Introduction to deep learning

Armand Joulin

Facebook AI Research

Announcement

- Assignment 3 is out:

  http://www.di.ens.fr/willow/teaching/recvis17/assignment3/

Introduction

Cancer detection

Self driving cars

AlphaGo

- Many applications are based on “deep learning”
- But what is exactly “deep learning”?
- We are going to do an overview of this field

Plan of this lecture

- Supervised neural networks
- Optimization for neural networks
- Convolutional network
- Recurrent network
- Unsupervised learning

Plan of this lecture

- **Supervised neural networks**
- Optimization for neural networks
- Convolutional network
- Recurrent network
- Unsupervised learning

Linear classifier

- **Supervised problem**: predict output *Y* given input *X*
- **Classification problem**: *Y* is discrete
- **Linear classifier**: direct mapping between *X* and *Y*
- For example:

$$\text{Dataset : } \left(X^{(i)}, Y^{(i)}\right) \text{ pairs, } i = 1, \ldots, N.$$

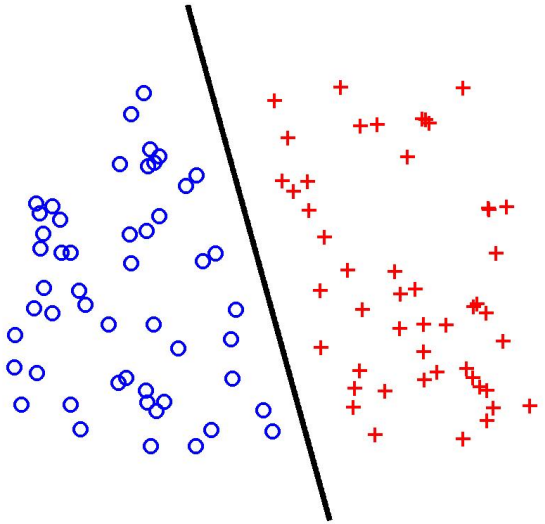
$$X^{(i)} \in \mathbb{R}^n, \; Y^{(i)} \in \{-1, 1\}.$$

$$\text{Goal : Find } w \text{ and } b \text{ such that } \operatorname{sign}(w^\top X^{(i)} + b) = Y^{(i)}.$$

From N. Le Roux's 2016 lecture

Linear classifier

- Linear models work well on **linearly separable data**



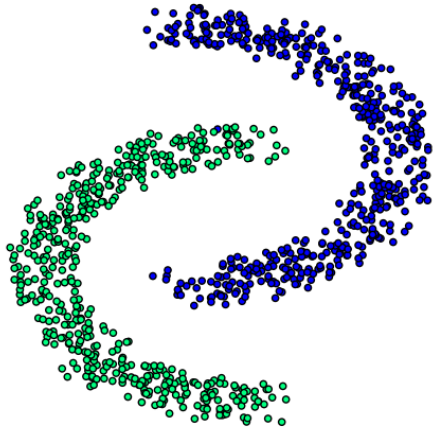
Perceptron algorithm (Rosenblatt, 57)

- $w_0 = 0, b_0 = 0$
- $\widehat{Y}^{(i)} = \text{sign}(w^\top X^{(i)} + b)$
- $w_{t+1} \leftarrow w_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right) X^{(i)}$
- $b_{t+1} \leftarrow b_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right)$

From N. Le Roux's 2016 lecture

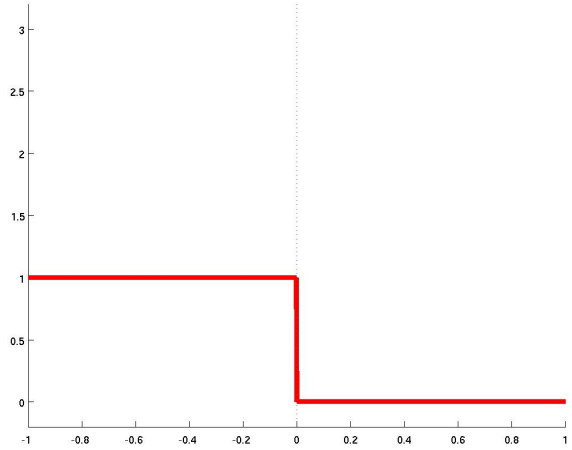
Linear classifier

- How does a Perceptron work on **non-linearly separable data**?

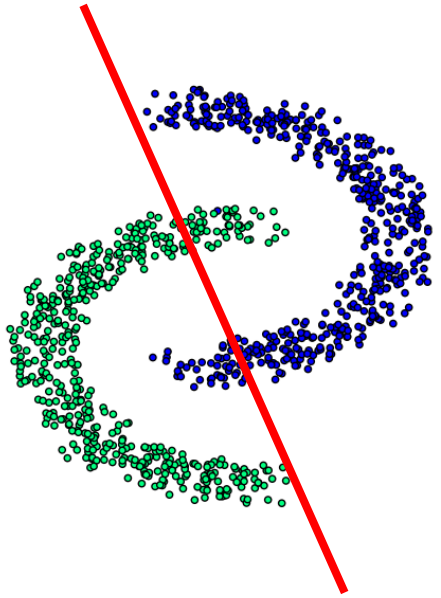


- It doesn’t **converge**!

Why is the Perceptron non-converging?

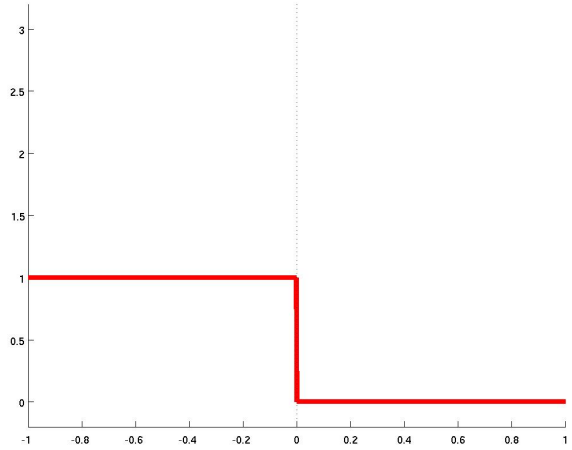


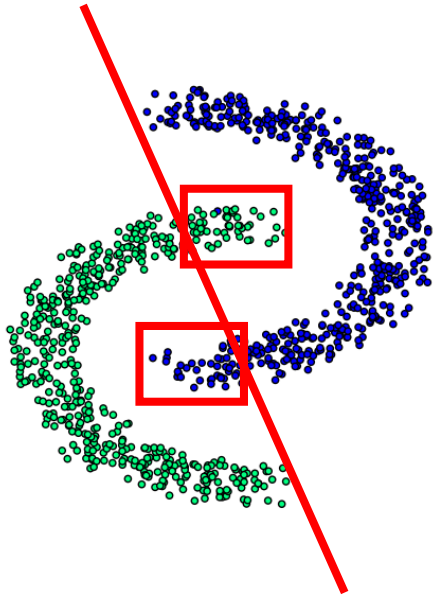
- Non-smooth loss function
- Every misclassified point cost 1



Perceptron classifier

Why is the Perceptron non-converging?

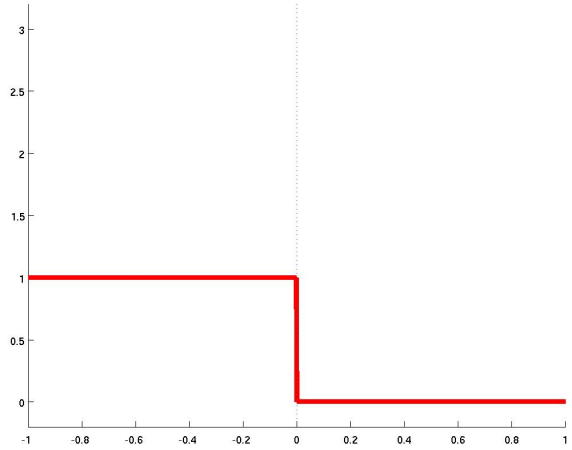


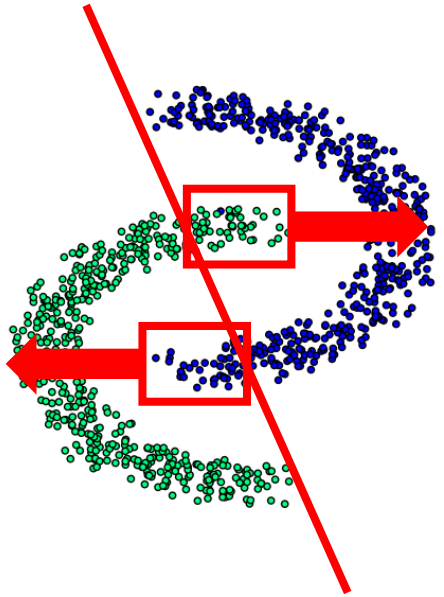


- Non-smooth loss function
- Every misclassified point cost 1

Perceptron classifier

Why is the Perceptron non-converging?

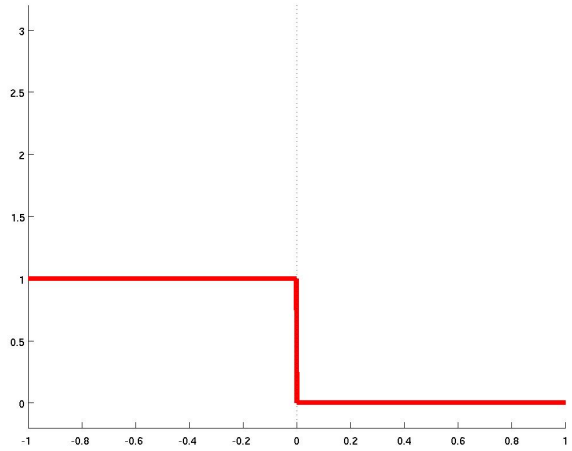


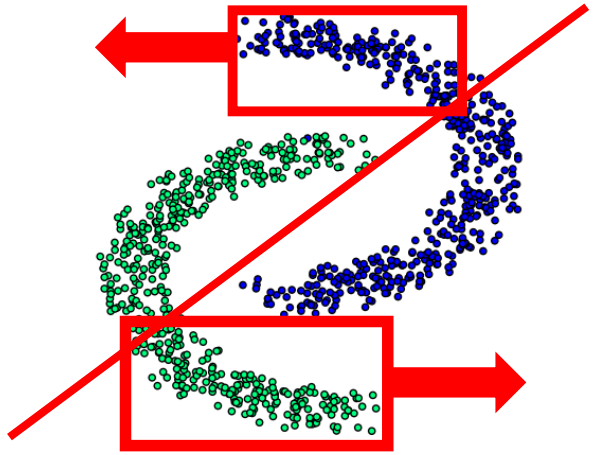


- Non-smooth loss function
- Every misclassified point cost 1

Perceptron classifier

Why is the Perceptron non-converging?



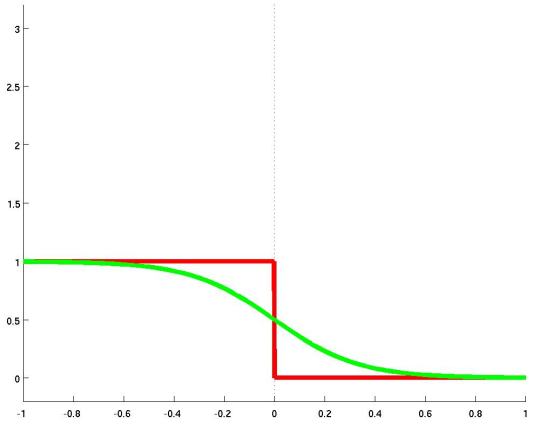


- Non-smooth loss function
- Every misclassified point cost 1

→ Very Unstable

Perceptron classifier

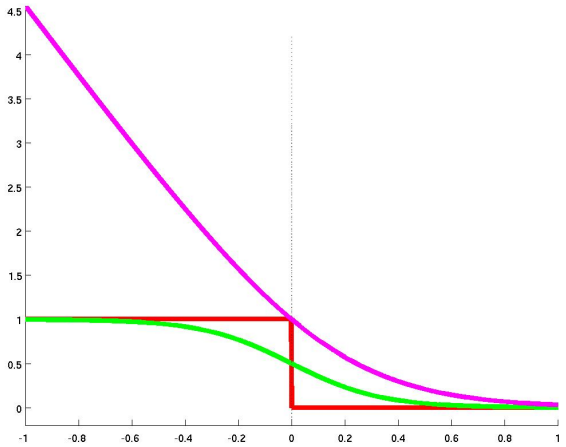
Sigmoid function



- Good approximation of the 0-1 loss:
  - Smooth → stable
  - Not Convex → many poor solutions

From N. Le Roux's 2016 lecture

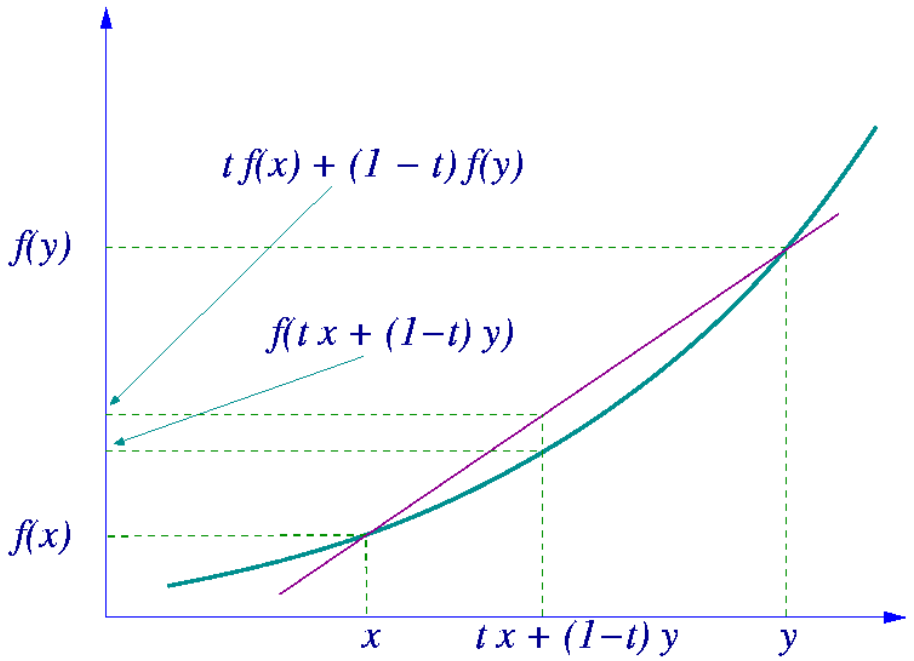
Logistic Regression (in purple)



- Good approximation of the 0-1 loss:
  - Smooth → stable
  - Convex → only one solution!

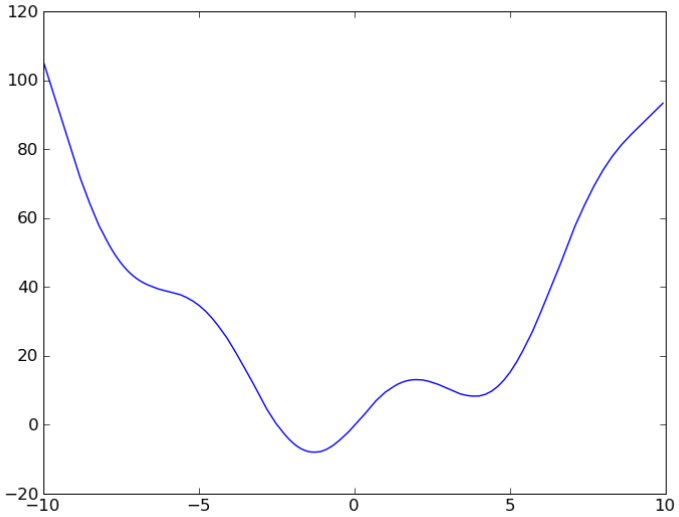
From N. Le Roux's 2016 lecture

What is convex?



From N. Le Roux's 2016 lecture

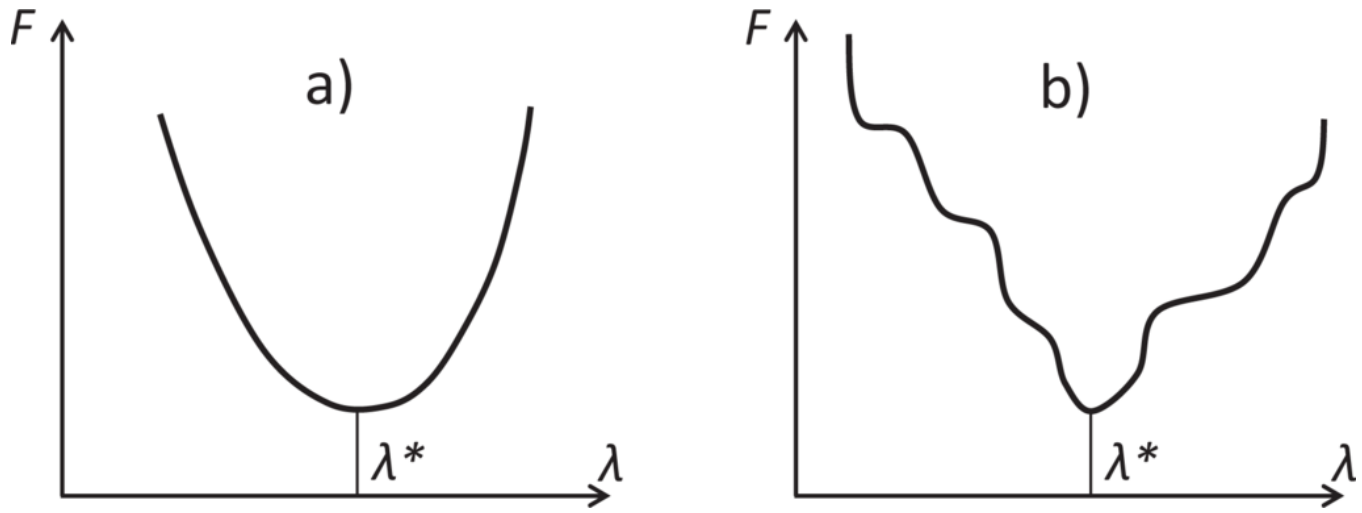
Why is convex important?



- Non-convex functions can have many poor local minima

From N. Le Roux's 2016 lecture

Kind of convex is ok



- Else Neural networks wouldn't work!

