Model Predictive Control and Reinforcement Learning
– Introduction to Deep Learning –

Joschka Boedecker and Moritz Diehl

University Freiburg

October 5, 2023

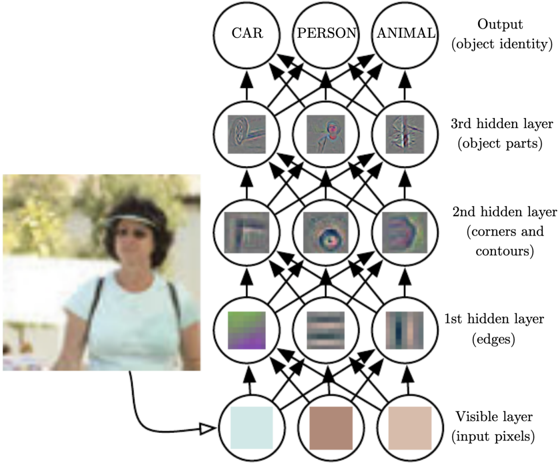
Lecture Overview

1. Multilayer Peceptrons
2. Recap: Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Basics of Gradient Descent Optimization
5. Convolutional Neural Networks
6. Recurrent Neural Networks

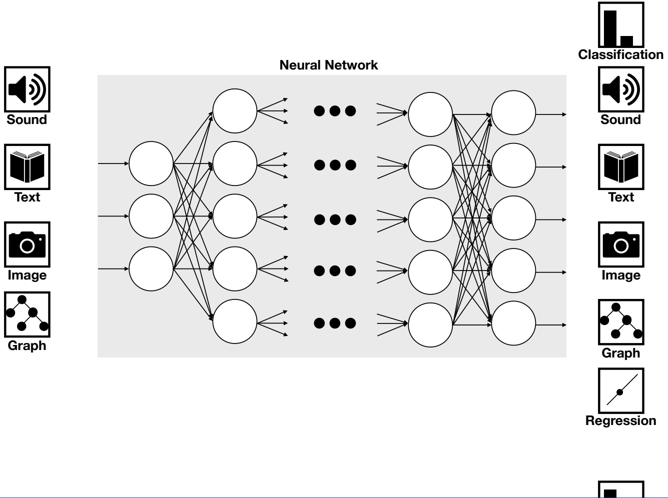
Acknowledgement

Slides contain contents from a lecture designed together with our colleagues Frank Hutter and Abhinav Valada. Some contents are from the Stanford course CS231n Convolutional Neural Networks for Visual Recognition and from the Deep Learning book by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

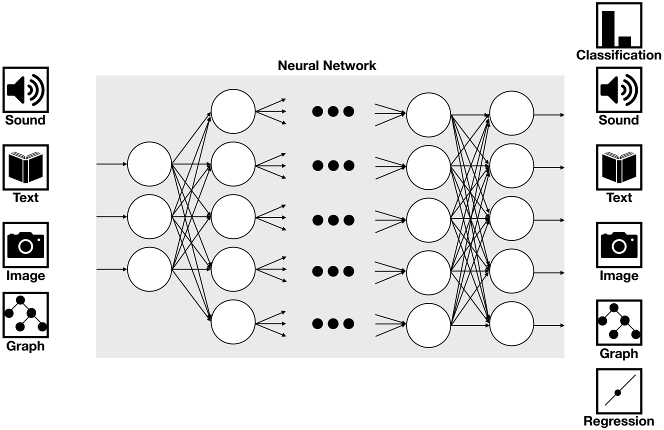
Motivation: Representation Learning



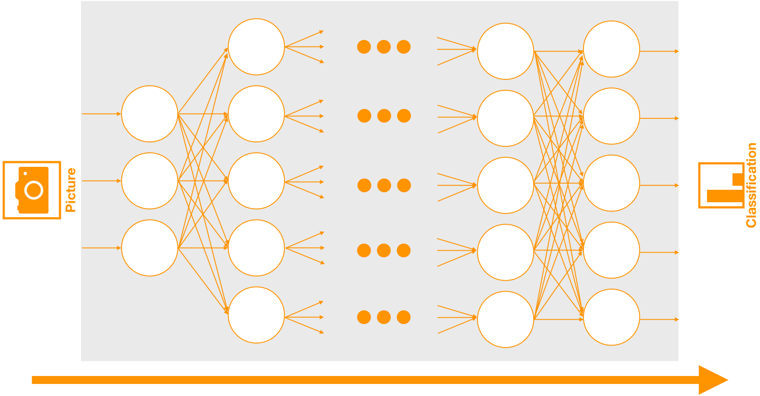
Multilayer Perceptrons (MLPs): Fully-Connected Feedforward Neural Networks



