Deep Learning as an Engineer: The nuts and bolts and dirty tricks

Jan Schlüter
OFAI, Vienna, Austria
September 11, 2017
Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice
Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice
Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice
Application examples
Application examples
Nonlinear regression

**Task:** Predict at what force a concrete cylinder bursts, depending on component quantities and age

<table>
<thead>
<tr>
<th>Component</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>cement</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>blast furnace slag</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>fly ash</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>water</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>superplasticizer</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>coarse aggregate</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>fine aggregate</td>
<td>... kg/m³</td>
</tr>
<tr>
<td>age</td>
<td>... days</td>
</tr>
<tr>
<td>compressive strength</td>
<td>?? MPa</td>
</tr>
</tbody>
</table>
**Task:** Distinguish grayscale photographs of chihuahuas and blueberry muffins
Categorical image classification

**Task:** Recognize hand-written digits

**Task:** Recognize photographed objects
  (with a fixed set of possible answers)
Task: Create colored image from grayscale image

**Task:** Create colored image from scratch (possibly domain-specific)

Task: Create text from scratch (possibly domain-specific)

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.
Task: Create text from scratch (possibly domain-specific)

```c
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

May 2015: The Unreasonable Effectiveness of RNNs, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Acoustic event detection

**Task:** Detect boundaries between different parts of a music piece (e.g., verse \(\rightarrow\) chorus)
Basic ideas behind deep learning
Basic ideas behind deep learning
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss
Formalize task: regression

**Task:** Predict at what force a concrete cylinder bursts, depending on component quantities and age

**Solution form:** $y = f(x)$

**Input $x$:** 8-dimensional vector

**Output $y$:** scalar

<table>
<thead>
<tr>
<th>Component</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>cement</td>
<td>kg/m³</td>
</tr>
<tr>
<td>blast furnace slag</td>
<td>kg/m³</td>
</tr>
<tr>
<td>fly ash</td>
<td>kg/m³</td>
</tr>
<tr>
<td>water</td>
<td>kg/m³</td>
</tr>
<tr>
<td>superplasticizer</td>
<td>kg/m³</td>
</tr>
<tr>
<td>coarse aggregate</td>
<td>kg/m³</td>
</tr>
<tr>
<td>fine aggregate</td>
<td>kg/m³</td>
</tr>
<tr>
<td>age</td>
<td>days</td>
</tr>
<tr>
<td>compressive strength</td>
<td>MPa</td>
</tr>
</tbody>
</table>
Task: Distinguish grayscale photographs of chihuahuas and blueberry muffins

Solution form: $y = f(X)$
Input $X$: matrix of gray values
Output $y$: scalar “muffinness”

$X \in [0,1]^{236 \times 236}$
$y \in [0,1]$
Formalize task: categorical image classification

**Task:** Recognize hand-written digits

**Solution form:** $y = f(X)$

**Input $X$:** matrix of gray values

**Output $y$:** vector of class probabilities

$$X \in [0,1]^{28 \times 28}$$

$$y \in [0,1]^{10}; \sum_i y_i = 1.0$$

(1,0,0,0, ... 0)

**Task:** Recognize photographed objects (with a fixed set of possible answers)

**Solution form:** $y = f(X)$

**Input $X$:** 3-tensor of RGB values

**Output $y$:** vector of class probabilities

$$X \in [0,1]^{3 \times 32 \times 32}$$

$$y \in [0,1]^{10}; \sum_i y_i = 1.0$$

(0,0,1,0, ... 0)
Formalize task: image colorization

**Task:** Create colored image from grayscale image

**Solution form:** \( Y = f(X) \)

**Input \( X \):** matrix of gray values

**Output \( Y \):** 3-tensor of RGB values

Formalize task: image generation

**Task:** Create colored image from scratch (possibly domain-specific)

**Solution form:** \( Y = f(x) \)

**Input** \( x \): vector of random values  
(0.392, -0.124, ...) \( x \in \mathbb{R}^{100} \)

**Output** \( Y \): 3-tensor of RGB values  
\( Y \in [0,1]^{3 \times 128 \times 128} \)

Formalize task: text generation

**Task:** Create text from scratch (possibly domain-specific)

**Solution form:** $y, h' = f(x, h)$

**Input $x$:** vector encoding of seed or previously emitted character

**Input $h$:** vector of initial or previously emitted internal state

**Output $y$:** vector of next character probabilities

**Output $h'$:** vector of next internal state

May 2015: The Unreasonable Effectiveness of RNNs, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
**Task:** Detect boundaries between different parts of a music piece (e.g., verse → chorus)

**Solution form:** $y = f(X)$

**Input $X$:** magnitude spectrogram excerpt  

**Output $y$:** scalar “boundariness” of excerpt center

**Prediction process:** apply $f(X)$ to overlapping excerpts, pick peaks

ISMIR 2014: Boundary detection in music structure analysis using convolutional neural networks

$x \in \mathbb{R}^{115 \times 80}$  

"1.0"  

$y \in [0,1]$
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[ Y = f(X) \]
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[ Y = f(X; \theta) \]
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta) \\
l = L(\theta; f)
\]
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

\[ Y = f(X; \theta) \]

\[ l = L(\theta; f, D) \]
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta) \\
l = L(\theta; f, D) = \sum_{(X,T) \in D} J(f(X; \theta), T)
\]
How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta)
\]

\[
l = L(\theta; f, D) = \sum_{(X, T) \in D} J(f(X; \theta), T)
\]

\[
\theta^* = \min_\theta L(\theta; f, D)
\]
How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[ Y = f(X; \theta) \]

**Design choice:** make \( f \) deep (= a composition of multiple nonlinear functions), often an artificial neural network
What are Artificial Neural Networks?

