Introduction to Machine Learning
Lecture 2

Michael Kagan

SLAC

Hadron Collider Physics Summer School

August 23, 2021

Outline

- Lecture 1
  - Brief introduction to probability and statistics
  - Introduction to Machine Learning fundamentals
  - Linear Models

- Lecture 2
  - Neural Networks
  - Deep Neural Networks
  - Convolutional, Recurrent, and Graph Neural Networks

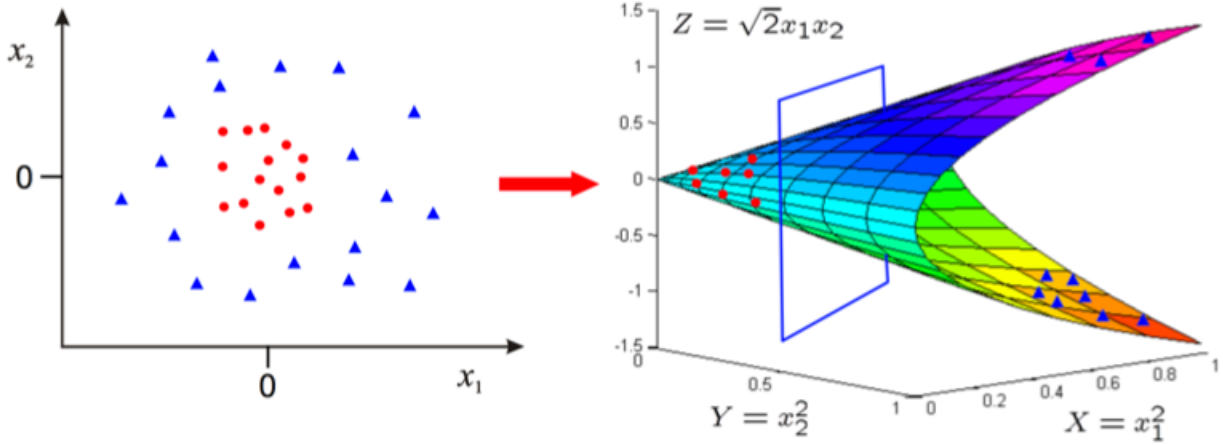
- Lecture 3
  - Unsupervised Learning
  - Autoencoders
  - Generative Adversarial Networks and Normalizing Flows

Adding non-linearity to Logistic Regression

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\phi(\mathbf{x}; \mathbf{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where $\mathbf{u}$ is a set of parameters for the transformation

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\phi(\mathbf{x}; \mathbf{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation
  - Combines basis selection and learning
  - Several different approaches, focus here on neural networks
  - Complicates the optimization

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

- Put all $\mathbf{u}_j \in \mathbb{R}^{1xm}$ vectors into matrix $\mathbf{U}$

$$\phi(\mathbf{x}; \mathbf{U}) = \sigma(\mathbf{U}\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{u}_1^T \mathbf{x}) \\ \sigma(\mathbf{u}_2^T \mathbf{x}) \\ \ldots \\ \sigma(\mathbf{u}_d^T \mathbf{x}) \end{bmatrix} \in \mathbb{R}^d$$

  - σ is a point-wise non-linearity acting on each vector element

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\mathbf{x}; \mathbf{u}) = \sigma(\mathbf{u}_j^T \mathbf{x})$$

- Put all $\mathbf{u}_j \in \mathbb{R}^{1xm}$ vectors into matrix $\mathbf{U}$

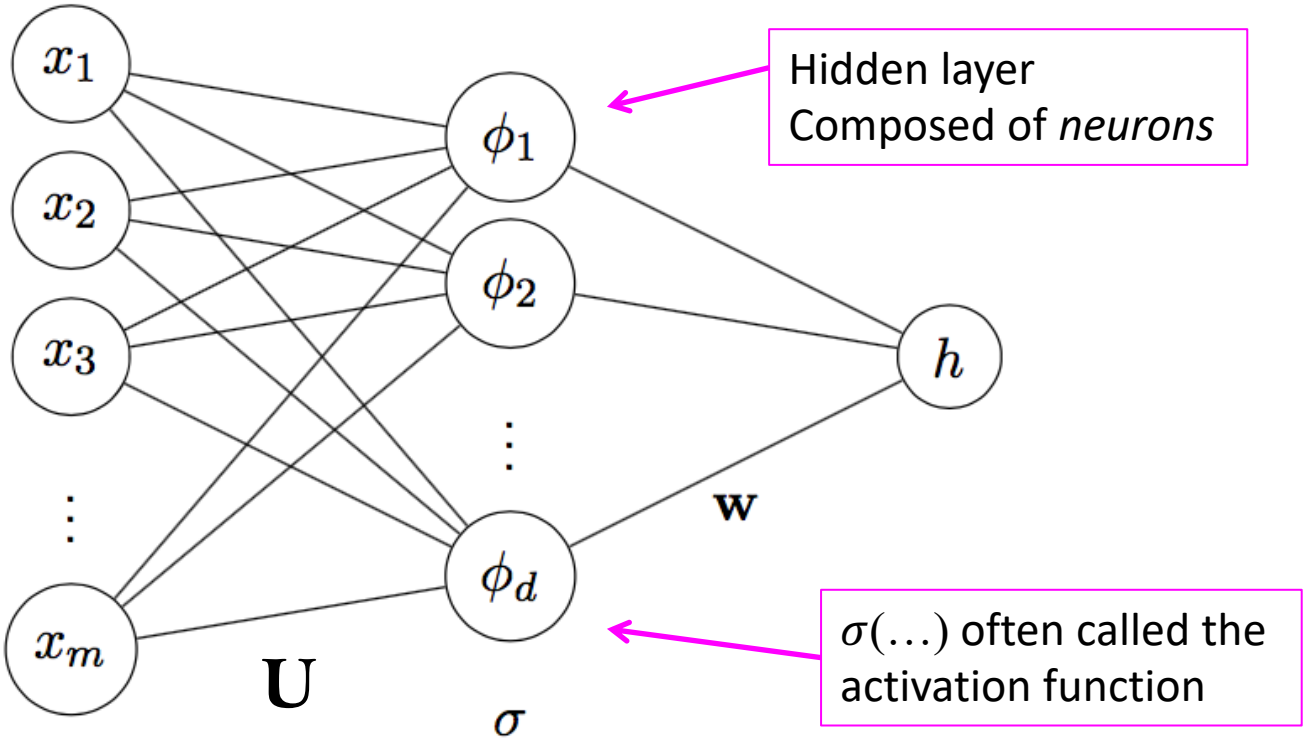
$$\phi(\mathbf{x}; \mathbf{U}) = \sigma(\mathbf{U}\mathbf{x}) = \begin{bmatrix} \sigma(\mathbf{u}_1^T \mathbf{x}) \\ \sigma(\mathbf{u}_2^T \mathbf{x}) \\ \ldots \\ \sigma(\mathbf{u}_d^T \mathbf{x}) \end{bmatrix} \in \mathbb{R}^d$$

  - σ is a point-wise non-linearity acting on each vector element

- Full model becomes

$$h(\mathbf{x}; \mathbf{w}, \mathbf{U}) = \mathbf{w}^T \phi(\mathbf{x}; \mathbf{U})$$

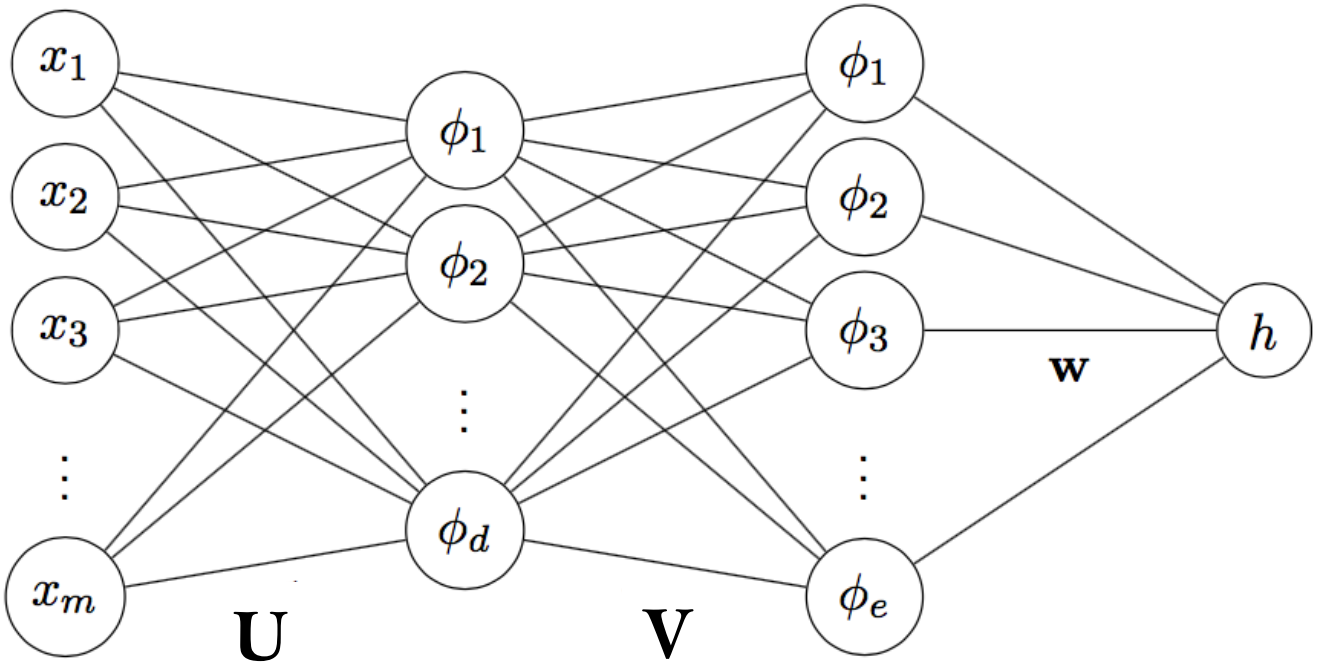
Feed Forward Neural Network



$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\mathbf{x})$$

$$h(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

Multi-layer Neural Network



- Multilayer NN
  - Each layer adapts basis functions based on previous layer

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

- Minimize loss with respect to weights **w, U**

Minimizing loss with gradient descent:

- Parameter update:

$$w \leftarrow w - \eta \frac{\partial L(w, U)}{\partial w}$$

$$U \leftarrow U - \eta \frac{\partial L(w, U)}{\partial U}$$

- How to compute gradients?
  - In principle, we could compute them analytically, but the resulting expressions quickly get out of hand, are hard to simplify, and can drain memory

Automatic Differentiation

$$w_1 = x_1 = 2$$
$$w_2 = x_2 = 3$$
$$w_3 = w_1 w_2 = 6$$
$$w_4 = \sin(w_1) = 0.9$$
$$w_5 = w_3 + w_4 = 6.9$$
$$z = w_5$$

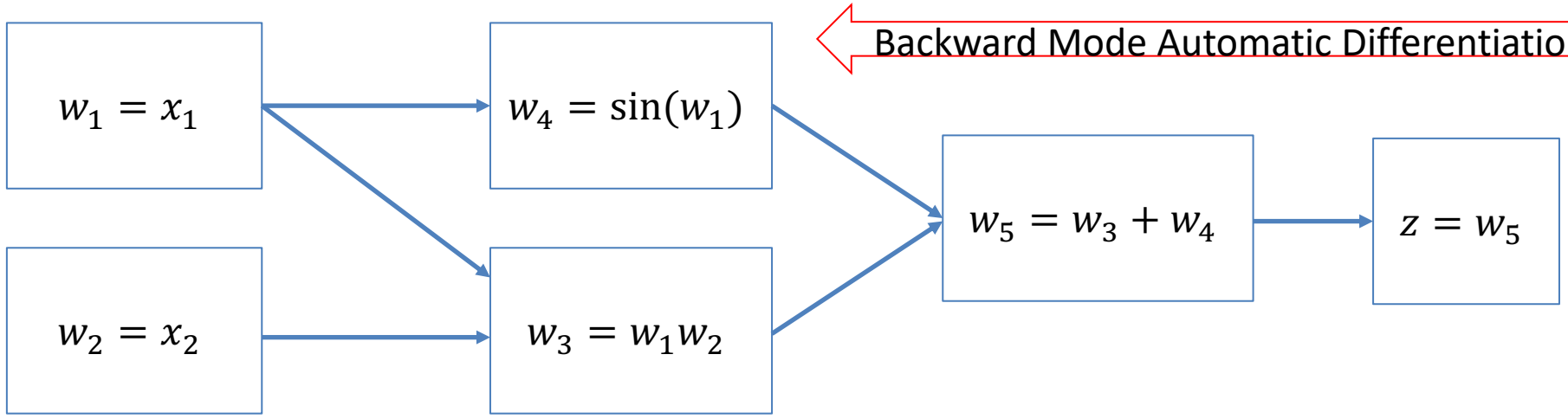
Organize function as computational graph of elementary operations

Evaluate blocks and their derivatives, and apply chain rule:

$$\frac{dz}{dw_1} = \sum_{p \in parents} \frac{dz}{dw_p} \frac{dw_p}{dw_i}$$

$$\frac{dw_1}{dx_1} = 1$$
$$\frac{dw_2}{dx_2} = 1$$
$$\frac{dw_3}{dw_1} = w_2 \quad \frac{dw_3}{dw_2} = w_1$$
$$\frac{dw_4}{dw_1} = \cos(w_1)$$
$$\frac{dw_5}{dw_3} = 1 \quad \frac{dw_5}{dw_4} = 1$$

$z = \sin(x_1) + x_1 x_2$



See Backup for more details

Backpropagation

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^N\left(\ldots\phi^1(x)\right)$$

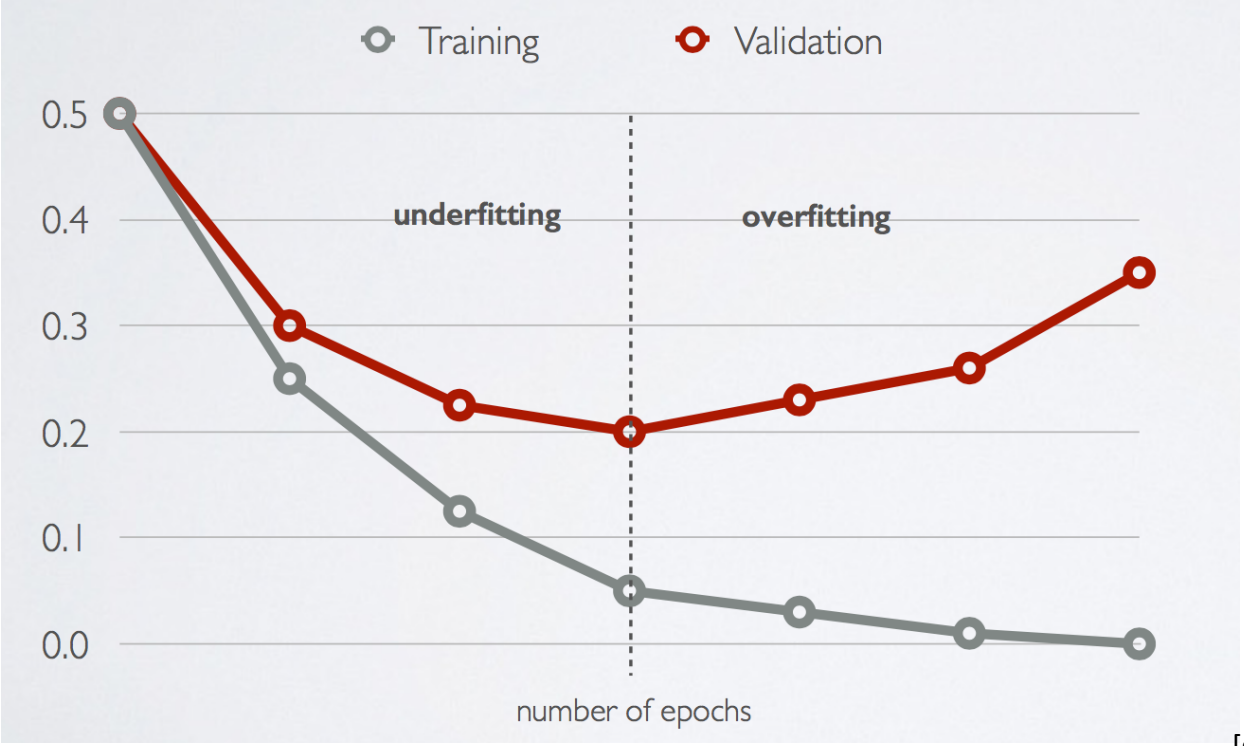
- Backward step (b-prop) $\dfrac{\partial L}{\partial \phi^a} = \sum_j \dfrac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \dfrac{\partial L}{\partial \phi_j^{(a+1)}}$

- Compute parameter gradients $\dfrac{\partial L}{\partial \mathbf{w}^a} = \sum_j \dfrac{\partial \phi_j^a}{\partial \mathbf{w}^a} \dfrac{\partial L}{\partial \phi_j^a}$

Training

- Repeat gradient update of weights to reduce loss
  - Each iteration through dataset is called an epoch

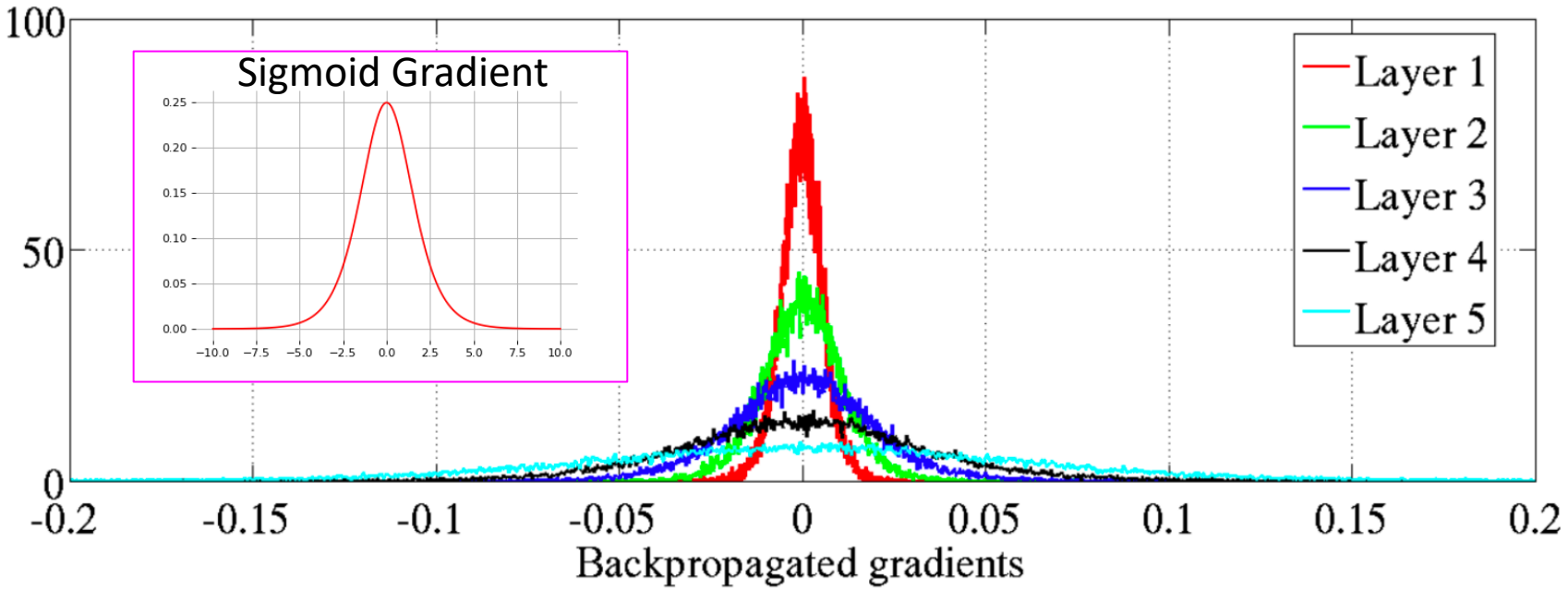
- Use validation set to examine for overtraining, and determine when to stop training



[graphic from H. Larochelle]

Vanishing Gradients

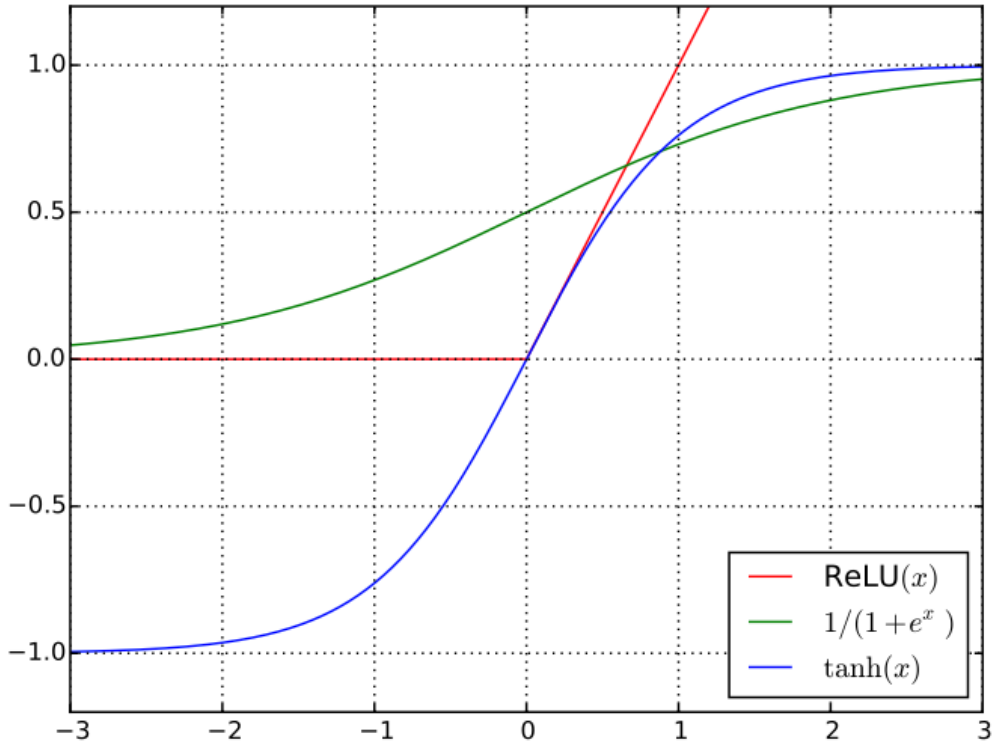
- Major challenge in DL: Vanishing Gradients
- Small gradients slow down / block, stochastic gradient descent → Limits ability to learn!



Backpropagated gradients normalized histograms (Glorot and Bengio, 2010). Gradients for layers far from the output vanish to zero.

Slide credit: G. Louppe

Activation Functions



- **Vanishing gradient problem**
  - Derivative of sigmoid:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

  - Nearly 0 when x is far from 0!
  - Can make gradient descent hard!