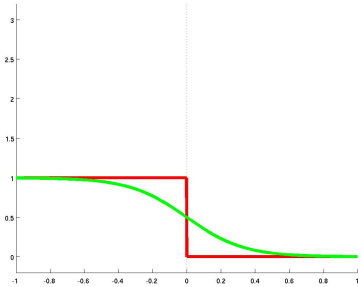
From N. Le Roux's 2016 lecture

Perceptron algorithm (Rosenblatt, 57)

- $w_0 = 0, b_0 = 0$
- $\widehat{Y}^{(i)} = \text{sign}(w^\top X^{(i)} + b)$
- $w_{t+1} \leftarrow w_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right) X^{(i)}$
- $b_{t+1} \leftarrow b_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right)$

From N. Le Roux's 2016 lecture

Logistic Regression



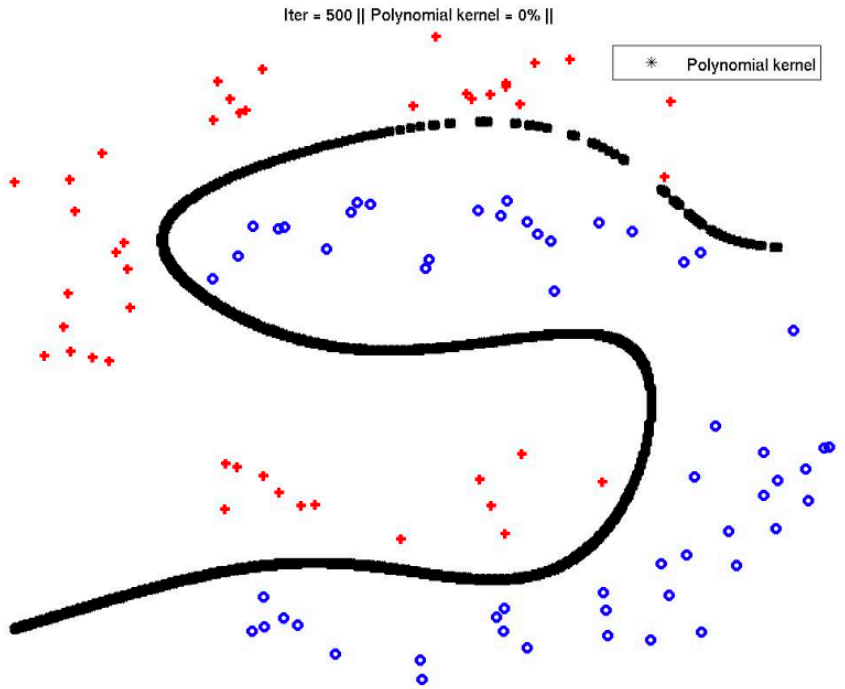
SIGM vs SIGN

- $w_0 = 0, b_0 = 0$
- $\widehat{Y}^{(i)} = \text{SIGM}\left(w^\top X^{(i)} + b\right)$
- $w_{t+1} \leftarrow w_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right) X^{(i)}$
- $b_{t+1} \leftarrow b_t + \sum_i \left(Y^{(i)} - \widehat{Y}^{(i)}\right)$

Updates comes from the gradient of the logistic loss

From N. Le Roux's 2016 lecture

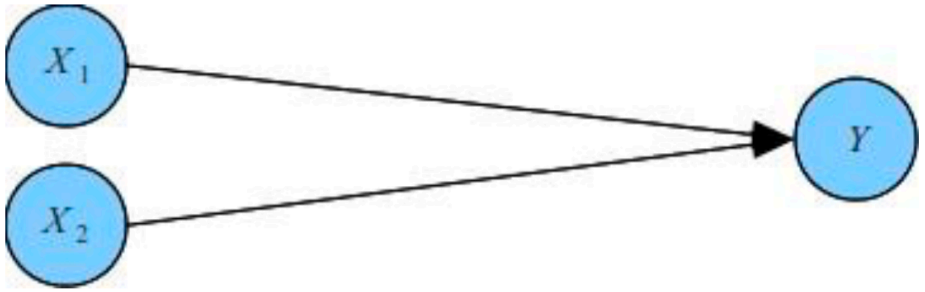
Non-linear classifier



- Features : $X_1, X_2 \rightarrow$ linear classifier
- Features : $X_1, X_2, X_1X_2, X_1^2, \ldots \rightarrow$ non-linear classifier

From N. Le Roux's 2016 lecture

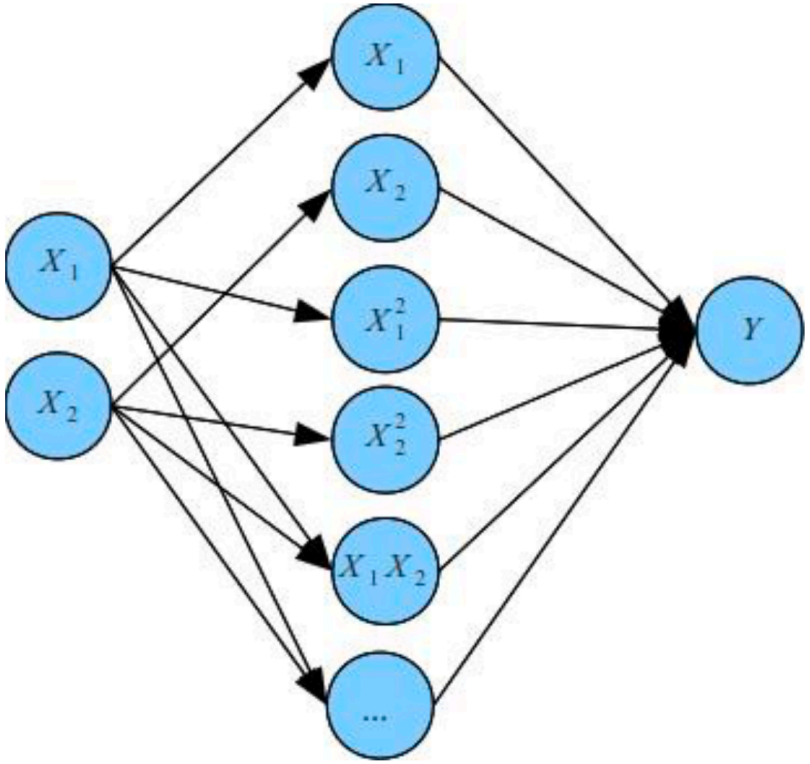
A graphical view of the classifiers



$$f(X) = w_1 X_1 + w_2 X_2 + b$$

From N. Le Roux's 2016 lecture

A graphical view of the classifiers



$$f(X) = w_1X_1 + w_2X_2 + w_3X_1^2 + w_4X_2^2 + w_5X_1X_2 + \dots$$

From N. Le Roux's 2016 lecture

Non-linear classifier

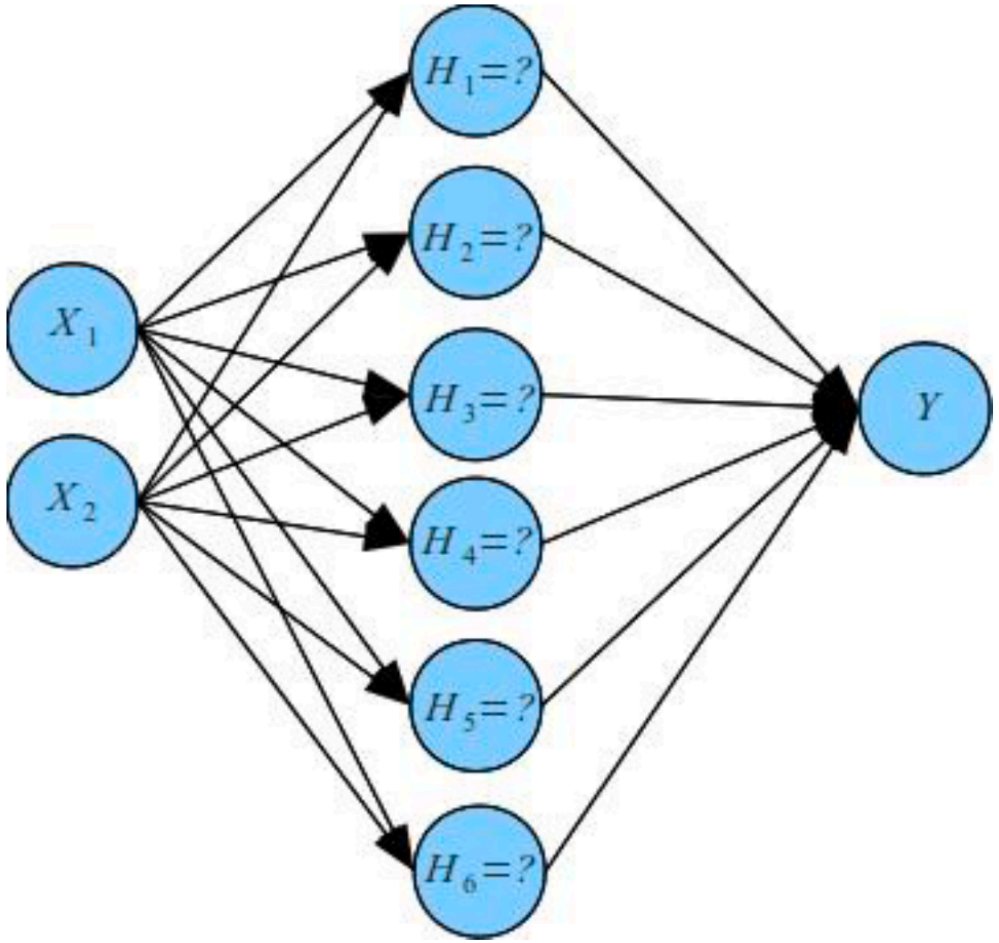
- Linear classifier on non-linear features is a **non-linear classifier**
- Choosing the non-linear features is **hard**
- Example of non-linear features: Kernel methods
- Can we automatically learn them? Goal of **Neural networks!**

Simple neural network

1. Pick an example x
2. Transform it in $\hat{x} = Vx$ with some matrix V
3. Apply a nonlinear function g to all the elements of $\hat{x}$
4. Compute $w^\top \hat{x} + b = \underbrace{w^\top g(Vx)}_{\text{Non-linear}} + b$

From N. Le Roux's 2016 lecture

Simple neural network



- Usually, we use

  $H_j = g(v_j^\top X)$

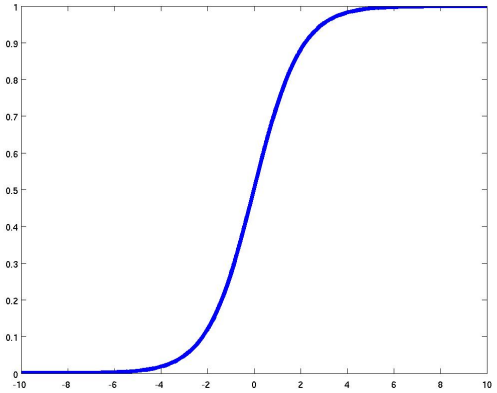
- H_j : Hidden unit
- v_j : Input weight
- g : Transfer function

From N. Le Roux's 2016 lecture

Simple neural network

$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^\top X) + b$$

- *g* is an *activation function* or *transfer function*



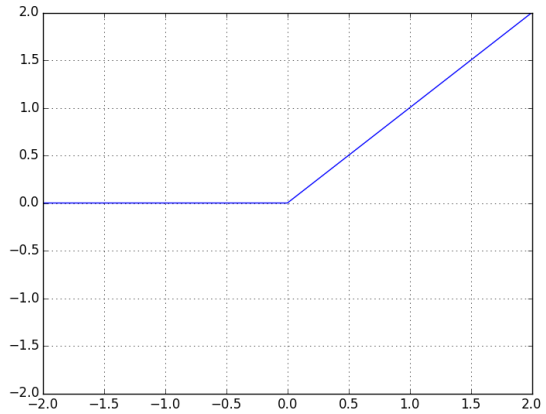
- *g* is often a **sigmoid** or a **tanh**

From N. Le Roux's 2016 lecture

Simple neural network

$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^\top X) + b$$

- g is an *activation function* or *transfer function*



- Nowadays g is sometimes a **Rectified Linear Unit (ReLU)**

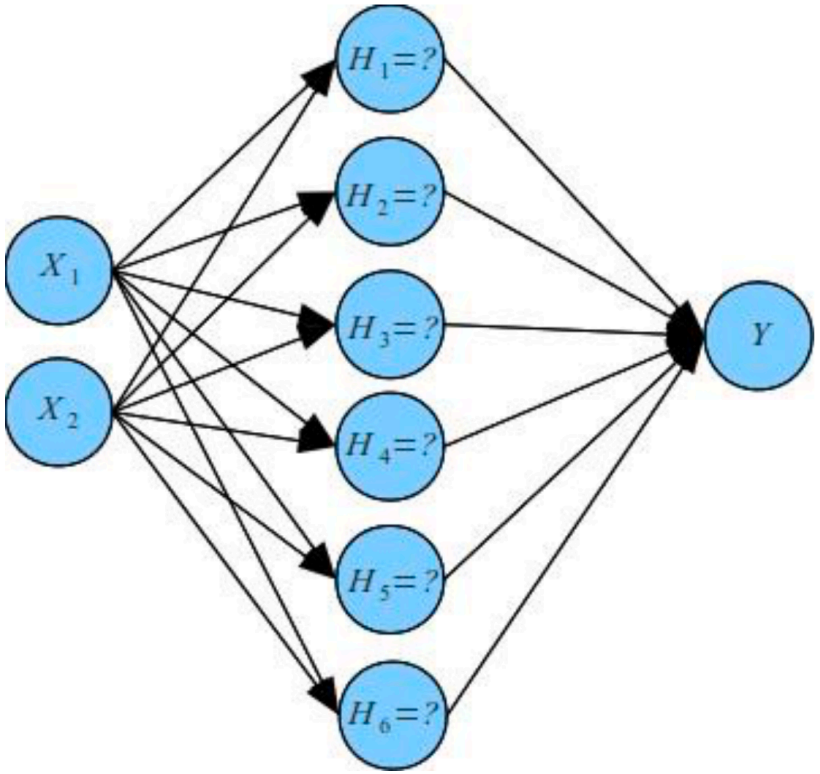
Maas et al. 2013

From N. Le Roux's 2016 lecture

Why these non-linearities?

- Mostly differentiable → can compute the gradient (aka **backpropagation**)
- With ``enough'' non-linear hidden units → **universal approximation**!

Simple neural network



$$f(X) = \sum_j w_j H_j(X) + b = \sum_j w_j g(v_j^\top X) + b$$

From N. Le Roux's 2016 lecture

Train a neural network

- Dataset : $\left(X^{(i)}, Y^{(i)}\right)$ pairs, $i = 1, \ldots, N$.
- Goal : Find V and W to minimize

$$\sum_i \ell(f(X^{(i)}), Y^{(i)}) = \sum_i \ell(Wg(VX^{(i)}), Y^{(i)})$$

Where ℓ is a loss function (e.g. logistic regression)

From N. Le Roux's 2016 lecture

Standard neural network learning

- **Forward pass**: apply your network to produce output
- **Evaluation**: Compute the error given by your loss function
- **Backward pass**: propagate this error through the network, i.e. compute the gradient
- **Update** the parameters with gradient descent

Backpropagation

- What is “backpropagation”?
- → Simply computing the gradient
- gradient computed by “back-propagating the error from output to input”
- This process is also known as “chain rule” for function composition

Backpropagation

- Consider the problem: $f_{W,V}(X) = W\,g(V\,X)$

$$\ell(W,V) = \ell(f_{W,V}(X), Y)$$

- Apply **chain rule** to compute gradient of the loss for *W*:

$$\frac{\delta \ell(W,V)}{\delta W} = \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} \frac{\delta f_{W,V}(X)}{\delta W}$$

$$= \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} g(V\,X)^T$$

Backpropagation

- Apply **chain rule** to compute gradient of the loss for *V*:

$$\begin{aligned}
\frac{\delta \ell(W,V)}{\delta V} &= \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} \frac{\delta f_{W,V}(X)}{\delta V} \\
&= \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} \frac{\delta \left(W g(VX)\right)}{\delta V} \\
&= \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} W \frac{\delta g(VX)}{\delta V} \\
&= \left.\frac{\delta \ell(Z,Y)}{\delta Z}\right|_{Z=f_{W,V}(X)} W g'(VX) X^T
\end{aligned}$$

Backpropagation

- Consider the problem: $f_{W,V}(X) = W g(VX)$

$$\ell(W, V) = \ell(f_{W,V}(X), Y)$$

- Apply **chain rule** to compute gradient of the loss w.r.t *V* and *W*:

$$\frac{\delta \ell(W,V)}{\delta W} = \frac{\delta \ell(Z,Y)}{\delta Z}\Big|_{Z=f_{W,V}(X)} g(VX)^T$$

$$\frac{\delta \ell(W,V)}{\delta V} = \frac{\delta \ell(Z,Y)}{\delta Z}\Big|_{Z=f_{W,V}(X)} W g'(VX) X^T$$

Backpropagation - example

- In the case of logistic regression:

$$\frac{\ell(W,V)}{f(X)} = Y - f(X)$$

- This is the **prediction error**.
- And it gets propagating from the output to the input:

$$\frac{\delta\ell(W,V)}{\delta W} = (Y - f(X))g(VX)^T$$

$$\frac{\delta\ell(W,V)}{\delta V} = W(Y - f(X))g'(VX)X^T$$

Limit of a simple neural network

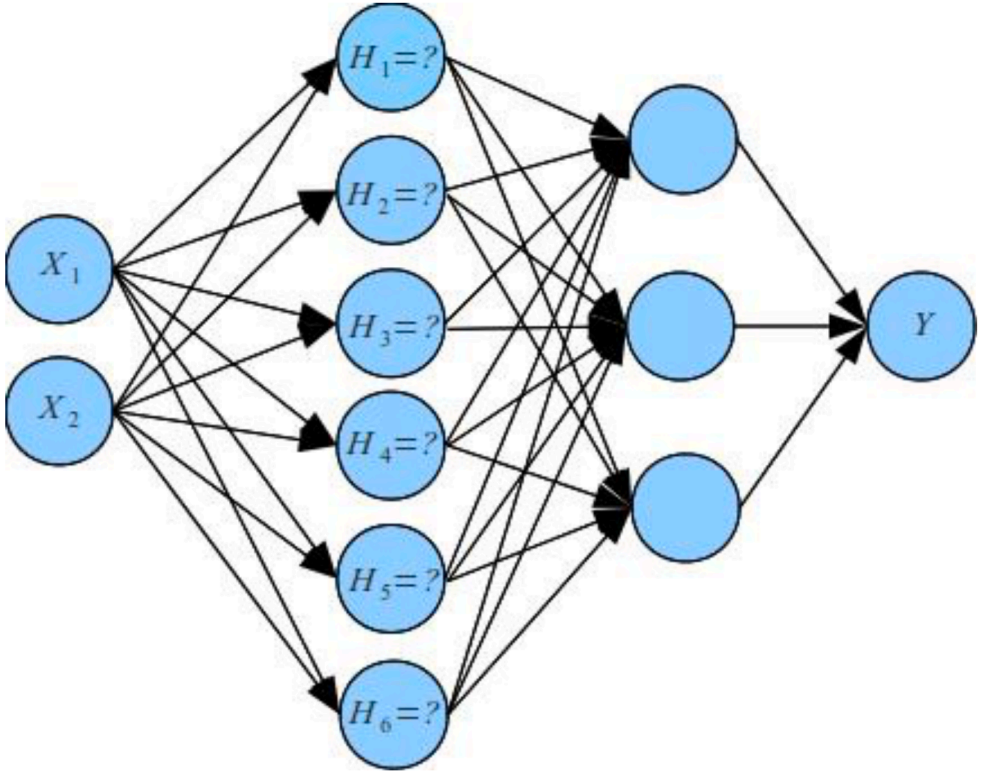
- With ``enough of hidden units'' → **universal approximation**!

- What is "enough" non-linear hidden units? → can be a lot

- Example:

A **parity function over the integers** cannot be approximated with a simple neural network and finite number of neurons. (Papert & Minsky, 1969)

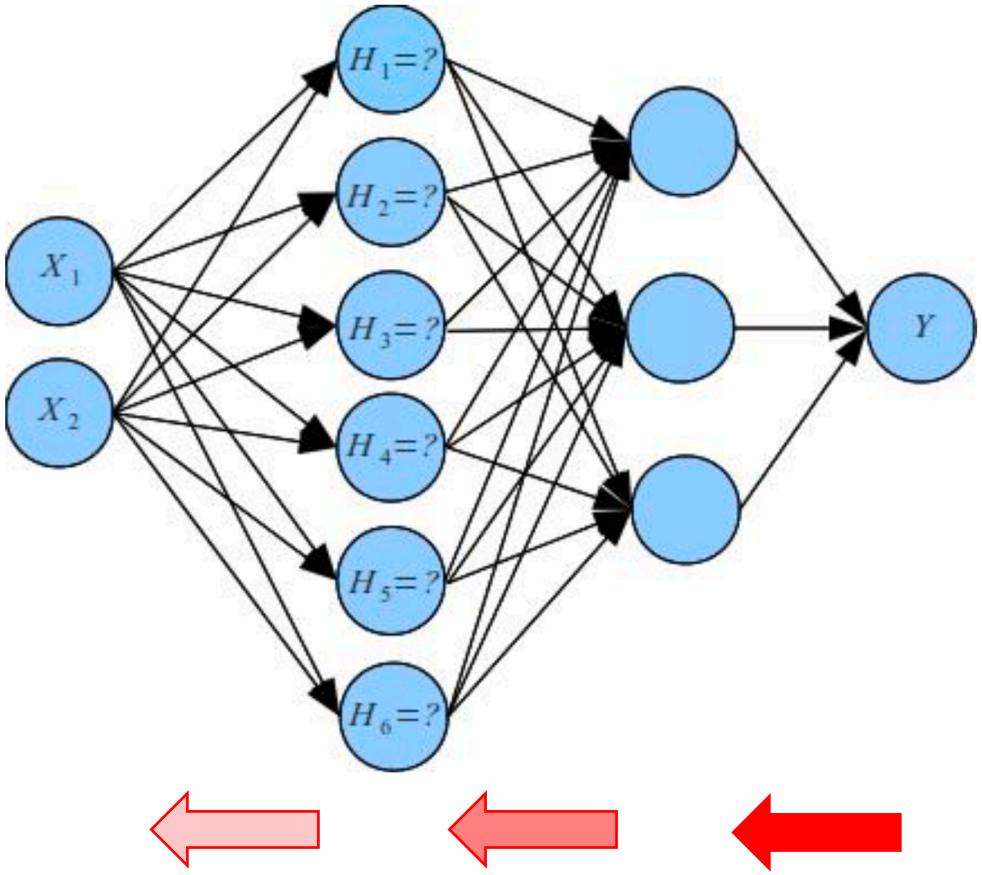
Solution: go deeper!



- For example 3 layers... But we could go deeper!
- **Pro**: more function can be approximated with less parameters
- **Cons**:
  - harder to optimize
  - overfitting

From N. Le Roux's 2016 lecture

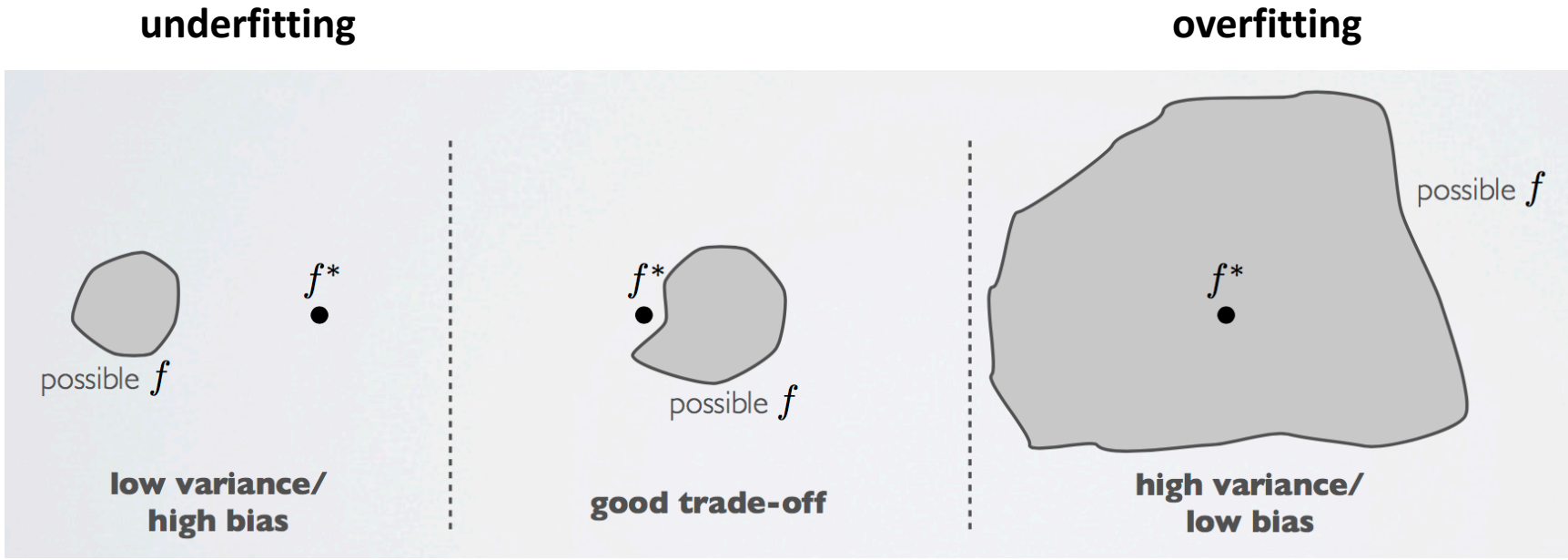
Why is it harder to optimize?