“a simulation of a small brain”
What are Artificial Neural Networks?

“a simulation of a small brain”
What are Artificial Neural Networks?

a fancy name for a family of functions, including:

\[ y = \sigma(b + w^T x) \]  

(equivalent to logistic regression)
What are Artificial Neural Networks?

A fancy name for a family of functions, including:

\[ y = \sigma(b + \mathbf{w}^T \mathbf{x}) \]

(expression can be visualized as a graph:)

Output value is computed as a weighted sum of its inputs,

\[ b + \mathbf{w}^T \mathbf{x} = b + \sum_i w_i x_i \]

followed by a nonlinear function.
What are Artificial Neural Networks?

A fancy name for a family of functions, including:

\[ y = \sigma(b + W^T x) \]  

(multiple logistic regressions)

Expression can be visualized as a graph:

Output values are computed as weighted sums of their inputs,

\[ b + W^T x = b_j + \sum_i w_{ij} x_i \]

followed by a nonlinear function.
What are Artificial Neural Networks?

a fancy name for a family of functions, including:

\[ y = \sigma(b_2 + W_2^T \sigma(b_1 + W_1^T x)) \]  

(stacked logistic regressions)

expression can be visualized as a graph:

\[ x \quad b_1 + W_1^T x \quad h \quad b_2 + W_2^T h \quad y \]
What are Artificial Neural Networks?

a fancy name for a family of functions, including:

\[ y = \sigma(b_2 + W_2^T \sigma(b_1 + W_1^T x)) \]  
(stacked logistic regressions)

expression can be visualized as a graph:

Universal Approximation Theorem:
This can model any continuous function from \( \mathbb{R}^n \) to \( \mathbb{R}^m \) arbitrarily well (if \( h \) is made large enough).
Interlude: Why go any deeper than two layers?

A neural network with a single hidden layer of enough units can approximate any continuous function arbitrarily well. In other words, it can solve whatever problem you’re interested in! (Cybenko 1998, Hornik 1991)

But:

- “Enough units” can be a very large number. There are functions representable with a small, but deep network that would require exponentially many units with a single layer. (e.g., Hastad et al. 1986, Bengio & Delalleau 2011)

- The proof only says that a shallow network exists, it does not say how to find it. Evidence indicates that it is easier to train a deep network to perform well than a shallow one.
What are Artificial Neural Networks?

A fancy name for a family of functions, including:

\[ y = \sigma(b_2 + W_2^T \sigma(b_1 + W_1^T x)) \]  

(stacked logistic regressions)

Expression can be visualized as a graph:
What are Artificial Neural Networks?

a fancy name for a family of functions, including:

\[ y = \sigma(b_3 + W_3^T \sigma(b_2 + W_2^T \sigma(b_1 + W_1^T x))) \]

expression can be visualized as a graph:
What are Artificial Neural Networks?

A fancy name for a family of functions, including:

\[ f_{w,b}(x) = \sigma(b + W^T x) \]
\[ y = (f_{W_3,b_3} \circ f_{W_2,b_2} \circ f_{W_1,b_1})(x) \]

Expression can be visualized as a graph:

```
  x -> h_1 -> h_2 -> y
```

“dense layer” composed of simpler functions, commonly termed “layers”
Why dense layers are great

Fully-connected layer:
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.

**Consequence:** Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

**Example task:**
Distinguish *iris setosa, iris versicolour* and *iris virginica*

**Input:** (sepal length, sepal width, petal length, petal width)
**Equivalent:** (sepal width, petal length, sepal length, petal width)
Why dense layers are great

Fully-connected layer:
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.

**Consequence:** Swapping **two inputs** does not change the task. We can just swap the weights as well. (Or retrain the network.)

**Example task:**
Distinguish *iris setosa*, *iris versicolour* and *iris virginica*

**Input:** (sepal length, sepal width, petal length, petal width)

**Equivalent:** (sepal width, petal length, sepal length, petal width)
Why dense layers are great

Fully-connected layer:
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.

**Consequence:** Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

**Example task:**
Distinguish 3 and 6

**Input:**

```
3 3 3 3 6 6 6 6
3 3 3 3 6 6 6 6
```
Why dense layers are great - not so great

Fully-connected layer:
Each input is a scalar value, each weight is a scalar value, each output is the sum of all inputs multiplied by weights.

Consequence: Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

Example task:
Distinguish 3 and 6

Input:  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>6</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Equivalent:  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>6</th>
<th>6</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Convolutional layers

Fully-connected layer:
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of inputs **multiplied** by weights.

Convolutional layer:
Each **input** is a **tensor** (e.g., 2D), each **weight** is a **tensor**, each output is the sum of inputs **convolved** by weights.
Why convolutional layers are great

Convolutional layer:
Each **input** is a **tensor**, each **weight** is a **tensor**, each output is the sum of inputs **convolved** by weights.