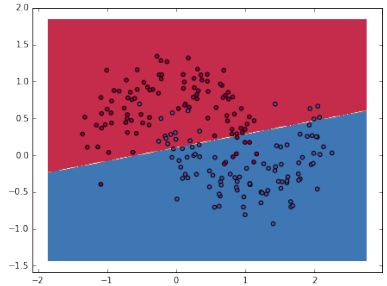
- **Rectified Linear Unit (ReLU)**
  - ReLU(x) = max{0, x}
  - Derivative is constant!

$$\frac{\partial \mathrm{Re}LU(x)}{\partial x} = \begin{cases} 1 & when\ x > 0 \\ 0 & otherwise \end{cases}$$

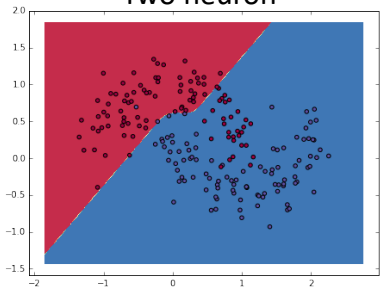
  - ReLU gradient doesn't vanish

Neural Network Decision Boundaries

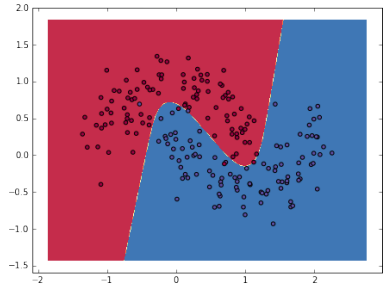
One neuron



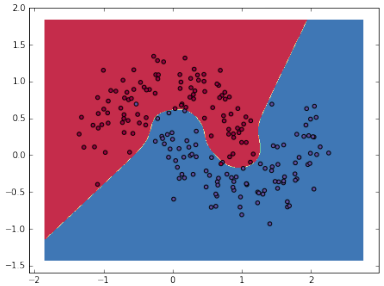
Two neuron



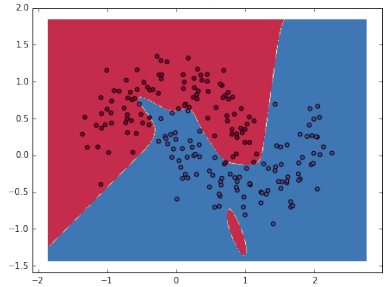
Three neurons



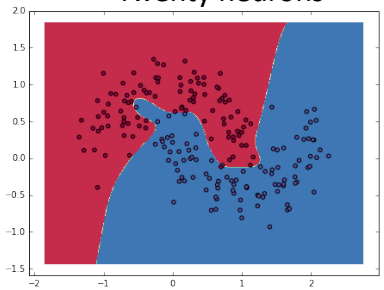
Four neurons



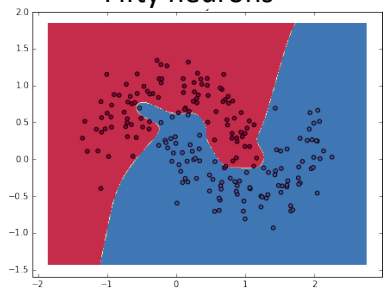
Five neurons



Twenty neurons



Fifty neurons



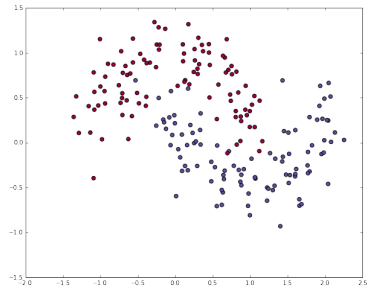
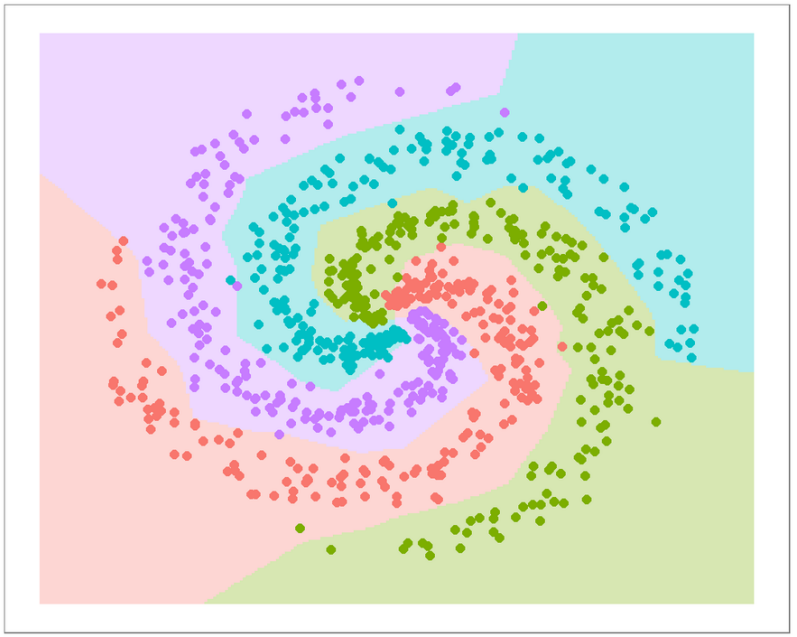


Image source

4-class classification
2-hidden layer NN
ReLU activations
L2 norm regularization

x_2



x_1

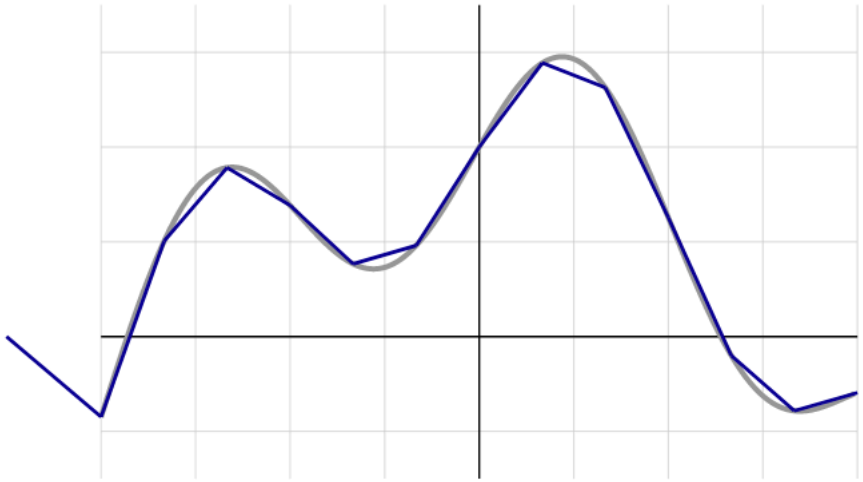
2-class classification
1-hidden layer NN
L2 norm regularization

Image source

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$



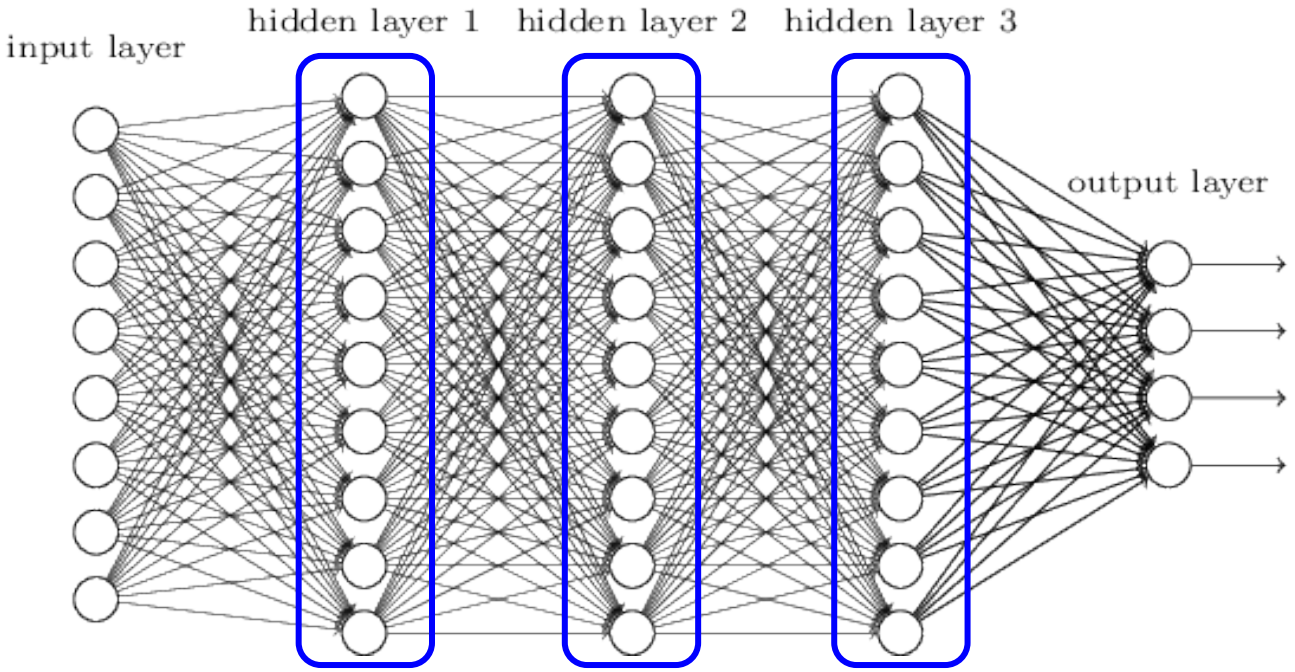
Fleuret, Deep Learning Course

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

- NOTE!
  - A better approximation requires a larger hidden layer and this theorem says nothing about the relation between the two.
  - We can make training error as low as we want by using a larger hidden layer. Result states nothing about test error
  - Doesn't say how to find the parameters for this approximation

Deep Neural Networks



- As data complexity grows, need exponentially large number of neurons in a single-hidden-layer network to capture all structure in data
- Deep neural networks *factorize the learning* of structure in data across many layers
- Difficult to train, only recently possible with large datasets, fast computing (GPU / TPU) and new training procedures / network structures

Neural Network Zoo

- Structure of the networks, and the node connectivity can be adapted for problem at hand
- Moving inductive bias from feature engineering to model design
  - *Inductive bias*: Knowledge about the problem
  - *Feature engineering*: Hand crafted variables
  - *Model design*: The data representation and the structure of the machine learning model / network

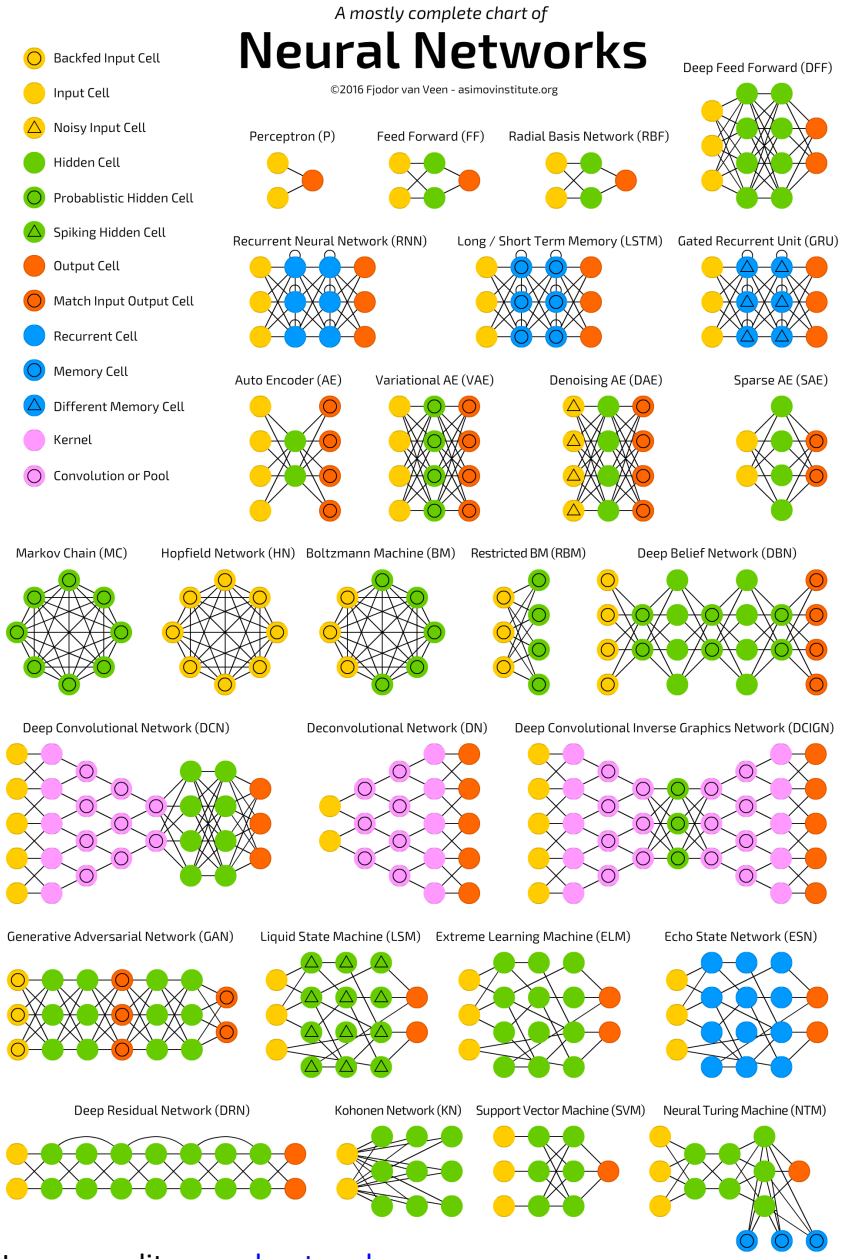
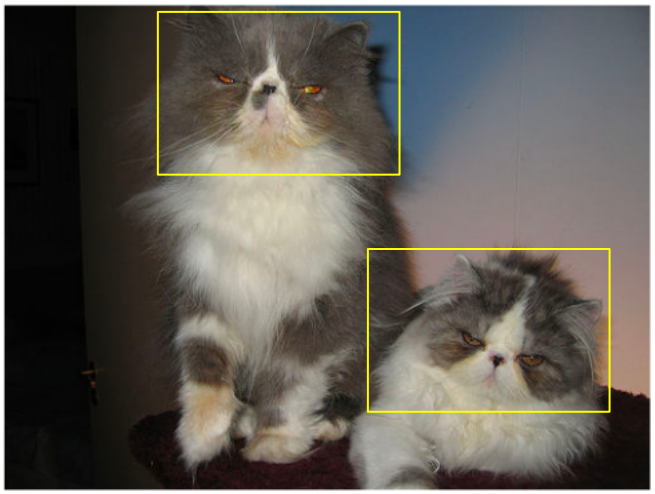


Image credit: [neural-network-zoo]

Convolutional Neural Networks

Convolutional Neural Networks

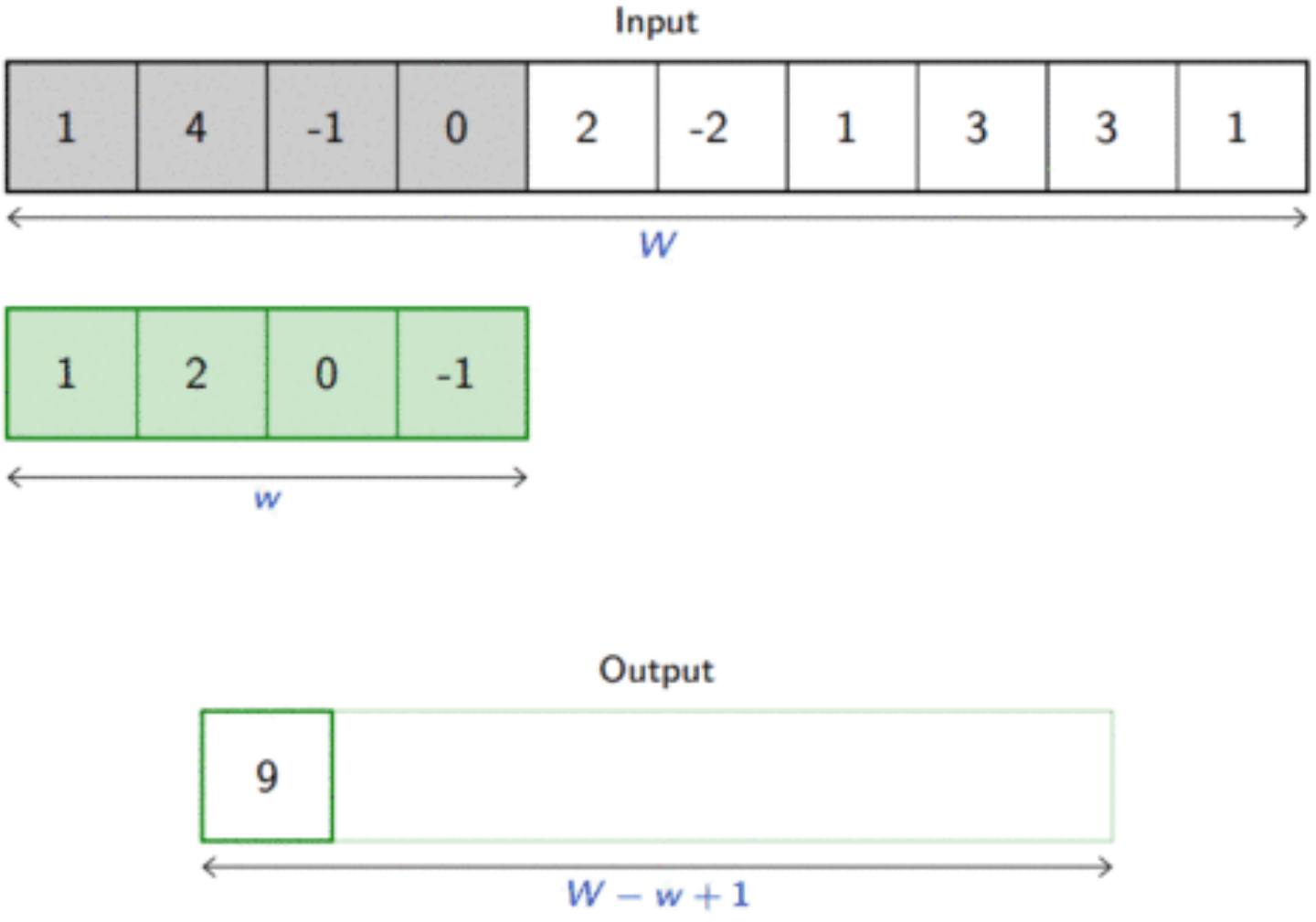
- When the structure of data includes "invariance to translation", a representation meaningful at a certain location can / should be used everywhere



- Covolutional layers build on this idea, that the same "local" transformation is applied everywhere and preserves the signal structure

Fleuret, Deep Learning Course

1D Convolutional Layer Example



Fleuret, Deep Learning Course

1D Convolutional Layers

- **Data:** $x \in \mathbb{R}^M$
- **Convolutional kernel of width k:** $u \in \mathbb{R}^k$

- Convolution $x \circledast u$ is vector of size M-k+1

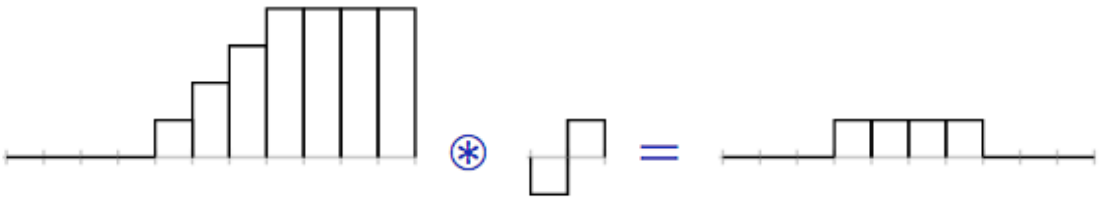
$$(x \circledast \mathrm{u})_i = \sum_{b=0}^{k-1} x_{i+b} u_b$$

- Scan across data and multiply by kernel elements

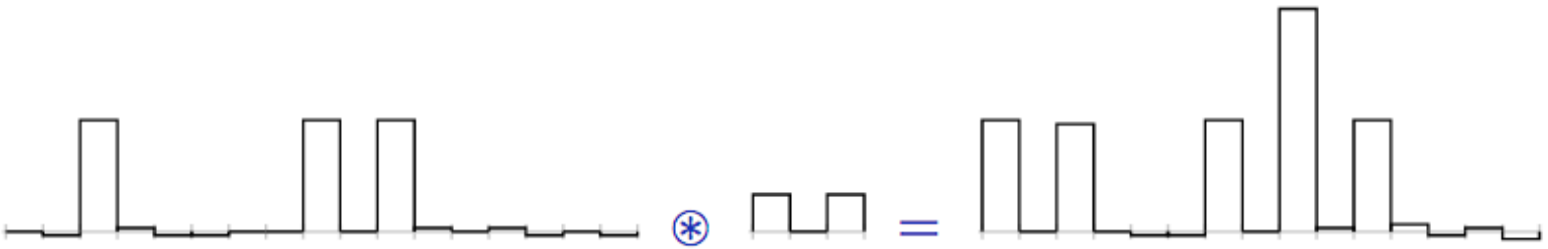
Convolutional Filters

Convolution can implement in particular differential operators, *e.g.*

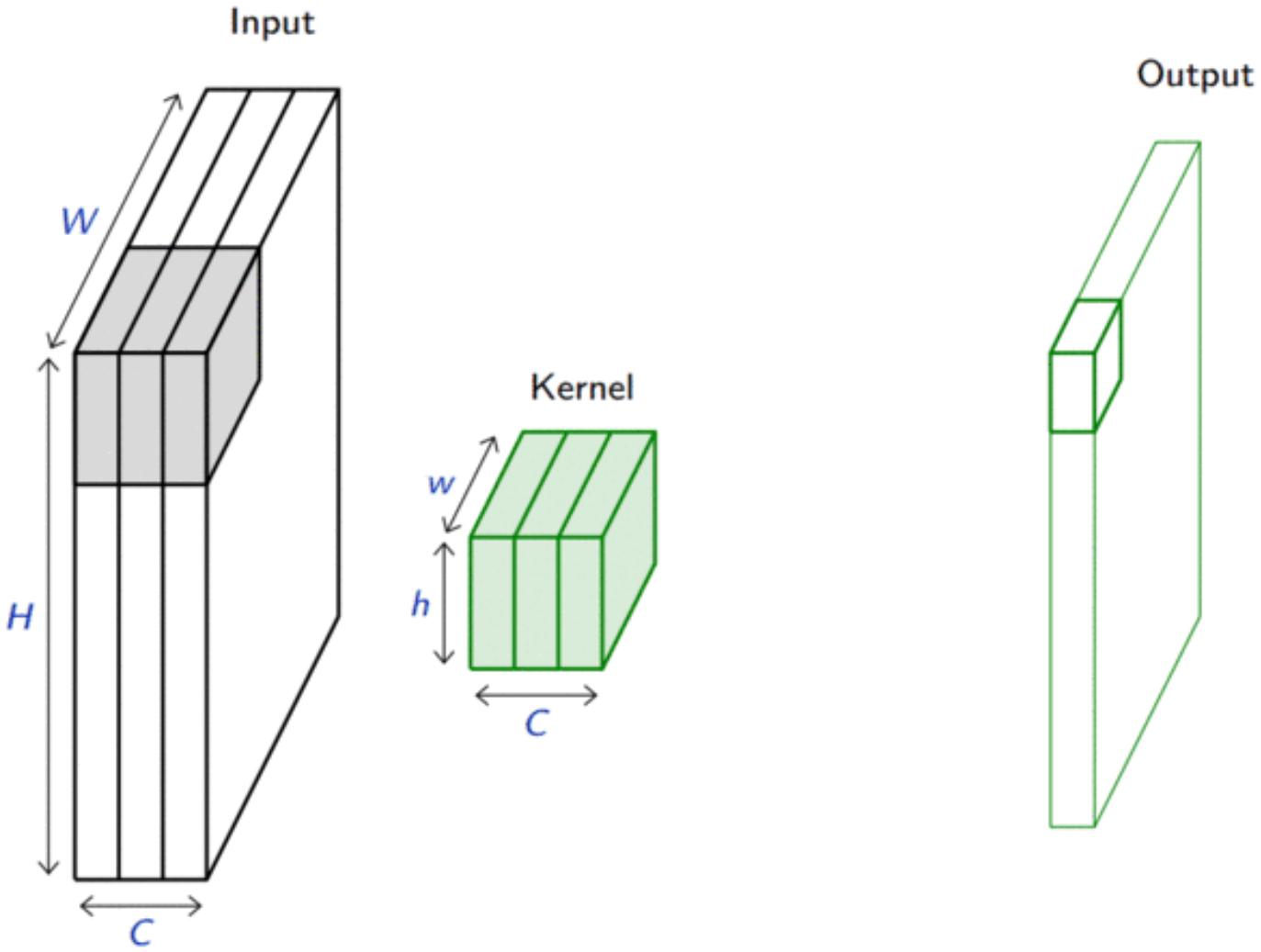
$$(0,0,0,0,1,2,3,4,4,4,4) \circledast (-1,1) = (0,0,0,1,1,1,1,0,0,0).$$



or crude "template matcher", *e.g.*

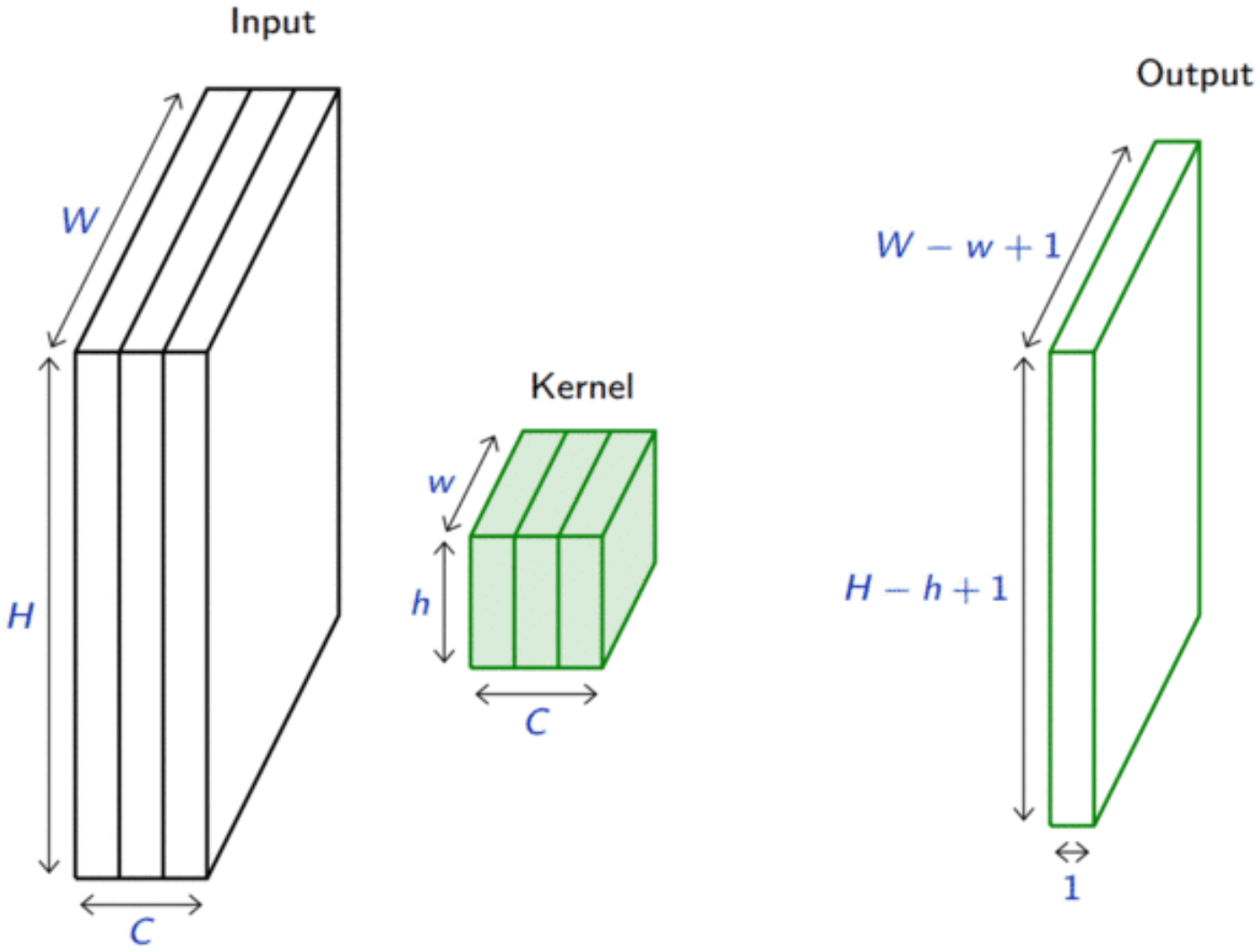


2D Convolution Over Multiple Channels



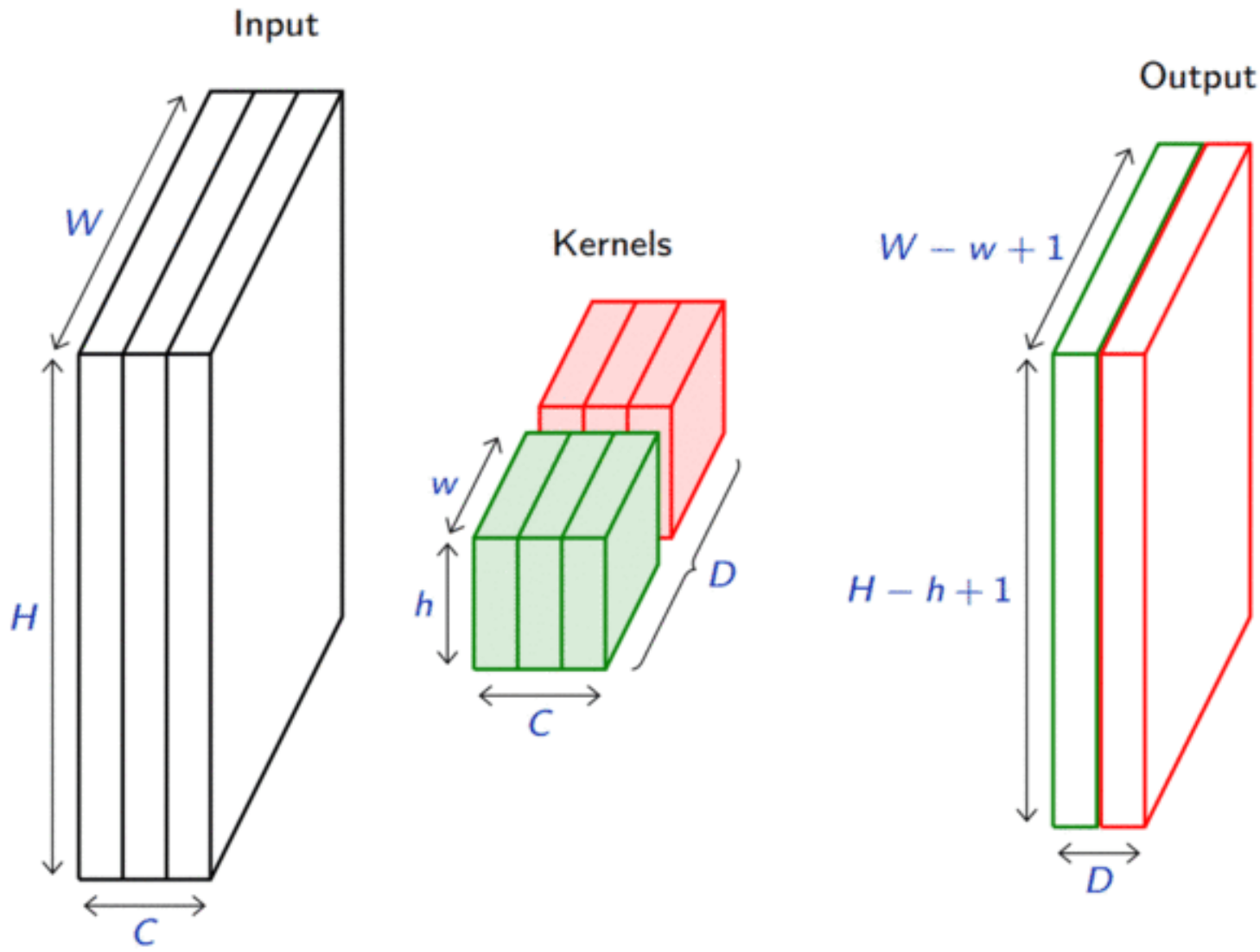
Fleuret, Deep Learning Course

2D Convolution Over Multiple Channels



Fleuret, Deep Learning Course

2D Convolution Over Multiple Channels



Fleuret, Deep Learning Course

2D Convolutional Layer

- Input data (tensor) $\mathbf{x}$ of size $C \times H \times W$
  - C channels (e.g. RGB in images)

