frac{\partial J}{\partial w} = x \cdot \frac{\partial J}{\partial q}$$

$$\frac{\partial J}{\partial q} = 1 \cdot \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial J} = 1$$

$$\frac{\partial J}{\partial x} = w \cdot \frac{\partial J}{\partial q}$$

$$\frac{\partial J}{\partial b} = 1 \cdot \frac{\partial J}{\partial \hat{y}}$$

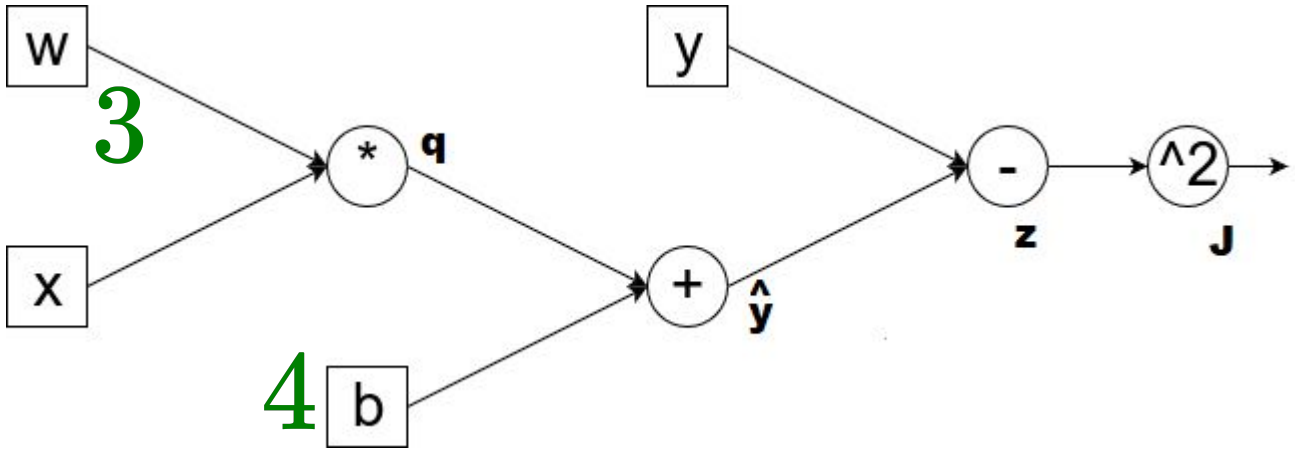
$$\frac{\partial J}{\partial \hat{y}} = -1 \cdot \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial z} = 2z \cdot \frac{\partial J}{\partial J}$$

Last week - Training an MLP

Step 2: Parameter initialization

E.g., $w = 3,\ b = 4$



$$q = wx$$

$$\hat{y} = q + b$$

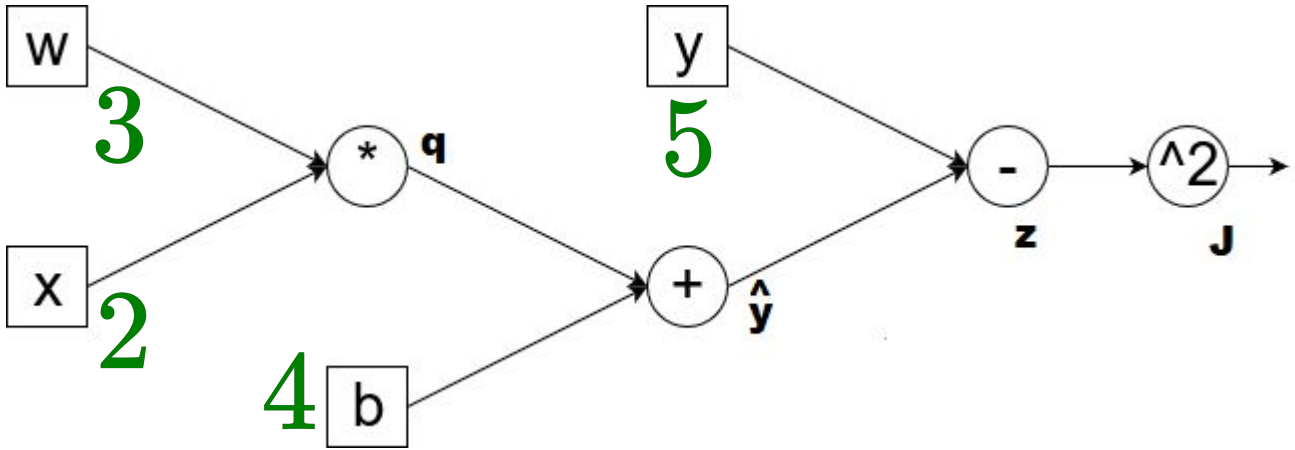
$$z = y - \hat{y}$$

$$J = z^2$$

Last week - Training an MLP

Step 3: Choosing an example from the training set

E.g., $x = 2,\ y = 5$



$$q = wx$$

$$\hat{y} = q + b$$

$$z = y - \hat{y}$$

$$J = z^2$$

Last week - Training an MLP

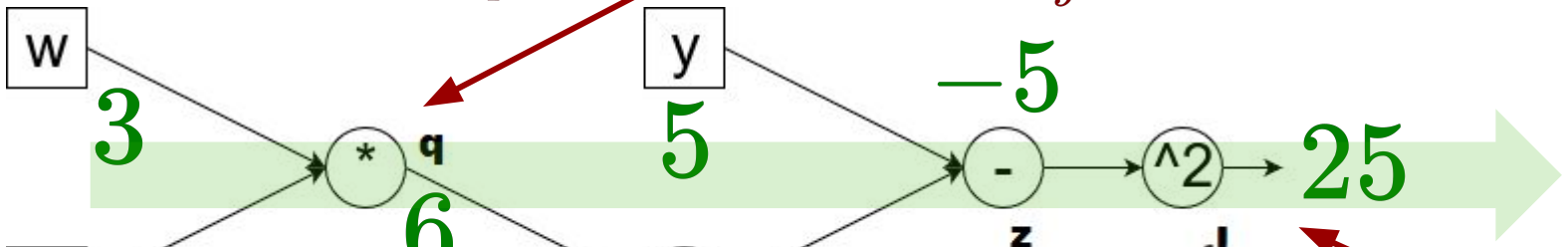
Step 4: Substitution - feed forward

$$q = wx$$

$$\hat{y} = q + b$$

$$z = y - \hat{y}$$

$$J = z^2$$



$$\frac{\partial J}{\partial w} = x \cdot \frac{\partial J}{\partial q}$$

$$\frac{\partial J}{\partial q} = 1 \cdot \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial J} = 1$$

$$\frac{\partial J}{\partial x} = w \cdot \frac{\partial J}{\partial q}$$

$$\frac{\partial J}{\partial b} = 1 \cdot \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial \hat{y}} = -1 \cdot \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial z} = 2z \cdot \frac{\partial J}{\partial J}$$

Last week - Training an MLP

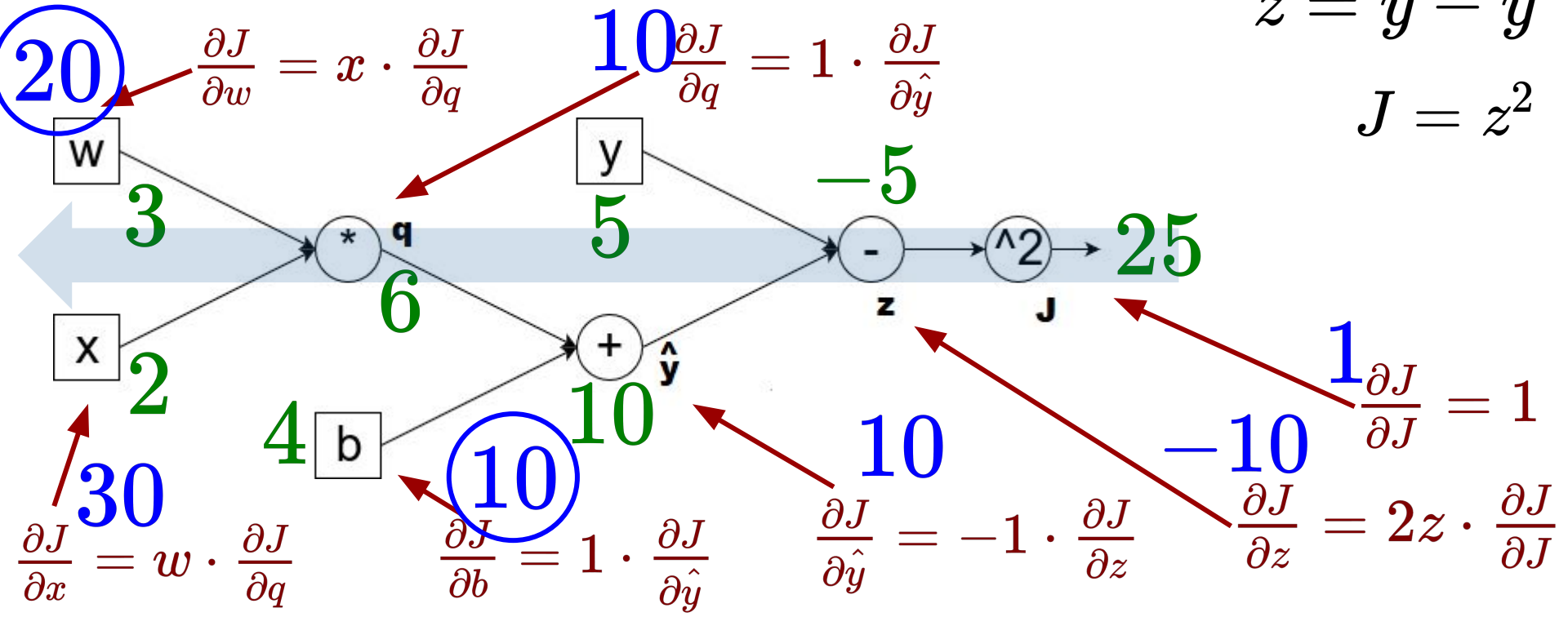
Step 5: Substitution - backpropagation

$$q = wx$$

$$\hat{y} = q + b$$

$$z = y - \hat{y}$$

$$J = z^2$$



Last week - Training an MLP

Step 6: Updating the weights / parameters

$$x = 2,\ w = 3,\ b = 4,\ y = 5$$

$$\frac{\partial J}{\partial w} = 20 \qquad \frac{\partial J}{\partial b} = 10$$

$$w := 3 - 0.1 \cdot 20 = 1$$

$$b := 4 - 0.1 \cdot 10 = 3$$

repeat until convergence {
  for $\forall\, \theta \in \Theta$ {
    $grad_\theta = \frac{\partial}{\partial \theta} J(\Theta)$
  }
  for $\forall\, \theta \in \Theta$ {
    $\theta = \theta - \alpha\, grad_\theta$
  }
}

$$\alpha = 0.1$$

Last week - Training an MLP

Step 1: Symbolic derivation
Step 2: Parameter initialization

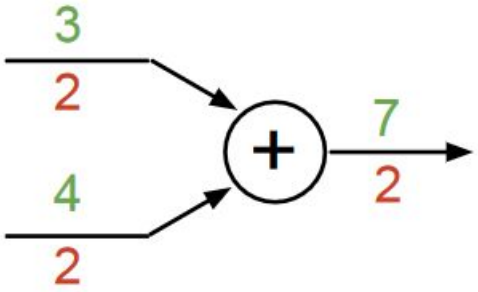
Step 3: Choosing an example (or a mini-batch) from the training set
Step 4: Substitution - feed forward
Step 5: Substitution - backpropagation
Step 6: Updating the parameters

GOTO STEP 3 - Stop after a given num. of iterations OR early stopping

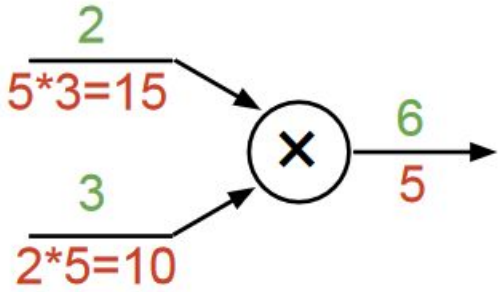
Image source: Stanford CS231n, Fei-Fei Li, J. Johnson, S. Yeung

Backpropagation rules

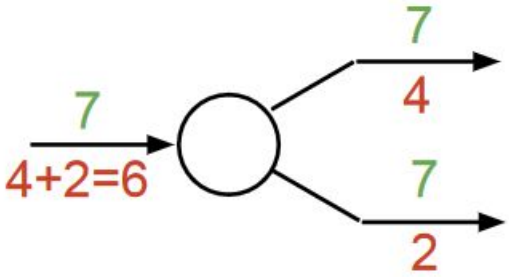
add gate: gradient distributor



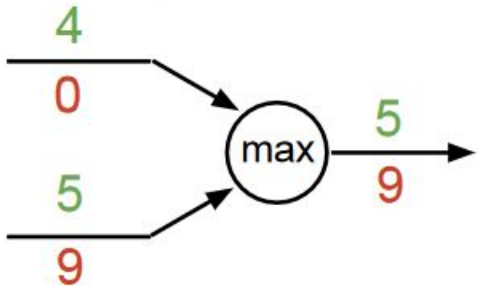
mul gate: “swap multiplier”



copy gate: gradient adder



max gate: gradient router



Measuring model performance

How can we determine how well our trained network is performing?

Measuring model performance

How can we determine how well our trained network is performing?

One option: Evaluating the loss function on the test dataset.

$$J(\Theta) = \frac{1}{2mk}\sum_{j=1}^{m}||\hat{y}^{(j)} - y^{(j)}||_2^2 = \frac{1}{2mk}\sum_{j=1}^{m}\sum_{i=1}^{k}(\hat{y}_i^{(j)} - y_i^{(j)})^2$$

Regression: MSE

$$J(\theta) = \frac{1}{m}\sum_{j=1}^{m}[-y^{(j)}\log(\hat{y}^{(j)}) - (1-y^{(j)})\,log(1-\hat{y}^{(j)})]$$

Bin. class.: Bin. CE

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{k} y_i log(\hat{y}_i)$$

Multi-class class.: CE

Measuring model performance

How can we determine how well our trained network is performing?

One option: Evaluating the loss function on the test dataset.

$$J(\Theta) = \frac{1}{2mk} \sum_{j=1}^{m} ||\hat{y}^{(j)} - y^{(j)}||_2^2 = \frac{1}{2mk} \sum_{j=1}^{m} \sum_{i=1}^{k} (\hat{y}_i^{(j)} - y_i^{(j)})^2$$

Regression: MSE

$$J(\theta) = \frac{1}{m} \sum_{j=1}^{m} [-y^{(j)} \log(\hat{y}^{(j)}) - (1 - y^{(j)}) \, log(1 - \hat{y}^{(j)})]$$

Bin. class.: Bin. CE

$$J(\theta) = -\frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{k} y_i log(\hat{y}_i)$$

Multi-class class.: CE

For example: “Cross-entropy on the test set is 0.74”.

