PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V13: ROUND-UP

Özgür L. Özçep
Universität zu Lübeck
Institut für Informationssysteme

What this course was about

Differentiable Programming and
Probabilistic Programming for
Machine Learning

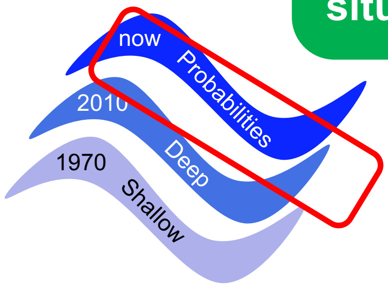
What this lecture V_{13} is about

Nearly the same as V1, but even shorter

Now you should understand this – if this was not already the case before

„ **The third wave of differentiable programming**

Getting deep systems that know when they do not know and, hence, recognise new situations and adapt to them



" 1)

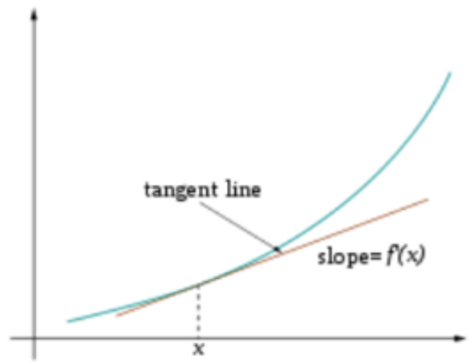
Kristian Kersting - Sum-Product Networks: The Third Wave of Differentiable Programming

Gradient Descent

V2

- Total loss

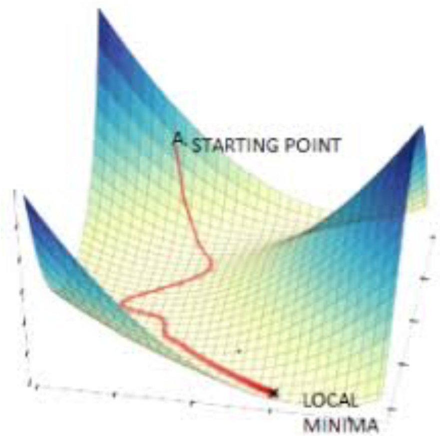
$$L = -\sum_{(x,y)\in D} l(g(x,\theta),y)$$



for some loss function l, dataset D

and model g with parameters θ

- Define how many passes (epochs) over the data to make
- learning rate η
- Gradient Descent: update θ by gradient in each epoch $\theta \leftarrow \theta - \eta \nabla_\theta L$



Backprop: efficient implementation of gradient descent

V2

Sample labeled data (**batch**) → **Forward** it through the network, get predictions → **Back-propagate** the errors → **Update** the network weights →

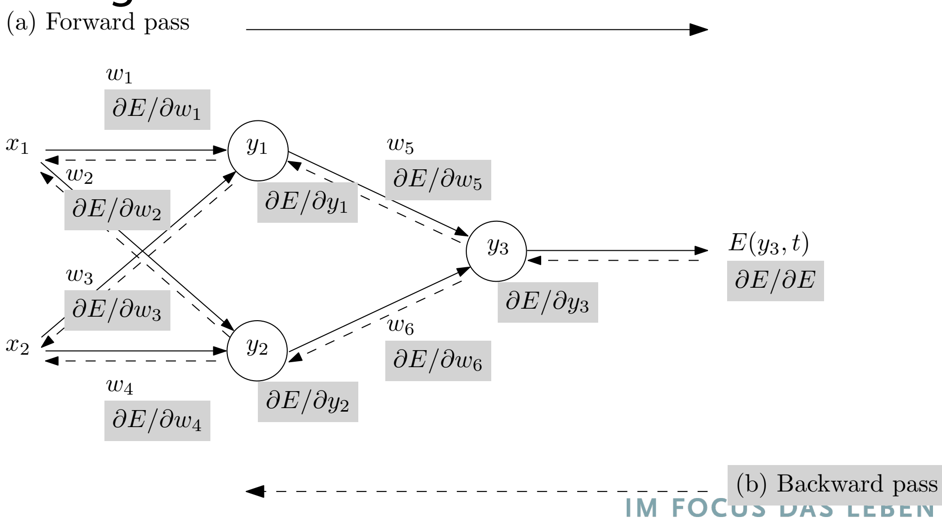
Backpropagation idea

- Generate **error signal** that measures difference between predictions and target values
- Use error signal to change the weights and get more accurate predictions backwards
- Underlying mathematics: chain rule