- Learnable Kernel $\mathbf{u}$ of size $C \times h \times w$
  - The size $h \times w$ is the *receptive field*

$$(\boldsymbol{x} \circledast \boldsymbol{u})_{i,j} = \sum_{c=0}^{C-1} (\boldsymbol{x}_c \circledast \boldsymbol{u}_c)_{i,j} = \sum_{c=0}^{C-1} \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \boldsymbol{x}_{c,n+i,m+j} \boldsymbol{u}_{c,n,m}$$

- Output size $(H - h + 1) \times (W - w + 1)$ for each kernel
  - Often called *Activation Map* or *Output Feature Map*

Shared Weights: Economic and Equivariant

- Parameters are *shared* by each neuron producing an output in the activation map

- Dramatically reduces number of weights needed to produce an activation map
  - Data: $256 \times 256 \times 3$ RGB image
  - Kernel: $3 \times 3 \times 3 \rightarrow 27$ weights
  - Fully connected layer:
    - $256 \times 256 \times 3$ inputs → $256 \times 256 \times 3$ outputs → $O(10^{10})$ weights

Y. LeCun et. al. 1998

Shared Weights: Economic and Equivariant

- Parameters are *shared* by each neuron producing an output in the activation map

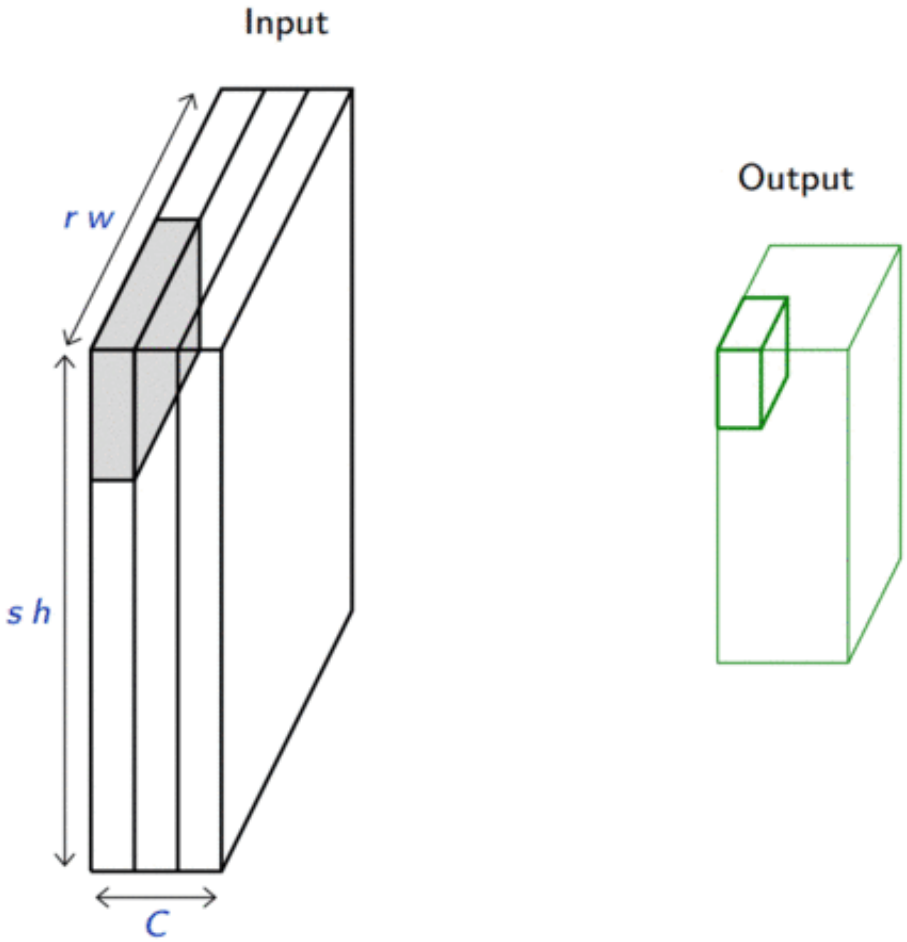
- Dramatically reduces number of weights needed to produce an activation map

- Convolutional layer does pattern matching at any location → Equivariant to translation

Y. LeCun et. al. 1998

Pooling

- In each channel, find *max* or *average* value of pixels in a pooling area of size $h \times w$

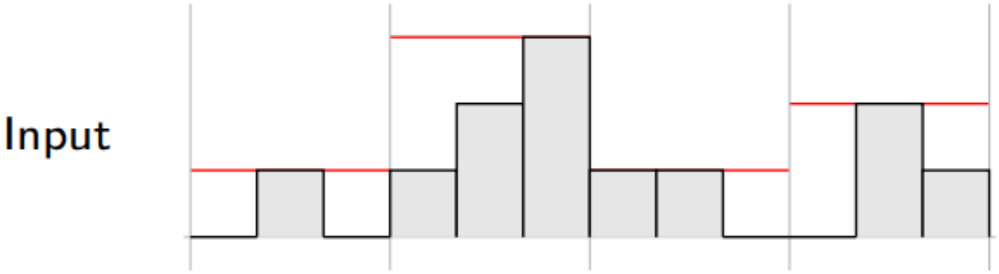


Fleuret, Deep Learning Course

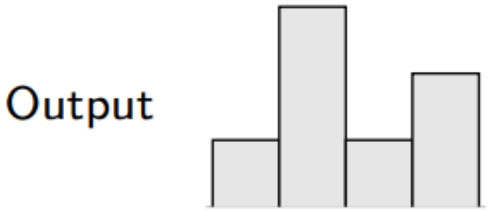
Pooling

- In each channel, find *max* or *average* value of pixels in a pooling area of size $h \times w$

- Invariance to permutation within pooling area

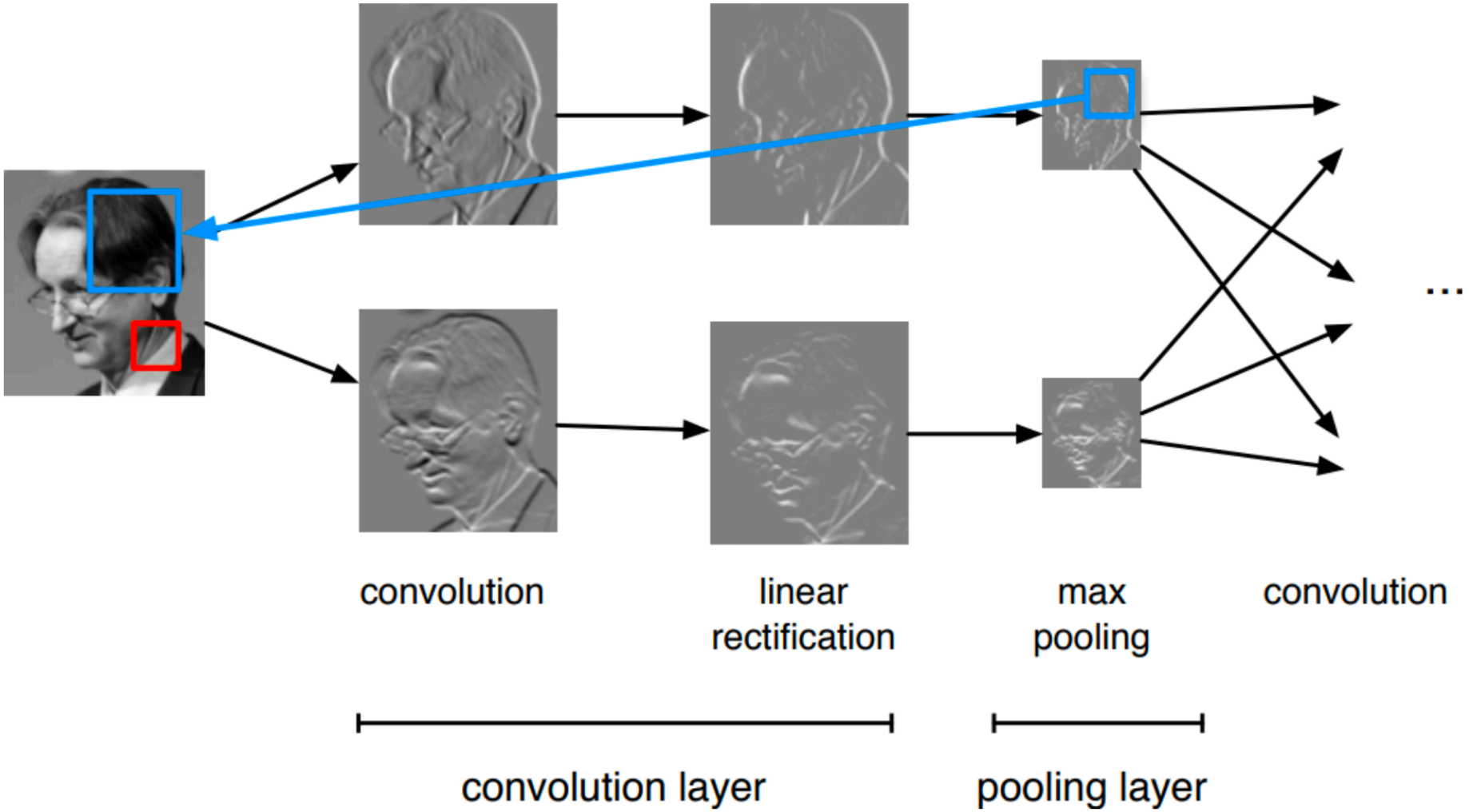


- Invariance to local perturbations

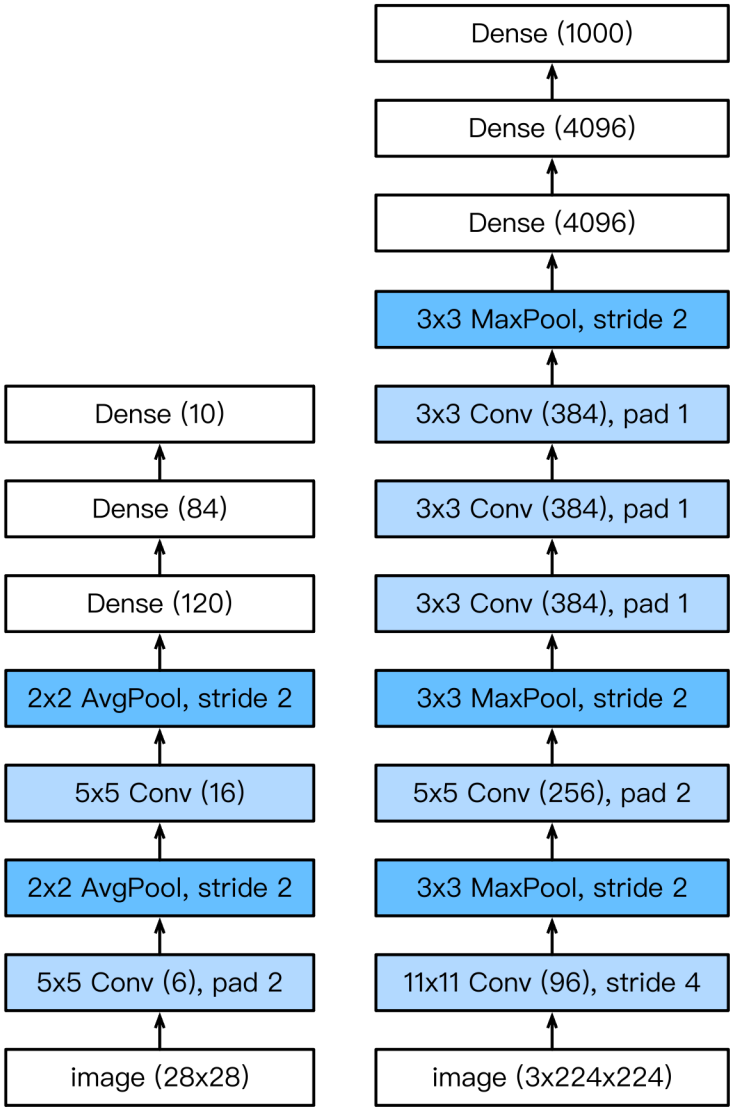


Convolutional Network

- A combination of convolution, pooling, ReLU, and fully connected layers



Convolutional Networks

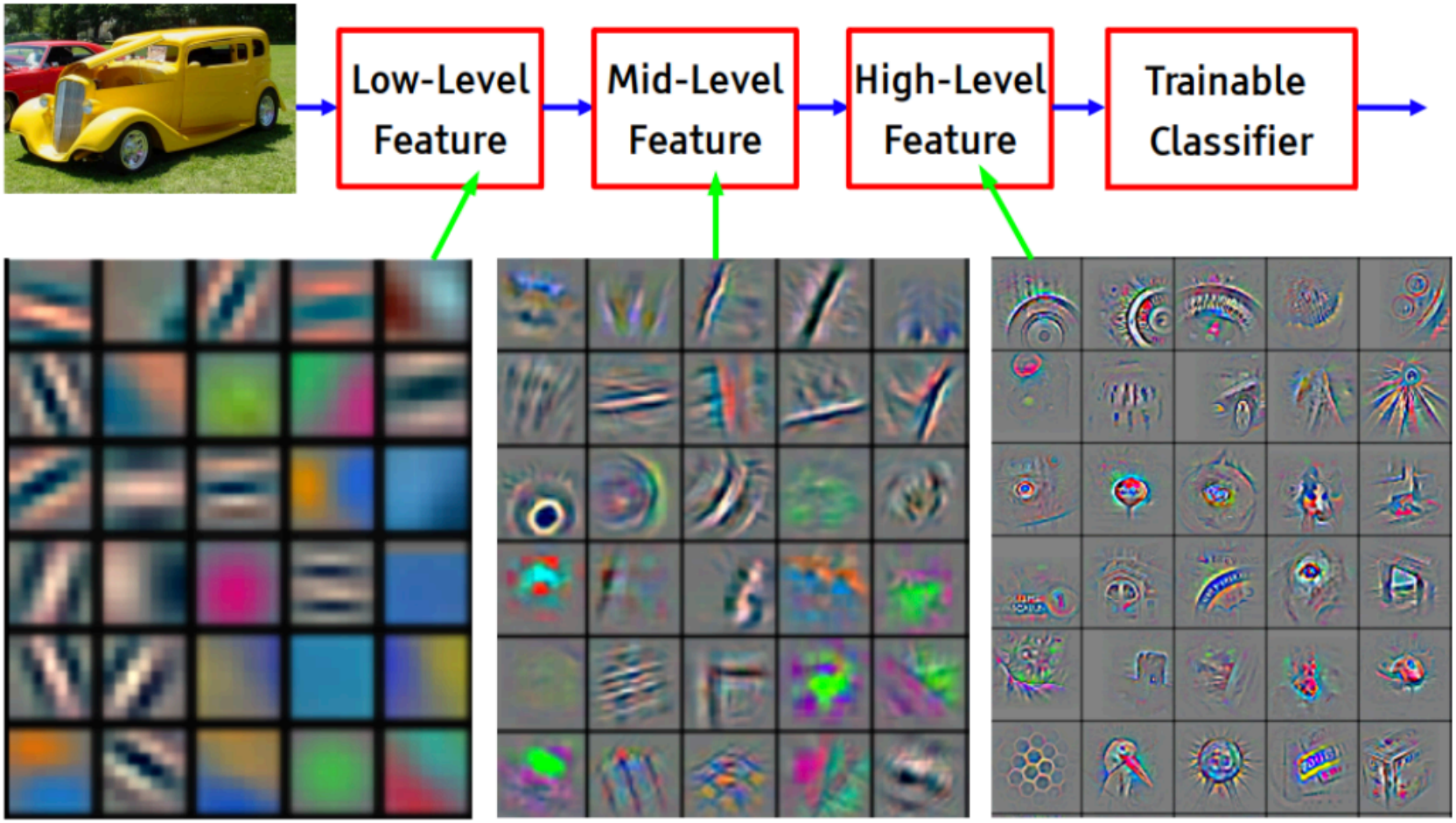


LeNet
(LeCun et al, 1998)

AlexNet
(Krizhevsky et al, 2012)

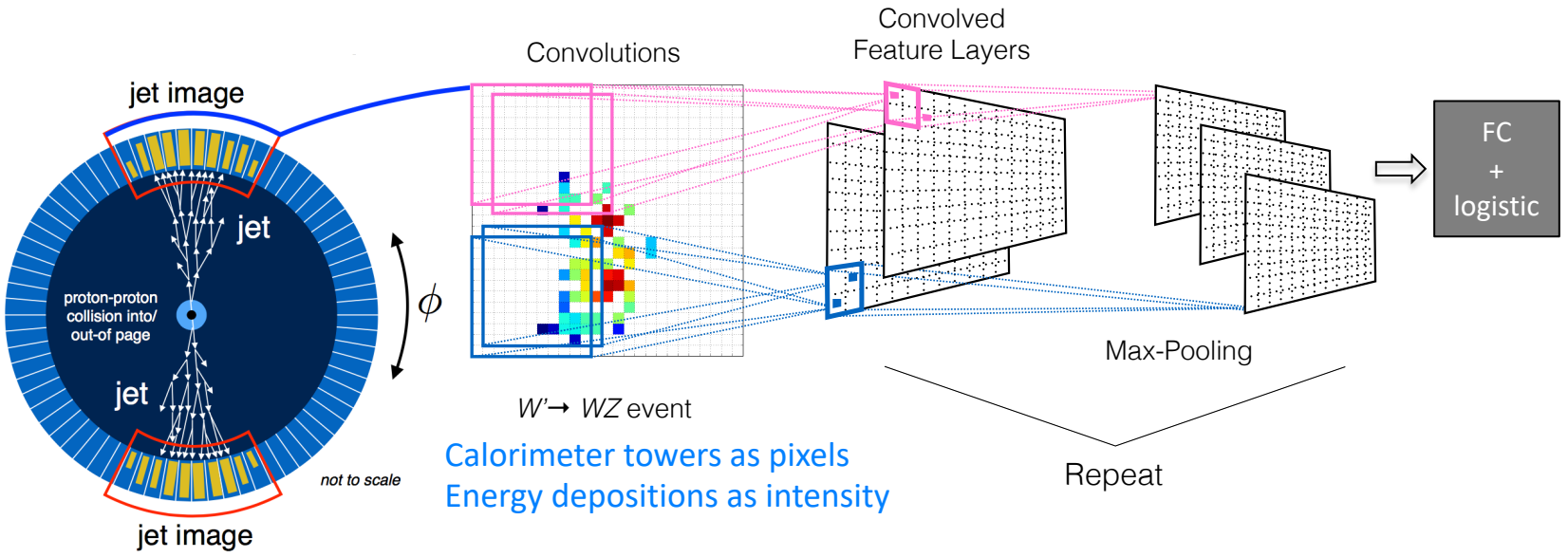
ImageNet Classification

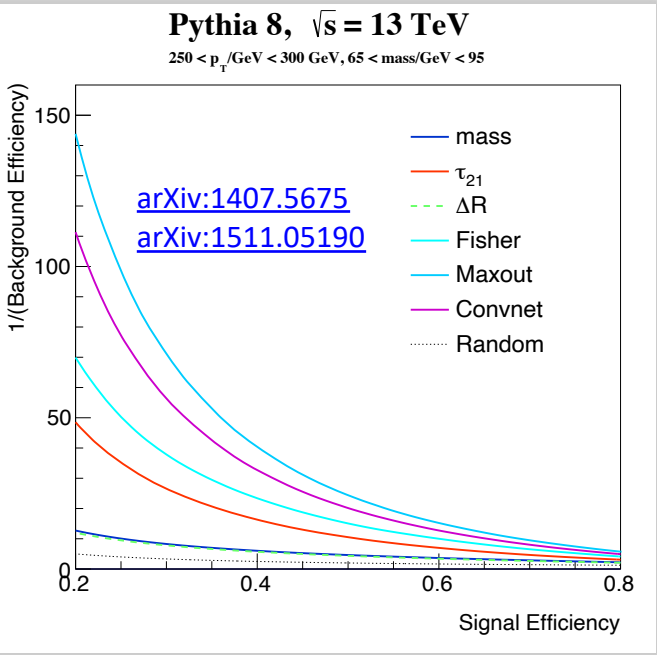
Hierarchical Composition of Features

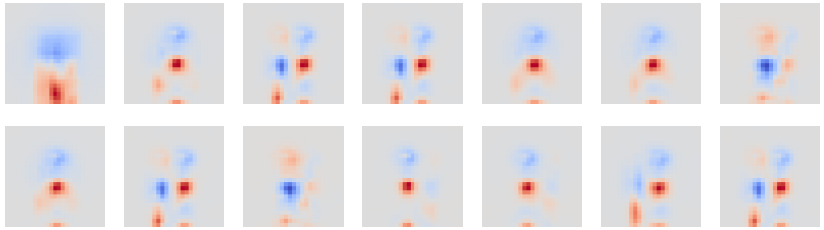


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNNs for Jet Tagging

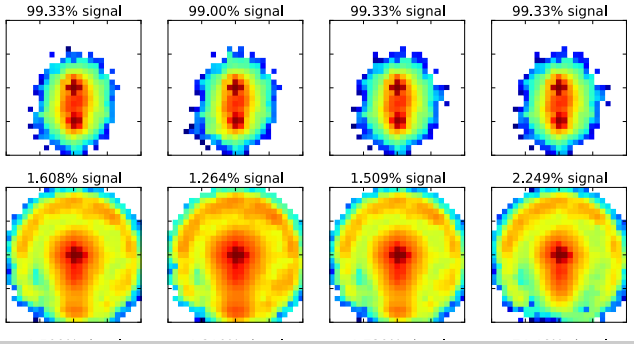






Filters convolved with images

Most activating images



Recurrent Neural Networks

Sequential Data

- Many types of data are not fixed in size
- Many types of data have a temporal or sequence-like structure
  - Text
  - Video
  - Speech
  - DNA
  - …
- MLP expects fixed size data
- How to deal with sequences?

