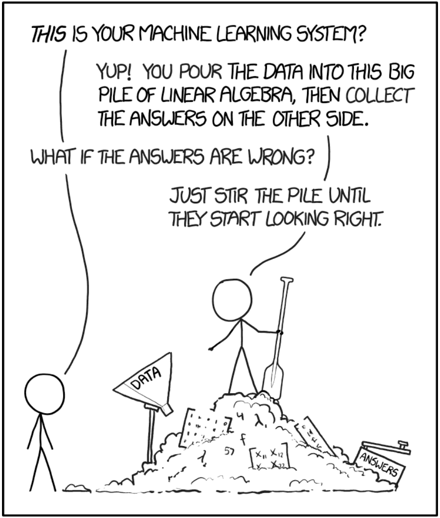


Source: https://xkcd.com/1838/

Introduction to Feed-forward and Convolutional Neural Networks

Adrian Braemer

23.04.2019

Outline

What is Machine Learning?

- Goal: Make predictions based on given data

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T
  - Dataset $\mathbf{X} = (\mathbf{x}_1, ...)$ and $\mathbf{Y} = (y_1, ...)$ f. e. experimental data

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T
  - Dataset $\mathbf{X} = (\mathbf{x}_1, ...)$ and $\mathbf{Y} = (y_1, ...)$ f. e. experimental data
  - Model $f(\mathbf{x}; \mathbf{w}), f : \mathbf{x} \to y$ with parameters $\mathbf{w}$, f. e. $k^{w_1} m^{w_2}$

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T
  - Dataset $\mathbf{X} = (\mathbf{x}_1, \ldots)$ and $\mathbf{Y} = (y_1, \ldots)$ f. e. experimental data
  - Model $f(\mathbf{x}; \mathbf{w}), f : \mathbf{x} \to y$ with parameters $\mathbf{w}$, f. e. $k^{w_1} m^{w_2}$
  - Cost function $\mathcal{C}(\mathbf{Y}, f(\mathbf{X}; \mathbf{w}))$, f.e. L_2 norm $\sum_i [y_i - f(\mathbf{x}_i; \mathbf{w})]^2$

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T
  - Dataset $\mathbf{X} = (\mathbf{x}_1, ...)$ and $\mathbf{Y} = (y_1, ...)$ f. e. experimental data
  - Model $f(\mathbf{x}; \mathbf{w}), f : \mathbf{x} \to y$ with parameters $\mathbf{w}$, f. e. $k^{w_1} m^{w_2}$
  - Cost function $\mathcal{C}(\mathbf{Y}, f(\mathbf{X}; \mathbf{w}))$, f.e. L_2 norm $\sum_i [y_i - f(\mathbf{x}_i; \mathbf{w})]^2$

$\to$ Best parameters $\hat{\mathbf{w}} = \mathrm{argmin}_{\mathbf{w}}\ \mathcal{C}(\mathbf{Y}, f(\mathbf{X}; \mathbf{w}))$

What is Machine Learning?

- Goal: Make predictions based on given data
- Formalization:
  - Independent quantities $\mathbf{x}$, f. e. spring constant k, mass m
  - Dependent quantity y, f. e. period of pendulum T
  - Dataset $\mathbf{X} = (\mathbf{x}_1, ...)$ and $\mathbf{Y} = (y_1, ...)$ f. e. experimental data
  - Model $f(\mathbf{x}; \mathbf{w}), f : \mathbf{x} \to y$ with parameters $\mathbf{w}$, f. e. $k^{w_1} m^{w_2}$
  - Cost function $\mathcal{C}(\mathbf{Y}, f(\mathbf{X}; \mathbf{w}))$, f.e. L_2 norm $\sum_i [y_i - f(\mathbf{x}_i; \mathbf{w})]^2$

$\to$ Best parameters $\hat{\mathbf{w}} = \mathrm{argmin}_{\mathbf{w}} \, \mathcal{C}(\mathbf{Y}, f(\mathbf{X}; \mathbf{w}))$

$\to$ "Training the model"

Comparison to Fitting

Comparison to Fitting

Fitting:

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$

Learning:

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit

Learning:

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown $\mathbf{x}$

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown $\mathbf{x}$
- Split data set in training and test sets

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters **w**
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown **x**
- Split data set in training and test sets

$\rightarrow$ Cross validation

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters **w**
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown **x**
- Split data set in training and test sets

$\rightarrow$ Cross validation

- Test set accuracy is important!