Gradient flow from output to input

- **Vanishing gradient problem**
- As gradient flow from output to input → **norm decreases**
- True for sigmoid, not for ReLU

What is overfitting?



- **Overfitting**: too many learnable functions, not enough data

- **Solutions**: use a regularization

Slide taken from Hugo Larochelle

Regularization

- Several strategies exists:
  - Use a smaller network
  - Don’t learn too much (aka “**early stopping**”)
  - Constrain the norm of the parameters (aka **weight decay**)
  - Force redundancy in hidden units (e.g., **dropout**)

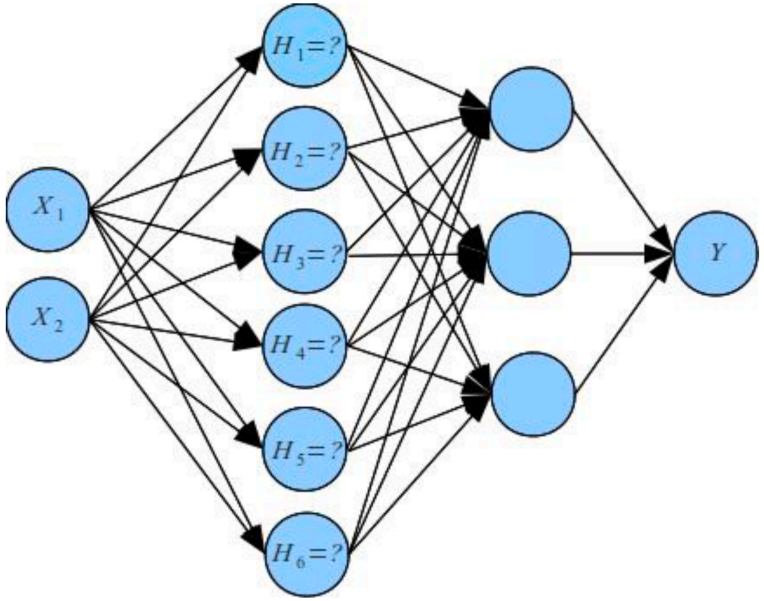
Dropout

- Avoid unit to get too specialized
- By forcing the model to work even when many units are *dropped out*

Srivastava et al., 2014

Dropout

Iteration 0

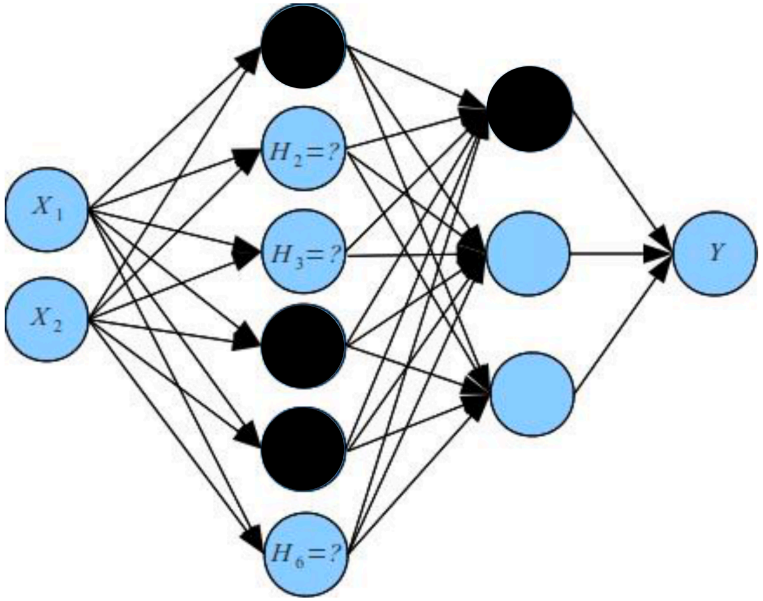


- During training, at each iteration, a unit is dropped with probability *p*.
- Each unit is dropped **independently**

Srivastava et al., 2014

Dropout

Iteration 1



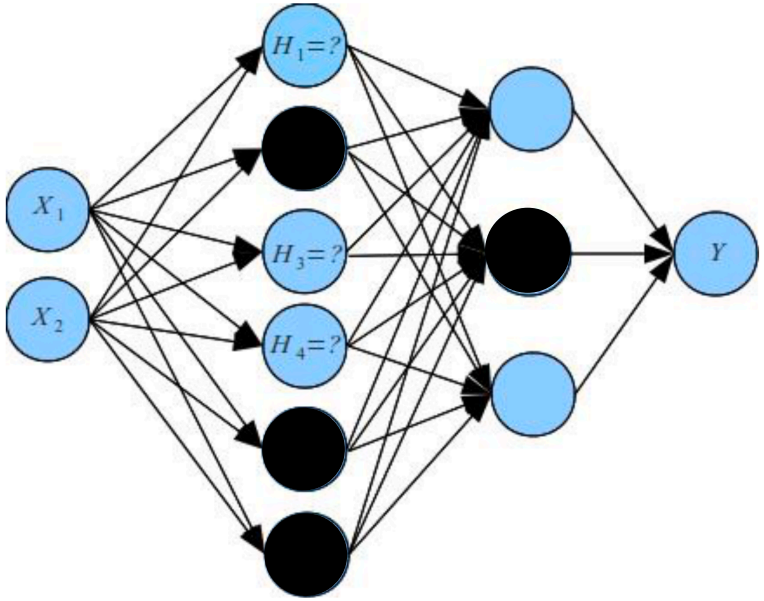
● = unit not used to compute output

- During training, at each iteration, a unit is dropped with probability p .
- Each unit is dropped independently

Srivastava et al., 2014

Dropout

Iteration 2

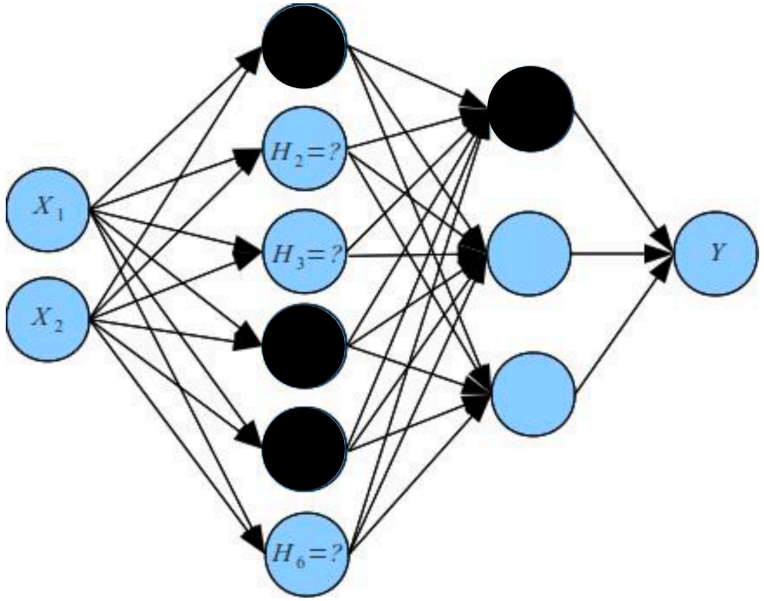


= unit not used to compute output

- During training, at each iteration, a unit is dropped with probability p .
- Each unit is dropped independently

Srivastava et al., 2014

Dropout



- Force **redundancy**
- Acts like an activation mask *m* over units:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathbf{m}^{(k)}$$

Where: h = hidden units

m = drop out mask

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

Srivastava et al., 2014

Slide taken from Hugo Larochelle

Dropout

- Dropout can be seen as an cheap approximation of model assembling
- **DON’T USE DROPOUT AT TEST TIME**

Srivastava et al., 2014

Summary of simple neural networks

- Things to try to improve model performance:
  - Different nonlinearities (sigmoid, tanh, ReLU)
  - Try different number of hidden layers with different size
  - Regularizations (drop out, weight decay, early stopping)
  - Different optimization schemes

Plan of this lecture

- Supervised neural networks
- **Optimization for neural networks**
- Convolutional network
- Recurrent network
- Unsupervised learning

Optimization

- Neural networks needs many examples to perform well (>millions)
- They are slow to train (many parameters to learn)

→ need to be careful about optimization

Gradient descent (GD)

$$L(\theta) = \frac{1}{N} \sum_i \ell(\theta, X^{(i)}, Y^{(i)})$$

$$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$

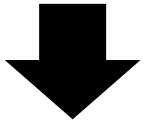
- Take a step in the direction of the gradient to minimize the loss
- Need to go over all the data to compute a single gradient!
- Impractical with millions of examples

Gradient descent: how to make it faster?

- Use less data → no.
- Use an approximation of GD → yes
- Idea: Are all the data needed to have a good gradient?
- Idea++: Why wait to process some data before updating the model?

Stochastic gradient descent (SGD)

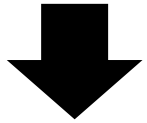
GD: $$\theta_{t+1} \rightarrow \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$



SGD: $$\theta_{t+1} \rightarrow \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$$

Stochastic gradient descent (SGD)

GD: $$\theta_{t+1} \to \theta_t - \frac{\alpha_t}{N} \sum_i \frac{\partial \ell(\theta, X^{(i)}, Y^{(i)})}{\partial \theta}$$



Replacing the average by picking a random data point

SGD: $$\theta_{t+1} \to \theta_t - \alpha_t \frac{\partial \ell(\theta, X^{(i_t)}, Y^{(i_t)})}{\partial \theta}$$

Stochastic gradient descent (SGD)

- In expectancy, SGD's gradient == GD gradient

$$\mathbb{E}_{\hat{P}}(\nabla_\theta f_i(\theta)) = \frac{1}{N}\sum_j \nabla_\theta f_j(\theta)$$

- **Pros**:
  - For 1 update in GD, N updates are made with SGD
  - Can work on infinite data!
- **Con**:
  - Introduce some variance in the gradient

SGD with mini-batch

- Pick K random points instead of picking 1 (with K << N):

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \frac{1}{K} \sum_k \frac{\delta \ell(\theta, X_{i_k}, Y_{i_k})}{\delta \theta}$$

- Reduce the variance but still fast

- Work well with GPUs

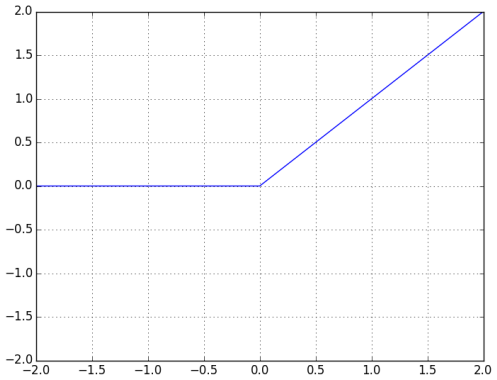
Practical questions

- How to initialize the parameters?
- How to choose the learning rate α_t ?
- Can we do (a bit) better than plain gradients?

Initialization

- If two units are equal, they stay equals
- Waste of capacity
- Random initialization breaks the symmetry

Initialization



- Many nonlinearities have regions with gradient with 0 norm
- Initialization should avoid this ”saturation” of the hidden units (else they don ‘t move)
- … or use nonlinearities with no saturation:

$$\text{Leaky ReLU}(x) = \text{ReLU}(x) + ax$$ With $a > 0$

Initialization

- **Fan-in**: number of inputs used to compute a hidden units
- Rule of thumb to avoid saturation: **initialize weights as 1 / sqrt(fan-in)**
- Why?

  More inputs → higher chance of saturation

- …however, **mostly true for sigmoid and tanh**.

Reduce gradient variance

- **Whitening:** remove average and reduce variance to 1

$$x \leftarrow (x - \mu)/\sigma$$

- Whitening inputs to avoid gradient with large weights

- Same is for input of each layer.

- How to proceed since they change during the optimization?

Batch normalization

- The mean μ_i and variance σ_i of the input to layer *i* are
  - Updated over time
  - Keep for the next weight update

- These parameters are update according to the statistic of the current batch → batch normalization

Batch normalization

$$o_i = BN_{\alpha,\beta}(h_i)$$

$$\mu_B \leftarrow \frac{1}{b}\sum_{i=1}^{b} h_i$$

$$\sigma_B^2 \leftarrow \frac{1}{b}\sum_{i=1}^{b}(h_i - \mu_B)^2$$

Compute batch statistics

$$\hat{h}_i \leftarrow \frac{h_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

Normalize hidden state h_i

$$o_i \leftarrow \alpha\hat{h}_i + \beta$$

Shift the normalized hidden

Ioffe & Christian Szegedy, 2015

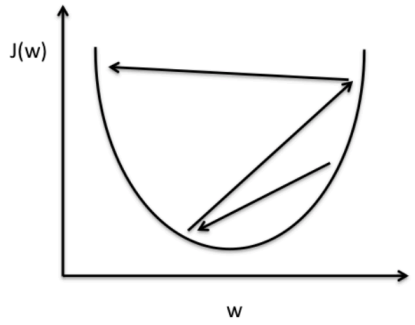
Batch normalization

- At test time, batch statistics are replaced by statistics on the whole population:

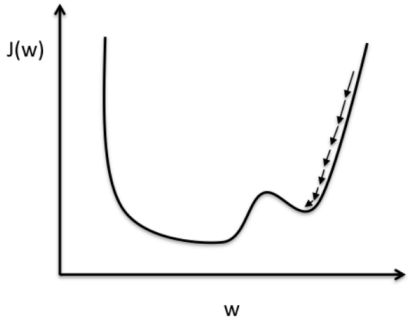
$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Ioffe & Christian Szegedy, 2015

Setting the learning rate (aka step size)



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

- Fix learning rate works surprisingly well
- Other strategies are:
  - linear decay: $\alpha_t = a/(b+t)$
  - Divide learning rate by 2 every time the loss on validation doesn't decrease
  - Fix nb of iterations K and make learning rate = 0 at the last one: $\alpha_t = \alpha_0(K-t)/K$

Setting the initial learning rate

- For logistic regression, a good initialization is in the range:

  [0.01, 0.05, 0.1, 0.2, 0.5, 1]

- Often people do [10^{-8}, 10^{-7}, …, 1, …, 10^{5}]… this rarely works

- Rule of thumb:

  pick the learning rate that is ”just below” the one where the network diverge.

Going beyond SGD: ADAGRAD

$$G_{t,i} = \sum_{j=1}^{t} g_{t,i}^2$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \varepsilon}} g_{t,i}$$

- No need to set a learning rate schedule!
- Each variable is updated at its own rhythm!
- The gradients are normalized by their squared norm → avoid exploding or vanishing gradient
- Simpler popular (and almost as good) method: ADAM

Duchi et al. 2011

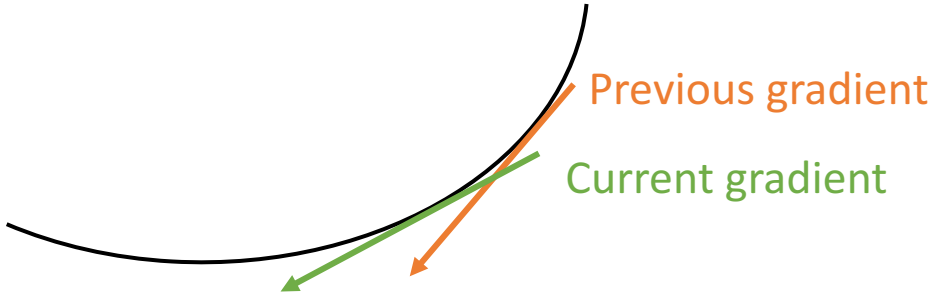
ADAGRAD

SGD **ADAGRAD**

- ADAGRAD (or ADAM) normalizes the gradient across axis

Duchi et al. 2011

Momentum



- Throwing away previous gradient is wasteful
- If a function is “smooth enough”, previous gradients are informative
- Especially with SGD where the variance is high

Momentum

$$\boldsymbol{v}_t = \gamma \boldsymbol{v}_{t-1} + \eta \frac{\partial \ell(\theta, \boldsymbol{X}^{(i_t)}, \boldsymbol{Y}^{(i_t)})}{\partial \theta}$$

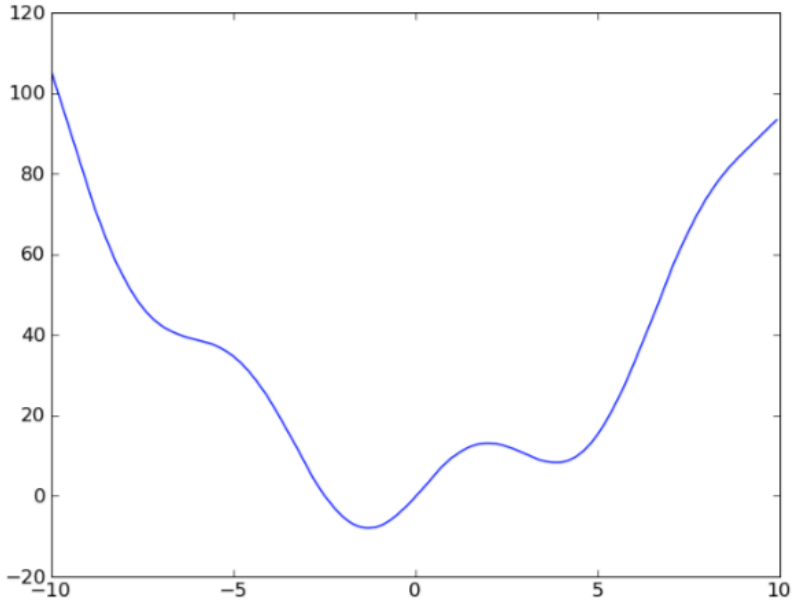
$$\theta_{t+1} = \theta_t - \boldsymbol{v}_t$$

- Moving average the gradient over time with a decay
- Smooth gradient with little overhead
- Typically eta > 0.5 and gamma = 1 - eta
- Going even further: Nesterov acceleration, SAG…

Summary of optimization

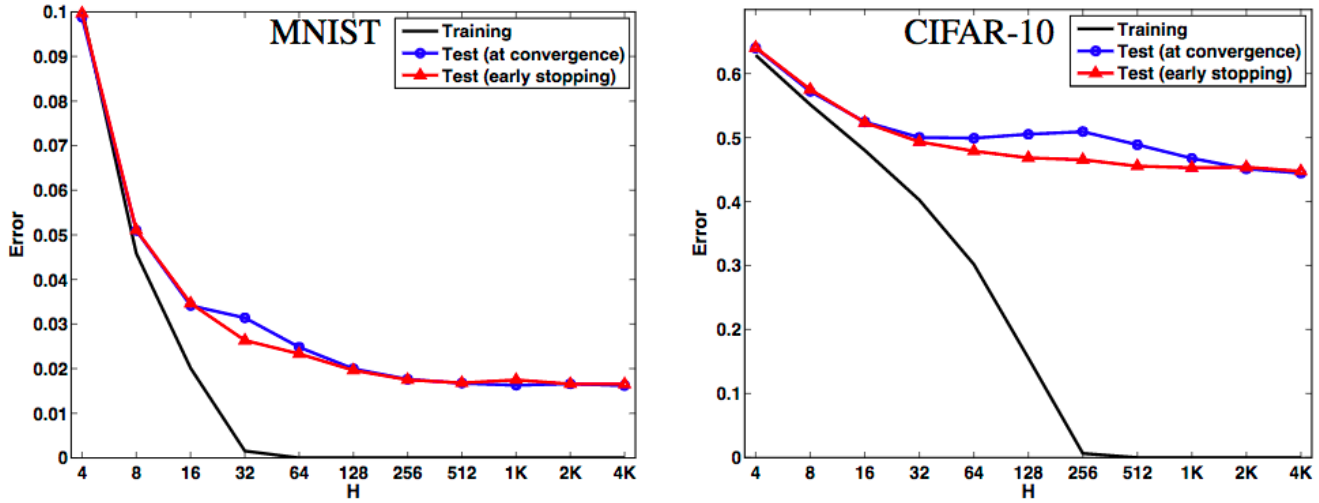
- Be careful about your initialization (avoid saturated non-linearities)
- With lot of data: use SGD over GD
- On GPUs, use mini-batch
- Batch-norm works for some type of models (e.g., CNNs but not RNNs)
- ADAGRAD and momentum rarely work together

Optimization and neural network



- Neural networks are **not convex**
- **Open question**: how comes we are not stuck in poor local minima?
- **Current belief**: most local minima are close to the global one

Optimization and neural network



- Few strange observations:
  - Optimization never gets stuck on saddle points
  - Even if the training error has converged, test error continues to decrease!

