CSE 176 Introduction to Machine Learning

Final Exam Review

Neural Network (Topic 9, 10, 11,12)

Neural Unit

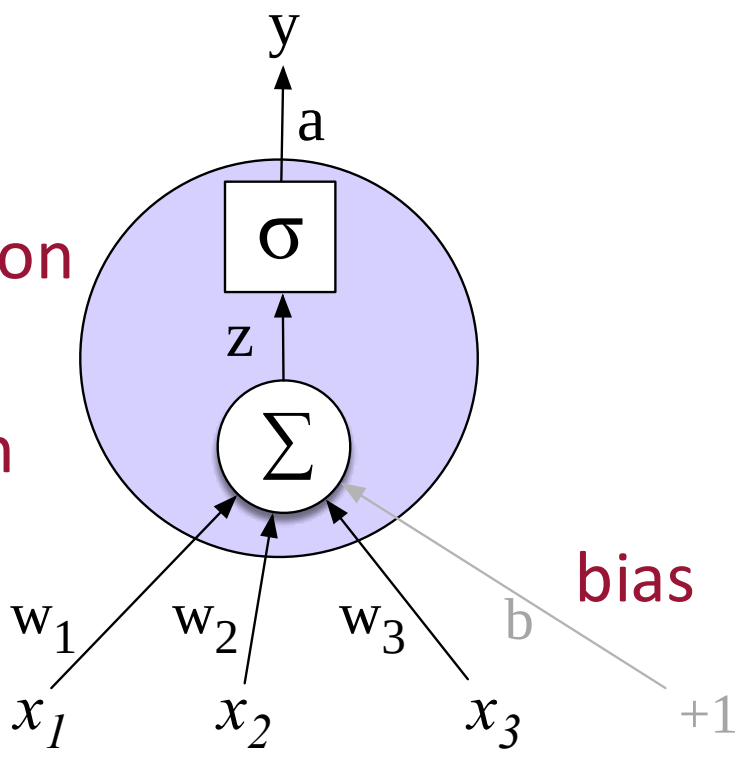
Output value

Non-linear activation function

Weighted sum

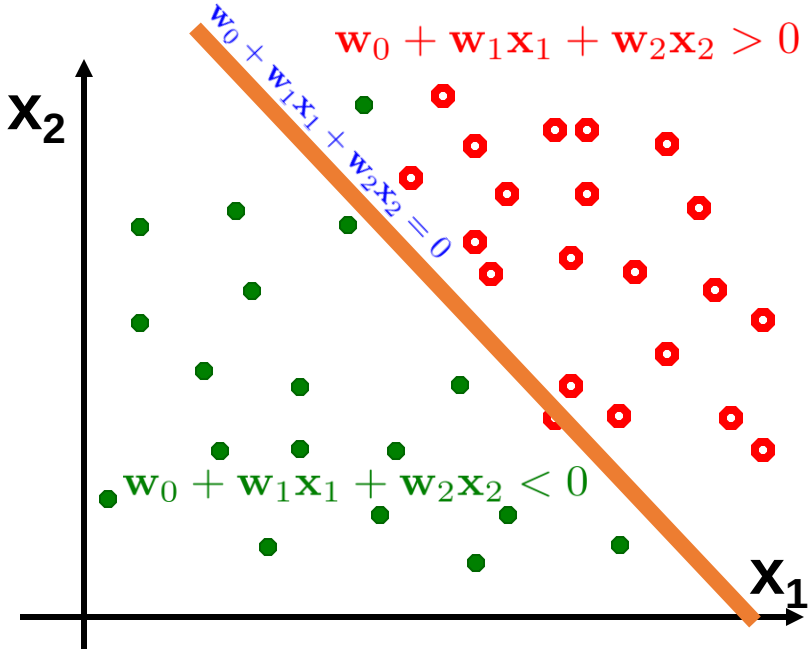
Weights

Input layer



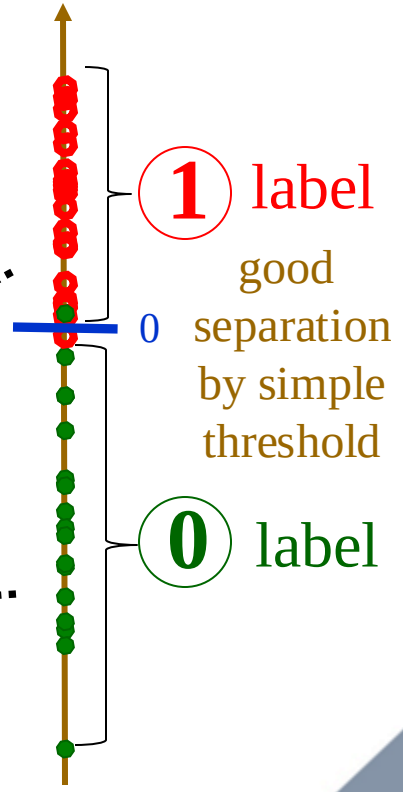
Linear Classification (perceptron)

- For two class problem and 2-dimensional data (feature vectors)



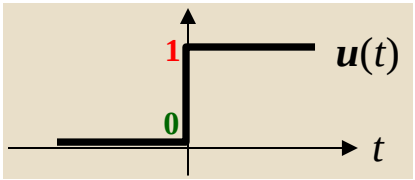
"good"
linear transformation
from 2D space to 1D

$\mathbf{w_0^*+w_1^*x_1+w_2^*x_2}$



thresholding can be formally represented by this prediction function

$$\mathbf{f(w,x)} = u\,(\mathbf{w_0+w_1x_1+w_2x_2}) \qquad \mathbf{f(w,x)} \in \{0,1\}$$



unit step function
(a.k.a. *Heaviside* function)

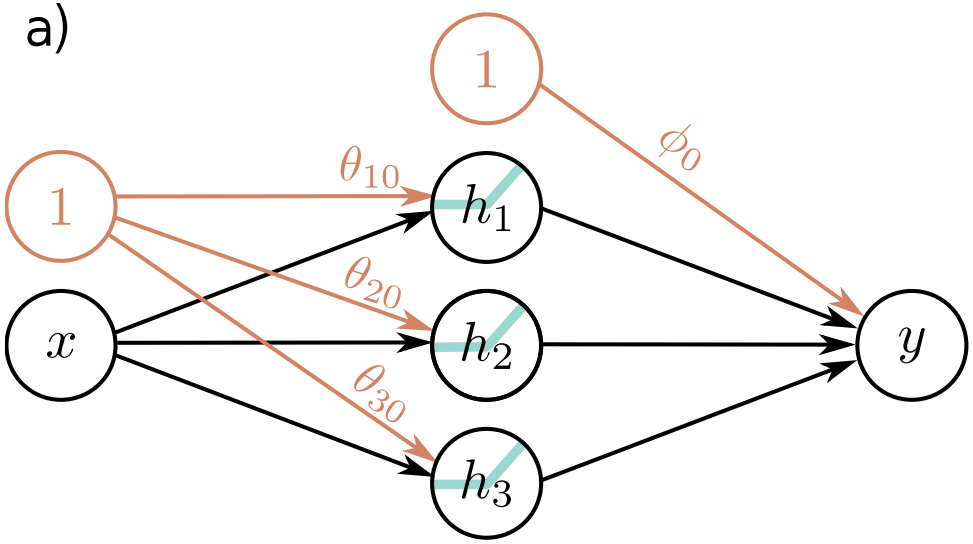
$$u(t) := \begin{cases} 1 & \text{if } t > 0 \\ 0 & OW \end{cases}$$

Depicting shallow neural networks

$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

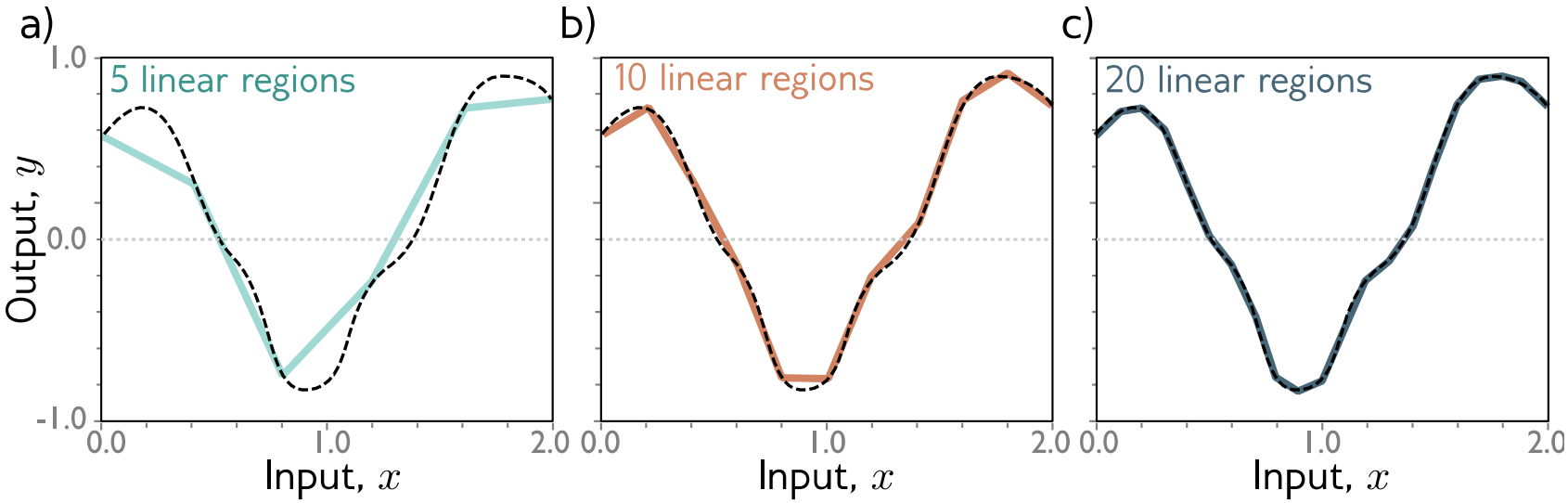
a)



Each parameter multiplies its source and adds to its target

With enough hidden units

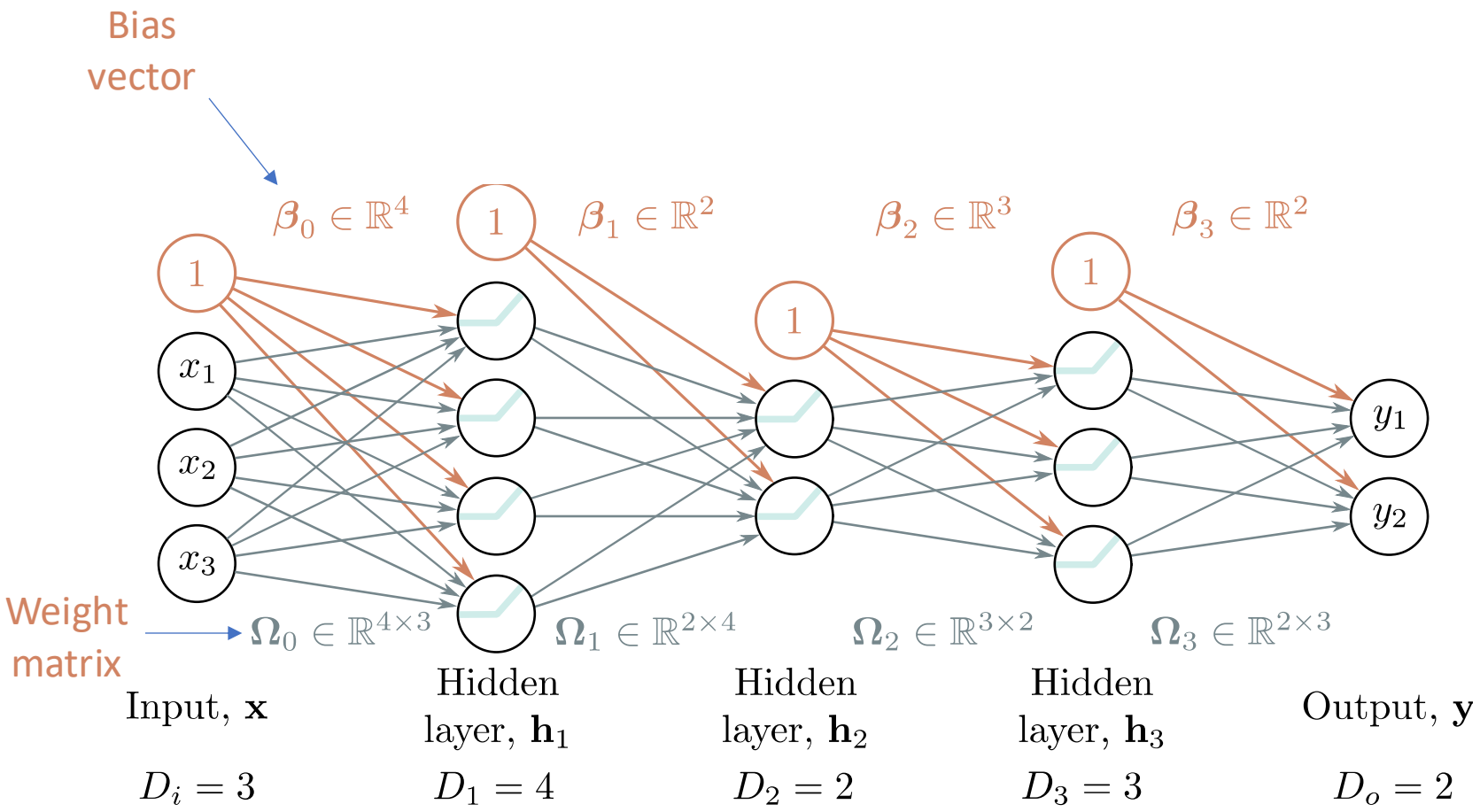
❑… we can describe any 1D function to arbitrary accuracy



Universal approximation theorem

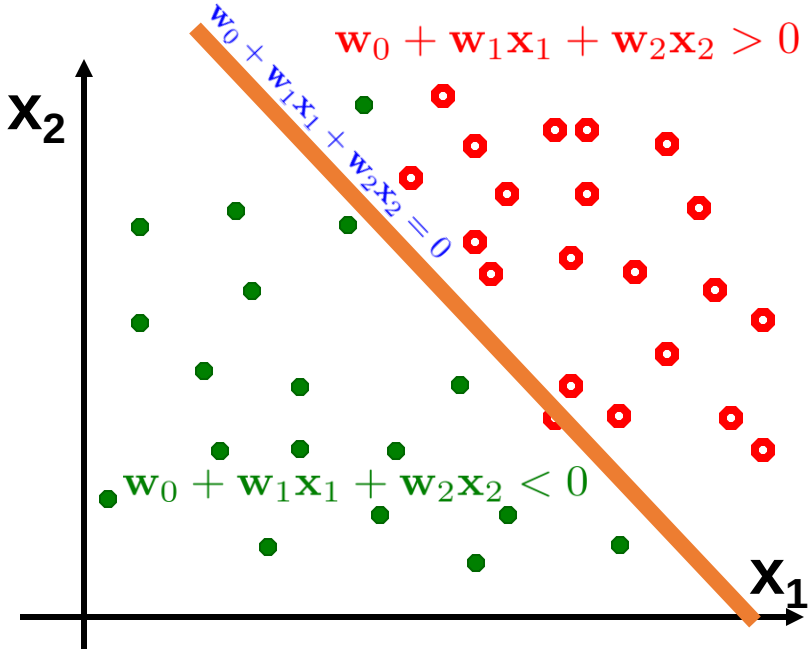
"a formal proof that, with enough hidden units, a shallow neural network can describe any continuous function on a compact subset of $\mathbb{R}^D$ to arbitrary precision"

Example of Multi Layer Perceptron (MLP)



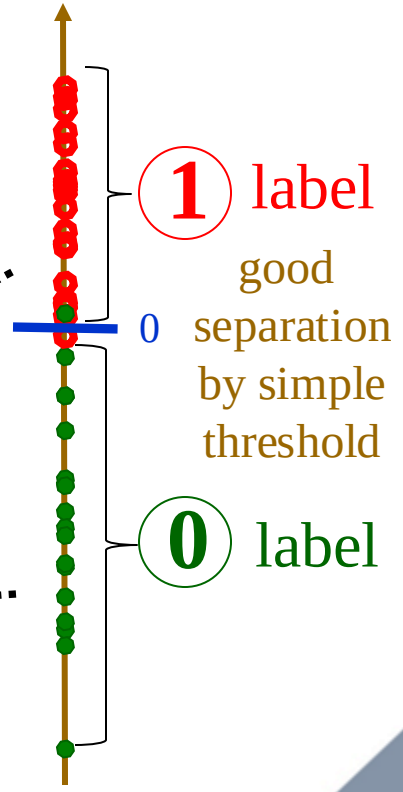
Linear Classification (perceptron)

- For two class problem and 2-dimensional data (feature vectors)



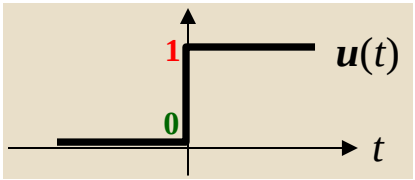
"good"
linear transformation
from 2D space to 1D

$\mathbf{w_0^*+w_1^*x_1+w_2^*x_2}$



thresholding can be formally represented by this prediction function

$$\mathbf{f(w,x)} = u\,(\mathbf{w_0+w_1x_1+w_2x_2}) \qquad \mathbf{f(w,x)} \in \{0,1\}$$