Chain rule (1-dim)

$$\frac{dh}{dx} = \frac{df}{dg}\frac{dg}{dx}$$

(for $h(x) = f(g(x))$)



V3

V4

V6

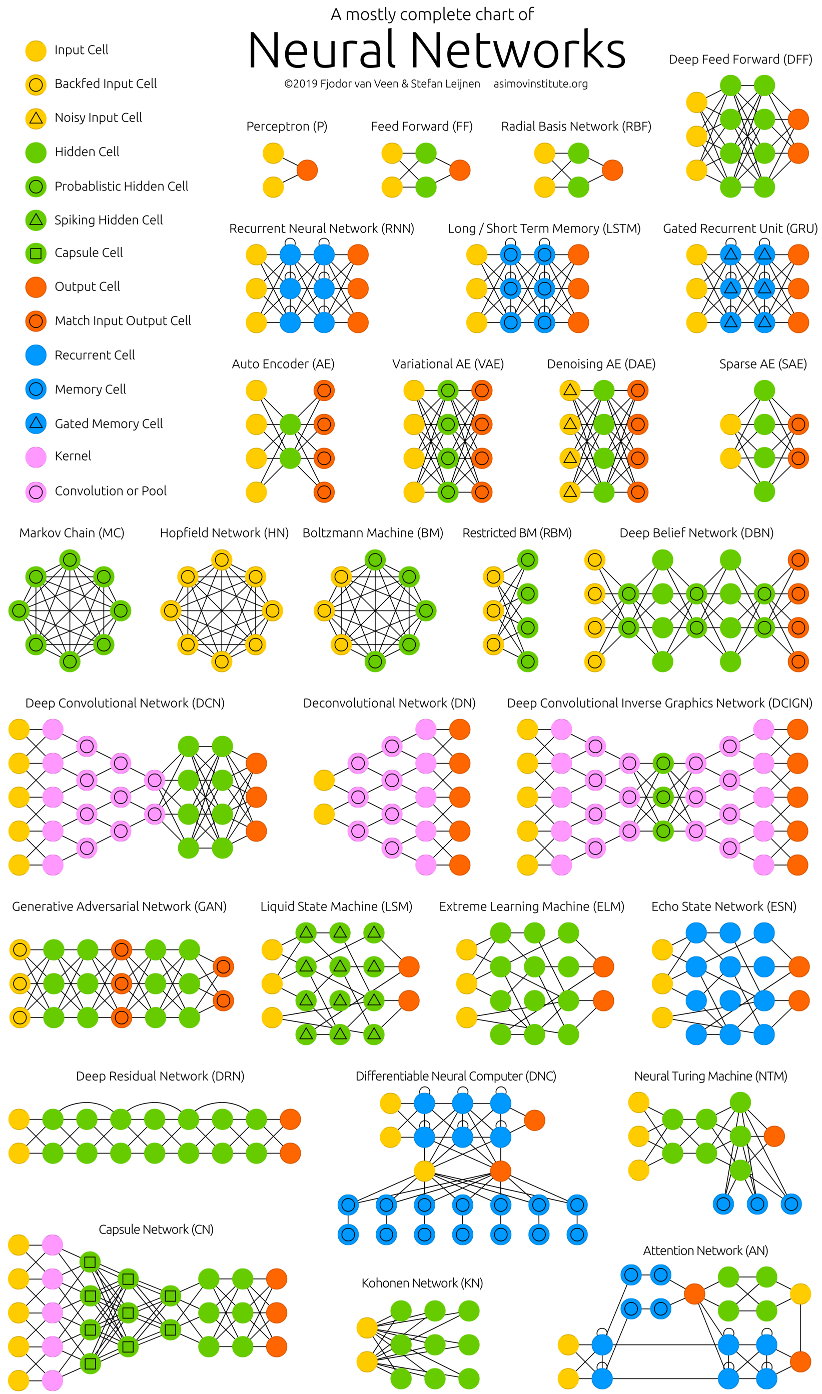
(V8)