Types of Layers in an MLP

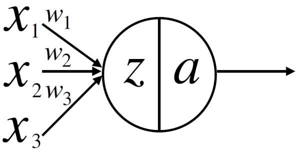


Computation is Performed Layer-by-Layer



Computations in a Single Neuron

- Each connection between two neurons has a weight, w
- A single neuron performs two simple steps of computation:



1. Compute a weighted sum of the inputs: $z = x_1 w_1 + x_2 w_2 + x_3 w_3$
2. Perform a nonlinear transformation: $a = h(z)$.

Computation in the First Hidden Layer

- For input vector $\mathbf{x}$, compute pre-activations $\mathbf{z}^{(1)}$ in layer 1 as

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)^{\top}} \mathbf{x} + \mathbf{b}^{(1)}$$

- Pre-activations are transformed through a differentiable, nonlinear activation function $g^{(1)}(\cdot)$, resulting in activation vector $\mathbf{h}^{(1)}$ of the first hidden layer:

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{z}^{(1)})$$

- The units in this layer implement the adaptable basis functions.

Computation in the Second Hidden Layer etc.

- Outputs $\mathbf{h}^{(1)}$ from layer 1 are combined linearly in the next layer 2:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)^\top} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$

- Preactivations $\mathbf{z}^{(2)}$ are again transformed through a nonlinear activation function $g^{(2)}$ to compute the activations $\mathbf{h}^{(2)}$:

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{z}^{(2)})$$

- This repeats from each layer k to $k+1$, all the way to output layer K
  - The network then outputs the output layer's activations: $\hat{\mathbf{y}} := \mathbf{h}^{(K)}$.

Summary of Layer-by-layer Computations

- Layer 1 pre-activations:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)^\intercal}\mathbf{x} + \mathbf{b}^{(1)}$$

- Layer 1 activations:

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{z}^{(1)})$$

- Layer i pre-activations:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)^\intercal}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}$$

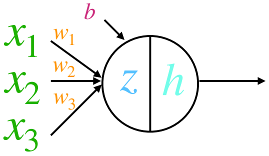
- Layer i activations:

$$\mathbf{h}^{(i)} = g^{(i)}(\mathbf{z}^{(i)})$$

- Overall network output as one big nested function (network with one hidden layer):

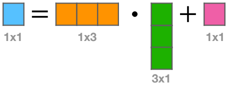
$$\hat{\mathbf{y}} = g^{(2)}(\mathbf{W}^{(2)^\intercal} g^{(1)}(\mathbf{W}^{(1)^\intercal}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

One Neuron, One Input Vector

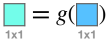


$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

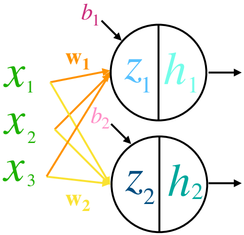
$$z = \mathbf{w}^\top \mathbf{x} + b$$



$$h = g(z)$$



Two Neurons, One Input Vector



$$z_1 = \mathbf{w_1}^\top \mathbf{x} + b_1$$
$$h_1 = g(z_1)$$
$$z_2 = \mathbf{w_2}^\top \mathbf{x} + b_2$$
$$h_2 = g(z_2)$$

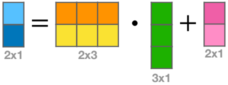
⇛

Put together into one:

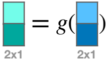
$$\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$
$$\mathbf{h} = g(\mathbf{z})$$

Visual illustration:

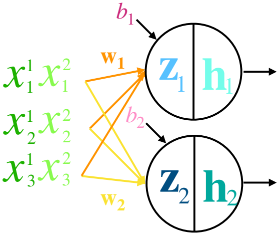
$$\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$



$$\mathbf{h} = g(\mathbf{z})$$



Two Neurons, Batch of Two Input Vectors



$$\mathbf{z_1} = \mathbf{w_1}^\top \mathbf{X} + b_1$$
$$\mathbf{h_1} = g(z_1)$$
$$\mathbf{z_2} = \mathbf{w_2}^\top \mathbf{X} + b_2$$
$$\mathbf{h_2} = g(z_2)$$

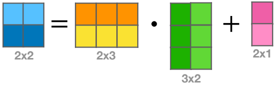
$\Rrightarrow$

Put together into one:

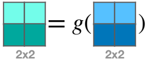
$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X} + \mathbf{b}$$
$$\mathbf{h} = \mathbf{g}(\mathbf{Z})$$

Visual illustration:

$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X} + \mathbf{b}$$

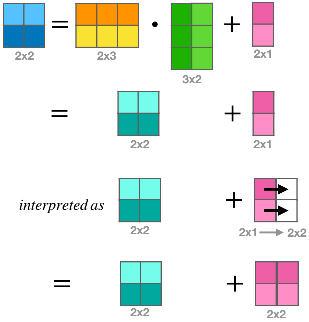


$$\mathbf{H} = \mathbf{g}(\mathbf{Z})$$



Two Neurons, Batch of Two Input Vectors

$$\mathbf{Z} = \mathbf{W}^\top \mathbf{X} + \mathbf{b}$$



Warning: Different Common Notations in Math and in Code

- Python frameworks for Deep Learning (like PyTorch) use a different notation
  - Here, we follow the (standard) notation of $\mathbf{x}$ being a column vector
  - In PyTorch, data points $\mathbf{x}$ are row vectors

- Summary of PyTorch notation
  - The inputs $\mathbf{X} \in \mathbb{R}^{N \times D}$ have N datapoints in the rows and D features in the columns
  - A single linear layer has weight $\mathbf{W} \in \mathbb{R}^{D \times M}$ and bias $\mathbf{b} \in \mathbb{R}^{M}$
  - The bias is expanded to $\mathbf{B} \in \mathbb{R}^{N \times M}$ by repeating it for each datapoint.

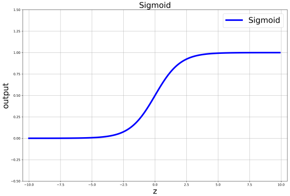
  - The formula for output $\mathbf{Z} \in \mathbb{R}^{N \times M}$ is then:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{B}$$

Activation Functions - Examples

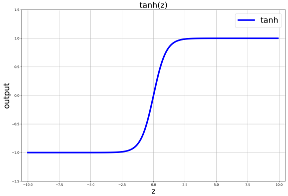
Logistic sigmoid activation function:

$$g_{logistic}(z) = \frac{1}{1 + \exp(-z)}$$



Logistic hyperbolic tangent activation function:

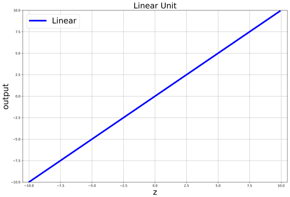
$$\begin{aligned} g_{tanh}(z) &= \tanh(z) \\ &= \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \end{aligned}$$



Activation Functions - Examples (cont.)

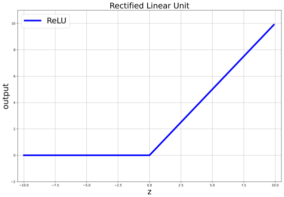
Linear activation function:

$$g_{linear}(z) = z$$



Rectified Linear (ReLU) activation function:

$$g_{relu}(z) = \max(0, z)$$



Output unit activation functions

Depending on the task, typically:

- for regression: output neurons with linear activation
- for binary classification: output neurons with logistic/tanh activation
- for multiclass classification with K classes: use K output neurons and softmax activation

$$(\hat{\mathbf{y}}(\mathbf{x},\mathbf{w}))_k = p(y_k = 1) = g_{softmax}((\mathbf{z})_k) = \frac{\exp((\mathbf{z})_k)}{\sum_j \exp((\mathbf{z})_j)}$$