→ **Hard to interpret for a human…**

Measuring model performance

Regression - Metrics

$$\text{MSE}(\hat{y}, y) = \frac{1}{2mk} \sum_{j=1}^{m} \sum_{i=1}^{k} (\hat{y}_i^{(j)} - y_i^{(j)})^2$$

$$\text{RMSE}(\hat{y}, y) = \sqrt{MSE(\hat{y}, y)}$$

Root MSE
(The root of the Mean Squared Error)

$$\text{MAE}(\hat{y}, y) = \frac{1}{mk} \sum_{j=1}^{m} \sum_{i=1}^{k} |\hat{y}_i^{(j)} - y_i^{(j)}|$$

Mean Absolute Error

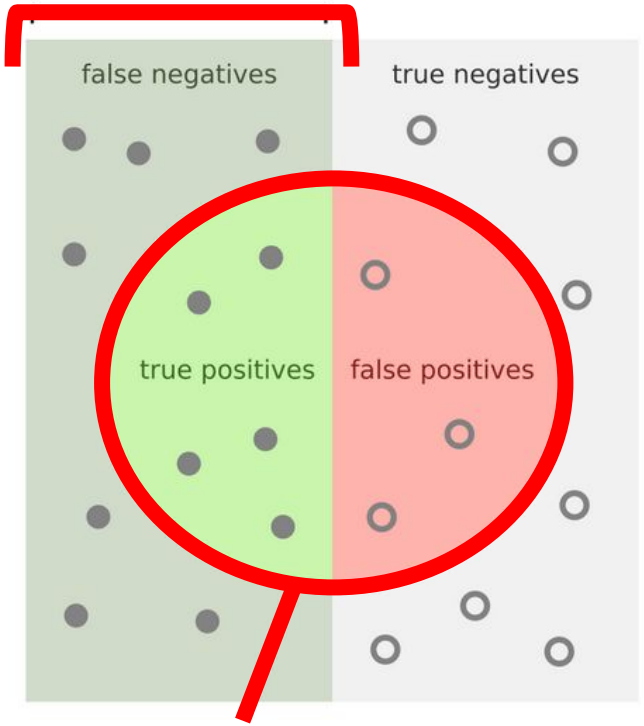
Image source: Walber @ en.wikipedia.org

Measuring model performance

Binary classification - Metrics

$$TP = |\{j \mid \hat{y}^{(j)} = 1 \ \wedge \ y^{(j)} = 1\}|$$
$$FP = |\{j \mid \hat{y}^{(j)} = 1 \ \wedge \ y^{(j)} = 0\}|$$
$$TN = |\{j \mid \hat{y}^{(j)} = 0 \ \wedge \ y^{(j)} = 0\}|$$
$$FN = |\{j \mid \hat{y}^{(j)} = 0 \ \wedge \ y^{(j)} = 1\}|$$

y = positive category (e.g., 1)



y^ = positive category (e.g., 1)

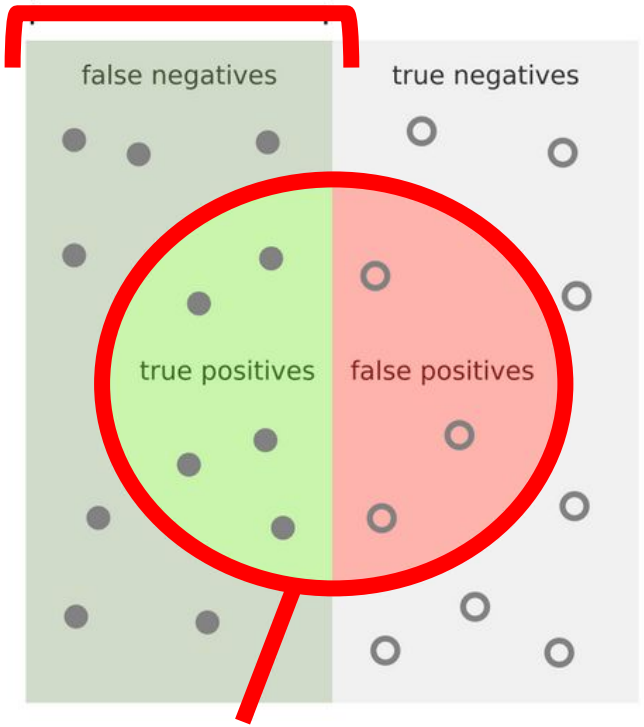
Image source: Walber @ en.wikipedia.org

Measuring model performance

Binary classification - Metrics

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\text{correct classifications}}{\text{all classifications}}$$

y = positive category (e.g., 1)



y^ = positive category (e.g., 1)

Image source: Walber @ en.wikipedia.org

Measuring model performance

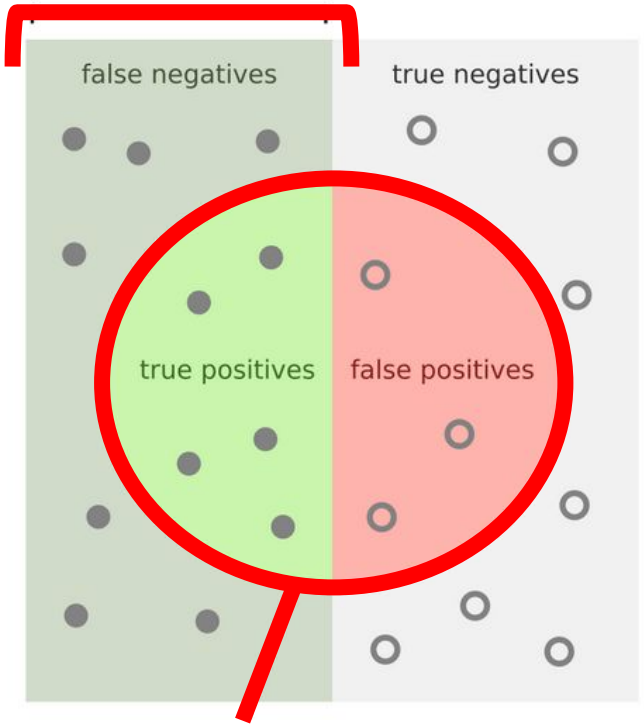
Binary classification - Metrics

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{\text{correct classifications}}{\text{all classifications}}$$

In what proportion did we correctly guess the true category of the data points?

y = positive category (e.g., 1)



y^ = positive category (e.g., 1)

Image source: Walber @ en.wikipedia.org

Measuring model performance

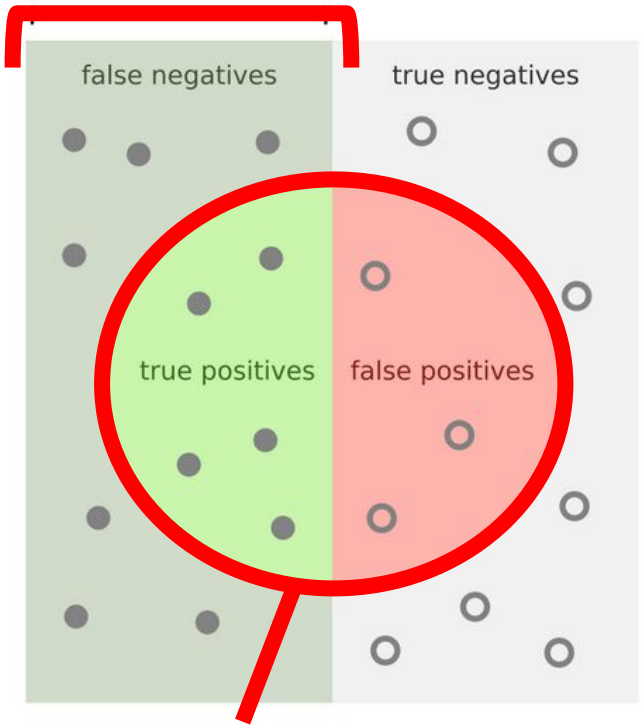
Binary classification - Metrics

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

y = positive category (e.g., 1)



y^ = positive category (e.g., 1)

Image source: Walber @ en.wikipedia.org

Measuring model performance

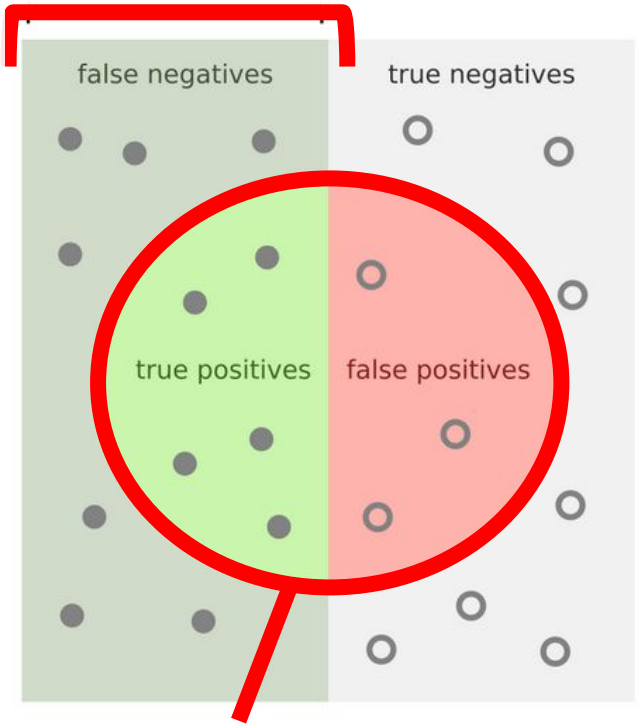
y = positive category (e.g., 1)

What ratio of the data points we classified as positive are actually in the positive category?

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

What ratio of the data points from the actual positive category did we identify as positive?



y^ = positive category (e.g., 1)

Measuring model performance

Binary classification - Metrics

Problem: These metrics are misleading when the category distribution is not uniform.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Measuring model performance

Binary classification - Metrics

Problem: These metrics are misleading when the category distribution is not uniform.

Example dataset:
98 dogs, 2 cats.

Let our estimate be “dog” regardless of the input!

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$ 98/100 = **98%**

$$\text{Precision} = \frac{TP}{TP + FP}$$ 98/100 = **98%**

$$\text{Recall} = \frac{TP}{TP + FN}$$ 98/98 = **100%**

Measuring model performance

Binary / multi-class classification - Balanced metrics

$$
\begin{aligned}
TP_k &= |\{j \mid \hat{y}^{(j)} = k \ \wedge \ y^{(j)} = k\}| \\
FP_k &= |\{j \mid \hat{y}^{(j)} = k \ \wedge \ y^{(j)} \neq k\}| \\
TN_k &= |\{j \mid \hat{y}^{(j)} \neq k \ \wedge \ y^{(j)} \neq k\}| \\
FN_k &= |\{j \mid \hat{y}^{(j)} \neq k \ \wedge \ y^{(j)} = k\}|
\end{aligned}
$$

$$
\text{Macro avg. precision} = \frac{1}{K} \sum_{k=1}^{K} \frac{TP_k}{TP_k + FP_k}
$$

$$
\text{Macro avg. recall} = \frac{1}{K} \sum_{k=1}^{K} \frac{TP_k}{TP_k + FN_k}
$$

Measuring model performance

Binary / multi-class classification - Balanced metrics

$$TP_k = |\{j \mid \hat{y}^{(j)} = k \ \wedge \ y^{(j)} = k\}|$$
$$FP_k = |\{j \mid \hat{y}^{(j)} = k \ \wedge \ y^{(j)} \neq k\}|$$
$$TN_k = |\{j \mid \hat{y}^{(j)} \neq k \ \wedge \ y^{(j)} \neq k\}|$$
$$FN_k = |\{j \mid \hat{y}^{(j)} \neq k \ \wedge \ y^{(j)} = k\}|$$

Category #k is **"positive"**, All other categories together are **"negative"**

Another name: balanced accuracy

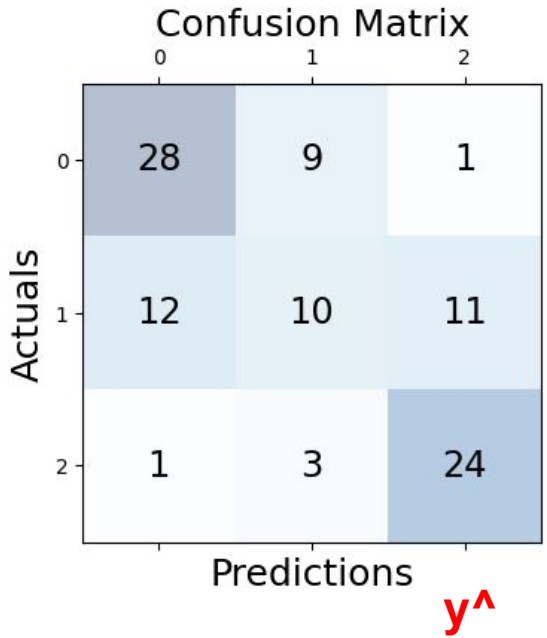
$$\text{Macro avg. precision} = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k + FP_k}$$

$$\text{Macro avg. recall} = \frac{1}{K}\sum_{k=1}^{K}\frac{TP_k}{TP_k + FN_k}$$

Averaging over k = 1, …, K (K is the number of categories)

Measuring model performance

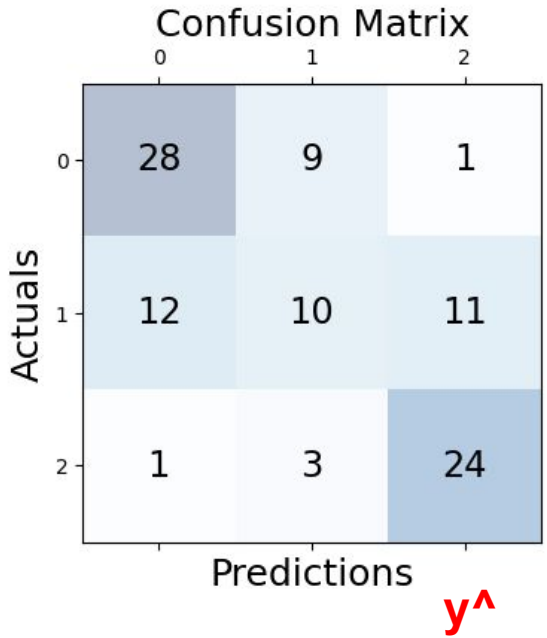
Binary / multi-class classification - Confusion matrix



$$C_{k,\hat{k}} = |\{j \mid y^{(j)} = k \ \wedge \ \hat{y}^{(j)} = \hat{k}\}|$$

Measuring model performance

Binary / multi-class classification - Confusion matrix



y

y^

$$C_{k,\hat{k}} = |\{j \mid y^{(j)} = k \ \wedge \ \hat{y}^{(j)} = \hat{k}\}|$$

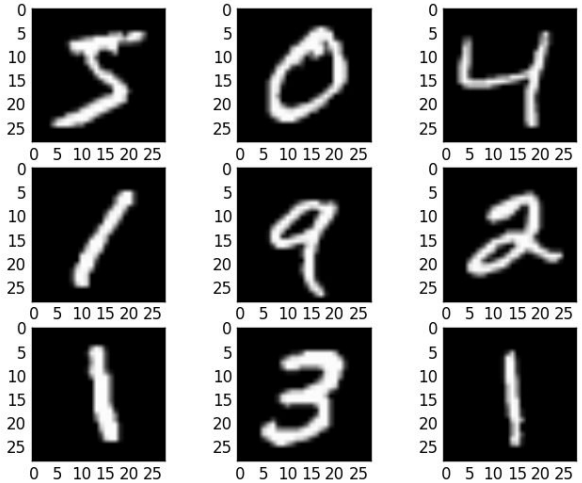
This is a diagonal matrix when our estimates are perfect…

Applying MLP for handwriting recognition

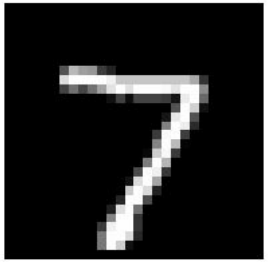
MNIST dataset

- Handwritten digits
- 28 × 28 sized grayscale images
- 10 categories (digits: 0 .. 9)
- 60k training examples,
  10k test examples

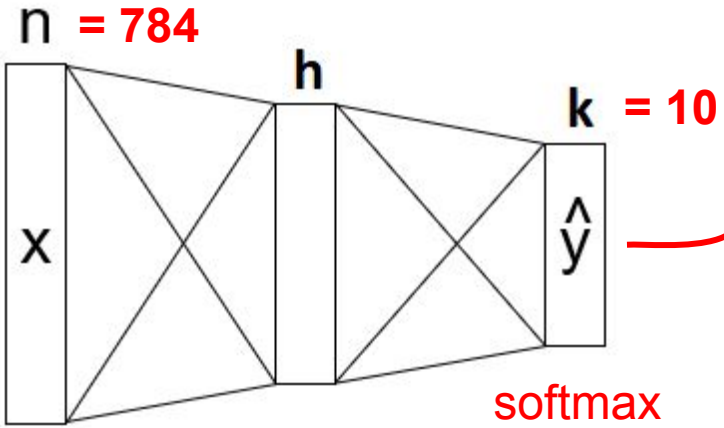
784 input variables: The brightness of each pixel is a variable.

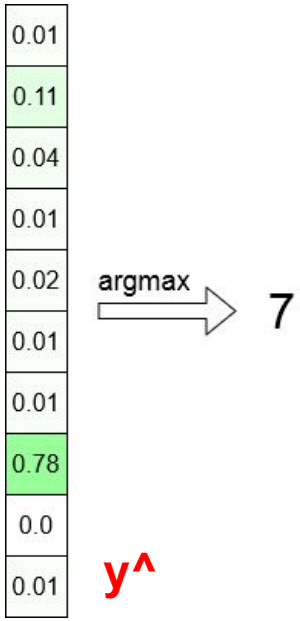


Applying MLP for handwriting recognition

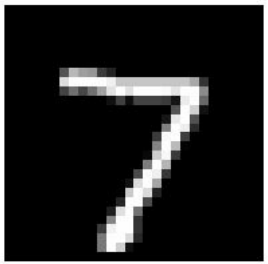


x





Applying MLP for handwriting recognition



x

Tuning the parameters

J (categorical crossentropy)

y

0
0
0
0
0
0
0
1
0
0

n = 784

h

k = 10

x

ŷ

softmax

0.01
0.11
0.04
0.01
0.02
0.01
0.01
0.78
0.0
0.01

argmax → 7

y^

How well does an MLP perform on MNIST?

The performance of different methods on the MNIST dataset:
http://yann.lecun.com/exdb/mnist/

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN) (log.reg)	none	12.0	LeCun et al. 1998
• • •			
Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998

How well does an MLP perform on MNIST?

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
...			
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
...			

How many parameters are there in the network?