Sequential Data

- Given a set $\mathcal{X}$, let $S(\mathcal{X})$ be the set of sequences, where each element of the sequence $x_i \in \mathcal{X}$
  - $\mathcal{X}$ could reals $\mathbb{R}^M$, integers $\mathbb{Z}^M$, etc.
  - Sample sequence $x = \{x_1, x_2, \ldots, x_T\}$

- Tasks related to sequences:
  - Classification $f\colon S(\mathcal{X}) \to \{\boldsymbol{p} \mid \sum_{c=1}^{N} p_i = 1\}$
  - Generation $f\colon \mathbb{R}^d \to S(\mathcal{X})$
  - Seq.-to-seq. translation $f\colon S(\mathcal{X}) \to S(\mathcal{Y})$

Credit: F. Fleuret

Recurrent States

- Input sequence $x \in S(\mathbb{R}^m)$ of *variable* length $T(x)$

- Standard approach: use recurrent model that maintains a **recurrent state** $\boldsymbol{h}_t \in \mathbb{R}^q$ updated at each time step t. For $t = 1, \ldots, T(x)$:

$$\boldsymbol{h}_{t+1} = \phi(\boldsymbol{x}_t, \boldsymbol{h}_t; \theta)$$

  – Simplest model:

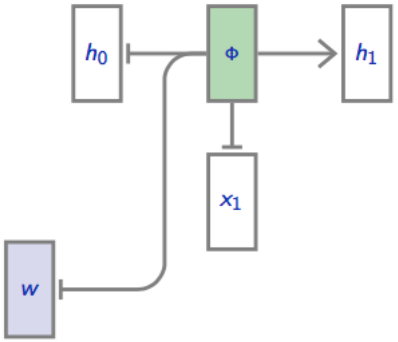
$$\phi(\boldsymbol{x}_t, \boldsymbol{h}_t; W, U) = \sigma(W\boldsymbol{x}_t + U\boldsymbol{h}_t)$$

- Predictions can be made at any time t from the recurrent state

$$\boldsymbol{y}_t = \psi(\boldsymbol{h}_t; \theta)$$

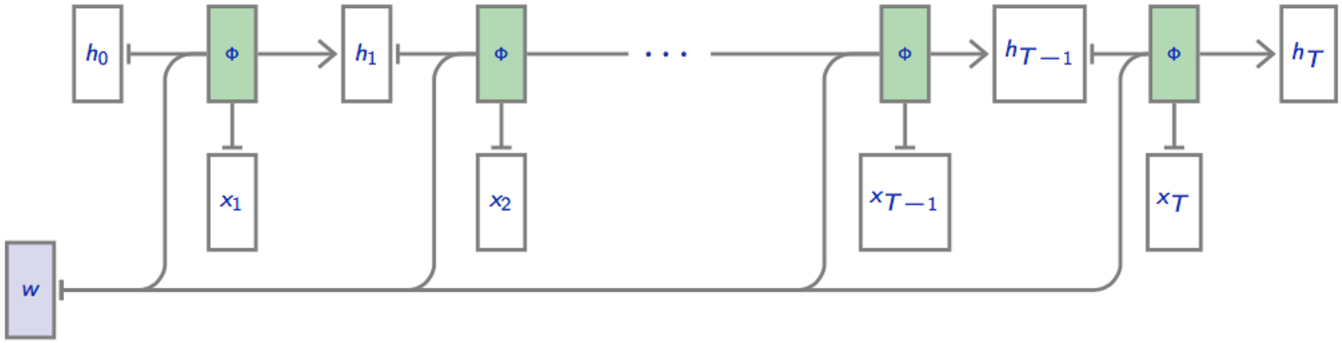
Credit: F. Fleuret

Recurrent Neural Networks



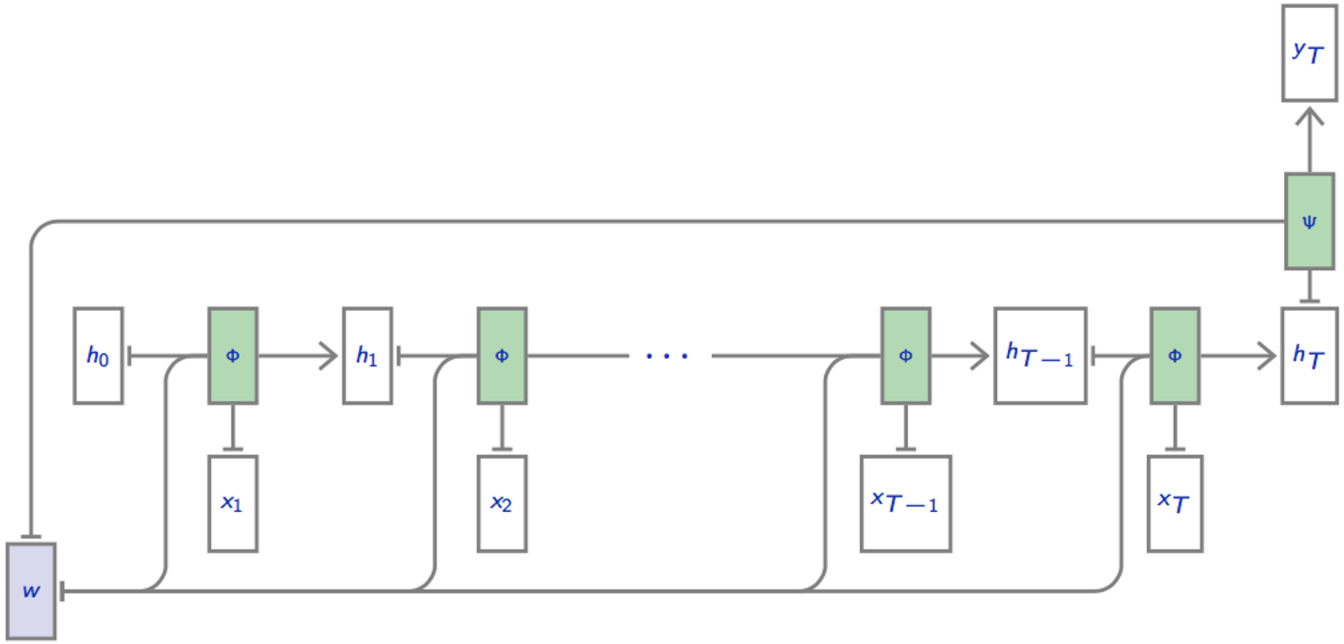
Credit: F. Fleuret

Recurrent Neural Networks



Credit: F. Fleuret

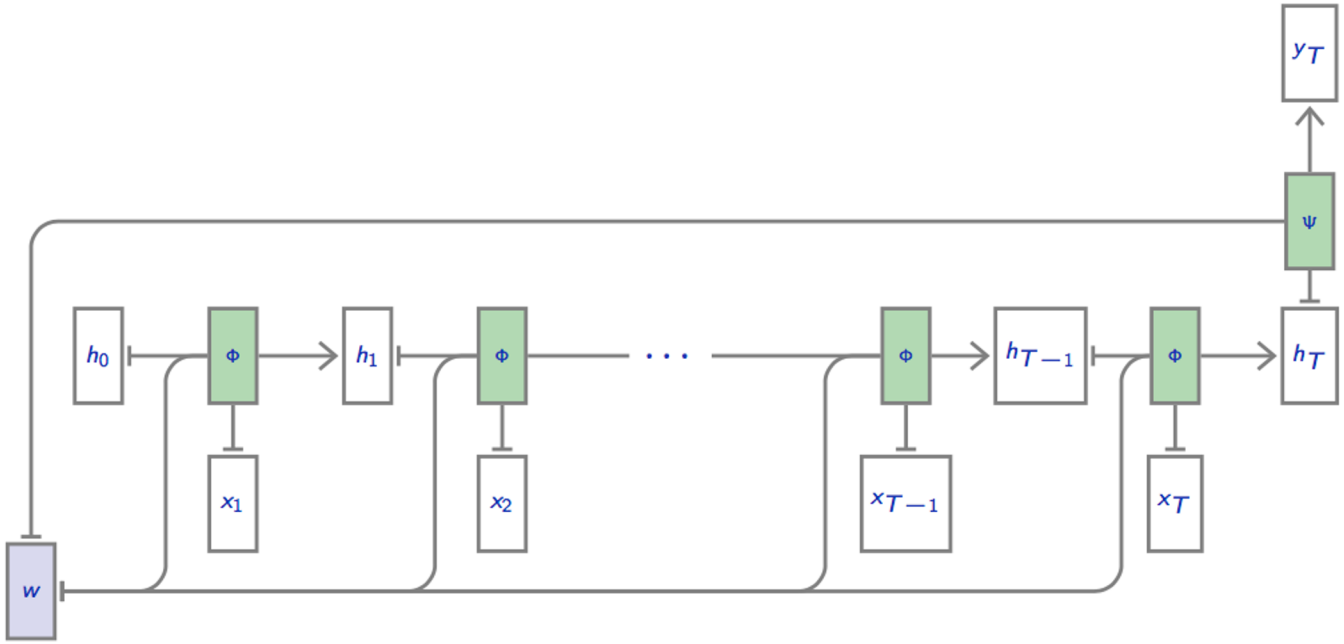
Recurrent Neural Networks



Credit: F. Fleuret

Recurrent Neural Networks

[0.98] → Positive Sentiment



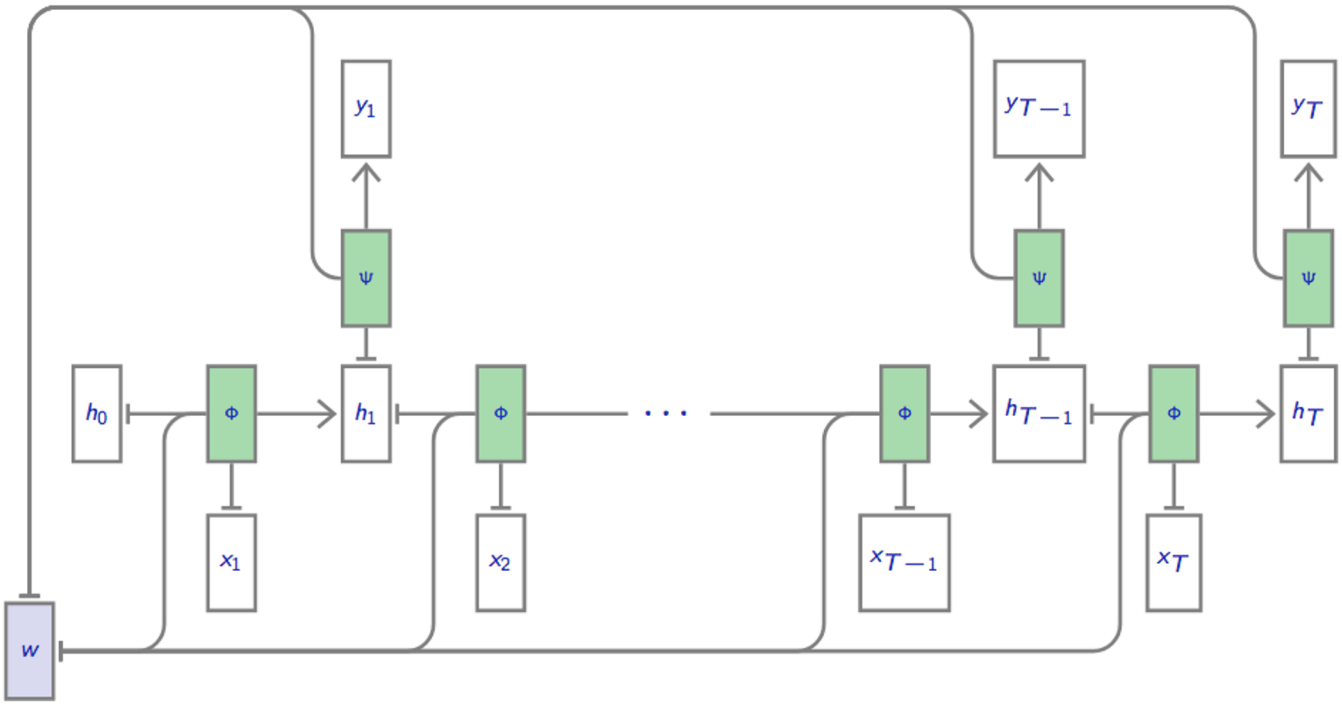
Sentiment Analysis

The movie was great

Credit: F. Fleuret

Recurrent Neural Networks

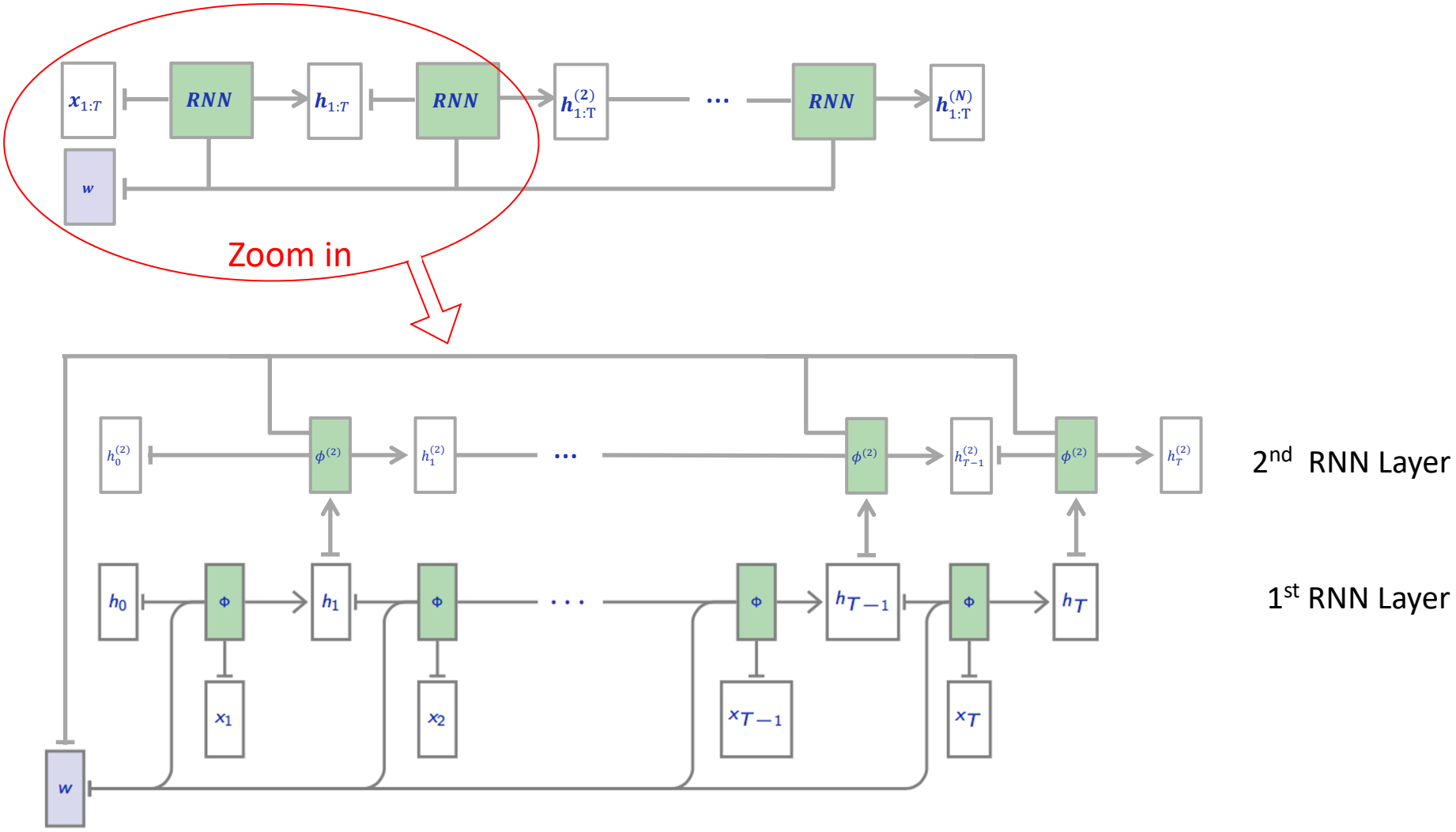
Prediction per sequence element



Although the number of steps $T(x)$ depends on x, this is a standard computational graph and automatic differentiation can deal with it as usual. This is known as "backpropagation through time" (Werbos, 1988)

Credit: F. Fleuret

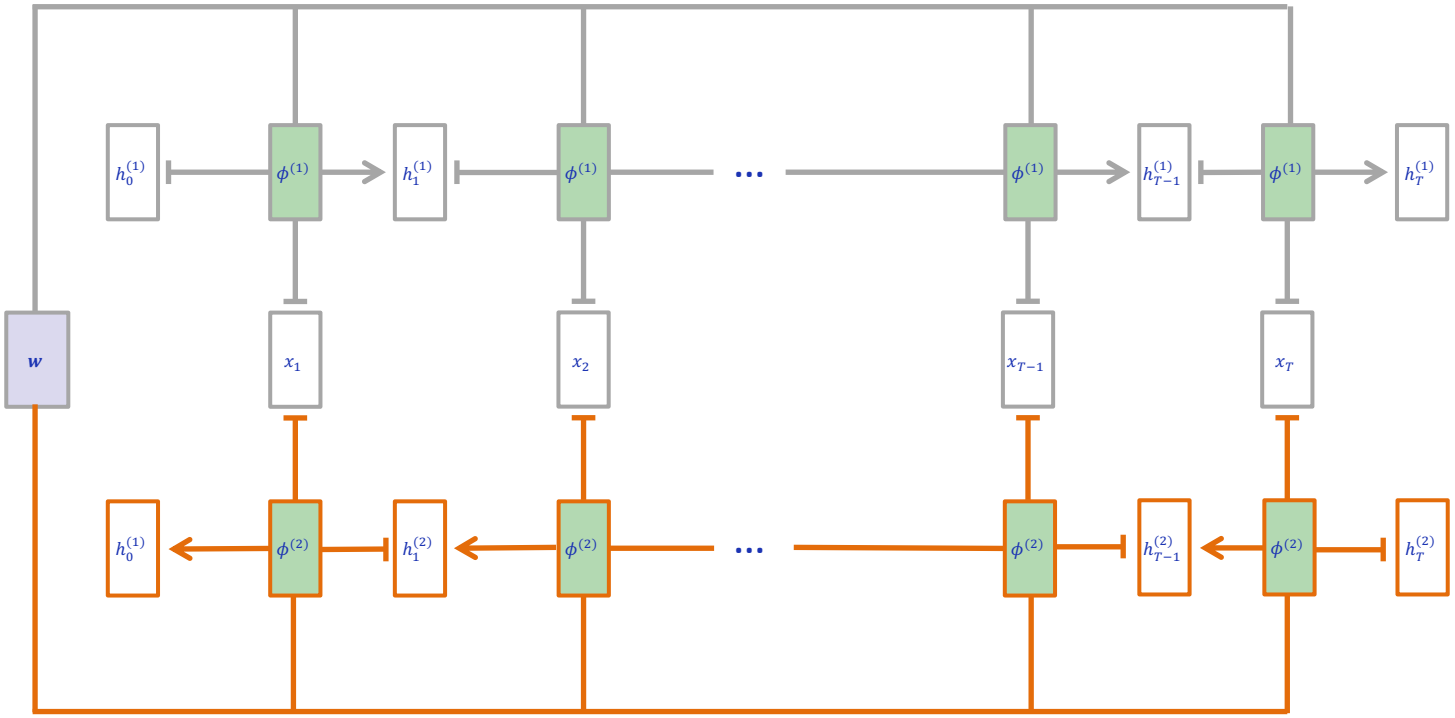
Stacked RNN



Two Stacked LSTM Layers

Bi-Directional RNN

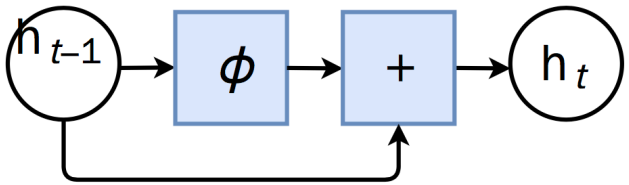
Forward in time RNN Layer



Backward in time RNN Layer

Gating

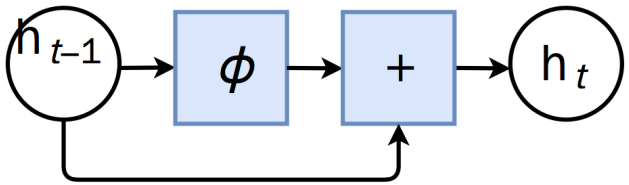
- Gating:
  - network can grow very deep, in time → vanishing gradients.
  - *Critical component*: add pass-through (additive paths) so recurrent state does not go repeatedly through squashing non-linearity.



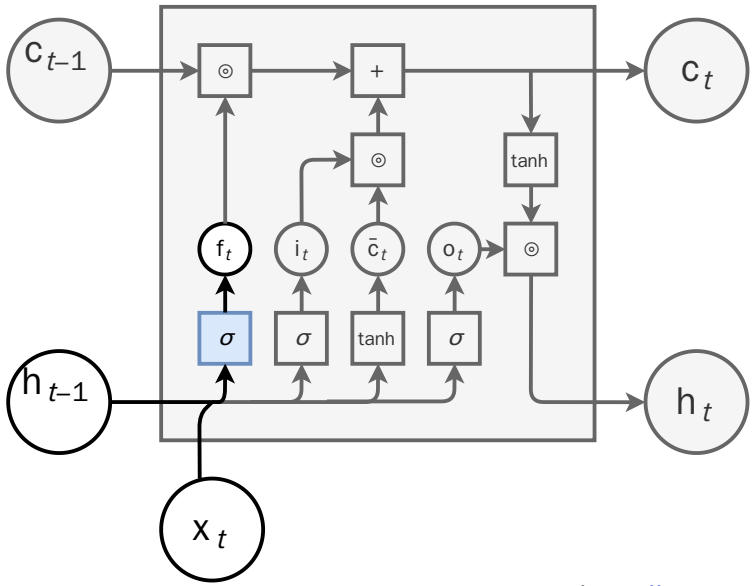
Credit: Gilles Louppe

Long Short Term Memory (LSTM)

- Gating:
  - network can grow very deep, in time → vanishing gradients.
  - *Critical component*: add pass-through (additive paths) so recurrent state does not go repeatedly through squashing non-linearity.



- LSTM:
  - Add internal state separate from output state
  - Add input, output, and forget gating

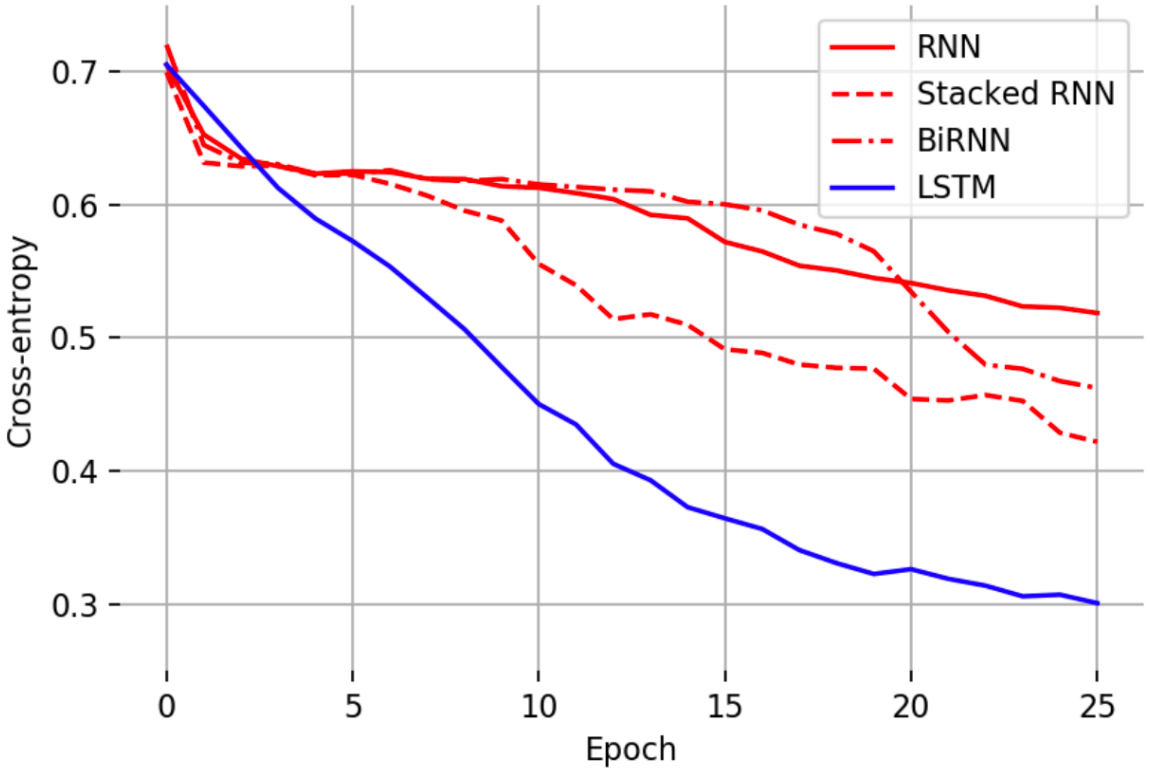


Credit: Gilles Louppe

Comparison on Toy Problem

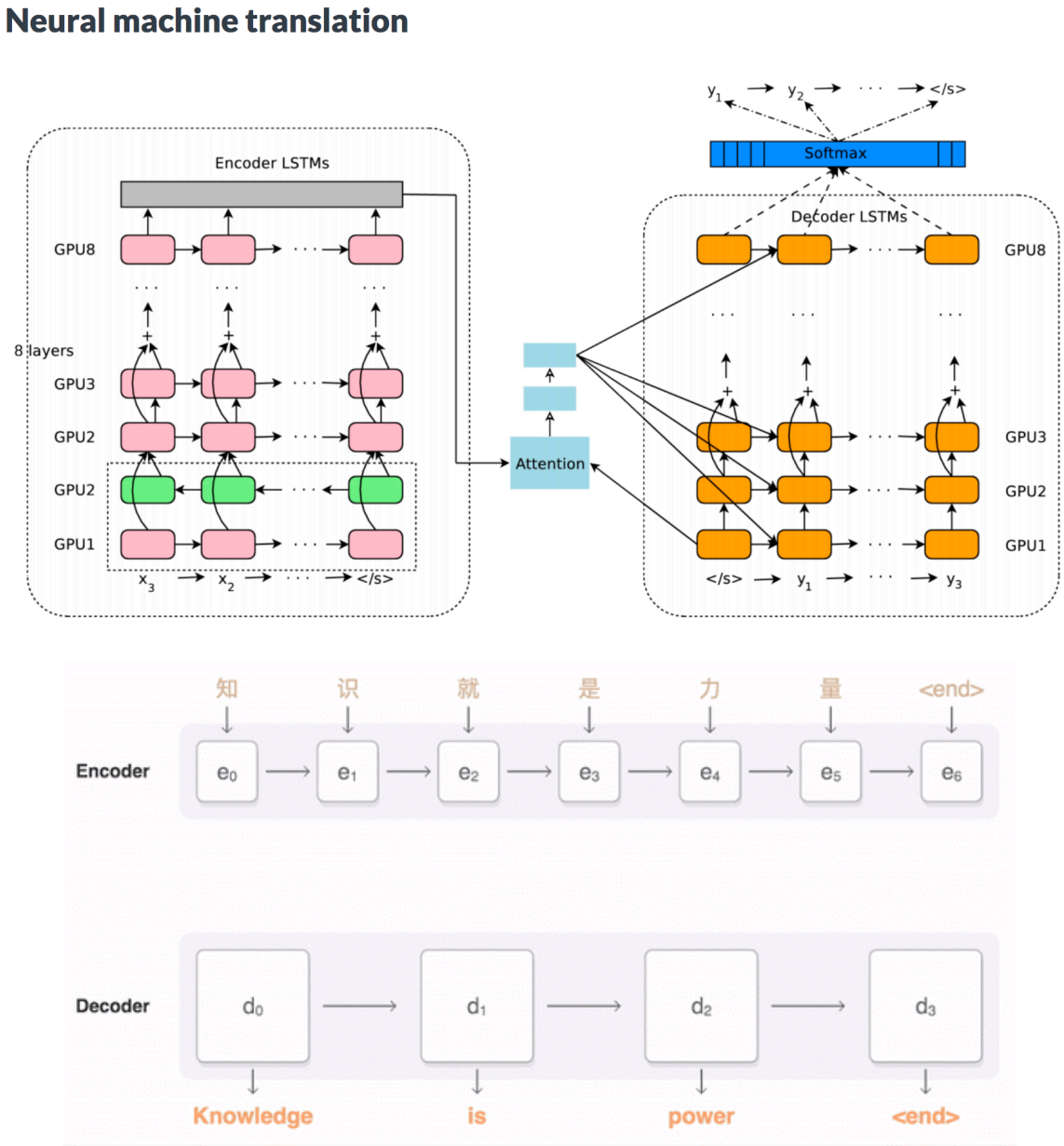
Learn to recognize palindrome
Sequence size between 1 to 10

