Deep Learning: Lecture 6

Alexander Schönhuth

UU

November 13, 2019

Reminder: Dropout, Regularization

Regularization Revisited

L1 versus L2 Regularization

- In L1 regularization, weights shrink by a *constant* amount.
- In L2 regularization, weights shrink by an amount *proportionally* to w.
- L1 regularization tends to bring forward a small number of *high-importance connections*.
- L2 regularization tends to keep all weights small.

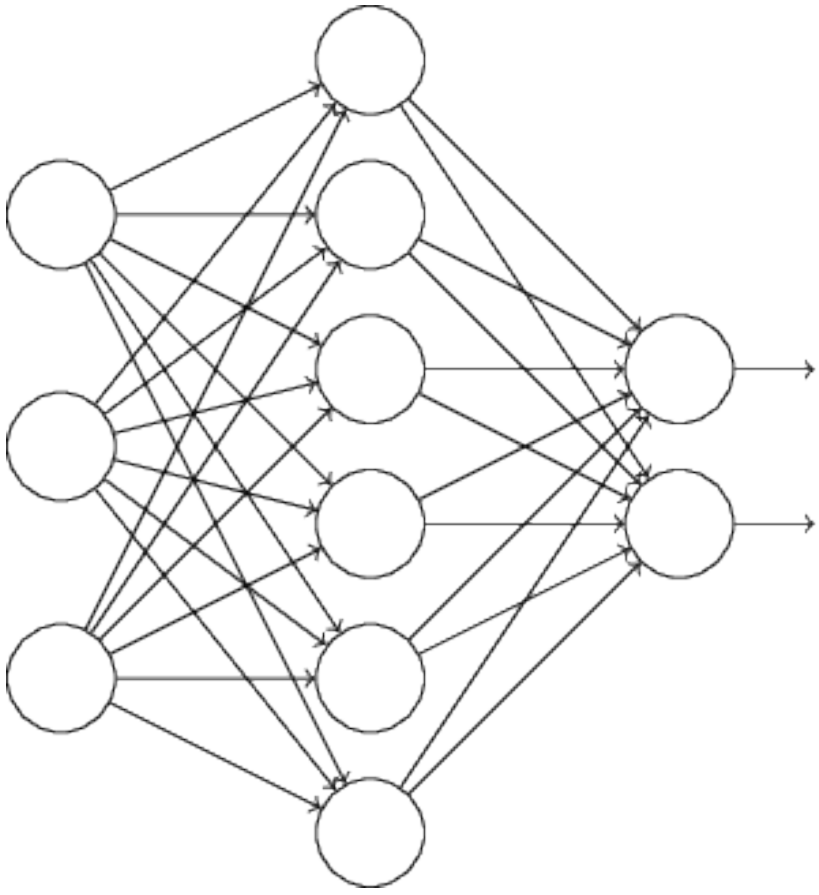
Regularization Revisited

L1 versus L2 Regularization

- In L1 regularization, weights shrink by a *constant* amount.
- In L2 regularization, weights shrink by an amount *proportionally* to w.
- L1 regularization tends to bring forward a small number of *high-importance connections*.
- L2 regularization tends to keep all weights small.

Regularization Revisited

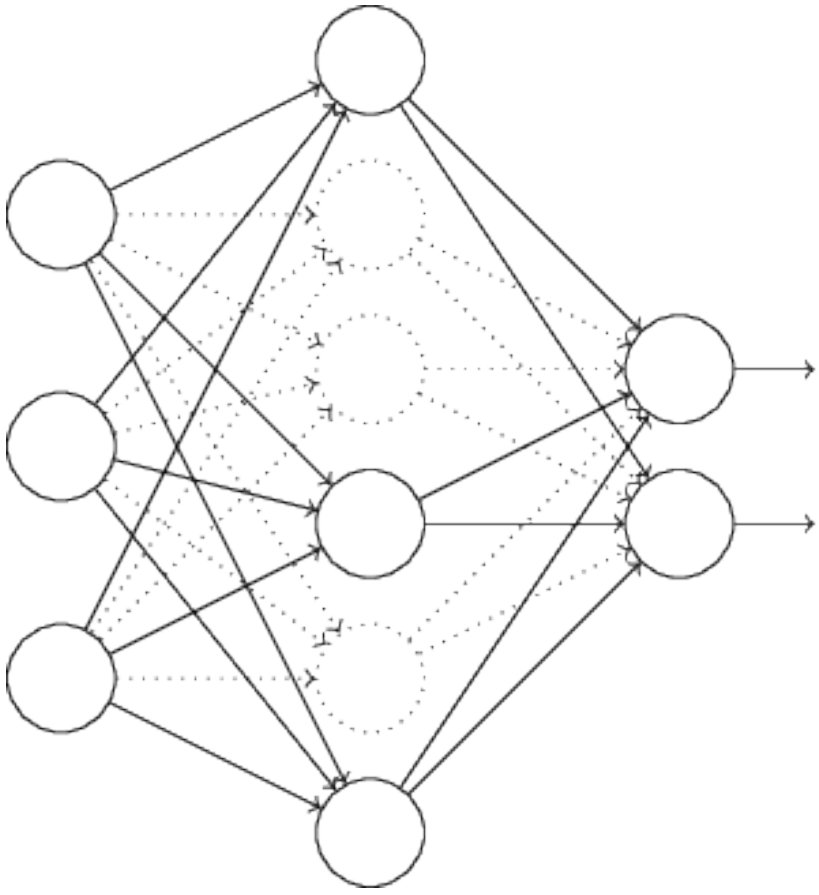
Dropout



Full network, before dropout

Regularization Revisited

Dropout



Network after having dropped half of the hidden nodes

Regularization Revisited

Dropout

Procedure

1. Choose a mini batch of training data of size $\hat{m}$
2. Randomly delete half of the hidden nodes, while keeping all input and output nodes
3. Train the resulting network using the mini batch; update all weights and biases
4. If validation accuracy not yet satisfying, return to 1.
5. After each epoch, decrease each weight by a factor of $\frac{1}{2}$

Regularization Revisited

Dropout

Procedure

1. Choose a mini batch of training data of size $\hat{m}$
2. Randomly delete half of the hidden nodes, while keeping all input and output nodes
3. Train the resulting network using the mini batch; update all weights and biases
4. If validation accuracy not yet satisfying, return to 1.
5. After each epoch, decrease each weight by a factor of $\frac{1}{2}$

Regularization Revisited

Dropout

Procedure

1. Choose a mini batch of training data of size $\hat{m}$
2. Randomly delete half of the hidden nodes, while keeping all input and output nodes
3. Train the resulting network using the mini batch; update all weights and biases
4. If validation accuracy not yet satisfying, return to 1.
5. After each epoch, decrease each weight by a factor of $\frac{1}{2}$

Regularization Revisited

Dropout

Procedure

1. Choose a mini batch of training data of size $\hat{m}$
2. Randomly delete half of the hidden nodes, while keeping all input and output nodes
3. Train the resulting network using the mini batch; update all weights and biases
4. If validation accuracy not yet satisfying, return to 1.
5. After each epoch, decrease each weight by a factor of $\frac{1}{2}$

Dropout

Explanations

- Dropout can be perceived as averaging over several smaller networks, where averaging over several models is generally helpful to prevent overfitting
- Dropout can be perceived as projecting points in parameter space onto the linear subspace defined by only half of the elementary basis vectors.
- Combining optima in subspaces yields a selection of parameters that are not optimal, but nearby an optimum
  ☞ experience shows that this prevents overfitting
- Dropout prevents “co-adaptation of neurons”

Dropout

Explanations

- Dropout can be perceived as averaging over several smaller networks, where averaging over several models is generally helpful to prevent overfitting
- Dropout can be perceived as projecting points in parameter space onto the linear subspace defined by only half of the elementary basis vectors.
- Combining optima in subspaces yields a selection of parameters that are not optimal, but nearby an optimum
  ☞ experience shows that this prevents overfitting
- Dropout prevents “co-adaptation of neurons”

Dropout

Explanations

- Dropout can be perceived as averaging over several smaller networks, where averaging over several models is generally helpful to prevent overfitting
- Dropout can be perceived as projecting points in parameter space onto the linear subspace defined by only half of the elementary basis vectors.
- Combining optima in subspaces yields a selection of parameters that are not optimal, but nearby an optimum
  ☞ experience shows that this prevents overfitting
- Dropout prevents “co-adaptation of neurons”

Dropout

Explanations

- Dropout can be perceived as averaging over several smaller networks, where averaging over several models is generally helpful to prevent overfitting
- Dropout can be perceived as projecting points in parameter space onto the linear subspace defined by only half of the elementary basis vectors.
- Combining optima in subspaces yields a selection of parameters that are not optimal, but nearby an optimum
  ☞ experience shows that this prevents overfitting
- Dropout prevents “co-adaptation of neurons”

L1/2 Regularization, Dropout, Early Stopping

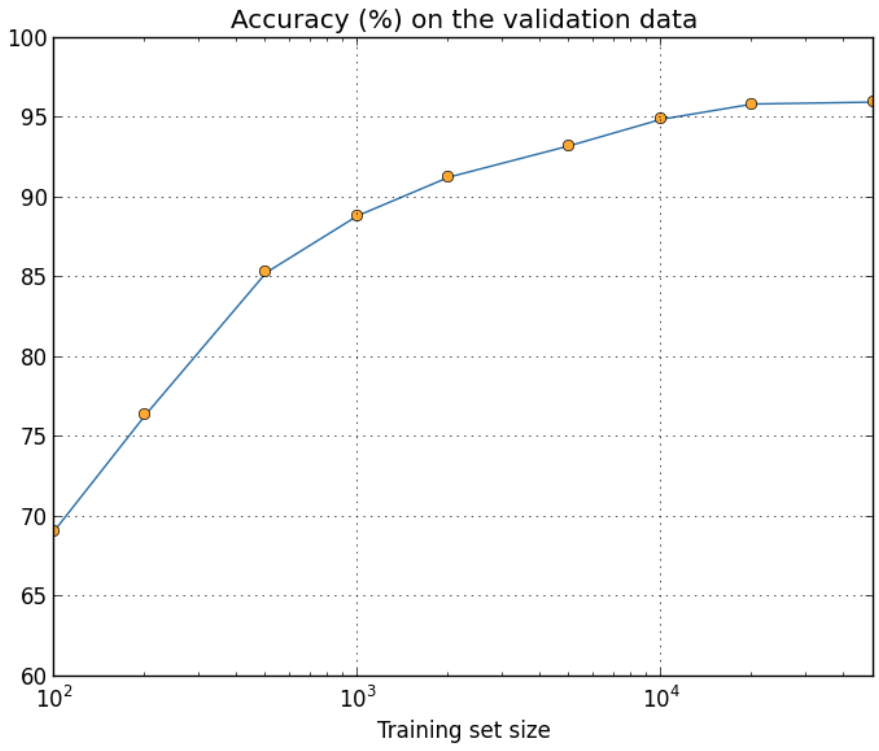
Take-Home Message

Try to find a reasonable point near the very optimum

- *L1/2 regularization*: shrink or eliminate weights that don't change much
- *Dropout*: Randomly project points to linear subspaces, and optimize there, and then average out
- *Early stopping*: Stop before reaching the optimum