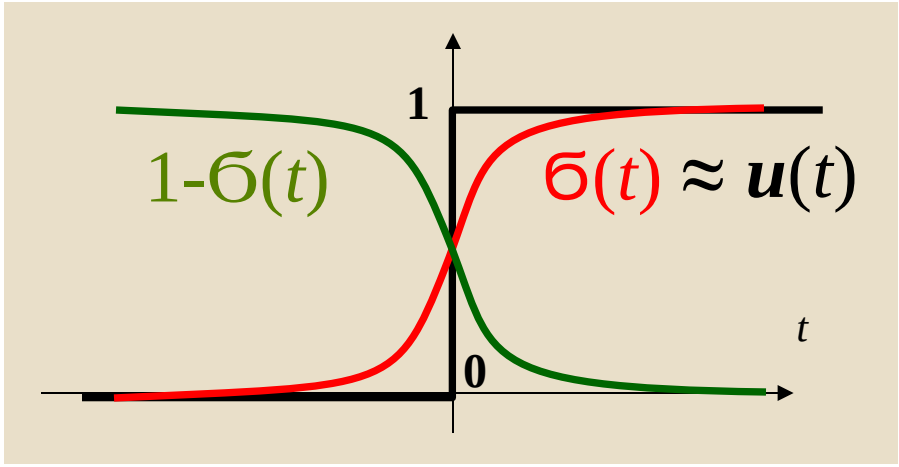


unit step function
(a.k.a. *Heaviside* function)

$$u(t) := \begin{cases} 1 & \text{if } t > 0 \\ 0 & OW \end{cases}$$

Perceptron: $\mathbf{f}(\mathbf{w}, \mathbf{x}^i) = u(W^T X^i) \approx \sigma(W^T X^i)$

approximate decision function *u* using its **softer** version (relaxation)



u(*t*) - ***unit step*** function
(a.k.a. *Heaviside* function)

σ(*t*) - ***sigmoid*** function

$$\sigma(t) := \frac{1}{1 + \exp(-t)}$$

Relaxed predictions are often interpreted as prediction **"probabilities"**

$$\Pr(\mathbf{x}^i \in \text{Class1} \mid W) = \sigma(W^T X^i)$$

$$\Pr(\mathbf{x}^i \in \text{Class0} \mid W) = 1 - \sigma(W^T X^i) \equiv \sigma(-W^T X^i)$$

(binary case)

Cross-Entropy Loss (related to *logistic regression* loss)

Perceptron approximation: $\mathbf{f}(\mathbf{w}, \mathbf{x}^i) = u(W^T X^i) \approx \sigma(W^T X^i)$

Consider two probability distributions
over two classes (e.g. bass or salmon) : $(\mathbf{y}, 1-\mathbf{y})$ and $(\sigma, 1-\sigma)$

(binary)
Cross-entropy loss: $L(\mathbf{y}, \sigma) = -\mathbf{y} \ln \sigma - (1-\mathbf{y}) \ln(1-\sigma)$

Distance between two distributions can be evaluated via **cross-entropy**
(equivalent to ***KL divergence*** for fixed target)

$$H(\boldsymbol{p}, \boldsymbol{q}) := -\sum_k p_k \ln q_k$$

(general multi-class case)

Cross-Entropy Loss

K-label perceptron's output: $\bar{\sigma}(\mathbf{W}X^i)$ for example X^i *k*-th index

Multi-valued label $\mathbf{y}^i = k$ gives **one-hot** distribution $\bar{\mathbf{y}}^i = (0, 0, 1, 0, \ldots, 0)$

Consider two probability distributions over K classes (e.g. bass, salmon, sturgeon): $\bar{\mathbf{y}}^i$ and $(\bar{\sigma}_1, \bar{\sigma}_2, \bar{\sigma}_3, \ldots, \bar{\sigma}_K)$

$$\Pr(\mathbf{x}^i \in \text{Class } k \mid W) = \bar{\sigma}_k(WX^i)$$

cross entropy

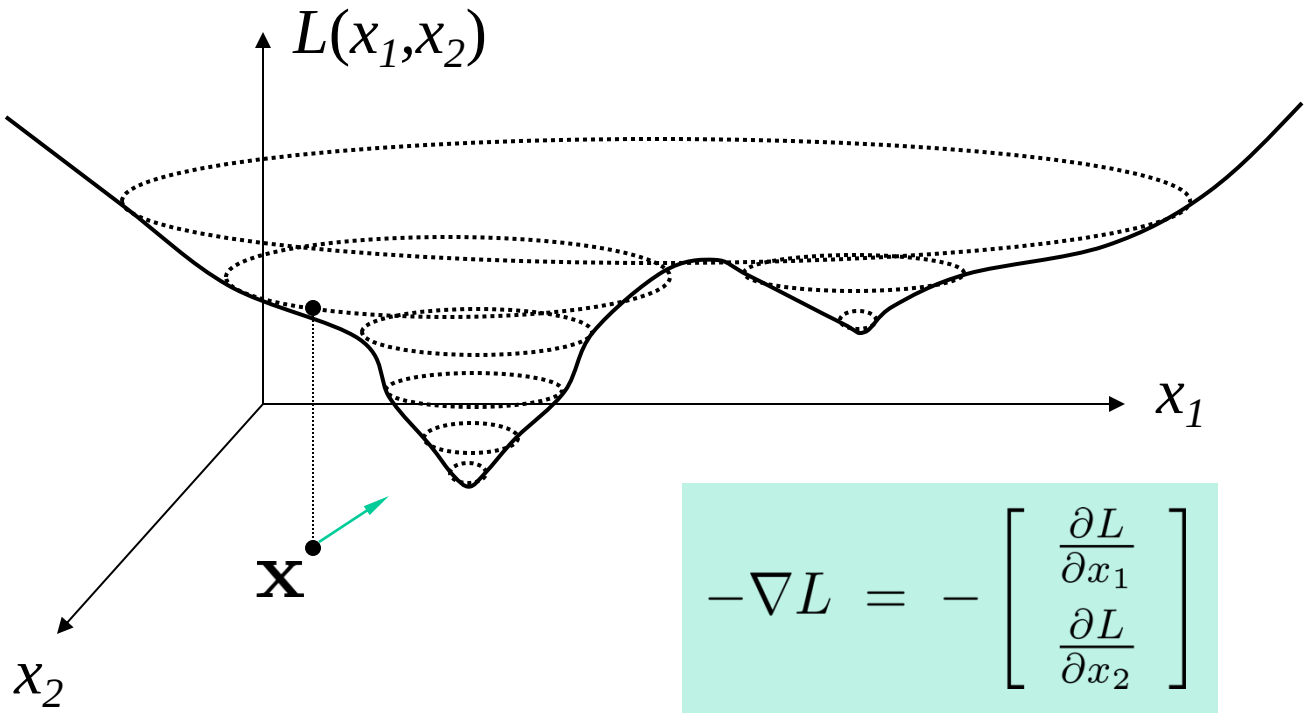
Total loss: $$L(W) = \sum_{i \in \text{train}} \sum_k -\bar{\mathbf{y}}^i_k \ln \bar{\sigma}_k(WX^i)$$

$$\Rightarrow \quad L(W) = -\sum_{i \in \text{train}} \ln \bar{\sigma}_{\mathbf{y}^i}(WX^i)$$

sum of **Negative Log-Likelihoods (NLL)**

Gradient Descent

Example: for a function of two variables

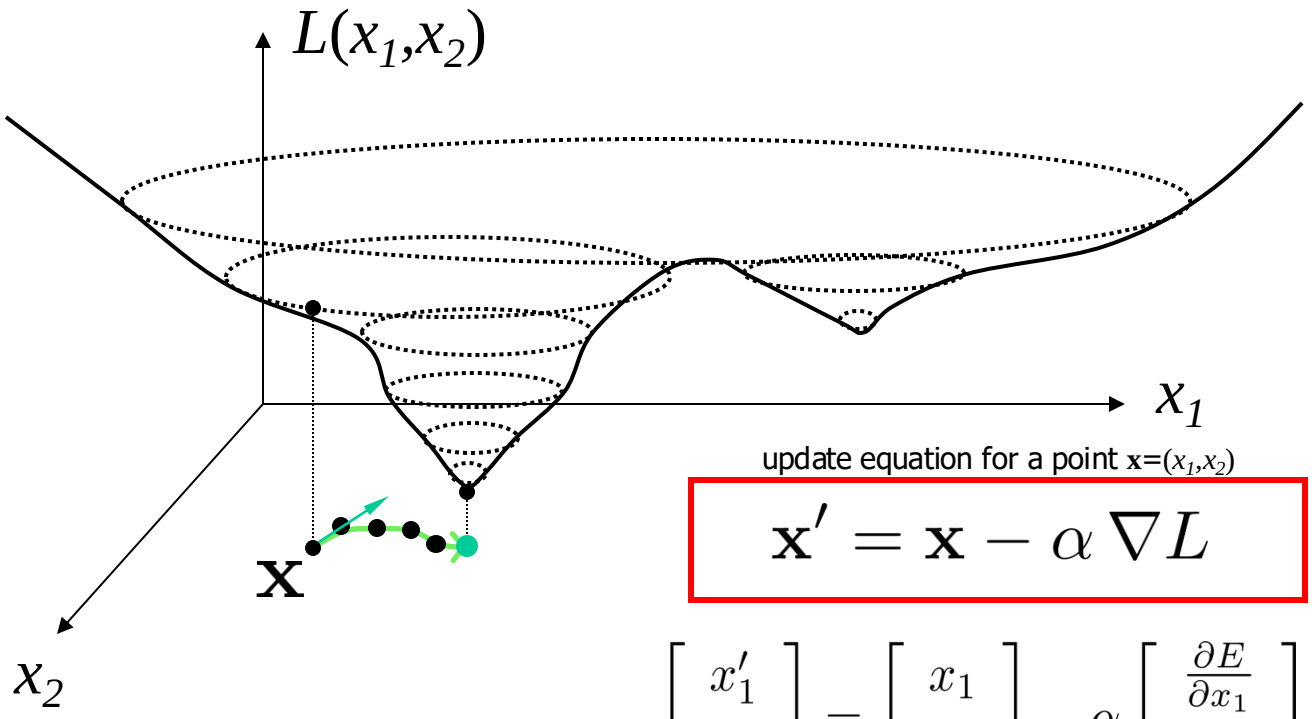


$$-\nabla L = -\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix}$$

- direction of (negative) gradient at point $\mathbf{x}=(x_1,x_2)$ is direction of the steepest descent towards lower values of function L
- magnitude of gradient at $\mathbf{x}=(x_1,x_2)$ gives the value of the slope

Gradient Descent

Example: for a function of two variables



update equation for a point $\mathbf{x}=(x_1,x_2)$

$$\mathbf{x}' = \mathbf{x} - \alpha \nabla L$$

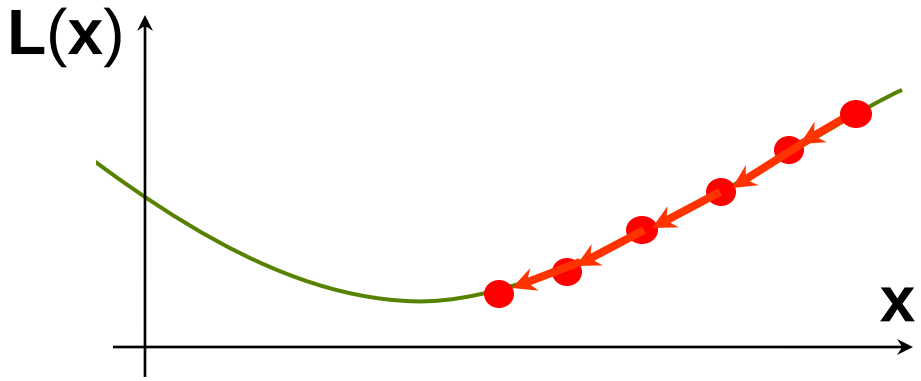
$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \end{bmatrix}$$

Stop at a local minima where $\nabla L = \vec{0}$

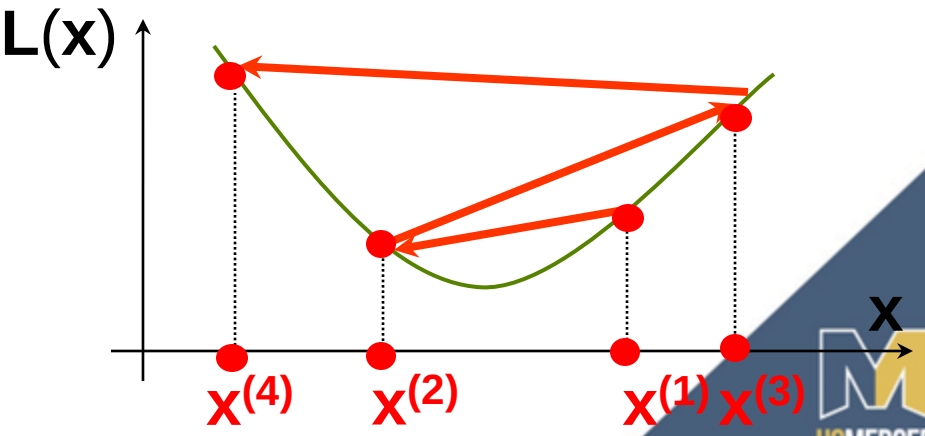
How to Set Learning Rate α?

$$\mathbf{x}' = \mathbf{x} - \alpha \nabla L$$

- If α too small, too many iterations to converge

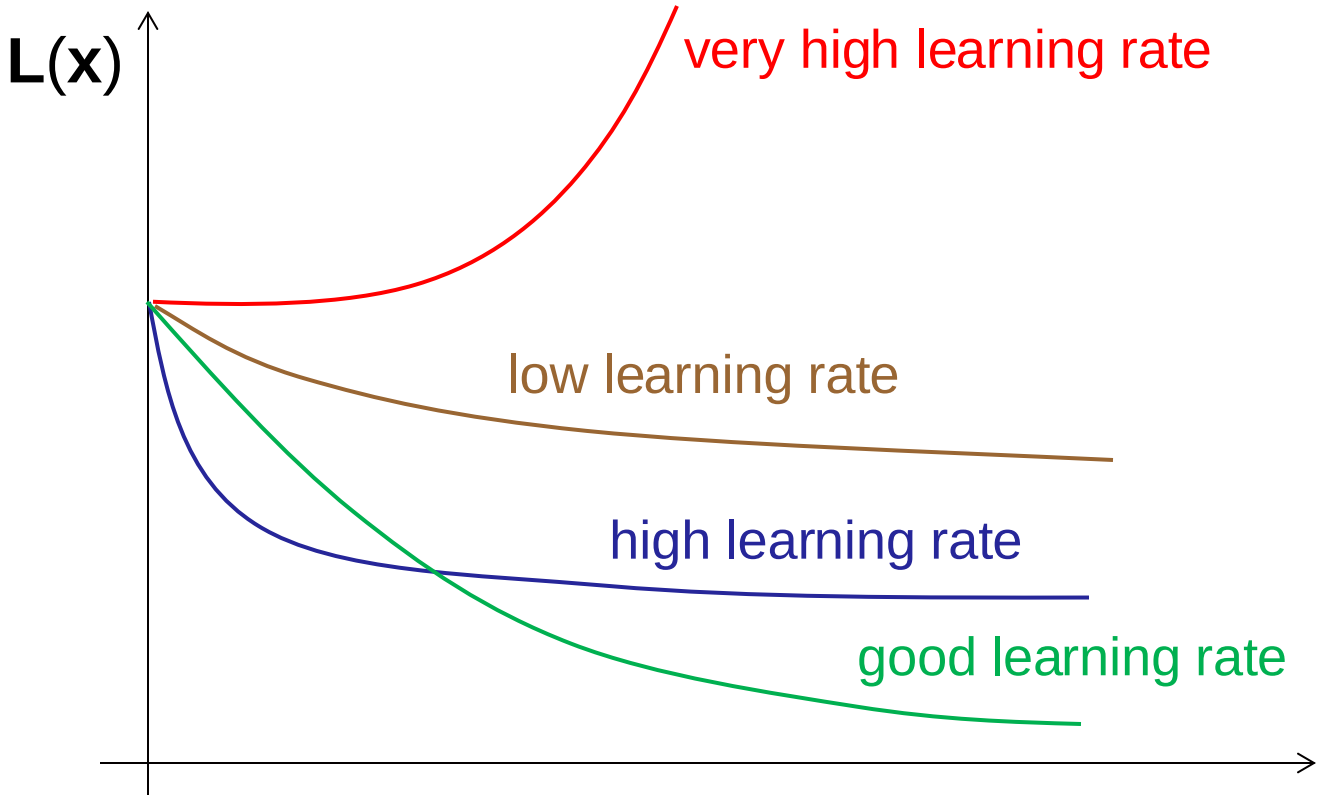


- If α too large, may overshoot the local minimum and possibly never even converge



Learning Rate

- Monitor learning rate by looking at how fast the objective function decreases



Variable Learning Rate

If desired, can change learning rate α at each iteration

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose α, ε

while $\alpha \|\nabla L(\mathbf{x}^{(k)})\| > \varepsilon$

$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla L(\mathbf{x}^{(k)})$

$\mathbf{k} = \mathbf{k} + 1$

fixed α
gradient descent

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose ε

while $\alpha \|\nabla L(\mathbf{x}^{(k)})\| > \varepsilon$

choose $\alpha^{(k)}$

$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)} \nabla L(\mathbf{x}^{(k)})$