How well does an MLP perform on MNIST?

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
• • •			
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
• • •			

How many parameters are there?

First layer: 784 x 800 + 800

Second layer: 800 × 10 + 10

A total of 636 010 parameters (!)

MLP - MNIST

Previously:

E.g., logistic regression for estimating cholesterol levels

→ 3 input variables, 4 parameters, several hundred data points

Now:

MLP with a single hidden layer for the classification of handwritten digits

→ 784 input variables, 636k parameters, 60k data points

MLP - MNIST

Previously:

E.g., logistic regression for estimating cholesterol levels

→ 3 input variables, 4 parameters, several hundred data points

<

Now:

MLP with a single hidden layer for the classification of handwritten digits

→ 784 input variables, 636k parameters, 60k data points

>

What can happen when we have too many parameters?

MLP - MNIST

Previously:

E.g., logistic regression for estimating cholesterol levels

→ 3 input variables, 4 parameters, several hundred data points

<

Now:

MLP with a single hidden layer for the classification of handwritten digits

→ 784 input variables, 636k parameters, 60k data points

>

What can happen when we have too many parameters?

Overfitting: An overly complex model is capable of memorizing the characteristics of individual data points → Loses its ability to generalize

How to deal with overfitting?

How can we avoid overfitting?

How to deal with overfitting?

How can we avoid overfitting?

Previously:

- Use a simpler model (e.g., fewer parameters)!
- Regularization methods (e.g., L2 regularization)
- Early stopping
- Obtain more training data!

How to deal with overfitting?

How can we avoid overfitting?

“Obtain more training data!”

- Unfortunately, this is usually not possible:
  Labeling data can be **expensive**, requires human supervision.

How to deal with overfitting?

How can we avoid overfitting?

“Obtain more training data!”

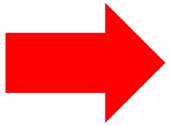
- Unfortunately, this is usually not possible:
  Labeling data can be **expensive**, requires human supervision.

Let’s try to create new training examples using the existing ones!

Image source: https://github.com/aleju/imgaug

Avoiding overfitting - Data augmentation

Data augmentation



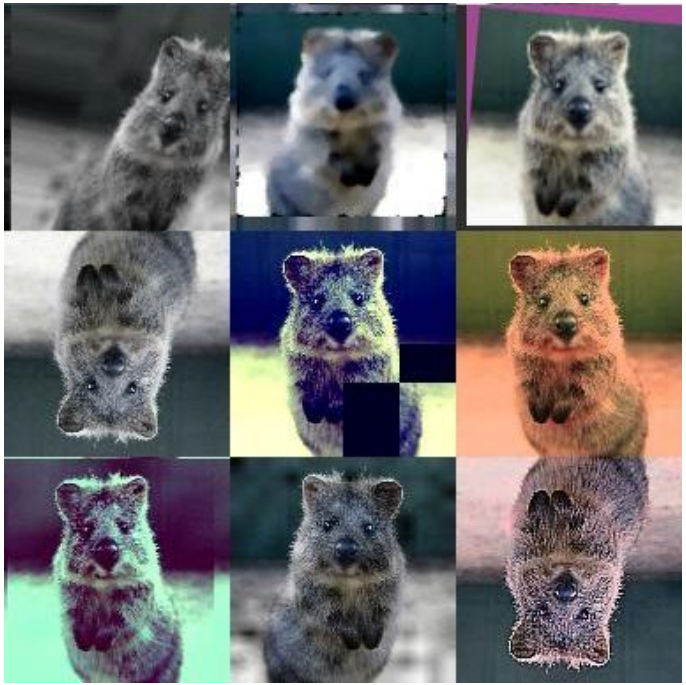
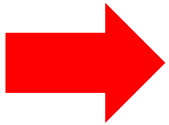
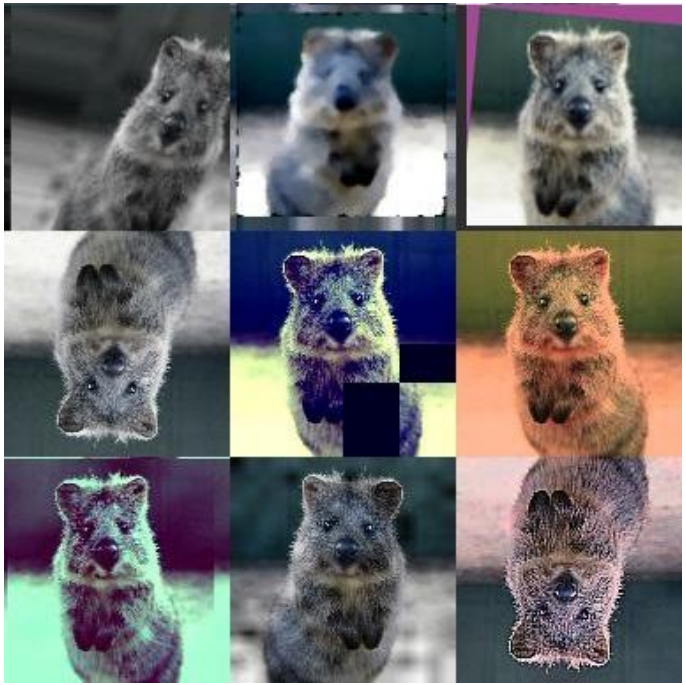


Image source: https://github.com/aleju/imgaug

Avoiding overfitting - Data augmentation

Data augmentation



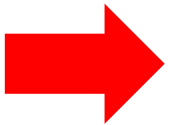


A hamster (?) - whether rotated, shifted, enlarged, or recolored - remains a hamster.

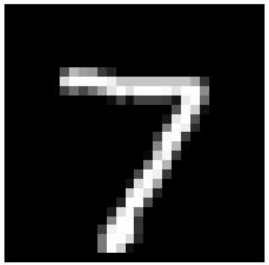
Image source: https://github.com/aleju/imgaug

Avoiding overfitting - Data augmentation

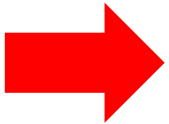
hamster

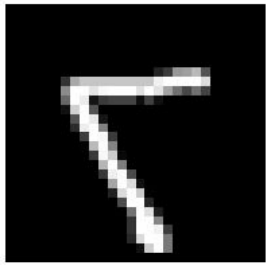


hamster



“7”





???

Not every type of transformation can be applied to every type of data...

Avoiding overfitting - Data augmentation

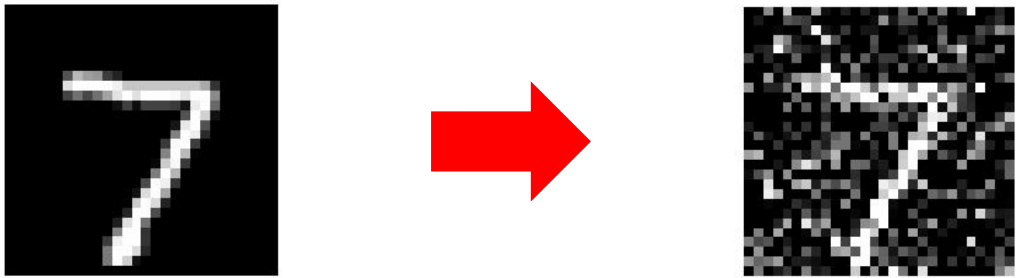
Data augmentation techniques for various types of data

- **MNIST:** translation, slight rotation, brightness adjustment, noise, …
- **Photos:** translation, rotation, flipping, distortion, changes in color and brightness, noise, masking of certain parts, …
- **Sounds:** stretching, frequency changes, noise, ...

Avoiding overfitting - Data augmentation

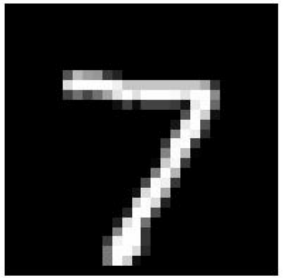
Adding noise (A type of data augmentation)

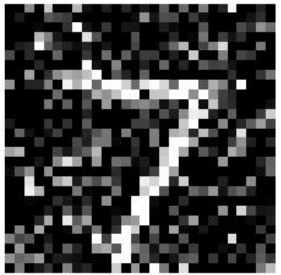
Adding Gaussian noise to the input can reduce overfitting.



Avoiding overfitting - Data augmentation

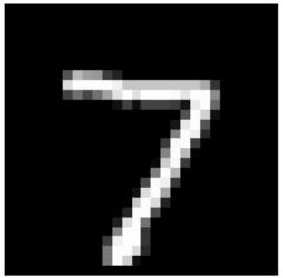
How can adding noise help reduce overfitting?

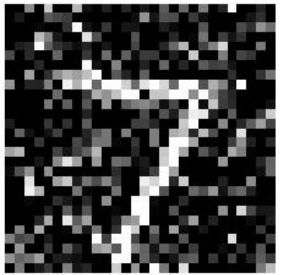




Avoiding overfitting - Data augmentation

How can adding noise help reduce overfitting?





A neural network consisting of fully connected layers easily "memorizes" specific examples from the training set:
For example: "if the sum of the pixel values in the image falls exactly between 6725.81 and 6725.83, then the image contains the digit 7".

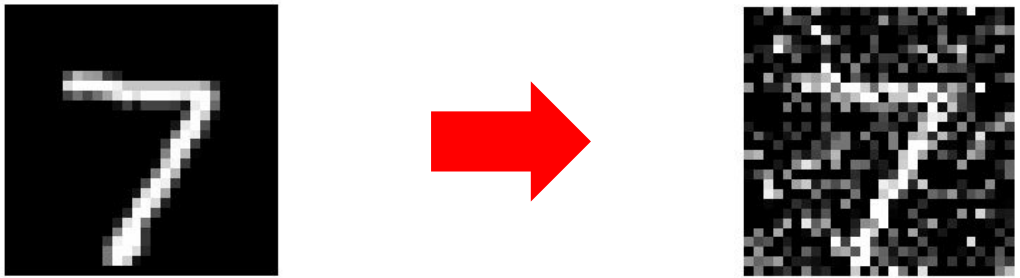
This is obviously not useful knowledge.

Adding noise prevents the network from "memorizing" precise details, thereby mitigating overfitting.

Avoiding overfitting - Data augmentation

Adding noise (A type of data augmentation)

Adding Gaussian noise to the input can reduce overfitting.



Do not add noise during validation/evaluation/prediction!

Image source: https://github.com/aleju/imgaug

Avoiding overfitting - Dropout

Dropout (A type of noise)

We randomly set some of the input variables to zero.

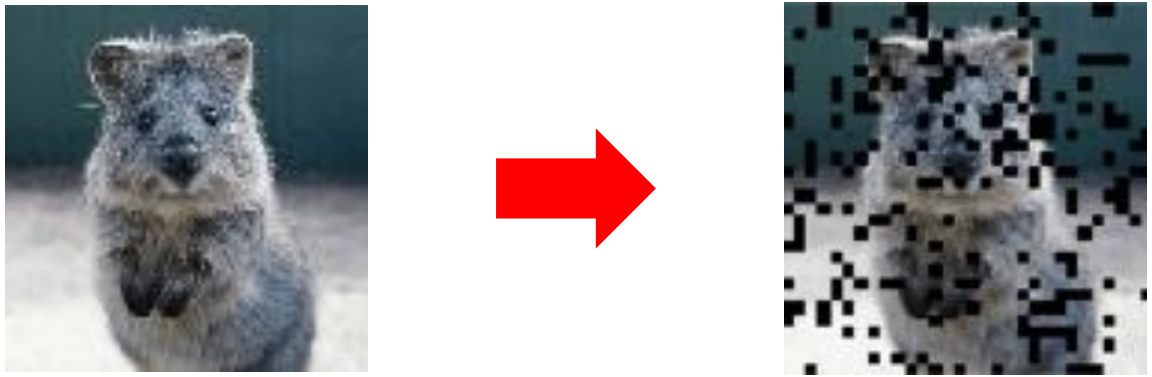
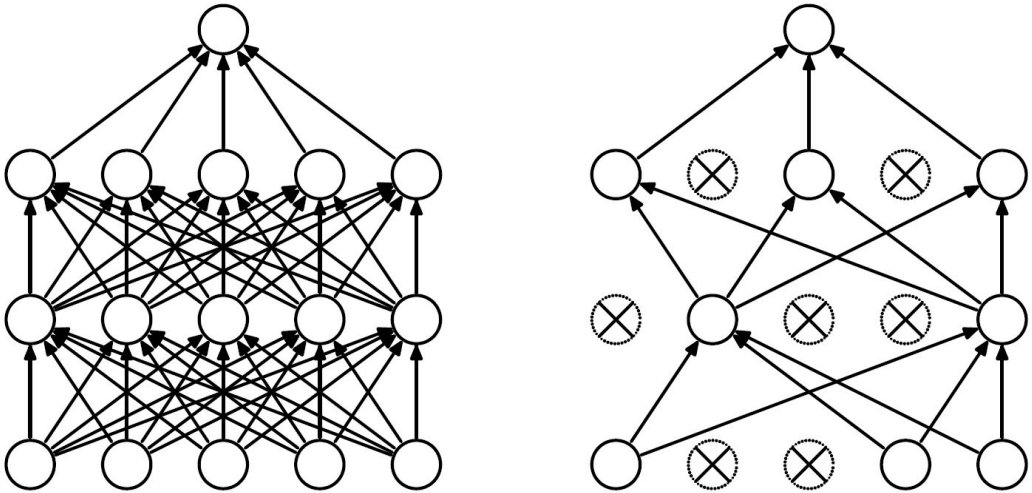


Image source: Srivastava et al., 2014

Avoiding overfitting - Dropout

Dropout: Usually applied on the hidden representations
(the outputs of intermediate layers)

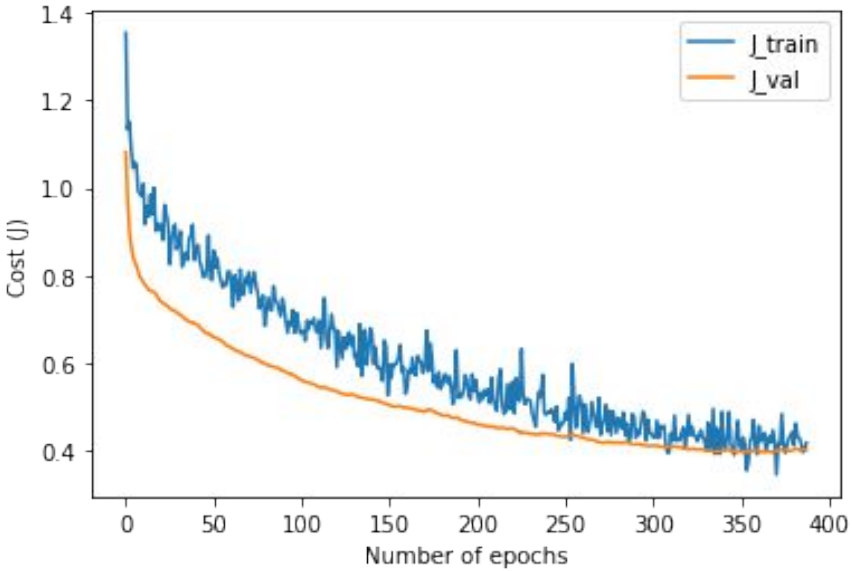


(a) Standard Neural Net

(b) After applying dropout.

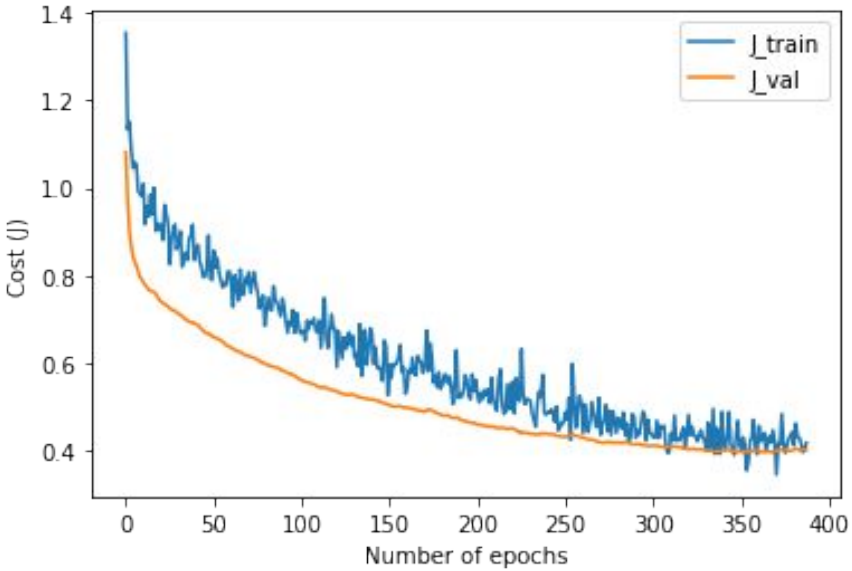
Avoiding overfitting - Dropout

When applying dropout, the training loss can appear higher than its true value.