Regularization Revisited

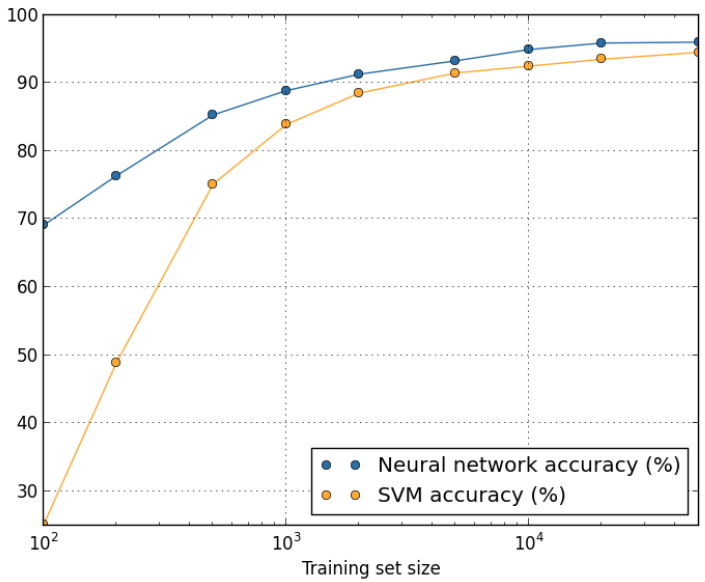
Artificial Expansion of Training Data



More training data improves test accuracy

Regularization Revisited

Artificial Expansion of Training Data

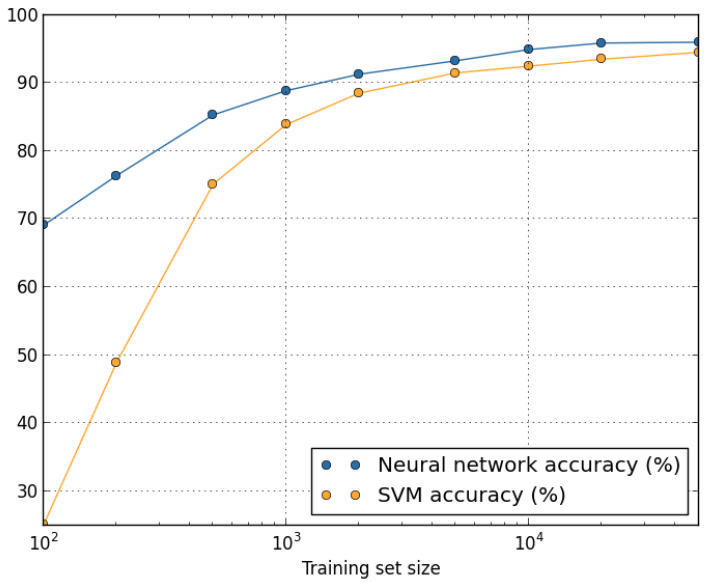


NN versus SVM on same training data

- Sometimes better training data delivers substantial improvements
- Always good to aim for methodical improvements, but:
- Don't miss "easy wins" by generating more and/or better training data

Regularization Revisited

Artificial Expansion of Training Data

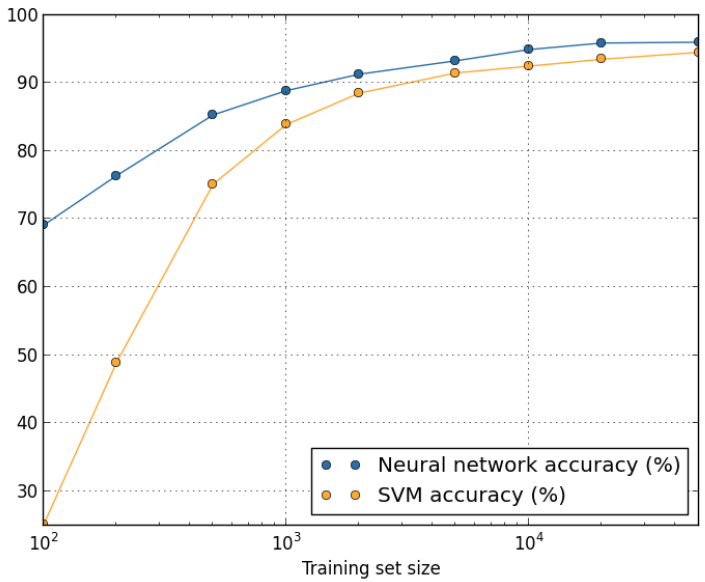


NN versus SVM on same training data

- Sometimes better training data delivers substantial improvements
- Always good to aim for methodical improvements, but:
- Don't miss "easy wins" by generating more and/or better training data

Regularization Revisited

Artificial Expansion of Training Data



NN versus SVM on same training data

- Sometimes better training data delivers substantial improvements
- Always good to aim for methodical improvements, but:
- Don't miss "easy wins" by generating more and/or better training data

Regularization Revisited

Generating Artificial Training Data

Rotating 5 by 15 degrees to the left yields new training datum

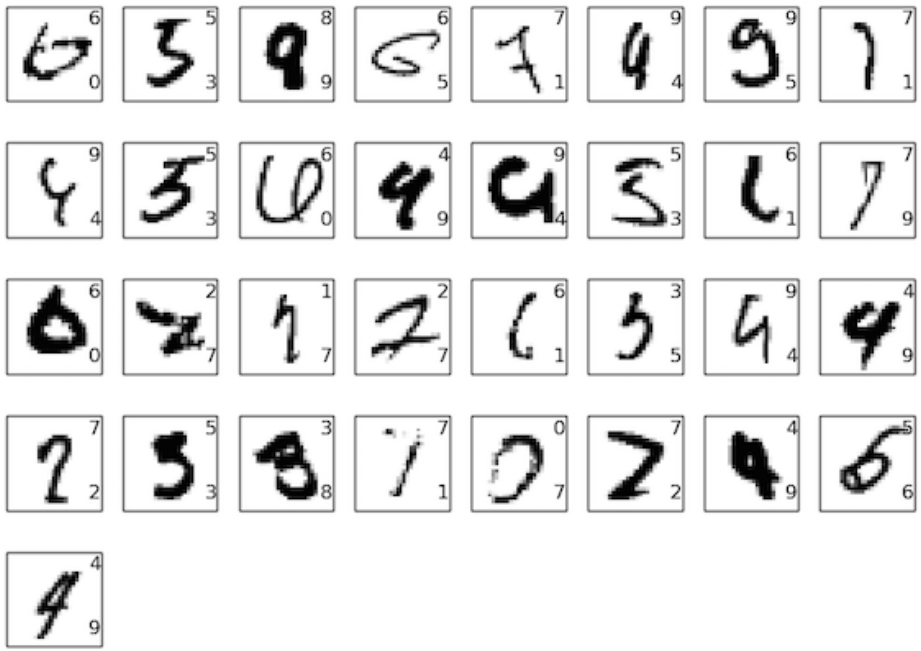
Other Techniques

- Translating, skewing
- "Elastic distortions"
- For more details, see [Simard, Steinkraus & Platt, 2003] `https://ieeexplore.ieee.org/document/1227801`

Convolutional Neural Networks (CNNs)

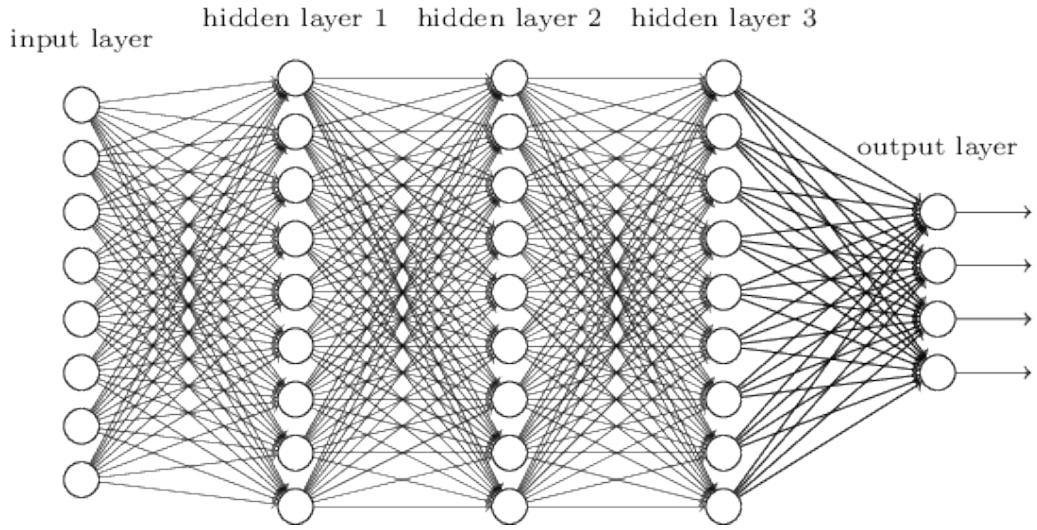
Goal

Setting up a neural network that correctly classifies 9967 out of 10 000 images; see below for the 33 misclassified ones.



33 misclassified images; correct/predicted classification upper/lower right corner

Fully Connected Networks



Fully connected neural network with 3 hidden layers

Issue: With fully connected NN's, we only reach about 98% accuracy in prediction.

Question: How to get to 99,67% accuracy?

Convolutional Neural Networks

Motivation

- Use that images have a spatial structure
  - ☞ Neighboring pixels are more likely to belong to the same structural elements
- Exploit this to speed up training, and reduce number of parameters (weights)

Convolutional Neural Networks

Motivation

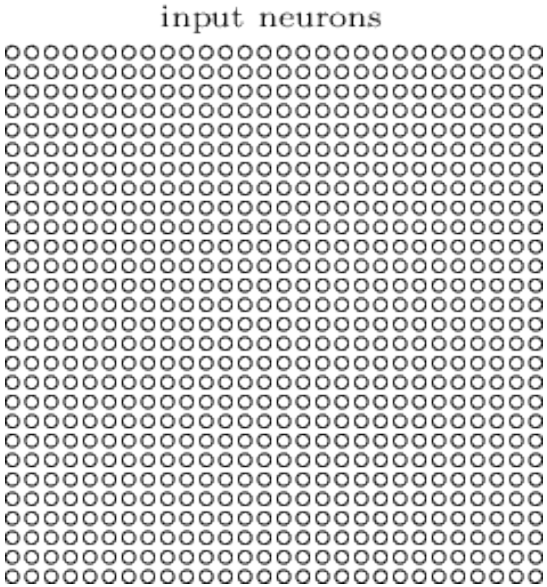
- Use that images have a spatial structure
  - ☞ Neighboring pixels are more likely to belong to the same structural elements
- Exploit this to speed up training, and reduce number of parameters (weights)

Basic Ideas

- Local receptive fields
- Shared weights
- Pooling

Convolutional Neural Networks

Local Receptive Fields



One image are 28 x 28 = 784 pixels

In a fully connected network

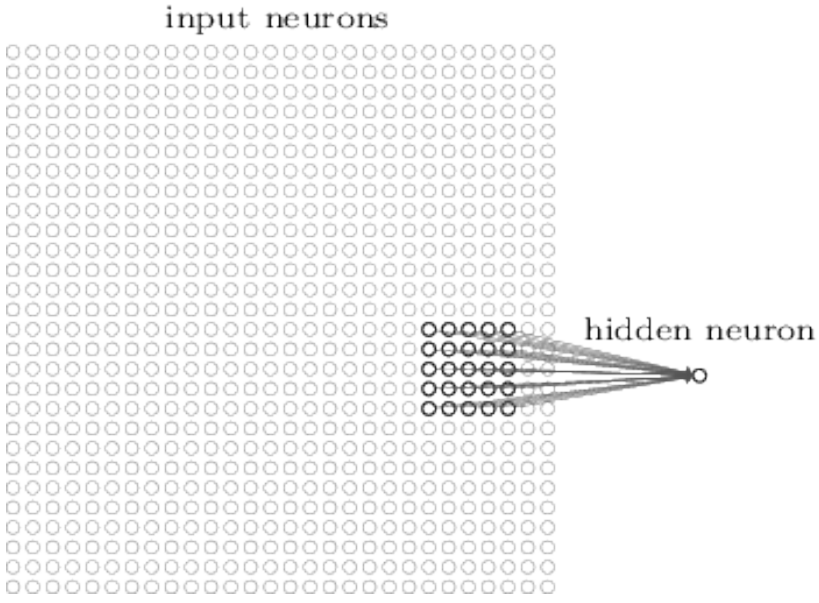
- Every node of the first hidden layer is connected to every input neuron (a.k.a pixel)
- Every node of the second layer is connected to every neuron in the first hidden layer

