Overview of Deep Learning Instruments

Sergey Ivanov (517)

qbrick@mail.ru

September 17, 2018

Table of contents

Section 1

Deep Learning

What is neural net?

What is neural net?

- parametric family $f(x, \theta), \quad \theta \in \Theta$

What is neural net?

- parametric family $f(x, \theta), \quad \theta \in \Theta$
- with universal approximation properties

What is neural net?

- parametric family $f(x, \theta), \quad \theta \in \Theta$
- with universal approximation properties
- differentiable

What is neural net?

- parametric family $f(x, \theta), \quad \theta \in \Theta$
- with universal approximation properties
- differentiable

Deep Learning is Machine Learning!

Machine Learning is always about searching for function:

$$\mathbb{E}_{(x,y)\sim \text{Data}} \text{Loss}(f(x, \theta), y) \to \min_{\theta}$$

Building neural nets

Common way to build complex functions — composition:

$$f(x, \theta) = f_1(f_2(f_3(\dots)))$$

Chain rule gives us the derivative $\nabla f(x, \theta)$

Building neural nets

Common way to build complex functions — composition:

$$f(x, \theta) = f_1(f_2(f_3(\ldots)))$$

Chain rule gives us the derivative $\nabla f(x, \theta)$

Same works for functions of vectors!

Building neural nets

Common way to build complex functions — composition:

$$f(x, \theta) = f_1(f_2(f_3(\dots)))$$

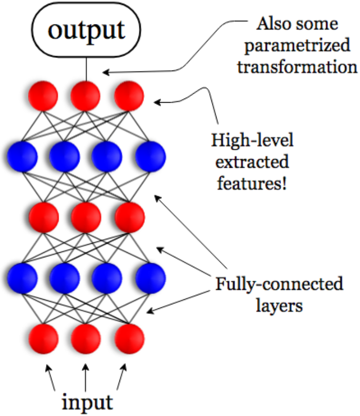
Chain rule gives us the derivative $\nabla f(x, \theta)$

Same works for functions of vectors!
Typical example:

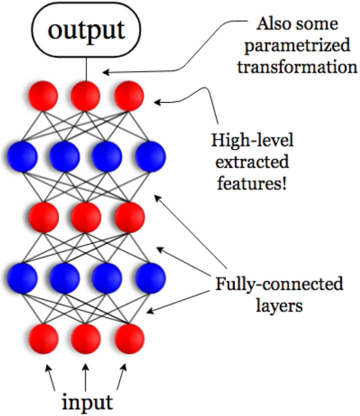
$$f_i(x, \theta) \in \{Ax, \sigma(x), \dots\}$$

where σ — some element-wise nonlinear function.

Typical example



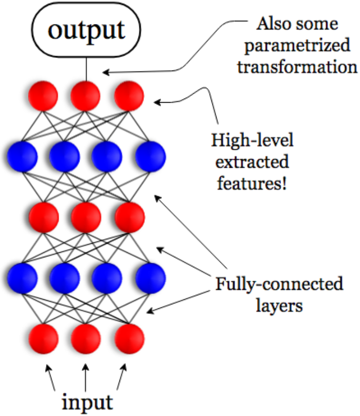
Typical example



Output:

- regression:
  - just numbers

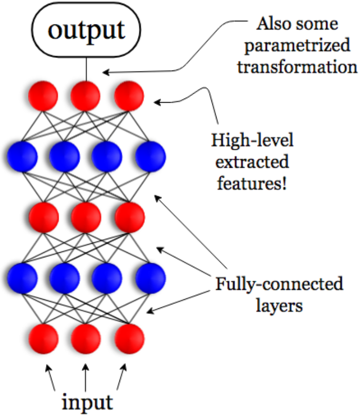
Typical example



Output:

- regression:
  - just numbers
  - parameters of distribution

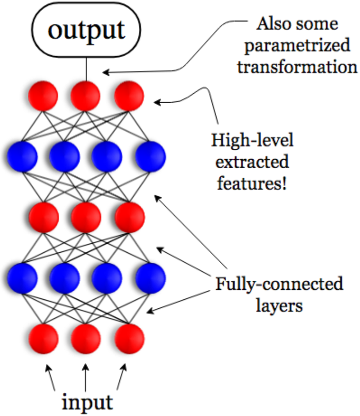
Typical example



Output:

- regression:
  - just numbers
  - parameters of distribution
- classification:
  - × just classes

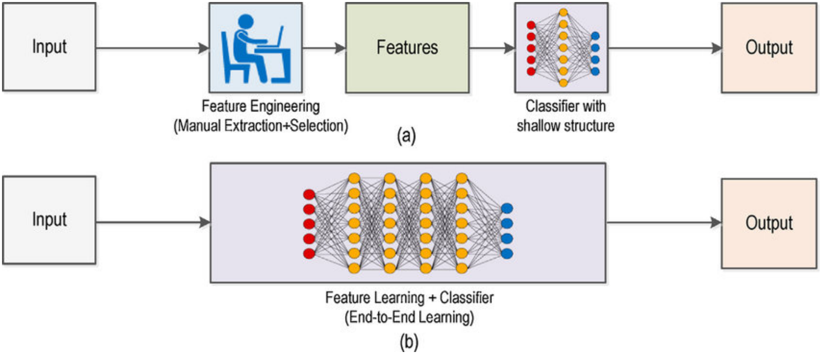
Typical example



Output:

- regression:
  - just numbers
  - parameters of distribution
- classification:
  - × just classes
  - probabilities of classes

End-to-end learning



Automation is the goal!

In DL we are required to specify:

- net topology

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method
  - use more or less universal methods like Adam

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method
  - use more or less universal methods like Adam
  - ✓ Meta-learning

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method
  - use more or less universal methods like Adam
  - ✓ Meta-learning
- data representation

Automation is the goal!

In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method
  - use more or less universal methods like Adam
  - ✓ Meta-learning
- data representation
  - "stack more layers"
  - "we need to go deeper"

Automation is the goal!

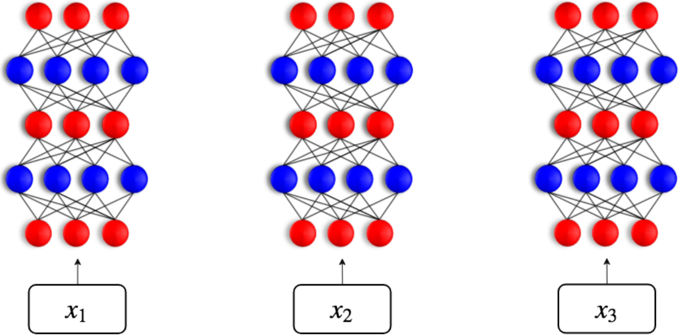
In DL we are required to specify:

- net topology
  - trial and error
  - evolutionary methods
  - ✓ AutoML
- regularization
  - dropout
  - batch normalization
  - ✓ Bayesian neural nets
- optimization method
  - use more or less universal methods like Adam
  - ✓ Meta-learning
- data representation
  - "stack more layers"
  - "we need to go deeper"
  - ✓ ?!?