Avoiding overfitting - Dropout

When applying dropout, the training loss can appear higher than its true value.

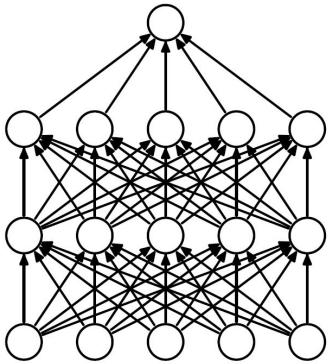


Since we only use noise during training and not during validation or evaluation,
the loss measured on the training set may appear higher than it is.

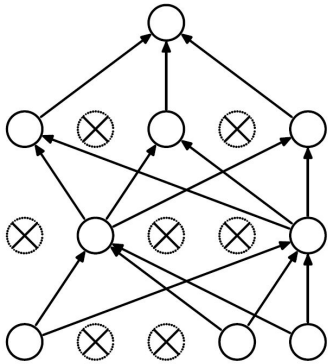
Image source: Srivastava et al., 2014

Avoiding overfitting - Dropout

- Using dropout **slows down training.**
- However, in many cases, its use can lead to **significant performance improvements.**



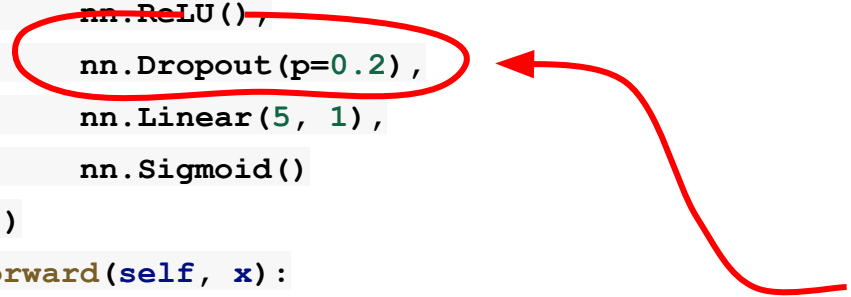
(a) Standard Neural Net



(b) After applying dropout.

Avoiding overfitting - Dropout, PyTorch

```
class MyTwoLayerMLP(nn.Module):
  def __init__(self, input_dim):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 5),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(5, 1),
            nn.Sigmoid()
        )
  def forward(self, x):
      return self.layers(x)
```



Dropout randomly sets the output variables of the first layer **to zero with a chance of 20% each**.

Avoiding overfitting - Dropout, PyTorch

```
class MyTwoLayerMLP(nn.Module):
  def __init__(self, input_dim):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 5),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.Linear(5, 1),
            nn.Sigmoid()
        )
  def forward(self, x):
      return self.layers(x)
```

```
model = MyTwoLayerMLP(input_dim=10)

model.train()

...
```

Dropout **enabled.**

```
model.eval()

...
```

Dropout **disabled.**

Avoiding overfitting - Adding noise, PyTorch

```
class MyTwoLayerMLP(nn.Module):
  def __init__(self, input_dim):
        super().__init__()
        self.layer1 = nn.Linear(input_dim, 5)
        self.layer2 = nn.Linear(5, 1)
        self.GAUSSIAN_SIGMA = 0.5

  def forward(self, x):
      x = self.layer1(x)
      x = torch.nn.functional.relu(x)
      if self.training:
        x = x + torch.randn(*x.shape, dtype=x.dtype) * self.GAUSSIAN_SIGMA
      x = self.layer2(x)
      x = torch.nn.functional.sigmoid(x)
      return x
```

Gaussian additive noise
(mu=0, sigma=0.5)
on the output of the first layer.

Avoiding overfitting - Adding noise, PyTorch

```
class MyTwoLayerMLP(nn.Module):
  def __init__(self, input_dim):
        super().__init__()
        self.layer1 = nn.Linear(input_dim, 5)
        self.layer2 = nn.Linear(5, 1)
        self.GAUSSIAN_SIGMA = 0.5

  def forward(self, x):
      x = self.layer1(x)
      x = torch.nn.functional.relu(x)
      if self.training:
        x = x + torch.randn(*x.shape, dtype=x.dtype) * self.GAUSSIAN_SIGMA
      x = self.layer2(x)
      x = torch.nn.functional.sigmoid(x)
      return x
```

```
model = MyTwoLayerMLP(input_dim=10)

model.train()

...    model.training == True

model.eval()

...    model.training == False
```

How well does an MLP perform on MNIST?

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
...			
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
...			

How well does an MLP perform on MNIST?

Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
• • •			
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
• • •			
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010

On the MNIST dataset, a sufficiently large **MLP achieves superhuman results** when combined **with appropriate augmentation and regularization methods.**

Application of an MLP for image classification

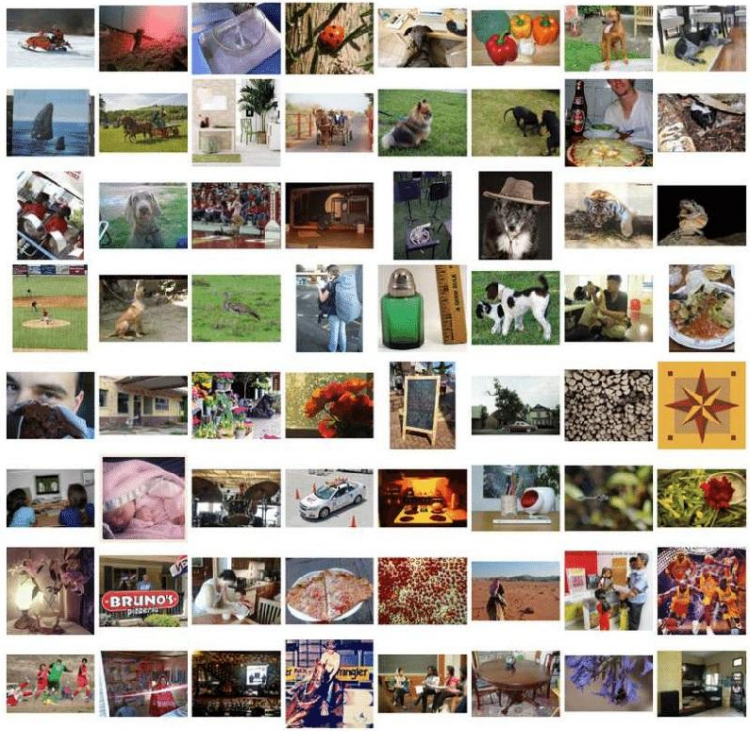
The MLP performs well in handwritten digit classification.

What about high resolution images?

ImageNet dataset

ImageNet - Photos of various object categories.

- Over 1,000 categories, such as:
  - swamp turtle
  - waffle iron
  - Norfolk terrier
  - viaduct
- Around 1 megapixel, color
  → **approx. 3 million input variables**



ImageNet dataset

The accuracy of various methods on the test set of ImageNet (Guess the right category out of 1000 categories in 5 attempts)**:**

Random guessing: **0.5%**

Human (expert): **94.9%**

“FixResNeXt-101 32x48d” (2019): **98%**

ImageNet dataset

The accuracy of various methods on the test set of ImageNet (Guess the right category out of 1000 categories in 5 attempts)**:**

Random guessing: **0.5%**

Human (expert): **94.9%**

“FixResNeXt-101 32x48d” (2019): **98%**

MLP: **~0.5% :(**

Application of an MLP for image classification

Why is an MLP not suitable for the classification of high resolution images?

Application of an MLP for image classification

Why is an MLP not suitable for the classification of high resolution images?

The network must learn to recognize each pattern at every possible location in the image.




Application of an MLP for image classification

Why is an MLP not suitable for the classification of high resolution images?

The network must learn to recognize each pattern at every possible location in the image.

Until now: e.g., estimating the cholesterol level of a patient -
$\mathbf{x}_1$ is the weight of the patient, $\mathbf{x}_2$ is the age of the patient, etc.,
→ **All input variables have a fixed meaning.**

Application of an MLP for image classification

Why is an MLP not suitable for the classification of high resolution images?

The network must learn to recognize each pattern at every possible location in the image.

In case of images: Any input variable can contain any detail of any object. Each weight/neuron would have to learn a vast amount of information!

Application of an MLP for image classification

Why is an MLP not suitable for the classification of high resolution images?

The network must learn to recognize each pattern at every possible location in the image.

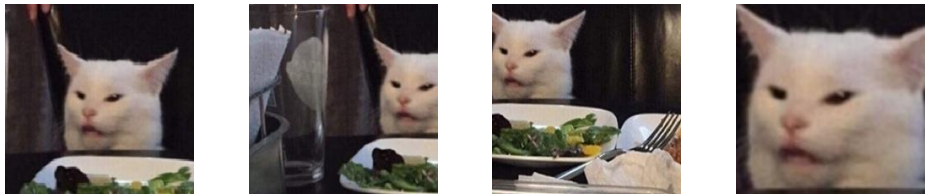
In case of images: Any input variable can contain any detail of any object.
Each weight/neuron would have to learn a vast amount of information!

To do this, the MLP must see each type of object
in every possible position, size and orientation as input,
and it must be able to learn them.

Detecting image patterns

How should we approach this problem?

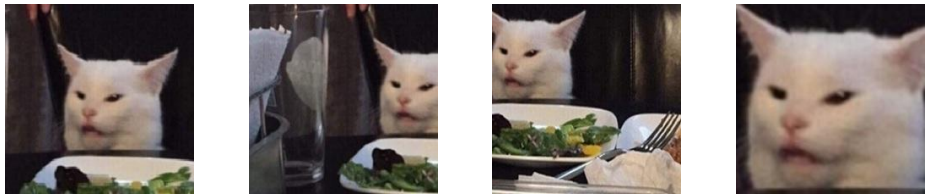
- Once we learn to recognize a pattern,
  we should be able to identify it anywhere in the image.
  → **Translation invariance / equivariance**



Detecting image patterns

How should we approach this problem?

- **Translation invariance / equivariance**
- The larger the apparent size of the object we are trying to recognize, the more variations in its appearance there may be in the image.
  → **Let's learn to recognize small patterns!**



Detecting image patterns

How should we approach this problem?

- Translation invariance / equivariance
- Learn to recognize small patterns

Idea: Let’s slide a window to every possible location in the image and learn to recognize the important patterns that may appear in the window!

Detecting image patterns

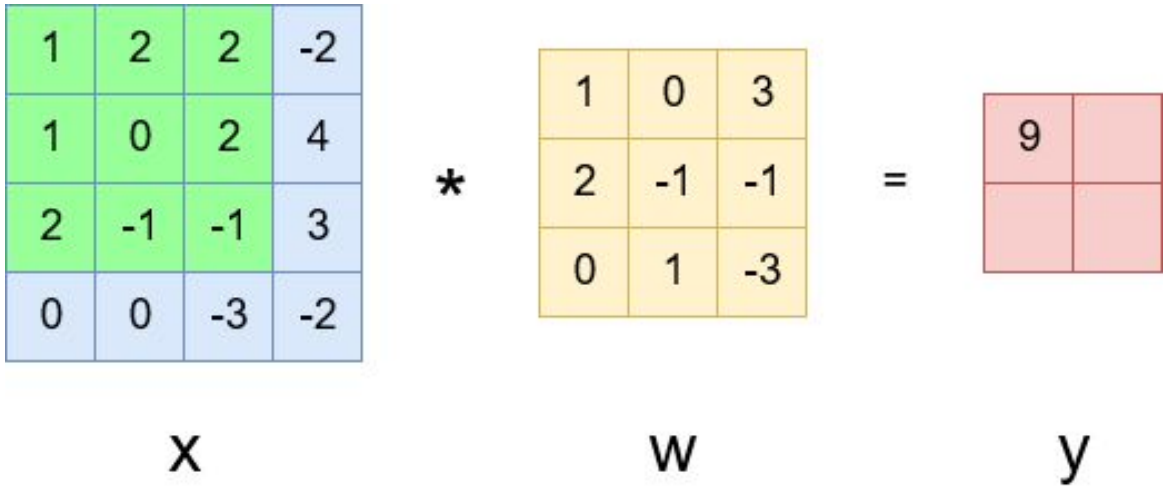
How should we approach this problem?

- Translation invariance / equivariance
- Learn to recognize small patterns

Idea: Let's slide a window to every possible location in the image and learn to recognize the important patterns that may appear in the window!

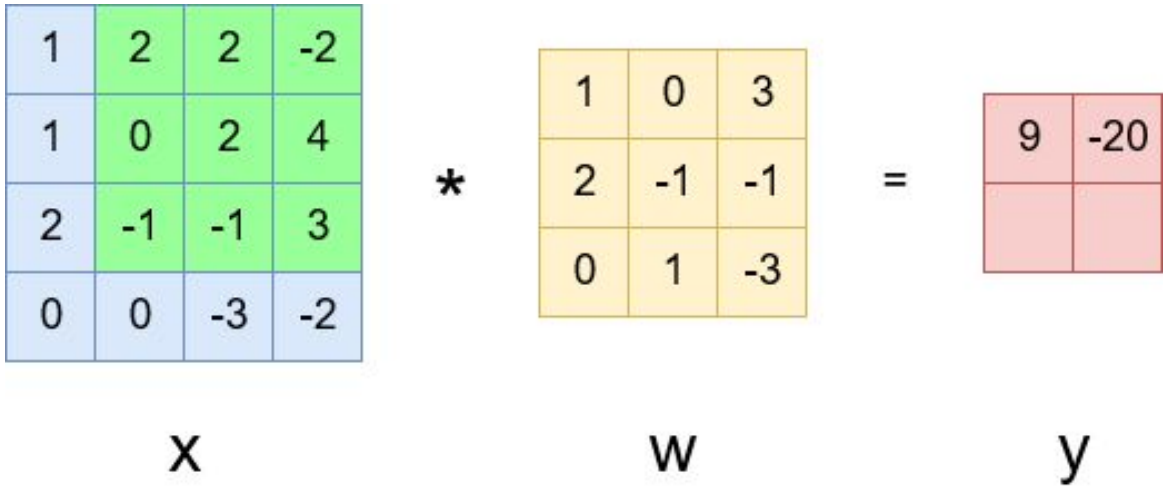
→ **Convolutional layer**

2D Convolution - Discrete case



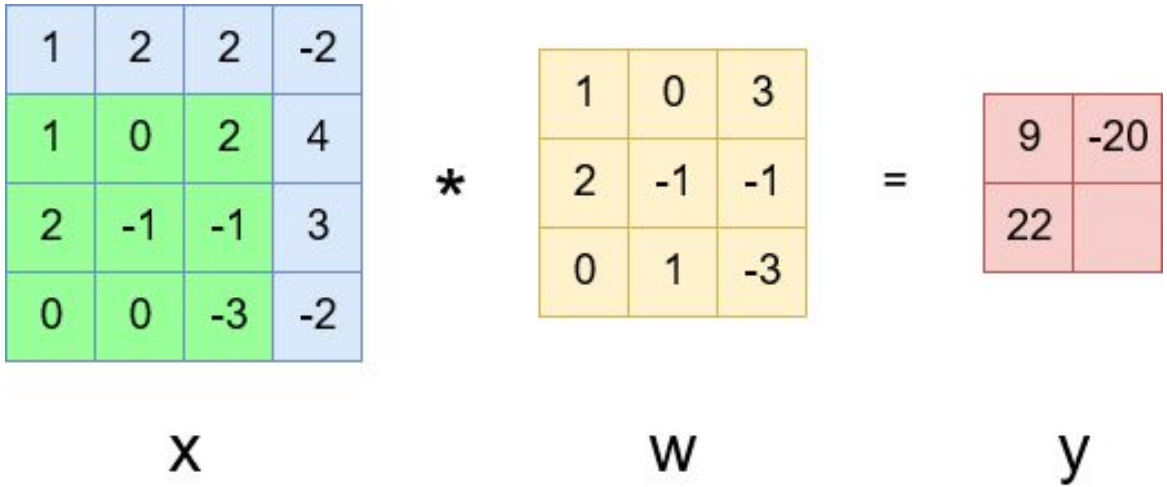
$$1 \cdot 1 + 2 \cdot 0 + 2 \cdot 3 + 1 \cdot 2 + 0 \cdot (-1) + 2 \cdot (-1) + 2 \cdot 0 + (-1) \cdot 1 + (-1) \cdot (-3) = 9$$