V3

V6

(V3)

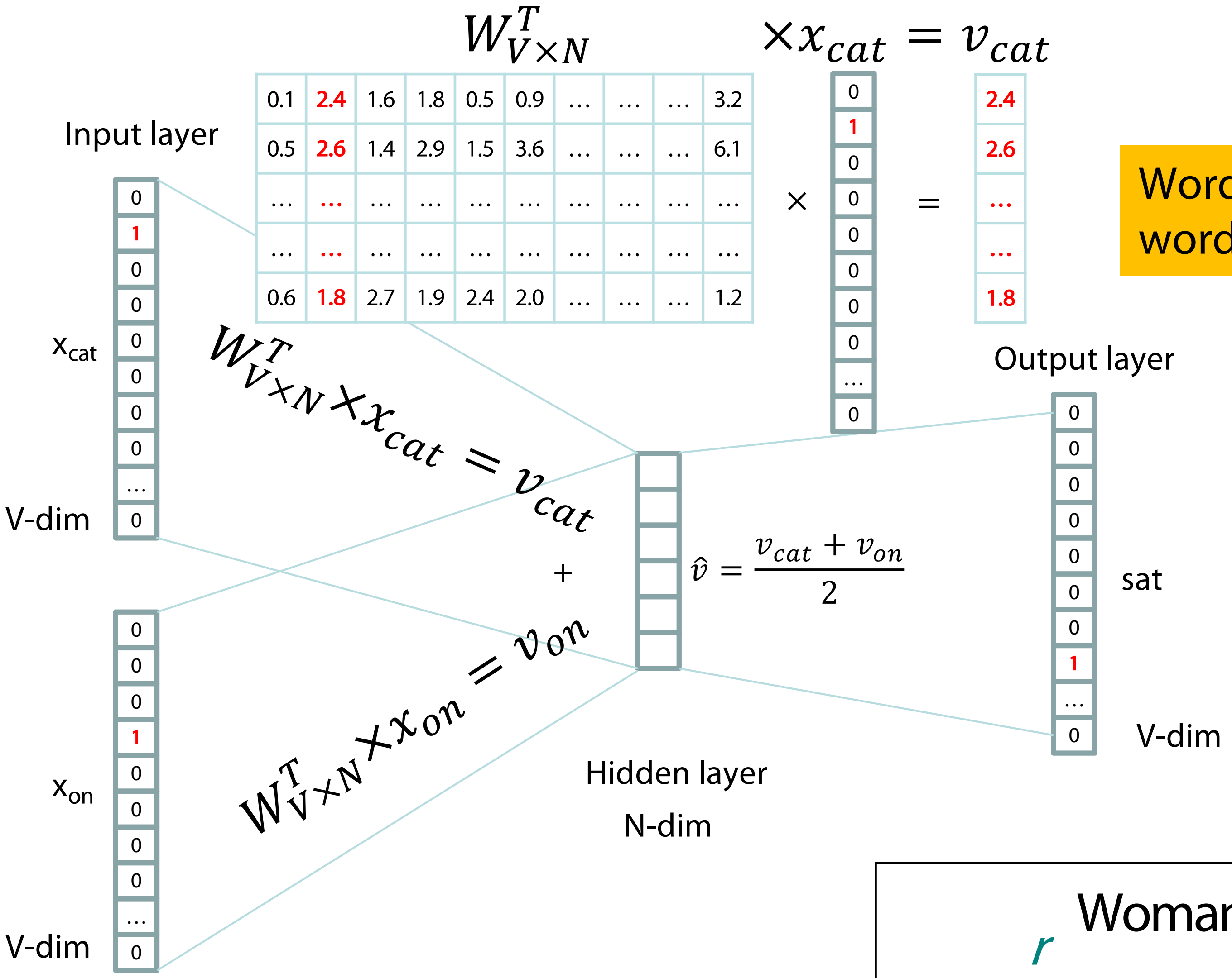
V4



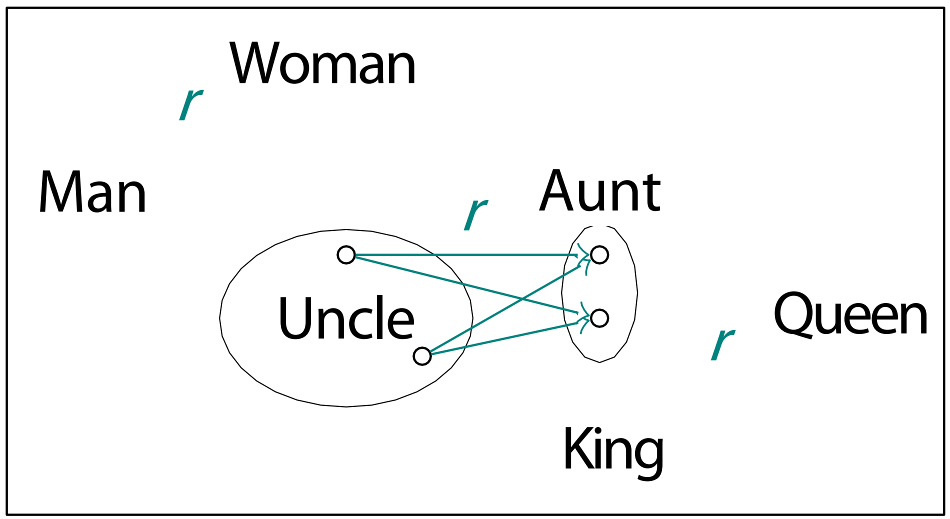
https://www.asimovinstitute.org/neural-network-zoo/

V6

Word embedding a la word2vec



Knowledge graph embedding



Automatic Differentiation (AD)

V7

- AD is a mix of
  - symbolic differentiation (SD) (rules s.a. chain rule, product rule)
  - numerical differentiation (ND): use $\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}$

$$\frac{d(f(x)\cdot g(x))}{dx} = \frac{d\,f(x)}{dx} g(x) + \frac{d\,g(x)}{dx} f(x) \quad \text{(Product rule)}$$

- $h(x) := \; g(x) \cdot f(x)$
- $\frac{dh(x)}{dx}$ and h have two components in common
- This may also be the case for f.
- Symbollicaly calculating f won‘t profit from common parts of f and $\frac{df(x)}{dx}$

$l_1 = x$
$l_{n+1} = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

```
f(x):
   v = x
   for i = 1 to 3
     v = 4*v*(1 - v)
   return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

Coding

Symbolic Differentiation of the Closed-form

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation

```
f'(x):
   (v,dv) = (x,1)
   for i = 1 to 3
     (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
   return (v,dv)
```

f'(x_0) $= f'(x_0)$
Exact

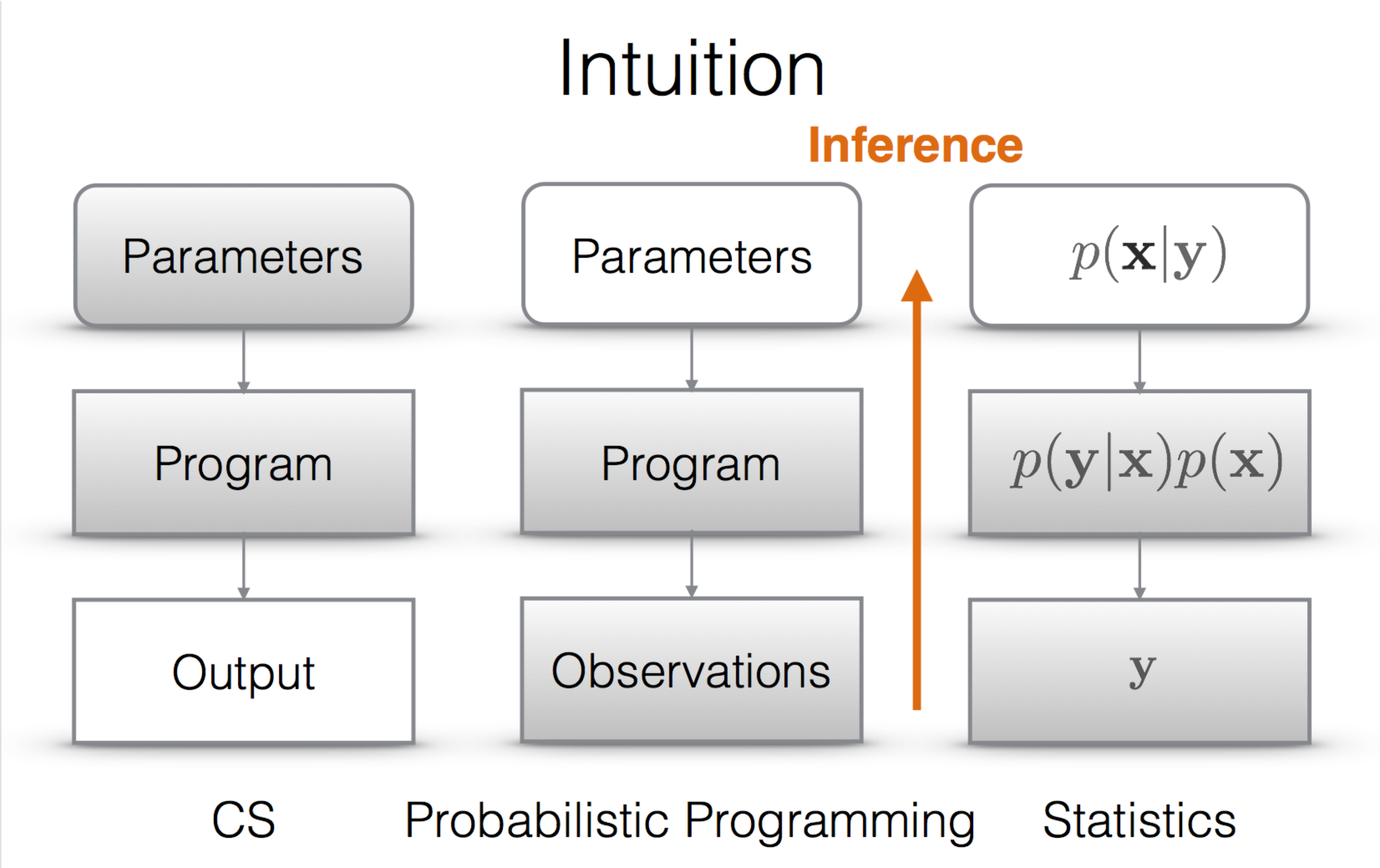
Numerical Differentiation