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown $\mathbf{x}$
- Split data set in training and test sets

$\rightarrow$ Cross validation

- Test set accuracy is important!

$\rightarrow$ Regularization

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown $\mathbf{x}$
- Split data set in training and test sets

$\rightarrow$ Cross validation

- Test set accuracy is important!

$\rightarrow$ Regularization

Comparison to Fitting

Fitting:

- Goal: Best *estimation* of parameters $\mathbf{w}$
- Use all data to fit
- Minimize Cost function by all means

Learning:

- Goal: Best *prediction* for unknown $\mathbf{x}$
- Split data set in training and test sets

$\rightarrow$ Cross validation

- Test set accuracy is important!

$\rightarrow$ Regularization

$\rightarrow$ Subtle differences, very different algorithms!

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)
- Dependent quantity: Category $y \in 0, 1$

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)
- Dependent quantity: Category $y \in 0, 1$
- Linear model: $f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{x}^T \mathbf{w} + b)$

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)
- Dependent quantity: Category $y \in 0, 1$
- Linear model: $f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{x}^T \mathbf{w} + b)$
- Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)
- Dependent quantity: Category $y \in 0, 1$
- Linear model: $f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{x}^T \mathbf{w} + b)$
- Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$\rightarrow$ Model maps input to *probability*

Explicit Example: Image Classification

- Independent quantities: Pixel data $\mathbf{x}$ (flattened to a vector)
- Dependent quantity: Category $y \in 0, 1$
- Linear model: $f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{x}^T \mathbf{w} + b)$
- Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$\rightarrow$ Model maps input to *probability*

$\rightarrow$ $f(\mathbf{x}; \mathbf{w}, b) = P(y = 1 | \mathbf{x}; \mathbf{w}, b)$

Cost function for classification

- Idea: Maximize probability of correct classification

Cost function for classification

- Idea: Maximize probability of correct classification
- Remember: $P(y = 1|\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w} + b) = 1 - P(y = 0|\mathbf{x})$

Cost function for classification

- Idea: Maximize probability of correct classification
- Remember: $P(y=1|\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w}+b) = 1 - P(y=0|\mathbf{x})$
- Probability $P_{correct}$ that classification is correct:

$$P_{correct}(\mathbf{x},y) = \begin{cases} P(y=1|\mathbf{x}) & y=1 \\ 1-P(y=1|\mathbf{x}) & y=0 \end{cases}$$
$$= P(y=1|\mathbf{x})^y[1-P(y=1|\mathbf{x})]^{1-y}$$

Cost function for classification

- Idea: Maximize probability of correct classification
- Remember: $P(y=1|\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w}+b) = 1 - P(y=0|\mathbf{x})$
- Probability $P_{correct}$ that classification is correct:

$$
\begin{aligned}
P_{correct}(\mathbf{x}, y) &= \begin{cases} P(y=1|\mathbf{x}) & y=1 \\ 1-P(y=1|\mathbf{x}) & y=0 \end{cases} \\
&= P(y=1|\mathbf{x})^y[1-P(y=1|\mathbf{x})]^{1-y}
\end{aligned}
$$

- For multiple predictions: $P_{correct}(\mathbf{X}, \mathbf{Y}) = \prod_i P_{correct}(\mathbf{x}_i, y_i)$