$\mathbf{k} = \mathbf{k} + 1$

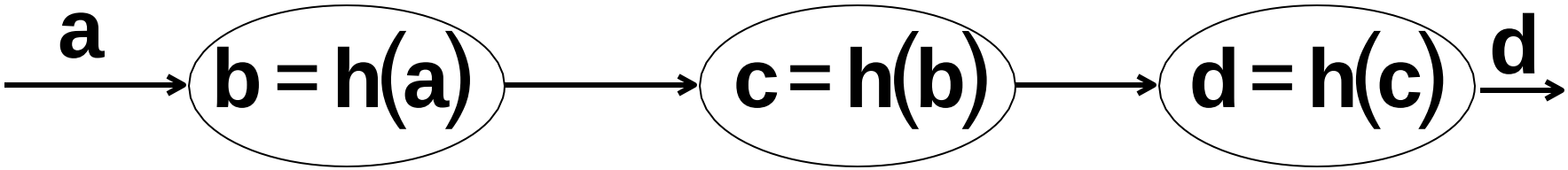
variable α
gradient descent

UCMERCED

Computing Derivatives: Chain of Chain Rule

- Compute $\frac{\partial \mathbf{d}}{\partial}$ from the end backwards
  - for each edge, with respect to the main variable at edge origin
  - using chain rule with respect to the variable at edge end, if needed

direction of computation

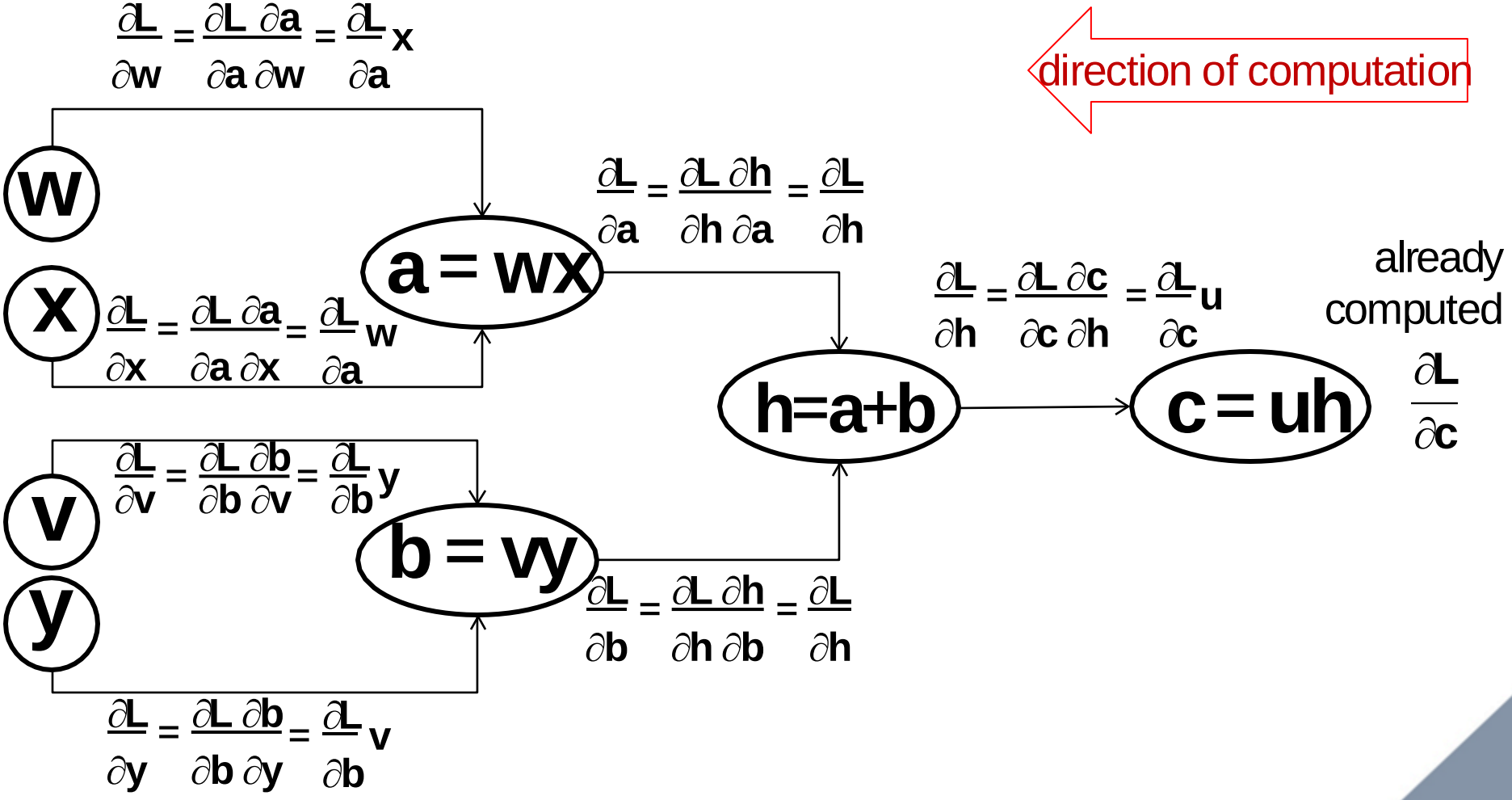


$$\frac{\partial \mathbf{d}}{\partial \mathbf{a}} = \frac{\partial \mathbf{d}}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \qquad \frac{\partial \mathbf{d}}{\partial \mathbf{b}} = \frac{\partial \mathbf{d}}{\partial \mathbf{c}} \frac{\partial \mathbf{c}}{\partial \mathbf{b}} \qquad \frac{\partial \mathbf{d}}{\partial \mathbf{c}}$$

prev local prev local local

example: if $h(c) = c^2$, then $\frac{\partial \mathbf{d}}{\partial \mathbf{c}} = \frac{\partial \mathbf{h}}{\partial \mathbf{c}} = 2\mathbf{c}$

Computing Derivatives: Look at One Node

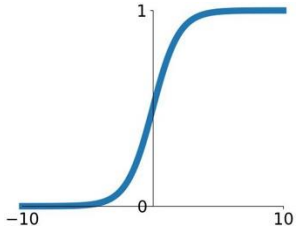


- Some of these partial derivatives are intermediate
  - their values will not be used for gradient descent

Activation Functions

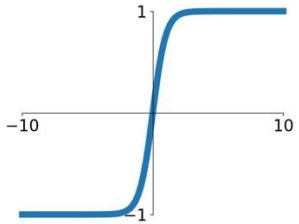
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



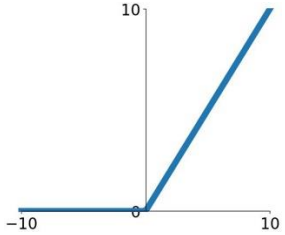
tanh

$$\tanh(x)$$



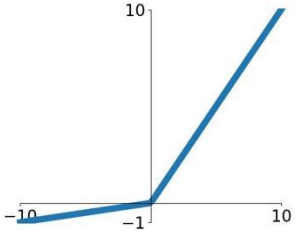
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

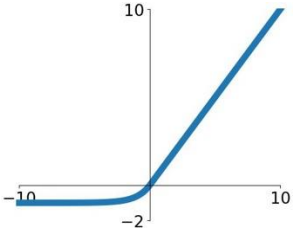


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

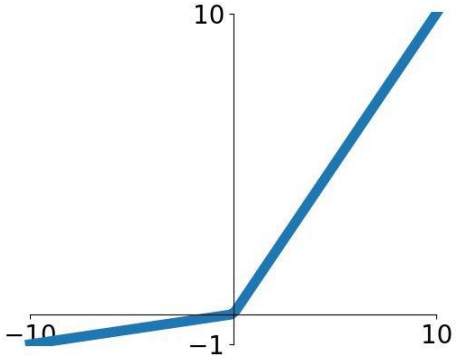
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions

[Mass et al., 2013]
[He et al., 2015]



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

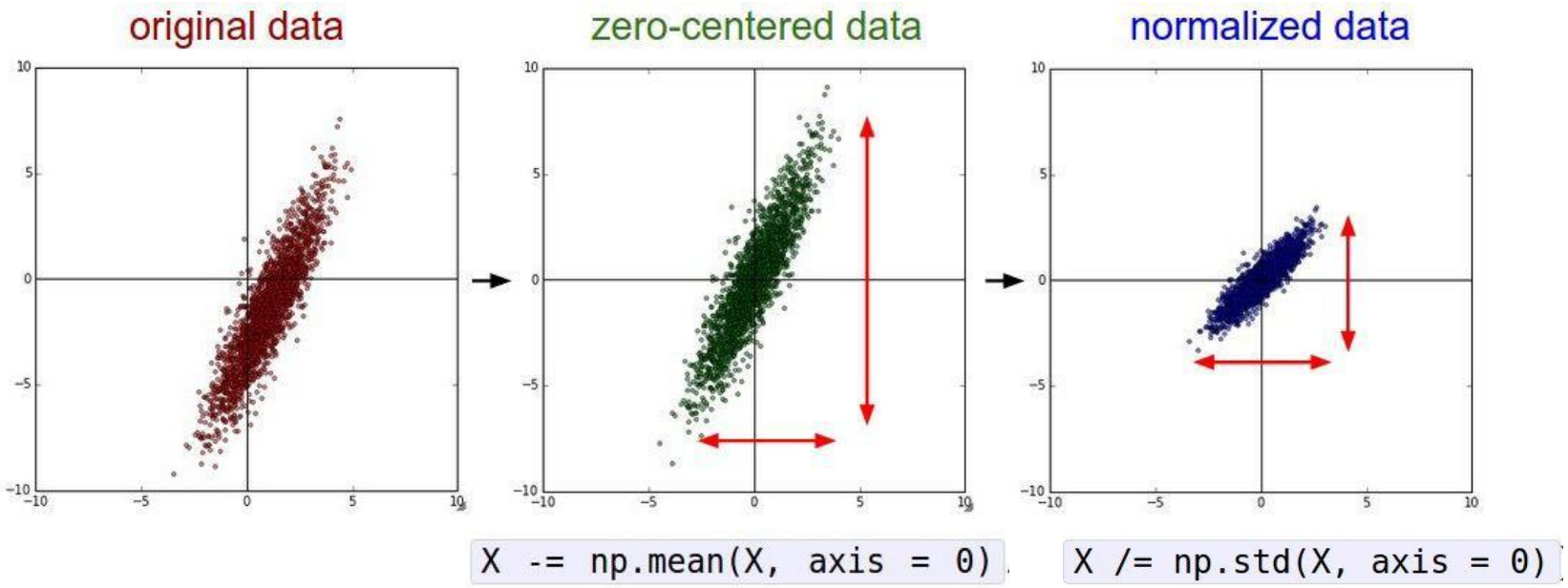
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

Data Preprocessing

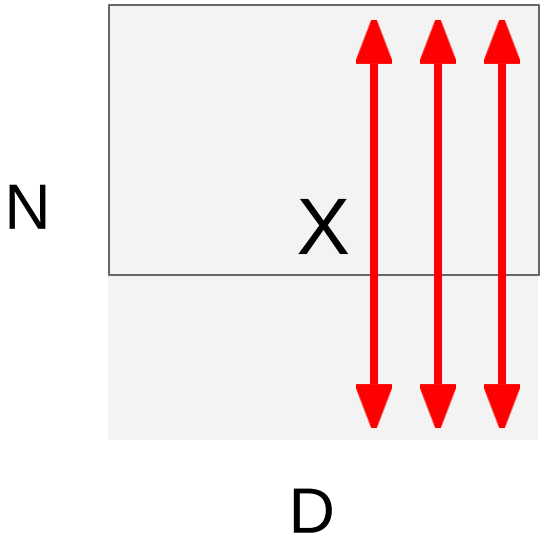


(Assume X [NxD] is data matrix, each example in a row)

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: $x : N \times D$

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

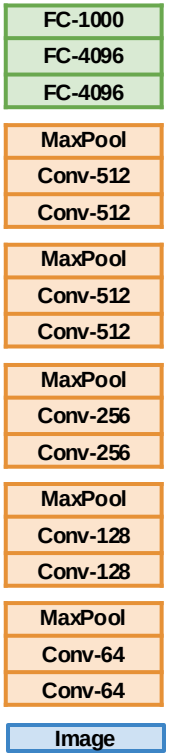
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

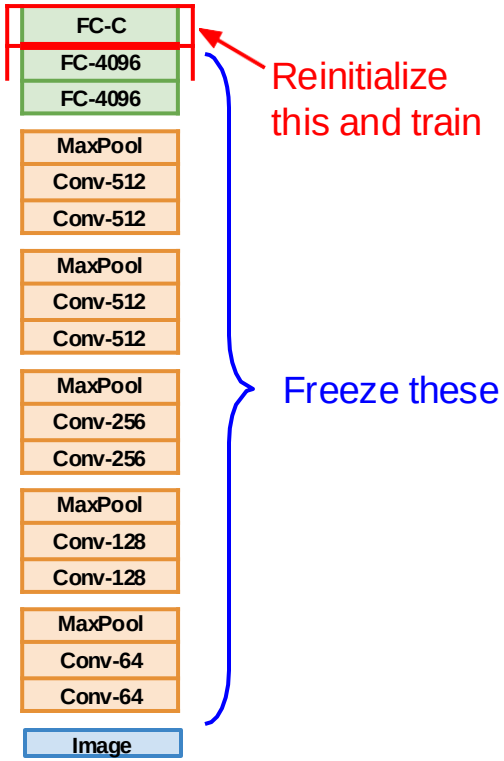
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



2. Small Dataset (C classes)



Batch Gradient Methods

- Batch or deterministic gradient methods:
  - Optimization methods that use all training samples are batch or deterministic methods

- *Somewhat confusing terminology*
  - Batch also used to describe *minibatch* used by minibatch stochastic gradient descent
  - Batch gradient descent implies use of full training set
  - Batch size refers the size of a minibatch

Stochastic or Online Methods

- Those using a single sample are called Stochastic or on-line
  - On-line typically means continually created samples drawn from a stream rather than multiple passes over a fixed size training set
- Deep learning algorithms usually use more than one but fewer than all samples
  - Methods traditionally called minibatch or minibatch stochastic now simply called stochastic

Ex: (stochastic gradient descent - SGD)

SGD Follows Gradient Estimate Downhill

Algorithm: SGD update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

A crucial parameter is the learning rate ε

At iteration k it is ε_k

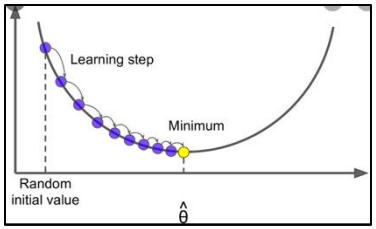
Need for Decreasing Learning Rate

- True gradient of total cost function
  - Becomes small and then 0
    - One can use a fixed learning rate
- But SGD has a source of noise
  - Random sampling of m training samples
    - Gradient does not vanish even when arriving at a minimum
  - Common to decay learning rate linearly until iteration τ: $\varepsilon_k=(1-\alpha)\varepsilon_0+\alpha\varepsilon_\tau$ with $\alpha=k/\tau$
  - After iteration τ, it is common to leave ε constant
    - Often a small positive value in the range 0.0 to 1.0

Learning Rate Decay

- Decay learning rate

  τ: $\varepsilon_k=(1-\alpha)\varepsilon_0+\alpha\varepsilon_\tau$ with $\alpha=k/\tau$



- Learning rate is calculated at each update
  - (e.g. end of each mini-batch) as follows:

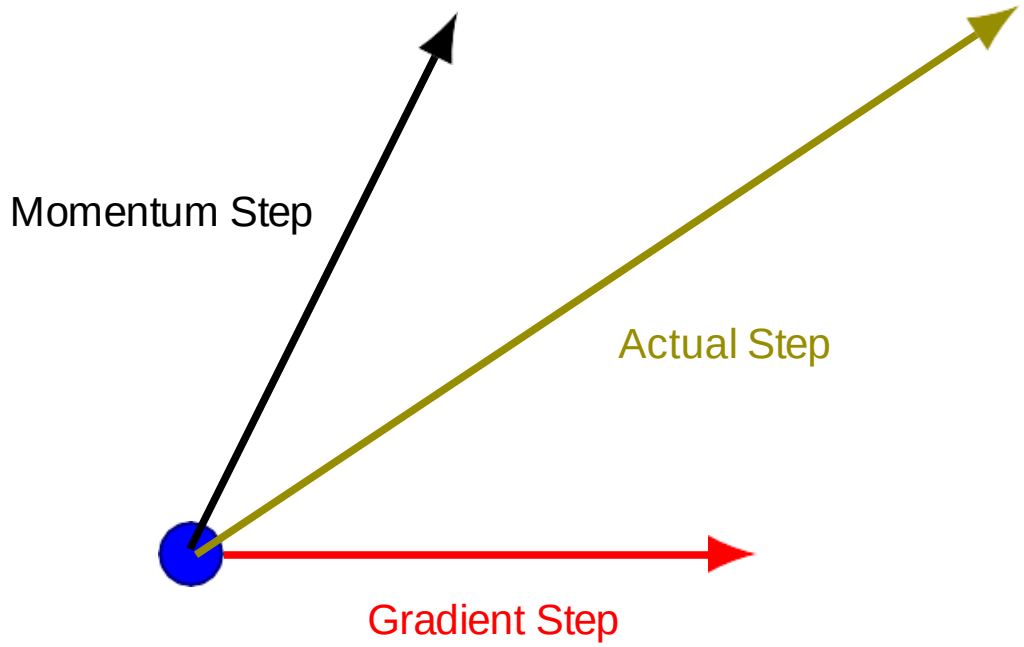
```
lrate = initial_lrate * (1 / (1 + decay * iteration))
```