Convolutional Neural Networks

Local Receptive Fields

In a convolutional NN,

- Every node in the first hidden layer is connected to a rectangular subregion
- Here: subregion = square of 5x5=25 input neurons



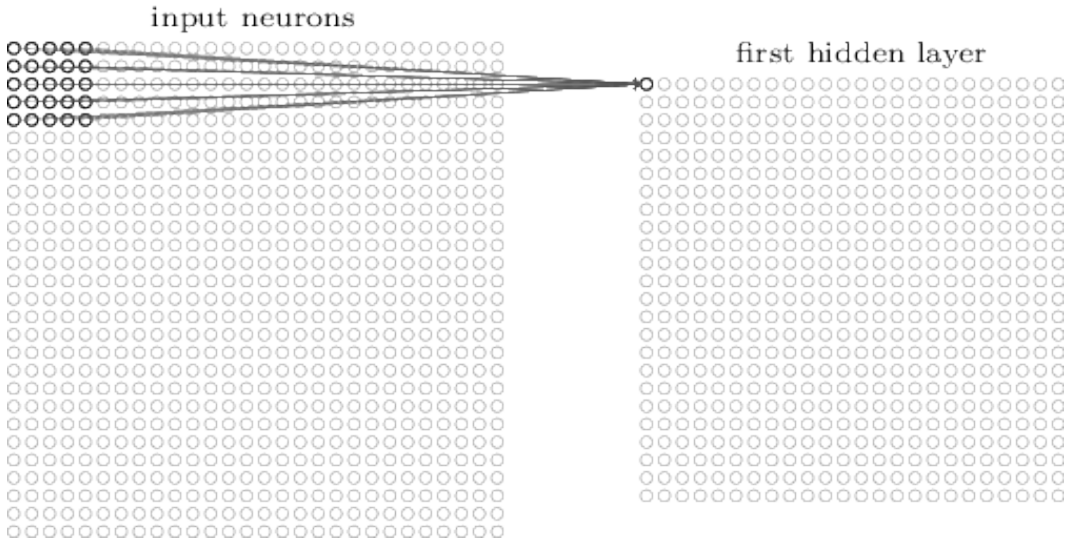
Convolutional filter of size 5 x 5

Definition

The region in the input images to which a hidden neuron is connected is called the *local receptive field (LRF)* of the hidden neuron.

Convolutional Neural Networks

Local Receptive Fields



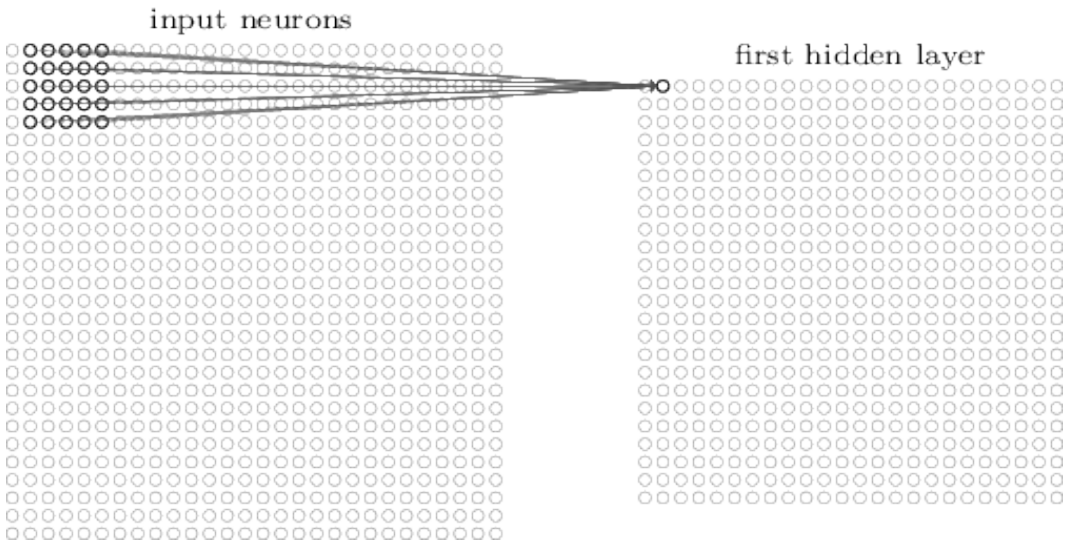
One receptive field is responsible for one hidden layer

Procedure

- Slide the local receptive field across the entire image
- *Stride length*: Step size in sliding field (example here: stride = 1)

Convolutional Neural Networks

Local Receptive Fields



One receptive field is responsible for one hidden layer

Procedure

- Slide the local receptive field across the entire image
- *Stride length*: Step size in sliding field (example here: stride = 1)

Convolutional Neural Networks

Computing Hidden Layers

- One *hidden layer* is generated by one pass of the LRF
- Several hidden layers will be generated by several passes of the LRF
- The activation a_{jk}^{l+1} of the j,k-th hidden neuron within the layer, using a $M \times M$ LRF, is computed as (σ may represent activation function of choice)

$$a_{jk}^{(l+1)} = \sigma(b + \sum_{l=0}^{M} \sum_{m=0}^{M} w_{l,m} a_{j+l,k+m}^{l}) \tag{1}$$

Convolutional Neural Networks

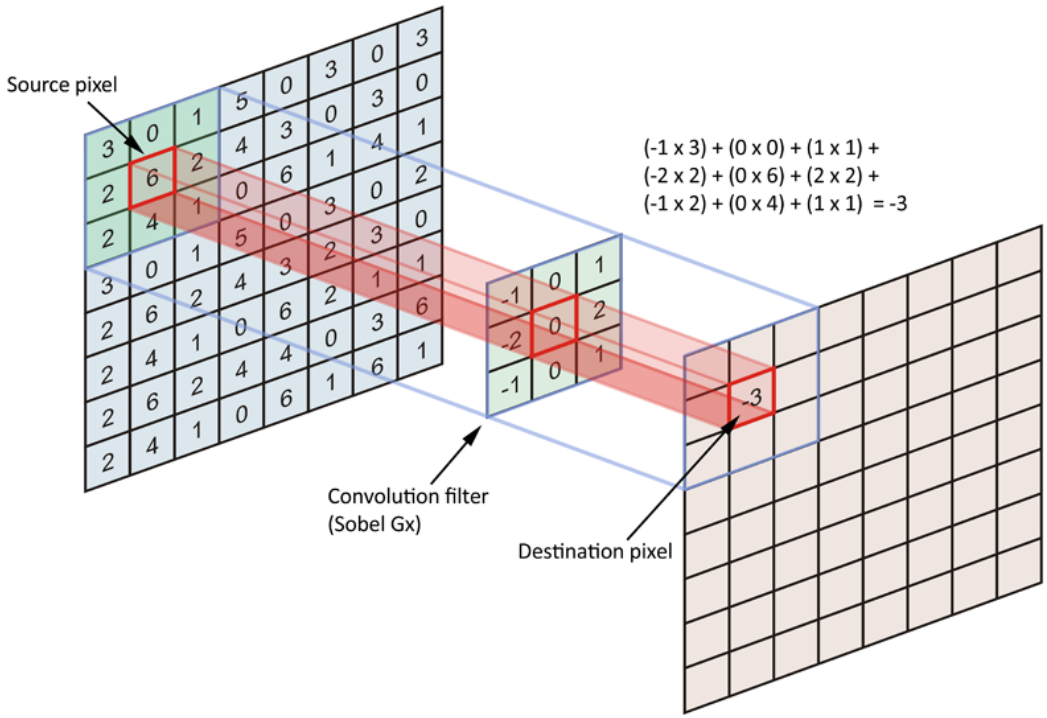
Computing Hidden Layers

- One *hidden layer* is generated by one pass of the LRF
- Several hidden layers will be generated by several passes of the LRF
- The activation a_{jk}^{l+1} of the j,k-th hidden neuron within the layer, using a $M \times M$ LRF, is computed as (σ may represent activation function of choice)

$$a_{jk}^{(l+1)} = \sigma(b + \sum_{l=0}^{M} \sum_{m=0}^{M} w_{l,m} a_{j+l,k+m}^{l}) \tag{1}$$

Convolutional Neural Networks

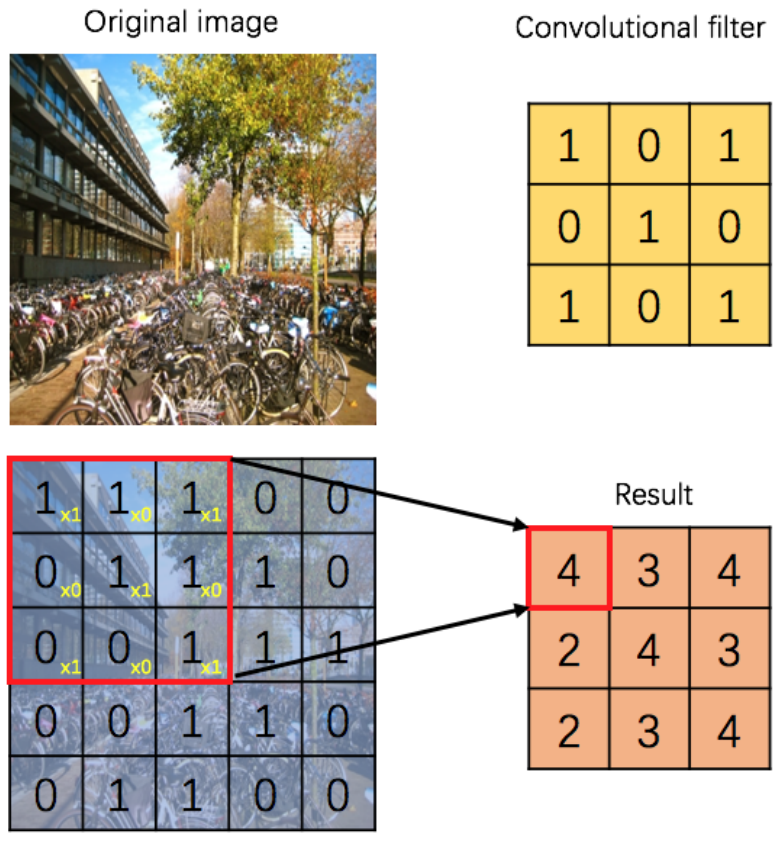
Convolutional Filters



For generating one hidden layer, identical parameters, together defining one convolutional filter, are used

Convolutional Neural Networks

Convolutional Filters

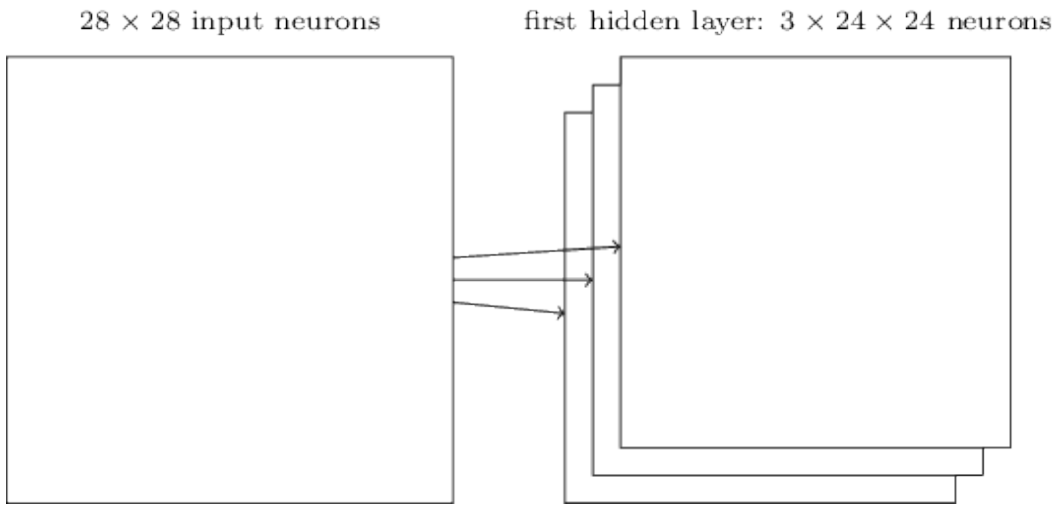


For generating one hidden layer, identical parameters, together defining one convolutional filter, are used