Consequences:
- Input permutation does make a difference now
- Output retains the spatial layout of the input
- Can process large images with few learnable weights
- Weights are required to be applicable at every position
Pooling layers

A **pooling layer** downsamples a tensor.

Max pooling: keep the largest values of local patches

```
<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>3</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td>-2</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Average pooling: keep the mean values of local patches

```
<table>
<thead>
<tr>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Traditional Convolutional Neural Network

- **Convolutional layers**: local feature extraction
- **Pooling layers**: some translation invariance, data reduction
- **Fully-connected layers**: integrate information over full input
Traditional Convolutional Neural Network

\[ \begin{array}{cccccc}
X & \rightarrow & \text{conv} & \rightarrow & \text{pool} & \rightarrow \\
X & \rightarrow & \text{conv} & \rightarrow & \text{pool} & \rightarrow \\
X & \rightarrow & \text{pool} & \rightarrow & \text{conv} & \rightarrow \\
X & \rightarrow & \text{pool} & \rightarrow & \text{conv} & \rightarrow \\
X & \rightarrow & \text{dense} & \rightarrow & \text{dense} & \rightarrow \\
\end{array} \]

\[ y = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \end{pmatrix} \]

\[ y = 0.0 \]
Traditional Convolutional Neural Network

\[
\begin{align*}
X & \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \xrightarrow{\text{dense}} \xrightarrow{\text{dense}} = 0.0 \\
X & \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \xrightarrow{\text{dense}} \xrightarrow{\text{dense}} = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \ldots \\ 0.0 \end{pmatrix} \\
X & \xrightarrow{\text{conv}} \xrightarrow{\text{conv}} \xrightarrow{\text{conv}} \xrightarrow{\text{conv}} \xrightarrow{\text{conv}} \xrightarrow{\text{conv}} = \begin{pmatrix} \vdots \end{pmatrix}
\end{align*}
\]
How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[ Y = f(X; \theta) \]

**Design choice:** make \( f \) *deep* (= a composition of multiple nonlinear functions), often an artificial neural network
1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta)
\]

\[
l = L(\theta; f, D) = \Sigma_{(X, T) \in D} J(f(X; \theta), T)
\]
Penalty functions

\[ y = 0.21 \quad t = 0.0 \quad J(y, t) = -\log(y) \cdot t - \log(1-y) \cdot (1-t) \]

“binary cross-entropy”

\[ y = \begin{pmatrix} 0.6 \\ 0.0 \\ 0.1 \\ 0.0 \\ \vdots \\ 0.1 \end{pmatrix} \quad t = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \end{pmatrix} \quad J(y, t) = -\sum_{i} \log(y_i) \cdot t_i \]

“categorical cross-entropy”

\[ Y = \quad T = \]

\[ J(Y, T) = 0.5 \cdot \sum_{i,j,k} (Y_{i,j,k} - T_{i,j,k})^2 \]

“squared error”
How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta) \\
l = L(\theta; f, D) = \sum_{(X, T) \in D} J(f(X; \theta), T) \\
\theta^* = \min_\theta L(\theta; f, D)
\]
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. **find direction in which** \( L \) **decreases**
2. move \( \theta \) a bit into that direction
3. go to step 1
Find direction in which the loss decreases

\[
\begin{align*}
X & \xrightarrow{\text{conv}} H_1 \xrightarrow{\text{pool}} H_2 \xrightarrow{\text{dense}} h_3 \xrightarrow{\text{dense}} y = \\
& = \begin{pmatrix} 0.6 \\ 0.0 \\ 0.1 \\ 0.0 \\ 0.0 \\ 0.2 \\ 0.0 \\ 0.0 \\ 0.1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix} = t = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix}
\end{align*}
\]
Find direction in which the loss decreases

\[ X \xrightarrow{\text{conv}} H_1 \xrightarrow{\text{pool}} H_2 \xrightarrow{\text{dense}} h_3 \xrightarrow{\text{dense softmax}} z \rightarrow y = \begin{pmatrix} 0.6 \\ 0.0 \\ 0.1 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix} \]

\[ t = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
Find direction in which the loss decreases

\[
X \xrightarrow{\text{conv}} H_1 \xrightarrow{\text{pool}} H_2 \xrightarrow{\text{dense}} h_3 \xrightarrow{\text{dense}} z = t - y
\]

\[
\nabla z = t - y \\
\nabla b_3 = t - y
\]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
Find direction in which the loss decreases

\[ \nabla z = t - y \\
\nabla b_3 = t - y \\
\nabla W_3 = h_3 (t - y)^T \]

\[
\begin{pmatrix}
0.4 \\
0.0 \\
-0.1 \\
0.0 \\
0.0 \\
-0.2 \\
0.0 \\
0.0 \\
0.0 \\
-0.1
\end{pmatrix} = (t - y)^T
\]

\[ = h_3 \]
Find direction in which the loss decreases

\[ \begin{align*}
0.4 & 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 
&= (t - y)^T \\
0.9 & 0.36 \\
0.1 & 0.04 \\
0.3 & 0.12 \\
0.0 & 0.0 \\
1.0 & 0.4 \\
0.0 & 0.0 \\
\ldots & \ldots \\
= h_3 
\end{align*} \]

\[ \begin{align*}
\nabla z &= t - y \\
\nabla b_3 &= t - y \\
\nabla W_3 &= h_3 (t - y)^T 
\end{align*} \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]

0.4 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 = \( (t - y)^T \)

0.9 .36 .0
0.1 .04 .0
0.3 .12 .0
0.0 .0 .0
1.0 .4 .0
0.0 .0 .0
...