Cost function for classification

- Idea: Maximize probability of correct classification
- Remember: $P(y=1|\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w}+b) = 1 - P(y=0|\mathbf{x})$
- Probability $P_{correct}$ that classification is correct:

$$P_{correct}(\mathbf{x},y) = \begin{cases} P(y=1|\mathbf{x}) & y=1 \\ 1-P(y=1|\mathbf{x}) & y=0 \end{cases}$$
$$= P(y=1|\mathbf{x})^y[1-P(y=1|\mathbf{x})]^{1-y}$$

- For multiple predictions: $P_{correct}(\mathbf{X},\mathbf{Y}) = \prod_i P_{correct}(\mathbf{x}_i, y_i)$
- Cross entropy:

$$-\sum_i y_i \log\left[\sigma(\mathbf{x}_i^T\mathbf{w}+b)\right] + (1-y_i)\log\left[1-\sigma(\mathbf{x}_i^T\mathbf{w}+b)\right]$$

Practical Example: MNIST

- Dataset of 70.000 handwritten digits

Practical Example: MNIST

- Dataset of 70.000 handwritten digits
- Commonly used for machine learning experiments

Practical Example: MNIST

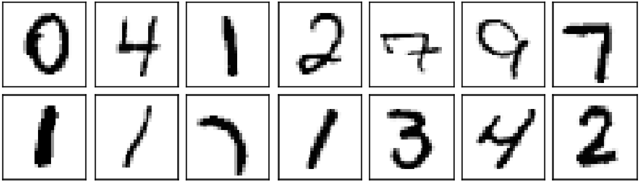
- Dataset of 70.000 handwritten digits
- Commonly used for machine learning experiments
- Lowest error rate 0,21% (CNN)

Practical Example: MNIST

- Dataset of 70.000 handwritten digits
- Commonly used for machine learning experiments
- Lowest error rate 0,21% (CNN)

Practical Example: MNIST

- Dataset of 70.000 handwritten digits
- Commonly used for machine learning experiments
- Lowest error rate 0,21% (CNN)



MNIST classifier

- Cross entropy: only 2 output states

MNIST classifier

- Cross entropy: only 2 output states

$\rightarrow$ Take 1 classifier for every class

MNIST classifier

- Cross entropy: only 2 output states

$\rightarrow$ Take 1 classifier for every class

- Probabilities don't add to 1

MNIST classifier

- Cross entropy: only 2 output states

$\rightarrow$ Take 1 classifier for every class

- Probabilities don't add to 1

$\rightarrow$ Take SoftMax (= Boltzmann distribution) function

$$x_i \mapsto \frac{e^{x_i}}{\sum_j e^{x_j}}$$

MNIST classifier

- Cross entropy: only 2 output states

→ Take 1 classifier for every class

- Probabilities don't add to 1

→ Take SoftMax (= Boltzmann distribution) function

$$x_i \mapsto \frac{e^{x_i}}{\sum_j e^{x_j}}$$

MNIST classifier

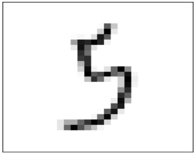
- Cross entropy: only 2 output states

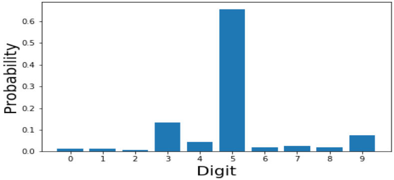
→ Take 1 classifier for every class

- Probabilities don't add to 1

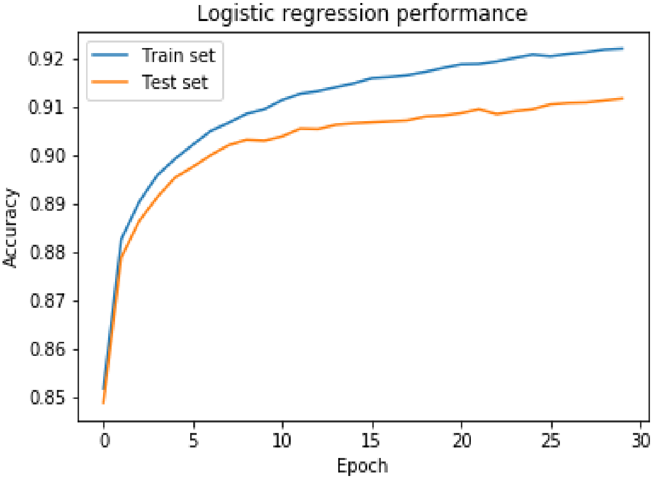
→ Take SoftMax (= Boltzmann distribution) function