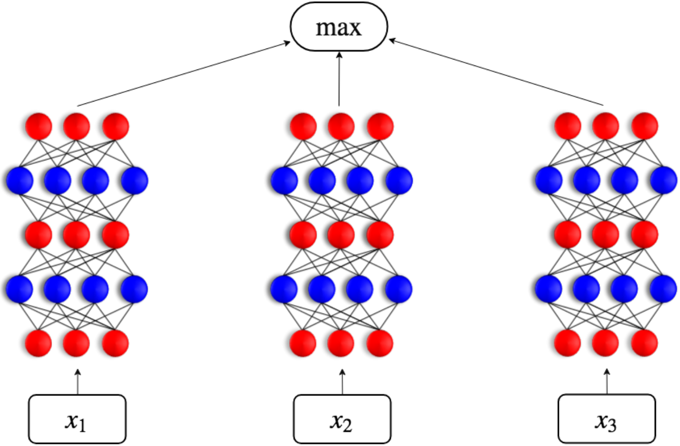
Section 2

Considering data structure

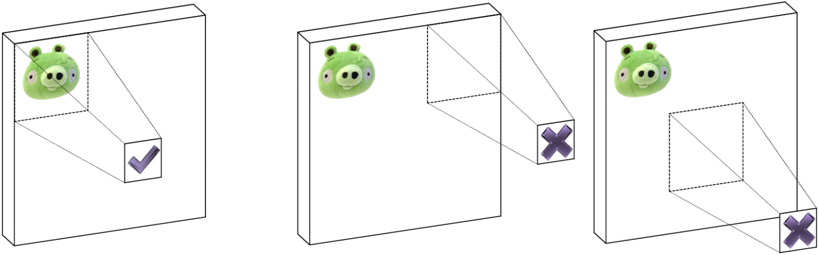
Pooling invariants



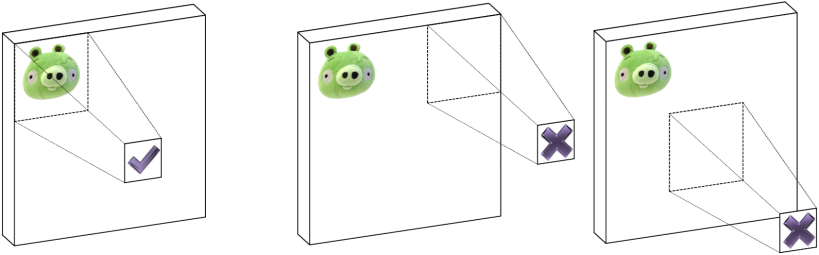
Pooling invariants



Translation invariance



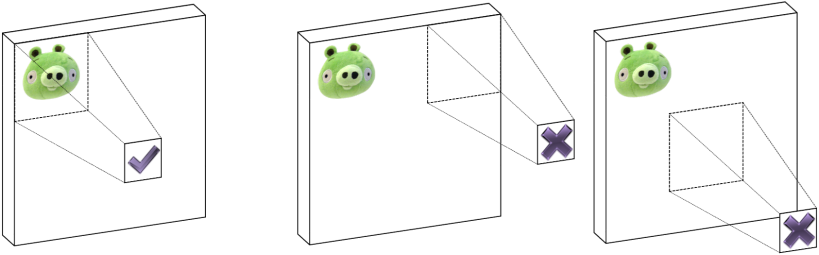
Translation invariance



Usually followed by:

- max pooling (one invariant is of a particular interest)
  - other pooling options possible

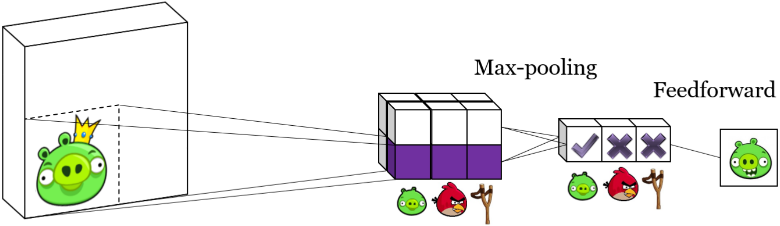
Translation invariance



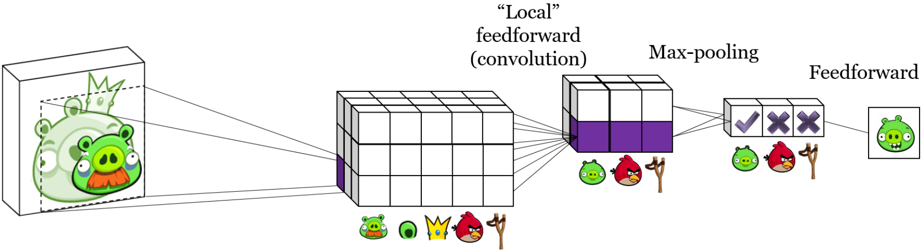
Usually followed by:

- max pooling (one invariant is of a particular interest)
  - other pooling options possible
- concatenation (for subtasks of same structure)

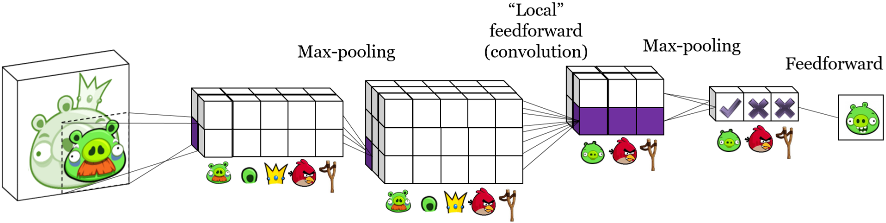
Size invariance



Size invariance

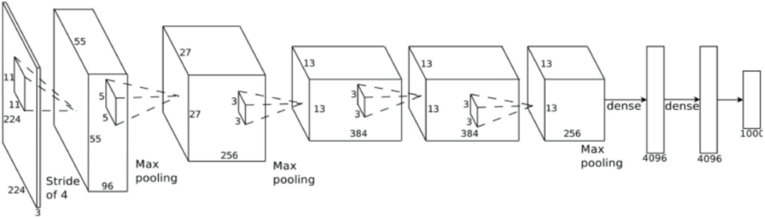


Size invariance

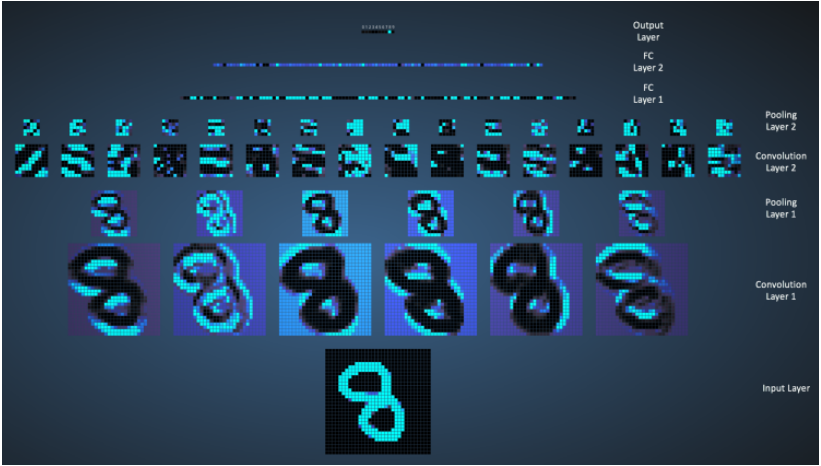


Convolutional neural network (CNN)

Resulting network:

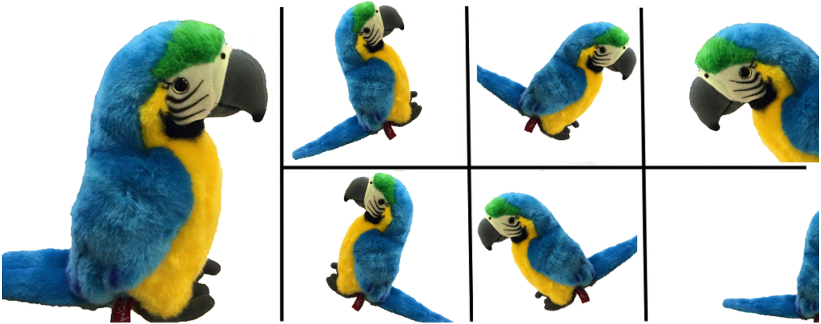


Convolutional neural network (CNN)



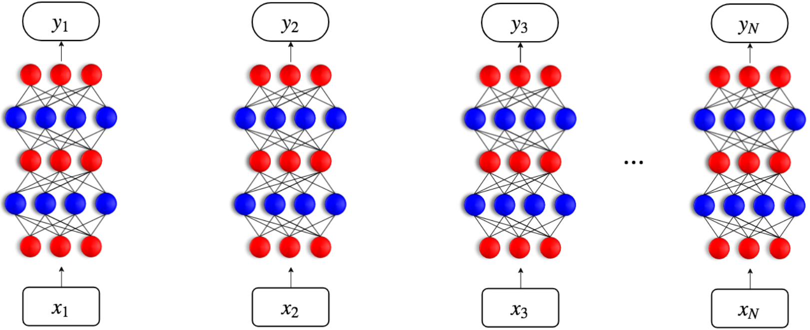
Augmentation

If you can't consider invariants in architecture, enlarge your dataset.



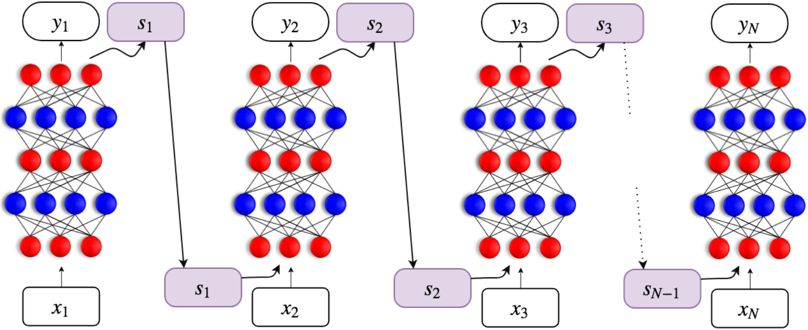
Sequences as input

Applying same idea:

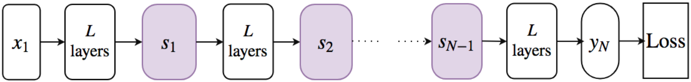


Sequences as input

Naive approach:



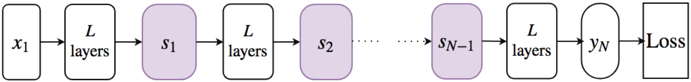
Gradients problem



Problem:

Gradient is required to pass LN layers.