...
...

= \( h_3 \)
Find direction in which the loss decreases

\[ \begin{array}{cccccccccc}
0.4 & 0.0 & -0.1 & 0.0 & 0.0 & 0.0 & -0.2 & 0.0 & 0.0 & -0.1
\end{array} = (t - y)^T \\
\begin{array}{cccccccccc}
0.9 & 0.36 & 0.0 & -0.09 \\
0.1 & 0.04 & 0.0 & -0.01 \\
0.3 & 0.12 & 0.0 & -0.03 \\
0.0 & 0.0 & 0.0 \\
1.0 & 0.4 & 0.0 & -1.0 \\
0.0 & 0.0 & 0.0 \\
\end{array} \\
\cdots \quad \cdots \quad \cdots \\
= h_3

\nabla z = t - y \\
\nabla b_3 = t - y \\
\nabla W_3 = h_3 (t - y)^T
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_6 = ? \]
\[ \Delta z = J_6^T \Delta h_3 \]
\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_6 = ? \]

\[ \Delta z = J_6^T \Delta h_3 \]

\[ z = W_3^T h_3 + b_3 \]

\[ \nabla z = t - y \]

\[ \nabla b_3 = t - y \]

\[ \nabla W_3 = h_3 (t - y)^T \]

\[ \nabla z_3 = ? \]

\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_6 = W_3 \]

\[ \Delta z = J_6^T \Delta h_3 \]

\[ z = W_3^T h_3 + b_3 \]

\[ \nabla z = t - y \]

\[ \nabla b_3 = t - y \]

\[ \nabla W_3 = h_3 (t - y)^T \]

\[ \nabla z_3 = ? \]

\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_6 = W_3 \]
\[ \Delta z = J_6^T \Delta h_3 \]
\[ \nabla h_3 = J_6 \nabla z \]
\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_5 = ? \]

\[ \Delta h_3 = J_5^T \Delta z_3 \]

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ J_5 = ? \]

\[ \Delta h_3 = J_5^T \Delta z_3 \]

\[ (h_3)_i = \sigma((z_3)_i) \]

\[ \nabla z = t - y \]

\[ \nabla b_3 = t - y \]

\[ \nabla W_3 = h_3(t - y)^T \]

\[ \nabla z_3 = ? \]

\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ \Delta h_3 = J_5^T \Delta z_3 \]

\[ (h_3)_i = \sigma((z_3)_i) \]

\[ \nabla z = t - y \]

\[ \nabla b_3 = t - y \]

\[ \nabla W_3 = h_3 (t - y)^T \]

\[ \nabla z_3 = ? \]

\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ W_1 b_1 \rightarrow J_1 J_2 \rightarrow J_3 \rightarrow J_4 J_5 \rightarrow J_6 \]

\[ t - y \]

\[ J_5 = ? \]

\[ \Delta h_3 = J_5^T \Delta z_3 \]

\[ (h_3)_i = \sigma((z_3)_i) \]

\[ \nabla z = t - y \]

\[ \nabla b_3 = t - y \]

\[ \nabla W_3 = h_3(t - y)^T \]

\[ \nabla z_3 = ? \]

\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ \nabla z = \nabla b_3 = \nabla b_3 = t - y \\
\nabla W_3 = h_3(t - y)^T \\
\nabla z_3 = \sigma((z_3)_i) \\
\nabla Z_1 = ? \\
\]

\[ J_5 = \? \\
\Delta h_3 = J_5^T \Delta z_3 \\
(h_3)_i = \sigma((z_3)_i) \]
Find direction in which the loss decreases

\[ (J_5)_{i,i} = \sigma'(z_3)_i \]
\[ \Delta h_3 = J_5^T \Delta z_3 \]
\[ (h_3)_i = \sigma(z_3)_i \]
\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = ? \]
\[ \nabla Z_1 = ? \]
Find direction in which the loss decreases

\[ X \rightarrow W_1 b_1 \rightarrow J_1, J_2 \rightarrow J_3 \rightarrow W_2 b_2 \rightarrow J_4, J_5 \rightarrow W_3 b_3 \rightarrow J_6 \rightarrow z \]

\[ z_3 = t - y \]

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = ? \]

\[ (J_5)_{i,i} = \sigma'((z_3)_i) \]

\[ \Delta h_3 = J_5^T \Delta z_3 \]

\[ \nabla z_3 = J_5 \nabla h_3 \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \nabla z = t - y \\
\nabla b_3 = t - y \\
\nabla W_3 = h_3(t - y)^T \\
\nabla z_3 = J_5 J_6 (t - y) \\
\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \mathbf{W}_1 \mathbf{b}_1 \rightarrow \mathbf{J}_1 \mathbf{J}_2 \rightarrow \mathbf{J}_3 \rightarrow \mathbf{J}_4 \mathbf{J}_5 \rightarrow \mathbf{J}_6 \rightarrow \mathbf{z} \]

\[ \mathbf{W}_3 \mathbf{b}_3 \rightarrow \begin{pmatrix} 0.4 \\ 0 \\ -0.1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.2 \\ 0 \end{pmatrix} = \mathbf{t} - \mathbf{y} \]

\[ \nabla \mathbf{z} = \mathbf{t} - \mathbf{y} \]
\[ \nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y} \]
\[ \nabla \mathbf{W}_3 = \mathbf{h}_3 (\mathbf{t} - \mathbf{y})^T \]
\[ \nabla \mathbf{z}_3 = \mathbf{J}_5 \mathbf{J}_6 (\mathbf{t} - \mathbf{y}) \]
\[ \nabla \mathbf{Z}_1 = \mathbf{J}_2 \mathbf{J}_3 \mathbf{J}_4 \mathbf{J}_5 \mathbf{J}_6 (\mathbf{t} - \mathbf{y}) \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[ \nabla \theta = - \frac{\partial}{\partial \theta} J(f(X; \theta), t) \]
Find direction in which the loss decreases

\[ \nabla \theta = - \frac{\partial}{\partial \theta} J(f(X; \theta), t) \]

\[- \frac{\partial}{\partial \theta} L(\theta; f, D) = - \sum_{(x, t) \in D} \frac{\partial}{\partial \theta} J(f(X; \theta), t) \]
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_{\theta} L(\theta; f, D)$$

Iterative scheme:

0. initialize $\theta$ randomly
1. find direction in which $L$ decreases
2. move $\theta$ a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly

1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function

2. **Define model** as a generic solution with free parameters