  - Where *lrate* is learning rate for current epoch
  - *initial_lrate* is specified as an argument to SGD
  - *decay* is the decay rate which is greater than zero and
  - *iteration* is the current update number

```
from keras.optimizers import SGD
...
opt = SGD(lr=0.01, momentum=0.9, decay=0.01)
model.compile(..., optimizer=opt)
```

UCMERCED

Momentum



SGD Algorithm with Momentum

Algorithm: SGD with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

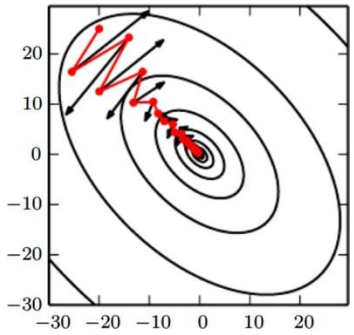
Keras: The learning rate can be specified via the *lr* argument and the momentum can be specified via the *momentum* argument.

```
1 from keras.optimizers import SGD
2 ...
3 opt = SGD(lr=0.01, momentum=0.9)
4 model.compile(..., optimizer=opt)
```

UCMERCED

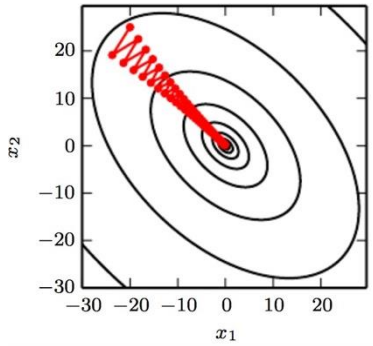
Momentum

- SGD with momentum



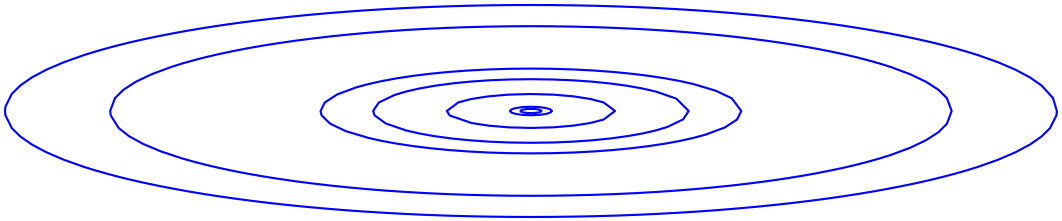
Contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix.
Red path cutting across the contours depicts path followed by momentum learning rule as it minimizes this function

- Comparison to SGD without momentum



At each step we show path that would be taken by SGD at that step
Poorly conditioned quadratic objective
Looks like a long narrow valley with steep sides
Wastes time

Motivation



Harder

AdaGrad

- Individually adapts learning rates of all parameters
  - Scale them inversely proportional to the sum of the historical squared values of the gradient
- The AdaGrad Algorithm:

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Performs well for some but not all deep learning

2D Convolution

A 2D image *f[i,j]* can be filtered by a **2D kernel** *h[u,v]* to produce an output image *g[i,j]*:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i+u, j+v]$$

This is called a **convolution** operation and written:

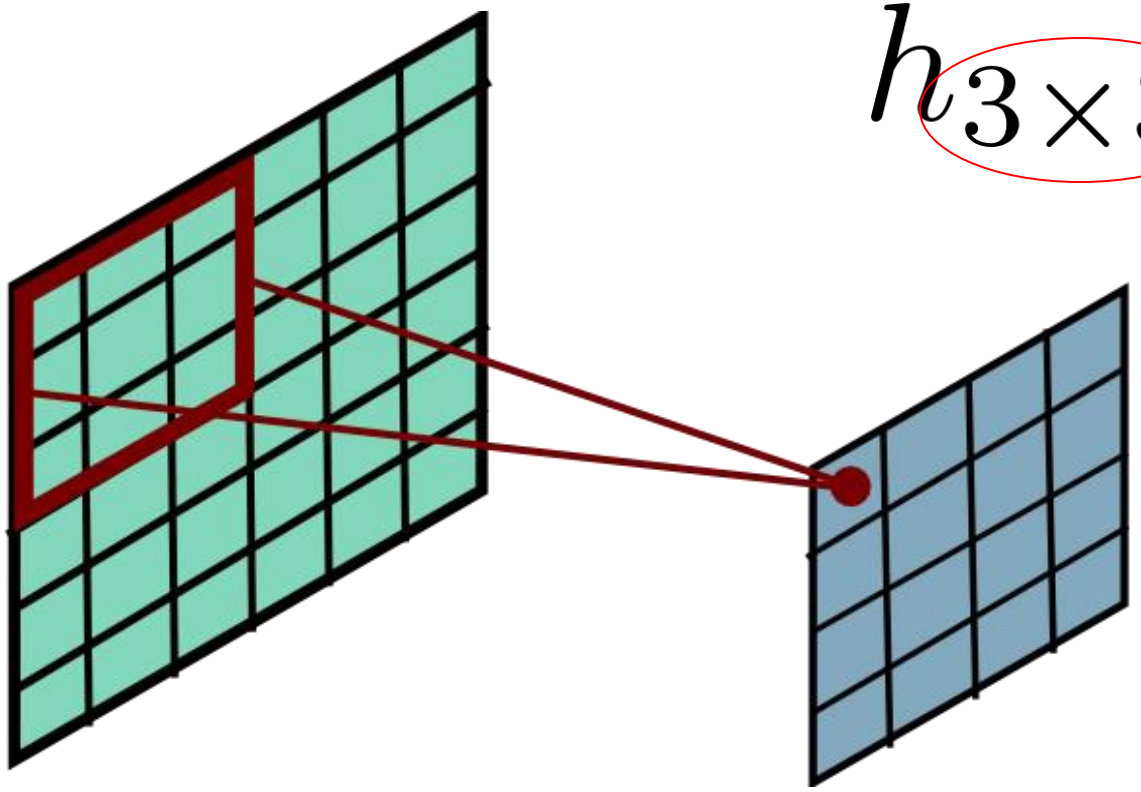
$$g = h \circ f$$

h is called "**kernel**" or "**mask**" or "**filter**" which representing a given "window function"

convolution kernel

$h_{3\times 3}$

size



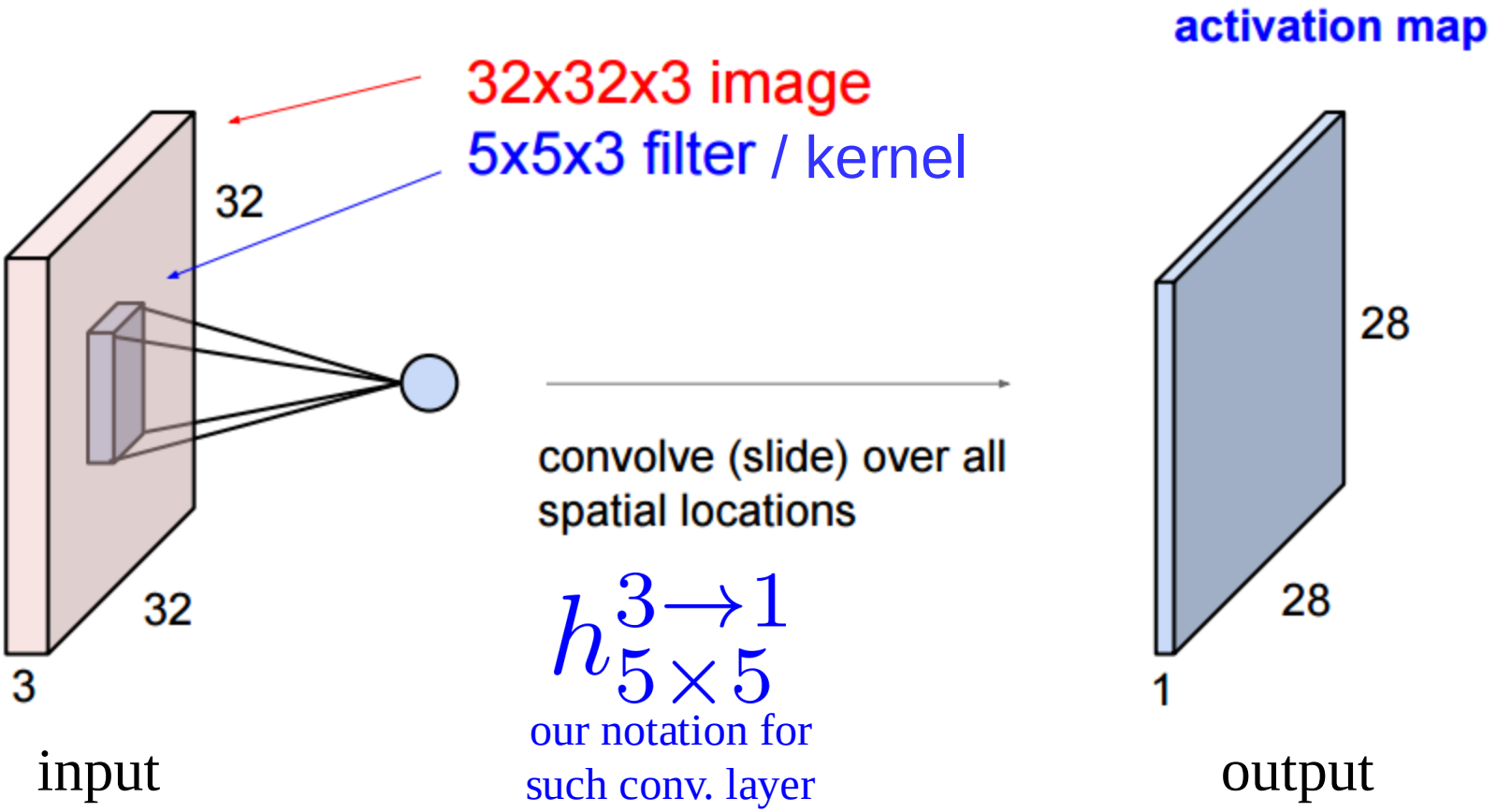
input

output

Convolutional Layer

Convolve 3D image with 3D filter

- result is a 28x28x1 activation map, no zero padding used

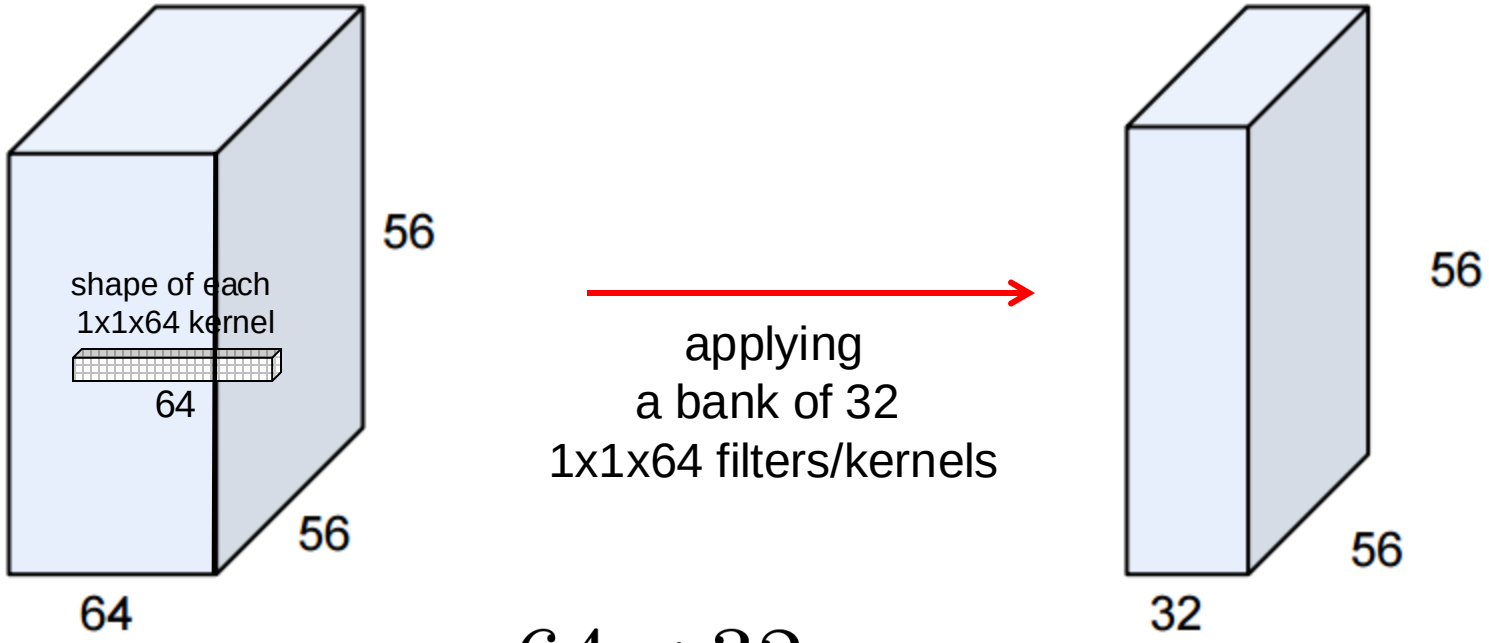


Convolutional Layer

1x1 convolutions make perfect sense

Example

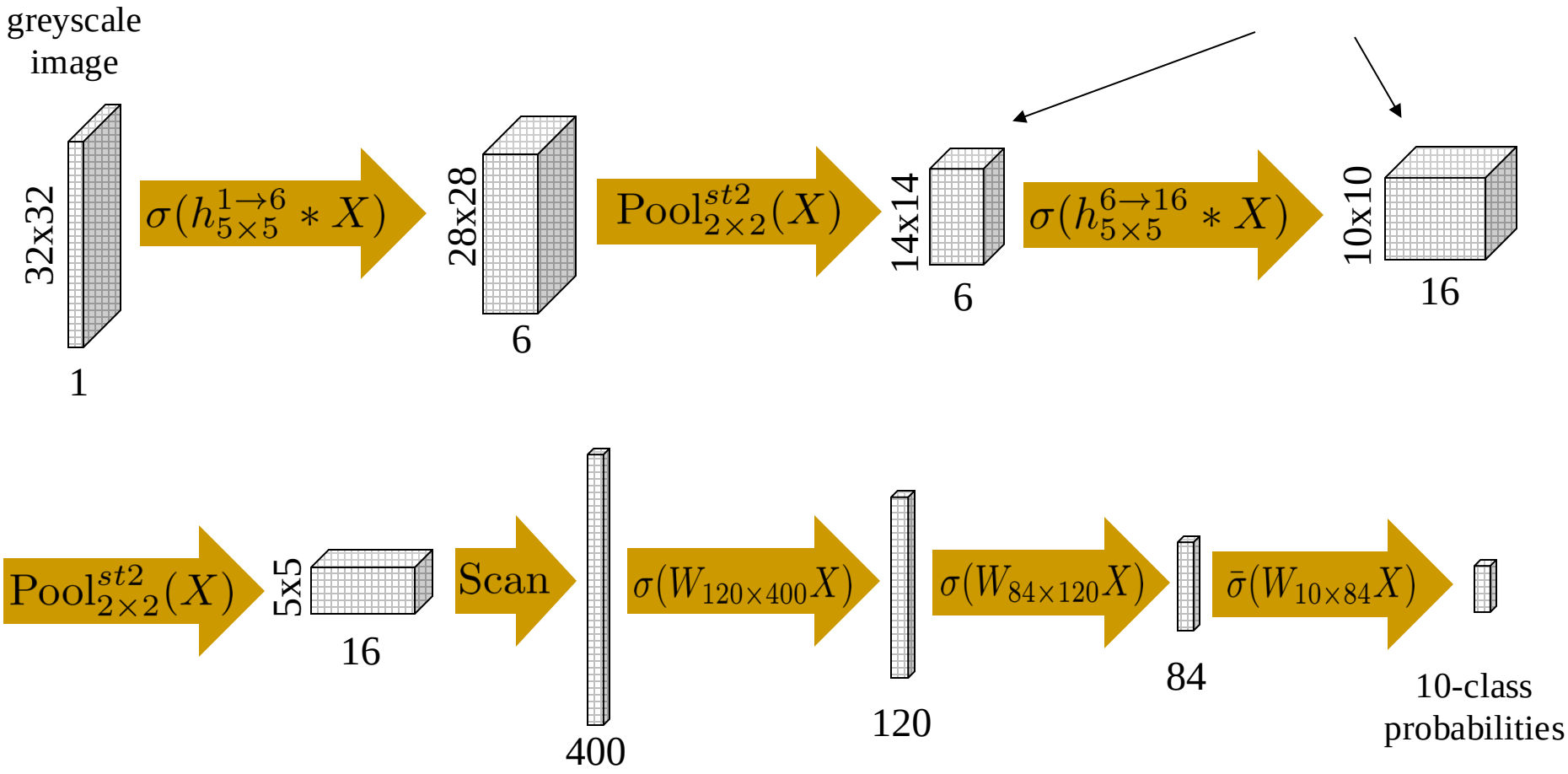
- Input image of size 56x56x64
- Convolve with 32 filters, each of size 1x1x64



$$h_{1\times 1}^{64\to 32} * X$$

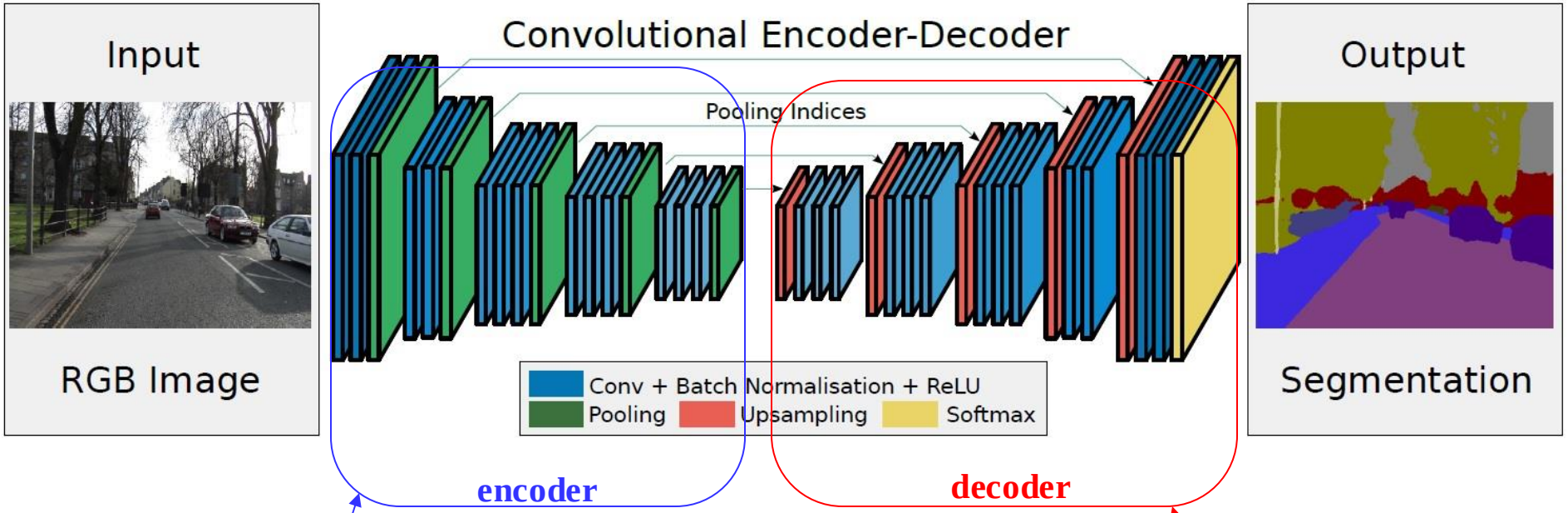
Basic CNN example (à la ***LeNet*** -1998)