Gradients problem

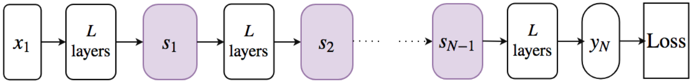


Problem:

Gradient is required to pass LN layers.

Chain rule says it's multiplication of LN quantities.

Gradients problem



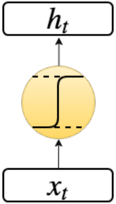
Problem:

Gradient is required to pass LN layers.

Chain rule says it's multiplication of LN quantities.

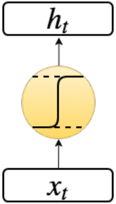
- most absolute values < 1: vanishing gradients problem
- most absolute values > 1: exploding gradients problem

Recurrent units

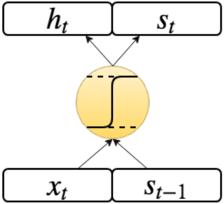


Neuron
(e.g. $\sigma(Ax_t)$)

Recurrent units

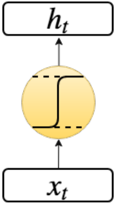


Neuron
(e.g. $\sigma(Ax_t)$)

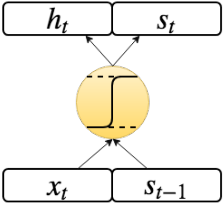


Same idea applied
(redundant)

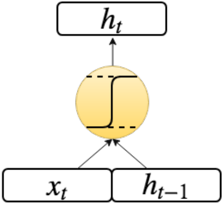
Recurrent units



Neuron
(e.g. $\sigma(Ax_t)$)

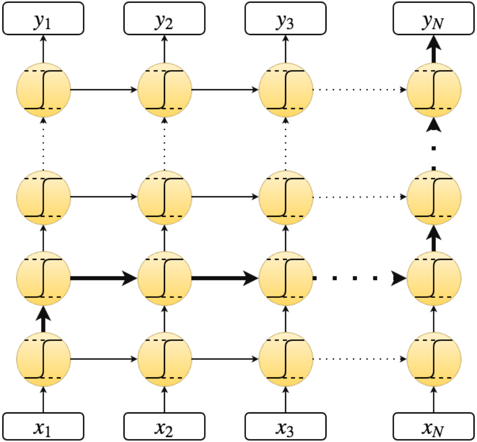


Same idea applied
(redundant)

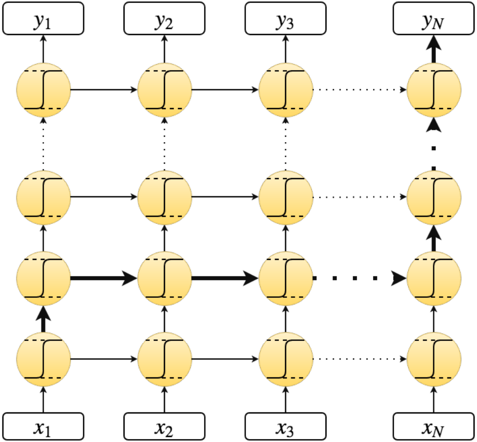


Recurrent neuron
(e.g. $\sigma(A[x_t, h_{t-1}])$)

Recurrent neural nets

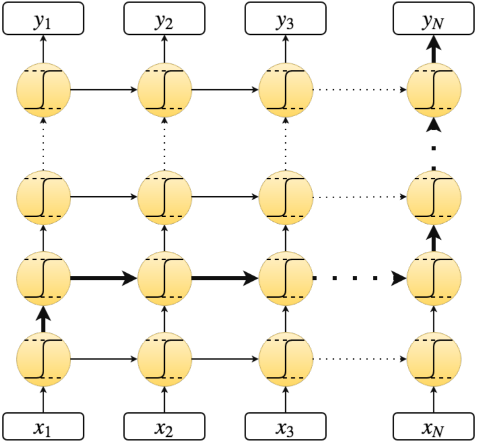


Recurrent neural nets



- ✓ $N + L$ layers for gradient to pass!

Recurrent neural nets



- ✓ $N + L$ layers for gradient to pass!
- ? Was previous option better at something?

Memory

Consider *writing to memory* task, i. e. the following operation:

```
if need_to_write(x):
    c = f(x)
```

How to express it in terms of computational graphs?

Memory

Consider *writing to memory* task, i. e. the following operation:

```
if need_to_write(x):
    c = f(x)
```

How to express it in terms of computational graphs?

```
c *= 1 - need_to_write(x)
c += need_to_write(x) * f(x)
```

Memory

Consider *writing to memory* task, i. e. the following operation:

```
if need_to_write(x):
    c = f(x)
```

How to express it in terms of computational graphs?

```
c *= 1 - need_to_forget(x)
c += need_to_write(x) * f(x)
```

Memory

Consider *writing to memory* task, i. e. the following operation:

```
if need_to_write(x):
    c = f(x)
```

How to express it in terms of computational graphs?

```
c *= 1 - need_to_forget(x)
c += need_to_write(x) * f(x)
```

Memory update formula

$$c_t = f_t \circ c_{t-1} + w_t \circ f(x_t) \quad w_t, f_t \in \{0, 1\}$$

where $\circ$ is element-wise multiplication.

Gates

w_t, f_t are also some functions of input! For example,

$$\mathbb{I}[Ax_t > 0]$$

Gates

w_t, f_t are also some functions of input! For example,

$$\mathbb{I}[Ax_t > 0]$$

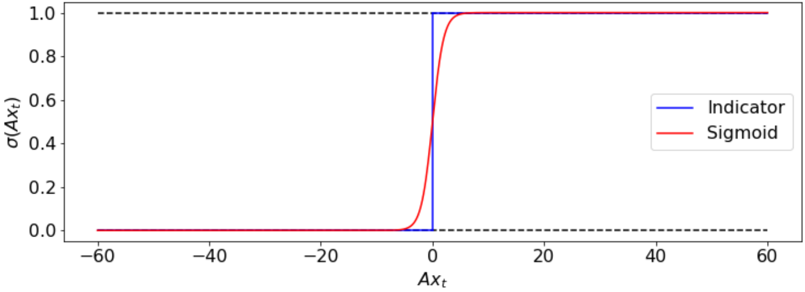
DL main rule: if something is not differentiable, make a smooth (*soft*) version of it!

Gates

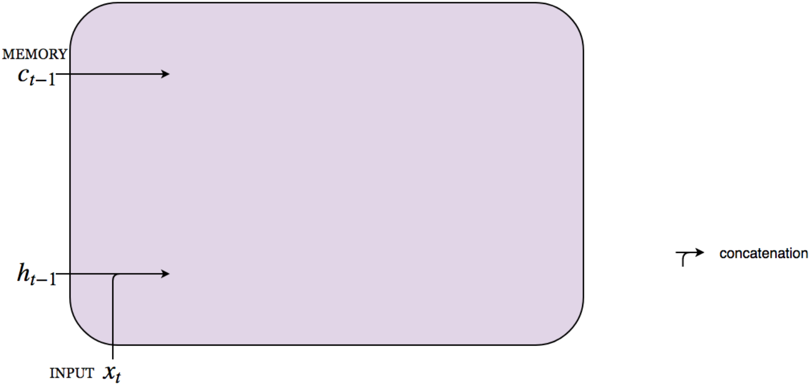
w_t, f_t are also some functions of input! For example,

$$\mathbb{I}[Ax_t > 0]$$

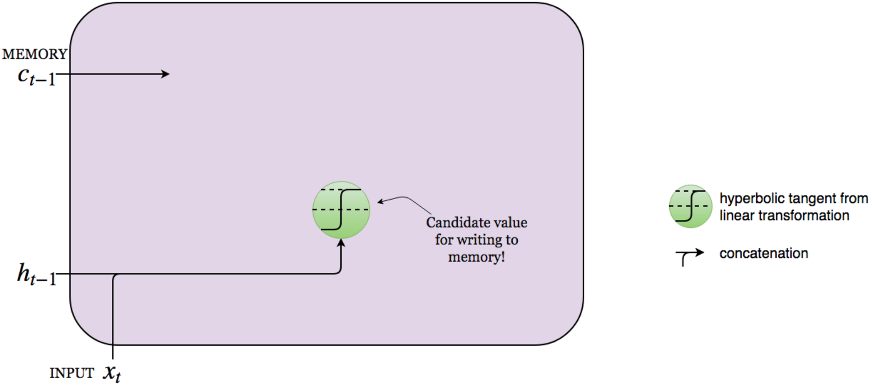
DL main rule: if something is not differentiable, make a smooth (*soft*) version of it!