3. **Define loss** function measuring how bad the solution is

4. **Optimize** model parameters to minimize loss

\[
Y = f(X; \theta)
\]

\[
l = L(\theta; f, D) = \sum_{(X, T) \in D} J(f(X; \theta), T)
\]

\[
\theta^* = \min_\theta L(\theta; f, D)
\]
Basic ideas behind deep learning
Deep learning in practice
Deep learning in practice

Optimization
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:
0. initialize \( \theta \) randomly
1. **find direction in which** \( L \) **decreases**
2. move \( \theta \) a bit into that direction
3. go to step 1
Find direction in which the loss decreases

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3(t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

\[(J_5)_{i,i} = \sigma'(z_3_i)\]

\[\nabla z = t - y\]
\[\nabla b_3 = t - y\]
\[\nabla W_3 = H_3 (t - y)^T\]
\[\nabla z_3 = J_5 J_6 (t - y)\]
\[\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)\]
Find direction in which the loss decreases

\( (J_5)_{i,i} = [(z_3)_i > 0] \)

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]

ReLU: \( \text{max}(x, 0) \)
Find direction in which the loss decreases

\[ J_6 = W_3 \]
\[ J_4 = W_2 \]
\[ J_1 = \text{“mumble } W_1 \text{ mumble mumble”} \]

\[ \nabla z = t - y \]
\[ \nabla b_3 = t - y \]
\[ \nabla W_3 = h_3 (t - y)^T \]
\[ \nabla z_3 = J_5 J_6 (t - y) \]
\[ \nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y) \]
Find direction in which the loss decreases

Problem:
Depending on $W_1$, $W_2$, $W_3$,
$\nabla Z_1$ may become very small
(“vanishing gradient”)
or large (“exploding gradient”)

\[
\begin{align*}
\nabla z &= t - y \\
\nabla b_3 &= t - y \\
\nabla W_3 &= h_3 (t - y)^T \\
\nabla z_3 &= J_5 J_6 (t - y) \\
\nabla Z_1 &= J_2 J_3 J_4 J_5 J_6 (t - y)
\end{align*}
\]
Initialization

**Problem:**
Depending on $\theta$, $-\frac{\partial}{\partial \theta} J(f(X; \theta), t)$ may become very small ("vanishing gradient") or large ("exploding gradient").
Initialization

**Problem:**
Depending on $\theta$, $-\frac{\partial}{\partial \theta} J(f(X; \theta), t)$ may become very small (“vanishing gradient”) or large (“exploding gradient”).

**Iterative scheme:**

0. **initialize $\theta$ randomly**
1. find direction in which $L$ decreases
2. move $\theta$ a bit into that direction
3. go to step 1
**Initialization**

**2006:** Initialize weights with unsupervised pretraining
**Initialization**

**2006:** Initialize weights with unsupervised pretraining

**2010:** Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)
Initialization

2006: Initialize weights with unsupervised pretraining

2010: Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)

2014: Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)
**Initialization**

- **2006:** Initialize weights with unsupervised pretraining
- **2010:** Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)
- **2014:** Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)
- **2016:** Initialize randomly, scaled by observed variance of actual training data at each layer (Krähenbühl; Mishkins; Salima)
**Initialization**

- **2006:** Initialize weights with unsupervised pretraining
- **2010:** Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)
- **2014:** Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)
- **2016:** Initialize randomly, scaled by observed variance of actual training data at each layer (Krähenbühl; Mishkins; Salima)
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. **initialize \( \theta \) randomly**
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. **find direction in which \( L \) decreases**
2. move \( \theta \) a bit into that direction
3. go to step 1
Find direction in which the loss decreases

\[ \nabla \theta = - \frac{\partial}{\partial \theta} J(f(X; \theta), t)\]

\[ - \frac{\partial}{\partial \theta} L(\theta; f, D) = - \sum_{(X, T) \in D} \frac{\partial}{\partial \theta} J(f(X; \theta), T) \]
Find direction in which the loss decreases

\[ \nabla \theta = - \frac{\partial}{\partial \theta} J(f(X; \theta), t) \]

\[ - \frac{\partial}{\partial \theta} L(\theta; f, D) = - \sum_{(x, t) \in D'} \frac{\partial}{\partial \theta} J(f(X; \theta), T) \quad \text{where } D' \subset D \]
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. **find direction in which** \( L \) **decreases**
2. move \( \theta \) a bit into that direction
3. go to step 1
Optimization

4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. **move \( \theta \) a bit into that direction**
3. go to step 1
Stochastic Gradient Descent (SGD):

\[ \theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta} \]

Take small step in direction of negative gradient.