NOTE: transformation of multi-dimensional arrays (**tensors**)



Common Structure: *Encoder/Decoder*

Segnet: A deep convolutional encoder-decoder architecture for image segmentation
Badrinarayanan, Kendall, Cipolla – TPAMI 2017



important:
encoder convolutional layers are typically pre-trained on *image net*

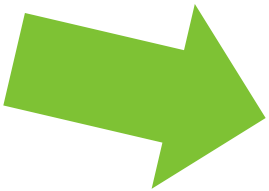
decoder upsamples encoder-generated features

Deconvolution: Example

Note: this result is equivalent to **Bilinear Interpolation**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image



Kernel

0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

kernel=3x3
stride=2
padding=1

Output Image

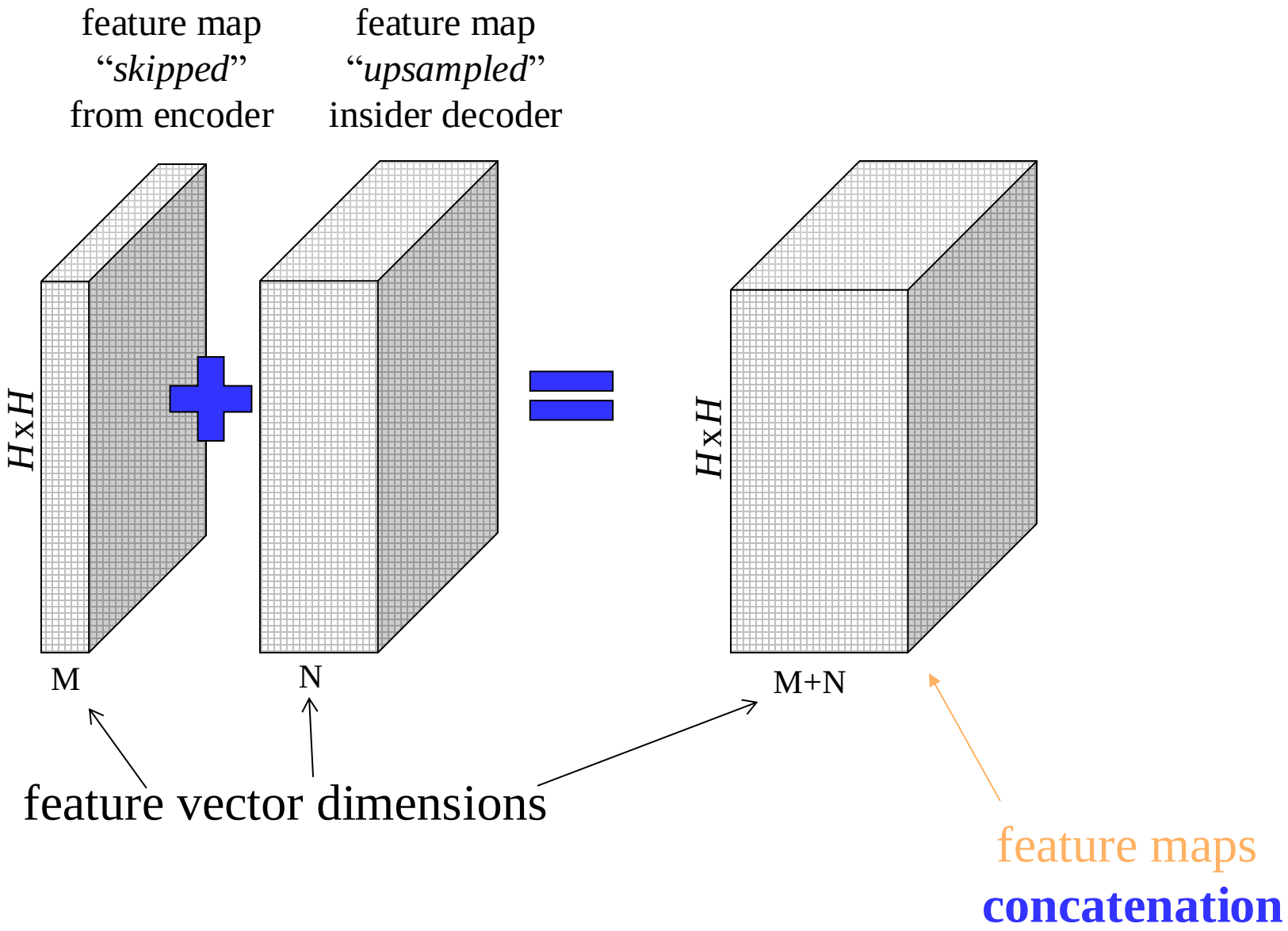
~~0~~	~~0~~	~~0.25~~	~~0.5~~	~~0.75~~	~~1~~	~~1.25~~	~~1.5~~	~~0.75~~
~~0~~	0	0.5	1	1.5	2	2.5	3	~~1.5~~
~~1~~	2	2.5	3	3.5	4	4.5	5	~~2.5~~
~~2~~	4	4.5	5	5.5	6	6.5	7	~~3.5~~
~~3~~	6	6.5	7	7.5	8	8.5	9	~~4.5~~
~~4~~	8	8.5	9	9.5	10	10.5	11	~~5.5~~
~~5~~	10	10.5	11	11.5	12	12.5	13	~~6.5~~
~~6~~	12	12.5	13	13.5	14	14.5	15	~~7.5~~
~~3~~	~~6~~	~~6.25~~	~~6.5~~	~~6.75~~	~~7~~	~~7.25~~	~~7.5~~	~~3.75~~

Bilinear Interpolation is a special case of deconvolution.

The corresponding transpose convolution kernels exists for any stride (code https://gist.github.com/mjstevens777/9d6771c45f444843f9e3dce6a401b183)

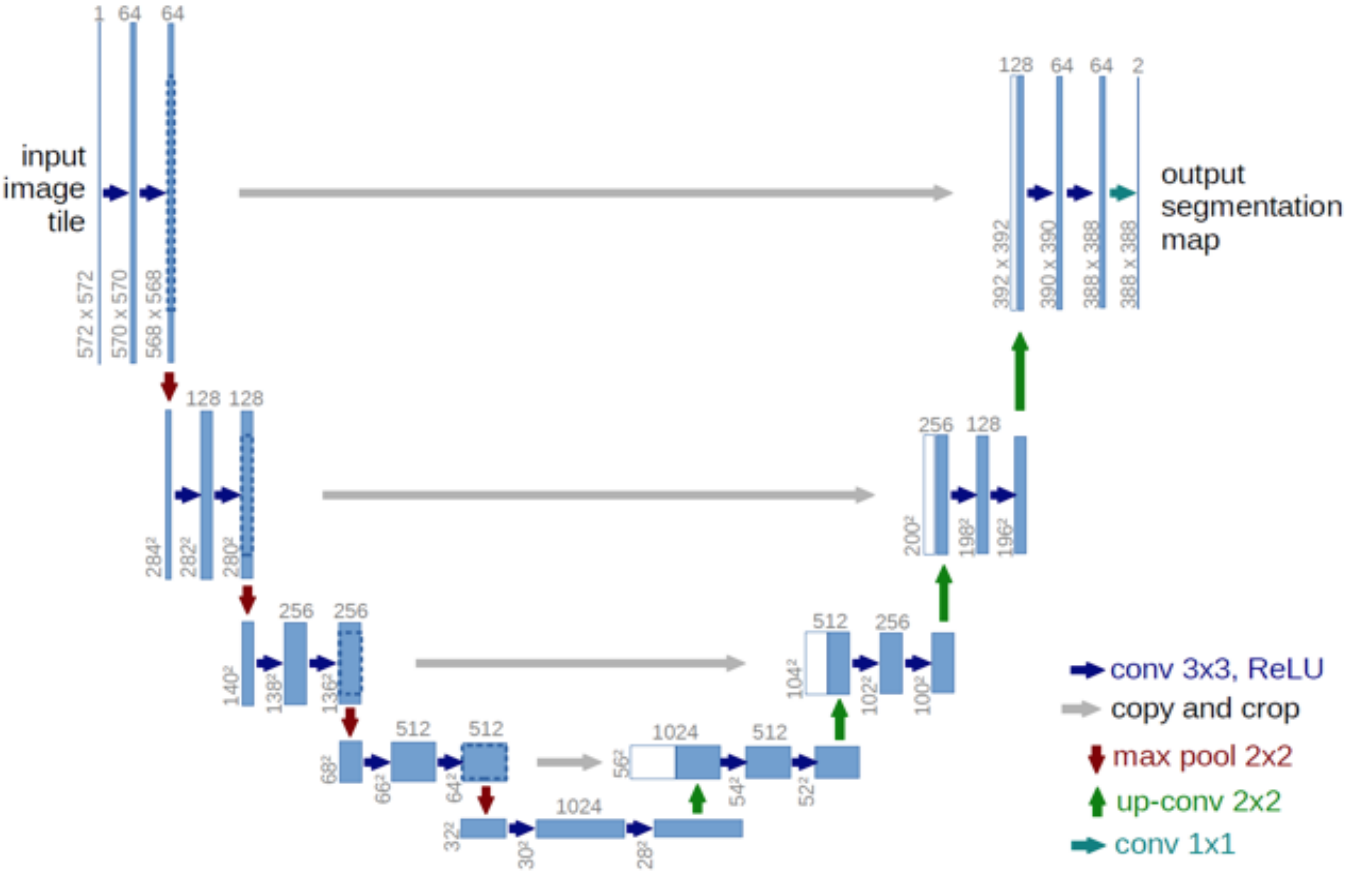
V. Dumoulin, and F. Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Skip connections: **concatenation**



U-net: expanding decoder with symmetry

and many **skip connections**



RNN and Transformer (Topic 13)

N-gram models

N-gram models *assume* each word (event) depends only on the previous n−1 words (events):

Unigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$

Bigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} | w^{(i-1)})$

Trigram model: $P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} | w^{(i-1)}, w^{(i-2)})$

Independence assumptions where the n-th event in a sequence depends only on the last n-1 events are called Markov assumptions (of order n−1).

How many parameters do n-gram models have?

Given a vocabulary V of |V| word types: so, for $|V| = 10^4$:

Unigram model: $|V|$ parameters 10^4 parameters

(one distribution $P(w^{(i)})$ with $|V|$ outcomes
[each $w \in V$ is one outcome])

Bigram model: $|V|^2$ parameters 10^8 parameters

Trigram model: $|V|^3$ parameters 10^{12} parameters

A bigram model for Alice

• Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{tired}) = 1$

$P(w^{(i)} = \textbf{of} \mid w^{(i-1)} = \textbf{use}) = 1$

$P(w^{(i)} = \textbf{sister} \mid w^{(i-1)} = \textbf{her}) = 1$

$P(w^{(i)} = \textbf{beginning} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{reading} \mid w^{(i-1)} = \textbf{was}) = 1/2$

$P(w^{(i)} = \textbf{bank} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{book} \mid w^{(i-1)} = \textbf{the}) = 1/3$

$P(w^{(i)} = \textbf{use} \mid w^{(i-1)} = \textbf{the}) = 1/3$

An n-gram model $P(w \mid w_1 \ldots w_k)$ as a feedforward net (naively)

Assumptions:

The **vocabulary** V contains V types (incl. UNK, BOS, EOS)
We want to condition each word on k preceding words

Our (naive) model:

— [Naive]
Each **input word** $w_i \in V$ is a ***V*-dimensional one-hot vector** $v(w)$
→ The **input layer** $\mathbf{x} = [v(w_1),\ldots,v(w_k)]$ has $V \times k$ elements

— We assume **one hidden layer h**

— The **output layer** is a softmax over V elements

$$P(w \mid w_1 \ldots w_k) = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$$

1D CNNs for text

Text is a (variable-length) **sequence** of words (word vectors)

[#channels = dimensionality of word vectors]

We can use a **1D CNN** to slide a window of *n* tokens across:

— Filter size n = 3, stride = 1, no padding

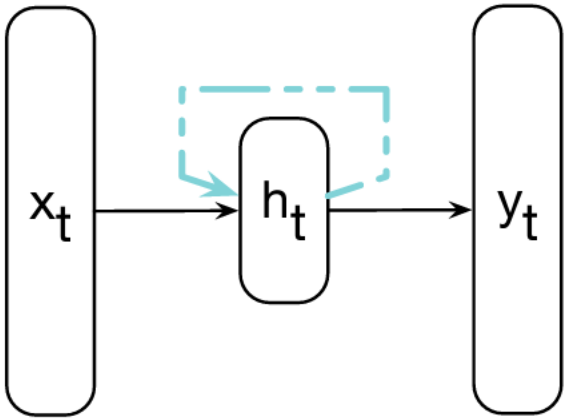
The quick brown fox jumps over the lazy dog
The **quick brown fox** jumps over the lazy dog
The quick **brown fox jumps** over the lazy dog
The quick brown **fox jumps over** the lazy dog
The quick brown fox **jumps over the** lazy dog
The quick brown fox jumps **over the lazy** dog

— Filter size n = 2, stride = 2, no padding:

The quick brown fox jumps over the lazy dog
The quick **brown fox** jumps over the lazy dog
The quick brown fox **jumps over** the lazy dog
The quick brown fox jumps over **the lazy** dog

Recurrent Neural Network

- Temporal nature in language processing
- RNN deals with sequential input data stream like language.