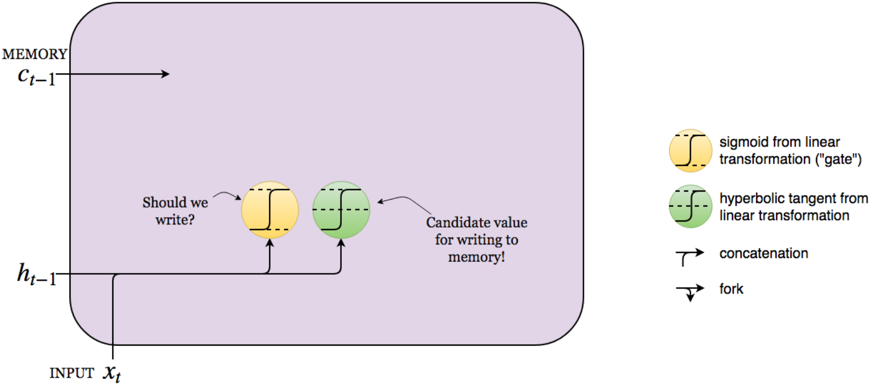
LSTM: recurrent neurons with memory.



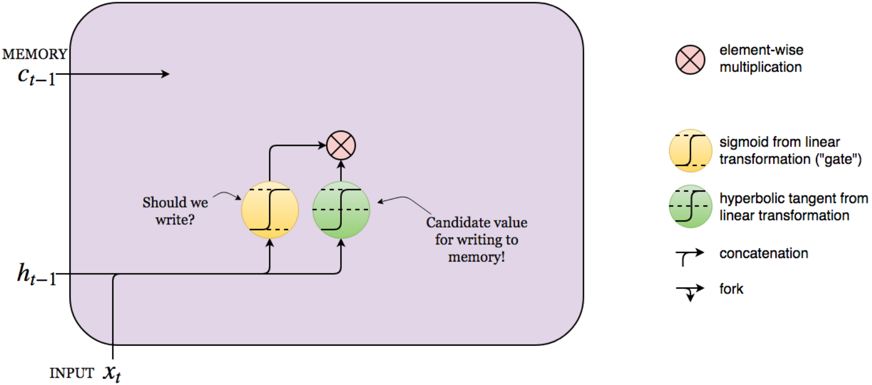
LSTM: transforming data: $c'_t = \tanh(A_c\,[x_t, h_{t-1}])$



LSTM: writing gate: $w_t = \sigma(A_w [x_t, h_{t-1}])$

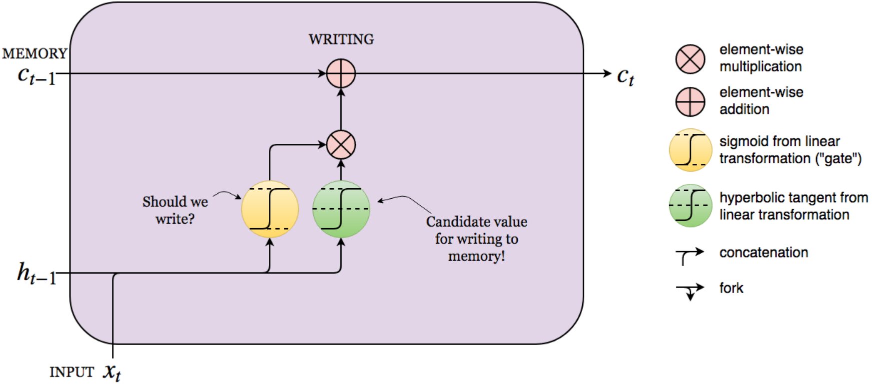


$$\text{LSTM: } c_t = f_t \circ c_{t-1} + w_t \circ c'_t$$

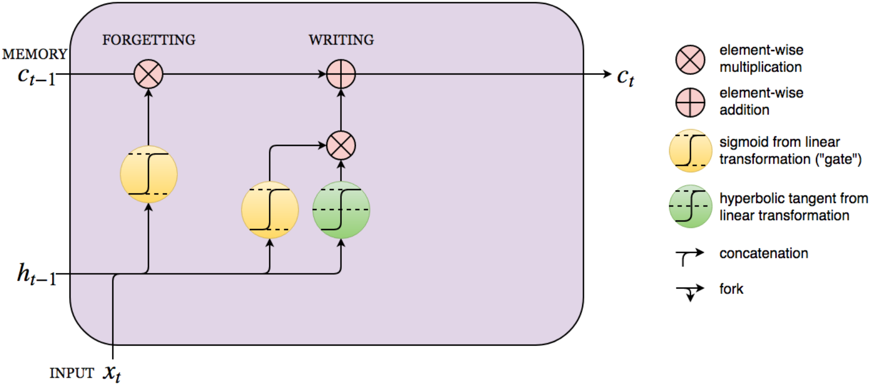


Long Short-Term Memory (LSTM)

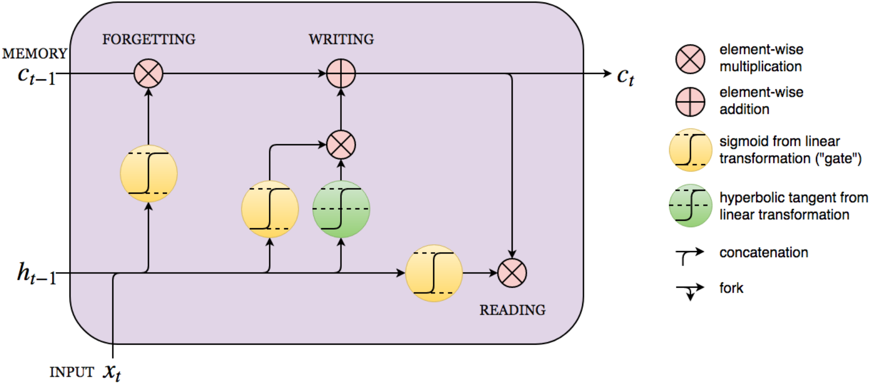
LSTM: $c_t = f_t \circ c_{t-1} + w_t \circ c'_t$



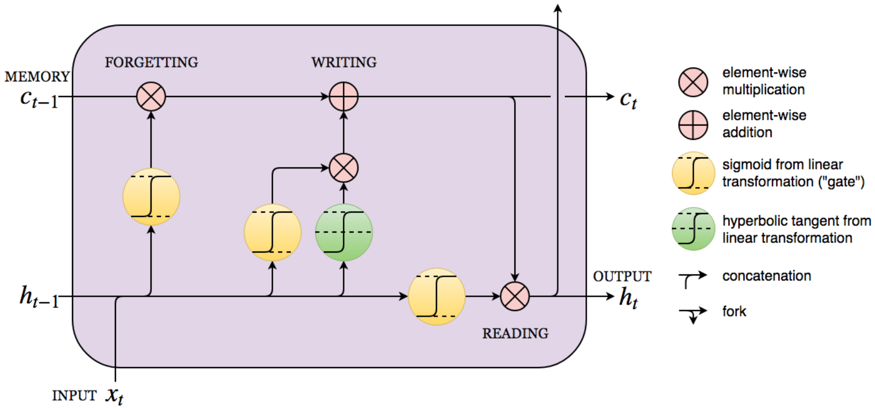
LSTM: $c_t = f_t \circ c_{t-1} + w_t \circ c'_t$



LSTM: $h_t = r_t \circ c_t$



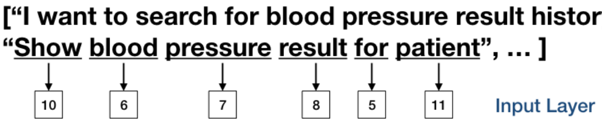
LSTM: full scheme



Section 3

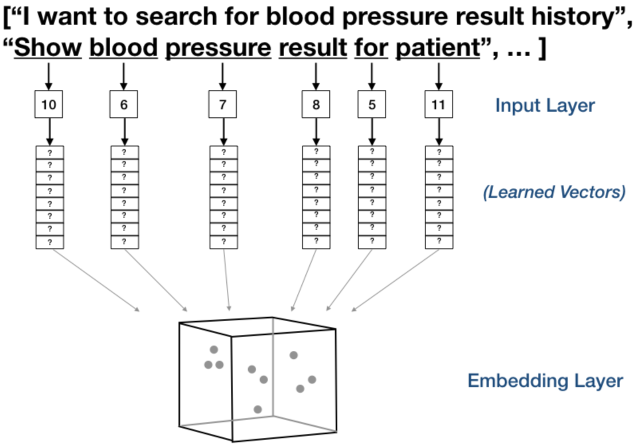
Data representation

Word embeddings



i	1
want	2
to	3
search	4
for	5
blood	6
pressure	7
result	8
history	9
show	10
patient	11
...	
LAST	20