$\rightarrow$ so for the complete output layer:

$$\hat{\mathbf{y}}(\mathbf{x},\mathbf{w}) = \begin{bmatrix} p(y_1 = 1|\mathbf{x}) \\ p(y_2 = 1|\mathbf{x}) \\ \vdots \\ p(y_K = 1|\mathbf{x}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp((\mathbf{z})_j)} \exp(\mathbf{z})$$

Typical error functions

- For binary classification, cross-entropy error:

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\{y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\}$$

- For linear outputs, mean squared error function:

$$L(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\{\hat{y}(\mathbf{x}_n,\mathbf{w}) - y_n\}^2$$

- For multiclass classification, generalization of cross-entropy error:

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn}\log \hat{y}_k(\mathbf{x}_n,\mathbf{w})$$

Lecture Overview

1 Multilayer Peceptrons

2 Recap: Chain Rule of Calculus

3 Calculating Gradients with Backpropagation

4 Basics of Gradient Descent Optimization

5 Convolutional Neural Networks

6 Recurrent Neural Networks

Chain rule

The chain rule computes derivatives for compositions of functions by using their individual derivatives and the product of their functions as below.

For two functions $g(x)$ and $f(y) = f(g(x))$, the chain rule states:

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

For $y = g(x)$ and $z = f(g(x)) = f(y)$:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)}\frac{\partial g(x)}{\partial x}$$

Let $z = f(a) = \ln(a)$ and $y = g(x) = \sin(x)$. Then:

$$\frac{\partial z}{\partial x} = \frac{\partial \ln(\sin(x))}{\partial \sin(x)}\frac{\partial \sin(x)}{\partial x} = \frac{1}{\sin(x)} \cdot \cos(x)$$

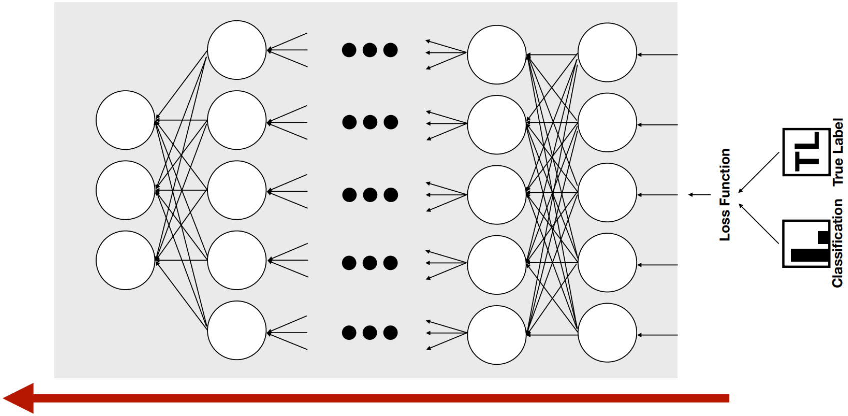
Chain rule

As a generalization of the scalar case, consider $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

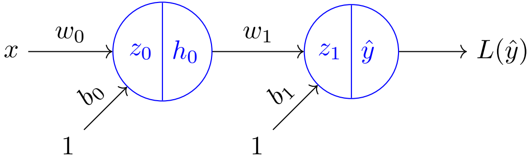
Lecture Overview

Backpropagation: Information Flow Illustration



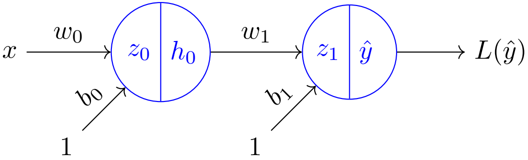
Calculating partial derivatives (cont.)

- We will look at how to derive the gradients in the context of a simple example network:



- We would like to know how the change in any of the weights and biases influences the loss, so we calculate $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for all weights and biases in the network.