Convolutional Neural Networks

Convolutional Filters

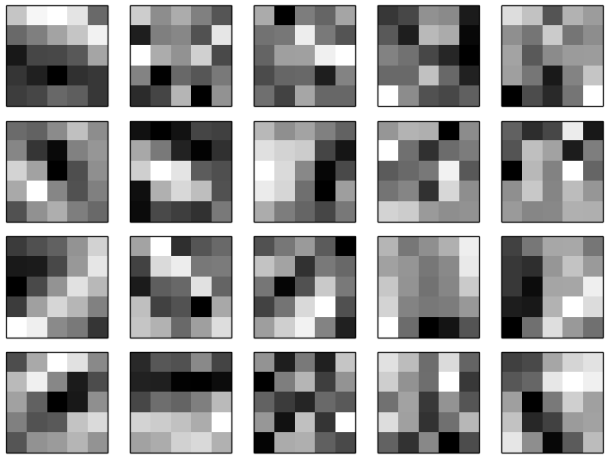
Definition
A *feature map* is a mapping associated with one convolutional filter.



- A complete convolutional layer consists of several hidden sublayers
- Each sublayer is defined by one feature map

Convolutional Neural Networks

Convolutional Filters Real World Example

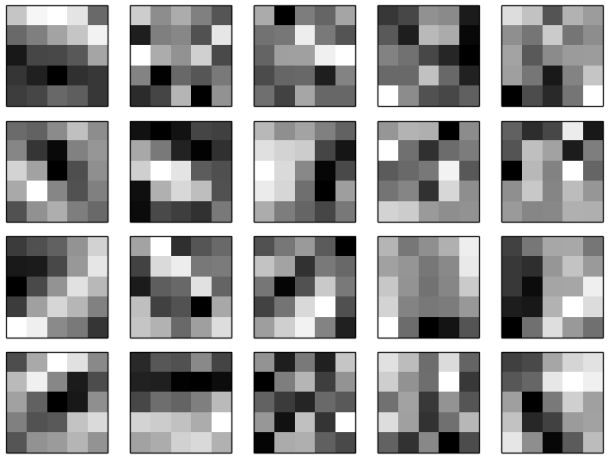


MNIST example, 20 different filters

- The darker the more positive, the whiter the more negative
- In reality, convolutional filters are hard to interpret
- Literature: M.D. Zeiler, R. Fergus, "Visualizing and Understanding Convolutional Networks", https://arxiv.org/abs/1311.2901

Convolutional Neural Networks

Convolutional Filters Real World Example



MNIST example, 20 different filters

- The darker the more positive, the whiter the more negative
- In reality, convolutional filters are hard to interpret
- Literature: M.D. Zeiler, R. Fergus, "Visualizing and Understanding Convolutional Networks", `https://arxiv.org/abs/1311.2901`

Convolutional Neural Networks

Shared Weights and Biases

- *Reminder*: The activation a_{jk}^{l+1} of the j,k-th hidden neuron within the layer, using a $M \times M$ LRF, is computed as (σ may represent activation function of choice)

$$a_{jk}^{l+1} = \sigma(b + \sum_{l=0}^{M} \sum_{m=0}^{M} w_{l,m} a_{j+l,k+m}^{l}) \tag{2}$$

- *Observation:* For each node in the same hidden layer, the same parameters $w_{l,m}, 1 \leq l, m \leq M$ are used
- That is, we only need $M \times M$ parameters to generate the entire hidden layer

Convolutional Neural Networks

Shared Weights and Biases

- *Reminder*: The activation a_{jk}^{l+1} of the j,k-th hidden neuron within the layer, using a $M \times M$ LRF, is computed as (σ may represent activation function of choice)

$$a_{jk}^{l+1} = \sigma(b + \sum_{l=0}^{M} \sum_{m=0}^{M} w_{l,m} a_{j+l,k+m}^{l}) \tag{2}$$

- *Observation:* For each node in the same hidden layer, the same parameters $w_{l,m}, 1 \leq l, m \leq M$ are used
- That is, we only need $M \times M$ parameters to generate the entire hidden layer

Convolutional Neural Networks

Shared Weights and Biases

MNIST example

:

- Convolutional layer, 20 feature maps, each of size 5×5, roughly requires $20 \times 5 \times 5 = 500$ weights
- Fully connected network, connecting 784 input neurons with 30 hidden neurons requires $784 \times 30 = 23\,520$ weights
- **CNN requires roughly 40 times less parameters**

Convolutional Neural Networks

Convolutional Layer

- *Remark*: Sometimes it helps to think of a convolutional layer, as a new type of image, where each sublayer refers to a different color.
- Note that colored pictures of size $N \times N$ come in 3 input layers of size $N \times N$, each of which refers to one of the 3 base colors red, green and blue.
- So, when using $M \times M$-filters, one applies a $3 \times M \times M$ sized *tensor* (and not an $M \times M$-sized matrix) to the input layer
- This principle can later be repeated: hence the name *tensor flow*.

Convolutional Neural Networks

Convolutional Layer

- *Remark*: Sometimes it helps to think of a convolutional layer, as a new type of image, where each sublayer refers to a different color.
- Note that colored pictures of size $N \times N$ come in 3 input layers of size $N \times N$, each of which refers to one of the 3 base colors red, green and blue.
- So, when using $M \times M$-filters, one applies a $3 \times M \times M$ sized *tensor* (and not an $M \times M$-sized matrix) to the input layer
- This principle can later be repeated: hence the name *tensor flow*.

Convolutional Neural Networks

Convolutional Layer

- *Remark*: Sometimes it helps to think of a convolutional layer, as a new type of image, where each sublayer refers to a different color.
- Note that colored pictures of size $N \times N$ come in 3 input layers of size $N \times N$, each of which refers to one of the 3 base colors red, green and blue.
- So, when using $M \times M$-filters, one applies a $3 \times M \times M$ sized *tensor* (and not an $M \times M$-sized matrix) to the input layer
- This principle can later be repeated: hence the name *tensor flow*.

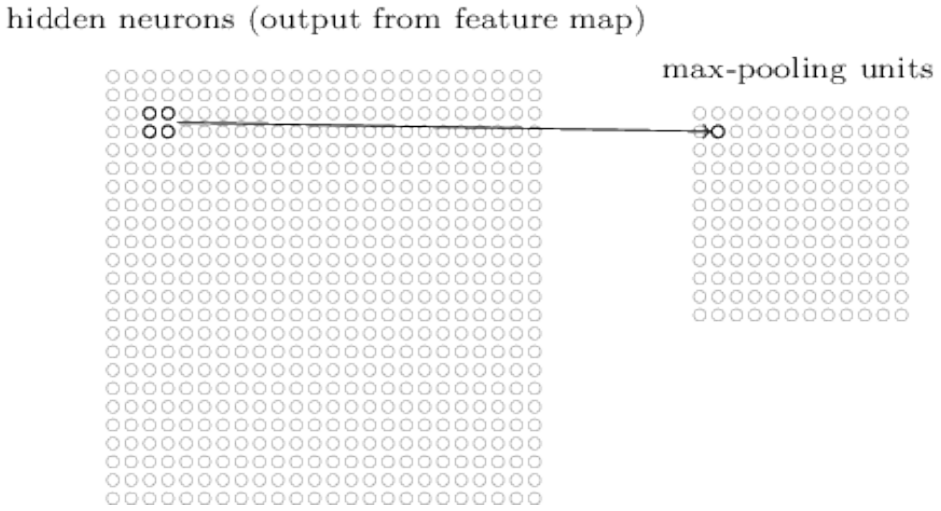
Convolutional Neural Networks

Pooling Layers

- In addition to convolutional layers, CNN's make use of *pooling layers*.
- Pooling layers generate *condensed feature maps*: it takes a rectangle of neurons, and summarizes their values into one value
- This generates a considerably smaller layer

Convolutional Neural Networks

Pooling Layers

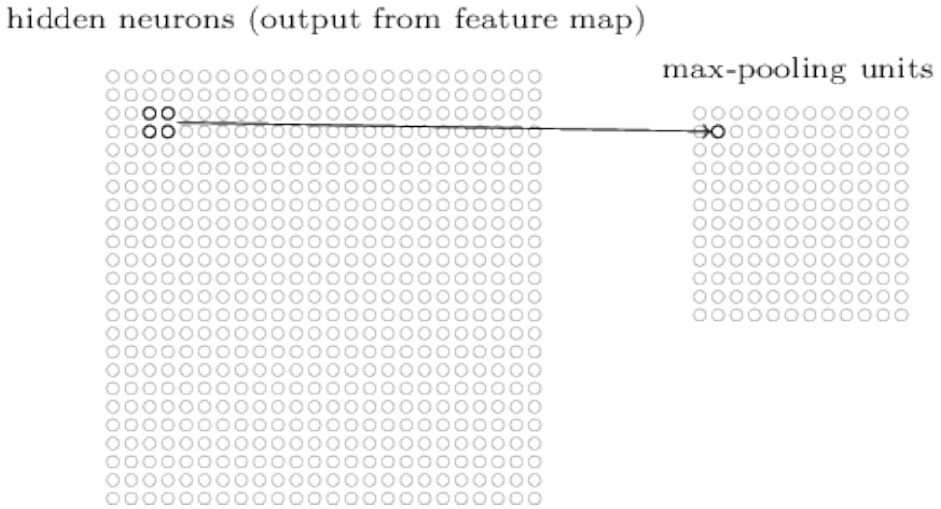


2×2 pooling

- *Max pooling*: Each $L \times L$ rectangle is mapped onto the maximum of its values
- *L2 pooling*: Each $L \times L$ rectangle is mapped to the rooted average of the squares of the values
- This overall yields a layer that is $L \times L$ times smaller
- Usually $L = 2$ is used

Convolutional Neural Networks

Pooling Layers

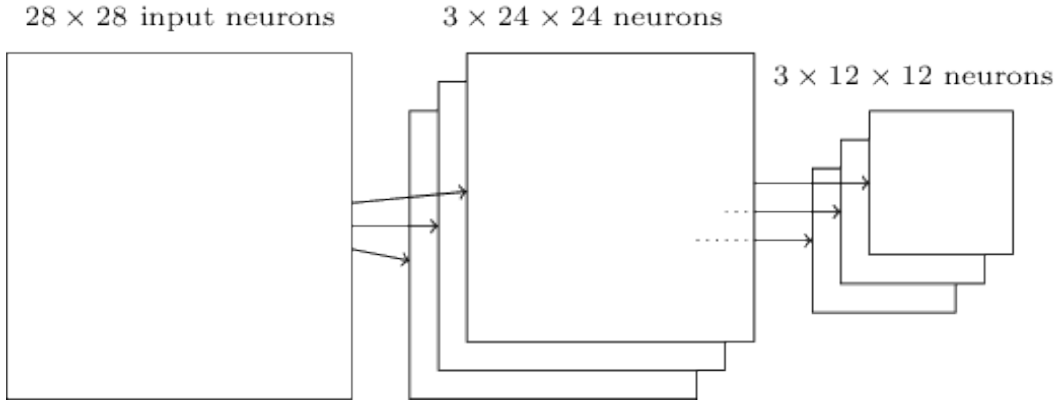


2×2 pooling

- *Max pooling*: Each $L \times L$ rectangle is mapped onto the maximum of its values
- *L2 pooling*: Each $L \times L$ rectangle is mapped to the rooted average of the squares of the values
- This overall yields a layer that is $L \times L$ times smaller
- Usually $L = 2$ is used