Word embeddings



i	1
want	2
to	3
search	4
for	5
blood	6
pressure	7
result	8
history	9
show	10
patient	11
...	
LAST	20

Embeddings can be:

- learned end-to-end

Embeddings can be:

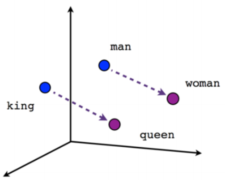
- learned end-to-end
- learned separately via special algorithms like Word2Vec

Embeddings can be:

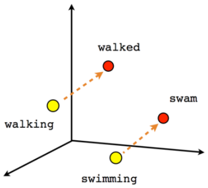
- learned end-to-end
- learned separately via special algorithms like Word2Vec
- pre-trained (which is an example of *transfer learning*)

Embeddings can be:

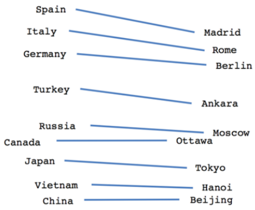
- learned end-to-end
- learned separately via special algorithms like Word2Vec
- pre-trained (which is an example of *transfer learning*)



Male-Female

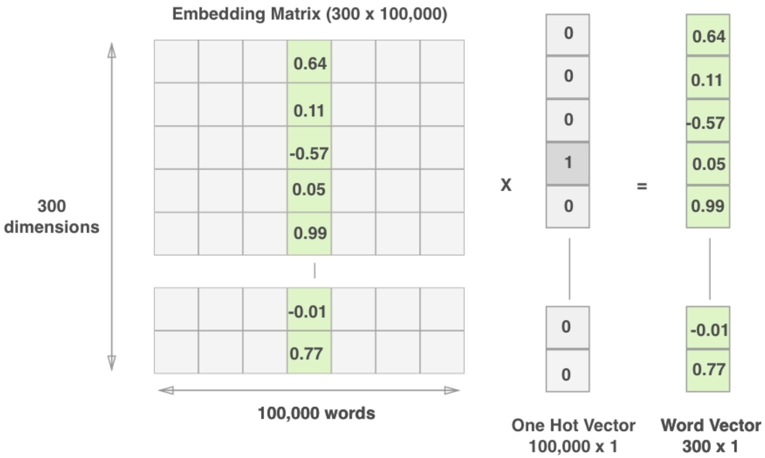


Verb tense

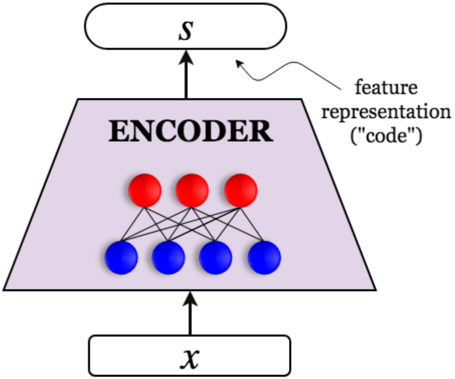


Country-Capital

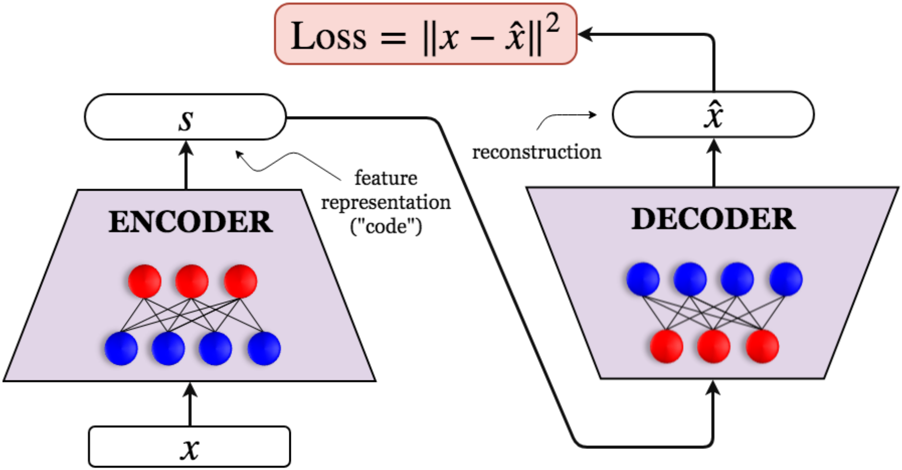
Embeddings demystified



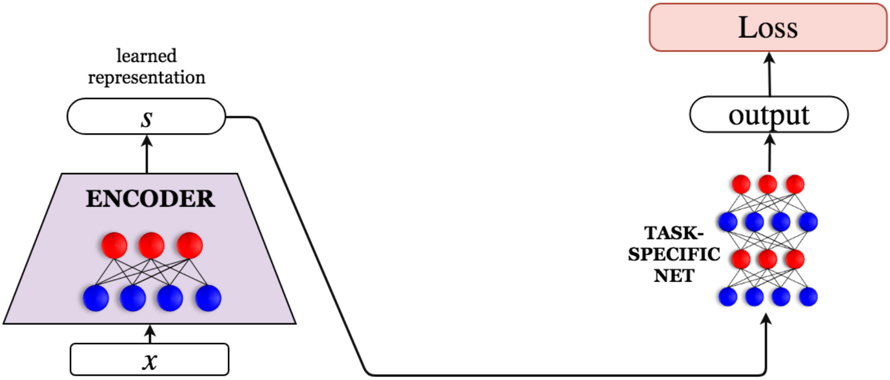
Autoencoder



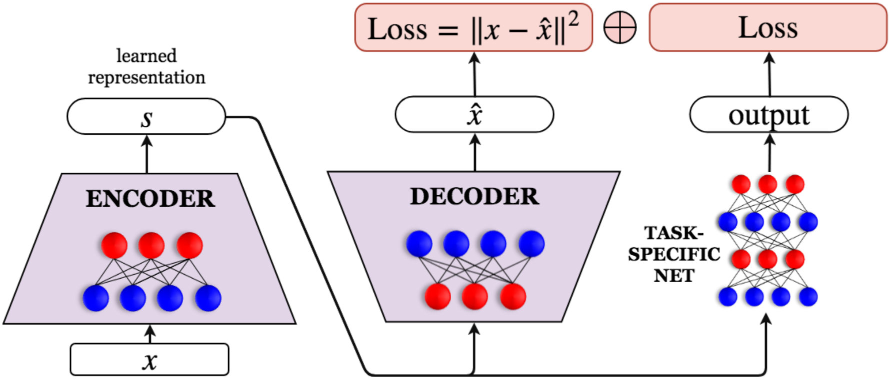
Autoencoder



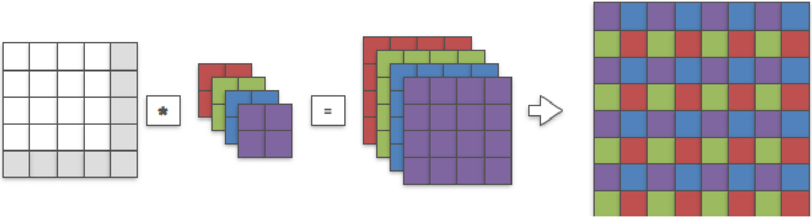
Possible usage

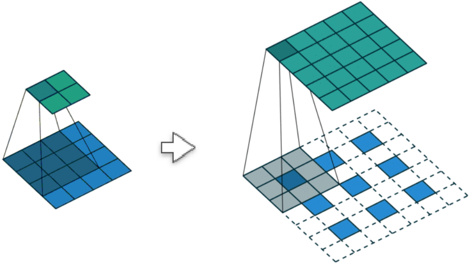


Possible usage

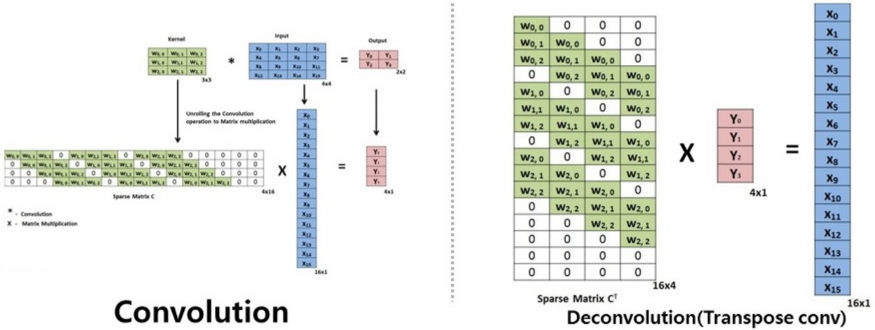


"Deconvolution"?