**Analogy**: Somebody walking among hills, always in direction of steepest descent.

How far to move?
Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Take small step in direction of negative gradient.

**Analogy:** Somebody walking among hills, always in direction of steepest descent.

How far to move?
Too small $\eta$: slow progress
Too large $\eta$: oscillation or divergence
Stochastic Gradient Descent (SGD):

\[ \theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta} \]

Take small step in direction of negative gradient.

**Analogy**: Somebody walking among hills, always in direction of steepest descent.

How far to move?
Too small \( \eta \): slow progress
Too large \( \eta \): oscillation or divergence
Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Take small step in direction of negative gradient.

**Analogy**: Somebody walking among hills, always in direction of steepest descent.

How far to move?
Too small $\eta$: slow progress
Too large $\eta$: oscillation or divergence
Stochastic Gradient Descent (SGD) with Momentum:

\[
v \leftarrow \alpha v - \eta \frac{\partial L}{\partial \theta}
\]

\[
\theta \leftarrow \theta + v
\]

Dampen velocity according to friction coefficient \(\alpha\) (e.g., 0.9).
Increase velocity in direction of negative gradient.
Move according to velocity.

**Analogy:** Ball rolling down hills.
Adam (Adaptive Moment Estimation):

- Compute **velocity (first moment)**: exponential moving average over past gradients (as before)
- Compute **second moment estimate**: exponential moving average over past gradient magnitudes
- Move according to velocity, **divided by second moment**

**Intuition**: counter notoriously small gradients by upscaling, and large gradients by downscaling, separately for each weight
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. **move \( \theta \) a bit into that direction**
3. go to step 1
Optimization

4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly

1. **find direction in which** \( L \) **decreases**
2. move \( \theta \) a bit into that direction
3. go to step 1
Find direction in which the loss decreases

\[ \begin{bmatrix} \mathbf{W}_1 b_1 \\
\mathbf{Z}_1 H_1 \end{bmatrix} \xrightarrow{\rightarrow} \begin{bmatrix} \mathbf{W}_2 b_2 \\
\mathbf{H}_2 \end{bmatrix} \xrightarrow{\rightarrow} \begin{bmatrix} \mathbf{W}_3 b_3 \\
\mathbf{Z}_3 H_3 \end{bmatrix} \xrightarrow{\rightarrow} \mathbf{z} \]

\[ \begin{bmatrix} \mathbf{0.4} \\
\mathbf{0.0} \\
\mathbf{-0.1} \\
\mathbf{0.0} \\
\mathbf{0.0} \\
\mathbf{0.0} \\
\mathbf{-0.2} \\
\mathbf{0.0} \\
\mathbf{0.0} \\
\mathbf{0.0} \\
\mathbf{-0.1} \end{bmatrix} \]

\[ \mathbf{z} = \mathbf{t} - \mathbf{y} \]

**Problem:**
Depending on \( \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3 \), \( \nabla \mathbf{Z}_1 \) may become very small ("vanishing gradient") or large ("exploding gradient")

\[ \nabla \mathbf{z} = \mathbf{t} - \mathbf{y} \]
\[ \nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y} \]
\[ \nabla \mathbf{W}_3 = \mathbf{h}_3 (\mathbf{t} - \mathbf{y})^T \]
\[ \nabla \mathbf{z}_3 = \mathbf{J}_5 \mathbf{J}_6 (\mathbf{t} - \mathbf{y}) \]
\[ \nabla \mathbf{Z}_1 = \mathbf{J}_2 \mathbf{J}_3 \mathbf{J}_4 \mathbf{J}_5 \mathbf{J}_6 (\mathbf{t} - \mathbf{y}) \]
Gradient clipping

\[ \begin{align*}
X & \rightarrow W_1 b_1 \rightarrow J_1 J_2 \leftarrow J_3 \rightarrow W_2 b_2 \rightarrow J_4 J_5 \leftarrow J_6 \rightarrow W_3 b_3
\end{align*} \]

Possible solution:
Scale/clip \( \nabla z, \nabla h_3, \nabla z_3, \nabla H_1, \nabla Z_1 \)
when they become too large.

\[ \begin{align*}
\nabla z &= t - y \\
\nabla b_3 &= t - y \\
\nabla W_3 &= h_3 (t - y)^T \\
\nabla z_3 &= J_5 J_6 (t - y) \\
\nabla Z_1 &= J_2 J_3 J_4 J_5 J_6 (t - y)
\end{align*} \]
Unitary weights

Possible solution:
Parameterize $W_1$, $W_2$, $W_3$ such that they always stay orthogonal matrices.