```
f'(x):
   h = 0.000001
   return (f(x + h) - f(x)) / h
```

f'(x_0) $\approx f'(x_0)$
Approximate

Comparison

V8/9



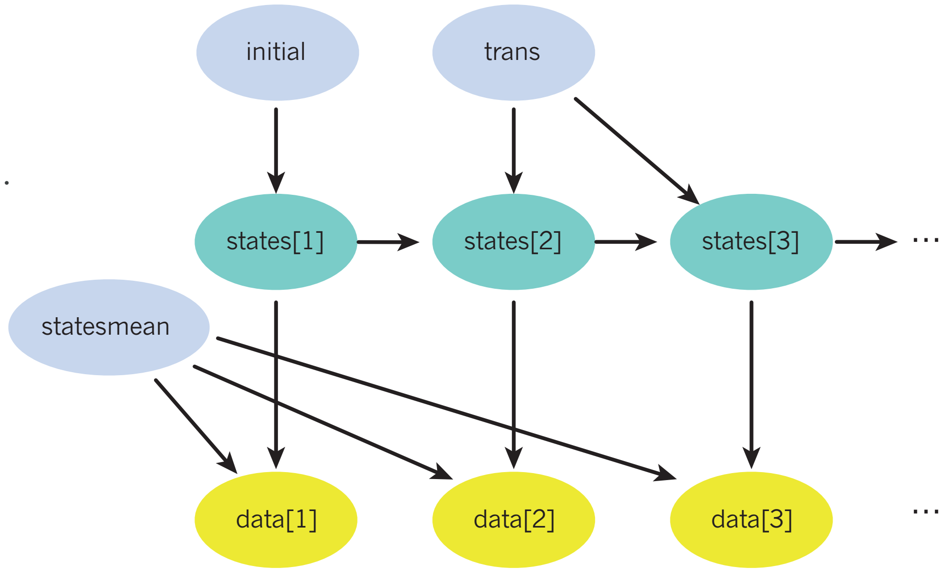
Ex: F. Wood: Probabilistic Programming, PPAML Summer School, Portland 2016

Probabilistic Programming Example

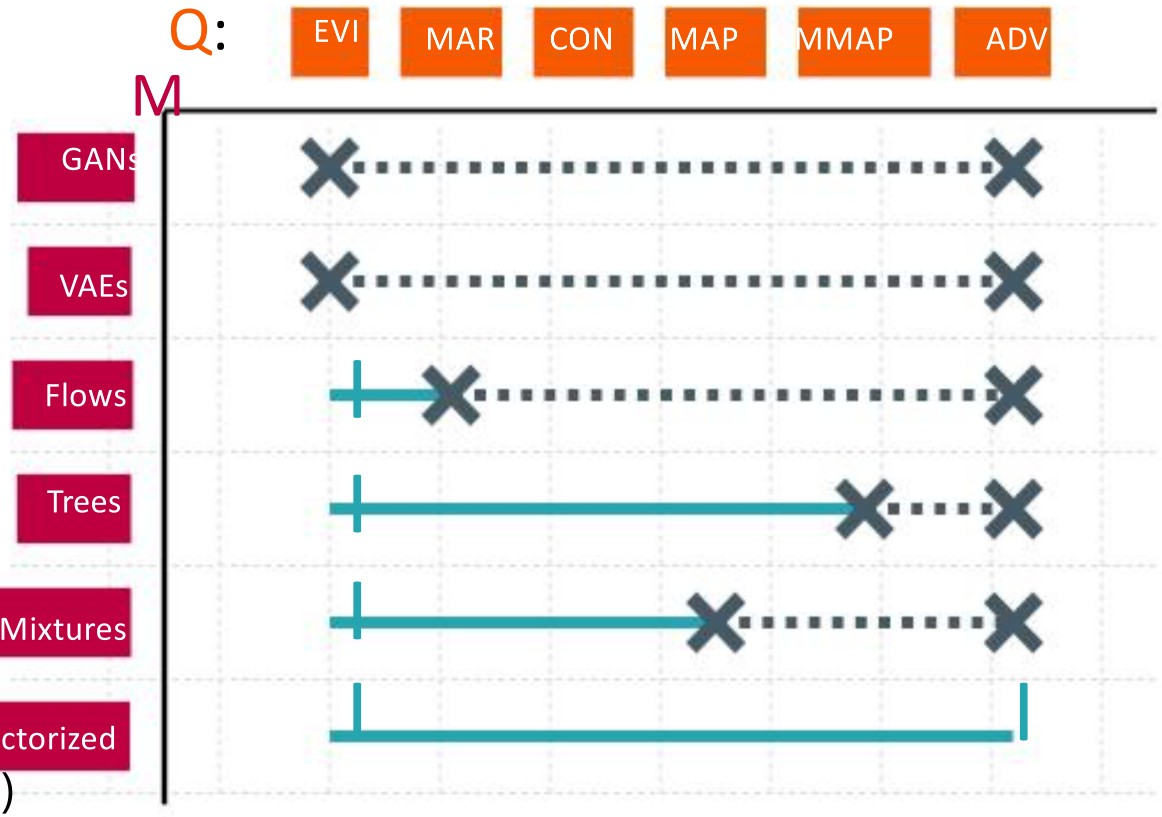
V8/9

```
statesmean = [-1, 1, 0]  # Emission parameters.
initial    = Categorical([1.0/3, 1.0/3, 1.0/3]) # Prob distr of state[1].
trans      = [Categorical([0.1, 0.5, 0.4]), Categorical([0.2, 0.2, 0.6]),
              Categorical([0.15, 0.15, 0.7])]   # Trans distr for each state.
data       = [Nil, 0.9, 0.8, 0.7, 0, -0.025, -5, -2, -0.1, 0, 0.13]