$$x_i \mapsto \frac{e^{x_i}}{\sum_j e^{x_j}}$$





MNIST classifier



→ Caps out at around 91%

Can we do better with a more complex model?

Neural Networks

Neural Networks

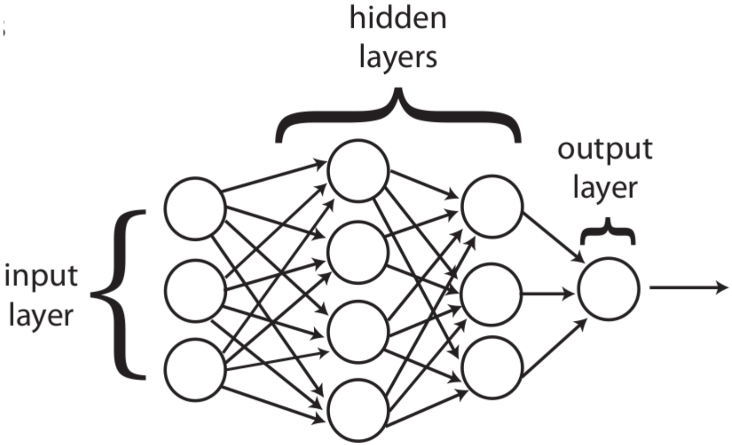


Figure: General architecture

How to train a NN

- Want to minimize $\mathcal{C}(X; \mathbf{w})$ w. r. t. $\mathbf{w}$

How to train a NN

- Want to minimize $\mathcal{C}(X;\mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:

How to train a NN

- Want to minimize $\mathcal{C}(X;\mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:
    1. Choose some initial $\mathbf{w}_0$

How to train a NN

- Want to minimize $\mathcal{C}(X; \mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:
  1. Choose some initial $\mathbf{w}_0$
  2. Compute gradient $\mathbf{v}_i = \nabla \mathcal{C}(X; \mathbf{w}_i)$

How to train a NN

- Want to minimize $\mathcal{C}(X; \mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:
  1. Choose some initial $\mathbf{w}_0$
  2. Compute gradient $\mathbf{v}_i = \nabla\mathcal{C}(X; \mathbf{w}_i)$
  3. Update weights $\mathbf{w}_{i+1} = \mathbf{w}_i - \eta\mathbf{v}_i$ where η is the learning rate

How to train a NN

- Want to minimize $\mathcal{C}(X; \mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:
  1. Choose some initial $\mathbf{w}_0$
  2. Compute gradient $\mathbf{v}_i = \nabla \mathcal{C}(X; \mathbf{w}_i)$
  3. Update weights $\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \mathbf{v}_i$ where η is the learning rate
  4. Repeat until converged to minimum

How to train a NN

- Want to minimize $\mathcal{C}(X; \mathbf{w})$ w. r. t. $\mathbf{w}$
- Gradient Descent:
  1. Choose some initial $\mathbf{w}_0$
  2. Compute gradient $\mathbf{v}_i = \nabla\mathcal{C}(X; \mathbf{w}_i)$
  3. Update weights $\mathbf{w}_{i+1} = \mathbf{w}_i - \eta\mathbf{v}_i$ where η is the learning rate
  4. Repeat until converged to minimum

$\rightarrow$ Problems at every step!

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

$\rightarrow$ Backpropagation

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

$\rightarrow$ Backpropagation

- How to choose learning rate η?