Calculating partial derivatives (cont.)



$$\begin{aligned}\hat{y} &= g_1(z_1)\\ z_1 &= w_1 h_0 + b_1\\ h_0 &= g_0(z_0)\\ z_0 &= w_0 x + b_0\end{aligned}$$

$$\begin{aligned}&\frac{\partial L}{\partial \hat{y}}\\ \frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_1}\\ \frac{\partial L}{\partial h_0} &= \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial h_0}\\ \frac{\partial L}{\partial z_0} &= \frac{\partial L}{\partial h_0}\frac{\partial h_0}{\partial z_0}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial w_1}\\ \frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial z_1}\frac{\partial z_1}{\partial b_1}\\ \frac{\partial L}{\partial w_0} &= \frac{\partial L}{\partial z_0}\frac{\partial z_0}{\partial w_0}\\ \frac{\partial L}{\partial b_0} &= \frac{\partial L}{\partial z_0}\frac{\partial z_0}{\partial b_0}\end{aligned}$$

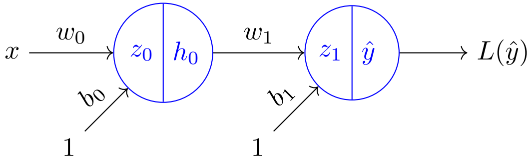
Calculating partial derivatives (cont.)

Derivative of the activation function w.r.t its activation $\frac{\partial h}{\partial z} = h'(z)$ depends on which activation we use:

- linear activation: $h(z) = z \rightarrow h'(z) = 1$
- logistic sigmoid activation: $h(z) = 1/(1 + \exp(-z)) \rightarrow h'(z) = h(z)(1 - h(z))$
- hyperbolic tangent sigmoid activation: $h(z) = \text{tanh}(z) \rightarrow h'(z) = 1 - h(z)^2$
- ReLU activation: $h'(z) = 0$ if $z < 0$, $h'(z) = 1$ if $z \geq 0$

  $$h(z) = \begin{cases} z & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \leq 0 \end{cases} \rightarrow h'(z) = \begin{cases} 1 & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \leq 0 \end{cases}$$

Reconsider 2-layer MLP as an example



For each pattern $\mathbf{x}_n$ in training set, perform forward pass:

- hidden layer:
  - $z_0 = xw_0 + b_0$
  - $h_0 = g_0(z_0) = \text{ReLU}(z_0)$
- output layer:
  - $z_1 = h_0 w_1 + b_1$
  - $\hat{y} = g_1(z_1) = z_1$
- g_0 being a ReLU, and g_1 being a linear activation function
- Consider squared error loss: $L = \frac{1}{2}(\hat{y} - y)^2$

Reconsider 2-layer example (cont.)

Forward pass:

$$
\begin{aligned}
L &= \frac{1}{2}(\hat{y}-y)^2 \\
\hat{y} &= g_1(z_1) && = z_1 \\
z_1 &= w_1 h_0 + b_1 \\
h_0 &= g_0(z_0) && = \begin{cases} 1 & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \le 0 \end{cases} \\
z_0 &= w_0 x + b_0
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &= \hat{y} - y \\
\frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_1} && = (\hat{y}-y) \cdot g_1'(z_1) && = (\hat{y}-y) \cdot 1 \\
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_1} && = \frac{\partial L}{\partial z_1} h_0 \\
\frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial b_1} && = \frac{\partial L}{\partial z_1} \cdot 1 \\
\frac{\partial L}{\partial h_0} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial h_0} && = \frac{\partial L}{\partial z_1} w_1 \\
\frac{\partial L}{\partial z_0} &= \frac{\partial L}{\partial h_0} \frac{\partial h_0}{\partial z_0} && = \begin{cases} \frac{\partial L}{\partial h_0} & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \le 0 \end{cases} \\
\frac{\partial L}{\partial w_0} &= \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial w_0} && = \frac{\partial L}{\partial z_0} x \\
\frac{\partial L}{\partial b_0} &= \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial b_0} && = \frac{\partial L}{\partial z_0} \cdot 1
\end{aligned}
$$

Generic MLP learning algorithm using Backpropagation

- generic MLP learning algorithm:

```
choose an initial weight vector $\vec{w}$
intialize minimization approach
while error did not converge do
    for all $(\mathbf{x}, y) \in \mathcal{D}$ do
        apply $\mathbf{x}$ to network and calculate the network output (forward pass)
        calculate $\frac{\partial L_n}{\partial w}$ and $\frac{\partial L_n}{\partial b}$ for all weights and biases (backward pass)
    end for
    calculate total gradients $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for all weights and biases, summing over all training patterns
    perform one update step of the minimization approach
end while
```

- learning by epoch: all training patterns are considered for one update step of function minimization