2D Convolution - Discrete case



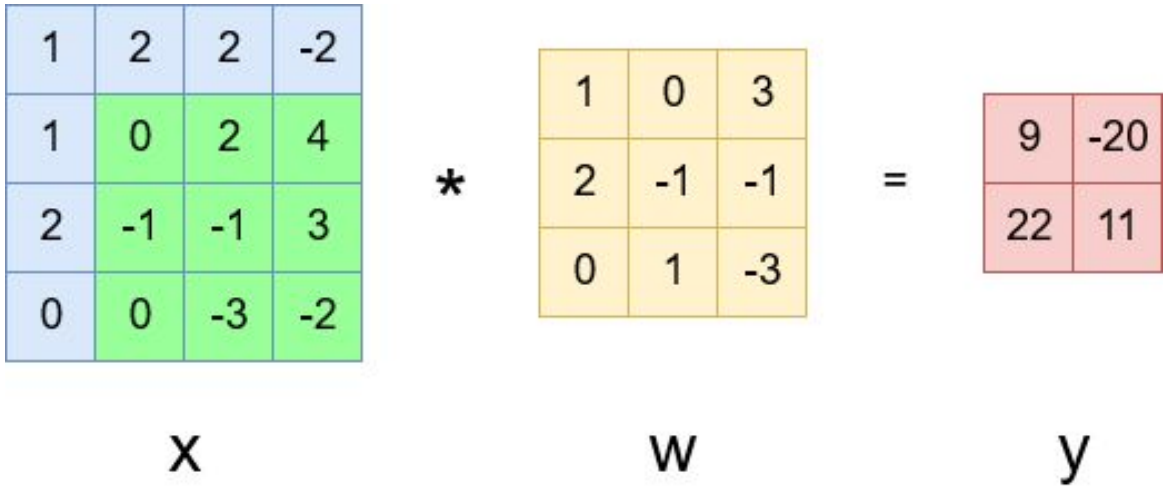
$$2 \cdot 1 + 2 \cdot 0 + (-2) \cdot 3 + 0 \cdot 2 + 2 \cdot (-1) + 4 \cdot (-1) + (-1) \cdot 0 + (-1) \cdot 1 + 3 \cdot (-3) = -20$$

2D Convolution - Discrete case



$$1 \cdot 1 + 0 \cdot 0 + 2 \cdot 3 + 2 \cdot 2 + (-1) \cdot (-1) + (-1) \cdot (-1) + 0 \cdot 0 + 0 \cdot 1 + (-3) \cdot (-3) = 22$$

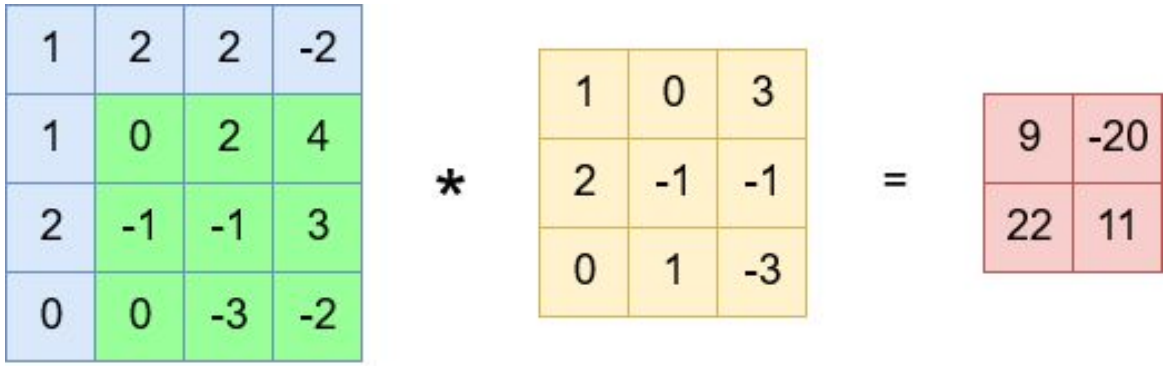
2D Convolution - Discrete case



$$0 \cdot 1 + 2 \cdot 0 + 4 \cdot 3 + (-1) \cdot 2 + (-1) \cdot (-1) + 3 \cdot (-1) + 0 \cdot 0 + (-3) \cdot 1 + (-2) \cdot (-3) = 11$$

2D Convolution - Discrete case

The output image (**heatmap**) is as large as many different locations we can place the filter.



x w y

We place the **w** filter (kernel) to every possible location of the **x** image and take their scalar product to generate a single output pixel value.

$$0 \cdot 1 + 2 \cdot 0 + 4 \cdot 3 + (-1) \cdot 2 + (-1) \cdot (-1) + 3 \cdot (-1) + 0 \cdot 0 + (-3) \cdot 1 + (-2) \cdot (-3) = 11$$

Filter **w** contains the image pattern to recognize. We learn this by gradient descent.

2D Convolution - Discrete case

Input: $x \in \mathbb{R}^{I \times J}$

Filter (kernel)**:** $w \in \mathbb{R}^{U \times V}$

Output (heatmap)**:** $\hat{y} \in \mathbb{R}^{(I-U+1) \times (J-V+1)}$

$$\hat{y}[i,j] = (x * w)[i,j] = \langle x[i..i+U, j..j+V] \, , \, w \rangle =$$

$$= \sum_{u=1}^{U} \sum_{v=1}^{V} x[i+u, j+v] \cdot w[u,v]$$

Image source: http://aishack.in/tutorials/image-convolution-examples/

Convolution in image post-processing

A filter for **Gaussian smoothing**

$$\frac{1}{1068} \cdot$$

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Image source: http://aishack.in/tutorials/image-convolution-examples/

Convolution in image post-processing

A filter for **detecting horizontal edges**

-1	-1	-1
2	2	2
-1	-1	-1

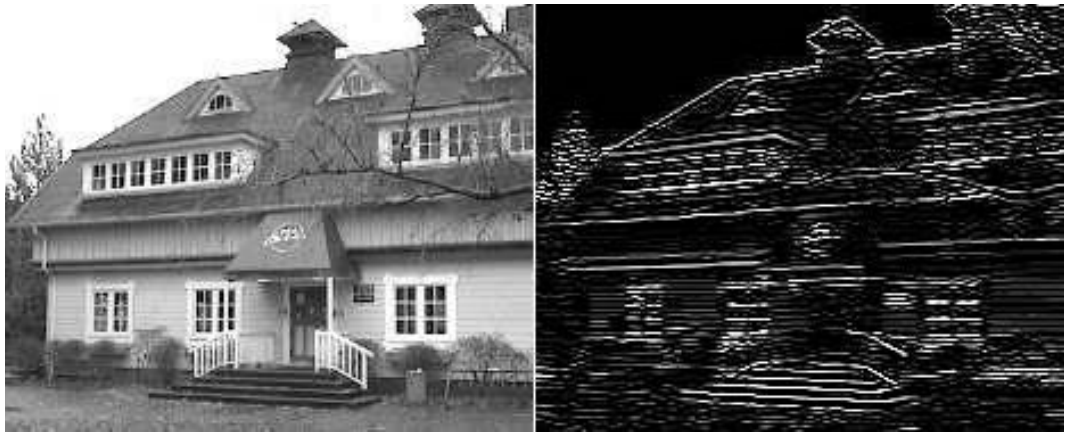


Image source: http://aishack.in/tutorials/image-convolution-examples/

Convolution in image post-processing

A filter for any **edge detection**

-1	-1	-1
-1	8	-1
-1	-1	-1

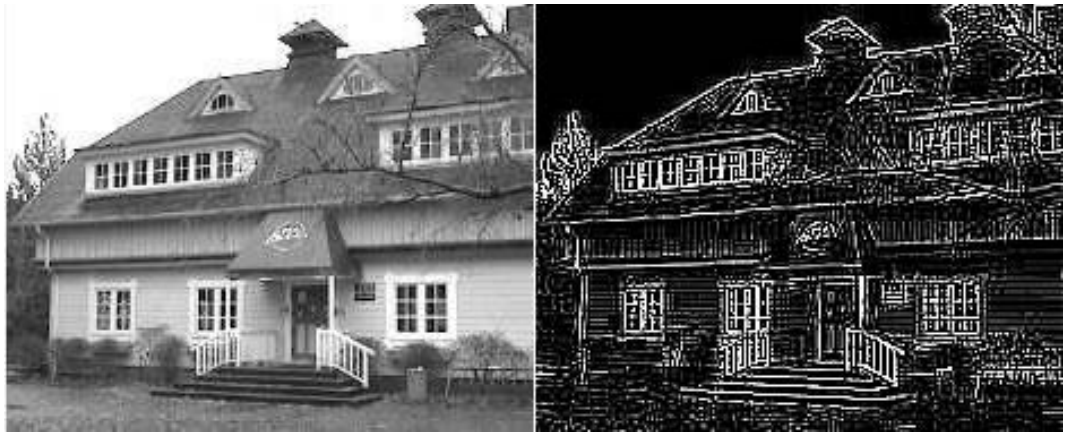


Image source: http://aishack.in/tutorials/image-convolution-examples/

Convolution in image post-processing

A filter for any **edge detection**

-1	-1	-1
-1	8	-1
-1	-1	-1

Image editing software (e.g., Photoshop), contain such **constant, pre-defined filters.**

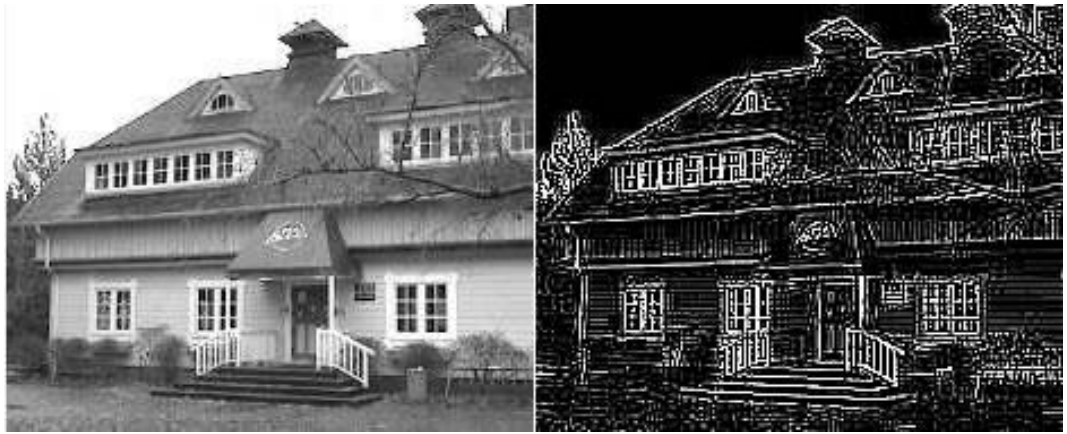


Image source: https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution for image pattern recognition

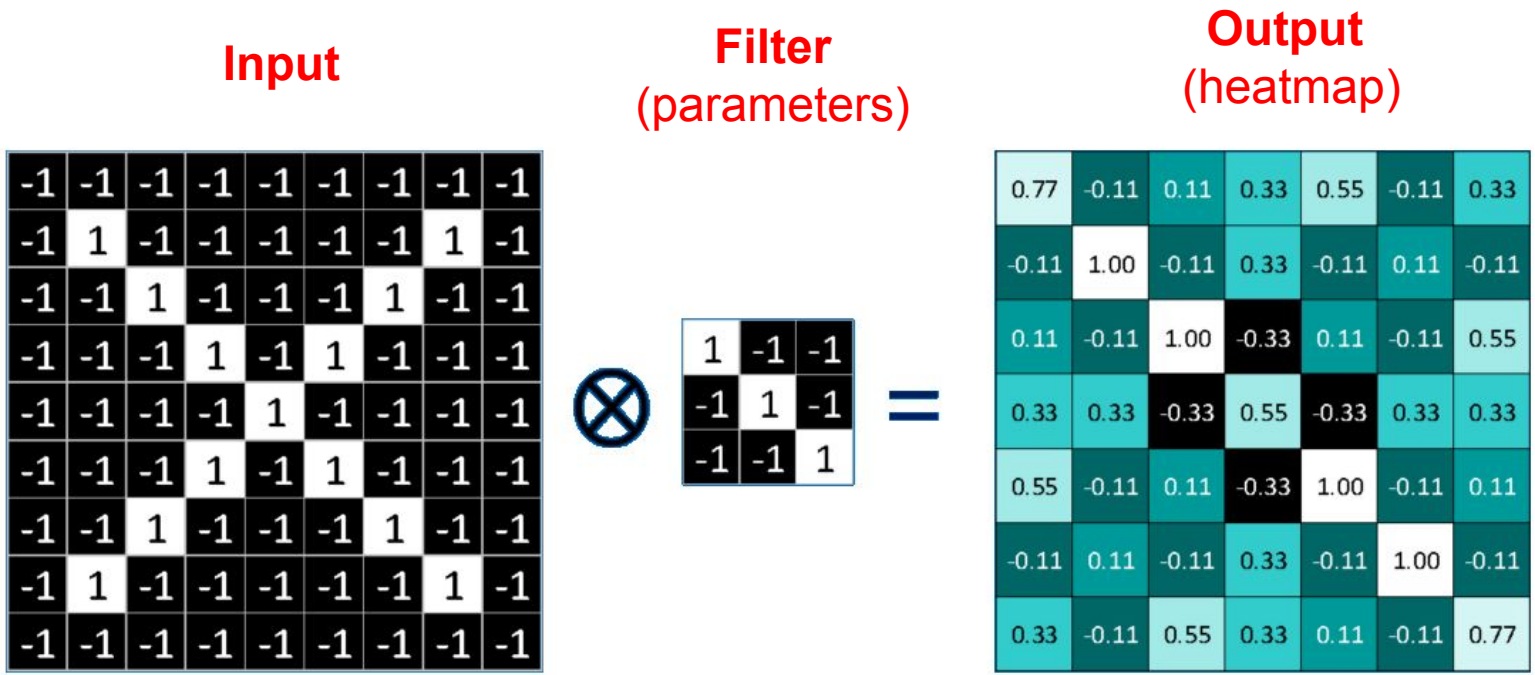


Image source: https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution for image pattern recognition

Example for a good match: High values at the corresponding heatmap pixels.

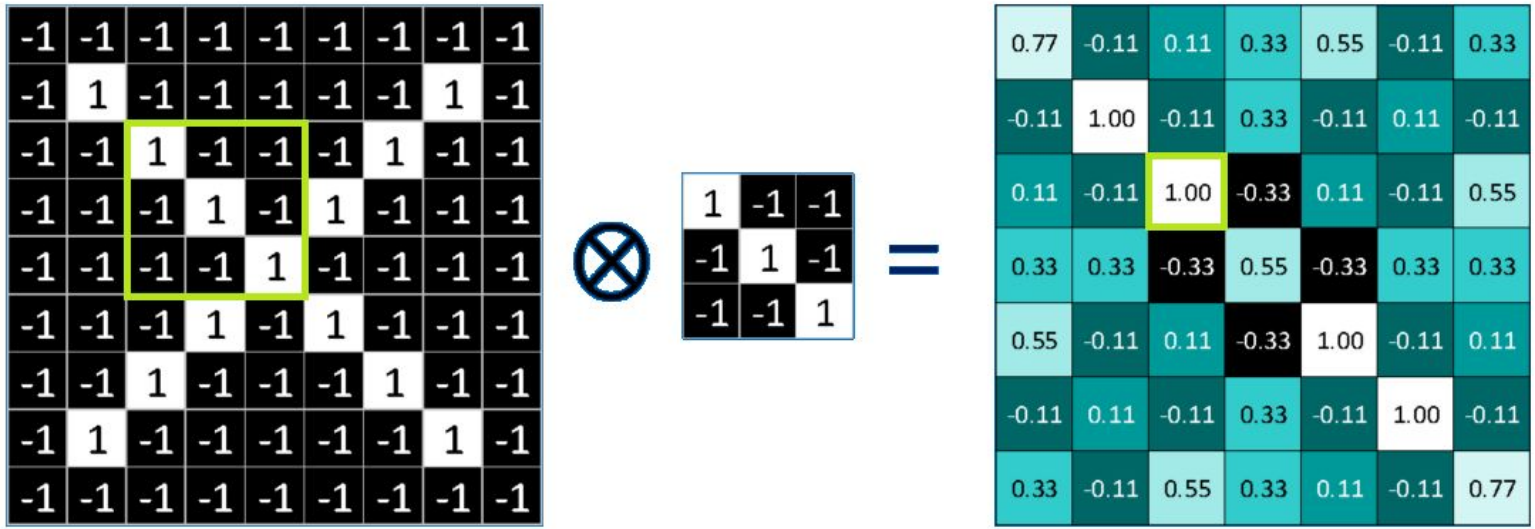


Image source: https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution for image pattern recognition

Example for a bad match: Low values at the corresponding heatmap pixels.