Figure form Neyshabur et al. 2017

Beyond simple neural network

- Fully connected feed-forward networks makes no assumption on the data
- They are slow and poorly designed for some applications
- Instead, let us consider special cases of neural networks:
  - Convolutional Neural Networks for images
  - Recurrent Neural Networks for text

Plan of this lecture

- Supervised neural networks
- Optimization for neural networks
- **Convolutional network**
- Recurrent network
- Unsupervised learning

Convolutional network (convnet)

- Light introduction in this lecture
- Wait for Ivan Laptev’s lecture on the subject for more details

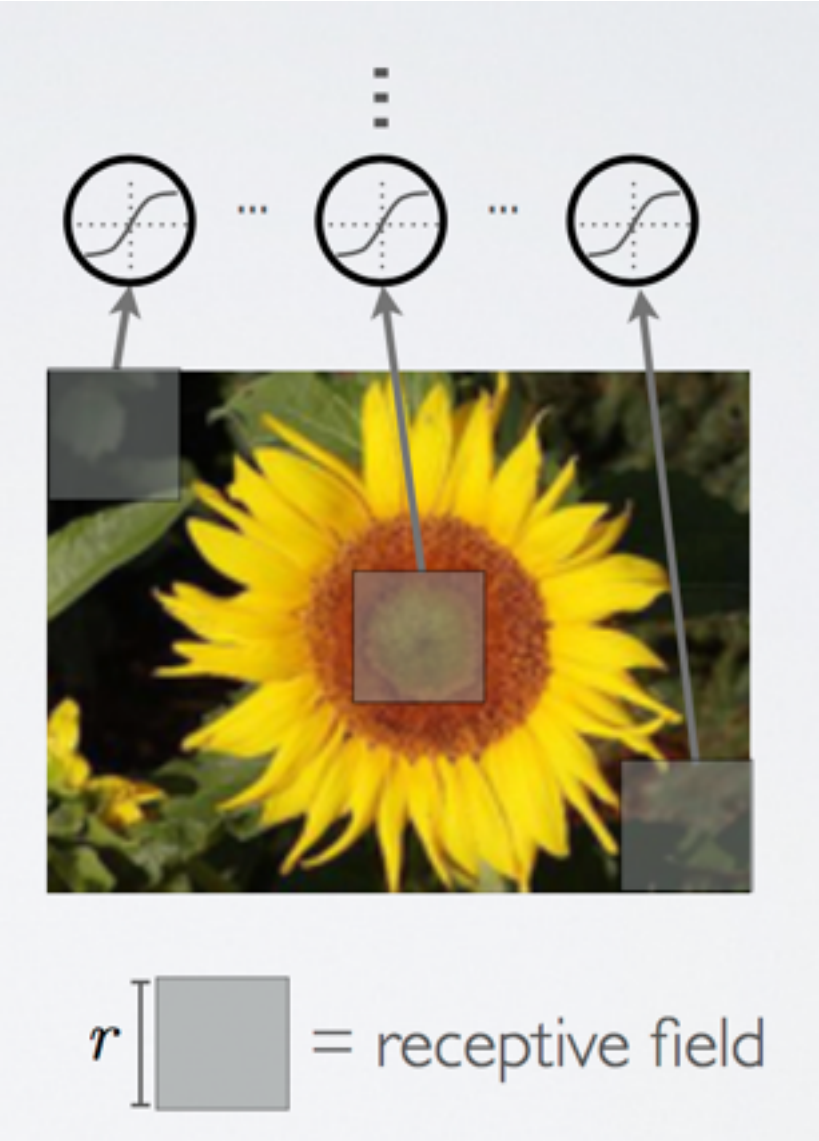
Convnets

- Architecture adapted to images:
  - Translation invariant
  - Multiscale
  - 2D topology of images

- How:
  - Local connectivity
  - Parameter sharing
  - Pooling/rescaling in a deep network

Convnets

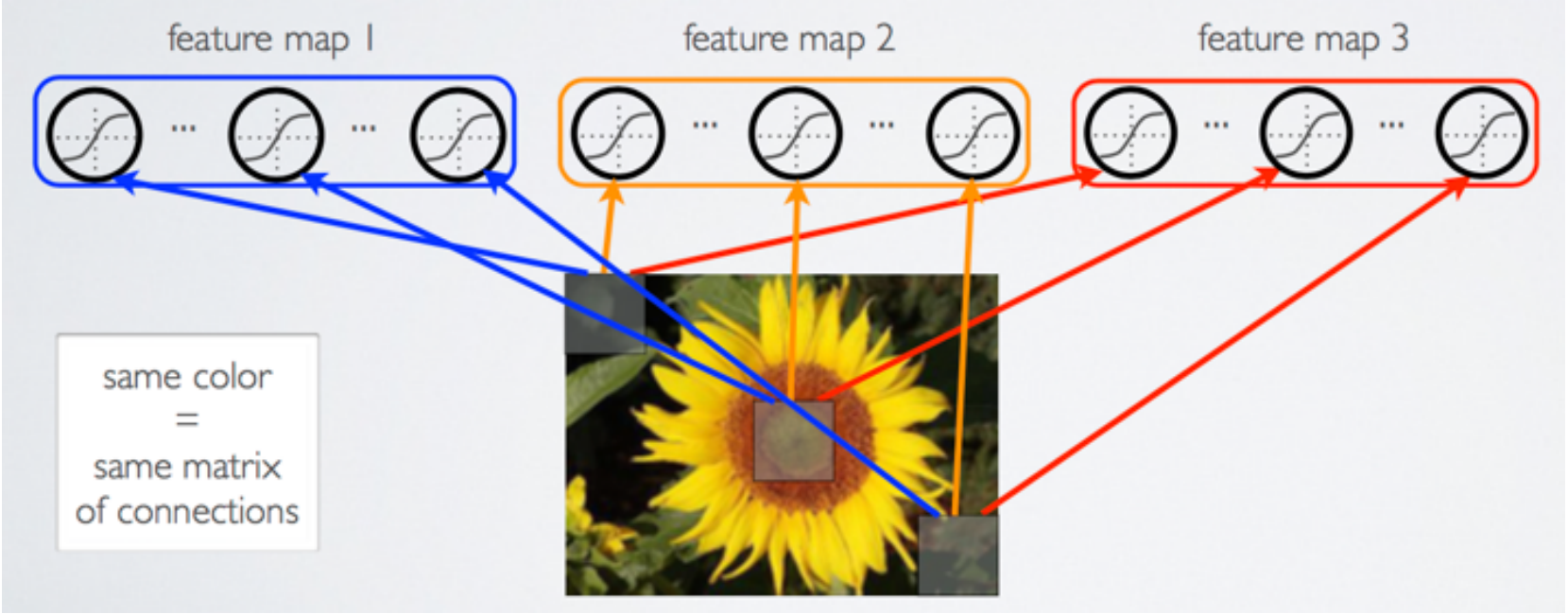
- **Local connectivity**
  - hidden units are computed from local information
  - Less parameters
  - Similar to wavelet on the 1st layer



Slide taken from Hugo Larochelle : http://www.dmi.usherb.ca/~larocheh/cours/ift725_A2014/contenu.html

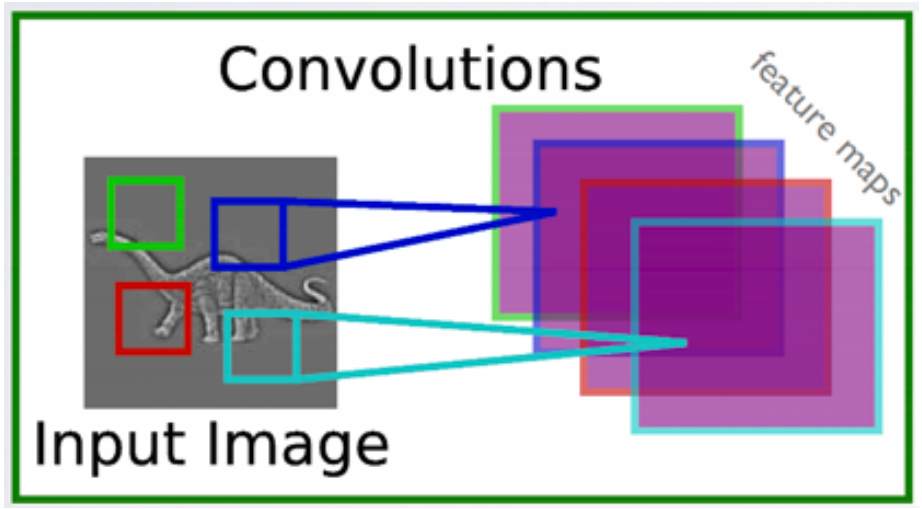
Convnets

- **Parameters sharing**
  - Compute feature maps
  - Less parameters
  - Translation invariance



Slide taken from Hugo Larochelle : http://www.dmi.usherb.ca/~larocheh/cours/ift725_A2014/contenu.html

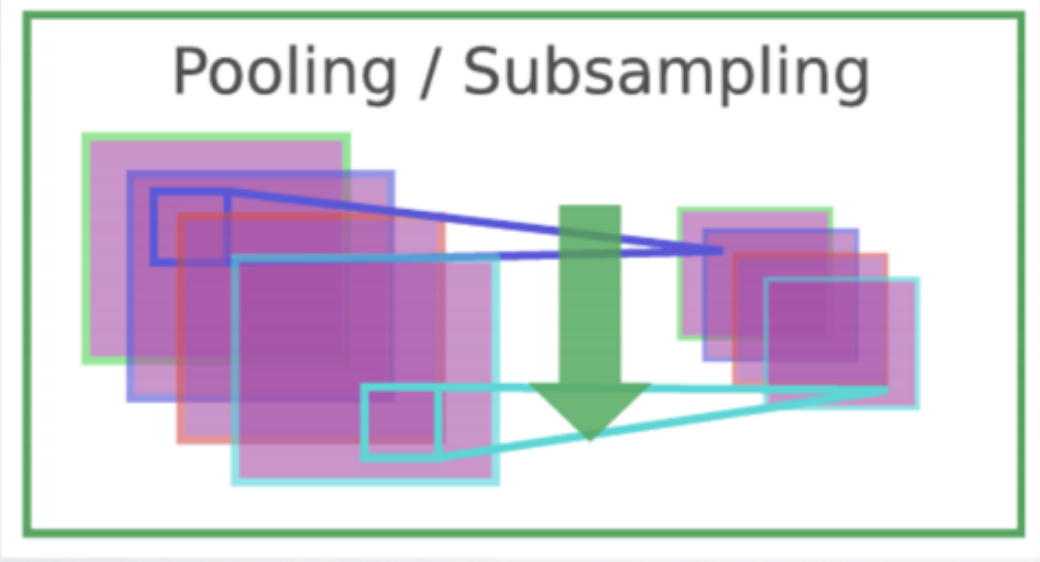
Convnets



- **Parameters sharing + local connectivity == convolution**
- Repeat the process in depth → multiscale features

Slide taken from Hugo Larochelle : http://www.dmi.usherb.ca/~larocheh/cours/ift725_A2014/contenu.html

Convnets



- **Pooling/rescaling:**
  - Reduce number of features in deep network
  - Summarized local information

Slide taken from Hugo Larochelle : http://www.dmi.usherb.ca/~larocheh/cours/ift725_A2014/contenu.html

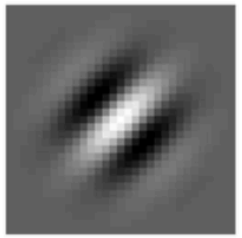
About pooling

- Different type of pooling:
  - Max pooling: take the max locally
  - Average pooling: take the average of the local activations
- Instead of pooling: use a convolution with a larger stride

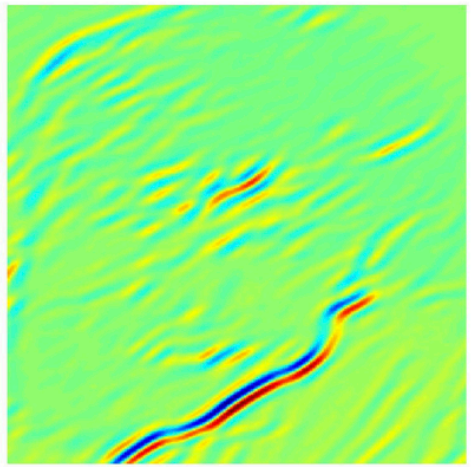
Example of convNet

Original image

Filter



Output image



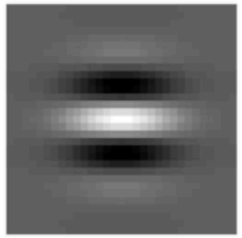
1st filter of 1st layer

From N. Le Roux's 2016 lecture

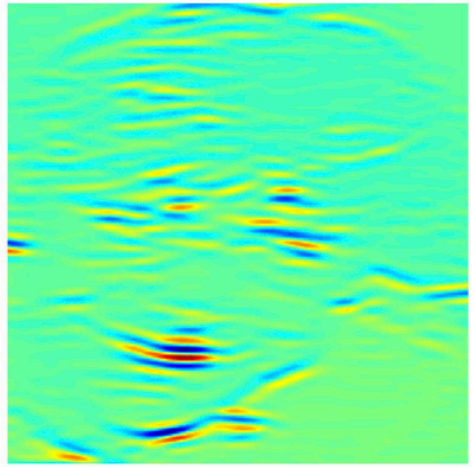
Example of convNet

Original image

Filter



Output image



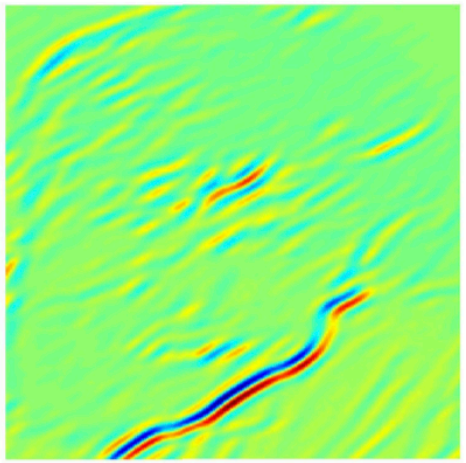
2^{nd} filter of 1^{st} layer

From N. Le Roux's 2016 lecture

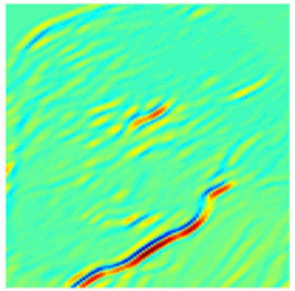
Example of convNet

Original image

Output image



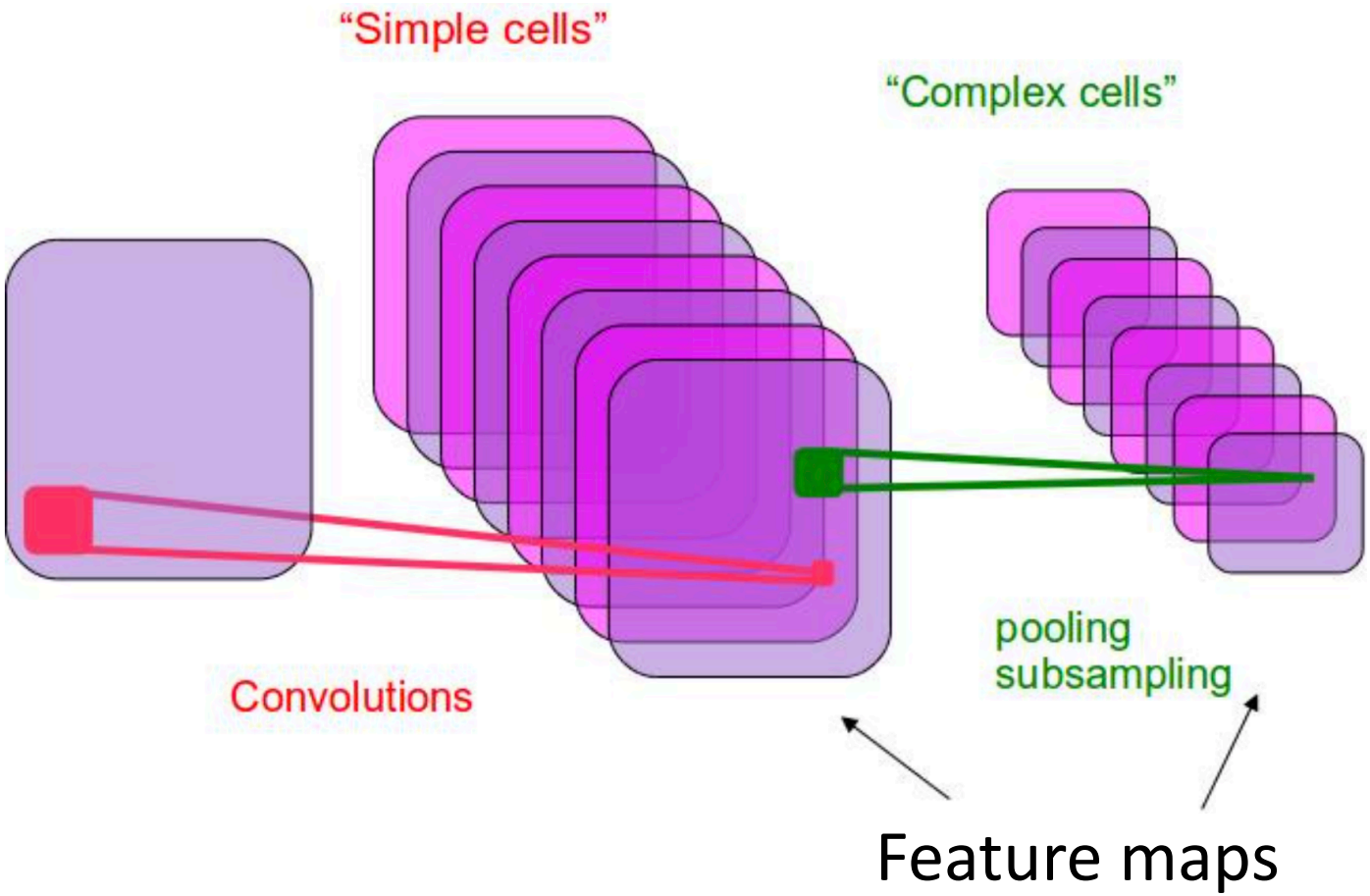
Subsampled image



Subsampling/pooling of the response map of the 1st feature

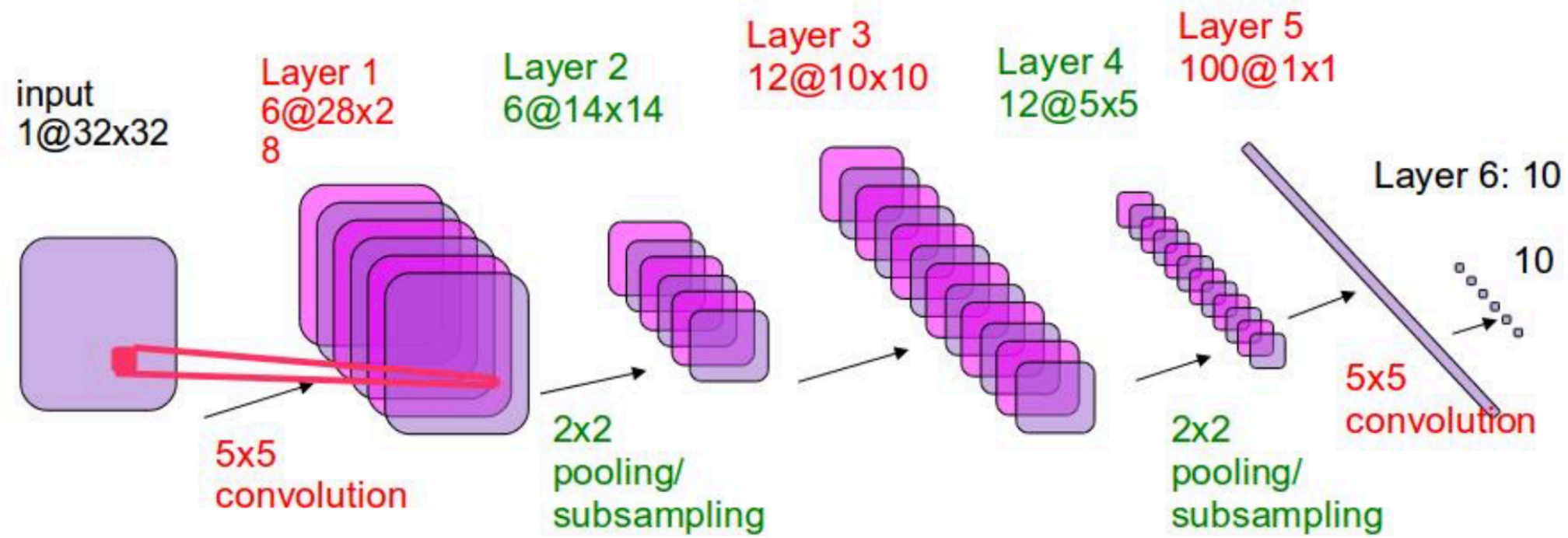
From N. Le Roux's 2016 lecture

ConvNets



From N. Le Roux's 2016 lecture

ConvNets



- Repeat the process
- Train end-to-end

Take away from convNets

- Filters are mostly local.
- Instead of using image-wide filters, use small ones over patches.
- Repeat for every patch to get a response image.
- Subsample the response image to get local invariance.