Lecture Overview

1 Multilayer Peceptrons

2 Recap: Chain Rule of Calculus

3 Calculating Gradients with Backpropagation

4 **Basics of Gradient Descent Optimization**

5 Convolutional Neural Networks

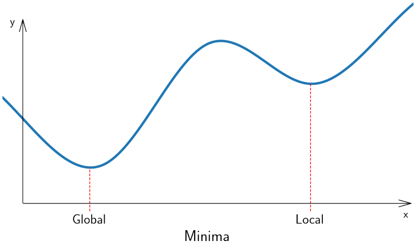
6 Recurrent Neural Networks

The Goal of Our Optimization Problem

- We're interested in problems of the form
$$\underset{\vec{x}}{minimize}\ f(\vec{x}),$$
where $\vec{x}$ is a vector of suitable size.

- A global minimum $\vec{x}^*$ is a point such that:
$$f(\vec{x}^*) \leq f(\vec{x})$$
for all $\vec{x}$.

- A local minimum $\vec{x}^+$ is a point such that there exists $r > 0$ with
$$f(\vec{x}^+) \leq f(\vec{x})$$
for all points $\vec{x}$ with $||\vec{x} - \vec{x}^+|| < r$



Gradient-based Optimization: Need For Iterative Solvers

- Analytical way to find a minimum:
  For a local minimum $\vec{x}^+$, the gradient of f becomes zero:

  $$\frac{\partial f}{\partial x_i}(\vec{x}^+) = 0 \quad \text{for all } i$$

  Hence, calculating all partial derivatives and looking for zeros is a good idea

- But: for neural networks, we can't write down a solution for the minimization problem in closed form
  - even though $\frac{\partial f}{\partial x_i} = 0$ holds at (local) solution points

  → need to resort to iterative methods

Gradient Descent: Intuition for the Update Equation

- Numerical way to find a minimum, searching:
  assume we start at point $\vec{x}$.

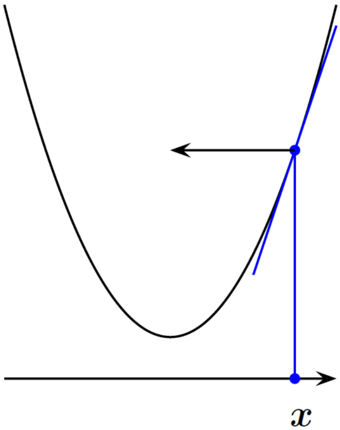
  Which is the best direction to search for a point $\vec{x}'$ with $f(\vec{x}') < f(\vec{x})$?

  Which is the best stepwidth?
- general principle:

$$x_i' \leftarrow x_i - \alpha \frac{\partial f}{\partial x_i}$$

$\alpha > 0$ is called learning rate

slope is large, large step!



Gradient Descent: The Full Algorithm

- Gradient descent approach:

```
Require: mathematical function $f$, learning rate $\alpha > 0$
Ensure: returned vector is close to a local minimum of $f$
choose an initial point $\vec{x}$
while $||\nabla f(\vec{x})||$ not close to $0$ do
    $\vec{x} \leftarrow \vec{x} - \alpha \nabla f(\vec{x})$
end while
return $\vec{x}$
```

- Note: $\nabla f := [\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_K}]$ for K dimensions

Lecture Overview

1 Multilayer Peceptrons

2 Recap: Chain Rule of Calculus

3 Calculating Gradients with Backpropagation

4 Basics of Gradient Descent Optimization

5 Convolutional Neural Networks

6 Recurrent Neural Networks

MLPs vs ConvNets

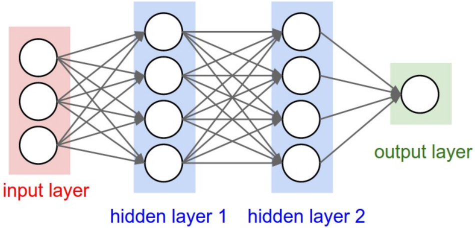


Figure: Multilayer Perceptron

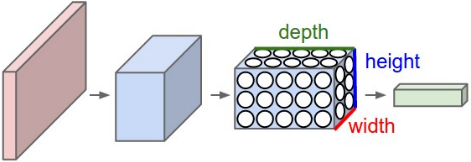


Figure: Convolutional Neural Network

[figure credit: Stanford CS231n]