Convolutional Neural Networks

Combining Convolutional and Pooling Layers

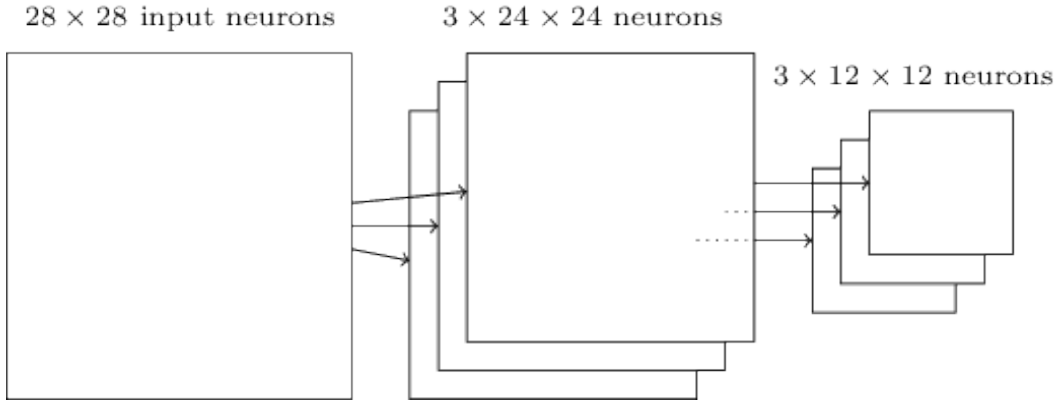


Convolutional layer followed by pooling layer

- Convolutional and pooling layers are used in combination
- Pooling layers usually follow convolutional layers
- *Intuition*:
  - The exact location of the occurrence of a feature is not important
  - Pooling helps to handle distortions and rotations

Convolutional Neural Networks

Combining Convolutional and Pooling Layers

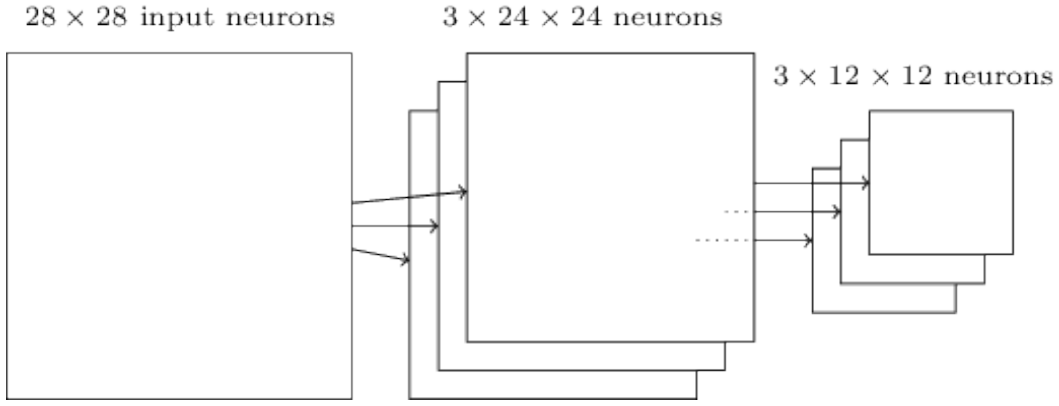


Convolutional layer followed by pooling layer

- Convolutional and pooling layers are used in combination
- Pooling layers usually follow convolutional layers
- *Intuition*:
  - The exact location of the occurrence of a feature is not important
  - Pooling helps to handle distortions and rotations

Convolutional Neural Networks

Combining Convolutional and Pooling Layers

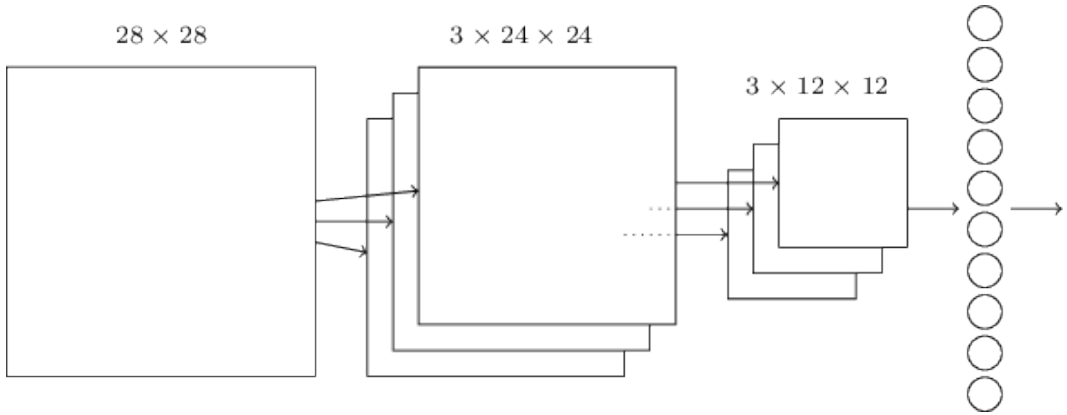


Convolutional layer followed by pooling layer

- Convolutional and pooling layers are used in combination
- Pooling layers usually follow convolutional layers
- *Intuition*:
  - The exact location of the occurrence of a feature is not important
  - Pooling helps to handle distortions and rotations

Convolutional Neural Networks

A Complete CNN



Convolution followed by pooling followed by fully connected output layer

- 10 output nodes, one for each digit
- Each output node is connected to *every* node of the pooling layer
- *Training*: Stochastic gradient descent plus backpropagation

Convolutional Neural Networks in Practice

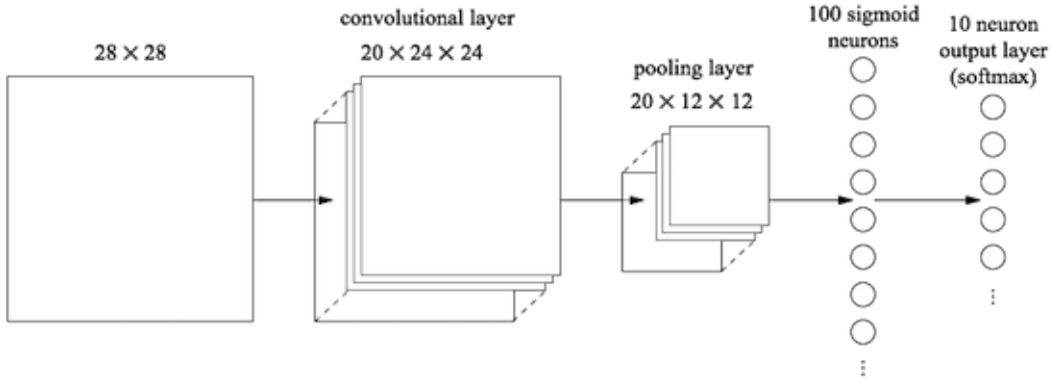
CNNs in Practice

Baseline: Simple Fully Connected Network

- *Baseline*:
  - One hidden layer, 100 neurons
  - Output layer, cost function: softmax + log-likelihood
- *Training*:
  - 60 epochs
  - Learning rate $\eta = 0.1$
  - Mini-batch size 10
- *Test accuracy*: 97.80%

CNNs in Practice

First CNN: one covolution-pooling layer

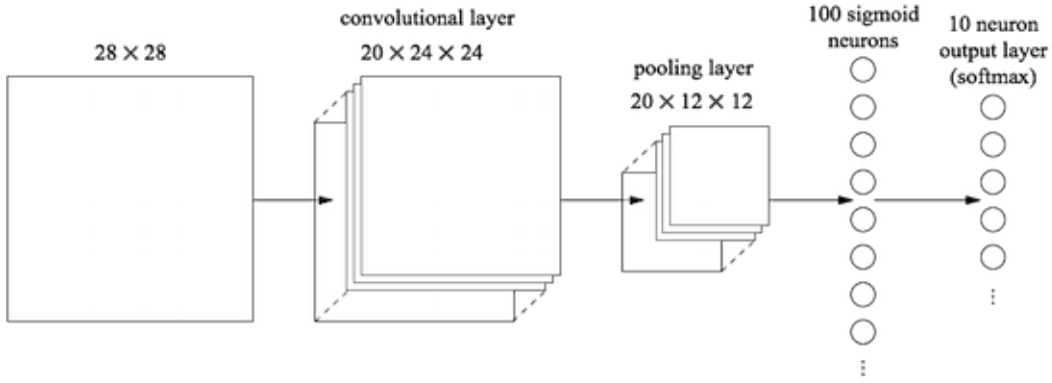


Inserting a convolution and max-pooling layer

- *Convolutional layer*:
  - 5×5 LRFs, stride length 1
  - 20 feature maps
- *Pooling layer*:
  - 2×2 max-pooling
- *Accuracy*: 98.78% test accuracy

CNNs in Practice

First CNN: one covolution-pooling layer



Inserting a convolution and max-pooling layer

- *Convolutional layer*:
  - 5×5 LRFs, stride length 1
  - 20 feature maps
- *Pooling layer*:
  - 2×2 max-pooling
- *Accuracy*: 98.78% test accuracy

CNNs in Practice

Two Convolution-Pooling Layers

- *2 Convolutional layers*:
  - *First convolution*: 20 feature maps, each associated with 5×5 LRFs, stride length 1
  - *Second convolution*: 40 feature maps, each associated with $20 \times 5 \times 5$ filter, stride length 1
- *Pooling layer*:
  - 2×2 max-pooling
- *Intuition*: after the first layer, each image consists of 12×12 pixels, where each pixel has 20 channels, each of which codes for a different "color"
- So, each LRF is a $20 \times 12 \times 12$ tensor
- Spatial strucure is still preserved in second conv-pooling layer, so employing conv-pooling makes sense
- *Accuracy*: 99.06% test accuracy

CNNs in Practice

Two Convolution-Pooling Layers

- *2 Convolutional layers*:
  - *First convolution*: 20 feature maps, each associated with 5×5 LRFs, stride length 1
  - *Second convolution*: 40 feature maps, each associated with $20 \times 5 \times 5$ filter, stride length 1
- *Pooling layer*:
  - 2×2 max-pooling
- *Intuition*: after the first layer, each image consists of 12×12 pixels, where each pixel has 20 channels, each of which codes for a different "color"
- So, each LRF is a $20 \times 12 \times 12$ tensor
- Spatial strucure is still preserved in second conv-pooling layer, so employing conv-pooling makes sense
- *Accuracy*: 99.06% test accuracy

CNNs in Practice

Two Convolution-Pooling Layers

- *2 Convolutional layers*:
  - *First convolution*: 20 feature maps, each associated with 5×5 LRFs, stride length 1
  - *Second convolution*: 40 feature maps, each associated with $20 \times 5 \times 5$ filter, stride length 1
- *Pooling layer*:
  - 2×2 max-pooling
- *Intuition*: after the first layer, each image consists of 12×12 pixels, where each pixel has 20 channels, each of which codes for a different "color"
- So, each LRF is a $20 \times 12 \times 12$ tensor
- Spatial strucure is still preserved in second conv-pooling layer, so employing conv-pooling makes sense
- *Accuracy*: 99.06% test accuracy