$\mathbf{x}$	y
$(1, 2, 3, 2, 1)$	1
$(2, 1, 2)$	1
$(3, 4, 1, 2)$	0
(0)	1
$(1, 4)$	0



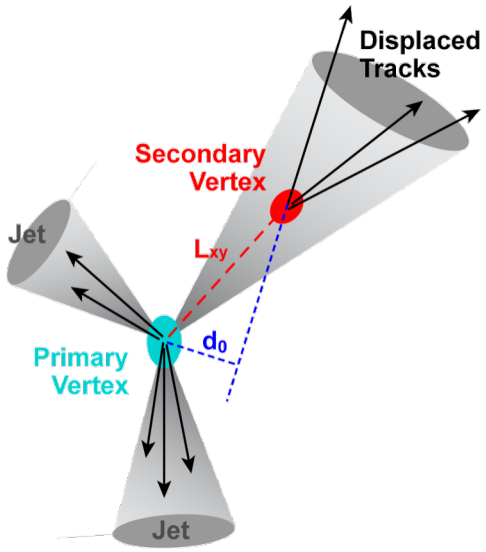
Slide credit: G. Louppe

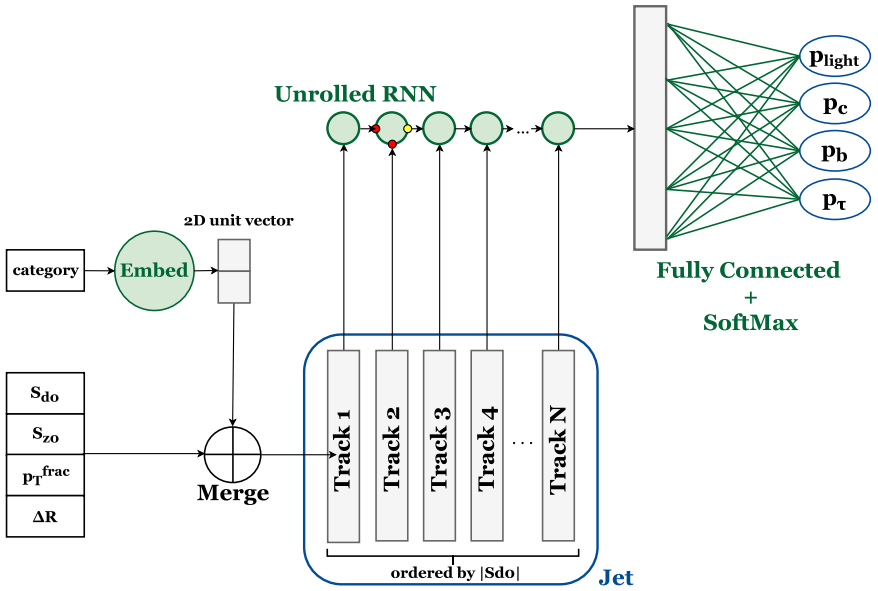
Examples

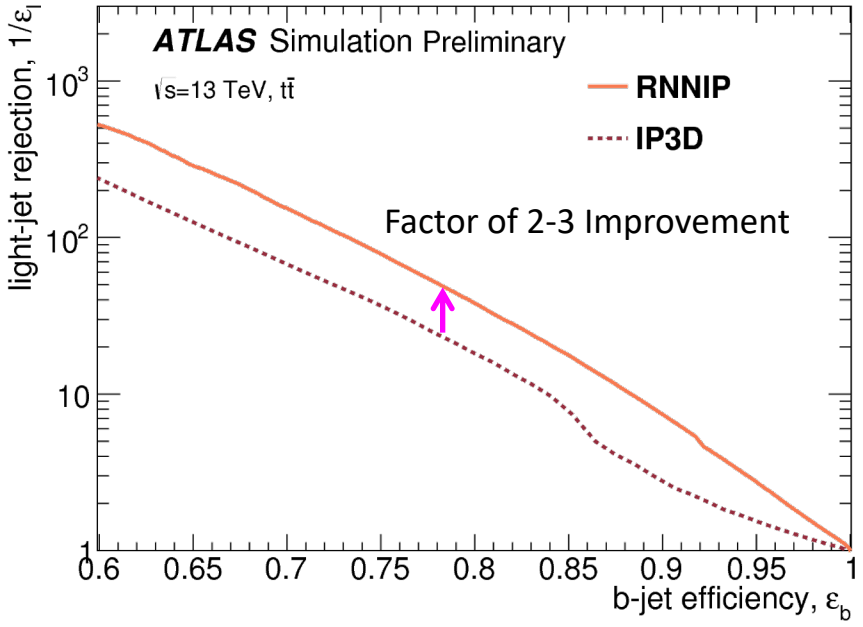


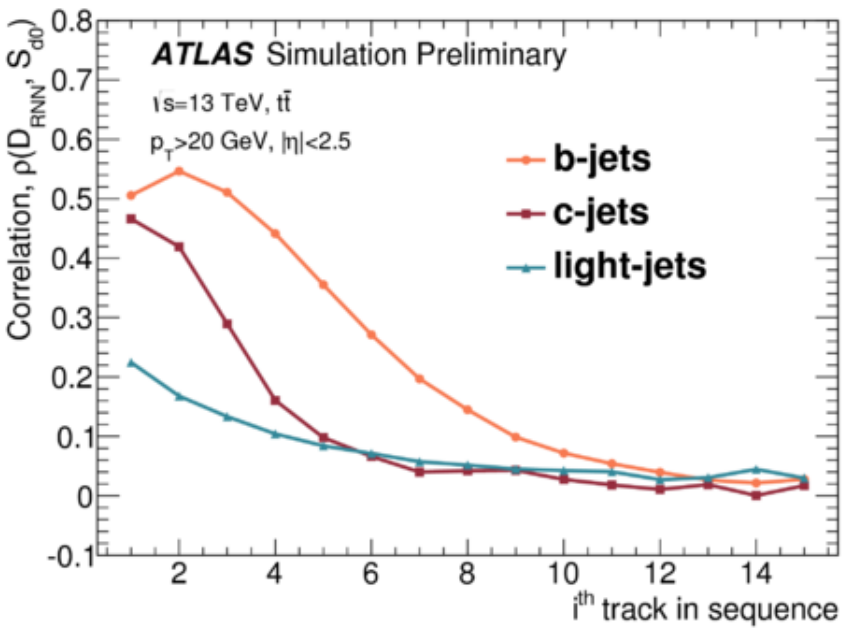
Y. Wu et al, 2016

RNNs for b-tagging



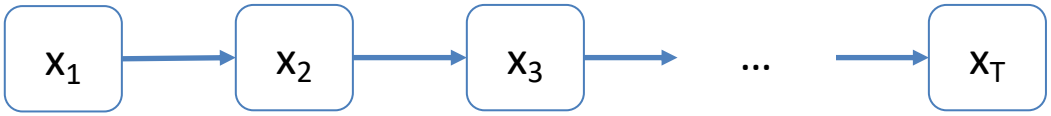






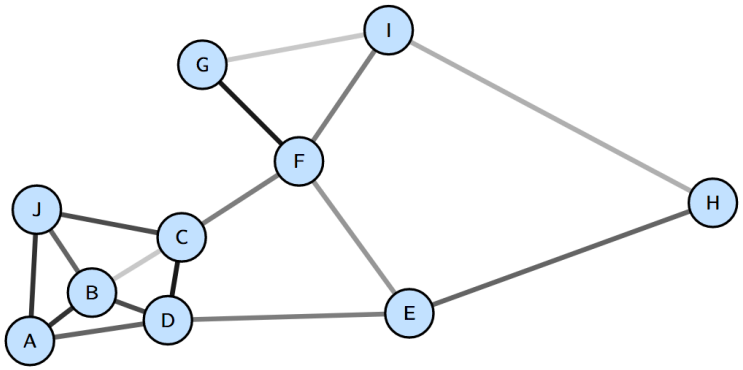
Graph Data

Graph Data



- Sequential data has single (directed) connections from data at current time to data at next time

- What about data with more complex dependencies



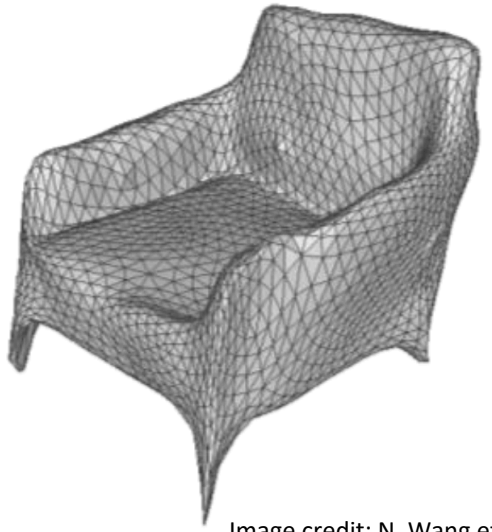
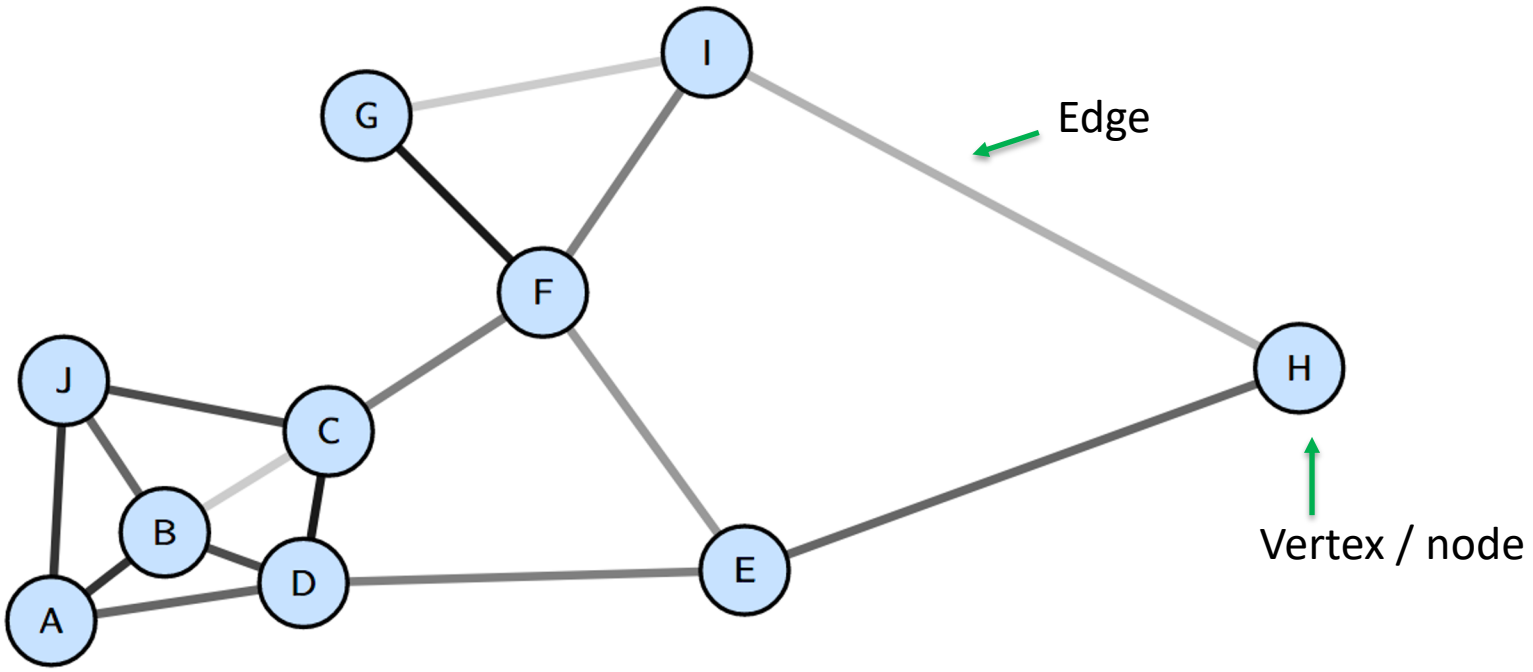


Image Credit: I. Henrion

Image credit: N. Wang et al., 2018

Graphs



- Adjacency matrix: $A_{ij} = \delta(edge\ between\ vertex\ i\ and\ j)$
- Each node can have features
- Each edge can have features, e.g. distance between nodes

Image Credit: I. Henrion

Neural Message Passing

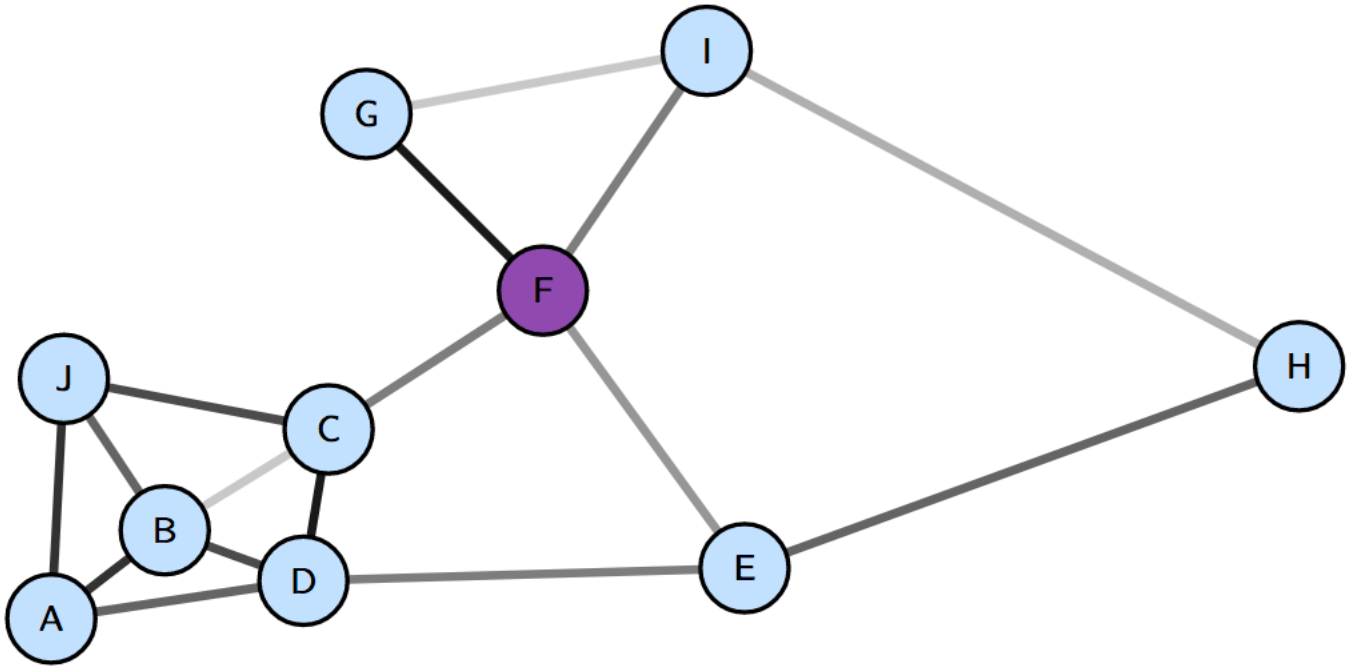
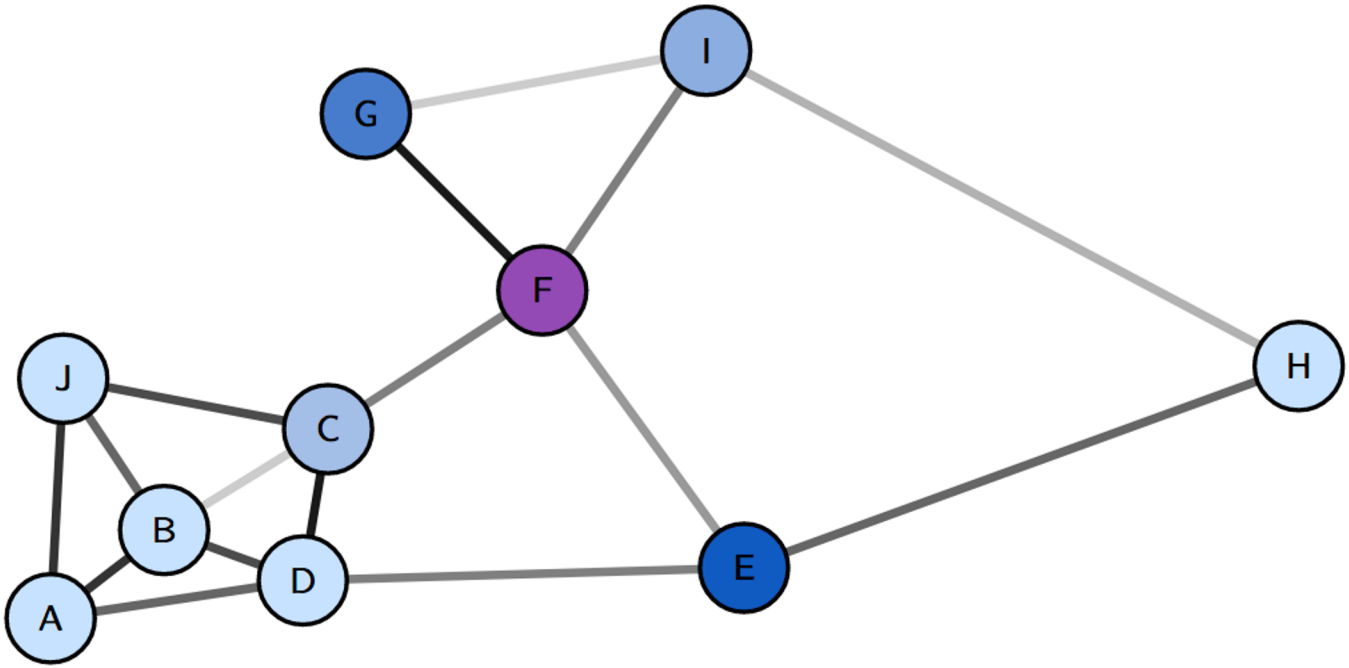


Image Credit: I. Henrion

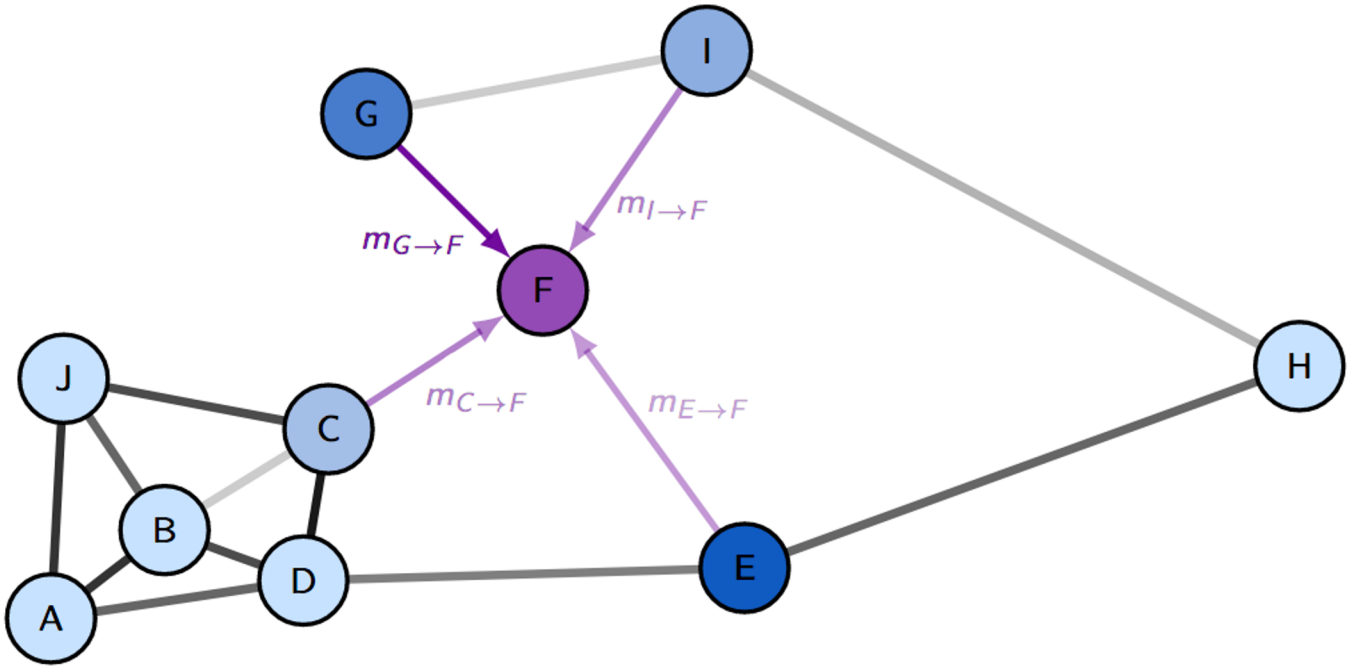
Neural Message Passing



$$\tilde{m}_j^t = f(h_j^{t-1})$$

Image Credit: I. Henrion

Neural Message Passing

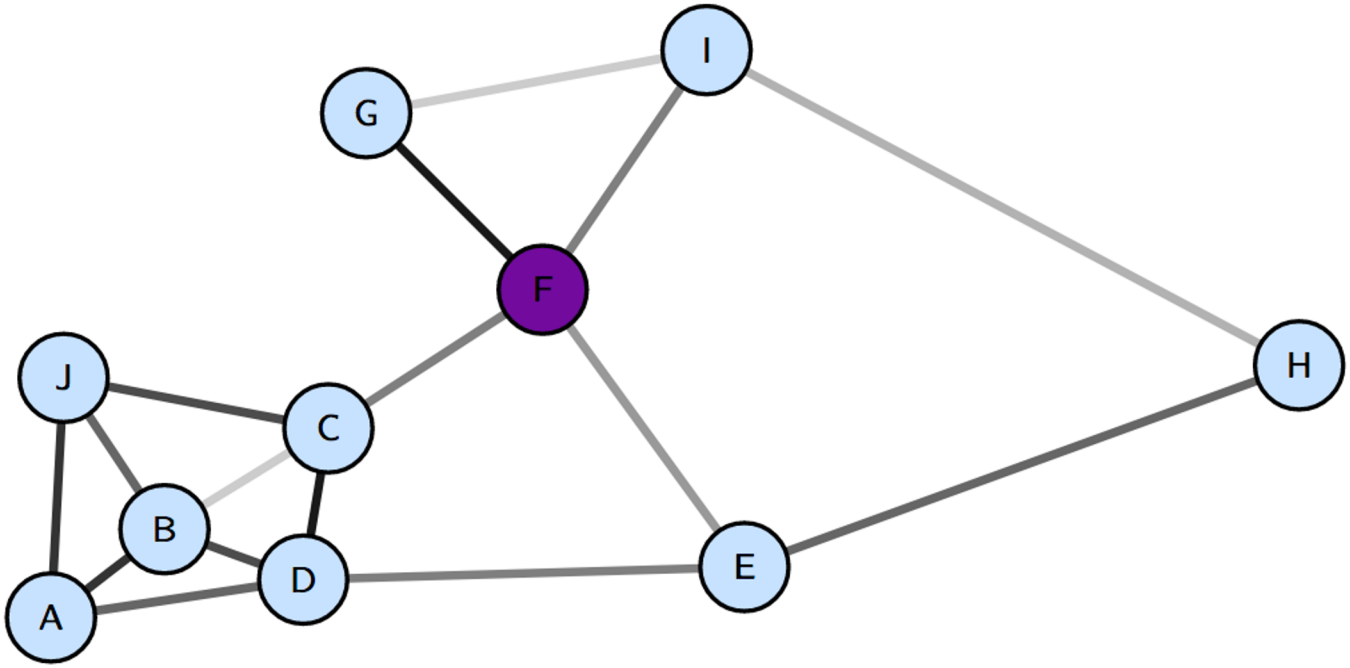


$$\tilde{m}_j^t = f(h_j^{t-1})$$

$$m_{j \to i}^t = \sigma(A_{ij} \tilde{m}_j^t)$$

Image Credit: I. Henrion

Neural Message Passing



$$\tilde{m}_j^t = f(h_j^{t-1})$$
$$m_{j\to i}^t = \sigma(A_{ij}\tilde{m}_j^t)$$
$$h_i^t = \mathrm{GRU}(h_i^{t-1}, \Sigma_j m_{j\to i}^t)$$

Image Credit: I. Henrion

Examples

Learning to simulate physics with graph networks

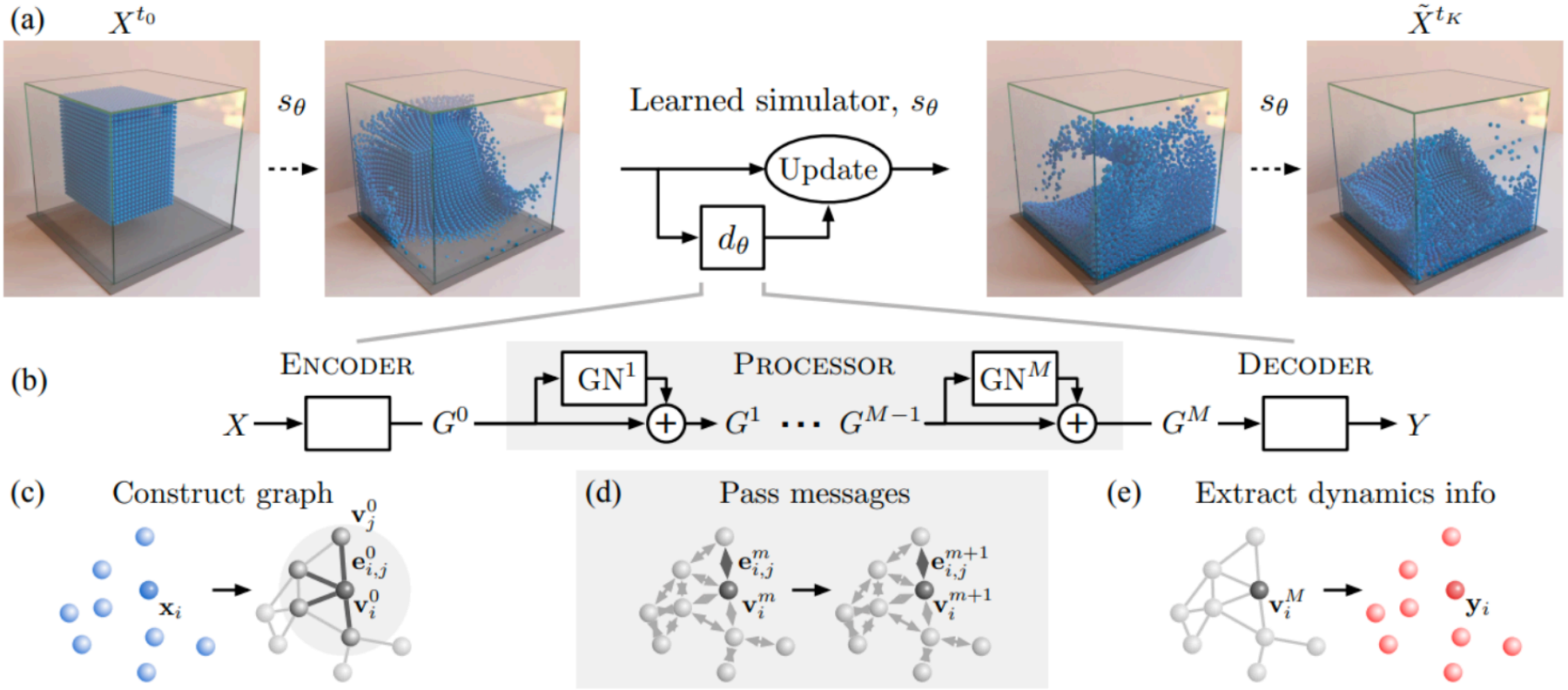
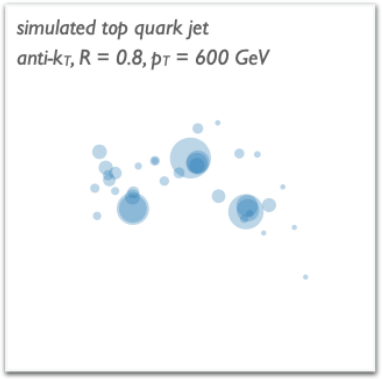
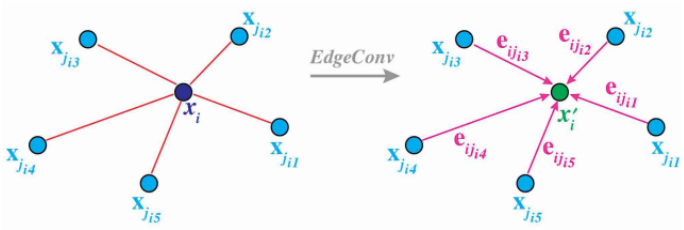


Figure 2. **(a)** Our GNS predicts future states represented as particles using its learned dynamics model, d_θ, and a fixed update procedure. **(b)** The d_θ uses an "encode-process-decode" scheme, which computes dynamics information, Y, from input state, X. **(c)** The ENCODER constructs latent graph, G^0, from the input state, X. **(d)** The PROCESSOR performs M rounds of learned message-passing over the latent graphs, $G^0, \ldots, G^M$. **(e)** The DECODER extracts dynamics information, Y, from the final latent graph, G^M.