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X; \mathbf{w})$?

$\rightarrow$ Backpropagation

- How to choose learning rate η?

$\rightarrow$ Need to tune

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

→ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

→ Backpropagation

- How to choose learning rate η?

→ Need to tune

- How do we know we have the correct minimum?

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

$\rightarrow$ Backpropagation

- How to choose learning rate η?

$\rightarrow$ Need to tune

- How do we know we have the correct minimum?

$\rightarrow$ We don't

Simple algorithm - many problems

- How to choose $\mathbf{w}_0$?

$\rightarrow$ Zero-mean, normal-distributed values work well enough in most cases

- How to compute $-\nabla\mathcal{C}(X;\mathbf{w})$?

$\rightarrow$ Backpropagation

- How to choose learning rate η?

$\rightarrow$ Need to tune

- How do we know we have the correct minimum?

$\rightarrow$ We don't

- Do we even converge?

Backpropagation Algorithm

Backpropagation Algorithm

- Weigths w_{jk}^{l} between j-th neuron in layer $l-1$ and k-th neuron in layer l

Backpropagation Algorithm

- Weigths w_{jk}^{l} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b_{j}^{l}

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$

Backpropagation Algorithm

- Weigths w_{jk}^l between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b_j^l
- Weighted inputs to neuron $z_k^l = \sum_j w_{jk}^l a_j^{l-1} + b_k^l$
- Activation levels a_k^l:
  - a_k^1 are the inputs
  - $a_k^l = \sigma(z_k^l) = \sigma\left(\sum_j w_{jk}^l a_j^{l-1} + b_k^l\right)$
- Error $\Delta_k^l = \frac{\partial \mathcal{C}}{\partial z_k^l}$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k} \frac{\partial a^l_k}{\partial z^l_k}$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\frac{\partial a^l_k}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\sigma'(z^l_k)$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\frac{\partial a^l_k}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\sigma'(z^l_k)$
- $\Delta^l_k = \sum_k \frac{\partial \mathcal{C}}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j}$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k} \frac{\partial a^l_k}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k} \sigma'(z^l_k)$
- $\Delta^l_k = \sum_k \frac{\partial \mathcal{C}}{\partial z^{l+1}_k} \frac{\partial z^{l+1}_k}{\partial z^l_j} = \sigma'(z^l_j)\left(\sum_k w^{l+1}_{jk} \Delta^{l+1}_k\right)$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\frac{\partial a^l_k}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\sigma'(z^l_k)$
- $\Delta^l_k = \sum_k \frac{\partial \mathcal{C}}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} = \sigma'(z^l_j)\left(\sum_k w^{l+1}_{jk}\Delta^{l+1}_k\right)$
- Gradient: $\frac{\partial \mathcal{C}}{\partial w^l_{jk}}$

Backpropagation Algorithm

- Weigths w^l_{jk} between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b^l_j
- Weighted inputs to neuron $z^l_k = \sum_j w^l_{jk} a^{l-1}_j + b^l_k$
- Activation levels a^l_k:
  - a^1_k are the inputs
  - $a^l_k = \sigma(z^l_k) = \sigma\left(\sum_j w^l_{jk} a^{l-1}_j + b^l_k\right)$
- Error $\Delta^l_k = \frac{\partial \mathcal{C}}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\frac{\partial a^l_k}{\partial z^l_k} = \frac{\partial \mathcal{C}}{\partial a^l_k}\sigma'(z^l_k)$
- $\Delta^l_k = \sum_k \frac{\partial \mathcal{C}}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} = \sigma'(z^l_j)\left(\sum_k w^{l+1}_{jk}\Delta^{l+1}_k\right)$
- Gradient: $\frac{\partial \mathcal{C}}{\partial w^l_{jk}} = \frac{\partial \mathcal{C}}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}}$