MLPs vs ConvNets

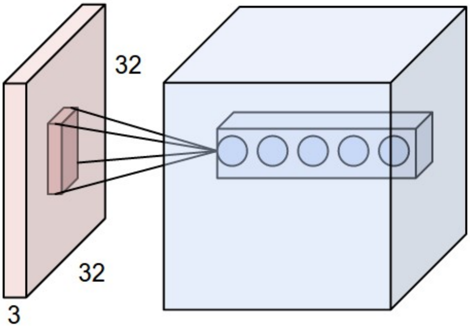


Figure: Example input volume and first conv layer

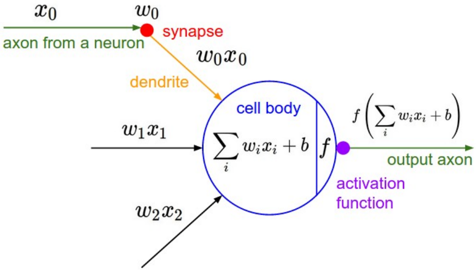
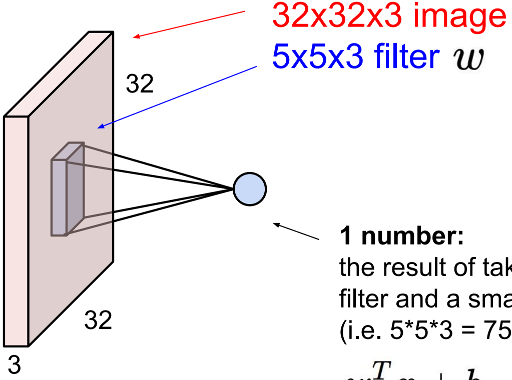


Figure: Computations of the neurons in the conv layer are unchanged

[figure credit: Stanford CS231n]

Convolutions illustrated (cont.)



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

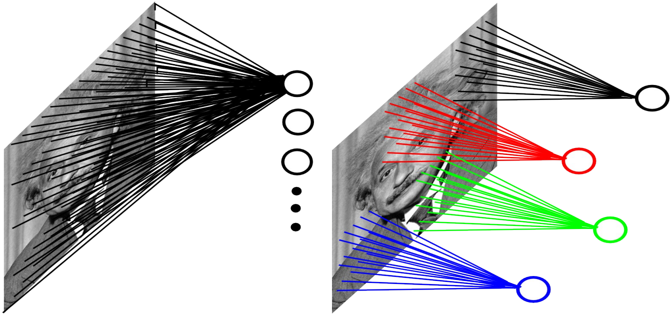
$$w^T x + b$$

[slide credit: Stanford CS231n]

Regularization through weight sharing

Example: 200×200 image

- Fully-connected: $400,000$ hidden units $= 160 * 10^9$ parameters
- Locally-connected: $400,000$ hidden units with 10×10 fields$= 40 * 10^6$ parameters

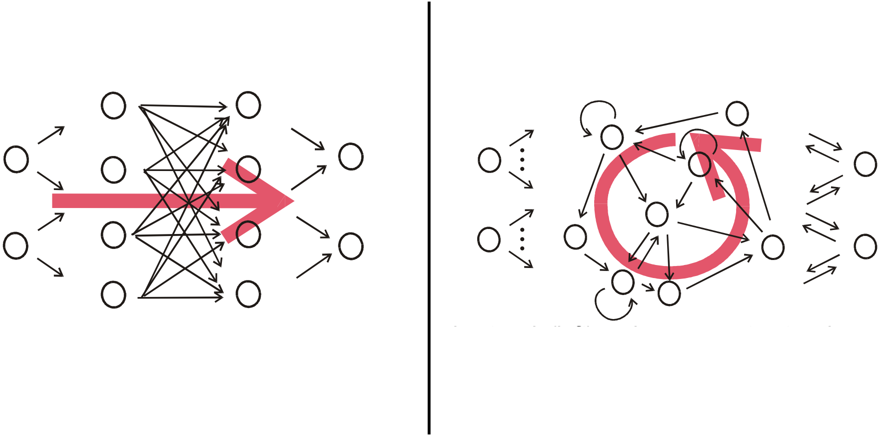


[figure credit: Y. LeCun and M.A. Ranzato]