CNNs in Practice

Trying alternative activation functions

- *Tanh activation function*:
  - Training is (a bit) faster
  - Results are near-identical
  - *Explanation*:

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2} \tag{3}$$

- *Rectified linear units (ReLUs)*:
  - Activation:

$$f(z) = \max(0, z)$$

  - Learning rate: $\eta = 0.03$ (earlier: 0.1)
  - L2 regularization at $\lambda = 0.1$
  - *Test accuracy*: 99.23%
  - Modest gain, but also in other experiments ReLUs have shown to consistently outperform sigmoid neurons

CNNs in Practice

Trying alternative activation functions

- *Tanh activation function*:
  - Training is (a bit) faster
  - Results are near-identical
  - *Explanation*:

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2} \tag{3}$$

- *Rectified linear units (ReLUs)*:
  - Activation:

$$f(z) = \max(0, z)$$

  - Learning rate: $\eta = 0.03$ (earlier: 0.1)
  - L2 regularization at $\lambda = 0.1$
  - *Test accuracy*: 99.23%
  - Modest gain, but also in other experiments ReLUs have shown to consistently outperform sigmoid neurons

CNNs in Practice

Trying alternative activation functions

- *Tanh activation function*:
  - Training is (a bit) faster
  - Results are near-identical
  - *Explanation*:

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2} \tag{3}$$

- *Rectified linear units (ReLUs)*:
  - Activation:

$$f(z) = \max(0, z)$$

  - Learning rate: $\eta = 0.03$ (earlier: 0.1)
  - L2 regularization at $\lambda = 0.1$
  - *Test accuracy*: 99.23%
  - Modest gain, but also in other experiments ReLUs have shown to consistently outperform sigmoid neurons

CNNs in Practice

Trying alternative activation functions

- *Tanh activation function*:
  - Training is (a bit) faster
  - Results are near-identical
  - *Explanation*:

$$\sigma(z) = \frac{1 + \tanh(z/2)}{2} \tag{3}$$

- *Rectified linear units (ReLUs)*:
  - Activation:

$$f(z) = \max(0, z)$$

  - Learning rate: $\eta = 0.03$ (earlier: 0.1)
  - L2 regularization at $\lambda = 0.1$
  - *Test accuracy*: 99.23%
  - Modest gain, but also in other experiments ReLUs have shown to consistently outperform sigmoid neurons

CNNs in Practice

Expanding the Training Data

- *Experiment*:
  - Displace each image by one pixel to above, the right, below, or to the left
  - Each image has 4 extra copies
  - ☞ 250 000 images instead of 50 000
- Run the same network with ReLU's (99.23%)
- Expanding training data yields 99.37%
- P. Simard, D. Steinkraus, J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", 2003:
  - Very similar architecture
  - Training data expansion: rotations, translations, skewing
  - "Elastic distortions": emulating random oscillations of hand muscles
  - Accuracy: 99.6%

CNNs in Practice

Expanding the Training Data

- *Experiment*:
  - Displace each image by one pixel to above, the right, below, or to the left
  - Each image has 4 extra copies
  - ☞ 250 000 images instead of 50 000
- Run the same network with ReLU's (99.23%)
- Expanding training data yields 99.37%
- P. Simard, D. Steinkraus, J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", 2003:
  - Very similar architecture
  - Training data expansion: rotations, translations, skewing
  - "Elastic distortions": emulating random oscillations of hand muscles
  - Accuracy: 99.6%

CNNs in Practice

Expanding the Training Data

- *Experiment*:
  - Displace each image by one pixel to above, the right, below, or to the left
  - Each image has 4 extra copies
  - ☞ 250 000 images instead of 50 000
- Run the same network with ReLU's (99.23%)
- Expanding training data yields 99.37%
- P. Simard, D. Steinkraus, J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", 2003:
  - Very similar architecture
  - Training data expansion: rotations, translations, skewing
  - "Elastic distortions": emulating random oscillations of hand muscles
  - Accuracy: 99.6%

CNNs in Practice

Expanding the Training Data

- *Experiment*:
  - Displace each image by one pixel to above, the right, below, or to the left
  - Each image has 4 extra copies
  - ☞ 250 000 images instead of 50 000
- Run the same network with ReLU's (99.23%)
- Expanding training data yields 99.37%
- P. Simard, D. Steinkraus, J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", 2003:
  - Very similar architecture
  - Training data expansion: rotations, translations, skewing
  - "Elastic distortions": emulating random oscillations of hand muscles
  - Accuracy: 99.6%

CNNs in Practice

Extra/Larger Fully Connected Layer

- *Larger fully connected layer*:
  - 300 neurons ☞ accuracy: 99.46%
  - 1000 neurons ☞ accuracy: 99.43%
  - Not really convincing
- *Extra fully connected layer*:
  - 2 fully connected layers, each of 100 neurons ☞ accuracy 99.43%
  - 2 fully connected layers, each of 300/1000 neurons ☞ accuracy 99.47/99.48%
- No convincing improvements

CNNs in Practice

Extra/Larger Fully Connected Layer

- *Larger fully connected layer*:
  - 300 neurons ☞ accuracy: 99.46%
  - 1000 neurons ☞ accuracy: 99.43%
  - Not really convincing
- *Extra fully connected layer*:
  - 2 fully connected layers, each of 100 neurons ☞ accuracy 99.43%
  - 2 fully connected layers, each of 300/1000 neurons ☞ accuracy 99.47/99.48%
- No convincing improvements

CNNs in Practice

Extra/Larger Fully Connected Layer

- *Larger fully connected layer*:
  - 300 neurons ☞ accuracy: 99.46%
  - 1000 neurons ☞ accuracy: 99.43%
  - Not really convincing
- *Extra fully connected layer*:
  - 2 fully connected layers, each of 100 neurons ☞ accuracy 99.43%
  - 2 fully connected layers, each of 300/1000 neurons ☞ accuracy 99.47/99.48%
- No convincing improvements

CNNs in Practice

Extra/Larger Fully Connected Layer

- *Larger fully connected layer*:
  - 300 neurons ☞ accuracy: 99.46%
  - 1000 neurons ☞ accuracy: 99.43%
  - Not really convincing
- *Extra fully connected layer*:
  - 2 fully connected layers, each of 100 neurons ☞ accuracy 99.43%
  - 2 fully connected layers, each of 300/1000 neurons ☞ accuracy 99.47/99.48%
- No convincing improvements

CNNs in Practice

Extra/Larger Fully Connected Layer

- *Larger fully connected layer*:
  - 300 neurons ☞ accuracy: 99.46%
  - 1000 neurons ☞ accuracy: 99.43%
  - Not really convincing
- *Extra fully connected layer*:
  - 2 fully connected layers, each of 100 neurons ☞ accuracy 99.43%
  - 2 fully connected layers, each of 300/1000 neurons ☞ accuracy 99.47/99.48%
- No convincing improvements

CNNs in Practice

Dropout

- 2 fully connected layers each of 1000 neurons
- *Dropout* (probability = 0.5) applied to neurons in fully connected layers
- Accuracy: 99.6% (which is substantial improvement)
- *Remarks*:
  - Less epochs (40 instead of 60), because of faster training
  - More hidden neurons (1000 instead of 300 or 100) slightly preferable when using dropout
  - No dropout on convolutional layers: those have in-built resistance to overfitting because of parameter sharing

CNNs in Practice

Dropout

- 2 fully connected layers each of 1000 neurons
- *Dropout* (probability = 0.5) applied to neurons in fully connected layers
- Accuracy: 99.6% (which is substantial improvement)
- *Remarks*:
  - Less epochs (40 instead of 60), because of faster training
  - More hidden neurons (1000 instead of 300 or 100) slightly preferable when using dropout
  - No dropout on convolutional layers: those have in-built resistance to overfitting because of parameter sharing

CNNs in Practice

Ensemble of Networks

Ensemble of networks: Idea

- Train several different networks
- For example, employ repeated random initialization while always using the same architecture
- For classification, take the majority vote of the different networks
- While each network performs similarly, the majority vote may yield improvements
- Here: 5 randomly initialized network of the architecture o described in the slides before
- Accuracy: 99.67%
- That has been our goal!

CNNs in Practice

Ensemble of Networks

Ensemble of networks: Idea

- Train several different networks
- For example, employ repeated random initialization while always using the same architecture
- For classification, take the majority vote of the different networks
- While each network performs similarly, the majority vote may yield improvements
- Here: 5 randomly initialized network of the architecture o described in the slides before
- Accuracy: 99.67%
- That has been our goal!

CNNs in Practice

Ensemble of Networks

Ensemble of networks: Idea

- Train several different networks
- For example, employ repeated random initialization while always using the same architecture
- For classification, take the majority vote of the different networks
- While each network performs similarly, the majority vote may yield improvements
- Here: 5 randomly initialized network of the architecture o described in the slides before
- Accuracy: 99.67%
- That has been our goal!

CNNs in Practice

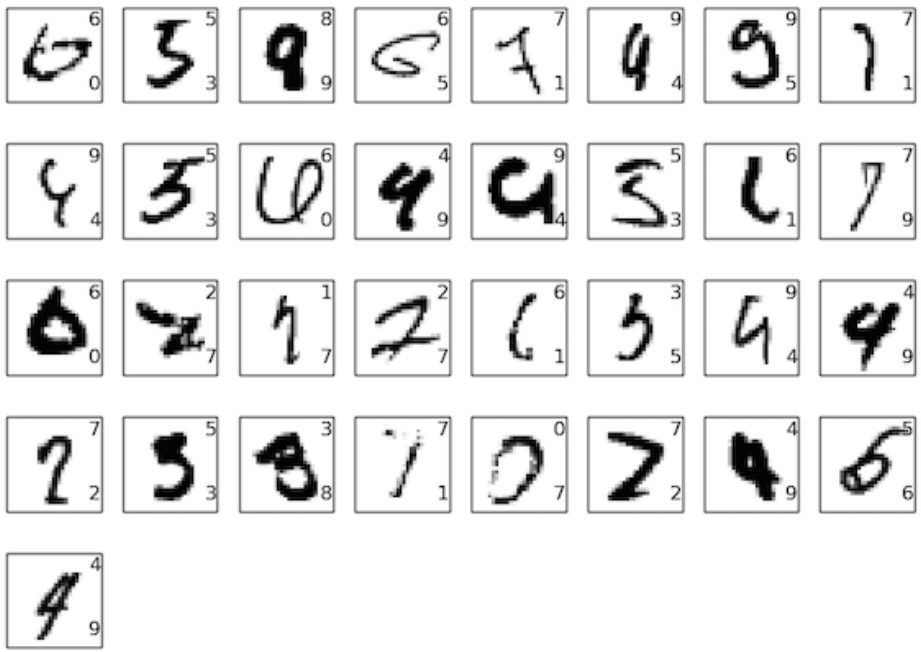
Ensemble of Networks

- Ensemble of 5 randomly initialized networks
- Architecture as described in the slides before
- Accuracy: 99.67% – that has been our goal!

CNNs in Practice

Ensemble of Networks

- Ensemble of 5 randomly initialized networks
- Architecture as described in the slides before
- Accuracy: 99.67% – that has been our goal!



33 misclassified images; correct/predicted classification upper/lower right corner

CNNs in Practice

References

- Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition", `http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf` [Architecture: "LeNet-5"]

CNNs on MNIST

Further Improvements

- For further improvements on MNIST (and on famous datasets in general see `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`
- *Noteworthy*:
  - See D.C. Ciresan, U. Meier, L.M. Gambardella, J. Schmidhuber, "Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition", `https://arxiv.org/abs/1003.0358`
  - Fully connected network, without convolutional layers that achieves 99.65% accuracy.
  - Training for that non-convolutional network proceeds very slow, however.