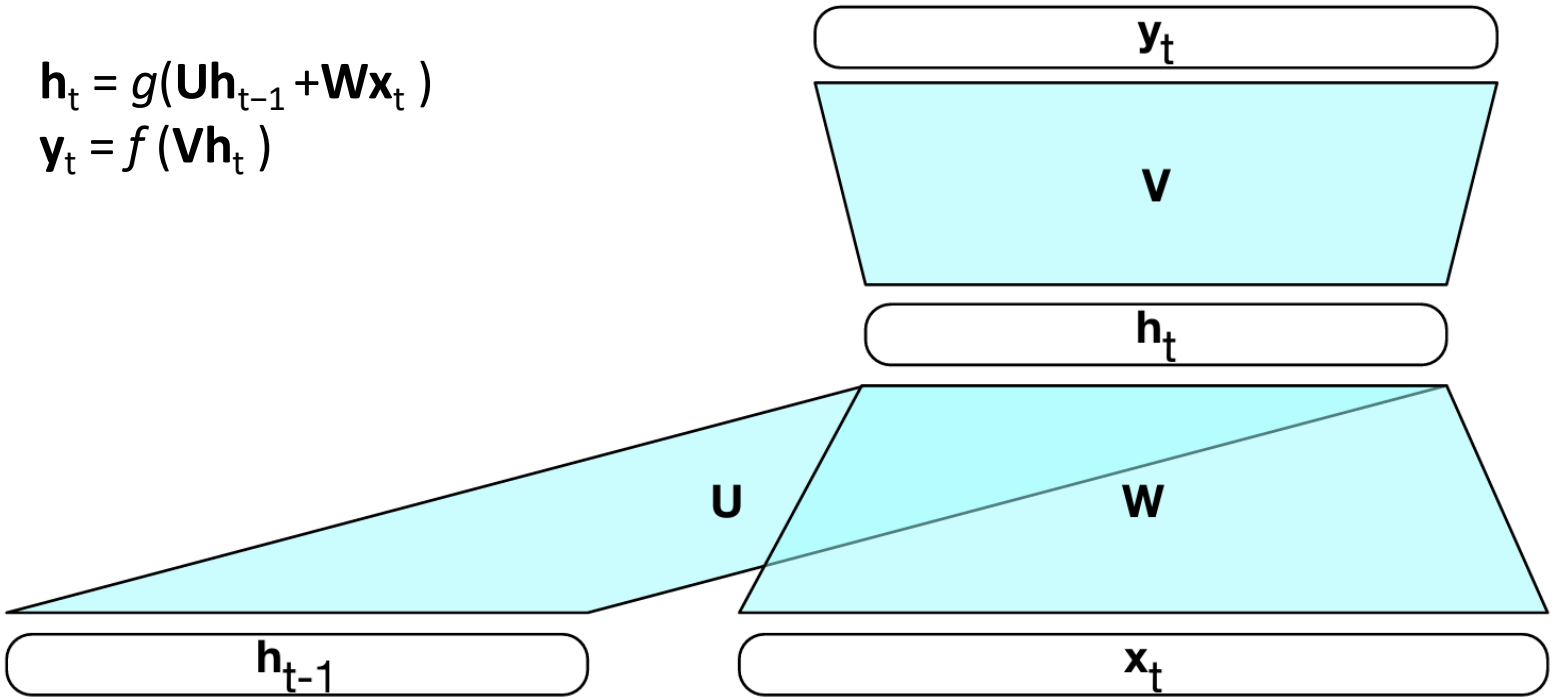


A simple RNN

A Simple Recurrent Neural Network

- RNN illustrated as a feed-forward network

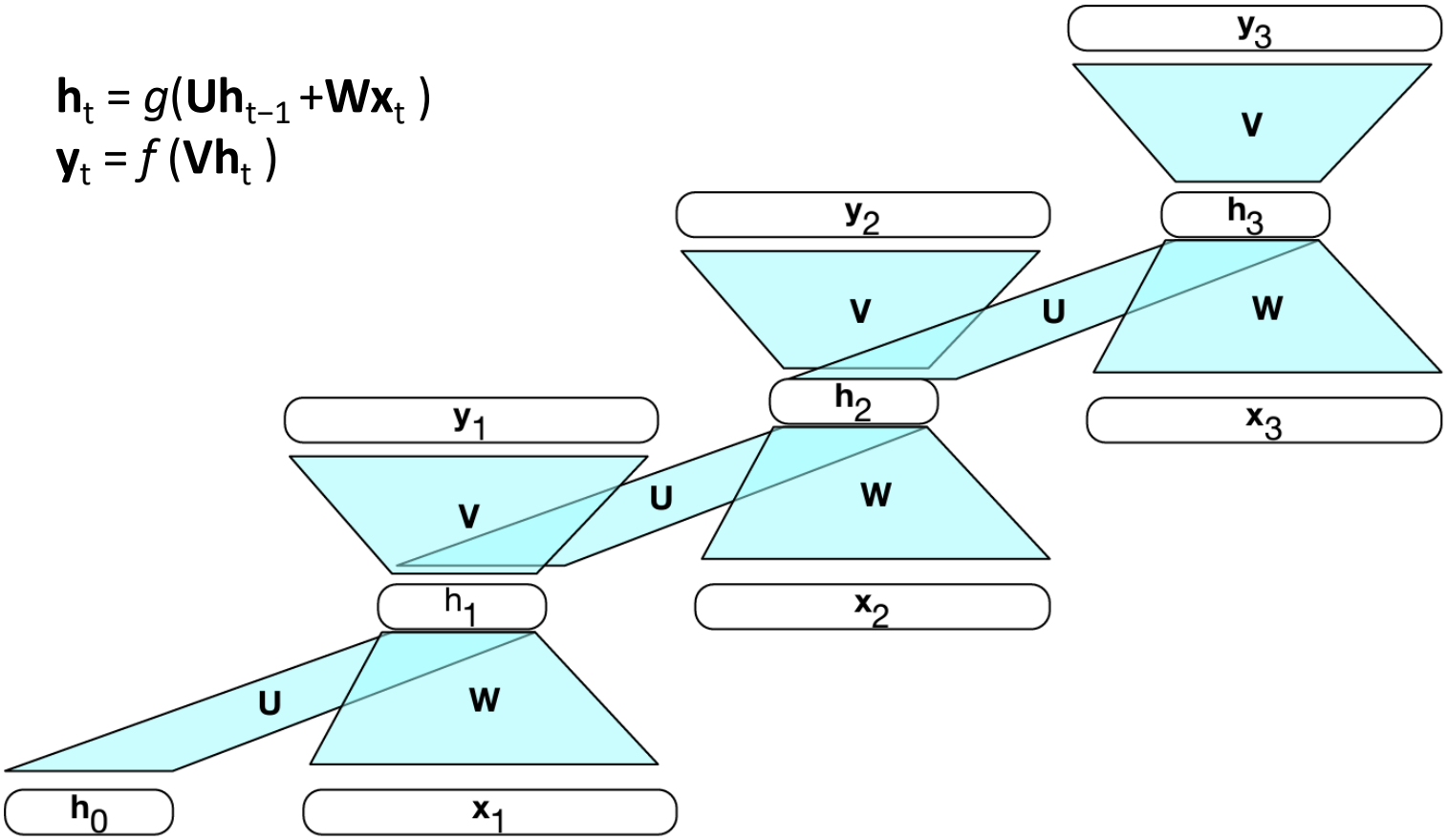
$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$



A Simple Recurrent Neural Network

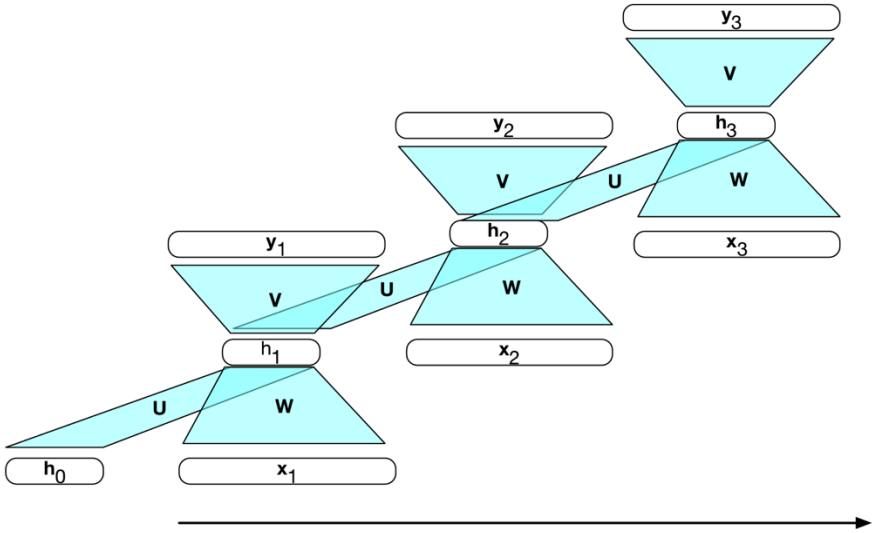
- RNN unrolled in time

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$



How to optimize Recurrent Neural Network?

$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$
$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$



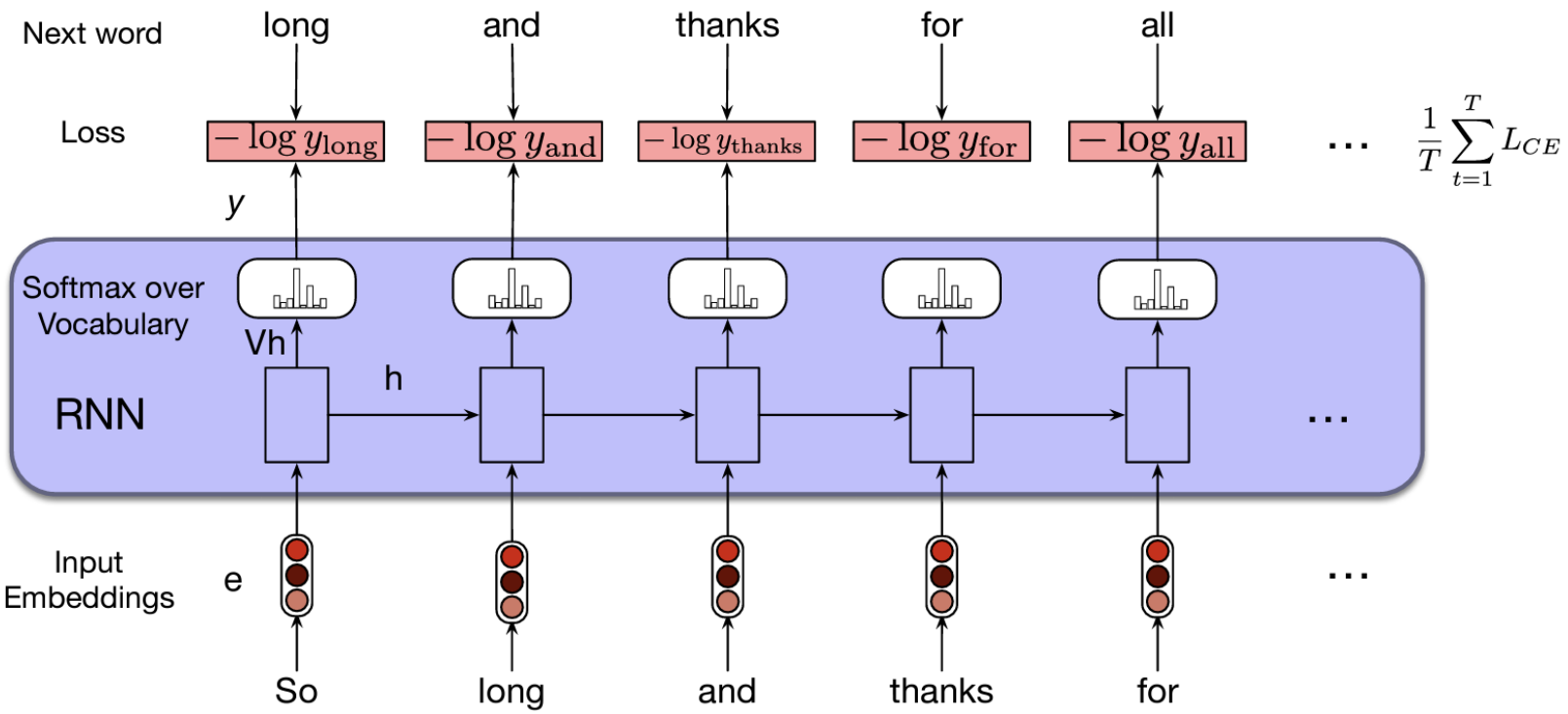
❑Backpropagation through time

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

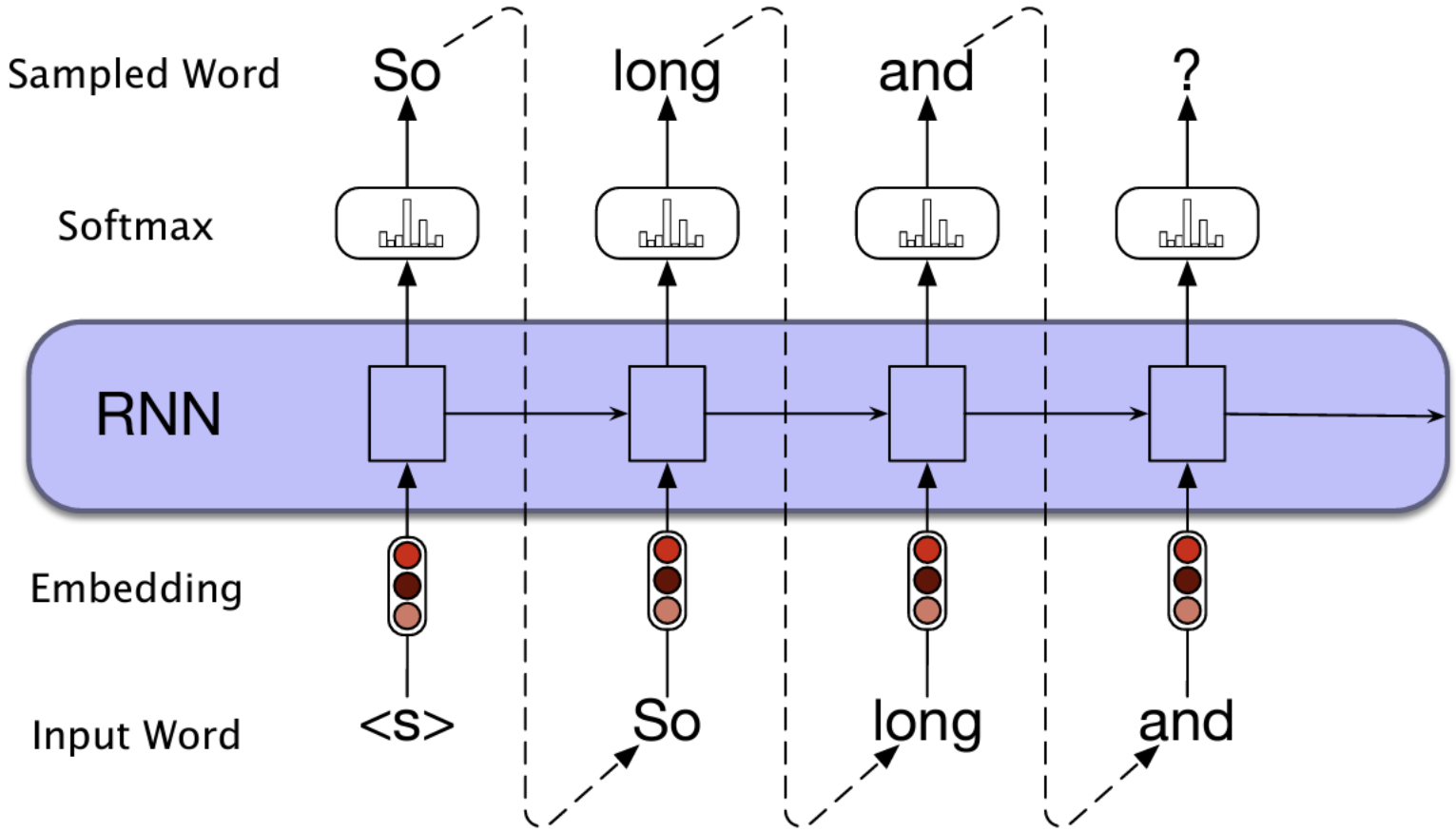
Training an RNN Language Model

- Maximum likelihood estimation



Generation with RNN Language Model

- Autoregressive (casual) generation



Vanishing/exploding gradients

- Consider the gradient of L_t at step t, with respect to the hidden state $\mathbf{h}_k$ at some previous step k (k<t):

$$\frac{\partial L_t}{\partial \mathbf{h}_k} = \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{t \geq j > k} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right)$$

- Recurrent multiplication
- Gradients too small (vanishing gradient) or too large (exploding gradient)

Exploding gradients

- What is the problem?
- We take a very large step in SGD
- Solution: Gradient clipping

Algorithm 1 Pseudo-code for norm clipping

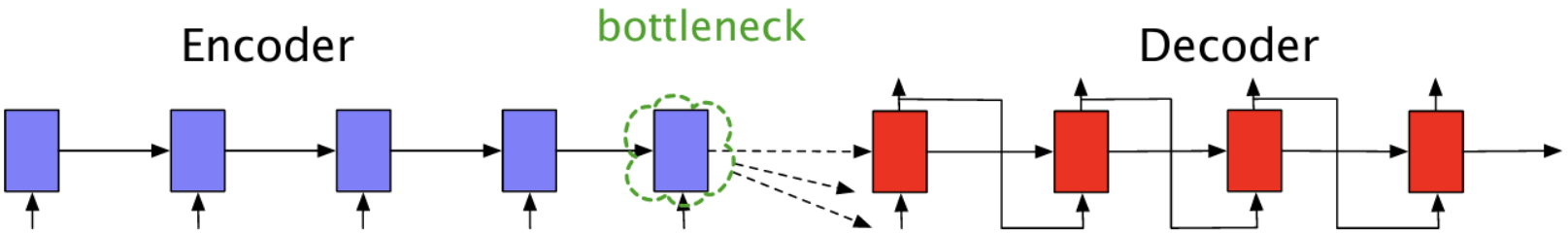
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

Vanishing gradients

- What is the problem?
- Parameters barely get updated (no learning)
- Solution:
  - LSTMs: Long short-term memory networks