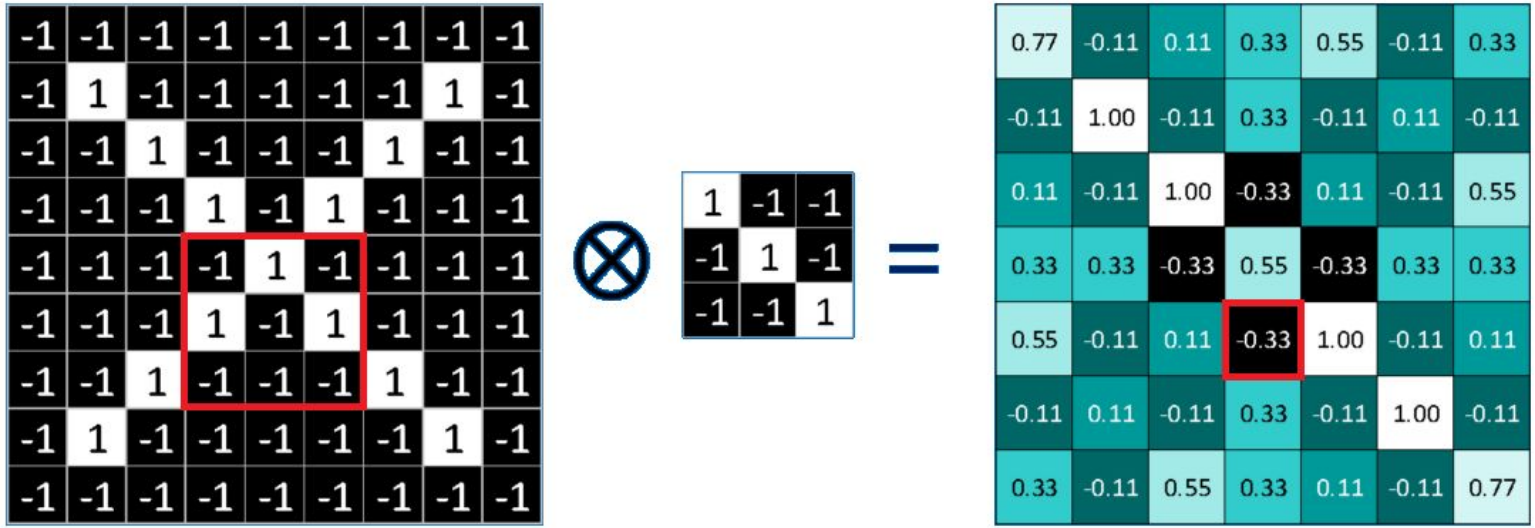
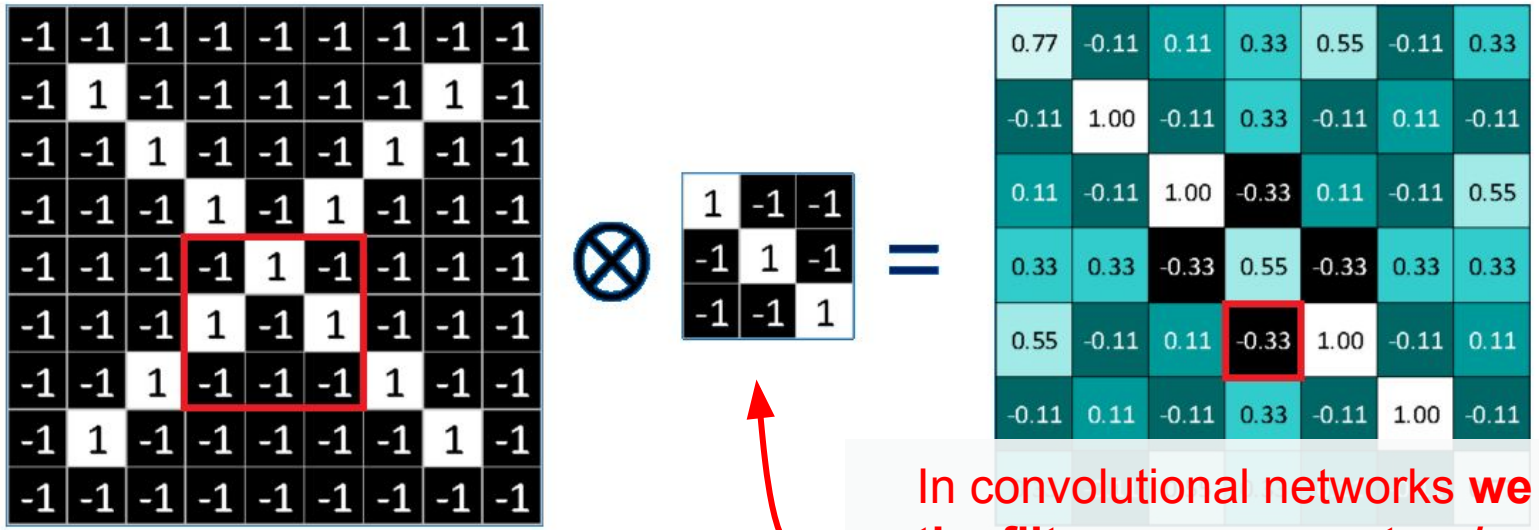


Image source: https://brohrer.github.io/how_convolutional_neural_networks_work.html

Convolution for image pattern recognition

Example for a bad match: Low values at the corresponding heatmap pixels.



In convolutional networks **we learn the filters as parameters/weights**

Convolutional layer

Convolutional layer

- **We learn the filters** instead of using fixed, pre-defined ones.
- The **values in the filters** and the corresponding **biases** are the **parameters.**
- An activation function is needed (g), e.g., ReLU.

$$\hat{y}[i,j] = g\Big(\sum_{u=1}^{U} \sum_{v=1}^{V} x[i+u, j+v] \cdot w[u,v] + b \Big)$$

$$x \in \mathbb{R}^{I \times J}$$

$$w \in \mathbb{R}^{U \times V}$$

$$b \in \mathbb{R}$$

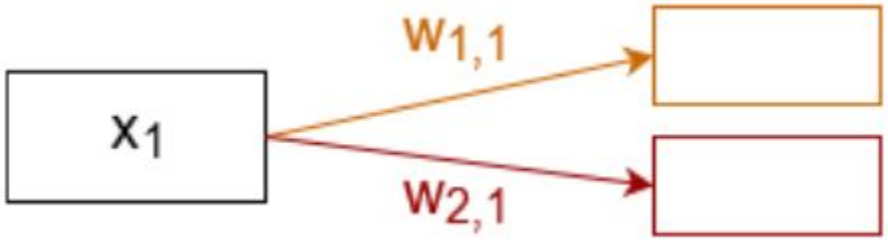
$$\hat{y} \in \mathbb{R}^{(I-U+1) \times (J-V+1)}$$

Indexing and sizes for the 2D case here, but can be defined for non-2D cases…

Convolutional layer - Channels

- A single filter will recognize a single pattern.
- We would like to recognize multiple patterns simultaneously.

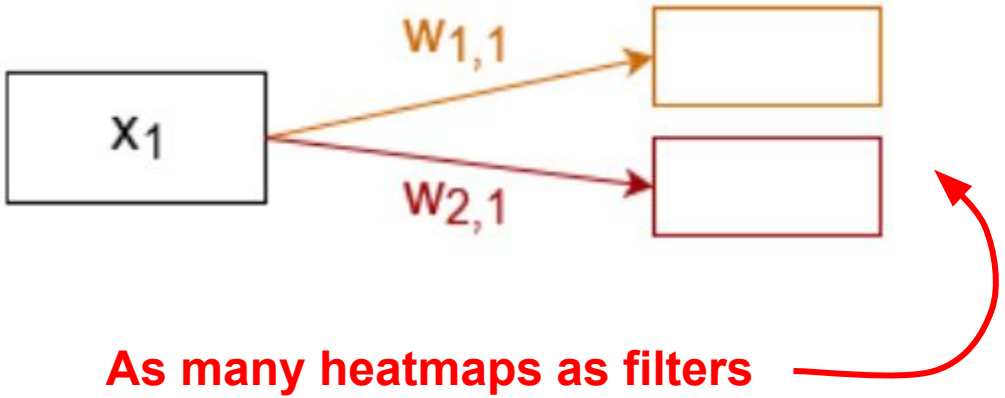
→ **Learning multiple filters** and bias parameters **at the same time.**



Convolutional layer - Channels

- A single filter will recognize a single pattern.
- We would like to recognize multiple patterns simultaneously.

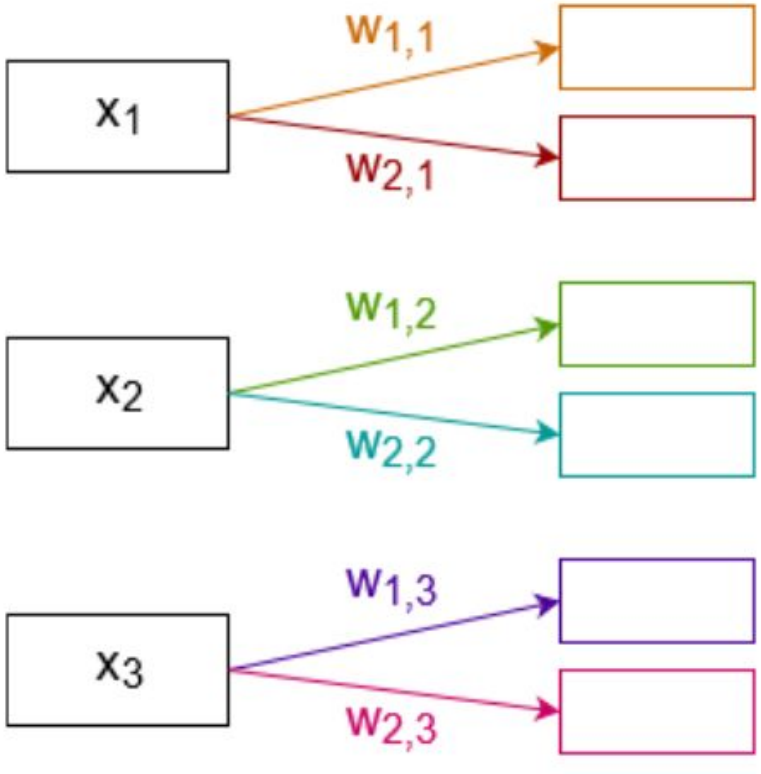
→ **Learning multiple filters** and bias parameters **at the same time.**



Convolutional layer - Channels

The input may have multiple channels:

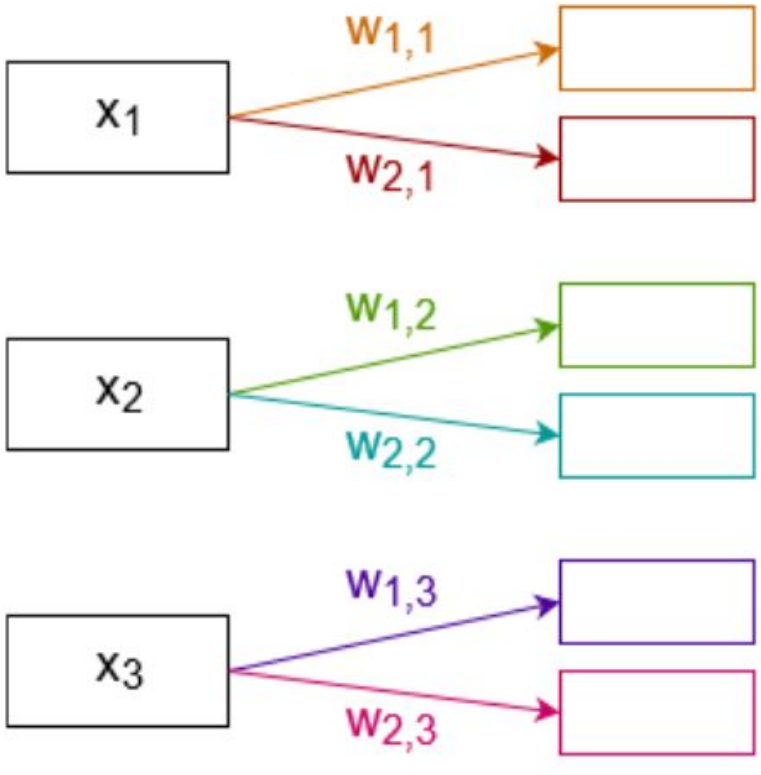
- Colored image: 3 channels (e.g., RGB).
- **Stacking** convolutional layers: We can apply a convolutional layer on the output of another one.



Convolutional layer - Channels

The input may have multiple channels:

- Colored image: 3 channels (e.g., RGB).
- **Stacking** convolutional layers: We can apply a convolutional layer on the output of another one.

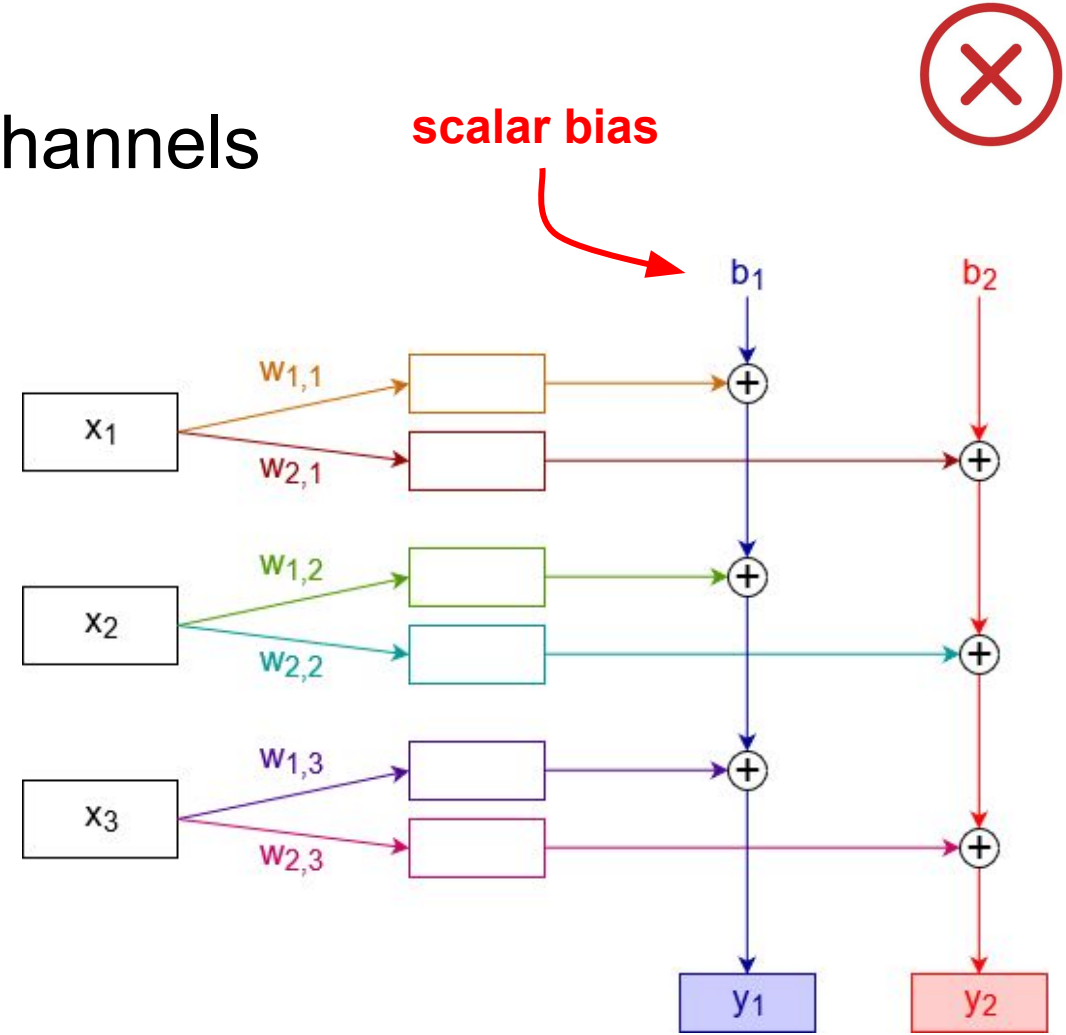


3 input channels, 2 filters:
Each filter has 3 channels now too…

Convolutional layer - Channels

The number of channels would explode combinatorially.