Initializing Weights

Weight Initialization

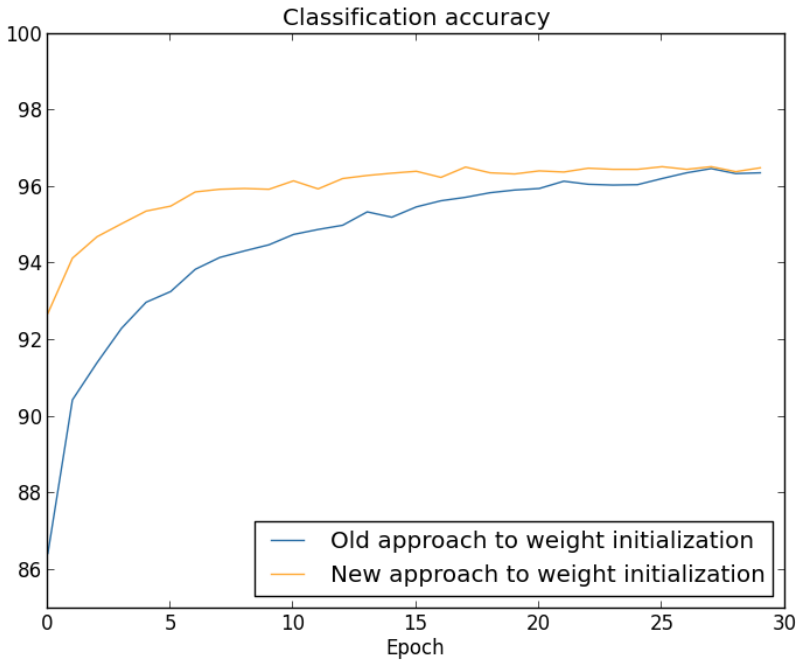
- Draw weights from normal distribution $\mathcal{N}_{0,\sigma}$ with mean 0 and standard deviation σ
- Let n_{in} be the number of inputs to the next node:

$$z = \sum_{j=1}^{n_{\text{in}}} x_j w_j + b \tag{4}$$

- Then $\sigma := \sqrt{n_{\text{in}}}$, so sample weights from $\mathcal{N}_{0,\sqrt{n_{\text{in}}}}$
- *Explanation*: So, input z to next node will (roughly) be sampled from $\mathcal{N}_{0,1}$

Weight Initialization

Consequence



- Improved initialization leads to faster learning
- See further ["Practical Recommendations for Gradient-Based Training of Deep Architectures", Bengio, 2012] (`https://arxiv.org/pdf/1206.5533v2.pdf`) for more details.

Choosing Hyperparameters

Choosing Hyperparameters

Hyperparameters to be determined:

- Number (and composition) of hidden neurons
- In stochastic gradient descent: mini-batch size
- Number of epochs in training
- Learning rate η
- Regularization parameter λ
- If chosen inappropriately
  ☞ random exploration of search space
  ☞ no training will take place

Choosing Hyperparameters

Hyperparameters to be determined:

- Number (and composition) of hidden neurons
- In stochastic gradient descent: mini-batch size
- Number of epochs in training
- Learning rate η
- Regularization parameter λ
- If chosen inappropriately
  ☞ random exploration of search space
  ☞ no training will take place

Choosing Hyperparameters

General Strategy

First Challenge

Establish *any* non-trivial learning, that is, train a network that classifies better than chance

Strategies

- Turn multi-class problem into binary problem
- Start experimenting with the simplest possible architecture
- Small batch size: monitor changes in classification accuracy after each batch
- Check that *weight decay* (see (26), Lecture 5) is constant with respect to number of training data ☞ affects both η and λ

Choosing Hyperparameters

General Strategy

First Challenge

Establish *any* non-trivial learning, that is, train a network that classifies better than chance

Strategies

- Turn multi-class problem into binary problem
- Start experimenting with the simplest possible architecture
- Small batch size: monitor changes in classification accuracy after each batch
- Check that *weight decay* (see (26), Lecture 5) is constant with respect to number of training data ☞ affects both η and λ

Choosing Hyperparameters

General Strategy

First Challenge

Establish *any* non-trivial learning, that is, train a network that classifies better than chance

Strategies

- Turn multi-class problem into binary problem
- Start experimenting with the simplest possible architecture
- Small batch size: monitor changes in classification accuracy after each batch
- Check that *weight decay* (see (26), Lecture 5) is constant with respect to number of training data ☞ affects both η and λ

Choosing Hyperparameters

General Strategy

First Challenge

Establish *any* non-trivial learning, that is, train a network that classifies better than chance

Strategies

- Turn multi-class problem into binary problem
- Start experimenting with the simplest possible architecture
- Small batch size: monitor changes in classification accuracy after each batch
- Check that *weight decay* (see (26), Lecture 5) is constant with respect to number of training data ☞ affects both η and λ

Choosing Hyperparameters

General Strategy

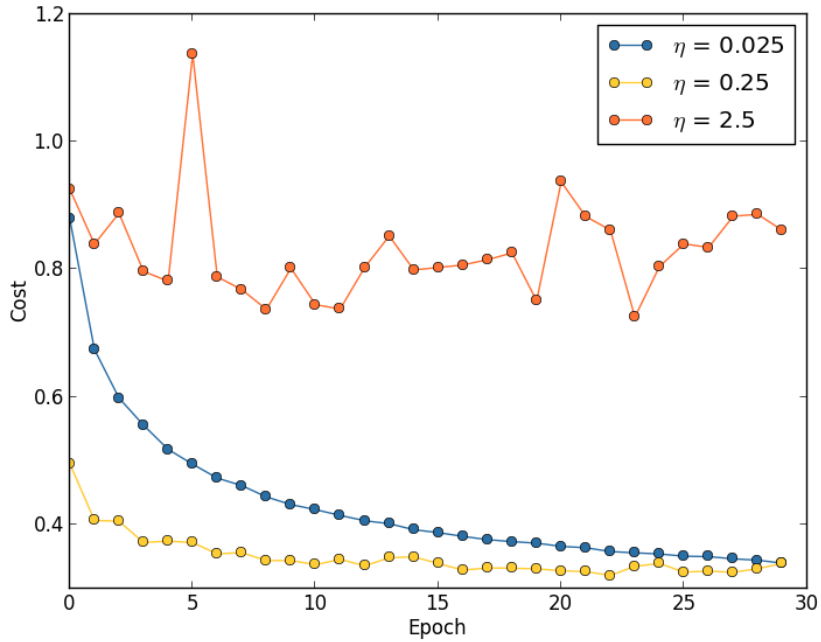
First Challenge

Establish *any* non-trivial learning, that is, train a network that classifies better than chance

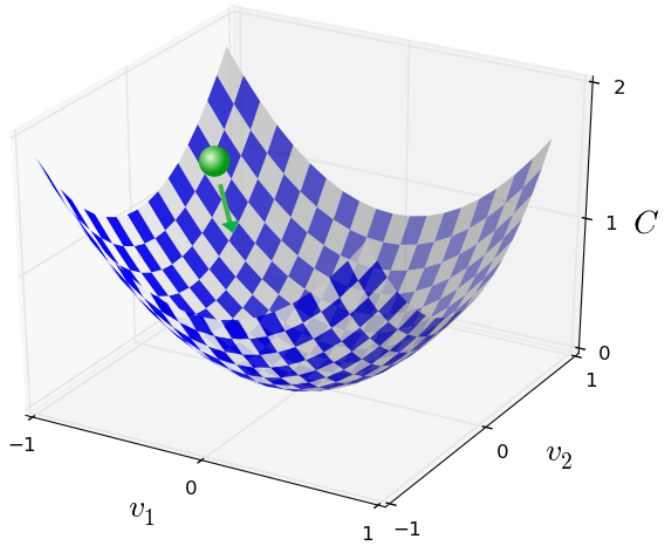
Strategies

- Turn multi-class problem into binary problem
- Start experimenting with the simplest possible architecture
- Small batch size: monitor changes in classification accuracy after each batch
- Check that *weight decay* (see (26), Lecture 5) is constant with respect to number of training data ☞ affects both η and λ

Choosing Learning Rate



Choosing Learning Rate



- Learning rate η too large: random oscillations ☞ parameters "jump across" optimum, back and forth
- Learning rate η too small: training too slow
- *Strategy*: Pick η as large as possible, while avoiding random oscillation
- To refine training, decrease η along epochs

Choosing Regularization Parameter

Suggestions

- Start with no regularization ($\lambda = 0$)
- Determine the learning rate η, as described above
- Then do $\lambda = 1.0$, and compare accuracy with $\lambda = 0$
- Depending on the outcome, multiply or divide by ten ($\lambda = 10.0$ or $\lambda = 0.1$)
- Once reached the right order of magnitude, finetune

Choosing Regularization Parameter

Suggestions

- Start with no regularization ($\lambda = 0$)
- Determine the learning rate η, as described above
- Then do $\lambda = 1.0$, and compare accuracy with $\lambda = 0$
- Depending on the outcome, multiply or divide by ten ($\lambda = 10.0$ or $\lambda = 0.1$)
- Once reached the right order of magnitude, finetune

Choosing Regularization Parameter

Suggestions

- Start with no regularization ($\lambda = 0$)
- Determine the learning rate η, as described above
- Then do $\lambda = 1.0$, and compare accuracy with $\lambda = 0$
- Depending on the outcome, multiply or divide by ten ($\lambda = 10.0$ or $\lambda = 0.1$)
- Once reached the right order of magnitude, finetune

Choosing Regularization Parameter

Suggestions

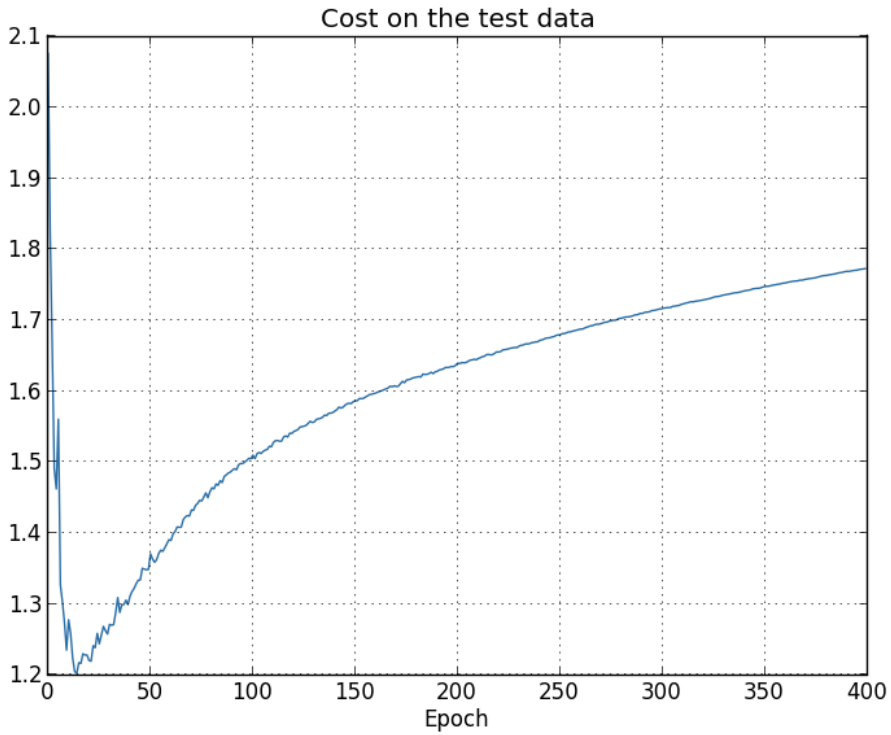
- Start with no regularization ($\lambda = 0$)
- Determine the learning rate η, as described above
- Then do $\lambda = 1.0$, and compare accuracy with $\lambda = 0$
- Depending on the outcome, multiply or divide by ten ($\lambda = 10.0$ or $\lambda = 0.1$)
- Once reached the right order of magnitude, finetune