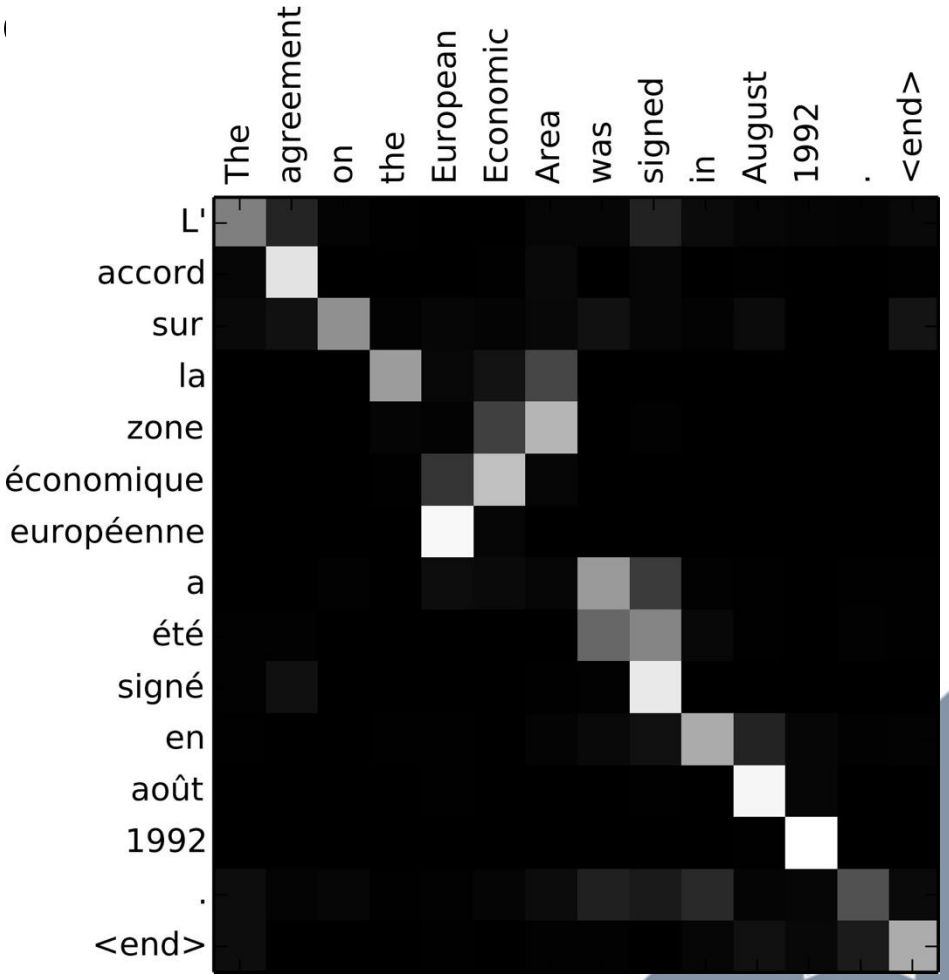
Problem of Encoder-decoder architecture

- Context vector encodes EVERYTHING about input sequence
- Context vector acts as a bottleneck



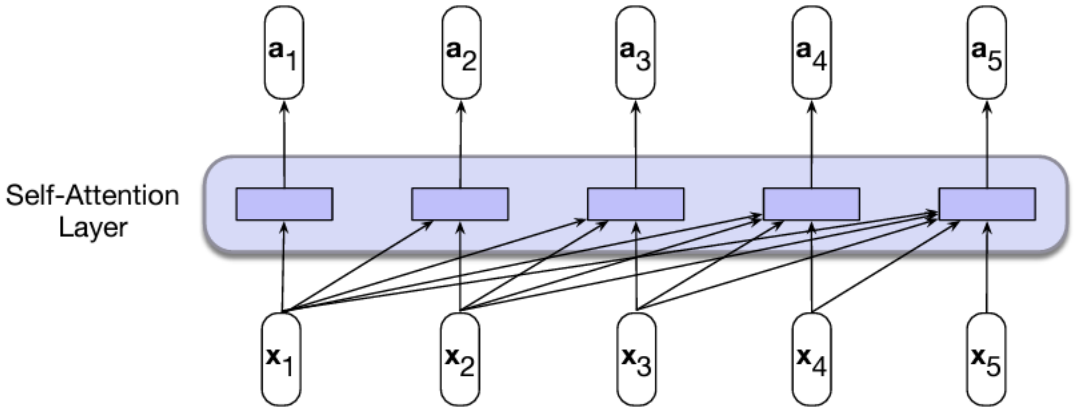
Attention weights between words

- Example: English to French translation
- Input: “The agreement on th
European Economic Area was
signed in August 1992.”
- Output: “L’accord sur la zone
économique européenne
a été signé en août 1992.”



Casual or backward-looking self-attention

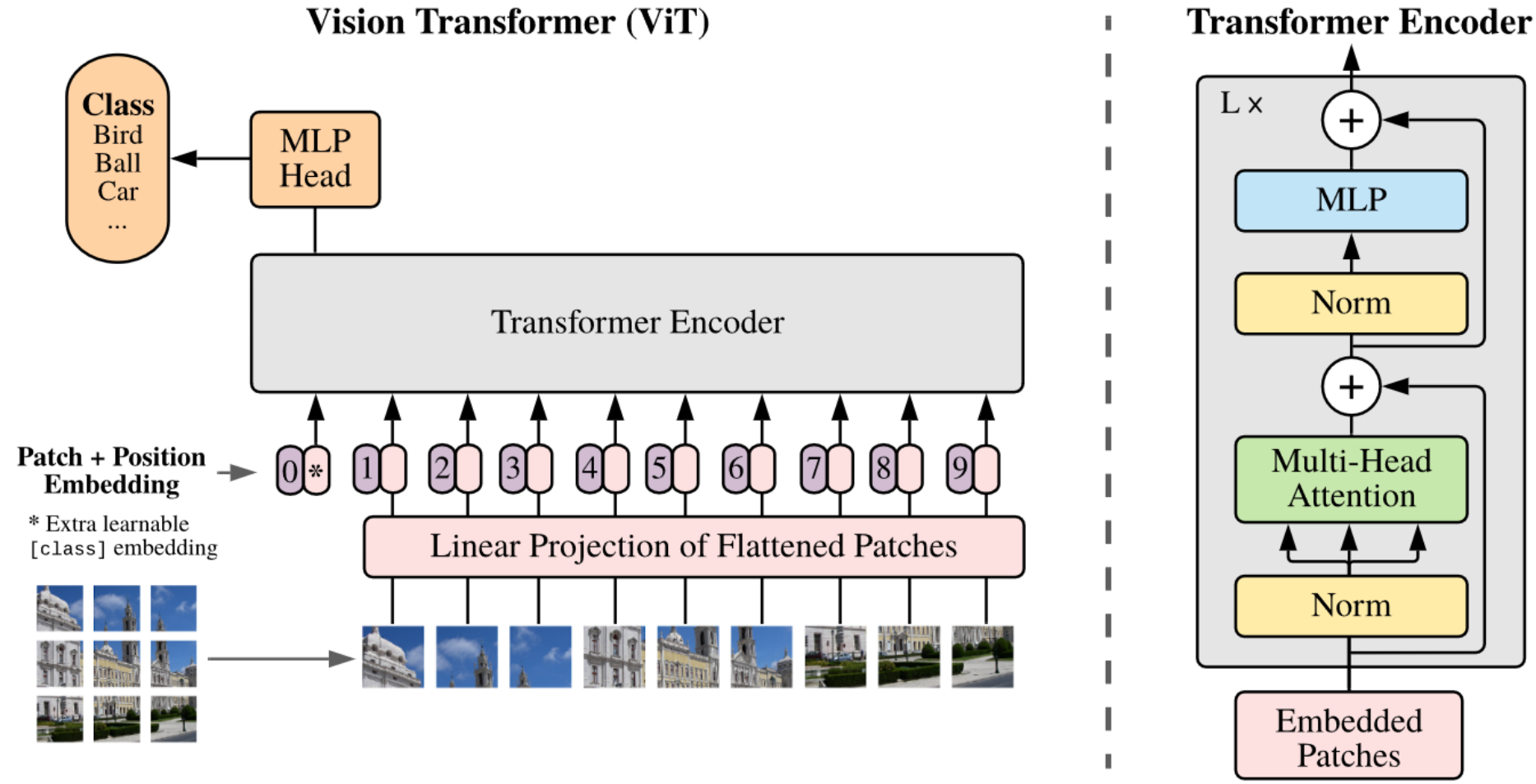
- Attends to all the inputs up to, and including, the current one



Self-attention

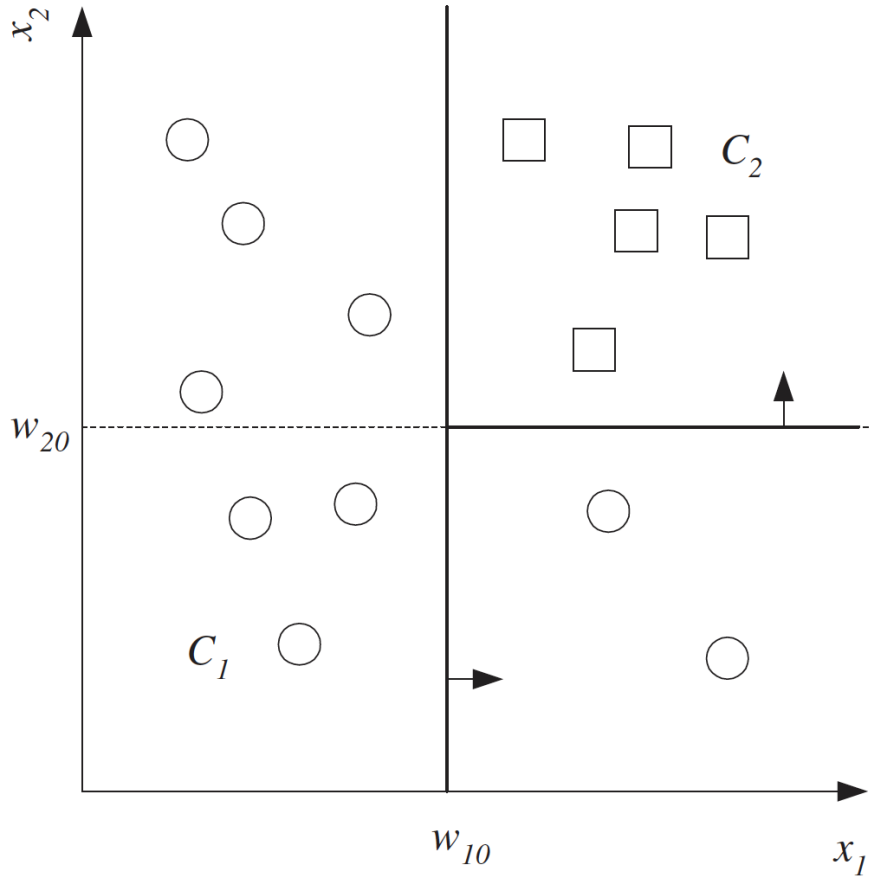
- Final Version

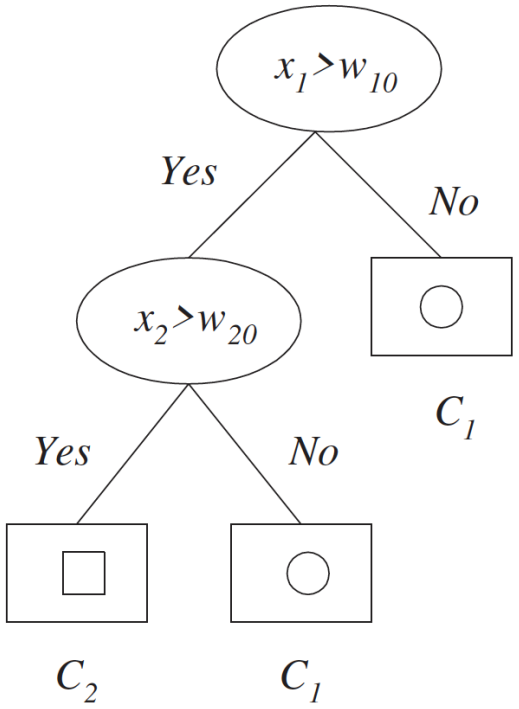
$$
\begin{aligned}
\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \mathbf{k}_i &= \mathbf{x}_i\mathbf{W}^{\mathbf{K}}; \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{V}} \\
\text{score}(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}} \\
\alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i \\
\mathbf{a}_i &= \sum_{j \leq i} \alpha_{ij}\mathbf{v}_j
\end{aligned}
$$


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP
Head
Transformer Encoder
Patch + Position
Embedding
* Extra learnable
[class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L ×
MLP
Norm
Multi-Head
Attention
Norm
Embedded
Patches

Decision Tree (Topic 14)

An example of Decision Tree



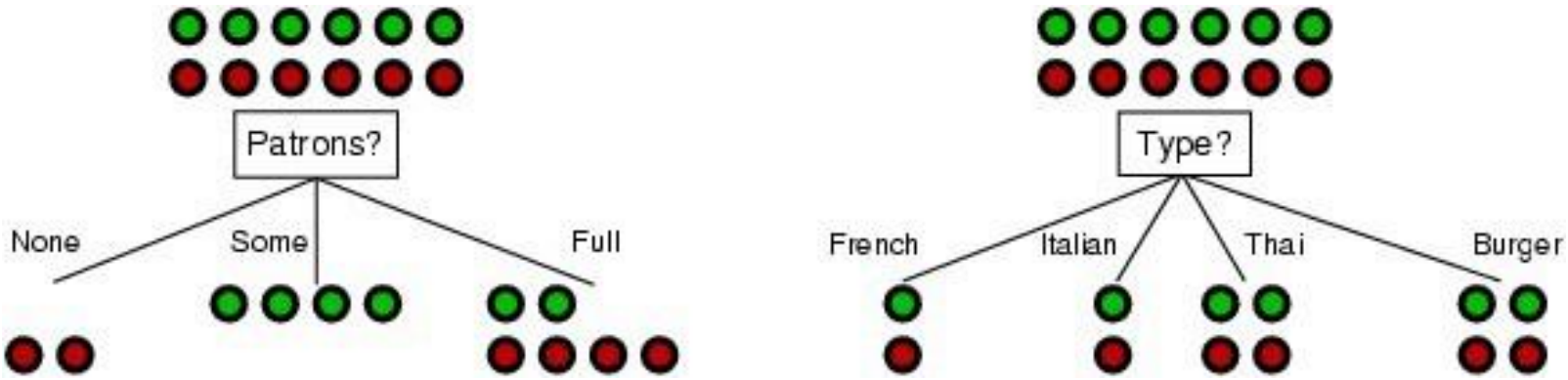


Which feature/attribute to split first?

- Probably Patron and Type

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Which feature/attribute to split first?

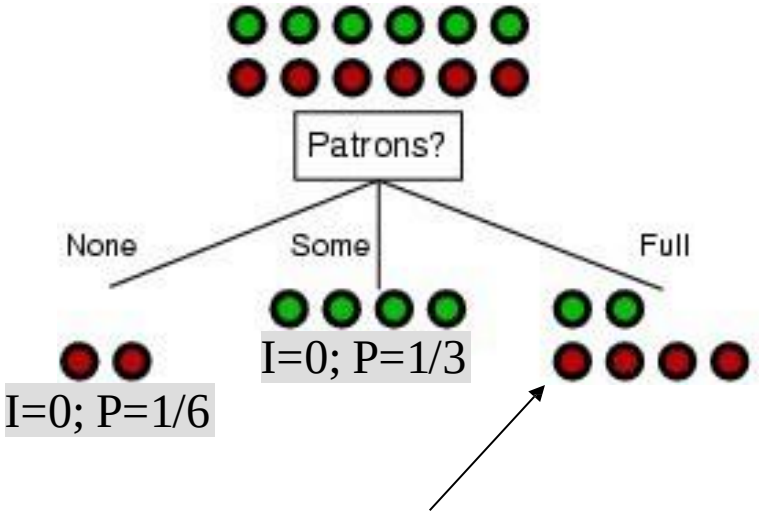


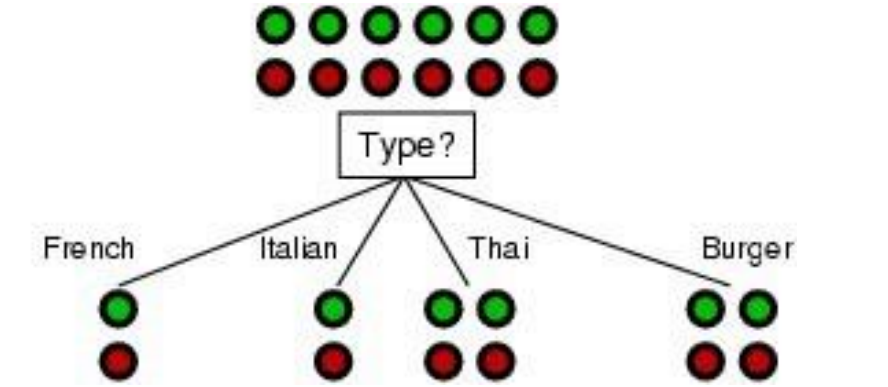
- Idea: good attribute splits examples into subsets that are (ideally) *all positive* or *all negative*