From N. Le Roux's 2016 lecture

Additional information on convNets

- GPUs are perfect for convNets.

- Several architectures exists:
  - Alexnet (2012)
  - VGG (2013)
  - ResNet (2014)

- Optimization:

mini-batch + batch-norm + momentum + learning rate set with validation set

Plan of this lecture

- Supervised neural networks
- Optimization for neural networks
- Convolutional network
- **Recurrent network**
- Unsupervised learning

Recurrent network

- What happens when your data are **sequences**?
  - Example: language modelling, weather forecast, videos…
  - Take advantage of the structure of the data to learn better model?

Sequence modeling

- At time t, we want the probability of output $y(t)$ given input $x(t)$
- The output $y(t)$ depends on the past inputs, i.e. $x(1),\ldots, x(t)$
- In other words we are interested in modeling the probability:

$$P(y(t) \mid x(1), \ldots , x(t))$$

Examples

- Language modelling:
  - Predicting the distribution of a sentence *S*

$$P(S) = P(w_1, \ldots, w_{|S|})$$

  - Since a sentence is a sequence, this means:

$$P(S) = \prod_t P(w_t \mid w_{t-1}, \ldots, w_1)$$

- Action classification in a video:
  - Predicting if action is happening based on the past frames:

$$P(a_t \mid f_{t-1}, \ldots, f_1)$$

Example: Language modelling

- One-hot encoding (or 1-of-K encoding):
  - Vocabulary = {“cat”, “in”, “is”, “room”, “the”, “.”}
  - “cat” = [1 0 0 0 0 0]
  - “in” = [0 1 0 0 0 0]
  - “is” = [0 0 1 0 0 0]
  - “room” = [0 0 0 1 0 0]
  - “the” = [0 0 0 0 1 0]
  - “.” = [0 0 0 0 0 1]

Example: Language modelling

- N-grams models are very popular to model a sequence:
  - Bigram:

$$P(\text{"}the\ cat\ is\ in\ the\ room.\text{"}) = P(\text{"}the\text{"})P(\text{"}cat\text{"} \mid \text{"}the\text{"})P(\text{"}is\text{"} \mid \text{"}cat\text{"})P(\text{"}in\text{"} \mid \text{"}is\text{"}) \times P(\text{"}the\text{"} \mid \text{"}in\text{"})P(\text{"}room\text{"} \mid \text{"}the\text{"})P(\text{"}.\text{"} \mid \text{"}room\text{"})$$

  - Trigram:

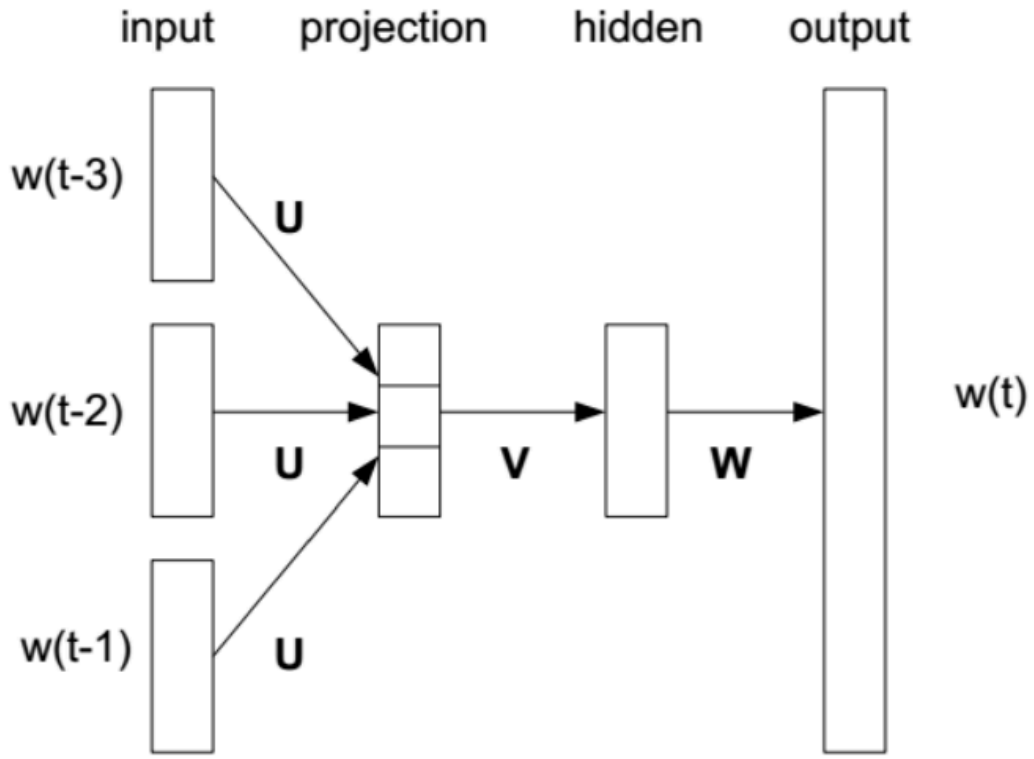
$$P(\text{"}the\ cat\ is\ in\ the\ room.\text{"}) = P(\text{"}the\text{"})P(\text{"}cat\text{"} \mid \text{"}the\text{"})P(\text{"}is\text{"} \mid \text{"}cat\text{"}, \text{"}the\text{"})P(\text{"}in\text{"} \mid \text{"}is\text{"}, \text{"}cat\text{"}) \times P(\text{"}the\text{"} \mid \text{"}in\text{"}, \text{"}is\text{"})P(\text{"}room\text{"} \mid \text{"}the\text{"}, \text{"}in\text{"})P(\text{"}.\text{"} \mid \text{"}room\text{"}, \text{"}the\text{"})$$

  - …

Example: Language modelling

- N-grams can be modeled by feed-forward networks (Bengio et al. 2003):

- 4-grams:



Example: Language modeling

- Limitation of Feed-forward networks:
  - size of the n-gram?
  - Very inefficient at test time

- strong assumption:

$$P(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid w_{t-1},\ w_{t-2})$$

- Can the network learn to “remember” what is important?

Recurrent neural network (RNN)

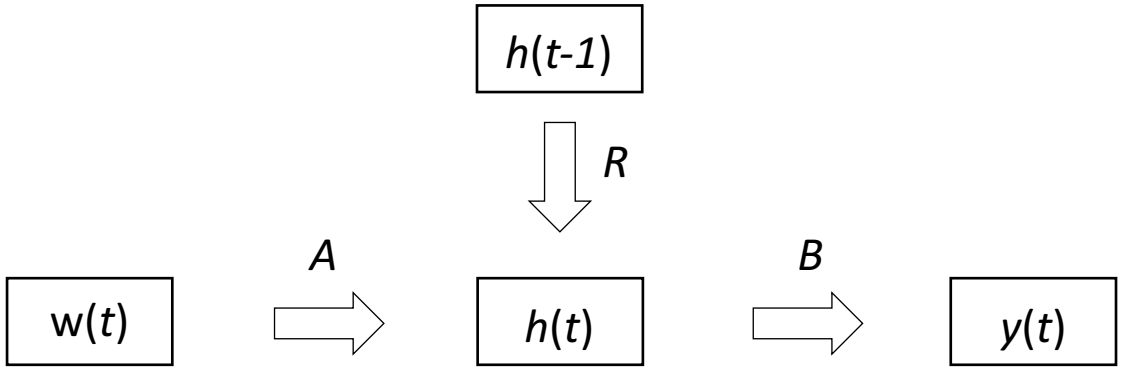
- **Main idea**: Keep a “memory” of the past in the hidden layers.
- previous state of the hidden layers influence the current one
- What does it mean?
  - In standard neural network, the hidden units *h* depends on input:

$$h(t) = f(w(t))$$

  - In RNN:

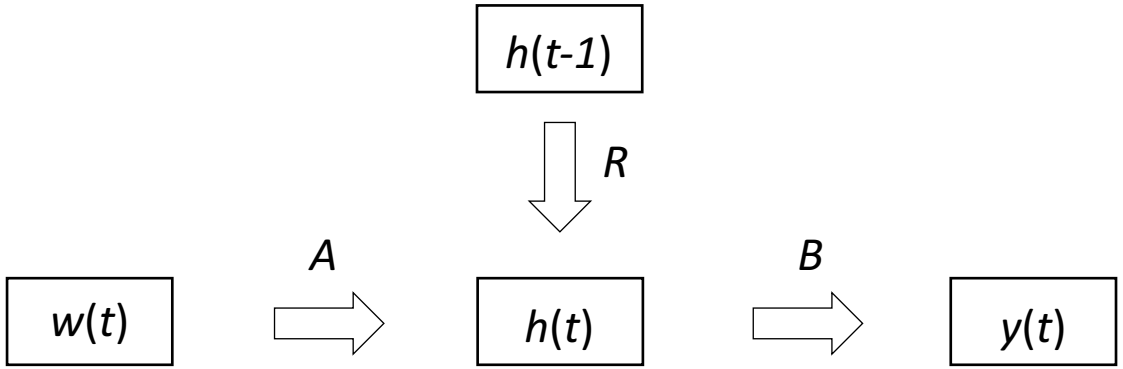
$$h(t) = f(w(t), h(t-1))$$

Simple Recurrent Network (from Elman, 1990)



- The simple RNN contains 1 hidden layer with a “Recurrent” connection
- Neural network equivalent of Hidden Markov Chain

Simple Recurrent Network



- The simple RNN:

$$h_t = \sigma(Aw_t + Rh_{t-1})$$

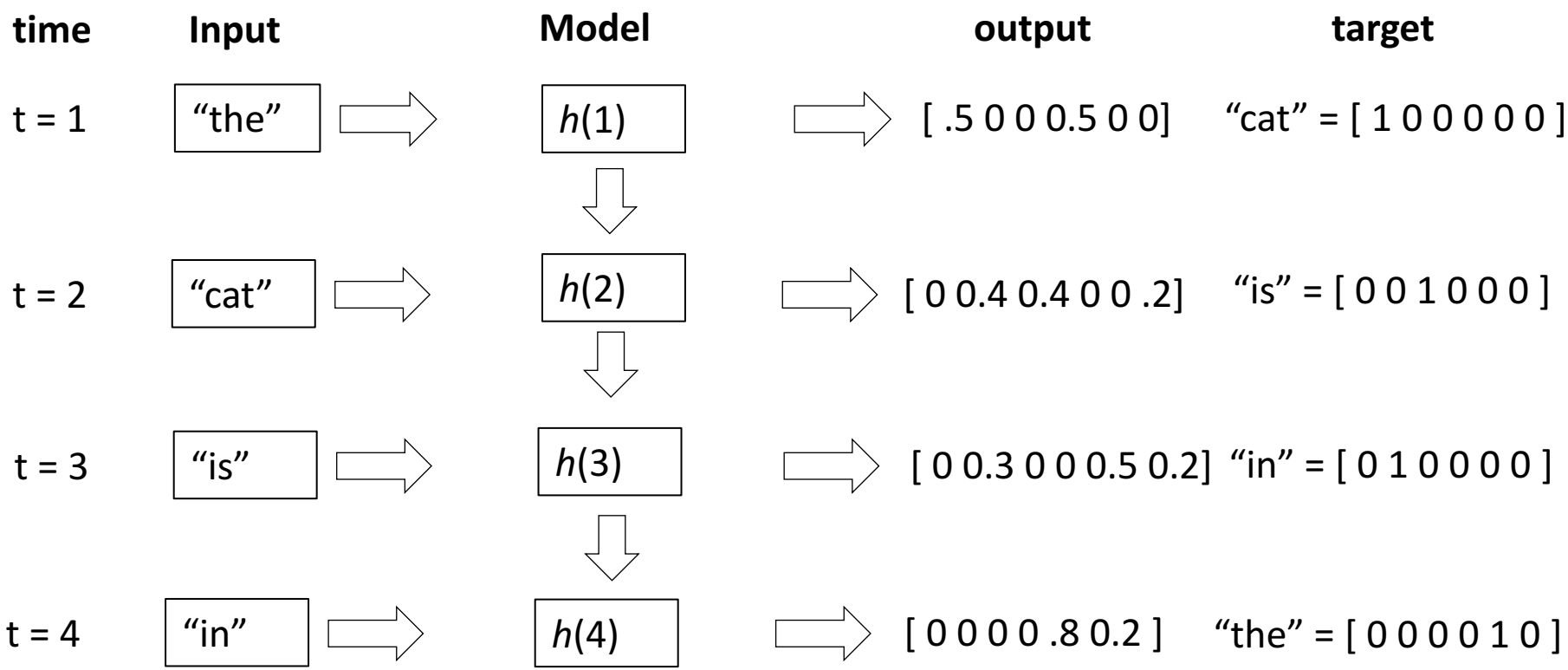
$$y_t = f(Bh_t)$$

where:

- *A, B* and *R* are matrices,
- Sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$
- Softmax $f(z)_k = \exp(z_k)/(\sum_i \exp(z_i))$

Back to the example

Vocabulary = {“cat”, “in”, “is”, “room”, “the”, ”.”}

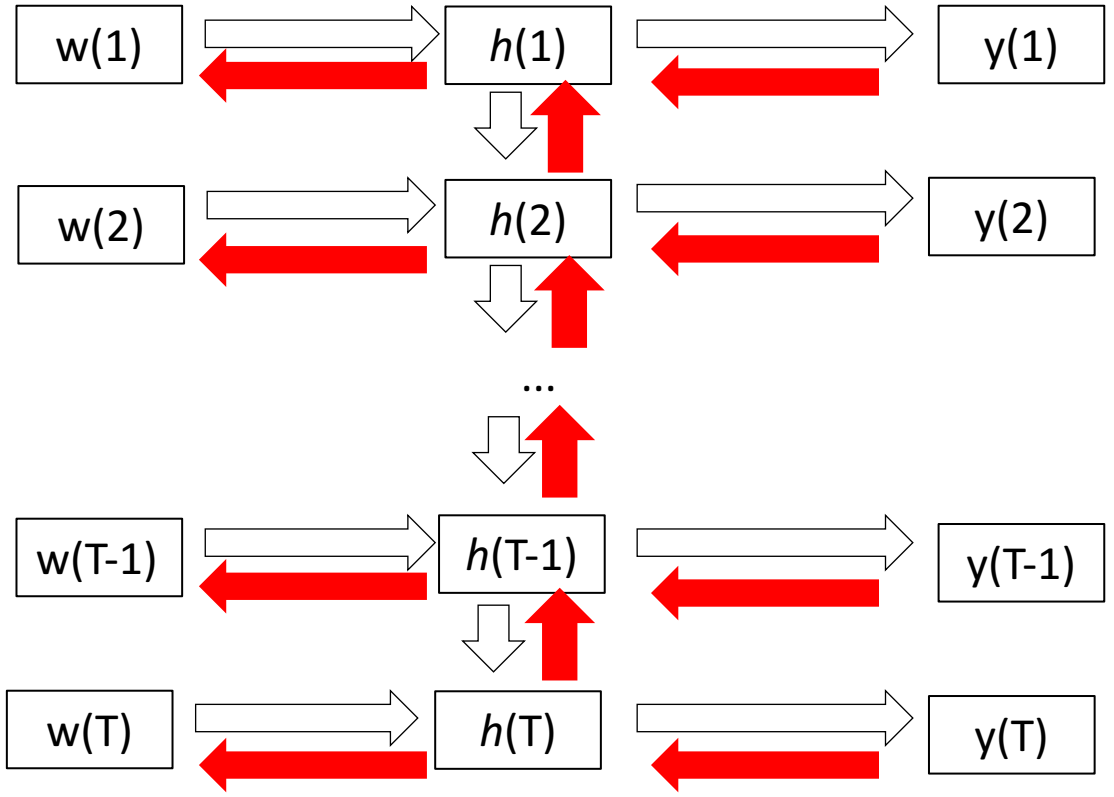


How to train a RNN?

- **Problem:** output at t = T depends on all the hidden units since t = 1
- backpropagation **until the beginning of the sequence!**
- This is called **backpropagation through time** (BPTT)
- RNN == very deep neural networks with weight sharing

Backpropagation through time

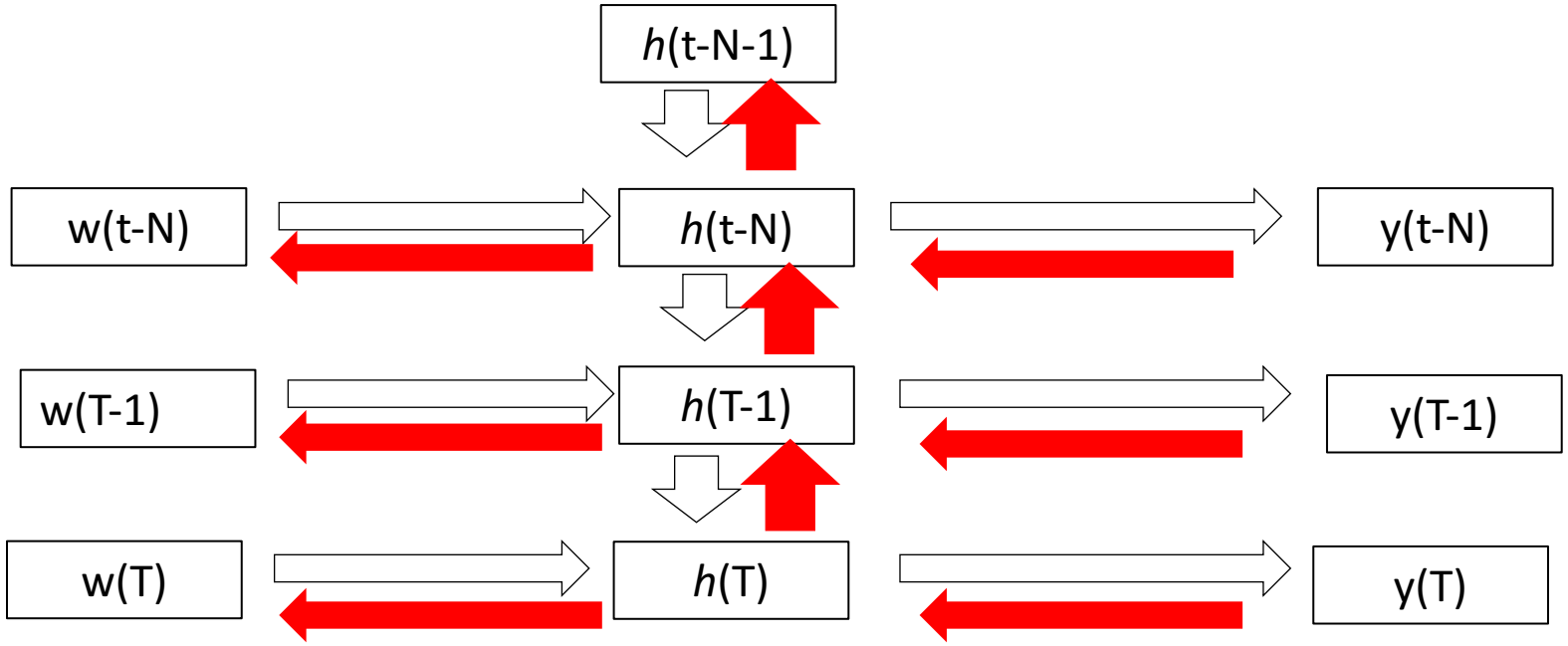
- Unrolling the model since the beginning
- This means that computing a gradient is O(T)
- backward propagation of error in red:



Backpropagation through time

- Trick to make it practical :
  - unfold the network for N fixed step -> O(N)
  - compute the gradient every N steps

  → gradient can be computed efficiently **online**



RNN in practice

Model	Perplexity
Kneser-Ney 5-gram	141
Maxent 5-gram	142
Random forest	132
Feedforward NNLM	140
Recurrent NNLM	125

- Penn TreeBank (~1M words, vocabulary size = 10K)

- RNNLM (Mikolov toolbox, 2010) show a significant improvement compared to standard models

Backpropagation through time

- Depending on the eigenvalues of the recurrent matrix *R*, two possible issues (Bengio, 94)**:**
  - **max(eigenvalue) > 1 → Exploding gradient**
  - **max(eigenvalue) < 1 → Vanishing gradient**

Backpropagation through time

- **max(eigenvalue) > 1 → Exploding gradient**:

As you backpropagate through time, you multiply the gradient over and over by *R*, making the norm of the gradient go to infinity

→ Clipping or normalizing the gradient fix that problem (Mikolov et al. 2010)

Backpropagation through time

- **max(eigenvalue) < 1 + nonlinearity → Vanishing gradient:**

The magnitude of the gradient decreases when backpropagate through time:

→Hard to keep very long dependency

→Solutions have been proposed to counter that effect :

- *Exponential trace memory* (Jordan 1987,Mozer1989)
- *Long Short-term Memory* (Hochreiter & Schmidhuber, 1997)

More complex RNN

- More complex architectures to capture long term dependency :
  - Longer Short Term Memory (LSTM)
  - Gated Recurrent Network (Cho et al, 2014)
  - Linear Unit Network (Mikolov et al. 2014)

- All trained with BPTT!