Lecture Overview

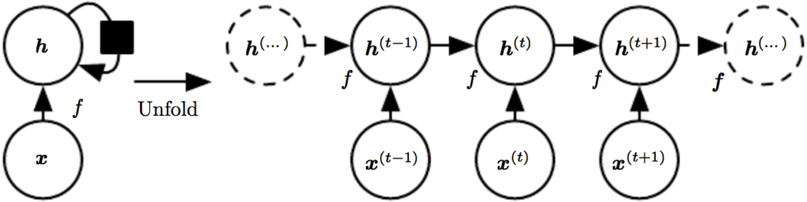
1. Multilayer Peceptrons
2. Recap: Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Basics of Gradient Descent Optimization
5. Convolutional Neural Networks
6. Recurrent Neural Networks

Feedforward vs Recurrent Neural Networks



[figure credit: H. Jaeger]

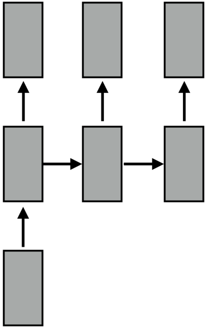
Unfolding the Computational Graph of an RNN



$$\begin{aligned}
\boldsymbol{h}^{(t)} &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \\
&= f(f(\boldsymbol{h}^{(t-2)}, \boldsymbol{x}^{(t-1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(t)}; \boldsymbol{\theta})
\end{aligned}$$

Sequence to sequence mapping - one to many

one to many



e.g. Image Caption Generation

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

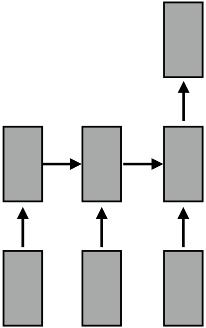
"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

[credit: A. Karpathy, F. Li, "Deep Visual-Semantic Alignments for Generating Image Descriptions"]

Sequence to sequence mapping - many to one

many to one

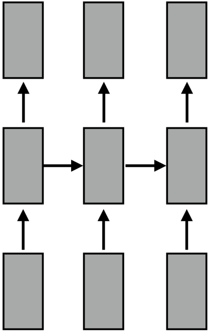


e.g. Sentiment Classification

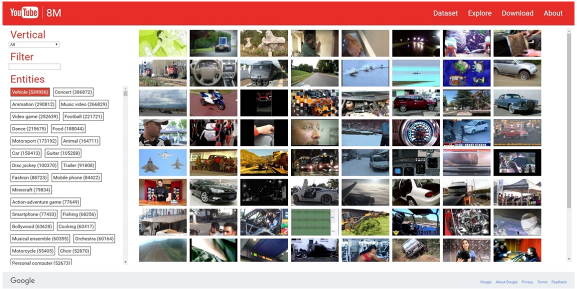
Review (X)	**Rating (Y)**
"This movie is fantastic! I really like it because it is so good!"	★★★★☆
"Not to my taste, will skip and watch another movie"	★★☆☆☆
"This movie really sucks! Can I get my money back please?"	★☆☆☆☆

Sequence to sequence mapping - many to many

many to many



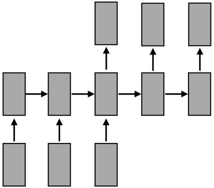
e.g. Video frame classification



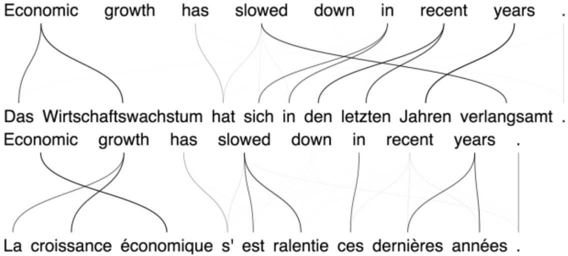
[credit: YouTube-8M]

Sequence to sequence mapping - many to many (cont'd)

many to many



e.g. Machine Translation



Economic growth has slowed down in recent years .

Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .

Economic growth has slowed down in recent years .

La croissance économique s' est ralentie ces dernières années .

[credit: nvidia]