$\nabla z = t - y$
$\nabla b_3 = t - y$
$\nabla W_3 = h_3 (t - y)^T$
$\nabla z_3 = J_5 J_6 (t - y)$
$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$

abs/1707.09520: Orthogonal Recurrent Neural Networks with Scaled Cayley Transform
Batch normalization

Possible solution:
Normalize to zero mean / unit variance after every layer
- learn scale and bias on top to not lose expressivity
- estimate mean / variance on minibatch, not full dataset
- use fixed statistics after training
- backpropagate error through mean / variance computation
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

Iterative scheme:

0. initialize \( \theta \) randomly
1. find direction in which \( L \) decreases
2. move \( \theta \) a bit into that direction
3. go to step 1
Deep learning in practice

- Initialization
- SGD+
- Batch normalization
- Optimization
Deep learning in practice

- Initialization
- SGD+
- Batch normalization
- Optimization

- Generalization
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

**What we get:**

\[ f(\mathbf{X}; \theta) = \mathbf{T} \text{ for all } (\mathbf{X}, \mathbf{T}) \in \mathbf{D} \]
Generalization

4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_{\theta} L(\theta; f, D) \]

**What we get:**
\[ f(\mathbf{X}; \theta) = T \text{ for all } (\mathbf{X}, T) \in D \]

**What we wanted:**
\[ f(\mathbf{X}; \theta) = T \text{ for all } (\mathbf{X}, T) \notin D \text{ (but from the same task)} \]
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

**What we get:**

\[ f(X; \theta) = T \text{ for all } (X, T) \in D \]

**What we wanted:**

\[ f(X; \theta) = T \text{ for all } (X, T) \notin D \text{ (but from the same task)} \]

**Problem:**

There exist \( \theta \) that fulfil the first, but not the second.
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

**What we get:**
\( f(\mathbf{X}; \theta) = \mathbf{T} \) for all \((\mathbf{X}, \mathbf{T}) \in D\)

**What we wanted:**
\( f(\mathbf{X}; \theta) = \mathbf{T} \) for all \((\mathbf{X}, \mathbf{T}) \notin D\)

**Problem:**
There exist \( \theta \) that fulfil the first, but not the second. \( \rightarrow \) **overfitting**
4. **Optimize** model parameters to minimize loss

\[ \theta^* = \min_\theta L(\theta; f, D) \]

**Goal:**

Modify optimization to avoid solutions \( \theta \) that only match the training examples.
Weight decay

**Goal:** Modify optimization to avoid solutions $\theta$ that only match the training examples.

**Observation:** Learning examples by heart often requires large jumps in the function = large gradients = large coefficients multiplied with inputs

**Countermeasure:** Shrink weights after each update (= L2 decay), or whenever too large (weight clipping)
Early stopping

**Goal:** Modify optimization to avoid solutions $\theta$ that only match the training examples.

**Observation:** Training is iterative. Initial model underfits.
Early stopping

**Goal:** Modify optimization to avoid solutions $\theta$ that only match the training examples.

**Observation:** Training is iterative. Initial model underfits. Final model overfits.
Early stopping

**Goal:** Modify optimization to avoid solutions $\theta$ that only match the training examples.

**Observation:** Training is iterative. Initial model underfits. Final model overfits.

**Solution:** Stop training in between. Monitor loss on extra data to find sweet spot.
Data augmentation

Goal: Modify optimization to avoid solutions $\theta$ that only match the training examples.

Observation: Overfitting may mean the solution depends on irrelevant properties of the input.

Possible solutions:
- More data
- Design invariant model
- Data augmentation
Data augmentation:
Transform training data, let classifier learn to ignore it.
Data augmentation:
Transform training data, let classifier learn to ignore it.

Typical transformations:
- For images: horizontal flip, scale, rotation, color, contrast
- For audio: time stretching, pitch shifting, equalizer
**Goal**: Modify optimization to avoid solutions $\theta$ that only match the training examples.

**Observation**: Units can learn to focus on few units in previous layer to distinguish training examples.

**Solution**: Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.
Solution: Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.
**Solution:** Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.

At test time, do not drop any units (and do not scale up weights). Can be interpreted as an ensemble of $2^N$ networks trained simultaneously with shared weights.
Solution: Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.

MNIST digit recognition:

First-layer features after training:

No dropout: noisy, possibly overfit to training set

20% input, 50% hidden dropout: cleaner global features, more general
Solution: Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.

MNIST digit recognition:

No dropout: quick overfitting, 169 test errors

20% input, 50% hidden dropout: validation error plateaus, 99 test errors
Deep learning in practice

Optimization
- Initialization
- SGD+
- Batch normalization

Generalization
- Weight decay
- Early stopping
- Data augmentation
- Dropout

Data augmentation: Techniques to increase the size of the training dataset by creating modified versions of existing data.

Dropout: A regularization technique for neural networks that randomly disables some neurons during training, preventing overfitting.

Weight decay: A regularization technique that adds a penalty term to the loss function to reduce the magnitude of weights, promoting sparsity.

Early stopping: A form of regularization that stops training before the learner passes beyond a point of generalization to prevent overfitting.

Batch normalization: A technique that normalizes the inputs to each layer by computing and applying the mean and variance of the activations of a mini-batch.