Long Short Term Memory (LSTM)

- General idea:
  - Each hidden units possess a “**memory cell**” which can be preserve, modify or erase
  - The state of the hidden computed with the current state of the memory cell.
  - In practice this means adding ``**gating**” mechanisms:
    - Preserve or “forget” the value of the hidden layers

Gating mechanism

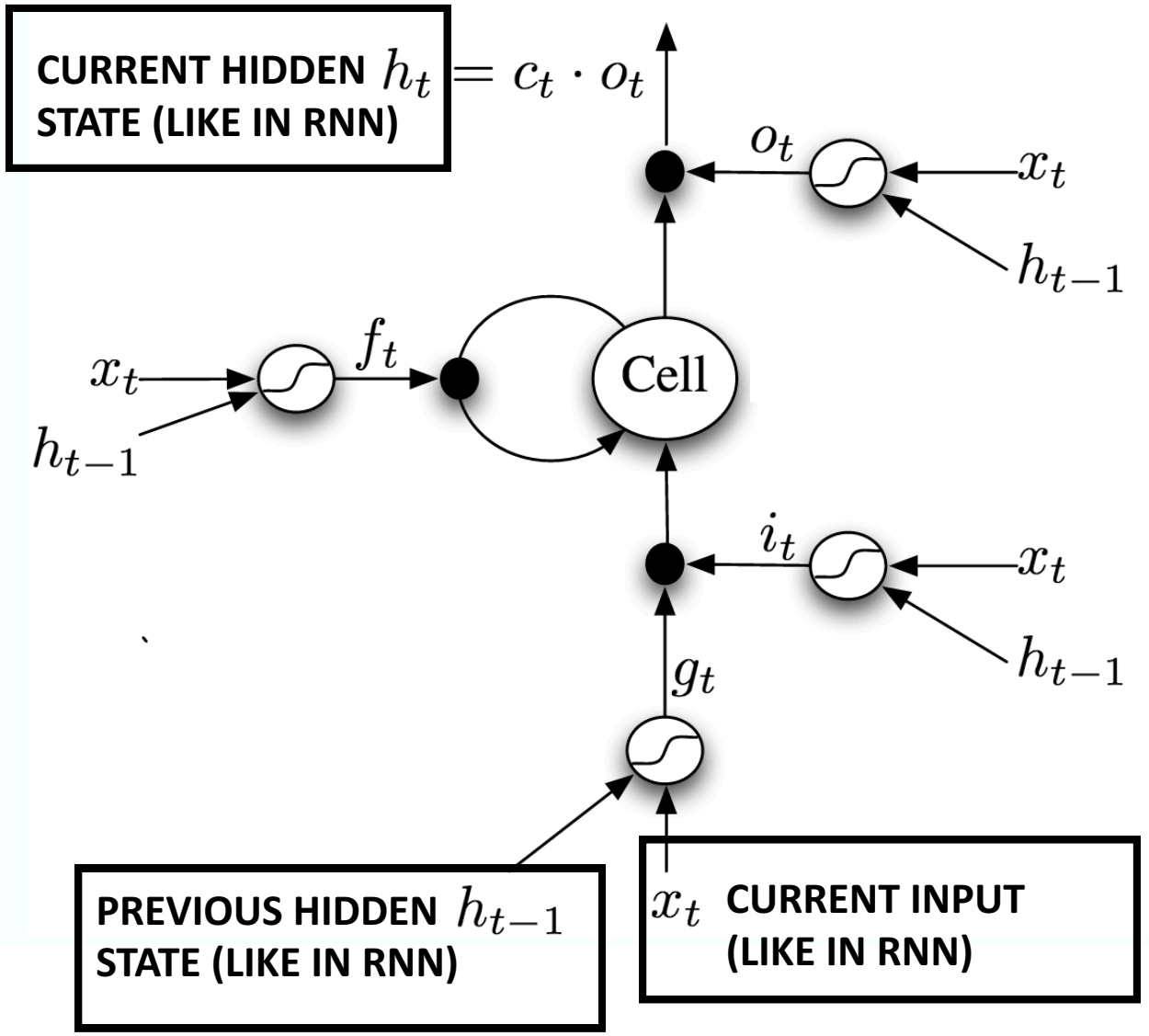
- A gating is a variable going from 0 to 1 which allows or block a signal.
- For example:

$$y = \sigma(x)a + (1 - \sigma(x))b$$

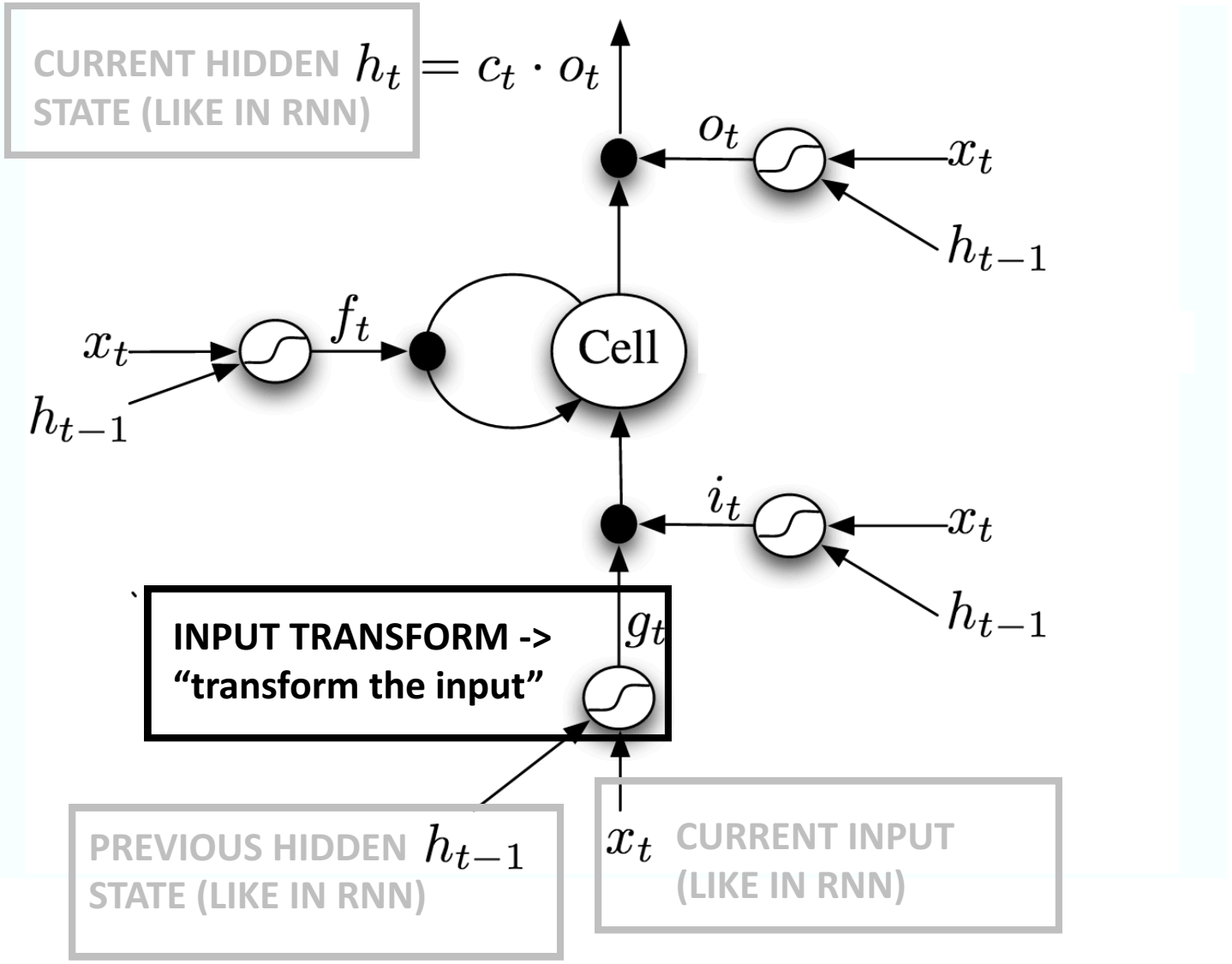
where σ is the sigmoid function (in [0,1]).

→$\sigma(x)$ is a **gate** controlled by *x,* which decides between value *a* or *b.*

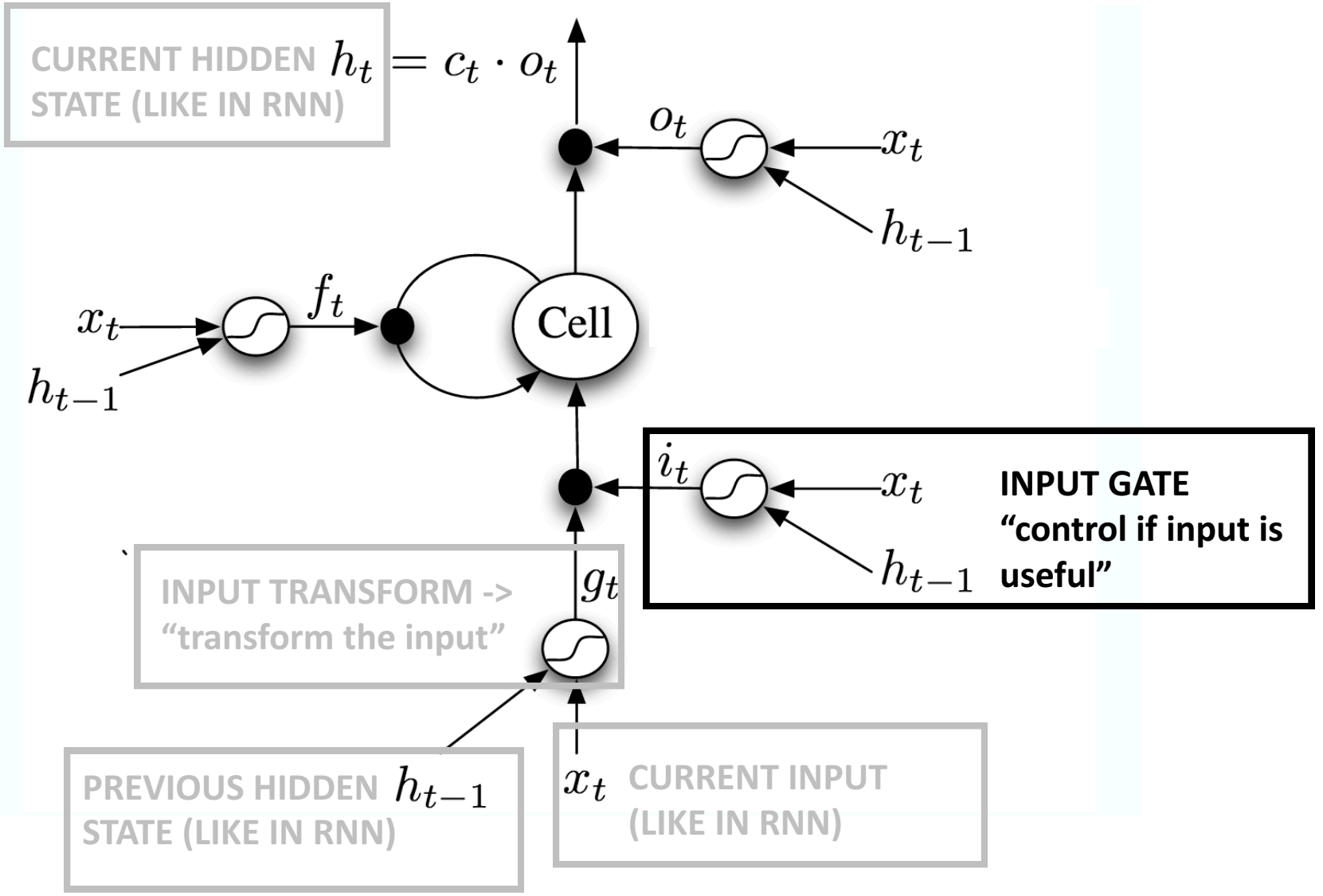
Long Short Term Memory (LSTM)



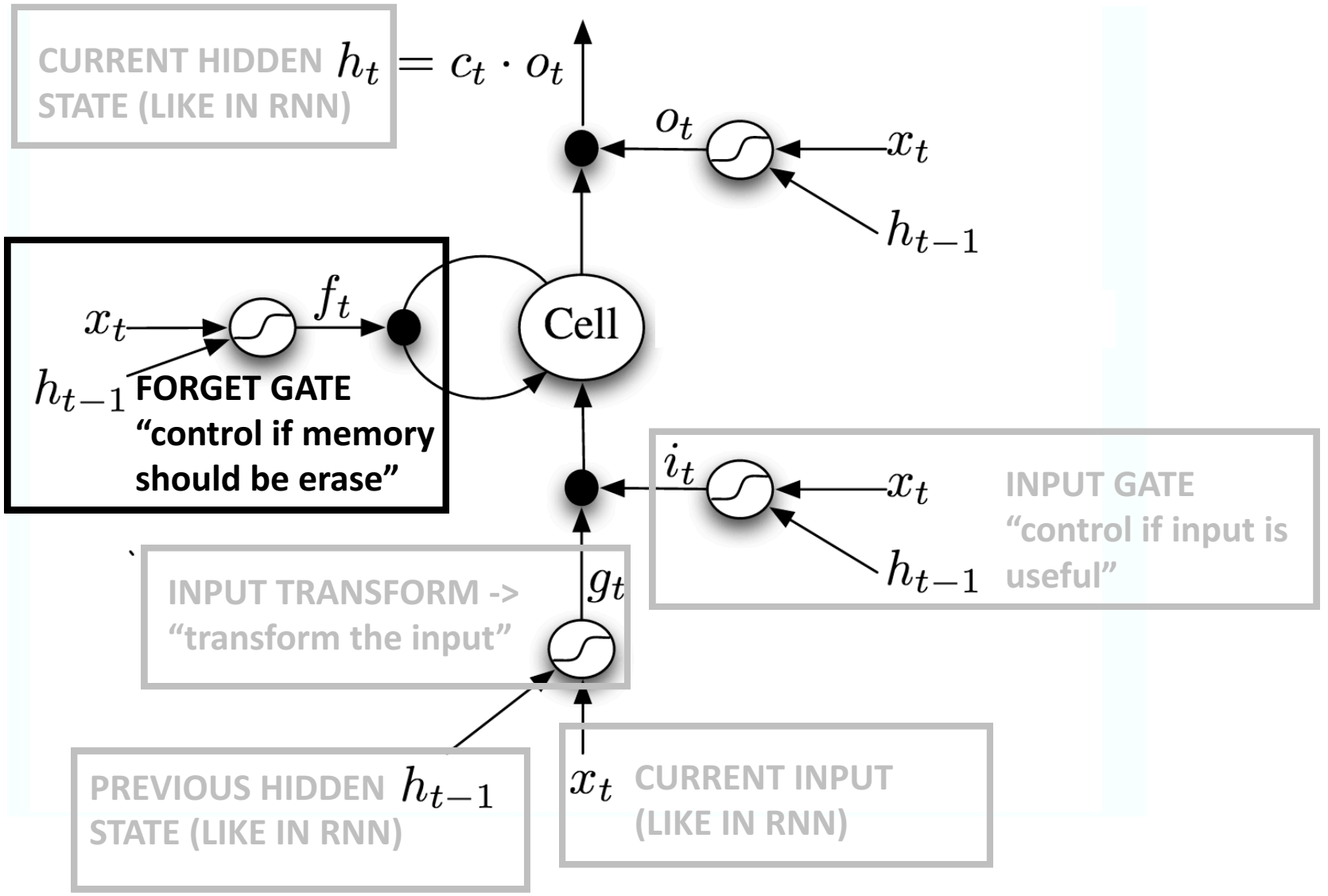
Long Short Term Memory (LSTM)



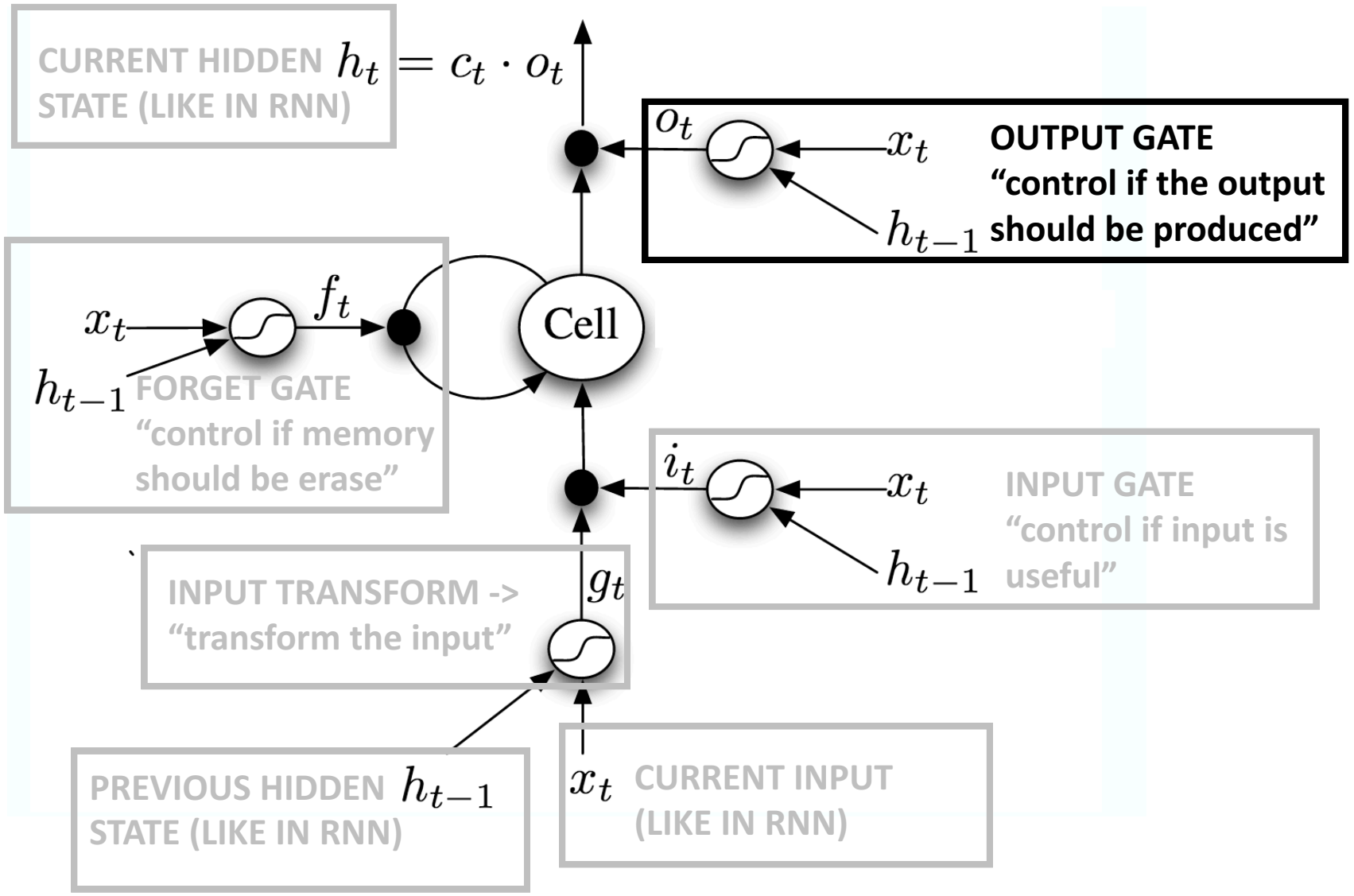
Long Short Term Memory (LSTM)



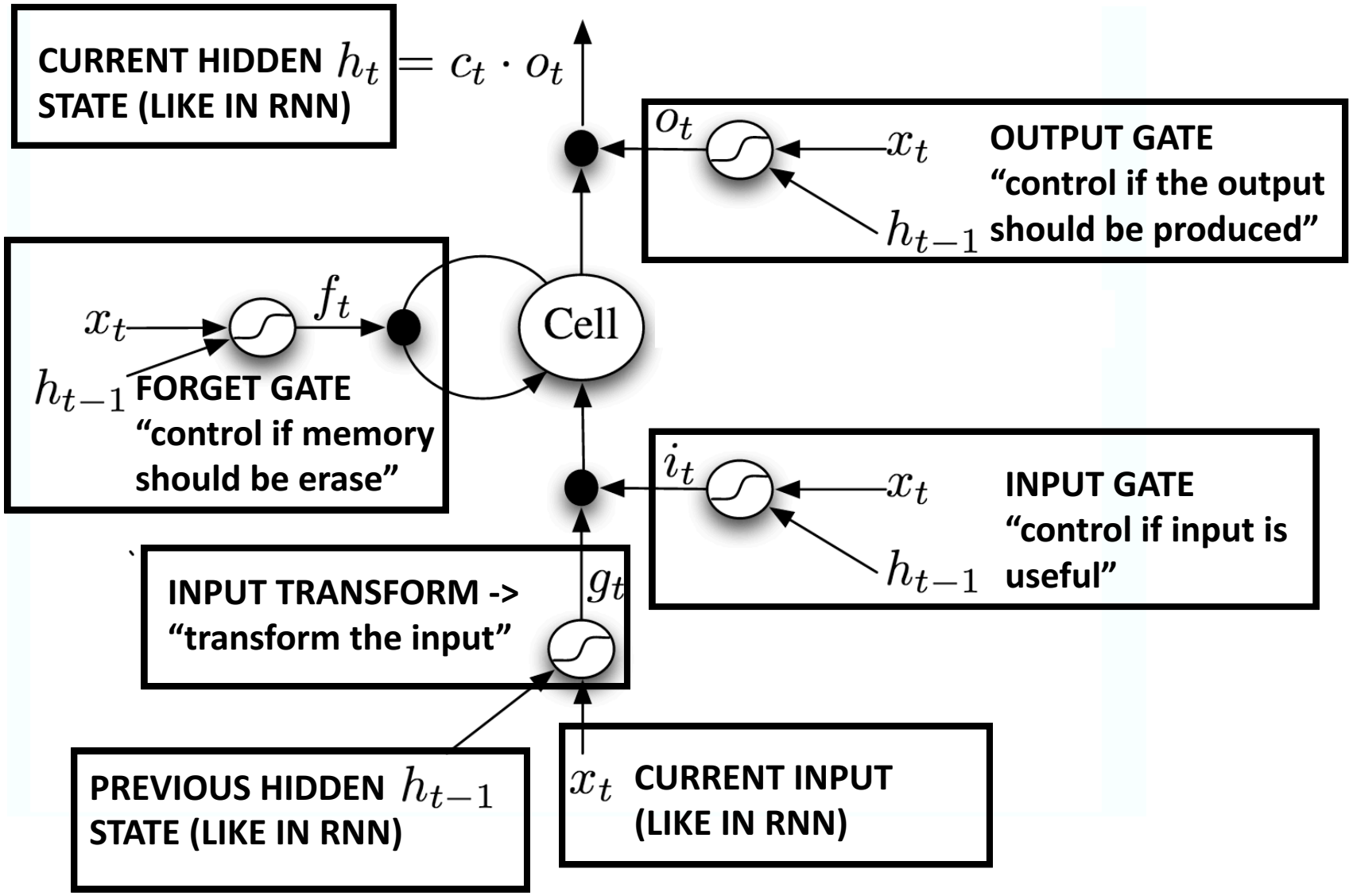
Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



Long Short Term Memory (LSTM)



LSTM for speech recognition

Table 1. TIMIT Phoneme Recognition Results. 'Epochs' is the number of passes through the training set before convergence. 'PER' is the phoneme error rate on the core test set.