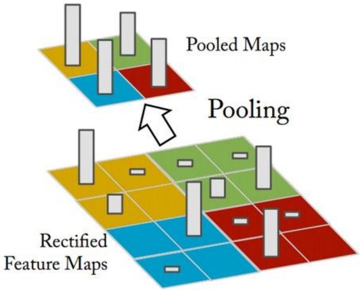


Transposed convolution

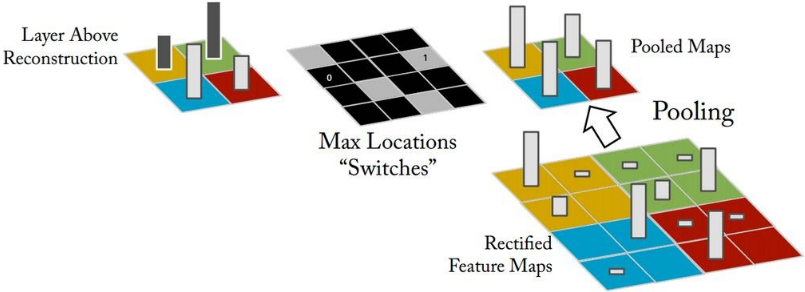


Unpooling

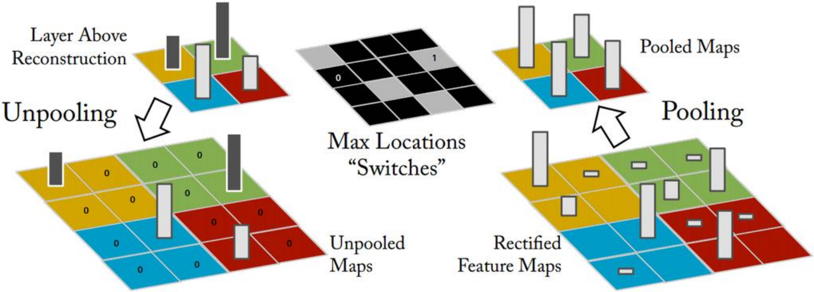

Layer Above
Reconstruction


Pooled Maps
Pooling
Rectified
Feature Maps

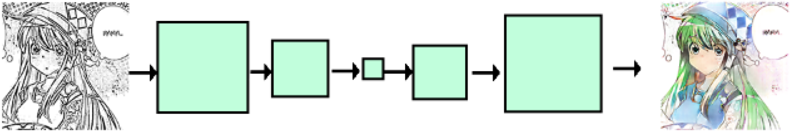
Unpooling

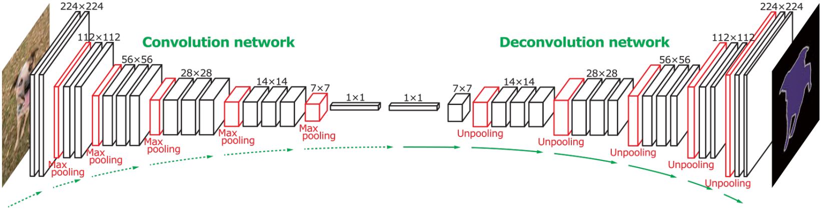

Layer Above
Reconstruction
0
1
Max Locations
"Switches"
Pooled Maps
Pooling
Rectified
Feature Maps

Unpooling

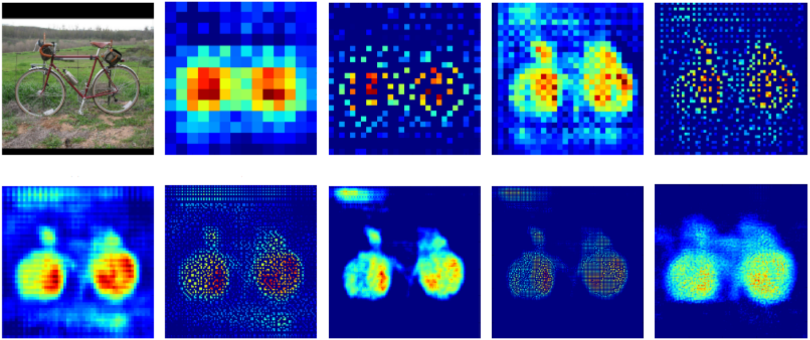

Layer Above
Reconstruction
Unpooling
Max Locations
"Switches"
Pooled Maps
Pooling
Unpooled
Maps
Rectified
Feature Maps

Examples

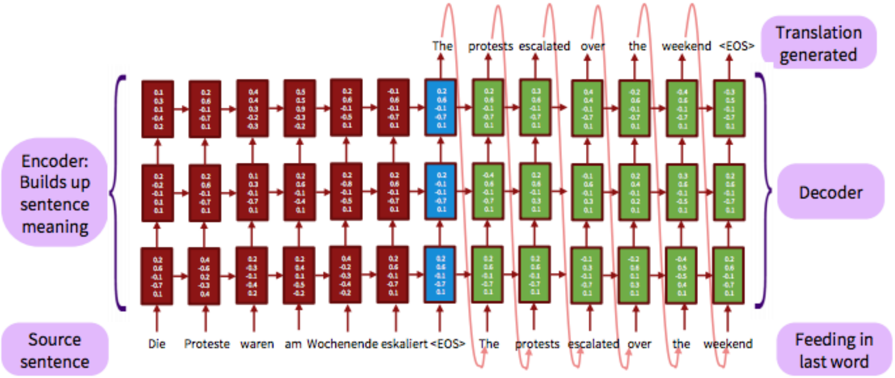



224×224
112×112
56×56
28×28
14×14
7×7
1×1
Convolution network
Deconvolution network
Max pooling
Unpooling

Inside decoder for segmentation

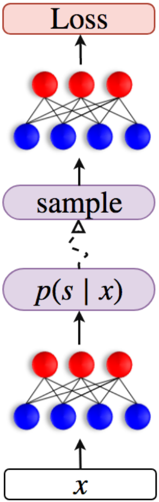


Machine translation



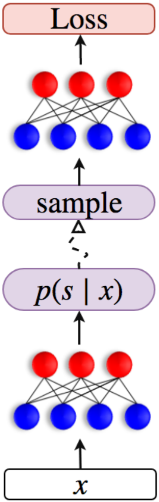
Section 4

Generative models



Stochastic nodes:

$$\mathbb{E}_{s \sim p(s|x,\theta)} f(s, \theta) \to \min_{\theta}$$



Stochastic nodes:

$$\mathbb{E}_{s \sim p(s|x,\theta)} f(s, \theta) \to \min_{\theta}$$

Where used:

- Hard attention mechanisms
- Reinforcement learning
- Generative models

REINFORCE[1]

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) = \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) \nabla_\theta \log p(s \mid x, \theta) + \\ + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s,\theta)$$

[1] see proof in appendix

REINFORCE[1]

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) = \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) \nabla_\theta \log p(s \mid x,\theta) + \\ + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s,\theta)$$

Monte-Carlo estimation

$$\approx f(s,\theta)\nabla_\theta \log p(s \mid x,\theta) + \nabla_\theta f(s,\theta), \quad s \sim p(s \mid x,\theta)$$

[1] see proof in appendix

REINFORCE[1]

$$\nabla_\theta \mathbb{E}_{s\sim p(s|x,\theta)} f(s,\theta) = \mathbb{E}_{s\sim p(s|x,\theta)} f(s,\theta) \nabla_\theta \log p(s \mid x,\theta) + \\ + \mathbb{E}_{s\sim p(s|x,\theta)} \nabla_\theta f(s,\theta)$$

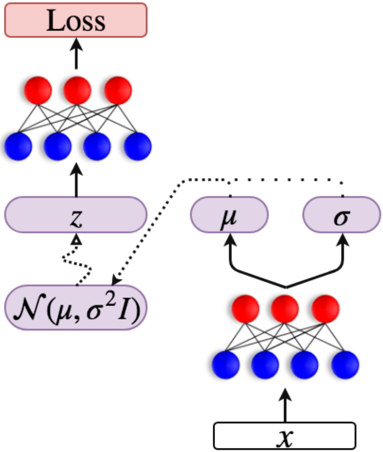
Monte-Carlo estimation

$$\approx f(s,\theta)\nabla_\theta \log p(s \mid x,\theta) + \nabla_\theta f(s,\theta), \quad s \sim p(s \mid x,\theta)$$

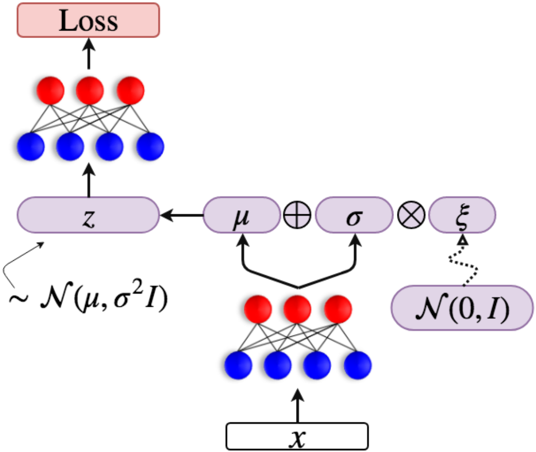
- ✓ universal approach
- × "high variance"

[1] see proof in appendix

Reparametrization trick



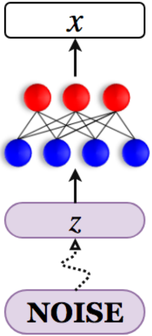
Reparametrization trick



Sampler

Suppose we want to model data distribution $p(x)$.
Problem: data space is usually too complex.