SGD+: Stochastic Gradient Descent with momentum and accelerated gradient.
Deep learning in practice

Optimization
- Initialization
- SGD+
- Batch normalization

Generalization
- Weight decay
- Early stopping
- Data augmentation
- Dropout

Architectures
Going Deeper

How many layers to use?

- Single hidden layer enough for universal approximation
- More hidden layers can express functions more compactly

ImageNet Large Scale Visual Recognition Challenge:
1.2 million training images of 1000 classes (incl. 120 dog breeds)

- 2012: AlexNet, 16.4% top-5 error, 8 layers.
- 2013: ZFNet, 11.2% top-5 error, 8 layers.
- 2014: GoogLeNet: 6.7% top-5 error, 22 layers.
- 2015: ResNets: 3.6% top-5 error, 152 layers.
How many layers to use?

- Single hidden layer enough for universal approximation
- More hidden layers can express functions more compactly
How many layers to use? How to use many layers?

**GoogLeNet:** 22 layers, *auxiliary classifiers*

**Idea:** Provide better gradient information to lower layers via additional classification layers

How many layers to use? How to use many layers?

ResNet: 152 layers (38 shown here), **shortcut connections**

**Idea:** Provide better gradient information to lower layers via bypasses. Input directly connected to output, learns residuals. Shown to learn networks of 1001 layers. But: seems to behave like an ensemble of many shallow networks, not a single deep one.

DenseNet

How many layers to use? How to use many layers?

DenseNet: like ResNet, but shortcuts append, not add features

**Idea:** Each layer expands the set of available feature maps. Avoids redundant features as learned in ResNet.

Aug 2016, abs/1608.06993: Densely Connected Convolutional Networks
**Three dimensions:** Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.

**AlexNet:** Two chains of identical structure joined in the end. Originally for technical reasons, later shown to improve results.

*NIPS 2012: ImageNet Classification with Deep Convolutional Neural Networks*
Grouped convolution

**Three dimensions:** Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.

**AlexNet:** Two chains of identical structure joined in the end. Originally for technical reasons, later shown to improve results.

NIPS 2012: ImageNet Classification with Deep Convolutional Neural Networks
Three dimensions: Depth, Width, Multiplicity

Can be advantageous to have separate processing chains.

Shake-Shake: Two parallel processing steps averaged.

May 2017, abs/1705.07485: Shake-Shake regularization
Three dimensions: Depth, Width, Multiplicity
Can be advantageous to have separate processing chains.

Shake-Shake: Two parallel processing steps averaged, randomly combined.
**Three dimensions:** Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.

**Shake-Shake:** Two parallel processing steps averaged, randomly combined, with different coefficients in forward/backward pass.

May 2017, abs/1705.07485: Shake-Shake regularization
Deep learning in practice

- **Optimization**
  - Initialization
  - SGD+
  - Batch normalization

- **Generalization**
  - Weight decay
  - Early stopping
  - Data augmentation
  - Dropout
  - Shake-shake

- **Architectures**
  - Inception
  - ResNet
  - DenseNet
  - Grouped convolution

Deep learning in practice

- Optimization:
  - Initialization
  - SGD+
  - Batch normalization

- Generalization:
  - Weight decay
  - Early stopping
  - Data augmentation
  - Dropout
  - Shake-shake

- Architectures:
  - Inception
  - ResNet
  - DenseNet
  - Grouped convolution

- Inspection
Inspection

\[ X \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \xrightarrow{\text{conv}} \xrightarrow{\text{pool}} \ldots \xrightarrow{\text{dense}} \xrightarrow{\text{dense}} = \begin{pmatrix} 0.0 \\ 0.8 \\ 0.0 \\ 0.1 \\ \vdots \\ 0.0 \end{pmatrix} \]

king snake
Inspection

\[ X \xrightarrow{\text{}} y = \begin{pmatrix} 0.0 \\ 0.8 \\ 0.0 \\ 0.1 \\ \vdots \end{pmatrix} \text{ king snake} \]
**Method:** Show convolution kernels in pixel space. Only possible for first layer.
**Method:** Show training patches that maximally activate some unit.
**Method:** Show training patches that maximally activate some unit.

Nov 2013, abs/1311.2901: Visualizing and Understanding Convolutional Networks
**Method:** Show training patches that maximally activate some unit.

Nov 2013, abs/1311.2901: Visualizing and Understanding Convolutional Networks
**Method:** Generate patches that maximally activate some unit.

**Guided backpropagation**

**Method:** Show gradient of some unit wrt. input example (modified backward pass).

![Diagram of a neural network]

Dec 2014, abs/1412.6806: Striving for Simplicity: The All Convolutional Net
Guided backpropagation

Method: Show gradient of some unit wrt. input example (modified backward pass).

ISMIR 2016: Learning to Pinpoint Singing Voice from Weakly Labeled Examples
Deep learning in practice

Optimization
- Initialization
- SGD+
- Batch normalization
- Weight decay
- Early stopping
- Data augmentation
- Dropout
- Shake-shake

Generalization
- Architectures
  - Inception
  - ResNet
  - DenseNet
  - Grouped convolution
- Inspection
  - Show filters
  - Show data
  - Generate data
  - Guided backpropagation
Deep learning in practice

If it looks stupid but works, it ain't stupid.