Backpropagation Algorithm

- Weigths w_{jk}^l between j-th neuron in layer $l-1$ and k-th neuron in layer l
- Biases b_j^l
- Weighted inputs to neuron $z_k^l = \sum_j w_{jk}^l a_j^{l-1} + b_k^l$
- Activation levels a_k^l:
    - a_k^1 are the inputs
    - $a_k^l = \sigma(z_k^l) = \sigma\left(\sum_j w_{jk}^l a_j^{l-1} + b_k^l\right)$
- Error $\Delta_k^l = \frac{\partial \mathcal{C}}{\partial z_k^l} = \frac{\partial \mathcal{C}}{\partial a_k^l}\frac{\partial a_k^l}{\partial z_k^l} = \frac{\partial \mathcal{C}}{\partial a_k^l}\sigma'(z_k^l)$
- $\Delta_k^l = \sum_k \frac{\partial \mathcal{C}}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sigma'(z_j^l)\left(\sum_k w_{jk}^{l+1}\Delta_k^{l+1}\right)$
- Gradient: $\frac{\partial \mathcal{C}}{\partial w_{jk}^l} = \frac{\partial \mathcal{C}}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1}$

Backpropagation Algorithm

1. Calculate the activation levels z_k^l and a_k^l iteratively from front to back

Backpropagation Algorithm

1. Calculate the activation levels z_k^l and a_k^l iteratively from front to back
2. Find the error at top level $\Delta_k^L = \frac{\partial \mathcal{C}}{\partial a_k^L}\sigma'(z_k^L)$

Backpropagation Algorithm

1. Calculate the activation levels z_k^l and a_k^l iteratively from front to back
2. Find the error at top level $\Delta_k^L = \frac{\partial \mathcal{C}}{\partial a_k^L}\sigma'(z_k^L)$
3. Propagate errors backwards $\Delta_k^l = \sigma'(z_j^l)\left(\sum_k w_{jk}^{l+1}\Delta_k^{l+1}\right)$

Backpropagation Algorithm

1. Calculate the activation levels z_k^l and a_k^l iteratively from front to back
2. Find the error at top level $\Delta_k^L = \frac{\partial \mathcal{C}}{\partial a_k^L} \sigma'(z_k^L)$
3. Propagate errors backwards $\Delta_k^l = \sigma'(z_j^l) \left(\sum_k w_{jk}^{l+1} \Delta_k^{l+1} \right)$
4. Put everything together to find the gradient $\frac{\partial \mathcal{C}}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1}$

Backpropagation Algorithm

1. Calculate the activation levels z_k^l and a_k^l iteratively from front to back
2. Find the error at top level $\Delta_k^L = \frac{\partial \mathcal{C}}{\partial a_k^L} \sigma'(z_k^L)$
3. Propagate errors backwards $\Delta_k^l = \sigma'(z_j^l) \left(\sum_k w_{jk}^{l+1} \Delta_k^{l+1} \right)$
4. Put everything together to find the gradient $\frac{\partial \mathcal{C}}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1}$

$\rightarrow$ problem of vanishing or exploding gradients

Repairing the gradient

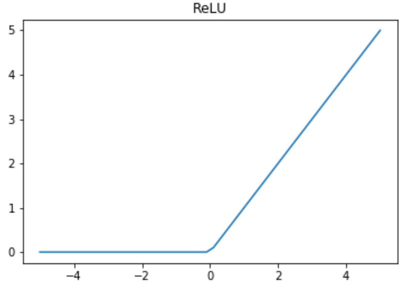
Repairing the gradient

- Truncate too high values

Repairing the gradient

- Truncate too high values
- Use non-saturating activation functions, f. e. ReLU (rectified linear unit)

$$\sigma(x) = \max(0, x)$$

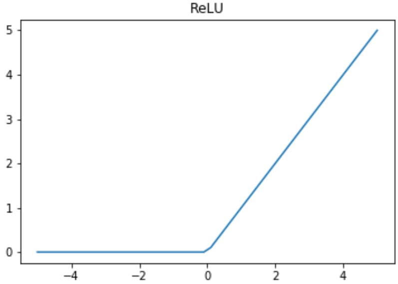


Repairing the gradient

- Truncate too high values
- Use non-saturating activation functions, f. e. ReLU (rectified linear unit)

$$\sigma(x) = \max(0, x)$$

- Regularization also helps



Regularization

- Stochastic Gradient Descent

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch ($=$ epoch)
- Dropout

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)
- Dropout
  - Randomly neglect neurons during training steps