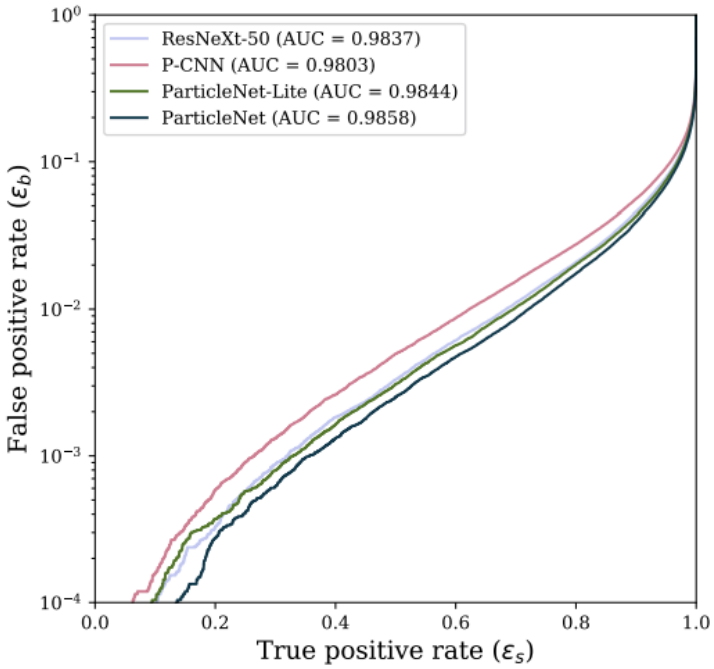
Sanchez-Gonzalez et al. 2020

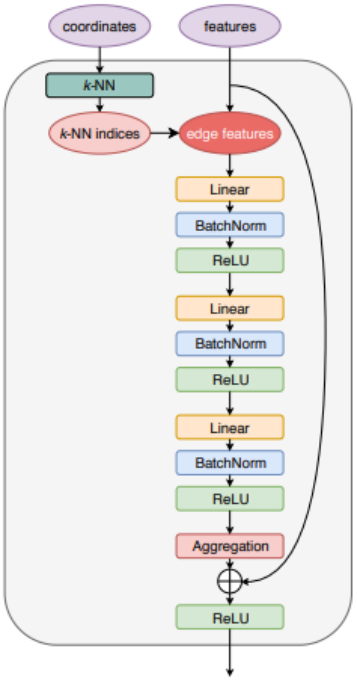
GNN for Jet Tagging



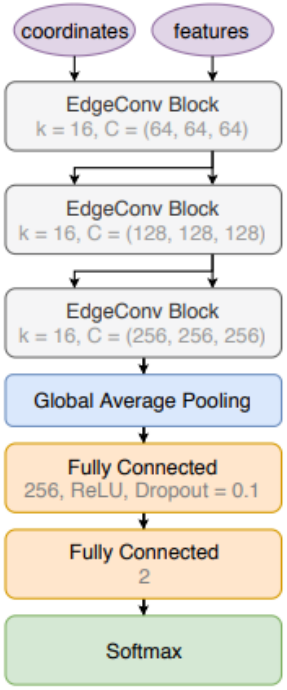


Top Jets vs QCD dijets

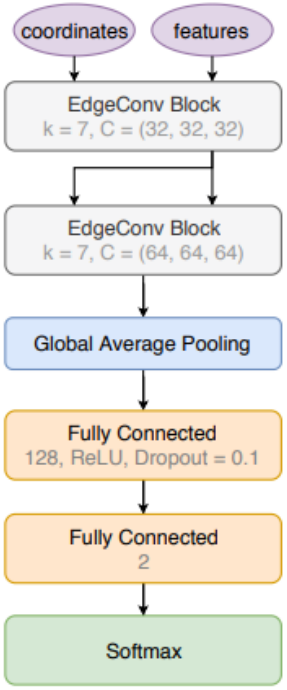




EdgeConv Block



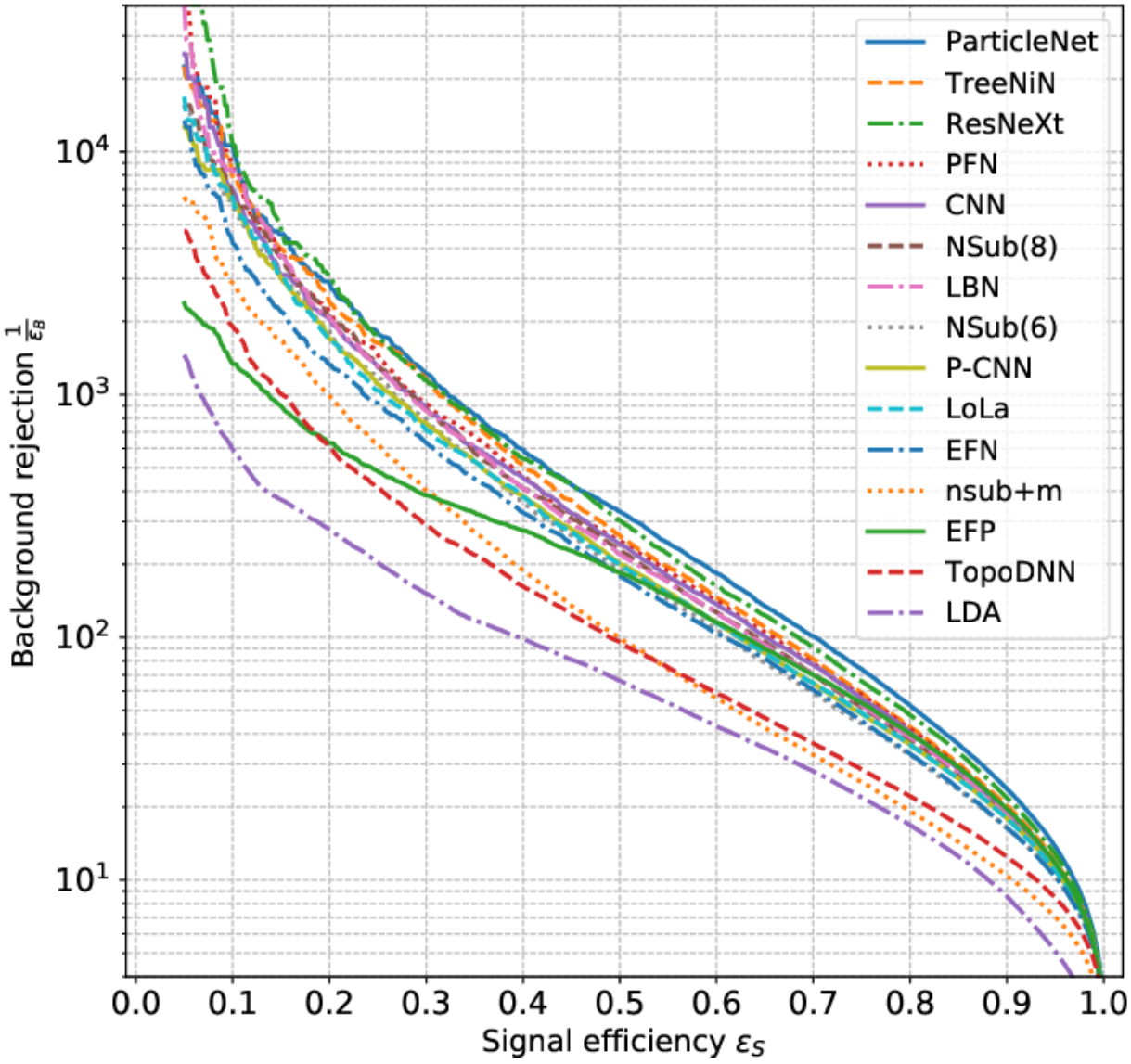
(a) ParticleNet



(b) ParticleNet-Lite

Phys. Rev. D 101, 056019 (2020)

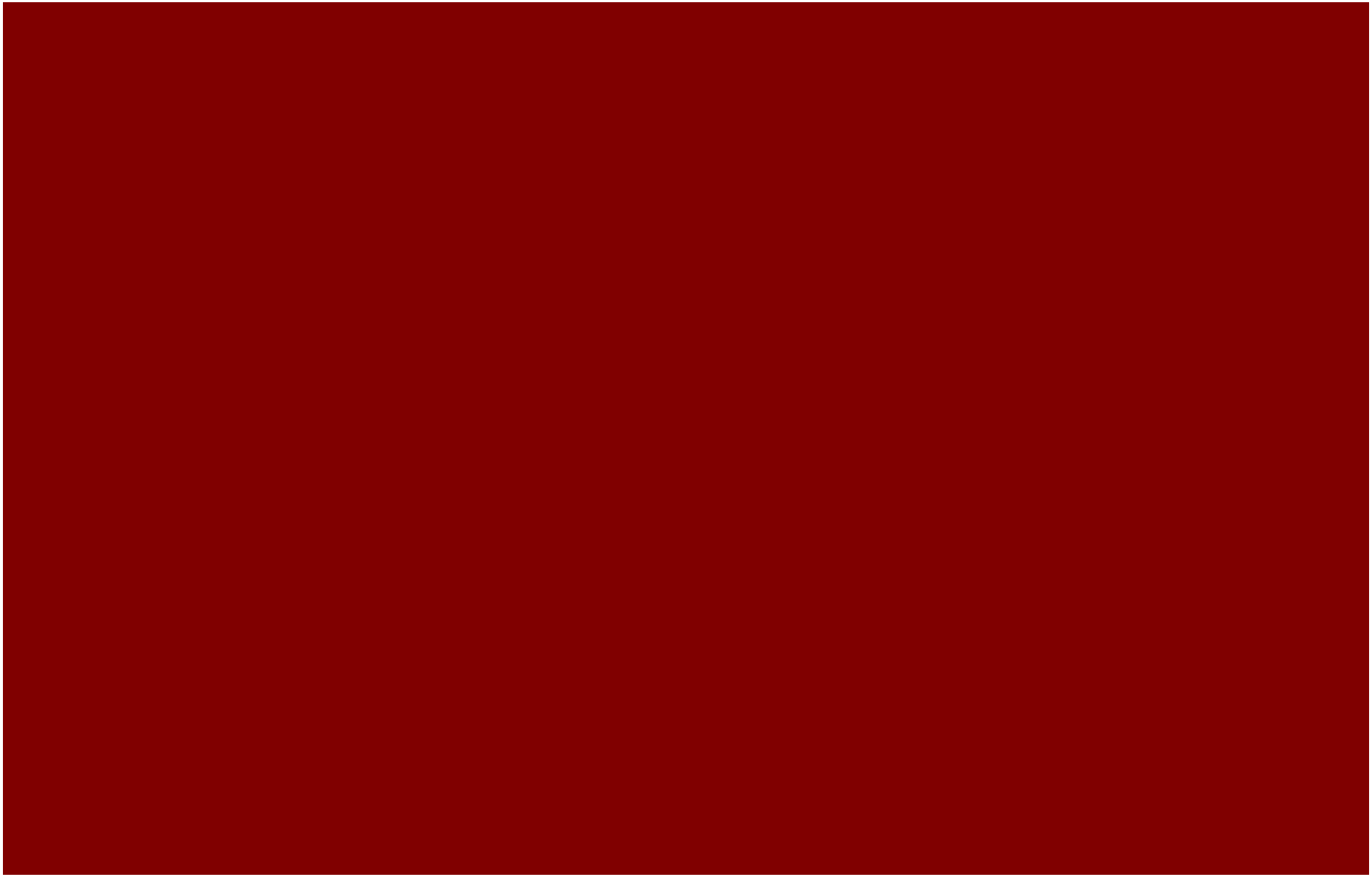
Comparing Methods for Jet Tagging



SciPost Phys. 7, 014 (2019)

Summary

- Neural Networks allow us to combine non-linear basis selection with feature learning
  - Care needed to train them and ensure they don't overfit
- Deep neural networks allow us to learn complex function by hierarchically structuring the feature learning
- We can use our inductive bias (knowledge) to define models that are well adapted to our problem
- Many neural networks structures are available for training models on a wide array of data types.



Automatic Differentiation

Chain Rule – Symbolic Differentiation Painful!

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(\sigma(h(\mathbf{x}_i))) + (1 - y_i) \ln(1 - \sigma(h(\mathbf{x}_i)))$$

- Derivative of sigmoid: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- Chain rule to compute gradient w.r.t. $\mathbf{w}$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \mathbf{w}} = \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))\sigma(\mathbf{U}\mathbf{x}) + (1 - y_i)\sigma(h(\mathbf{x}))\sigma(\mathbf{U}\mathbf{x}_i)$$

- Chain rule to compute gradient w.r.t. $\mathbf{u}_j$

$$\frac{\partial L}{\partial \mathbf{u}_j} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \sigma}\frac{\partial \sigma}{\partial \mathbf{u}_j} =$$

$$= \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i$$

$$+ (1 - y_i)\sigma(h(\mathbf{x}_i))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i$$

Automatic Differentiation

Problem: Compute gradients of z with respect to inputs $\{x_1, x_2\}$

$$z = \sin(x_1) + x_1 x_2$$

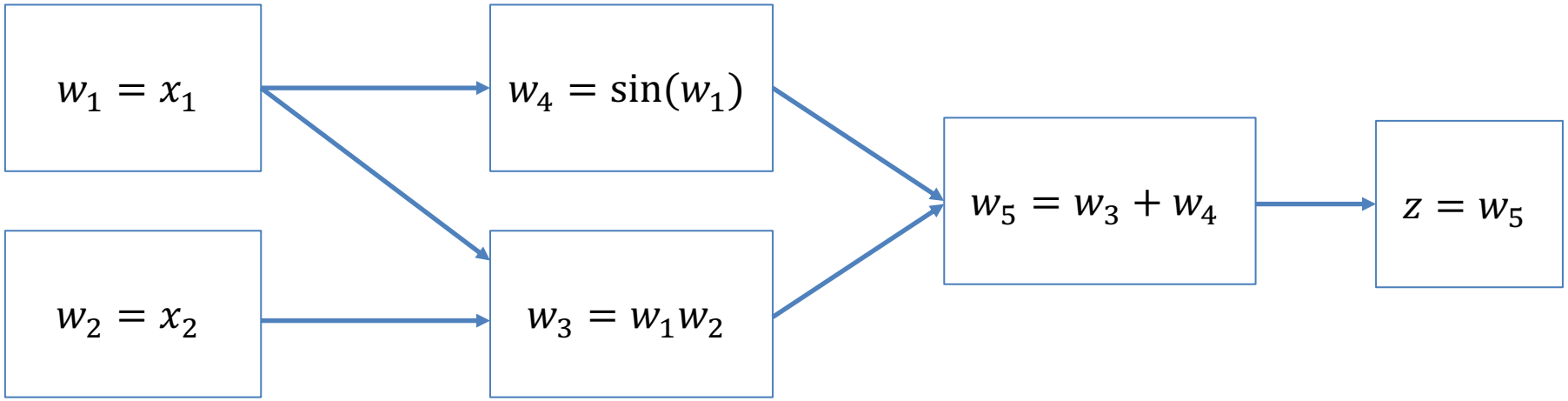
Automatic Differentiation

$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

Problem: Compute gradients of z with respect to inputs $\{x_1, x_2\}$

$$z = \sin(x_1) + x_1 x_2$$

Organize as a computational Graph



Automatic Differentiation

$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

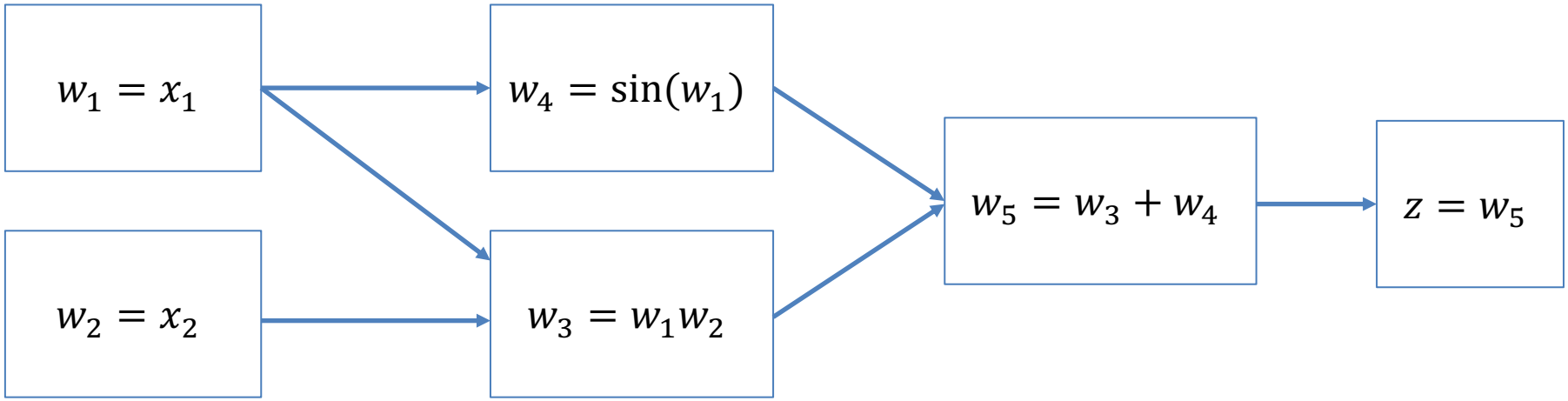
Problem: Compute gradients of z with respect to inputs $\{x_1, x_2\}$

We know the gradients of simple functions: $\sin(x), x * y, x + y \ldots$

Chain rule:

$$\frac{dz}{dw_1} = \sum_{p \in parents} \frac{dz}{dw_p} \frac{dw_p}{dw_i}$$

$\frac{dw_1}{dx_1} = 1$
$\frac{dw_2}{dx_2} = 1$
$\frac{dw_3}{dw_1} = w_2$ $\frac{dw_3}{dw_2} = w_1$
$\frac{dw_4}{dw_1} = \cos(w_1)$
$\frac{dw_5}{dw_3} = 1$ $\frac{dw_5}{dw_4} = 1$



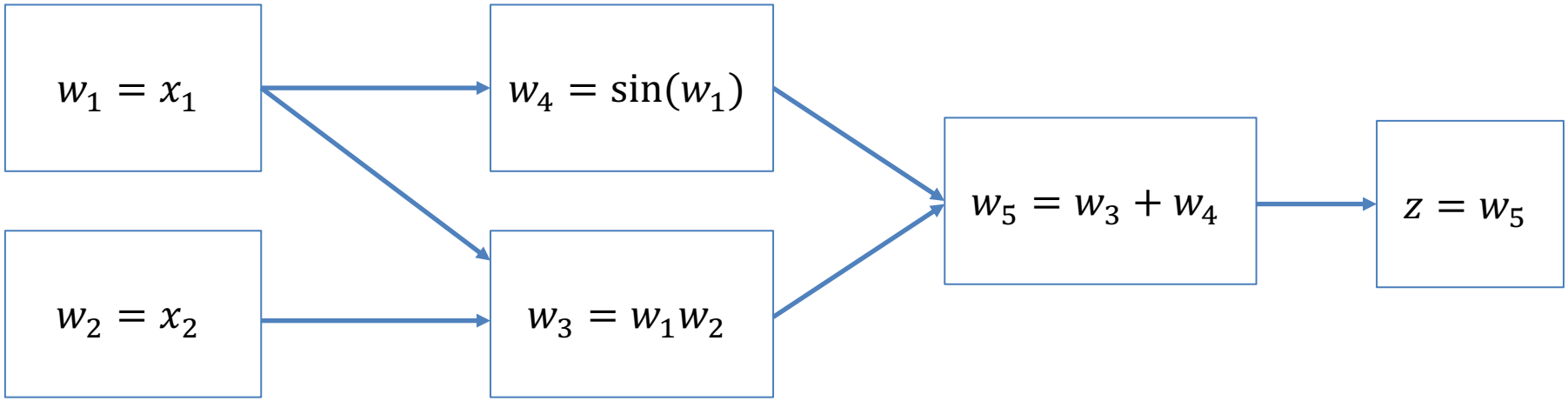
Automatic Differentiation

$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

Problem: Compute gradients of z with respect to inputs $\{x_1, x_2\}$

NOT going to find analytic derivative

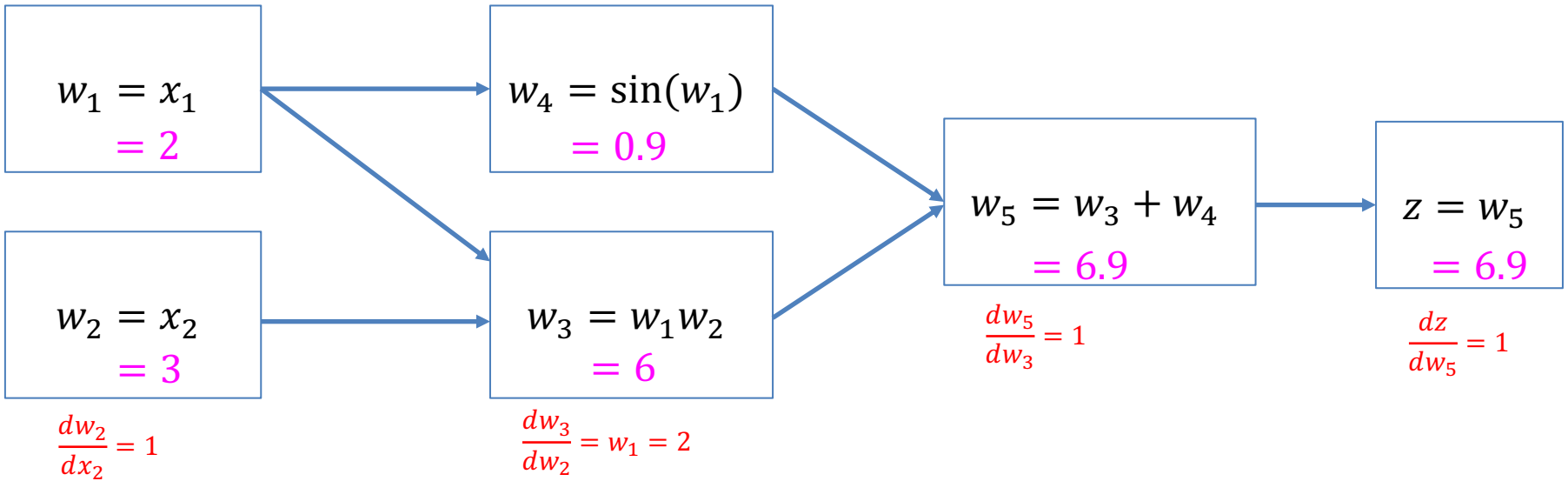
WILL find a way to compute value of gradient for a given input point



Forward Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each input, from input to output sequentially, evaluate graph and gradients and store values