Sampler



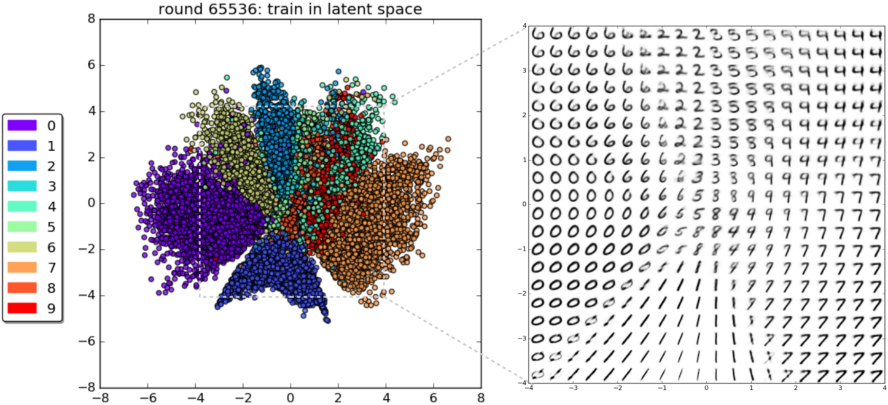
Suppose we want to model data distribution $p(x)$.
Problem: data space is usually too complex.

1. sample z from noise distribtuion, e. g. $\mathcal{N}(0, I)$

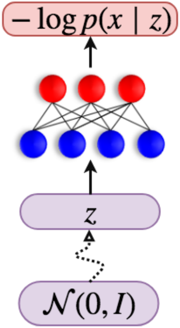
2(a). transform noise using neural net to object $x = f(z, \theta)$

2(b). sample $x \sim p(x \mid z, \theta)$

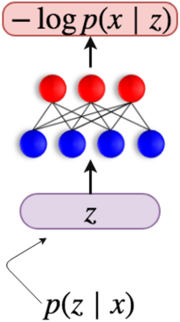
z contains all information about interdependencies!



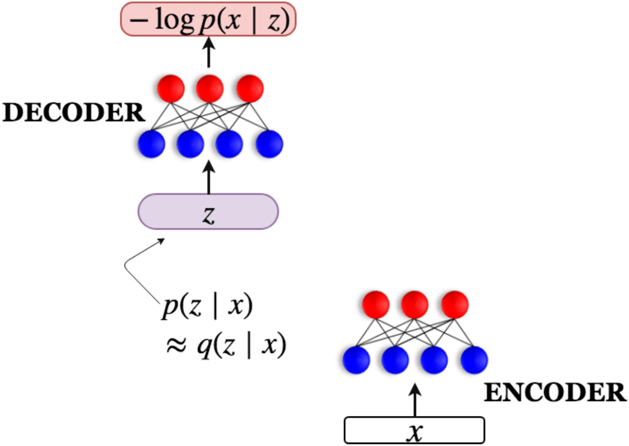
Variational AutoEncoder (VAE)



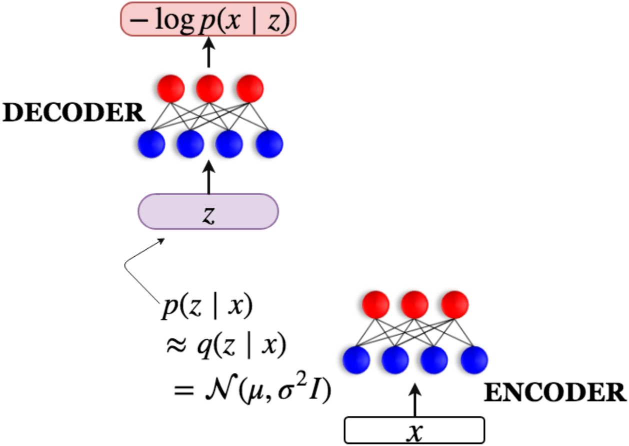
Variational AutoEncoder (VAE)



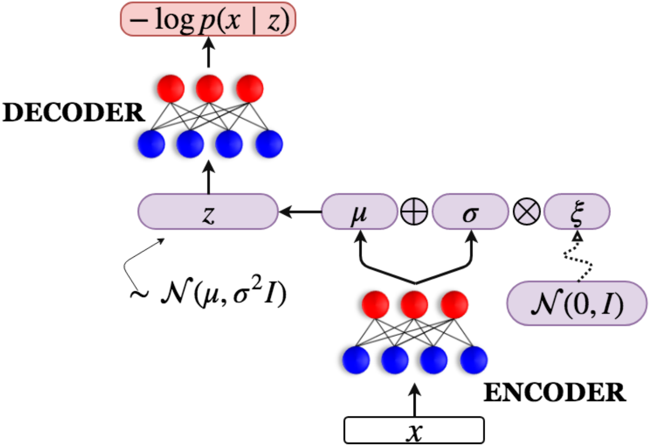
Variational AutoEncoder (VAE)



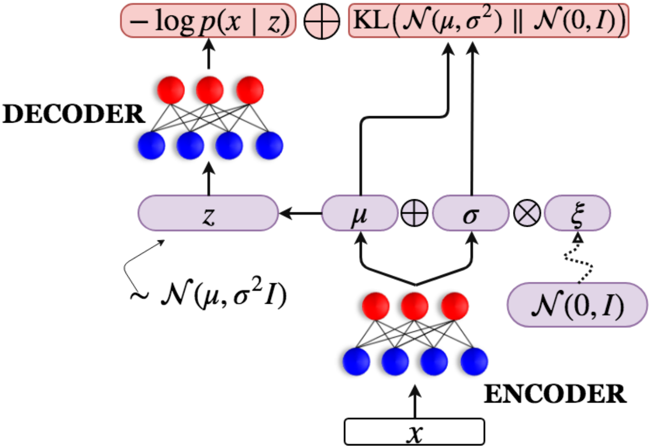
Variational AutoEncoder (VAE)



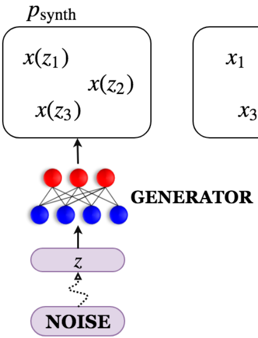
Variational AutoEncoder (VAE)



Variational AutoEncoder (VAE)

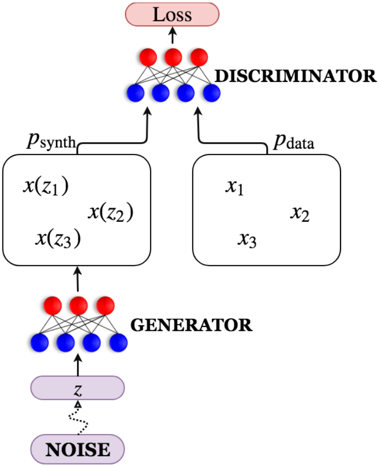


Generative Adversarial Networks (GAN)



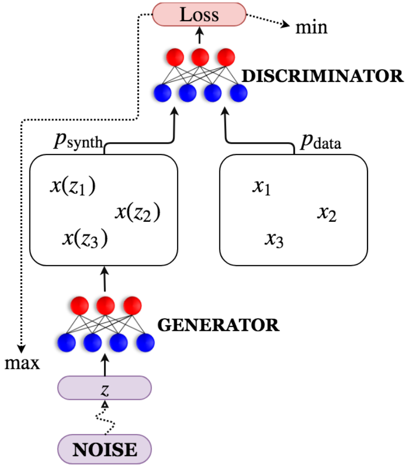
- most genius idea of our decade

Generative Adversarial Networks (GAN)