Choosing Number of Epochs

- Use validation data; see earlier lectures for training, validation and test data. Validation is to be used for determining hyperparameters.
- Stop as soon as validation accuracy, the ratio of correctly classified validation data samples over the total number of validation data samples, no longer improves
- "No improvement-in-ten rule": stop 10 epochs after classification accuracy starts to stall

Choosing Number of Epochs

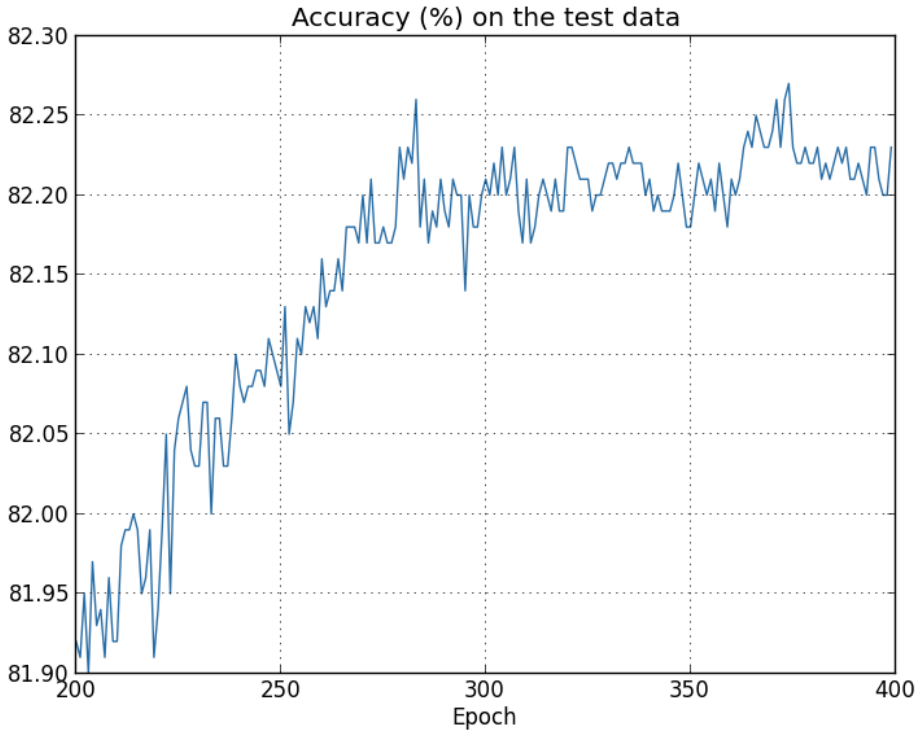
Cost versus Validation Accuracy



Validation accuracy (here: test accuracy) suggests to do ≈ 280 epochs

Choosing Number of Epochs

Cost versus Validation Accuracy

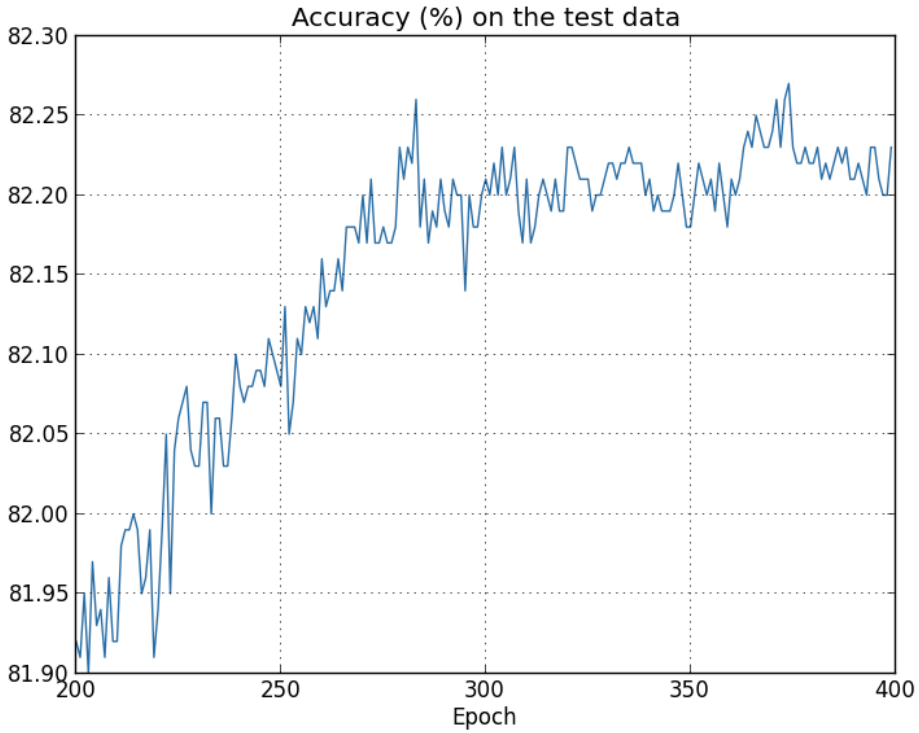


Validation cost (here: test cost) suggests to do ≈ 15 epochs

- What to do, use cost or validation accuracy to determine number of epochs?
- Cost has no meaning, while accuracy does: use accuracy!

Choosing Number of Epochs

Cost versus Validation Accuracy

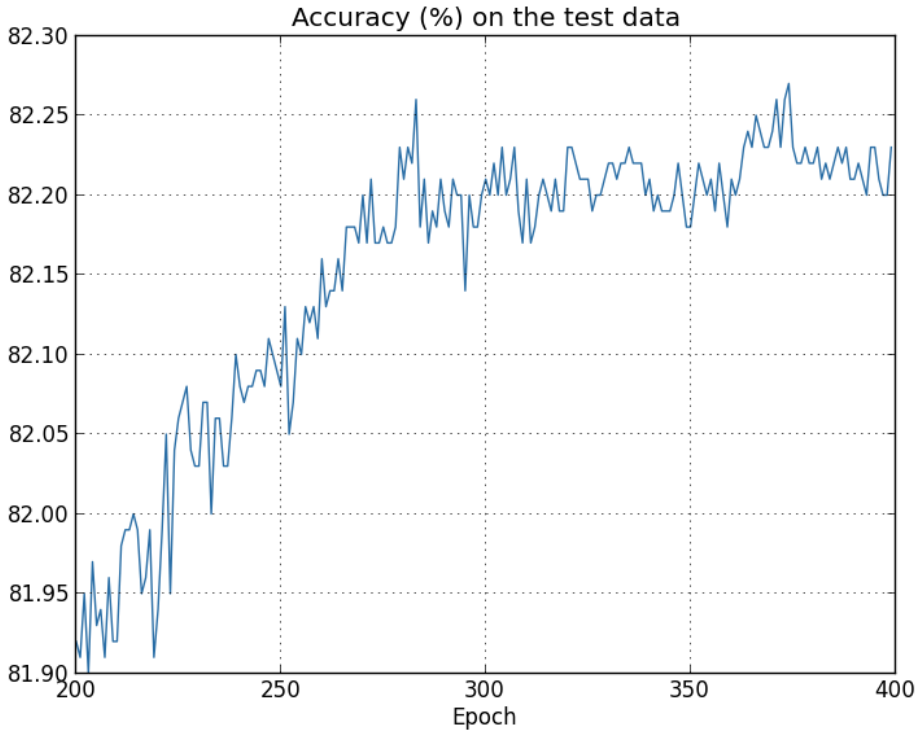


Validation cost (here: test cost) suggests to do ≈ 15 epochs

- What to do, use cost or validation accuracy to determine number of epochs?
- Cost has no meaning, while accuracy does: use accuracy!

Choosing Number of Epochs

Cost versus Validation Accuracy



Validation cost (here: test cost) suggests to do ≈ 15 epochs

- What to do, use cost or validation accuracy to determine number of epochs?
- Cost has no meaning, while accuracy does: use accuracy!

Choosing Mini-Batch Size

Prelude

See Problem "Fully matrix-based approach to backpropagation over a mini-batch" in chapter 2 of Nielsen's book, see `http://neuralnetworksanddeeplearning.com/chap2.html`:

- Instead of looping over each training example, computing the gradient for each of them, and averaging eventually, one can apply matrix-based techniques to do this computation in about half of the time.
- Consider updates for one training example versus averaging over one mini-batch of examples of size m:

$$w \leftarrow w - \eta \nabla_w C_x \quad \text{versus} \quad w \leftarrow w - \eta \frac{1}{m} \sum_x \nabla_w C_x \tag{5}$$

  That advocates against mini-batches however, because updates will be very small ☞ slow learning!

- Anything to do about this trade-off?

Choosing Mini-Batch Size

Solution

- *Observation*: Multiplying η by m yields

$$w \leftarrow w - \eta \sum_x \nabla_w C_x \tag{6}$$

  which looks like summing over all individual examples, so issue of too little, and too small updates when using mini-batches mended.

Summary

- *Mini-batch size too small*: One does not exploit the advantages of matrix computation libraries.
- *Mini-batch size too large*: Too little updates.
- *Overall solution*: Find a good trade-off!
- Mini-batch size is fairly independent of other parameters.
- So, first optimize other hyperparameters. Then tune mini-batch size scaling η according to (6).

Choosing Hyperparameters

Search Techniques

- *Grid Search*: Try combinations of hyperparameters, viewing them as points of a grid, where each dimension refers to one of the hyperparameters
  - See `http://www.deeplearningbook.org/` 11.4.3 for details
- *Random Search*: Randomly pick combinations of hyperparameters, selected according to reasonable probability distributions
  - See `http://www.deeplearningbook.org/` 11.4.4 for details
- *Model-Based Hyperparameter Optimization*: Cast selection of hyperparameters as optimization problem, and try to pick hyperparameters that yield minimal error on validation data
  - See `http://www.deeplearningbook.org/` 11.4.5 for details
  - And the following slides for further information on automated optimization strategies

Choosing Hyperparameters

Guidelines for Automated Techniques

Automated Techniques

- ["Random search for hyper-parameter optimization", Bergstra & Bengio, 2012; https://dl.acm.org/citation.cfm?id=2188395]
- ["Practical Bayesian optimization of machine learning algorithms", Snoek, Larochelle & Adams, 2012; http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf]

Choosing Hyperparameters

Guidelines for Automated Techniques

Automated Techniques

- Currently hot topic: "Auto Machine Learning (AutoML)", methods to pick optimal selections of hyperparameters, in particular to pick optimal network architectures.
- See
  - `https://hackernoon.com/a-brief-overview-of-automatic-machine-learning-solutions-automl-2826c7807a2a`
  - `https://ai.googleblog.com/2017/05/using-machine-learning-to-explore.html`

  if interested
- *However*: usually very expensive in terms of compute resources

Practical Recommendations

Further Reading

- "Practical Recommendations for gradient-based training of deep architectures", Y. Bengio, 2012, see `https://arxiv.org/abs/1206.5533`
- "Efficient BackProp", Y. LeCun, L. Bottou, G. Orr, K.-R. Müller, 1998, see `http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`
- "Neural Networks: Tricks of the Trade", edited by G. Montavon, G. Orr, K.-R. Müller, see `https://www.springer.com/de/book/9783642352881` This book contains the above articles, is expensive, but many of the articles that appear in the book are freely available

Summary

- Convolutional Neural Networks
  - `http://www.deeplearningbook.org/`, Chapter 9
  - `http://neuralnetworksanddeeplearning.com/`, "Deep Learning"
- Choosing hyperparameters
  - `http://www.deeplearningbook.org/`, Chapter 11 (selected parts)
  - `http://neuralnetworksanddeeplearning.com/`, "Weight initialization" and "How to choose a network's hyperparameters?"

Thanks for your attention