@model hmm begin # Define a model hmm.
 states = Array(Int, length(data))
 @assume(states[1] ~ initial)
 for i = 2:length(data)
   @assume(states[i] ~ trans[states[i-1]])
   @observe(data[i]  ~ Normal(statesmean[states[i]], 0.4))
 end
 @predict states
end
```



Hidden markov model in Julia



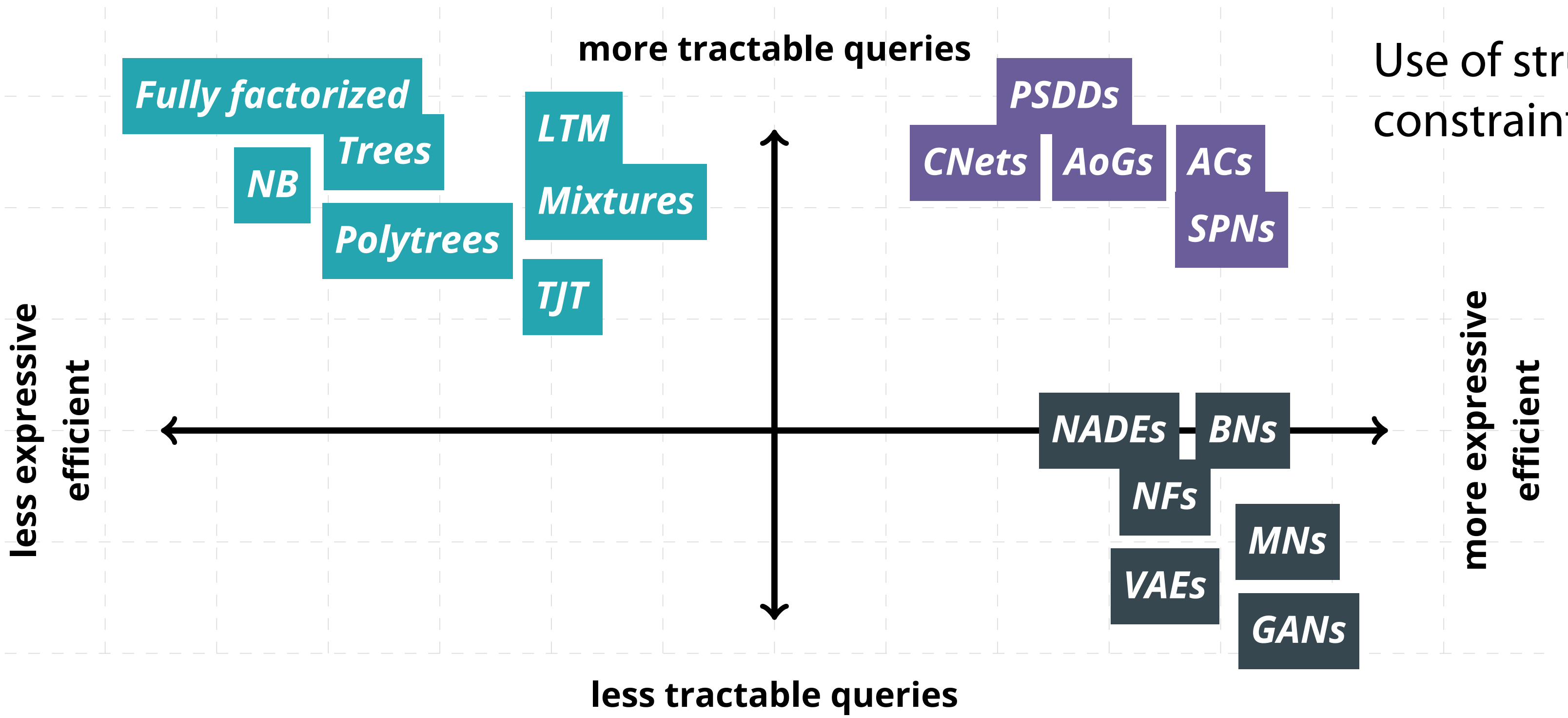
Use these as building blocks

(Sum-node)

(Product Node)

„Eat the cake and have it“

tractable bands



Use of structural constraints

tractability vs expressive efficiency

Can use efficiency also in learning

Thanks for your interest!