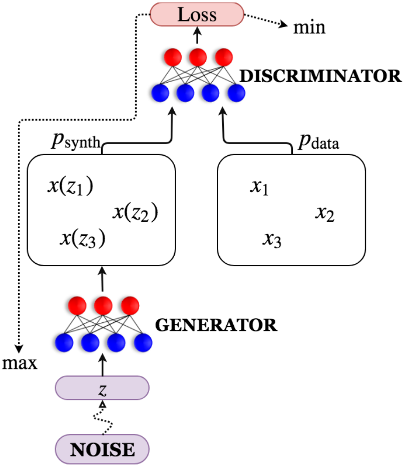
- most genius idea of our decade

Generative Adversarial Networks (GAN)



- most genius idea of our decade

Generative Adversarial Networks (GAN)



- most genius idea of our decade
- :) universal approach!
- :(adversarial training is unstable

Section 5

APPENDIX

REINFORCE derivation pt.1

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s, \theta) \quad =$$

REINFORCE derivation pt.1

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) \quad = \quad \nabla_\theta \int\limits_s p(s \mid x, \theta) f(s,\theta) ds =$$

REINFORCE derivation pt.1

$$\nabla_\theta \mathbb{E}_{s\sim p(s|x,\theta)} f(s,\theta) = \nabla_\theta \int\limits_s p(s \mid x,\theta) f(s,\theta) ds =$$

$$= \Big\{$$  $$\Big\} = \int\limits_s \nabla_\theta \left(p(s \mid x,\theta) f(s,\theta)\right) ds =$$

REINFORCE derivation pt.1

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) = \nabla_\theta \int\limits_s p(s \mid x,\theta) f(s,\theta) ds =$$

$= \{$  $\}$

$$= \int\limits_s \nabla_\theta \left(p(s \mid x,\theta) f(s,\theta) \right) ds =$$

$$= \int\limits_s \nabla_\theta p(s \mid x,\theta) f(s,\theta) ds + \int\limits_s p(s \mid x,\theta) \nabla_\theta f(s,\theta) ds =$$

REINFORCE derivation pt.1

$$\nabla_\theta \mathbb{E}_{s \sim p(s|x,\theta)} f(s,\theta) = \nabla_\theta \int_s p(s \mid x,\theta) f(s,\theta) ds =$$

$$= \{$$  $$\} = \int_s \nabla_\theta \left(p(s \mid x,\theta) f(s,\theta) \right) ds =$$

$$= \int_s \nabla_\theta p(s \mid x,\theta) f(s,\theta) ds + \int_s p(s \mid x,\theta) \nabla_\theta f(s,\theta) ds =$$

$$= \int_s \nabla_\theta p(s \mid x,\theta) f(s,\theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s,\theta) = \dots$$

REINFORCE derivation pt.2

$$\ldots \quad = \quad \int_s \nabla_\theta p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) =$$

REINFORCE derivation pt.2

Log-derivative trick

$$\nabla_\theta p(s \mid x, \theta) = p(s \mid x, \theta) \nabla_\theta \log p(s \mid x, \theta)$$

$$\ldots = \int_s \nabla_\theta p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s \mid x, \theta)} \nabla_\theta f(s, \theta) =$$

REINFORCE derivation pt.2

Log-derivative trick

$$\nabla_\theta p(s \mid x, \theta) = p(s \mid x, \theta) \nabla_\theta \log p(s \mid x, \theta)$$

$$\begin{aligned} \ldots &= \int_s \nabla_\theta p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) = \\ &= \int_s p(s \mid x, \theta) \nabla_\theta \log p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) = \end{aligned}$$

REINFORCE derivation pt.2

Log-derivative trick

$$\nabla_\theta p(s \mid x, \theta) = p(s \mid x, \theta)\nabla_\theta \log p(s \mid x, \theta)$$

$$\begin{aligned} \ldots &= \int_s \nabla_\theta p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) = \\ &= \int_s p(s \mid x, \theta) \nabla_\theta \log p(s \mid x, \theta) f(s, \theta) ds + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) = \\ &= \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta \log p(s \mid x, \theta) f(s, \theta) + \mathbb{E}_{s \sim p(s|x,\theta)} \nabla_\theta f(s, \theta) \end{aligned}$$

VAE: notation

Suppose we have:

- $p(z)$ — some fixed distribution
- $p_\theta(x \mid z)$ — distribution with parameters θ
- $q_\phi(z \mid x)$ — approximation of $p_\theta(z \mid x)$ (which is intractable for us) with parameters ϕ

VAE: notation

Suppose we have:

- $p(z)$ — some fixed distribution
- $p_\theta(x \mid z)$ — distribution with parameters θ
- $q_\phi(z \mid x)$ — approximation of $p_\theta(z \mid x)$ (which is intractable for us) with parameters ϕ

By definition, $p_\theta(x) = \int\limits_z p_\theta(x \mid z)p(z)dz$ is a function of θ and is also intractable.

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) =$$

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \log p_\theta(x) \int_z q_\phi(z \mid x) dz \quad =$$

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \log p_\theta(x) \int_z q_\phi(z \mid x) dz = \int_z q_\phi(z \mid x) \log p_\theta(x) dz =$$

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \log p_\theta(x) \int_z q_\phi(z \mid x) dz = \int_z q_\phi(z \mid x) \log p_\theta(x) dz =$$

$$= \int_z q_\phi(z \mid x) \log \frac{p_\theta(x) p_\theta(z \mid x)}{p_\theta(z \mid x)} dz =$$

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \log p_\theta(x) \int\limits_z q_\phi(z \mid x) dz = \int\limits_z q_\phi(z \mid x) \log p_\theta(x) dz =$$

$$= \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x) p_\theta(z \mid x)}{p_\theta(z \mid x)} dz = \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x, z)}{p_\theta(z \mid x)} dz =$$

VAE: Treating latent variables

For arbitrary $q_\phi(z \mid x)$:

$$\log p_\theta(x) = \log p_\theta(x) \int\limits_z q_\phi(z \mid x) dz = \int\limits_z q_\phi(z \mid x) \log p_\theta(x) dz =$$

$$= \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x) p_\theta(z \mid x)}{p_\theta(z \mid x)} dz = \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x, z)}{p_\theta(z \mid x)} dz =$$

$$= \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x, z) q_\phi(z \mid x)}{p_\theta(z \mid x) q_\phi(z \mid x)} dz$$

Split into summation of three components:

$$\begin{aligned}\log p_\theta(x) &= \int\limits_z q_\phi(z \mid x) \log \frac{p_\theta(x \mid z)}{q_\phi(z \mid x)} dz + \\ &+ \int\limits_z q_\phi(z \mid x) \log \frac{p(z)}{q_\phi(z \mid x)} dz + \\ &+ \int\limits_z q_\phi(z \mid x) \log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\end{aligned}$$

KL-divergence

For two distributions $p(\xi), q(\xi)$ with shared domain:

$$\mathrm{KL}(p(\xi) \parallel q(\xi)) := \int_{\xi} p(\xi) \log \frac{p(\xi)}{q(\xi)} d\xi \geq 0$$

$$\begin{aligned}
\log p_\theta(x) \quad & = \text{data term} & \mathbb{E}_{z \sim q_\phi(z|x)} \log p_\theta(x \mid z) - \\
& -\text{prior coherence} & \mathrm{KL}(q_\phi(z \mid x) \parallel p(z)) + \\
& +\text{approximation error} & \mathrm{KL}(q_\phi(z \mid x) \parallel p_\theta(z \mid x))
\end{aligned}$$

VAE justification

Variational lower bound

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z|x)} \log p_\theta(x \mid z) - \mathrm{KL}(q_\phi(z \mid x) \parallel p(z))$$

VAE justification

Variational lower bound

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z \mid x)} \log p_\theta(x \mid z) - \mathrm{KL}(q_\phi(z \mid x) \parallel p(z))$$

For every θ there is $q_\phi(z \mid x)$ so that inequality turns into equality (when $q_\phi(z \mid x) = p_\theta(z \mid x)$ almost everywhere)

---

if q is a model of *enough capacity*, i. e. can model any distribution

VAE justification

Variational lower bound

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z \mid x)} \log p_\theta(x \mid z) - \mathrm{KL}(q_\phi(z \mid x) \parallel p(z))$$

For every θ there is $q_\phi(z \mid x)$ so that inequality turns into equality (when $q_\phi(z \mid x) = p_\theta(z \mid x)$ almost everywhere) $\Rightarrow$ optimization of $\log p_\theta(x)$ is equivalent to optimization of lower bound.

---

if q is a model of *enough capacity*, i. e. can model any distribution