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)
- Dropout
  - Randomly neglect neurons during training steps
- Batch normalization

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)
- Dropout
  - Randomly neglect neurons during training steps
- Batch normalization
  - Add 'Batch Normalization' layers

Regularization

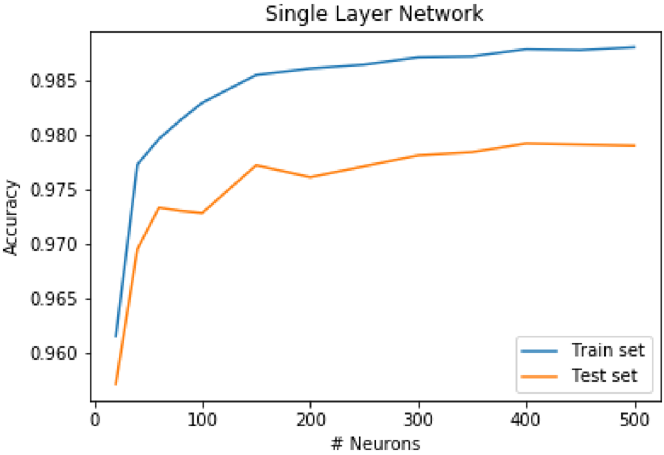
- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)
- Dropout
  - Randomly neglect neurons during training steps
- Batch normalization
  - Add 'Batch Normalization' layers
  - Standardize mean and variance between layers

Regularization

- Stochastic Gradient Descent
  - Randomly divide training data into m minibatches
  - Train on each minibatch (= epoch)
- Dropout
  - Randomly neglect neurons during training steps
- Batch normalization
  - Add 'Batch Normalization' layers
  - Standardize mean and variance between layers

→ Prevent overfitting

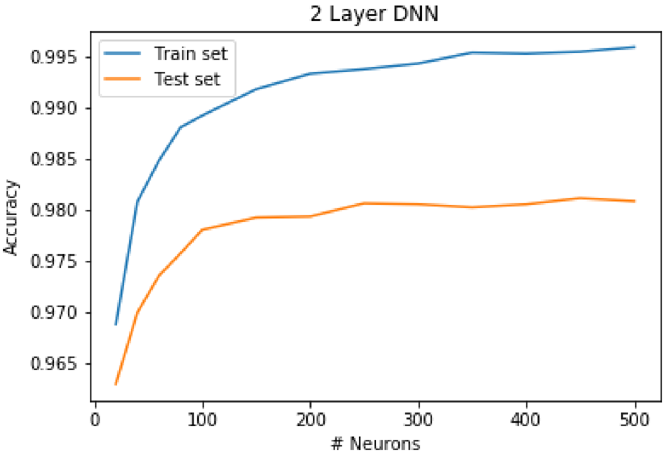
MNIST example revisited



→

Caps at 97.9%!

MNIST example revisited



→

Caps at 98.1%!

Weakpoints of DNNs

- No spatial structure

Weakpoints of DNNs

- No spatial structure
- Do not scale well with input size

Weakpoints of DNNs

- No spatial structure
- Do not scale well with input size

$\longrightarrow$ Can we reduce the network size?