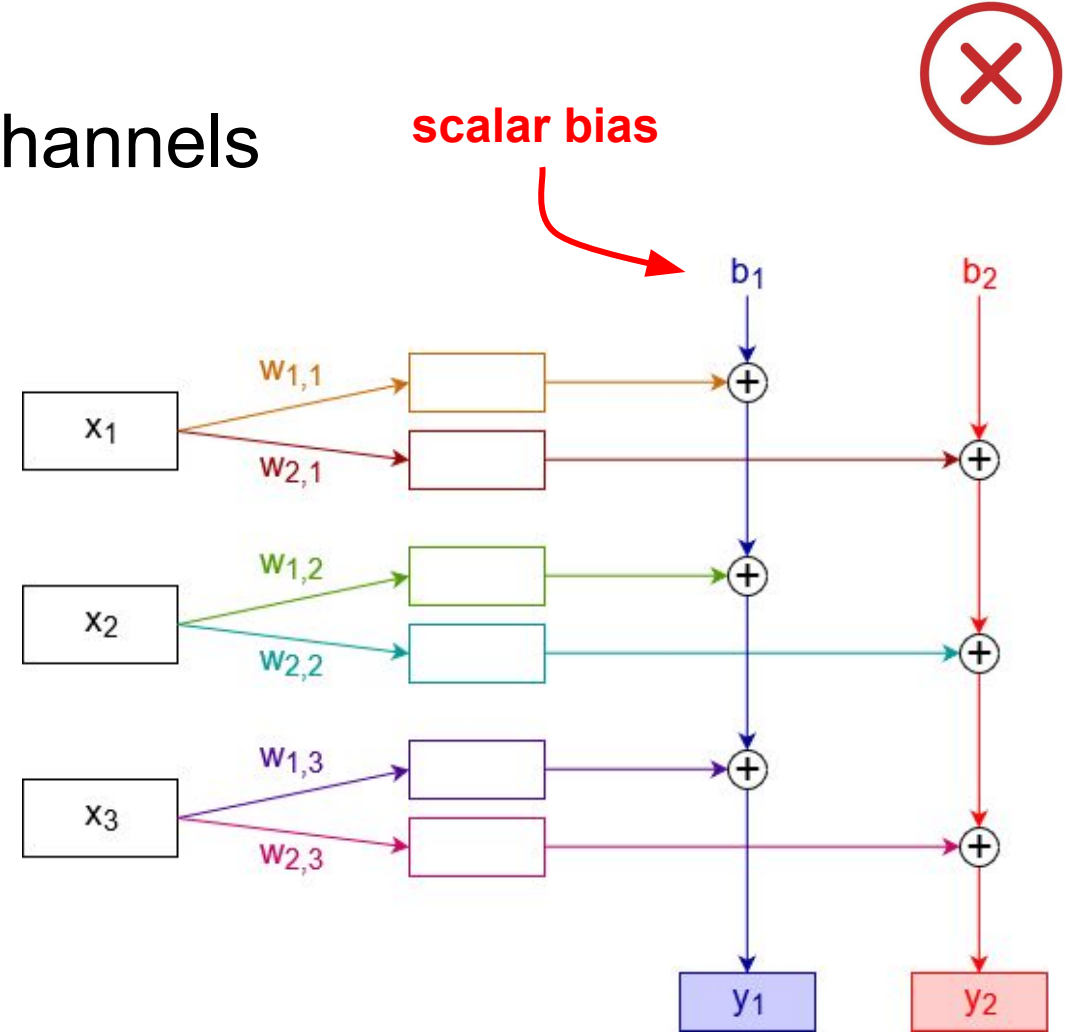
To avoid this:



Convolutional layer - Channels

The number of channels would explode combinatorially.

To avoid this:



We **sum up** the different heatmap channels for each filter (+ bias).

3 input channels, 2 filters →
2 output channels

Convolutional layer - Channels

Convolutional layer (multi-channel case)**:**

- P input channels, Q output channels

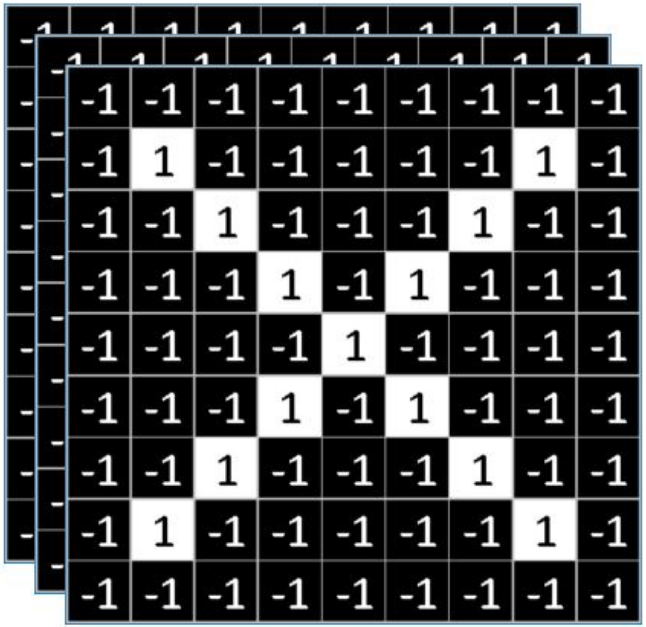
$$\hat{y}_q[i,j] = g\Big(\sum_{p=1}^{P} \sum_{u=1}^{U} \sum_{v=1}^{V} x_p[i+u, j+v] \cdot w_{p,q}[u,v] \quad + b_q \Big)$$

$$x_p \in \mathbb{R}^{I \times J} \qquad b_q \in \mathbb{R} \qquad q \in \{1, \ \ldots, \ Q\}$$

$$w_{p,q} \in \mathbb{R}^{U \times V} \qquad \hat{y}_q \in \mathbb{R}^{(I-U+1) \times (J-V+1)}$$

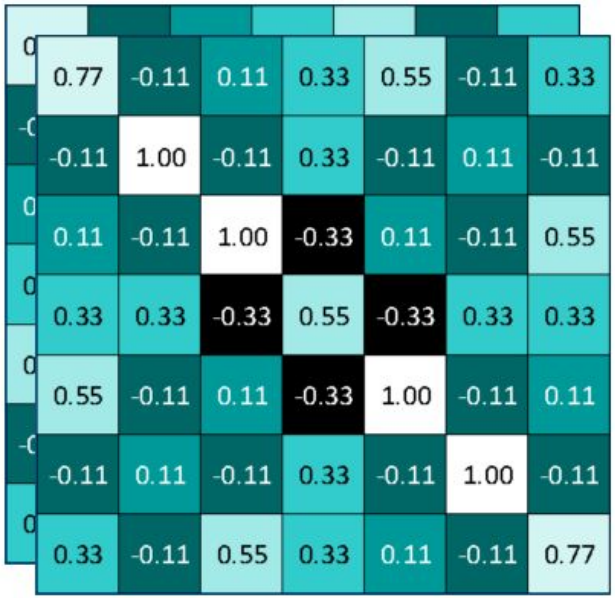
Convolutional layer - Channels

P input channels



Convolutional layer
Q filters
(each filter with **P** channels).

Q output channels



Convolutional layer

What have we achieved?

- We can learn to recognize several small, simple patterns.
- The output of the convolutional layer **returns high values** at those locations **where the content of the image matches the pattern of the filters.**

Convolutional layer

What have we achieved?

- We can learn to recognize several small, simple patterns.
- The output of the convolutional layer **returns high values** at those locations **where the content of the image matches the pattern of the filters.**

Problem:
If we want to learn regression/classification from the images,
we need to apply fully connected layers to the output of the convolutional layer.
We still have trouble with larger patterns...

Downsampling / Pooling layer

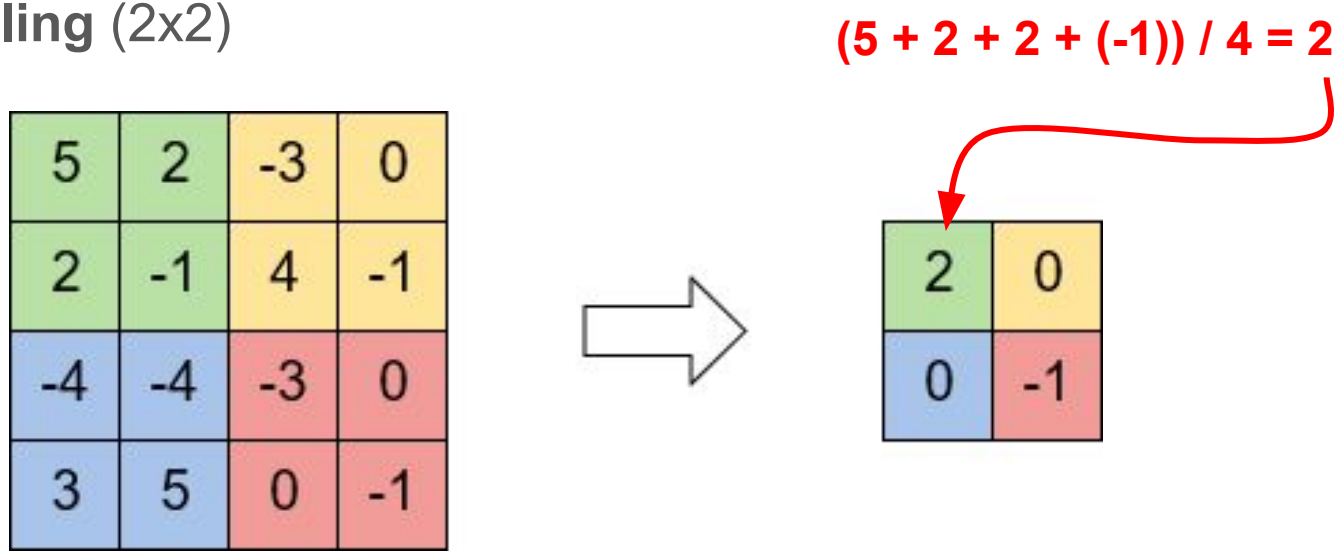
Idea:

In a simple regression/classification task, we don’t need to tell exactly where each object is located in the image down to the exact pixel.

Let’s reduce the resolution of the heatmaps!

Downsampling / Pooling layer

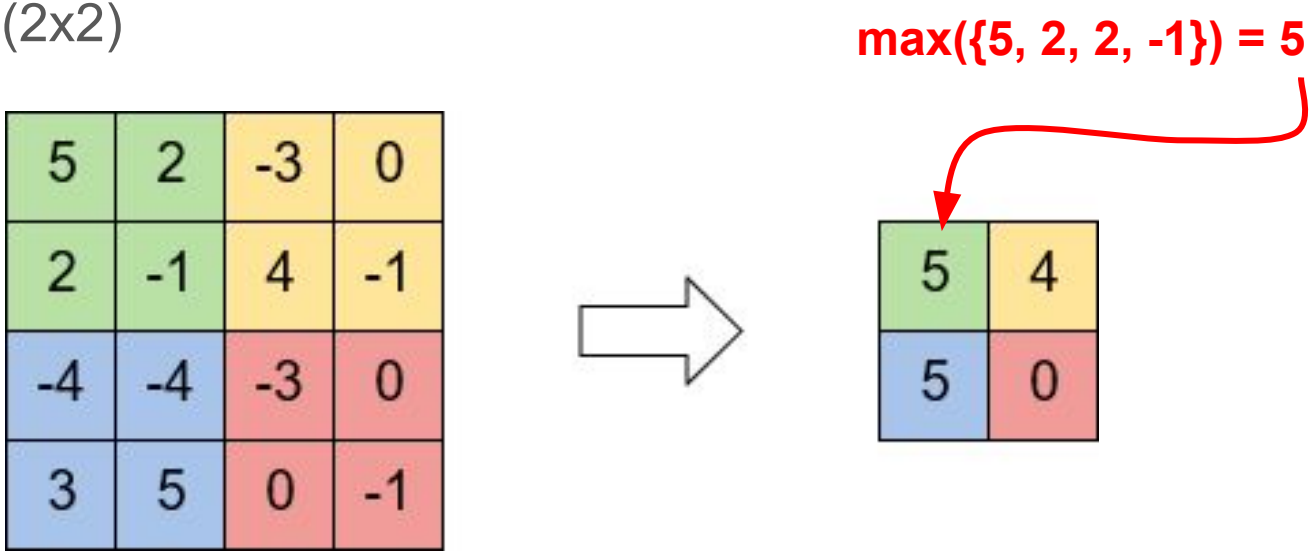
Average pooling (2x2)



Average pooling: Simple downsampling by taking the mean of a block

Downsampling / Pooling layer

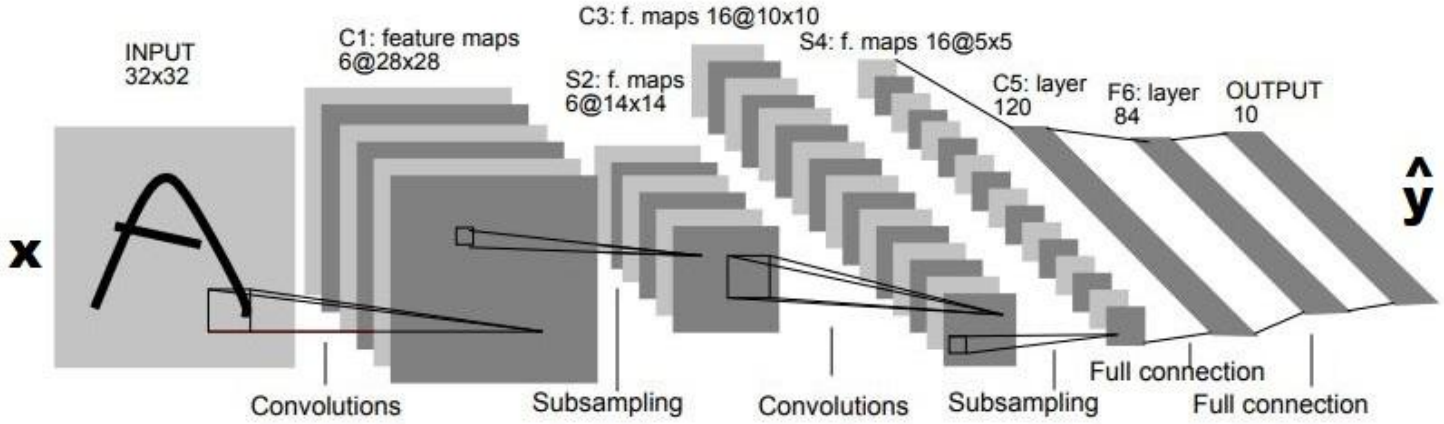
Max pooling (2x2)



Max pooling: We take the maximum value of the block.

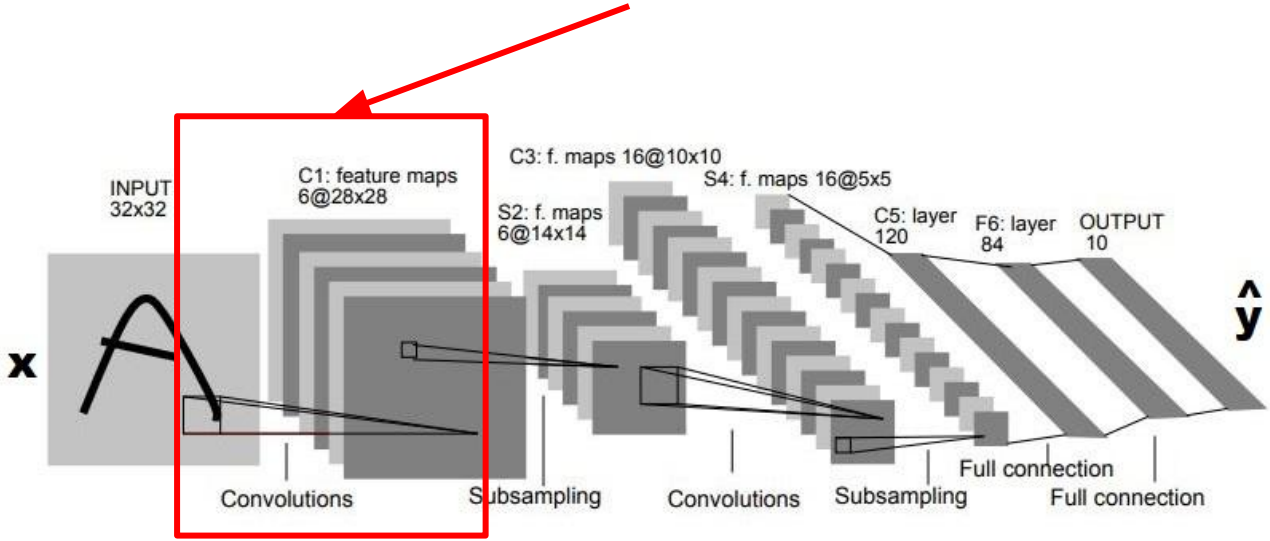
Convolutional network - A classic architecture

A classic convolutional architecture - LeNet-5 (1998, Y. LeCun et al.)