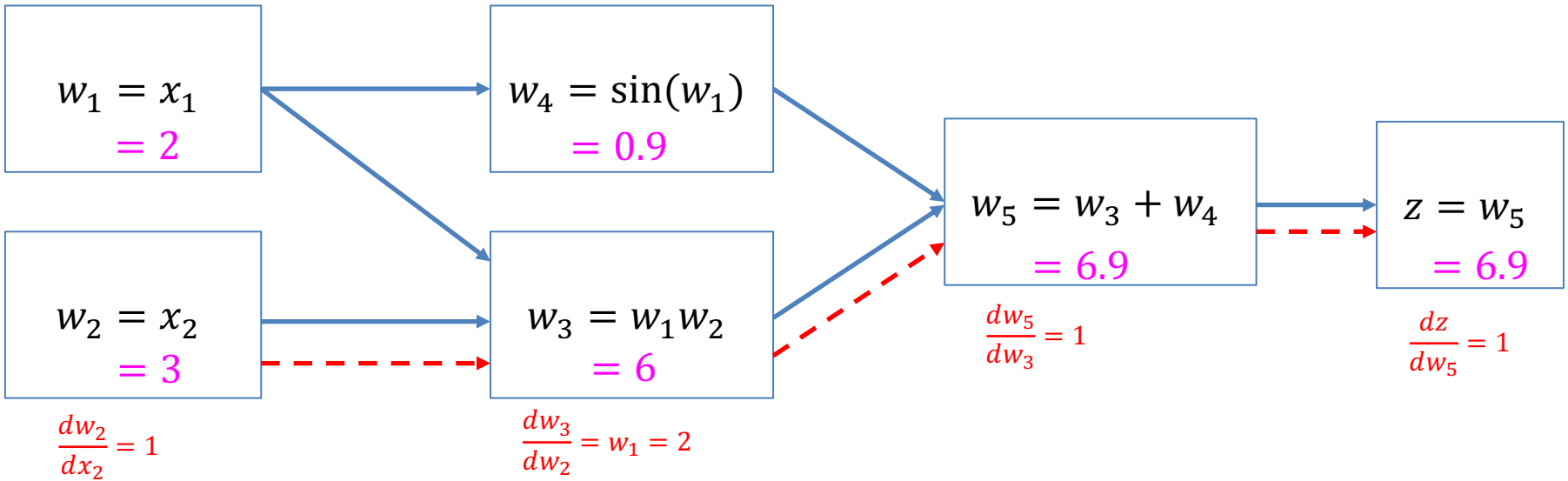
Forward Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each input, from input to output sequentially, evaluate graph and gradients and store values

Apply chain rule with multiplication

$$\frac{dz}{dx_2} = \frac{dw_2}{dx_2}\frac{dw_3}{dw_2}\frac{dw_5}{dw_3}\frac{dz}{dw_5} = 1 * 2 * 1 * 1 = 2$$



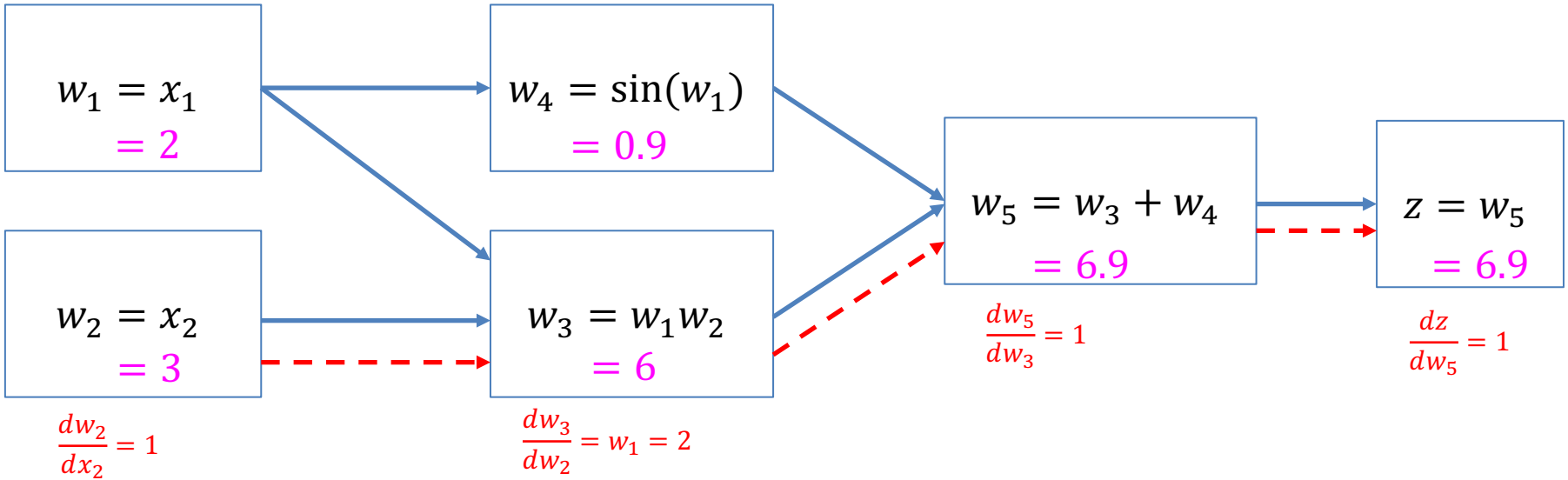
Forward Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Forward Mode allows us to compute the gradient of one input with respect to all the output

$$\text{Jacobian } \frac{dz}{dx} = \begin{pmatrix} \frac{dz_1}{dx_1} & \cdots & \frac{dz_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{dz_1}{dx_N} & \cdots & \frac{dz_M}{dx_N} \end{pmatrix}$$

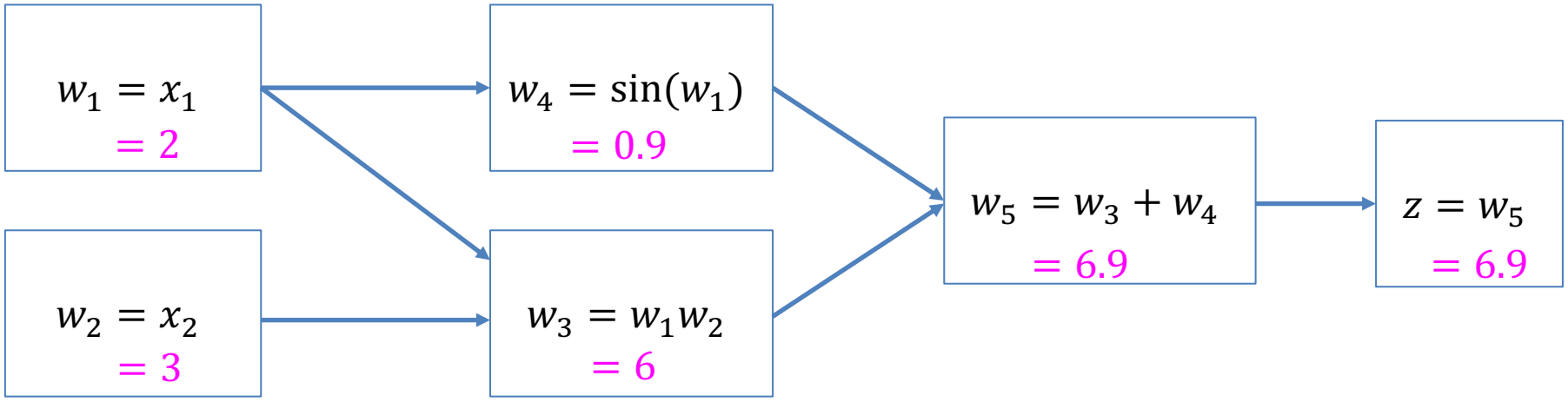
If we have 1 output (Loss) and many inputs → SLOW!



Reverse Mode Automatic Differentiation

$$w_1 = x_1 = 2$$
$$w_2 = x_2 = 3$$
$$w_3 = w_1 w_2 = 6$$
$$w_4 = \sin(w_1) = 0.9$$
$$w_5 = w_3 + w_4 = 6.9$$
$$z = w_5$$

Evaluate graph and store values

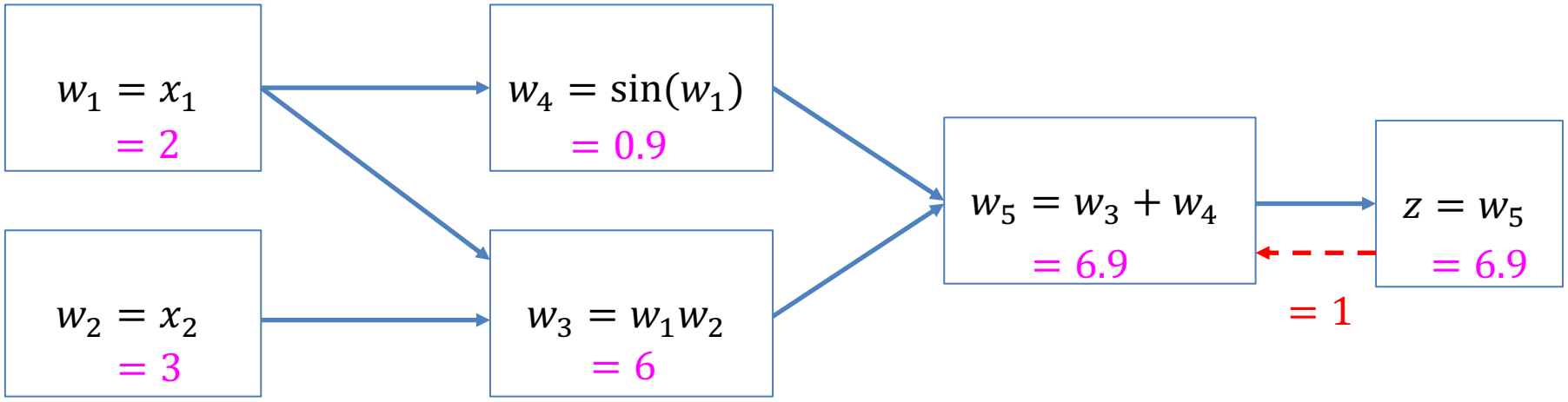


Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$



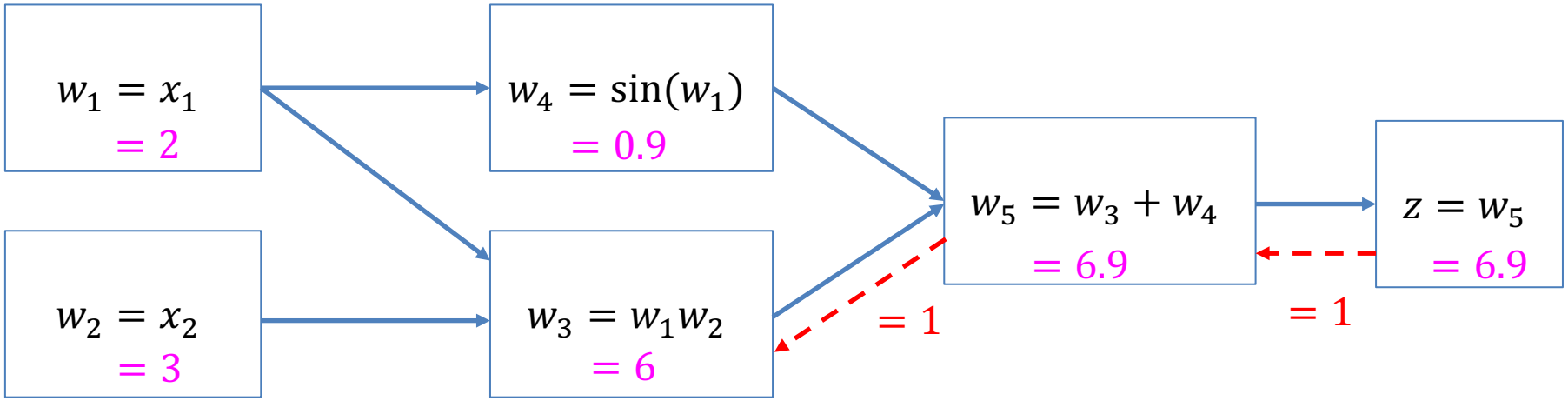
Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$

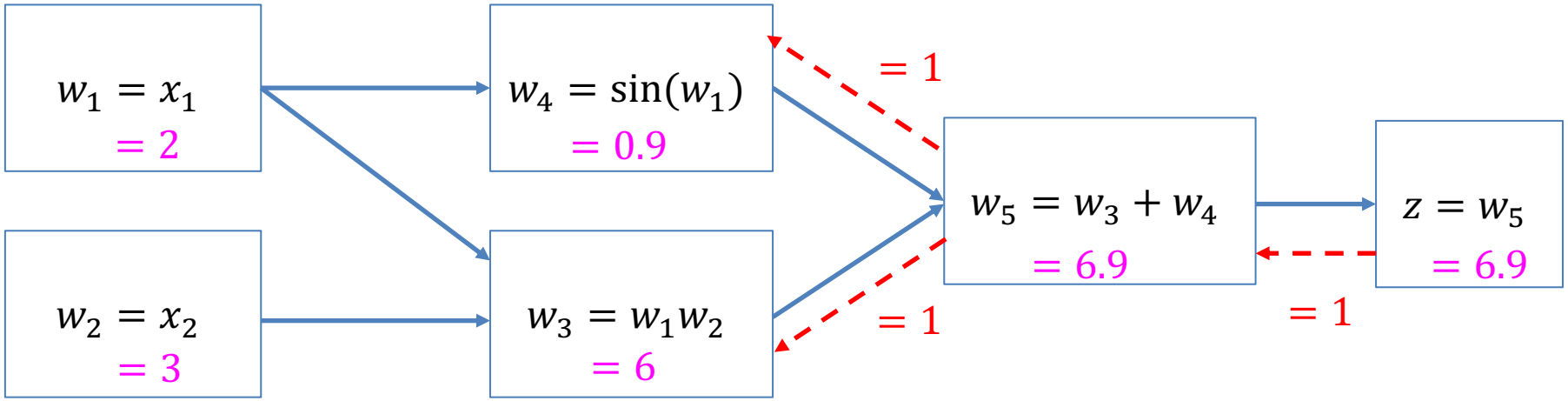


Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$
$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$
$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$



Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

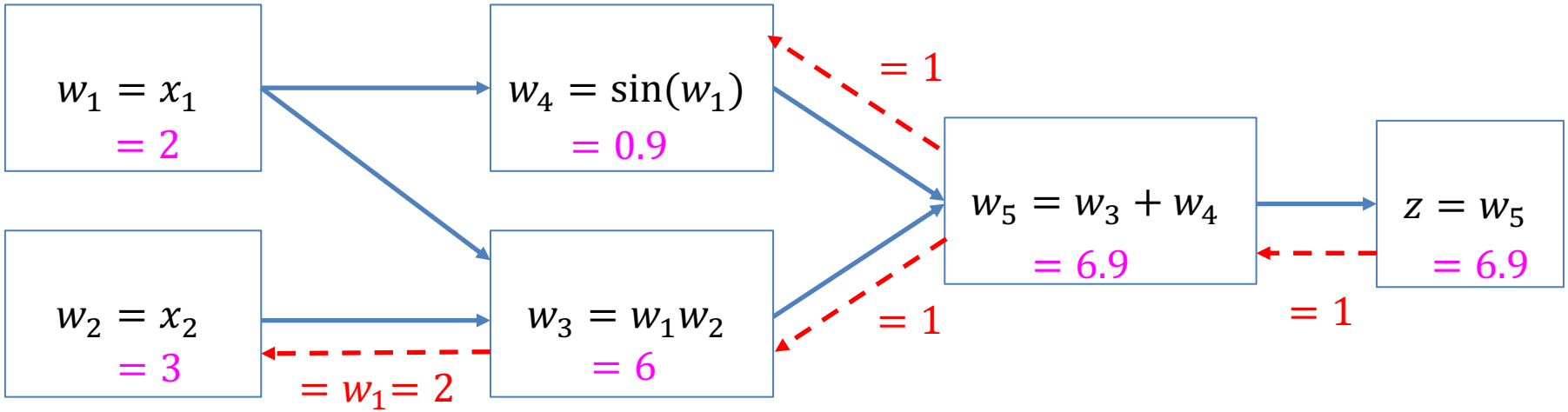
Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$

$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$

$$\frac{dz}{dw_2} = \frac{dz}{dw_3}\frac{dw_3}{dw_2} = 1 \times w_1 = w_1 = 2$$



Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

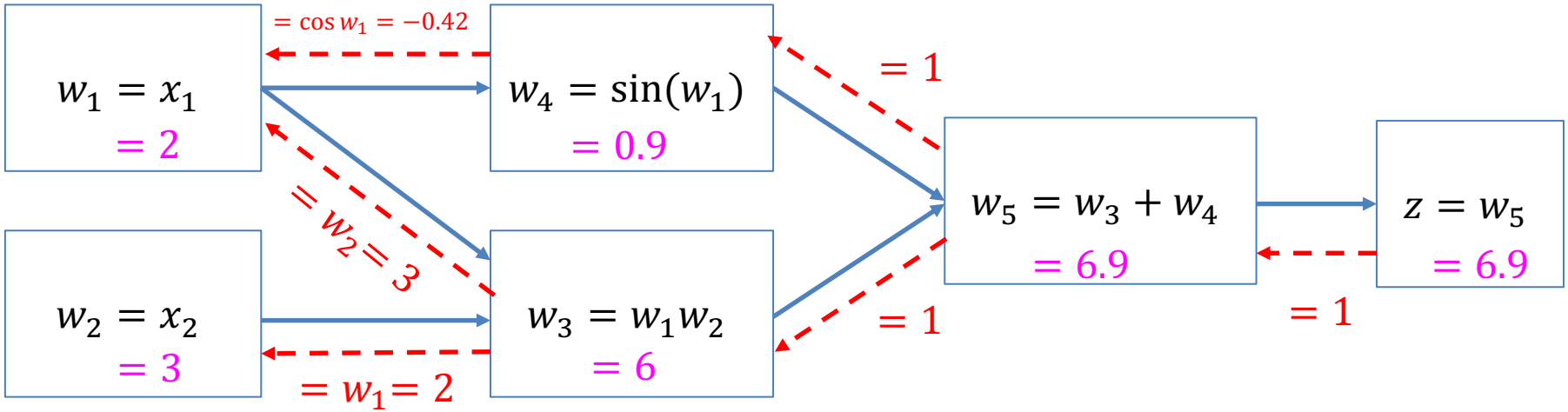
$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$

$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$

$$\frac{dz}{dw_2} = \frac{dz}{dw_3}\frac{dw_3}{dw_2} = 1 \times w_1 = w_1 = 2$$

$$\frac{dz}{dw_1} = \frac{dz}{dw_4}\frac{dw_4}{dw_1} + \frac{dz}{dw_3}\frac{dw_3}{dw_1}$$
$$= \cos(w_1) + w_2 = \cos(2) + 3$$
$$= 2.58$$

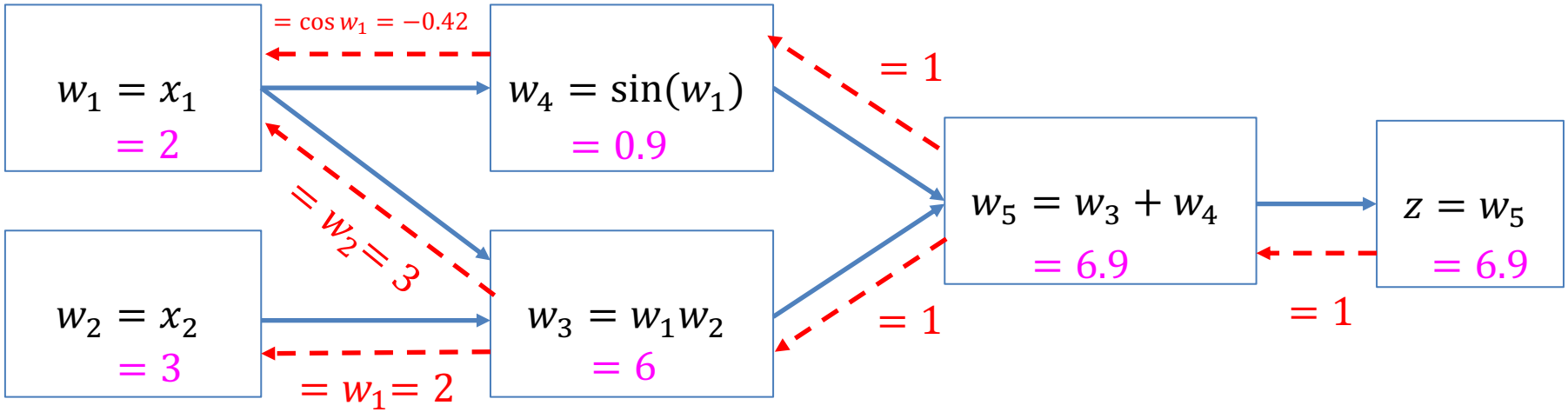


Reverse Mode Automatic Differentiation

$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each output, can compute the gradient w.r.t. all inputs in one pass!

$$\text{Jacobian } \frac{dz}{dx} = \begin{pmatrix} \frac{dz_1}{dx_1} & \cdots & \frac{dz_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{dz_1}{dx_N} & \cdots & \frac{dz_M}{dx_N} \end{pmatrix}$$



Neural Network Zoo – "Optimization" Perspective

Neural Network Zoo – "Optimization" Perspective

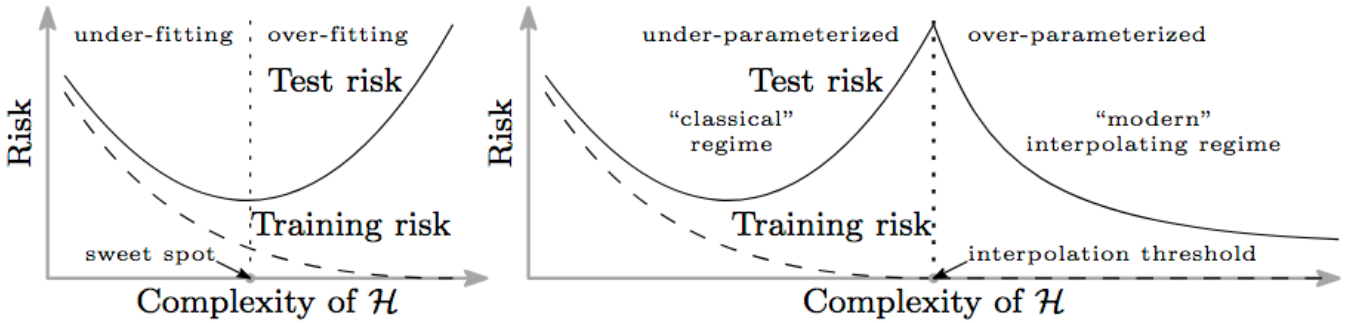
- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

Neural Network Zoo – "Optimization" Perspective

- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

Belkin et. al. 2018



(a) U-shaped "bias-variance" risk curve

(b) "double descent" risk curve

Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high complexity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Fleuret, Deep Learning Course

Neural Network Zoo – "Optimization" Perspective

- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

  - But we must control that:
    - Gradients don't vanish
    - Gradient amplitude is homogeneous across network
    - Gradients are under control when weights change

Neural Network Zoo – "Optimization" Perspective

- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

- Major part of deep learning is **trying to choose the right function…**

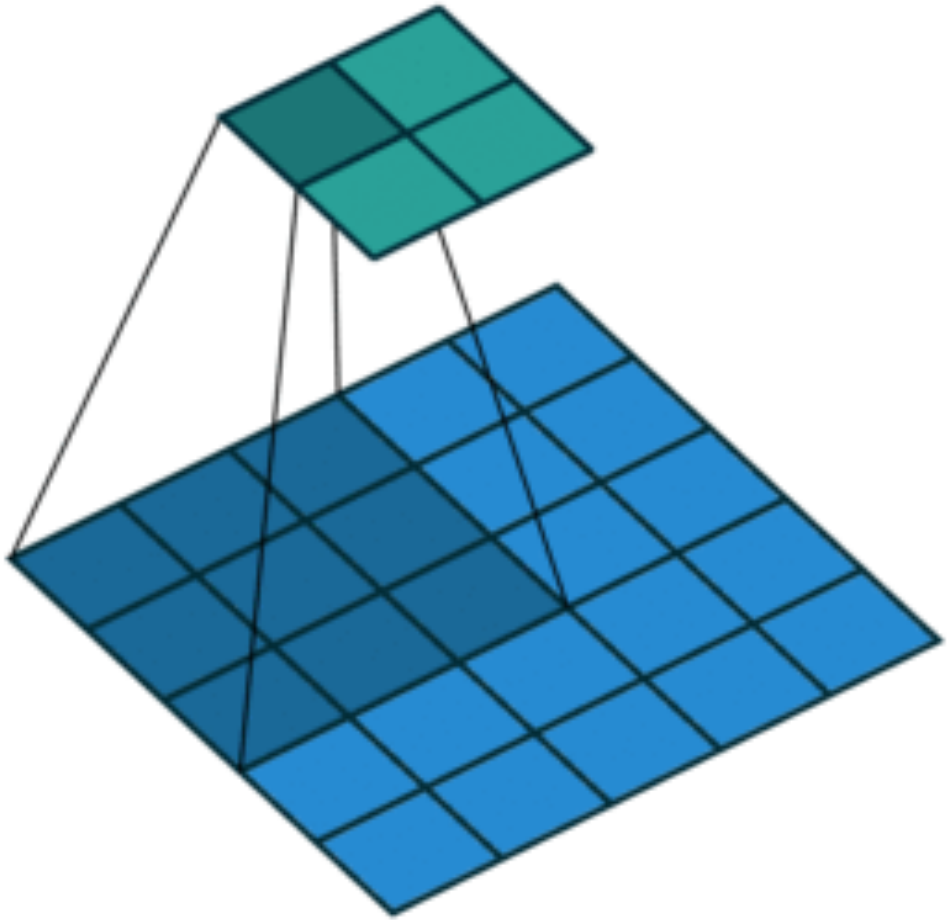
  … instead of trying to improve training with regularization and new optimizers

  - Need to **make gradient descent work**, even at the cost of a substantially engineering the model