Basic Idea

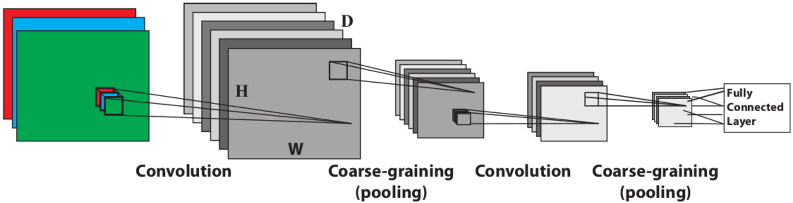
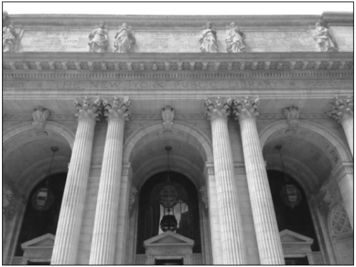
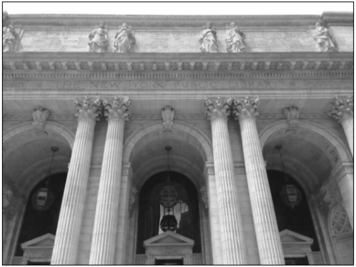


Figure: General Structure of CNNs

Example: Edge detection

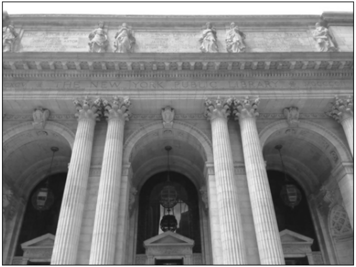


Example: Edge detection



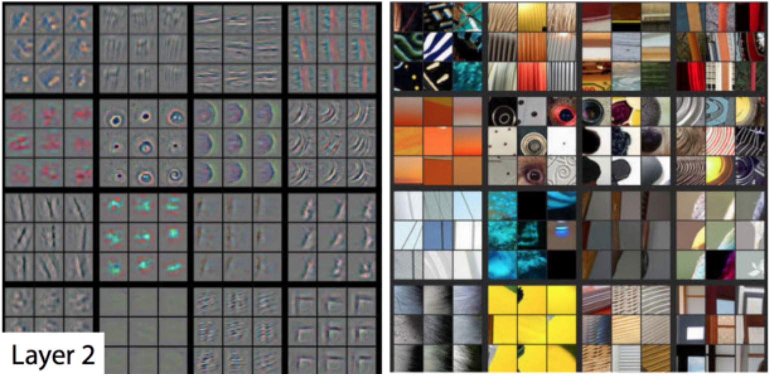
$$\xrightarrow{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}}$$

Example: Edge detection

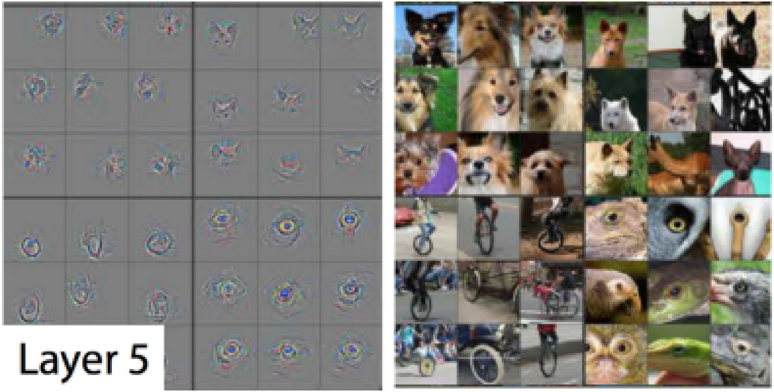


$$\xrightarrow{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}}$$

Low level convolutions



High level convolutions



Summary

- Basic Concepts of ML

Summary

- Basic Concepts of ML
- Motivation and Structure of DNNs

Summary

- Basic Concepts of ML
- Motivation and Structure of DNNs
- How to train a neural network

Summary

- Basic Concepts of ML
- Motivation and Structure of DNNs
- How to train a neural network
- Short outlook to CNNs

Discussion/Question