NETWORK	WEIGHTS	EPOCHS	PER
CTC-3L-500H-TANH	3.7M	107	37.6%
CTC-1L-250H	0.8M	82	23.9%
CTC-1L-622H	3.8M	87	23.0%
CTC-2L-250H	2.3M	55	21.0%
CTC-3L-421H-UNI	3.8M	115	19.6%
CTC-3L-250H	3.8M	124	18.6%
CTC-5L-250H	6.8M	150	18.4%
TRANS-3L-250H	4.3M	112	18.3%
PRETRANS-3L-250H	**4.3M**	**144**	**17.7%**

Graves et. al., 2014: *Speech Recognition with Deep Recurrent Neural Networks*

Gated Recurrent Network (GRU)

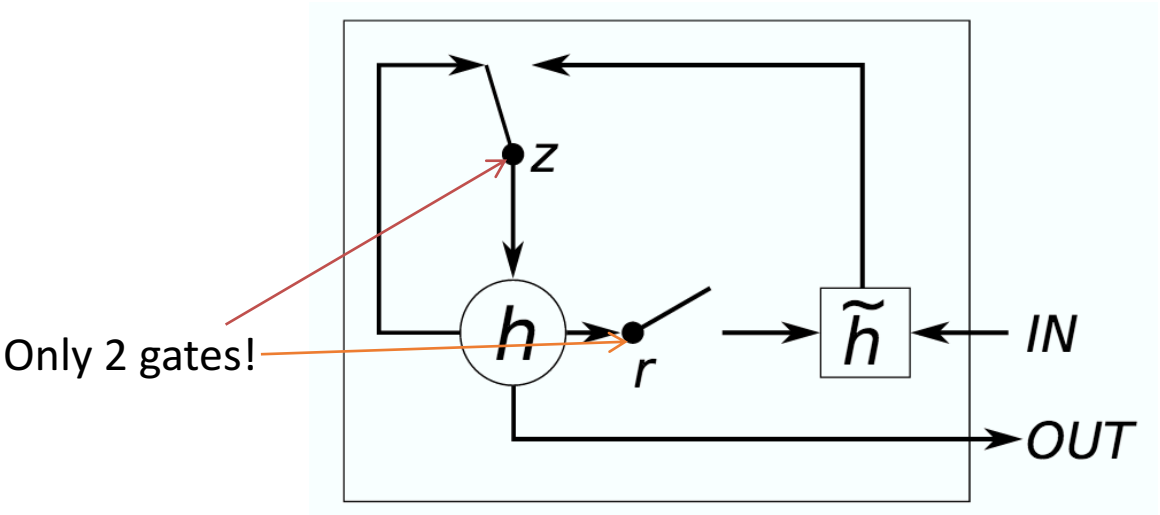
- LSTM is very popular and powerful model.

- However it seems unnecessarily complicated

→GRU is a simplified version of LSTM which performs as

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling; Chung et al., 2014

Gated Recurrent Network



2 gates:

- *r(t)* : control if the hidden should be reset
- *z(t)* : control if the hidden should be updated

$$\tilde{h}_t = \tanh(Wx_t + U\mathrm{Diag}(r_t)h_{t-1})$$

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

Structurally constrained Network

- A even simpler model than GRU
- Key idea:
  - Instead of using gating, why not having two types of hidden variable:
    - regular ones (similar to RNN)
    - Linear ones (able to capture long term dependencies)

Mikolov et.al., 2014: *Learning Longer Memory in Recurrent Neural Networks*

Structurally constrained network

- No gating:

$$
\begin{aligned}
s_t &= (1-\alpha)Bx_t + \alpha s_{t-1}, \\
h_t &= \sigma\left(Ps_t + Ax_t + Rh_{t-1}\right), \\
y_t &= f\left(Uh_t + Vs_t\right)
\end{aligned}
$$

SCRN in practice

MODEL	# hidden units	Perplexity
N-gram	-	141
N-gram + cache	-	125
SRN	100	129
LSTM	100 (x4 parameters)	115
SCRN	100 + 40	115

- Performs as well as LSTM on Penn TreeBank
- Performs a bit worse on text8 (~ 5 ppl worse)

→ Most of the long term dependency captured by LSTM is same as with **RNN + bag of words**?

Extensions to other domains

- So far, we mostly talked about language modeling

- How to apply them to other domains?
  - Machine translation
  - Image captioning
  - Frame prediction in videos

Machine Translation

- Learn from example how to translate sentences between a *source* language and a *destination* language

- The two sentences may have different length → Cannot use RNN directly

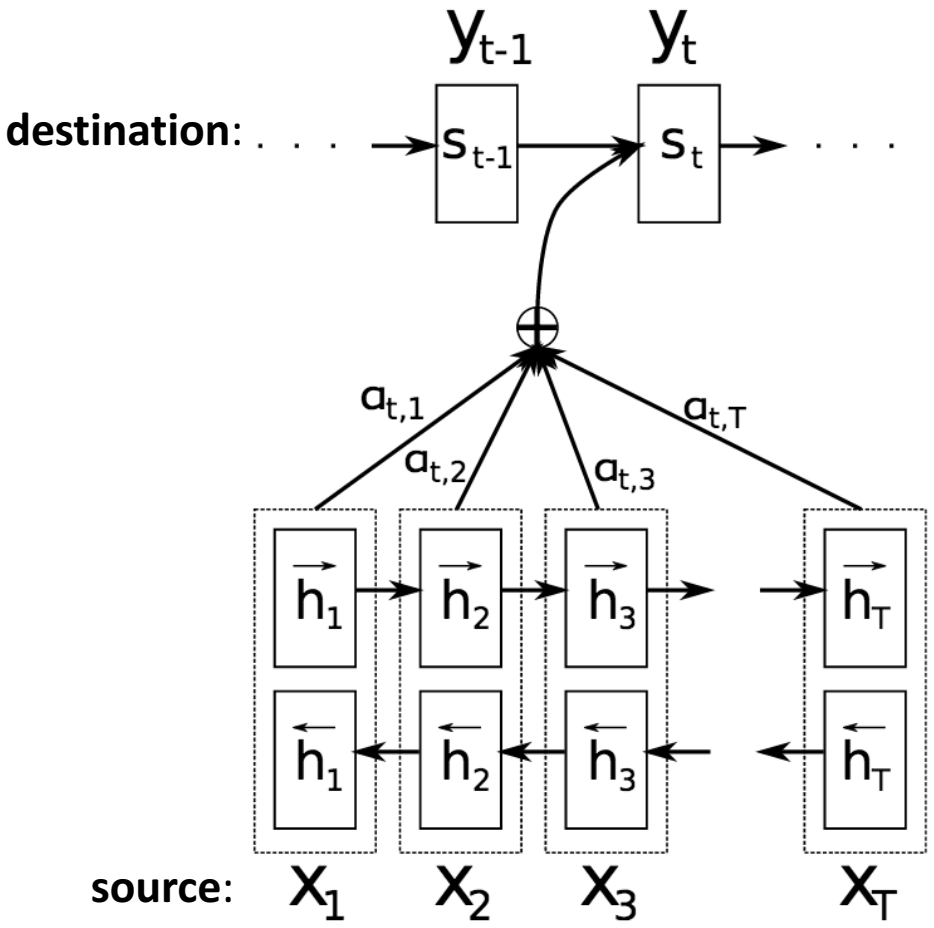
- 2 solutions:
  - Condition prediction of *each word* in destination sentence by the source sentence (Bahdanau et al. 2014)
  - Condition prediction of *the whole* destination sentence by the source sentence (Sutskever et al. 2014)

Word level conditioning

- Simple idea:
  - Use the RNN on the *destination* language
  - Condition each word by the *source* sentence
  - Example: condition with a Bag of word
  - Every words in the source has the same weight

  → Learn the weights

Word level conditioning



Each prediction *y(t)* is condition by a vector *c(t)* such that:

$$c_t = \sum_k \alpha_{tk} h_k$$

c(t) = weighted sum of the representation of source words.

If weights are equal → BoW.

Attention mechanism: Make the weights depends on the **relation** between source and destination representations:

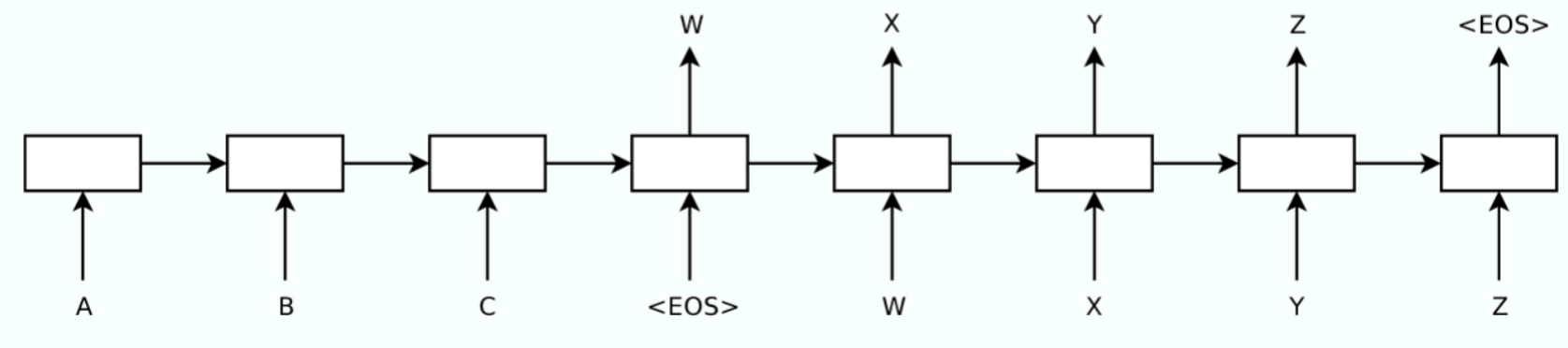
$$\alpha_{tk} = \exp(e_{tk}) / \sum_i \exp(e_{ti})$$

$$e_{tk} = f(s_t, h_k)$$

"*Neural machine translation by jointly learning to align and translate*"; D Bahdanau et al. 2014

Sentence level conditioning

- Simple idea:
  - process the source sentence with an RNN
  - use the states of the hidden layers to condition a RNN on the destination sentence
  - Drawback: requires massive RNN to remember information about source sentence



Sutskever at. al., 2014: *Sequence to Sequence Learning with Neural Network*

Machine Translation

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

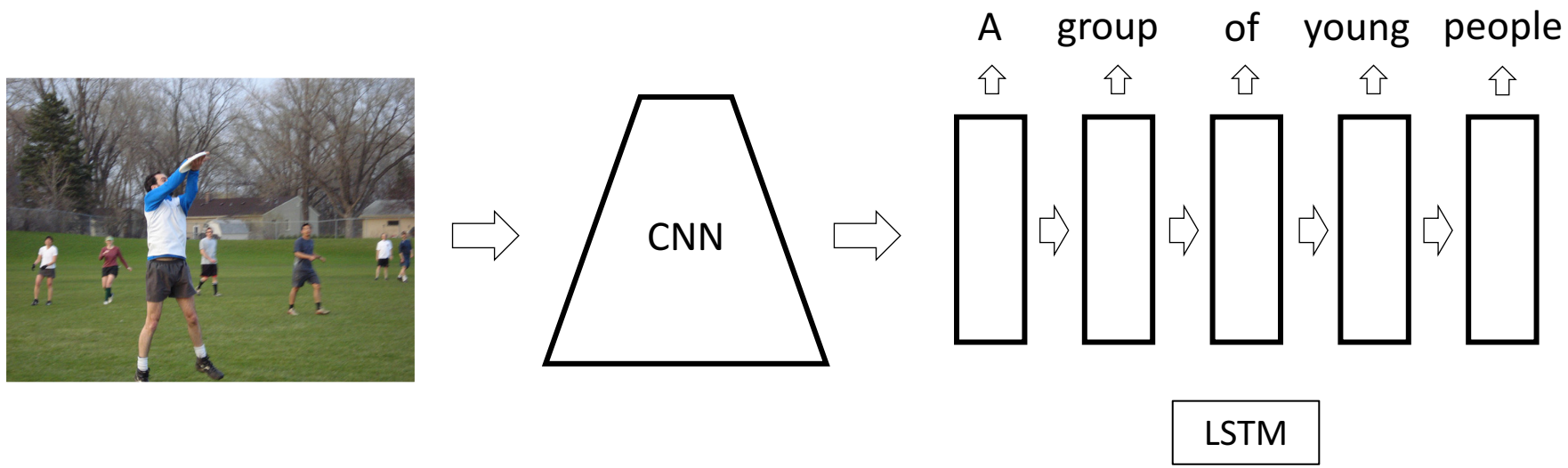
State-of-the-art WMT'14 result: 37.0

Sequence to Sequence Learning with Neural Network, Sutskever at. al., 2014

Image Captioning

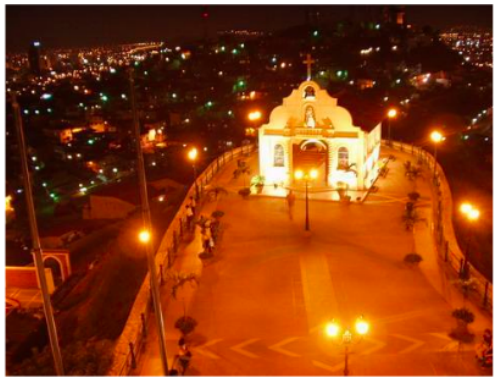
- Same idea
- Condition the RNN on an image feature (CNN):
  - either at each word prediction
  - or at the beginning of the sentence
- Lot of papers (Mao et al. 2014, Karpathy et al. 2014,Xu et al. 2015…)
- with all possible variants of conditioning

Example of captioning model



Show and Tell: A Neural Image Caption Generator, Vinyals et al., CVPR 2015

Image captioning: examples



A square with burning street lamps and a street in the foreground;

Tourists are sitting at a long table with a white table cloth and are eating;

A dry landscape with green trees and bushes and light brown grass in the foreground and reddish-brown round rock domes and a blue sky in the background;

A blue sky in the background;

Deep captioning with multimodal recurrent neural networks, Mao et al. 2014

Image captioning?

- Issue: The datasets are quite small (flickr8k, flickr30K, COCO80K)

- Larry Zitnick, Organizer of the caption challenge:

“35% - 85% of captions are identical to training captions”

- Is “captioning” simply sentence retrieval?

Nearest neighbor for captioning

A black and white cat sitting in a bathroom sink.

Two zebras and a giraffe in a field.

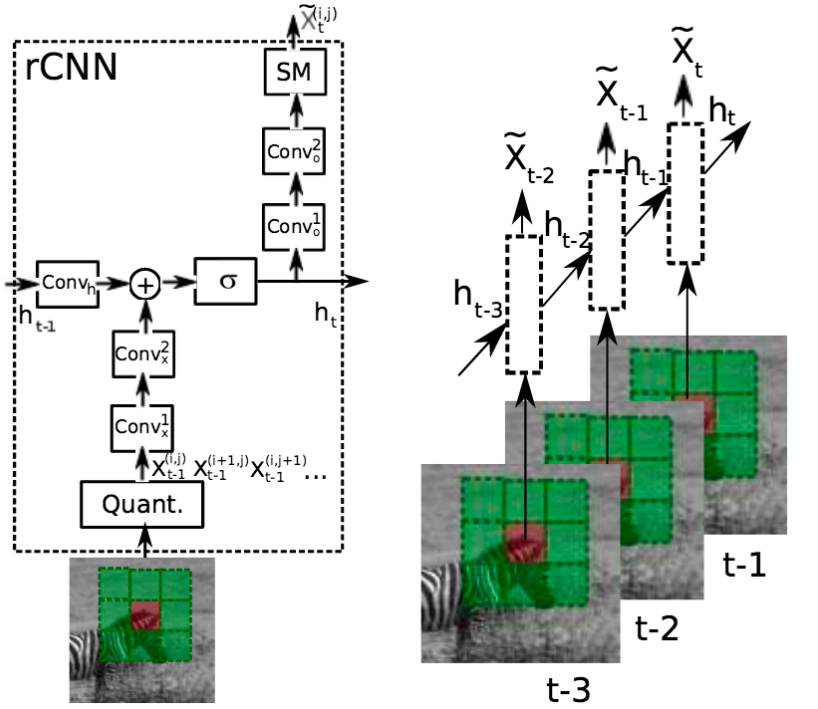
Exploring Nearest Neighbor Approaches for Image Captioning, Devlin et al., *ArXiv* 1505.04467, 2015

Frame prediction in videos

- RNN can also be used in the continuous domain
- Example:
  - Frame prediction in videos
  - Allows unsupervised learning of visual features!
  - Consider the video as a sequence and try to predict the next frames (Ranzato et al. 2014, Matthieu et al., 2015)

Frame prediction in videos

- Extract a CNN feature from a patch in a frame
- Run a RNN on top to predict how it will change



Video (language) modeling: a baseline for generative models of natural videos; Ranzato et al. 2014

Summary: RNN

- RNNs are simple sequence prediction models
- They can be trained efficiently on large corpus of data (with GPUs)
- They are state-of-the-art language models
- Successfully applied to speech, machine translation and computer vision

Plan of this lecture

- Supervised neural networks
- Optimization for neural networks
- Convolutional network
- Recurrent network
- **Unsupervised learning**

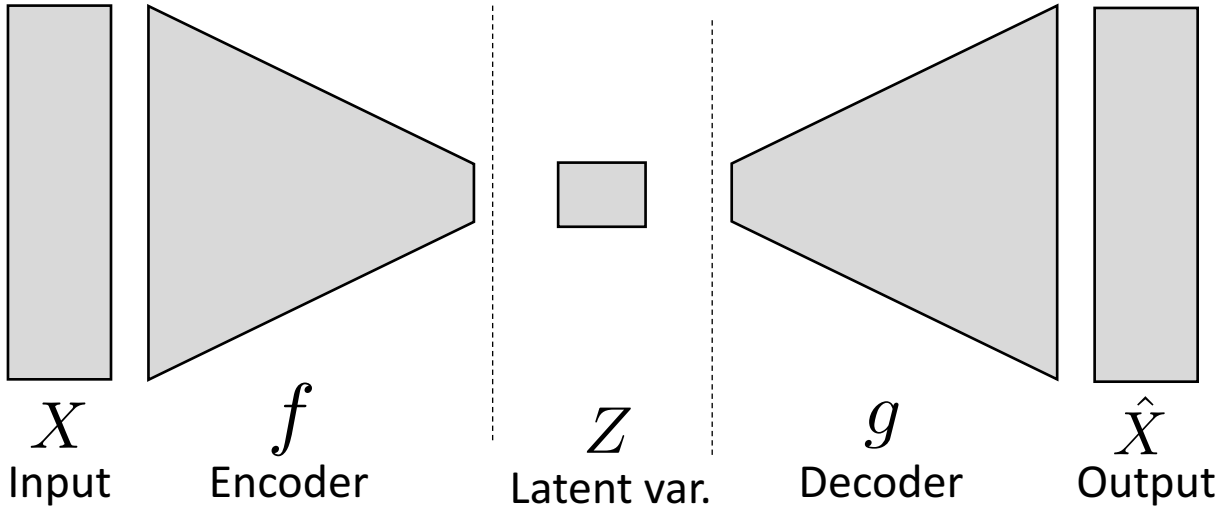
Unsupervised learning

- Standard setting: **learn classifier and assignment simultaneously**
- Example of algorithm: k-means, GMM…
- In deep learning, unsupervised learning also means **learning features with no annotation**.