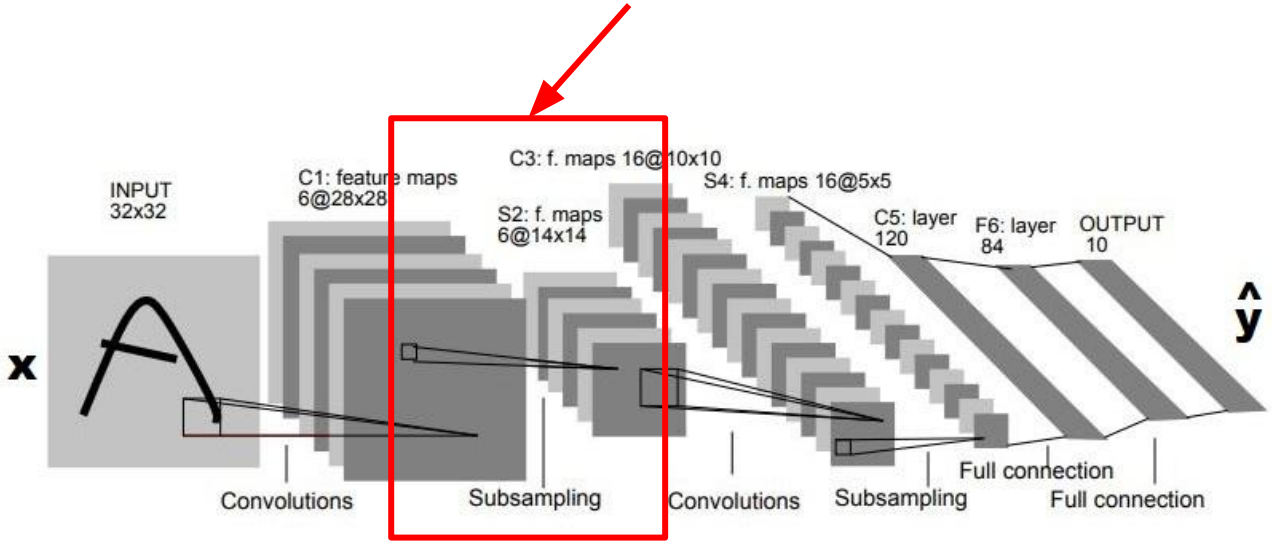
Convolutional network - A classic architecture

First convolutional layer: Learning different filters (here, specifically 6 of them with size 5x5) for the detection of edges, corners, shade transitions
→ A different heatmap is a result of a convolution with each filter.



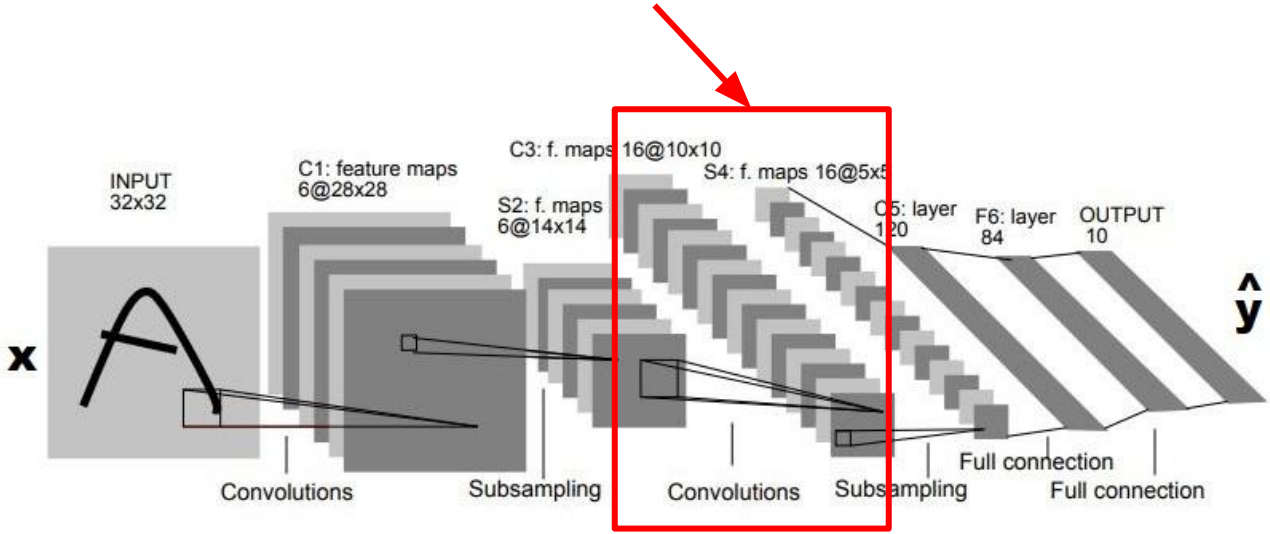
Convolutional network - A classic architecture

First pooling layer: Downsampling the (6) heatmap outputs from the previous layer, independently for each channel.
Output image size is halved due to the 2x2 pooling block size.



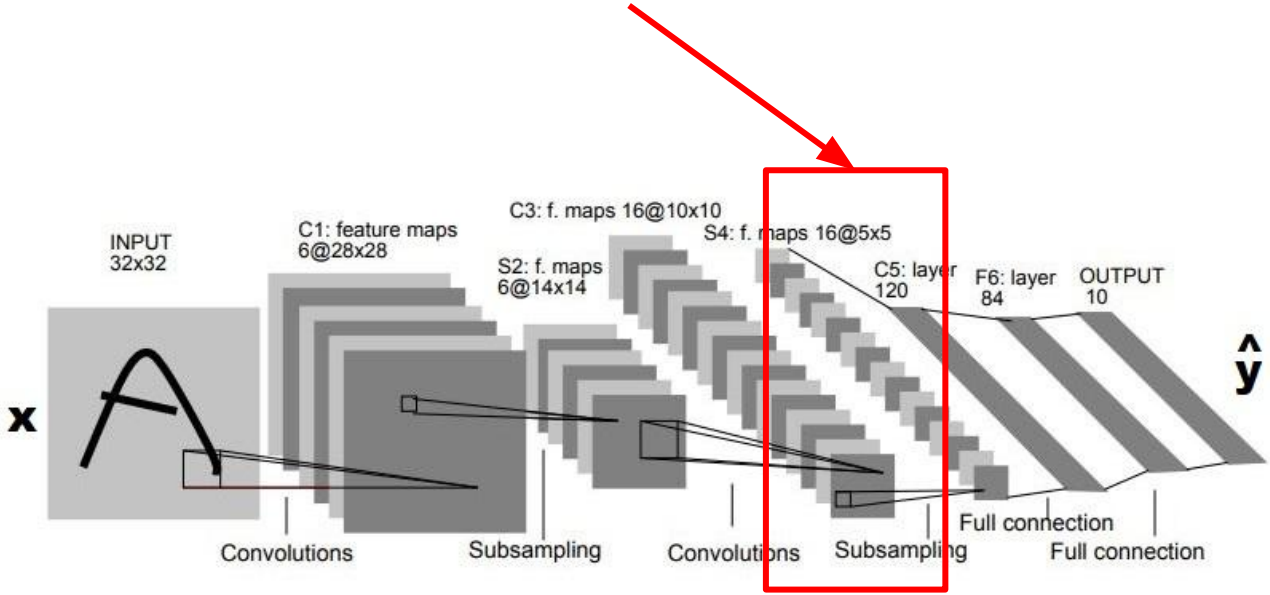
Convolutional network - A classic architecture

Second convolutional layer: Learning different filters (here, specifically 16 of them with 6 channels and size 5x5) for the detection of the parts of digits. The filters in this layer operate on the downscaled heatmaps from the previous layer.



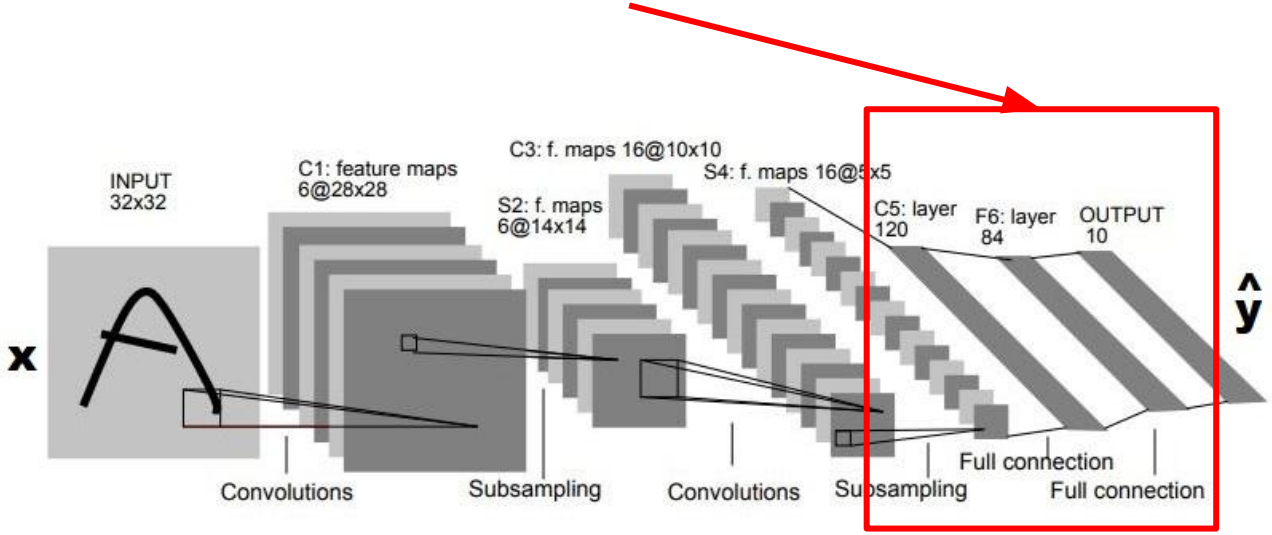
Convolutional network - A classic architecture

Second pooling layer: Downsampling the (16) heatmap outputs from the previous layer. Output image size is halved again.



Convolutional network - A classic architecture

Fully connected layers: The heatmaps of the last pooling layer are rearranged into a vector. This is fed into the final fully connected layers, solving a regression or a classification task.



Convolutional network

Alternating convolution and pooling layers
→ **Hierarchical pattern recognition**

The network learns to recognize larger, higher-level patterns as combinations of several smaller, simpler patterns.

Convolutional network

Alternating convolution and pooling layers
→ **Hierarchical pattern recognition**

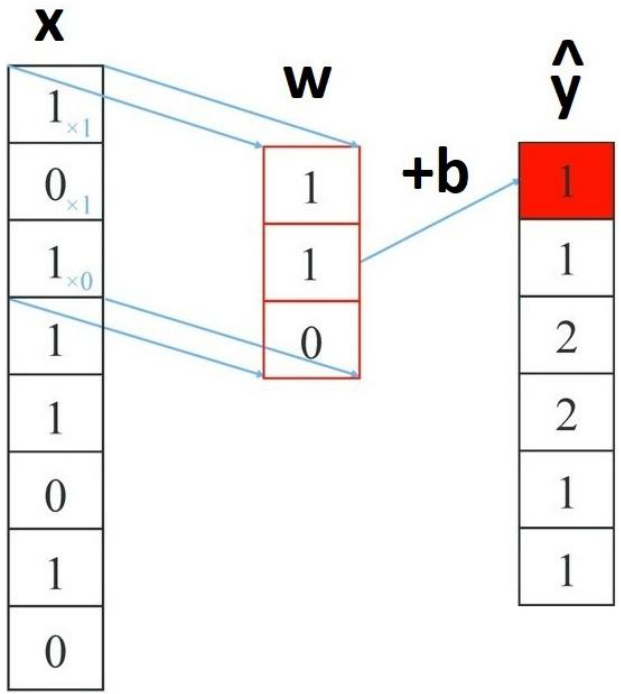
The network learns to recognize larger, higher-level patterns as combinations of several smaller, simpler patterns.

Convolutional layers are responsible for the recognition of patterns.

Pooling layers are responsible for downscaling heatmaps, thereby enabling the next convolutional layer to detect larger patterns. In other words, they increase the size of the receptive fields of the neurons in the next layer.

Convolutional layer - 1D

Convolution is not limited to two dimensions



$$\hat{y}_i = g(\langle w, x_{i,\dots,(i+U-1)} \rangle + b) =$$

$$= g(\sum_{u=1}^{U} w_u \cdot x_{i+u-1} + b)$$

$$w \in \mathbb{R}^U$$

$$b \in \mathbb{R}$$

1D convolutional layer!

Convolutional network

A trained convolutional network is (ideally) **translation invariant.**

Using a convolutional network is **ideal when the input variables can be indexed along one, or several dimensions.**

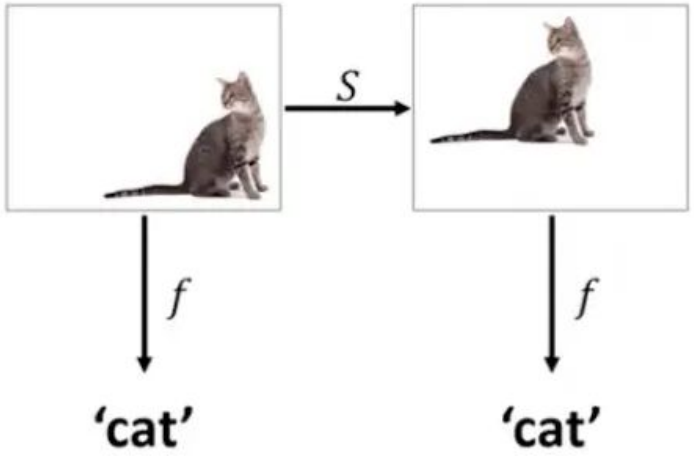
It is not limited to image recognition:

- Sound analysis, speech recognition (1D, 2D)
- MRI and CT scans, 3D models (3D)
- Game boards (e.g., AlphaGo, 2D)
- Financial data, time series (1D, 2D)

Image source: Bernard Kanz, Imperial College, London

Invariance vs. equivariance

(Translation) **invariance**



(Translation) **equivariance**

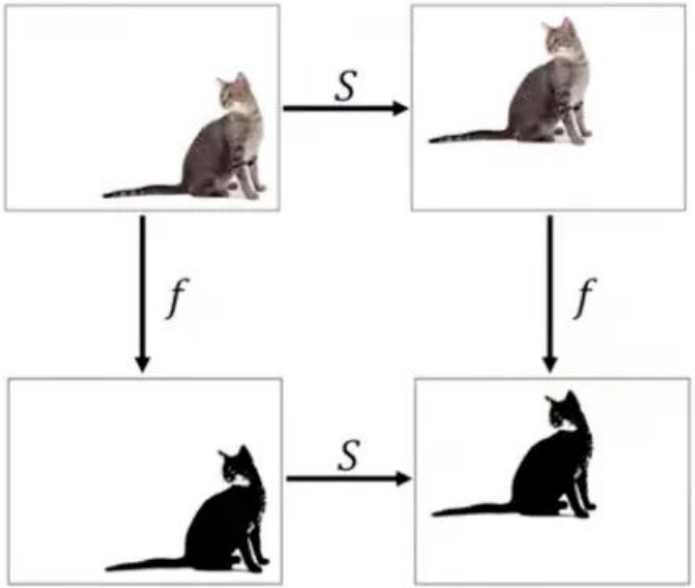
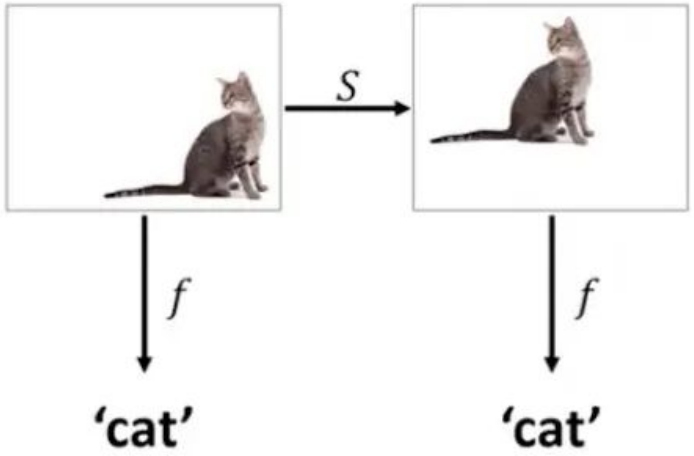


Image source: Bernard Kanz, Imperial College, London

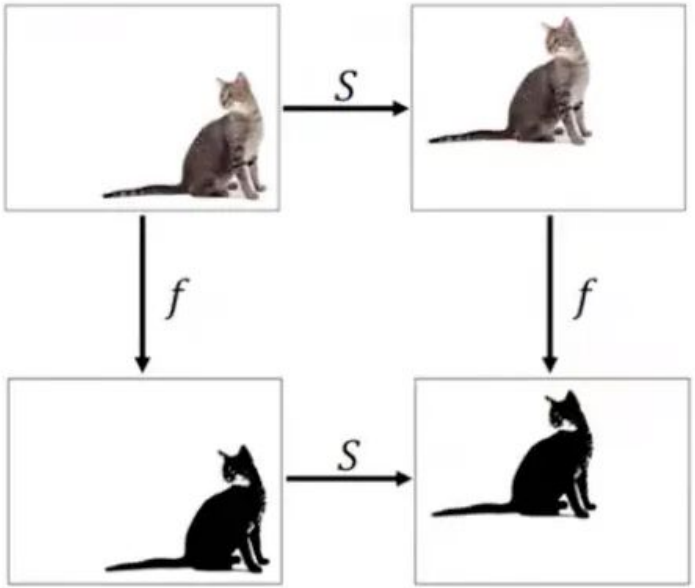
Invariance vs. equivariance

(Translation) **invariance**



A trained ConvNet is (ideally) **translation invariant.**

(Translation) **equivariance**



A convolutional layer is translation equivariant.