Fleuret, Deep Learning Course

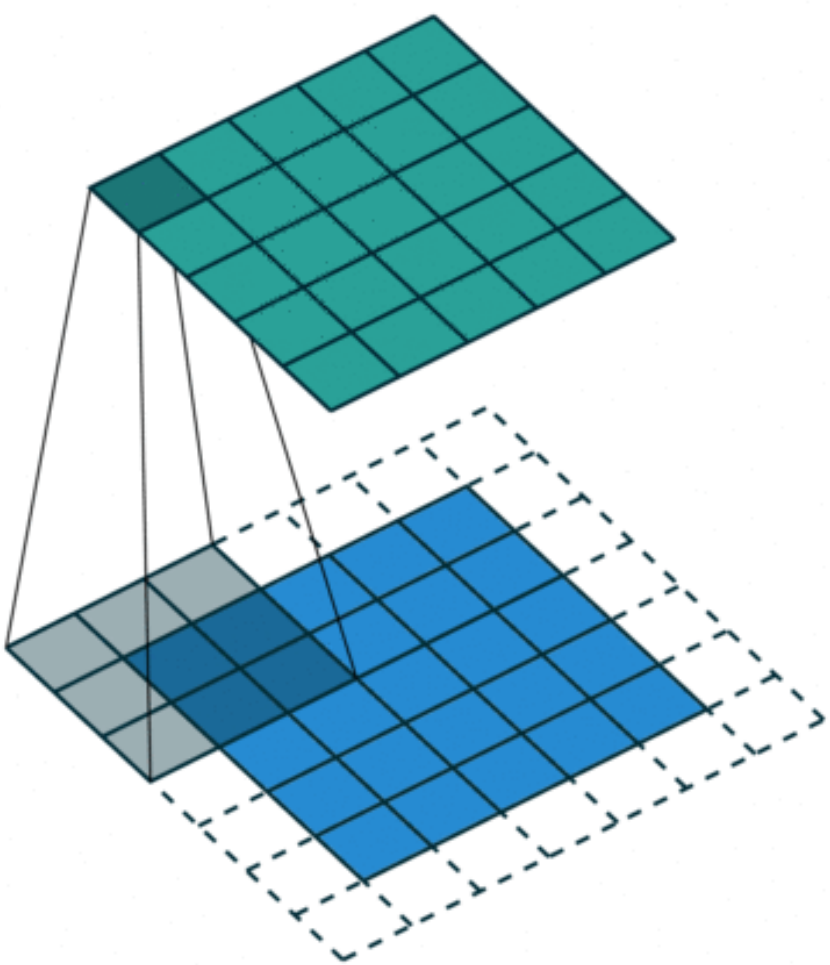
CNN Details

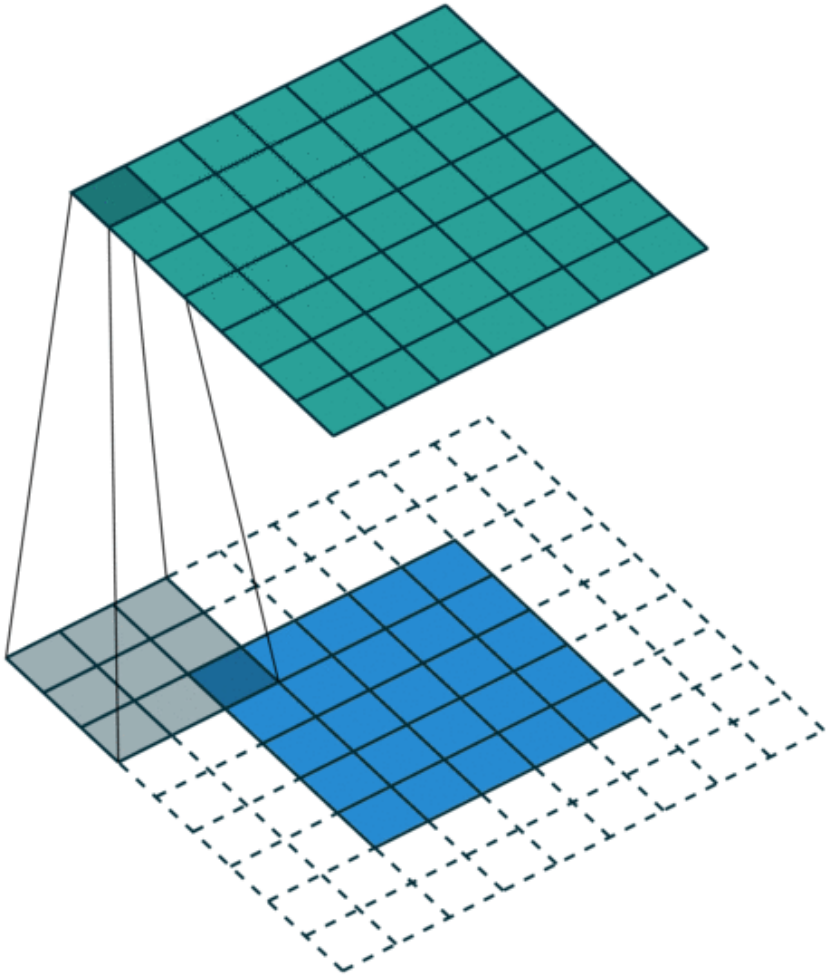
Stride – Step Size When Moving Kernel Across Input



Fleuret, Deep Learning Course

Padding – Size of Zero Frame Around Input





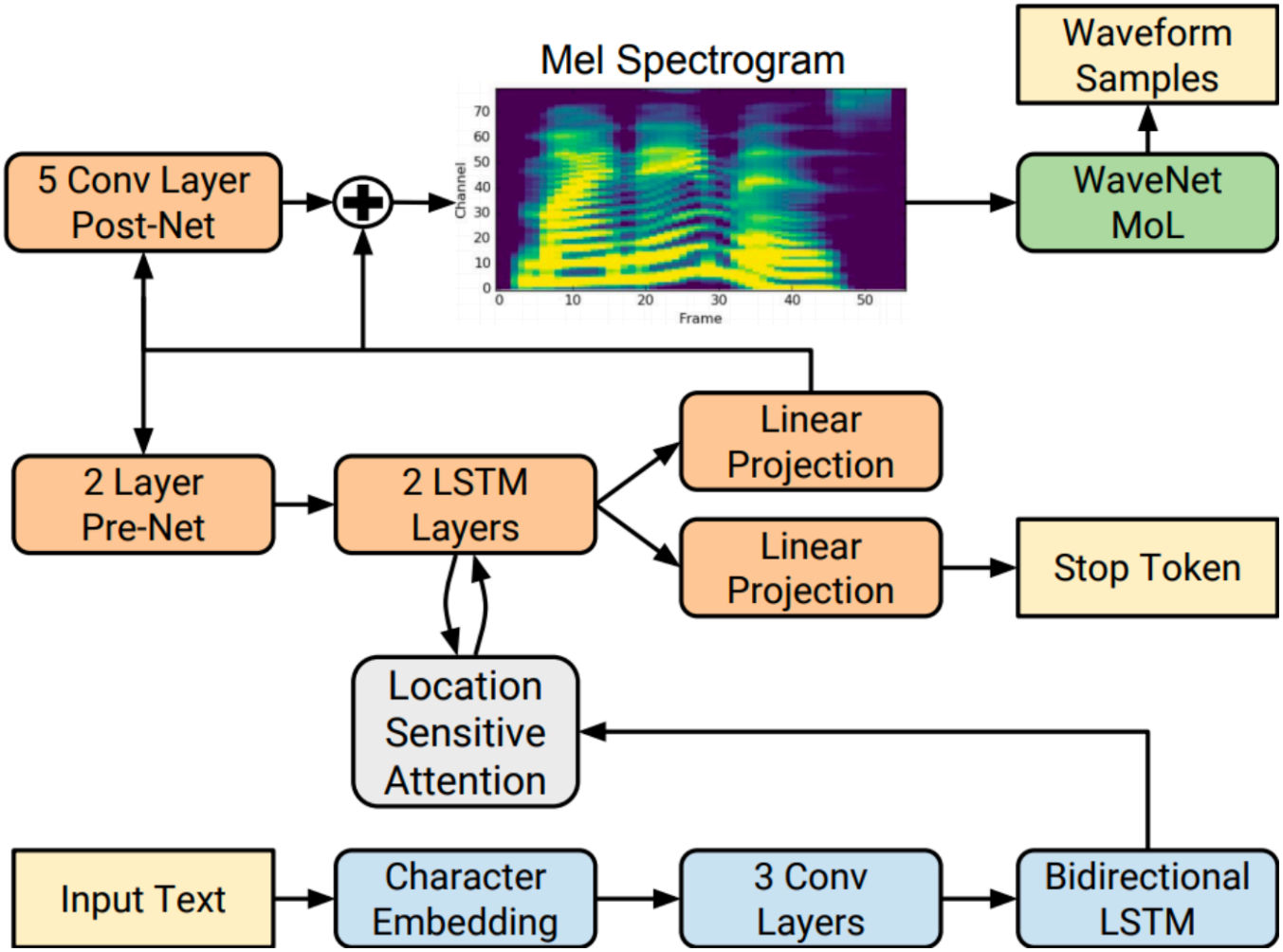
Fleuret, Deep Learning Course

Examples

Self-driving Mario Kart with RNN: YouTube video

Examples

Text-to-speech synthesis



Shen et al., 2017

Neural Message Passing

Algorithm 1 Message passing neural network

Require: $N \times D$ nodes $\mathbf{x}$, adjacency matrix A

$\mathbf{h} \leftarrow \text{Embed}(\mathbf{x})$

for $t = 1, \ldots, T$ **do**

$\mathbf{m} \leftarrow \text{Message}(A, \mathbf{h})$

$\mathbf{h} \leftarrow \text{VertexUpdate}(\mathbf{h}, \mathbf{m})$

end for

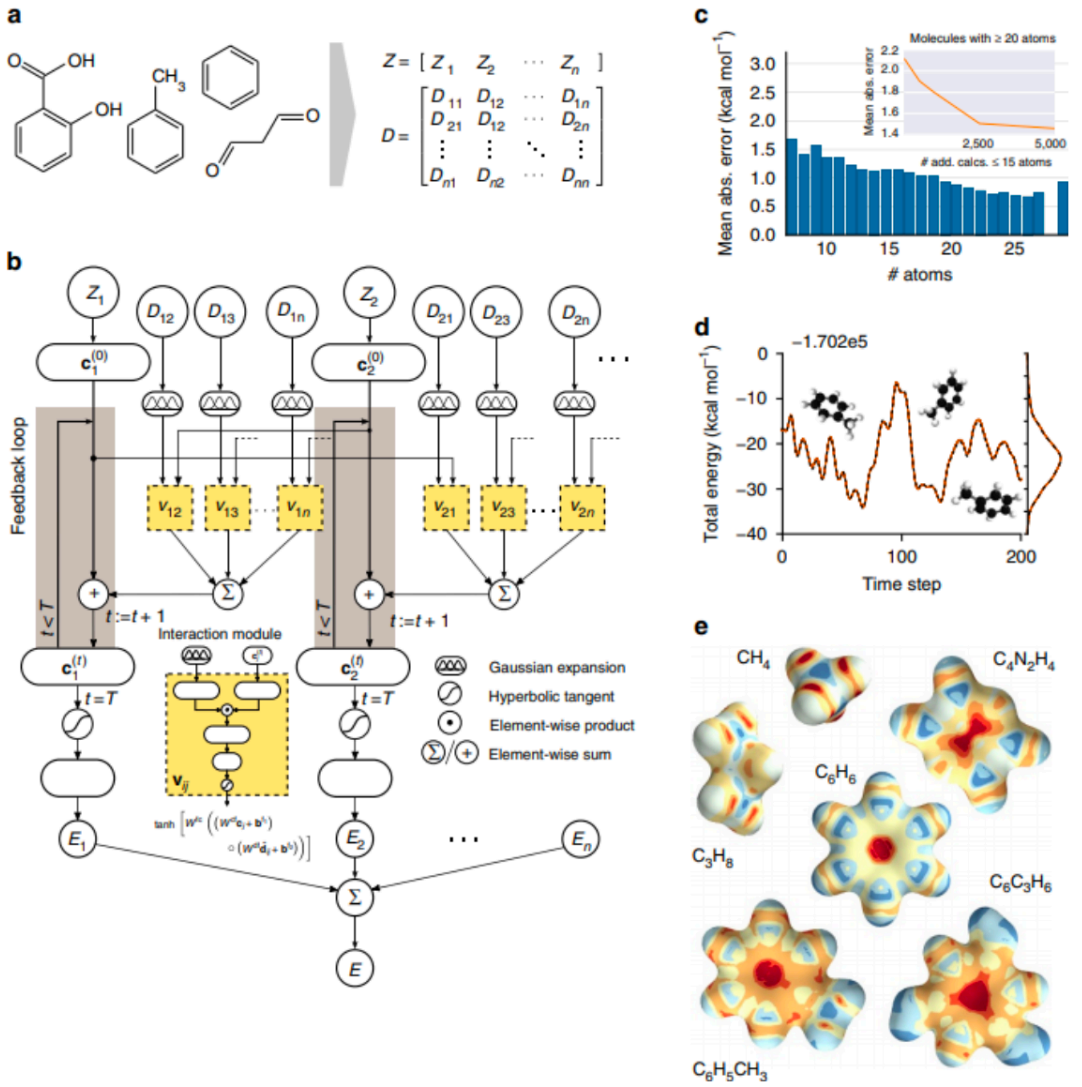
$\mathbf{r} = \text{Readout}(\mathbf{h})$

return $\text{Classify}(\mathbf{r})$

Image Credit: I. Henrion

Examples

Quantum chemistry with graph networks



Schutt et al. 2017

Deep Sets

What if our data has no time structure?

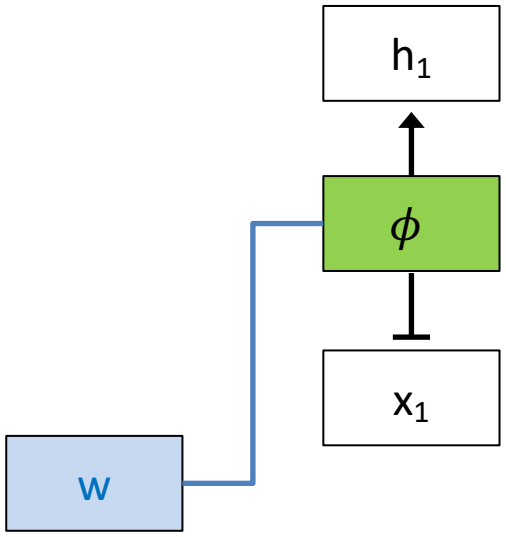
- Data may be variable in length but have no temporal structure → *Data are sets of values*

- *One option*: If we know about the data domain, could try to impose an ordering, then use RNN

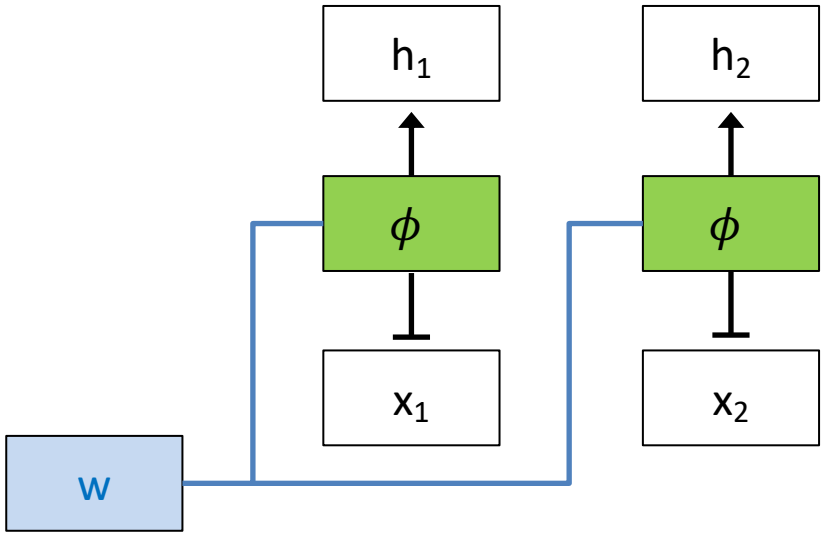
- *Better option*: use system that can operate on variable length sets in permutation invariant way

  – Why permutation invariant → so order doesn't matter

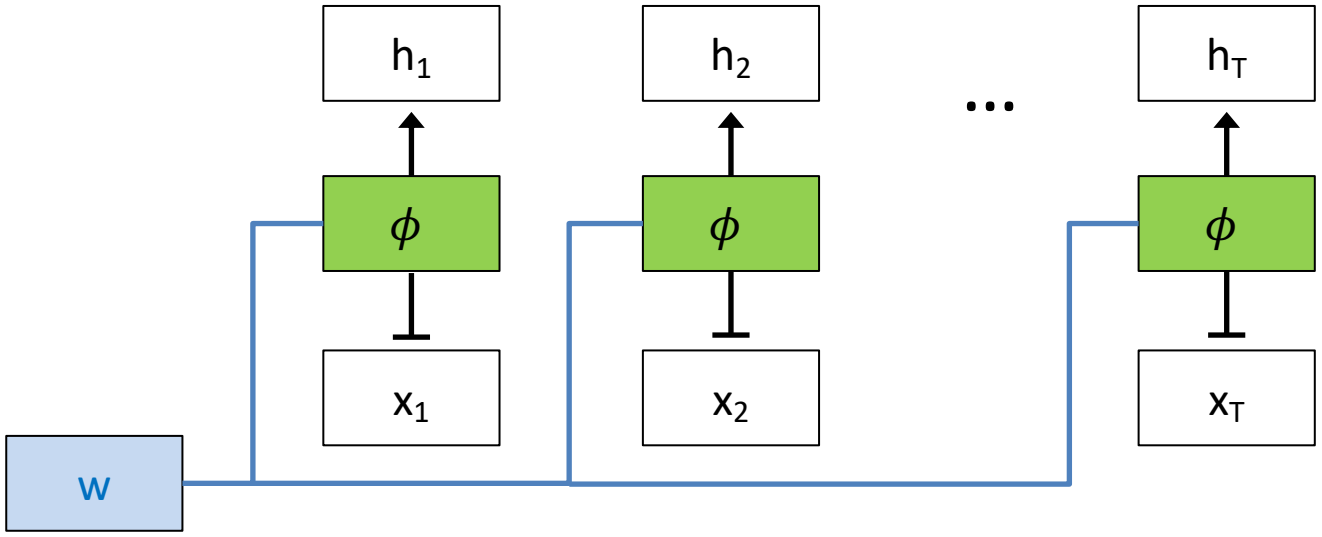
Deep Sets



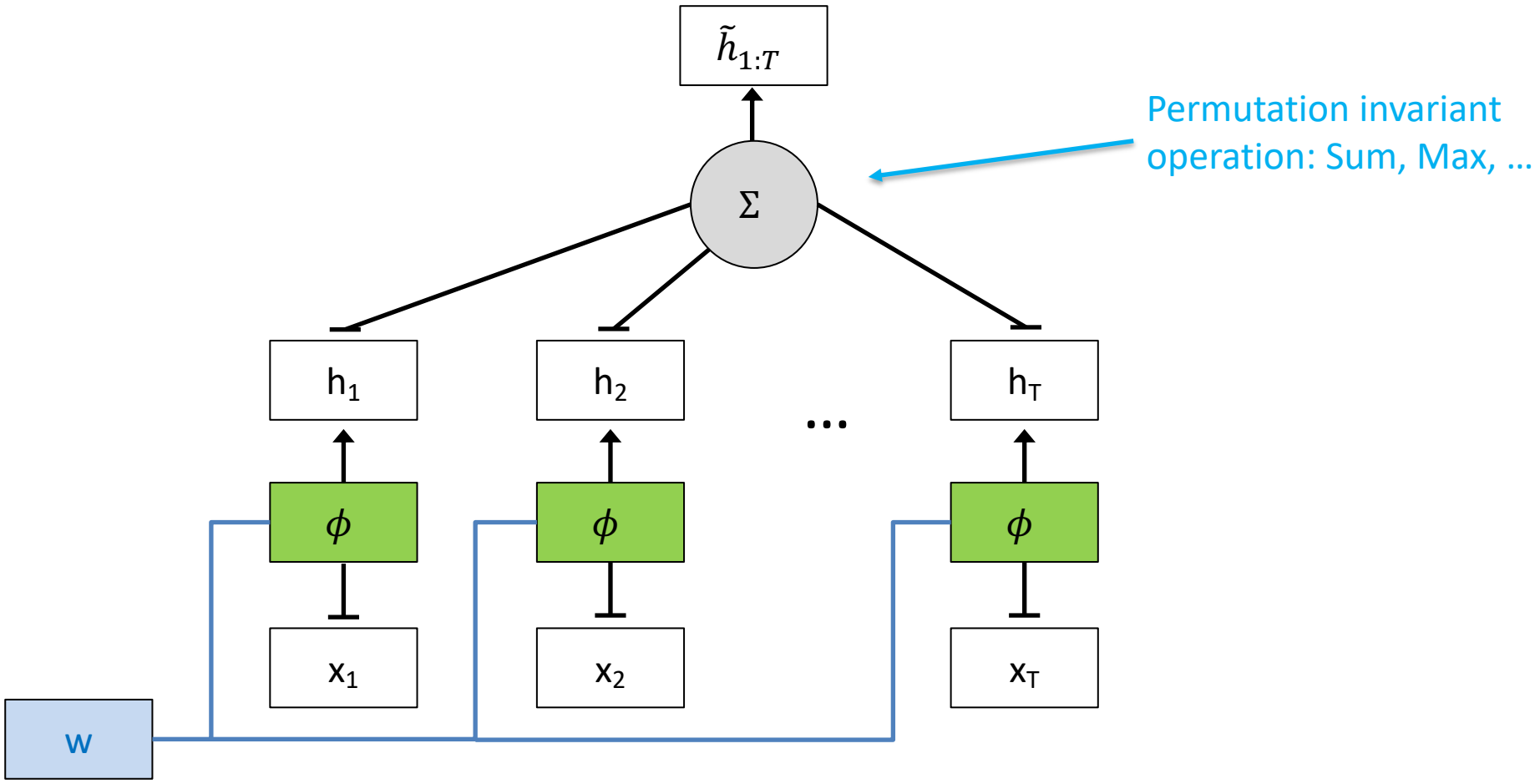
Deep Sets



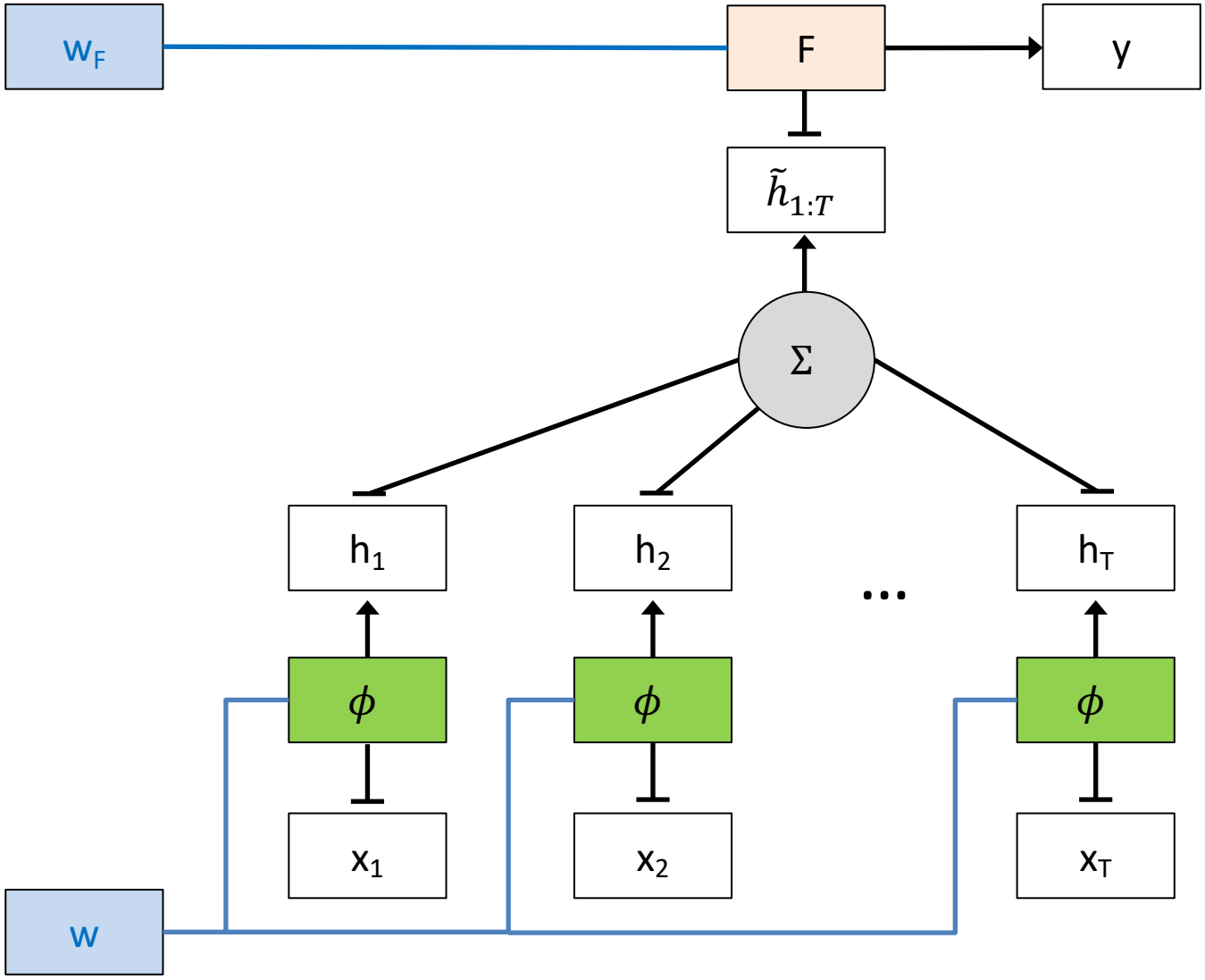
Deep Sets



Deep Sets


$\tilde{h}_{1:T}$
Σ
Permutation invariant operation: Sum, Max, …
h_1
h_2
…
h_T
ϕ
ϕ
ϕ
x_1
x_2
x_T
w

Deep Sets



Examples

Outlier detection

M. Zaheer et. al 2017

Medical Imaging

With more complex architecture

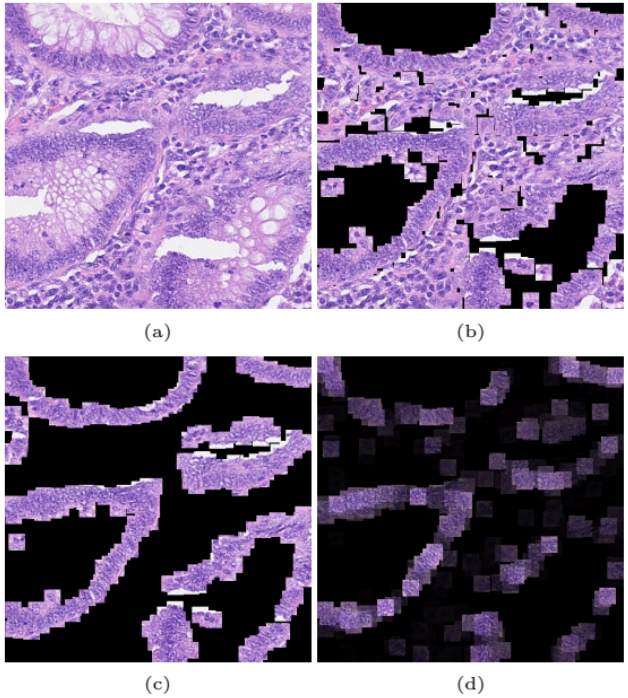


Figure 5. (a) H&E stained histology image. (b) 27×27 patches centered around all marked nuclei. (c) Ground truth: Patches that belong to the class epithelial. (d) Heatmap: Every patch from (b) multiplied by its corresponding attention weight, we rescaled the attention weights using $a'_k = (a_k - \min(\mathbf{a}))/(\max(\mathbf{a}) - \min(\mathbf{a}))$.

M. Ilse et al., 2018