Information Gain

stay
leave

$I = -.5*\log_2(.5) - .5*\log_2(.5) = 0.5+0.5 = 1$





$I=-(1/3*\log_2(1/3)-2/3*\log_2(2/3); P=1/2$
$I*P=0.46$

Information gain = 1 - 0.46 = 0.54

$I = 6/6*1 = 1$

Information gain = 1 - 1 = 0

- **Information gain** for asking Patrons is 0.54, for asking Type is 0

UCMERCED

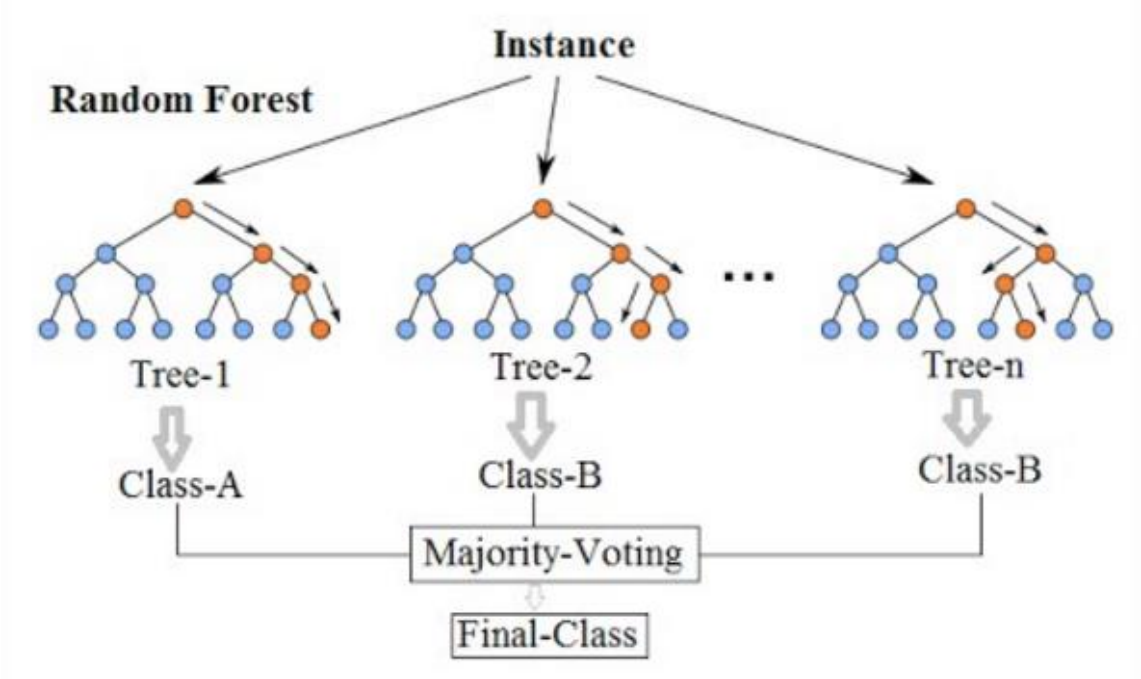
Overfitting, Early Stopping, and Pruning

- Growing the tree until each leaf is pure will produce a large tree that overfits.
- *Early stopping*: we stop splitting if the impurity is below a user threshold $\theta > 0$.
- *Pruning*: we grow the tree in full until all leaves are pure and the training error is zero. Then, we find subtrees that cause overfitting and prune them

Ensemble Model (Topic 15)

Example: Random forest

- Train an ensemble of L decision trees on L different subsets of the training set
- Define the ensemble output for a test instance as the majority vote (for classification) or the average (for regression) of the L trees



Bagging

- We generate L (partly different) subsets of the training set
- We train L learners, each on a different subset
- The ensemble output is defined as the vote or average
- Random forest: a variation of bagging

Boosting

- Weak learner: a learner that has probability of error < 1/2 (i.e., better than random guessing on binary classification).
  - Ex: decision trees with only 1 or 2 levels.
- Strong learner: a learner that can have arbitrarily small probability of error.
  - Ex: neural net
- Boosting combines many weak learners to a strong learner

Ada Boost for 2 Classes

Initialization step: for each example **x**, set

$$D(x) = \frac{1}{N}$$, where **N** is the number of examples

Iteration step (for **t** = 1…T):

1. Find best weak classifier $h_t(x)$ using weights $D(x)$
2. Compute the error rate ε_t as

$$\varepsilon_t = \sum_{i=1}^{N} D(x^i) \cdot I[y^i \neq h_t(x^i)]$$

3. compute weight α_t of classifier h_t

$$\alpha_t = \log((1-\varepsilon_t)/\varepsilon_t)$$

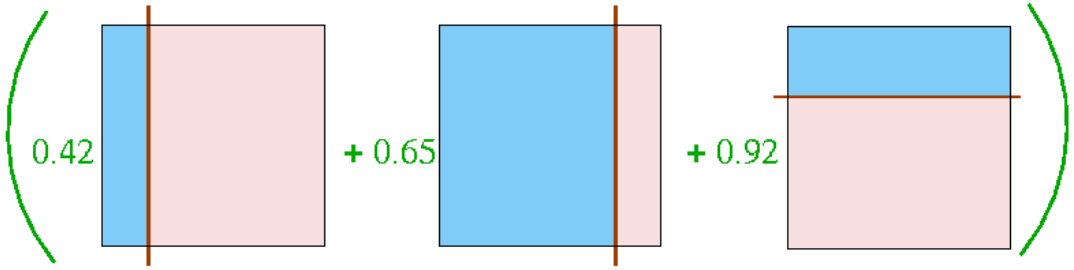
4. For each x^i, $D(x^i) = D(x^i) \cdot \exp(\alpha_t \cdot I[y^i \neq h_t(x^i)])$

5. Normalize $D(x^i)$ so that $\sum_{i=1}^{N} D(x^i) = 1$

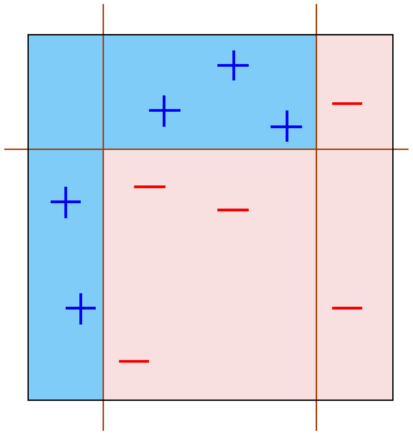
$$f_{final}(x) = \text{sign}[\sum \alpha_t h_t(x)]$$

AdaBoost Example

$f_{final}(x) =$ sign



=



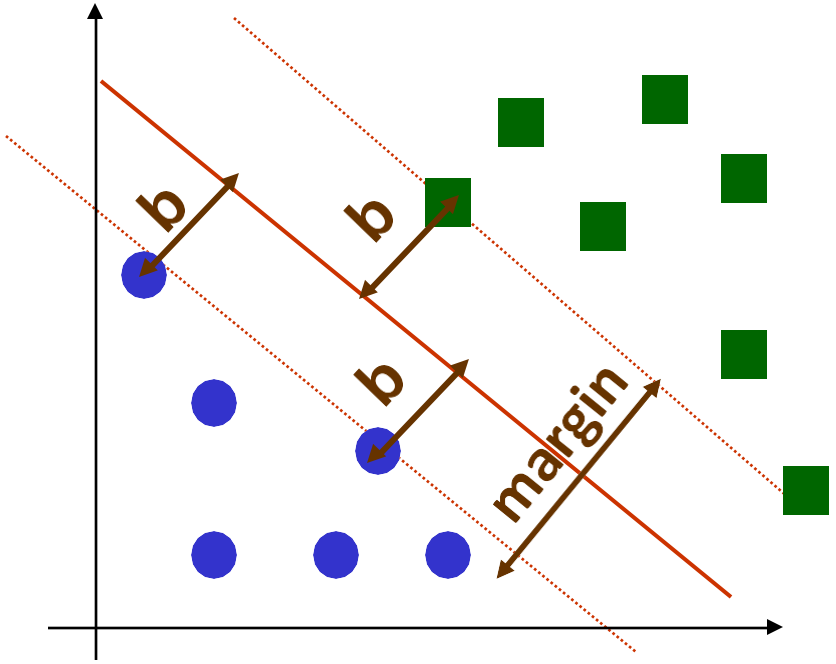
$$f_{final}(x) = \mathbf{sign}\left(0.42\,\mathbf{sign}(3 - x_1) + 0.65\,\mathbf{sign}(7 - x_1) + 0.92\,\mathbf{sign}(x_2 - 4)\right)$$

- Decision boundary non-linear

Support Vector Machine (Topic 16)

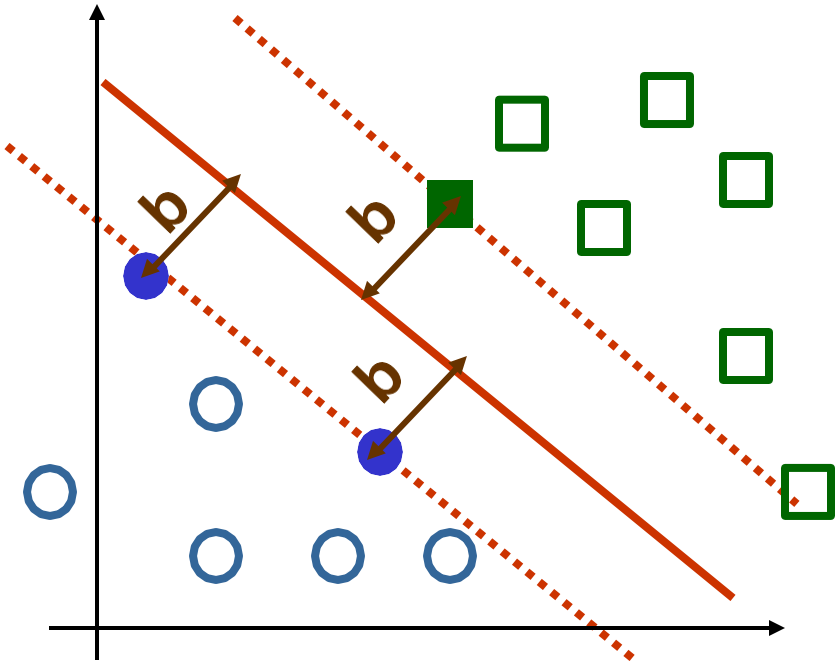
SVM: Linearly Separable Case

- SVM: maximize the ***margin***



- ***margin*** is twice the absolute value of distance ***b*** of the closest example to the separating hyperplane

SVM: Linearly Separable Case



- ***Support vectors*** are samples closest to separating hyperplane

SVM: Optimal Hyperplane

- Maximize margin $\mathbf{m} = \dfrac{2}{\|\mathbf{w}\|}$
  - subject to constraints

$$\begin{cases} \mathbf{w}^t\mathbf{x}_i + \mathbf{w}_0 \geq 1 & \textbf{if } \mathbf{x}_i \textbf{ is positive example} \\ \mathbf{w}^t\mathbf{x}_i + \mathbf{w}_0 \leq -1 & \textbf{if } \mathbf{x}_i \textbf{ is negative example} \end{cases}$$

- Let

$$\begin{cases} \mathbf{z}_i = 1 & \textbf{if } \mathbf{x}_i \textbf{ is positive example} \\ \mathbf{z}_i = -1 & \textbf{if } \mathbf{x}_i \textbf{ is negative example} \end{cases}$$

- Convert our problem to

$$\begin{aligned} &\text{minimize} && J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{constrained to} && \mathbf{z}^i\left(\mathbf{w}^t\mathbf{x}_i + \mathbf{w}_0\right) \geq 1 \quad \forall \mathbf{i} \end{aligned}$$

- $J(\mathbf{w})$ is a convex function, thus it has a single global minimum

SVM: Optimal Hyperplane

- Use Kuhn-Tucker theorem to convert our problem to:

maximize $\quad \mathbf{L_D}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \mathbf{z_i z_j x_i^t x_j}$

constrained to $\quad \alpha_i \geq 0 \quad \forall \mathbf{i} \quad \mathbf{and} \quad \sum_{i=1}^{n} \alpha_i \mathbf{z_i} = 0$

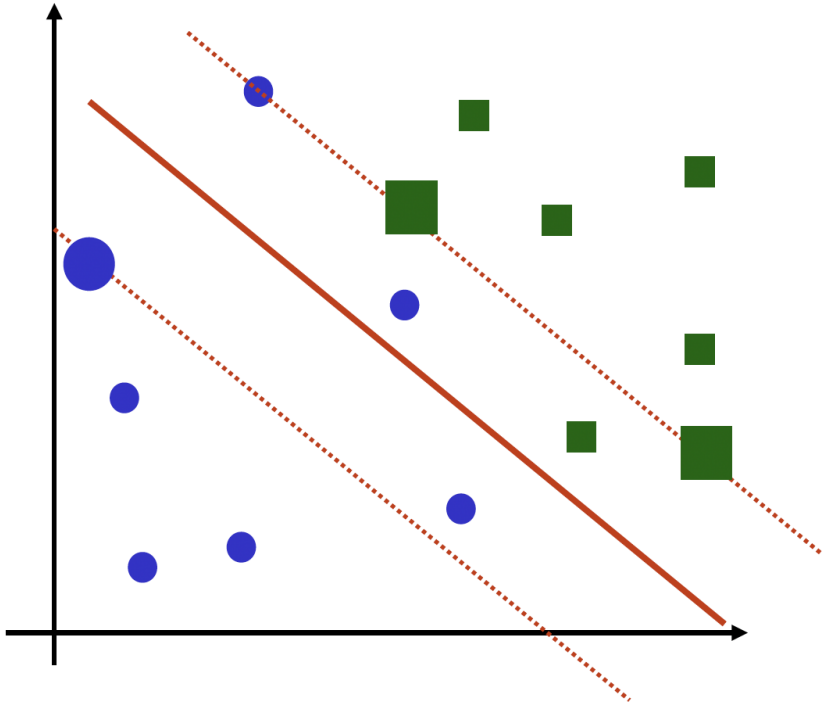
- $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_n\}$ are new variables, one for each sample
- $\mathbf{L_D}(\boldsymbol{\alpha})$ can be optimized by quadratic programming
- $\mathbf{L_D}(\boldsymbol{\alpha})$ formulated in terms of $\boldsymbol{\alpha}$
  - depends on $\mathbf{w}$ and $\mathbf{w_0}$

SVM as Unconstrained Minimization

- SVM objective can be rewritten as unconstrained optimization

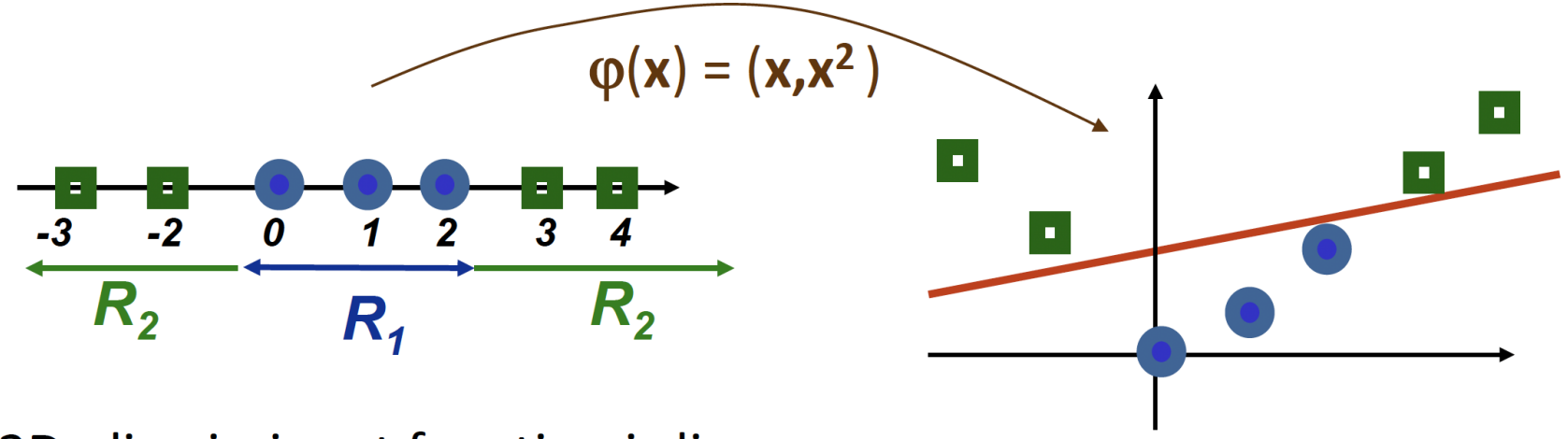
$$J(\mathbf{w}) = \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{weights regularization}} + \beta \underbrace{\sum_{i=1}^{n} \max(0, 1 - z_i f(\mathbf{x}_i))}_{\text{loss function}}$$

- $z_i f(\mathbf{x}_i) > 1$: $\mathbf{x}_i$ is on the right side of the hyperplane and outside margin, no loss
- $z_i f(\mathbf{x}_i) = 1$: $\mathbf{x}_i$ on the margin, no loss
- $z_i f(\mathbf{x}_i) < 1$: $\mathbf{x}_i$ is inside margin, or on the wrong side of the hyperplane, contributes to loss



Non Linear Mapping

- To solve a non linear problem with a linear classifier
  1. Project data ***x*** to high dimension using function $\varphi(\mathbf{x})$
  2. Find a linear discriminant function for transformed data $\varphi(\mathbf{x})$
  3. Final nonlinear discriminant function is $\mathbf{g(x)} = \mathbf{w}^t\,\varphi(\mathbf{x}) + \mathbf{w}_0$



- In 2D, discriminant function is linear

$$\mathbf{g}\left(\begin{bmatrix}\mathbf{x}^{(1)}\\ \mathbf{x}^{(2)}\end{bmatrix}\right) = \begin{bmatrix}\mathbf{w}_1 & \mathbf{w}_2\end{bmatrix}\begin{bmatrix}\mathbf{x}^{(1)}\\ \mathbf{x}^{(2)}\end{bmatrix} + \mathbf{w}_0$$

- In 1D, discriminant function is not linear $\quad \mathbf{g}(\mathbf{x}) = \mathbf{w}_1\mathbf{x} + \mathbf{w}_2\mathbf{x}^2 + \mathbf{w}_0$

Non Linear SVM

- Nonlinear discriminant function

$$g(\mathbf{x}) = \sum_{\mathbf{x}_i \in S} \alpha_i \; z_i \; K(\mathbf{x}_i, \mathbf{x})$$

$$g(\mathbf{x}) = \sum$$

most important training samples, i.e. support vectors

weight of support vector $\mathbf{x}_i$

∓ 1

similarity between $\boldsymbol{x}$ and support vector $\mathbf{x}_i$

$$K(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \mathbf{x}\|^2 \right)$$