Unsupervised learning

- Several standard approaches to learn features with no supervision:
  - **Auto-encoder (AE)**: learn features by reconstructing the input after compression
  - **Generative Adversarial Network (GAN)**: learn simultaneously to generate fake inputs and to differentiate them from real inputs
  - **Generative Latent Optimization (GLO)**: learn latent space and generator simultaneously
  - **Self-supervision**: learn domain specific signal
  - **Noise as Targets (NAT)**: learn features by maximizing feature coverage of the space

Autoencoder

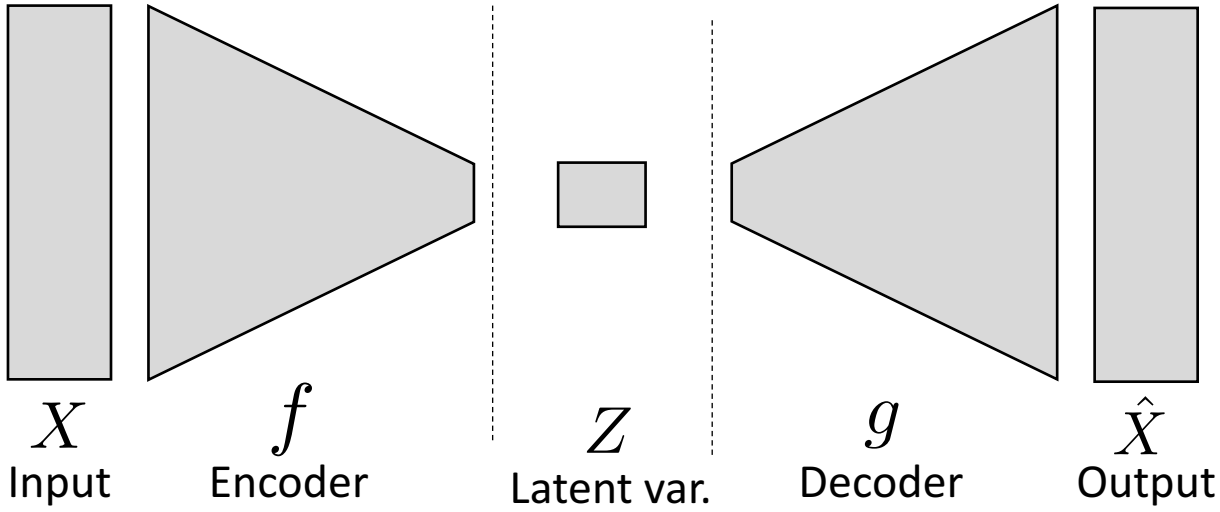


$$\hat{X} = g(f(X))$$

$$\min_{f,g} \|\hat{X} - X\|$$

- Compress input to a latent variable Z → **Encoder**
- Reconstruct input from that latent variable → **Decoder**
- Learn both encoder and decoder simultaneously

Autoencoder (AE)

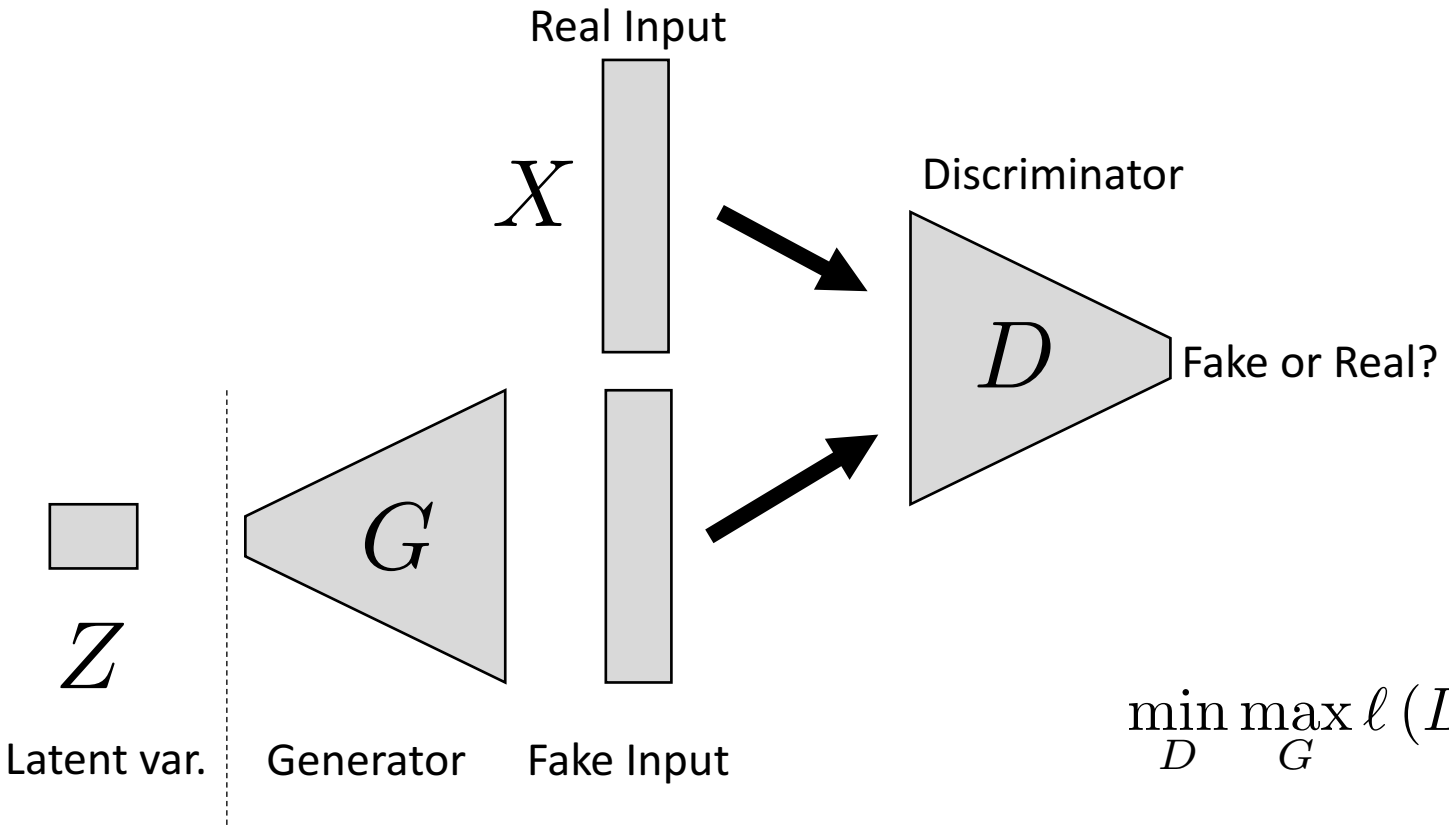


$$\hat{X} = g(f(X))$$

$$\min_{f,g} \|\hat{X} - X\|$$

- Trivial solutions if latent space is too large
- Solutions: add constraints on this space or inject noise (noisy AE)
- Choice of loss is important: L2 distance produces blurry reconstruction

Generative Adversarial Network (GAN)



- **Generator (G)** produces fake input from latent variable
- **Discriminator (D)** learns if an input is real or fake.

→ G tries to “foul” D

$$\min_D \max_G \ell\left(D(X), 1\right) + \ell\left(D(G(Z)), -1\right)$$

real fake

Generative Adversarial Network (GAN)

$$\min_{D} \max_{G} \ell\left(D(X), 1\right) + \ell\left(D(G(Z)), -1\right)$$

- Saddle point problem → very unstable
- Hard to learn, converges easily to poor solutions
- Often learns a few modes of the real distribution
- But when trained “correctly” produces great “fake images”

Generative Adversarial Network (GAN)

- Fake faces from Radford et al. 2015

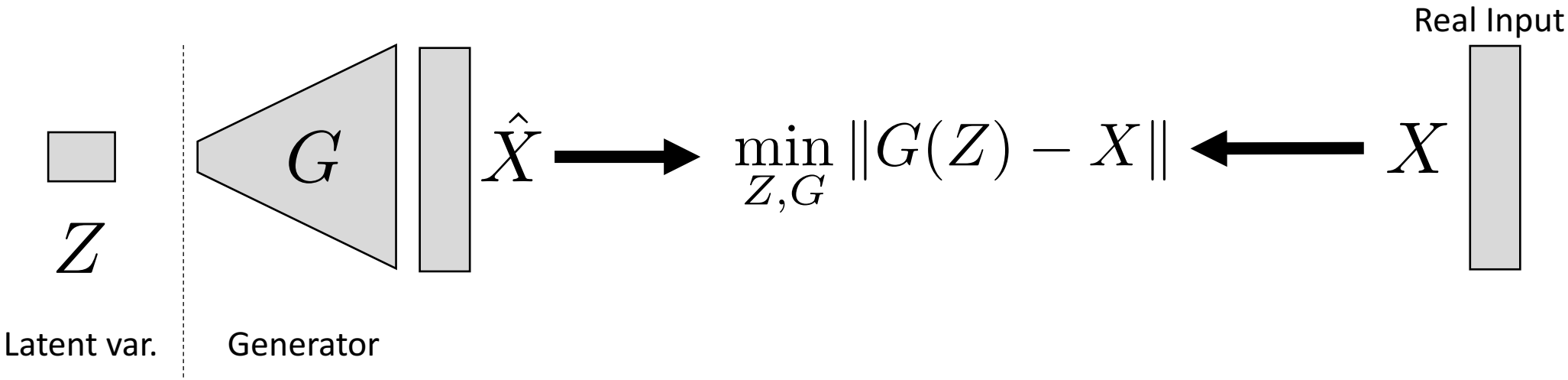
Generative Adversarial Network (GAN)

- Fake images from Karras, 2017
- Bigger model, more stable training... And a lot of engineering

Generative Adversarial Network (GAN)

- Generated images are impressive
- But the discriminator is rarely useful
- Open questions:
  - Do we really need a discriminator?
  - And can we train generators with simpler, more stable loss?

Generative Latent Optimization (GLO)



- Learn the latent space.
- Much simpler to train.

Generative Latent Optimization (GLO)

GAN

GLO

- Fake images from Bojanowski et al., 2017

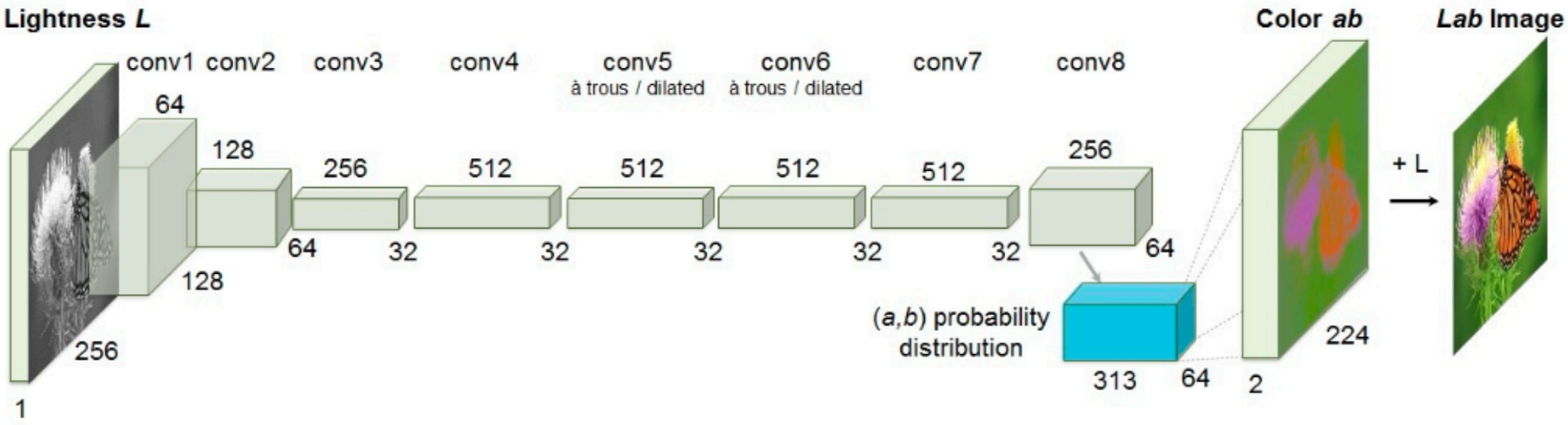
Self-supervision

- Predict a **domain specific signal** from the data

- Inspired by word2vec (Mikolov, 2013)

- For example:
  - learn to colorize gray images
  - Predict next frames in a video
  - Learn to re-order patches of an image
  - Predict half of an image with the other half…

Self-supervision



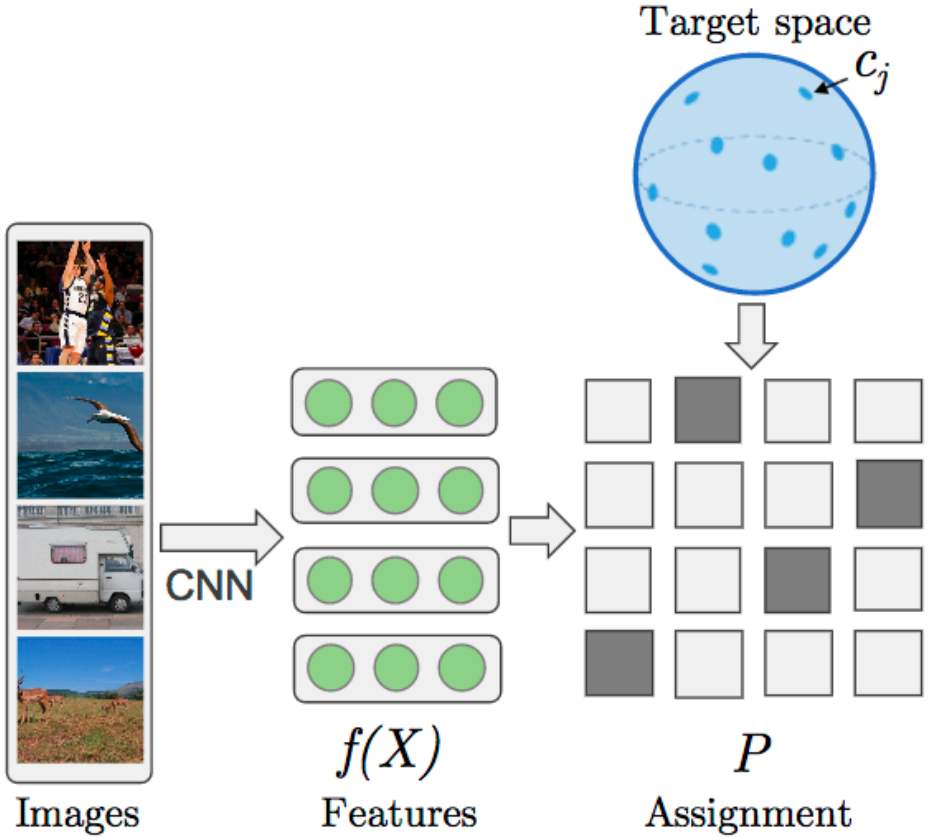
Colorful Image Colorization, Zhang et al. 2016

Self-supervision

Colorful Image Colorization, Zhang et al. 2016

Noise as Target (NAT)

$$\max_{\theta} \max_{P \in \mathcal{P}} \mathrm{Tr}\left(PC f_\theta(X)^\top\right)$$



- Learn image features by maximizing the coverage of a target space
- In some sense, it is a continuous version of k-means for deep learning
- Simple loss function

Noise as target (NAT)

	Classification		Detection
Trained layers	fc6-8	all	all
ImageNet labels	78.9	79.9	56.8
Agrawal et al. (2015)	31.0	54.2	43.9
Pathak et al. (2016)	34.6	56.5	44.5
Wang & Gupta (2015)	55.6	63.1	47.4
Doersch et al. (2015)	55.1	65.3	51.1
Zhang et al. (2016)	61.5	65.6	46.9
Autoencoder	16.0	53.8	41.9
GAN	40.5	56.4	-
BiGAN (Donahue et al., 2016)	52.3	60.1	46.9
NAT	56.7	65.3	49.4

VOC 2007

Learns features comparable with state-of-the-art unsupervised approaches!

From Bojanowski and Joulin (2017)

Unsupervised learning

- Very active topic
- Impressive results in image generation
- Feature learning is harder and less progress has been made

Conclusion

- Neural networks are complex non-linear models
- Optimizing them is hard
- But they have been applied successfully to many domains
- Allow end-to-end training of multimodal models
- Still unclear why do they work so well

References

- “Neural networks and optimization”, N. Le Roux. 2016. http://www.di.ens.fr/willow/teaching/recvis16/slides/lecture05_neural_networks_and_optimization.pdf
- “The Perceptron--a perceiving and recognizing automaton”, F. Rosenblatt. 1957. https://en.wikipedia.org/wiki/Perceptron
- “Perceptrons”, Papert and Minsky. 1969.
- Cours IFT 725. H. Larochelle. 2014. http://www.dmi.usherb.ca/~larocheh/cours/ift725_A2014/contenu.html
- “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. 2014. http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
- “Stochastic gradient descent tricks”. L. Bottou. 2012. https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf
- “Rectifier nonlinearities improve neural network acoustic models”, A. Maas, A Hannun and A. Ng. 2013. http://web.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf
- “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Ioffe and Szegedy. 2015. https://arxiv.org/abs/1502.03167
- “Adaptive subgradient methods for online learning and stochastic optimization”. J. Duchi, E. Hazan and Y. Singer. 2011. http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

References

- “Adam: A method for stochastic optimization”, Kingma and Ba. 2014. https://arxiv.org/pdf/1412.6980.pdf
- “Learning representations by back-propagating errors”, Rumelhart, Hinton and Williams. 1986
- “Exploring generalization in deep learning”, Neyshabur, Bhojanapalli, McAllester. 2017. https://arxiv.org/pdf/1706.08947
- “Gradient-based learning applied to document recognition”, LeCun, Bottou, Bengio and Haffner. 1998. http://www.dengfanxin.cn/wp-content/uploads/2016/03/1998Lecun.pdf
- “A neural probabilistic language model”, Bengio, Ducharme, Vincent and Jauvin. 2003. http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- “Finding structure in time”, Elman. 1990. http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1402_1/pdf
- “Recurrent neural network based language model”, Mikolov, Karafiat, Burget, Cernocky, Khudanpur. 2010. http://www.fit.vutbr.cz/research/groups/speech/servite/2010/rnnlm_mikolov.pdf
- “Learning long-term dependencies with gradient descent is difficult”, Bengio, Simard, Frasconi. 1994. http://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf
- “Long short-term memory”, S Hochreiter, J Schmidhuber. 1997. http://web.eecs.utk.edu/~itamar/courses/ECE-692/Bobby_paper1.pdf

References

- “Speech Recognition with Deep Recurrent Neural Networks”, A Graves, A Mohamed, G Hinton. 2013. https://arxiv.org/pdf/1303.5778.pdf
- “Gated feedback recurrent neural networks”, J Chung, C Gulcehre, K Cho, Y Bengio. 2014. http://proceedings.mlr.press/v37/chung15.pdf
- “Learning longer memory in recurrent neural networks”, T Mikolov, A Joulin, S Chopra, M Mathieu, MA Ranzato. 2014. https://arxiv.org/pdf/1412.7753
- “Neural machine translation by jointly learning to align and translate”, D Bahdanau, K Cho, Y Bengio. 2014. https://arxiv.org/pdf/1409.0473
- “Sequence to sequence learning with neural networks”, I Sutskever, O Vinyals, QV Le. 2014. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
- “Show and tell: A neural image caption generator”, O Vinyals, A Toshev, S Bengio, D Erhan. 2015. http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Vinyals_Show_and_Tell_2015_CVPR_paper.pdf
- “Deep captioning with multimodal recurrent neural networks”, Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z. and Yuille, A., 2014. https://arxiv.org/pdf/1412.6632
- “Exploring Nearest Neighbor Approaches for Image Captioning”, Devlin J, Gupta S, Girshick R, Mitchell M, Zitnick CL. 2015. https://arxiv.org/pdf/1505.04467

References

- “Video (language) modeling: a baseline for generative models of natural videos”, Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R. and Chopra, S.,. 2014. https://arxiv.org/pdf/1412.6604
- “Generative adversarial nets”, Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
- “Unsupervised representation learning with deep convolutional generative adversarial networks.” Radford A, Metz L, Chintala S. 2015. https://arxiv.org/pdf/1511.06434.pdf%C3%AF%C2%BC%E2%80%B0
- “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, Karras, T., Aila, T., Laine, S. and Lehtinen, J., 2017. https://arxiv.org/pdf/1710.10196
- “Optimizing the Latent Space of Generative Networks”, P Bojanowski, A Joulin, D Lopez-Paz, A Szlam. 2017. https://arxiv.org/pdf/1707.05776
- “Colorful image colorization”, Zhang R, Isola P, Efros AA. 2016. https://arxiv.org/pdf/1603.08511
- “Unsupervised Learning by Predicting Noise”, Bojanowski and Joulin, 2017. https://arxiv.org/pdf/1704.05310