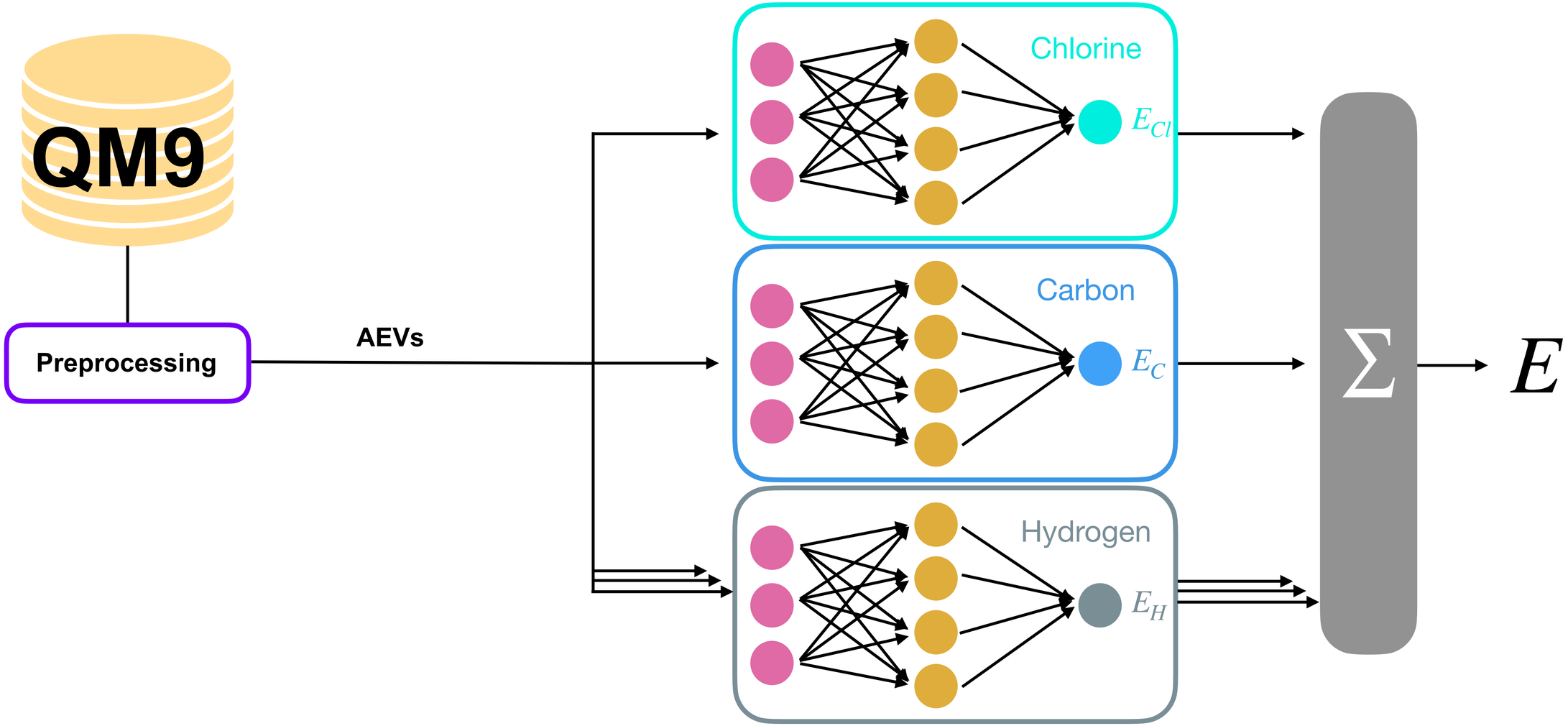
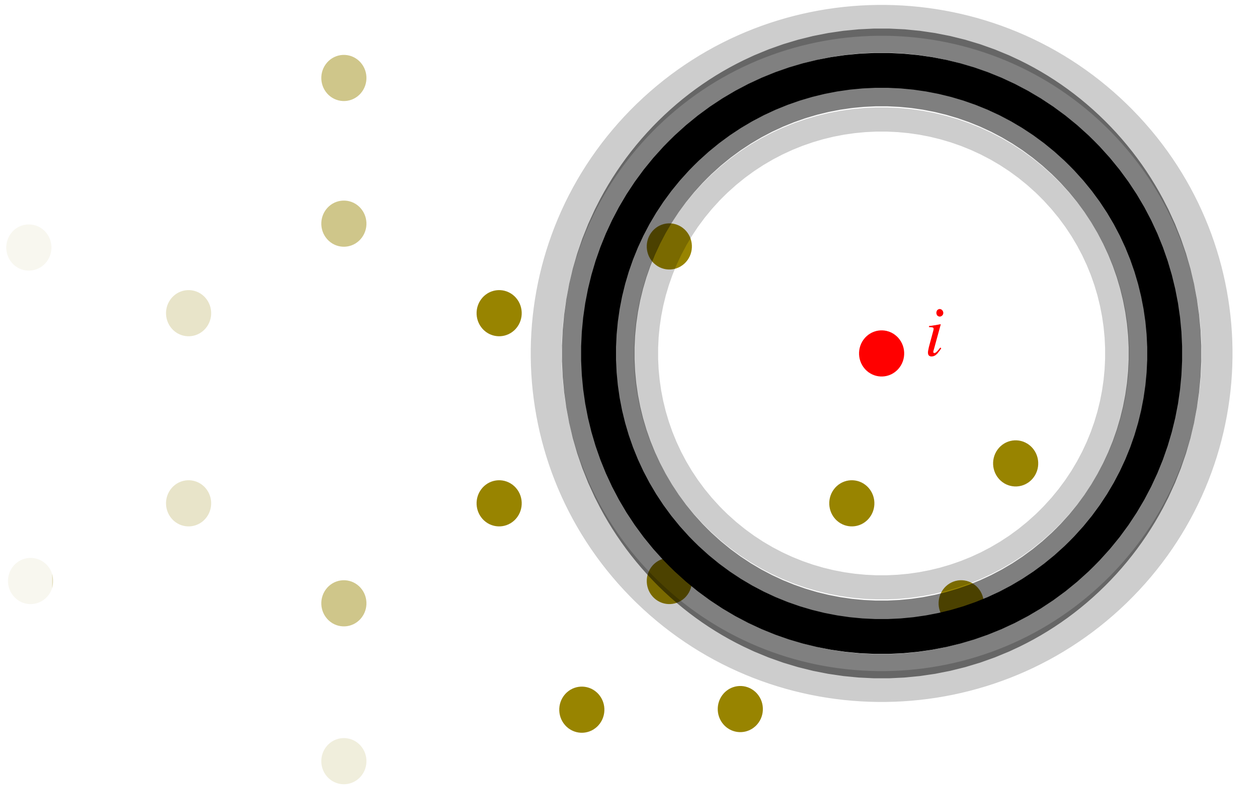
Practical Introduction to Neural Network Potentials Day 4:

# Message-passing neural network potentials

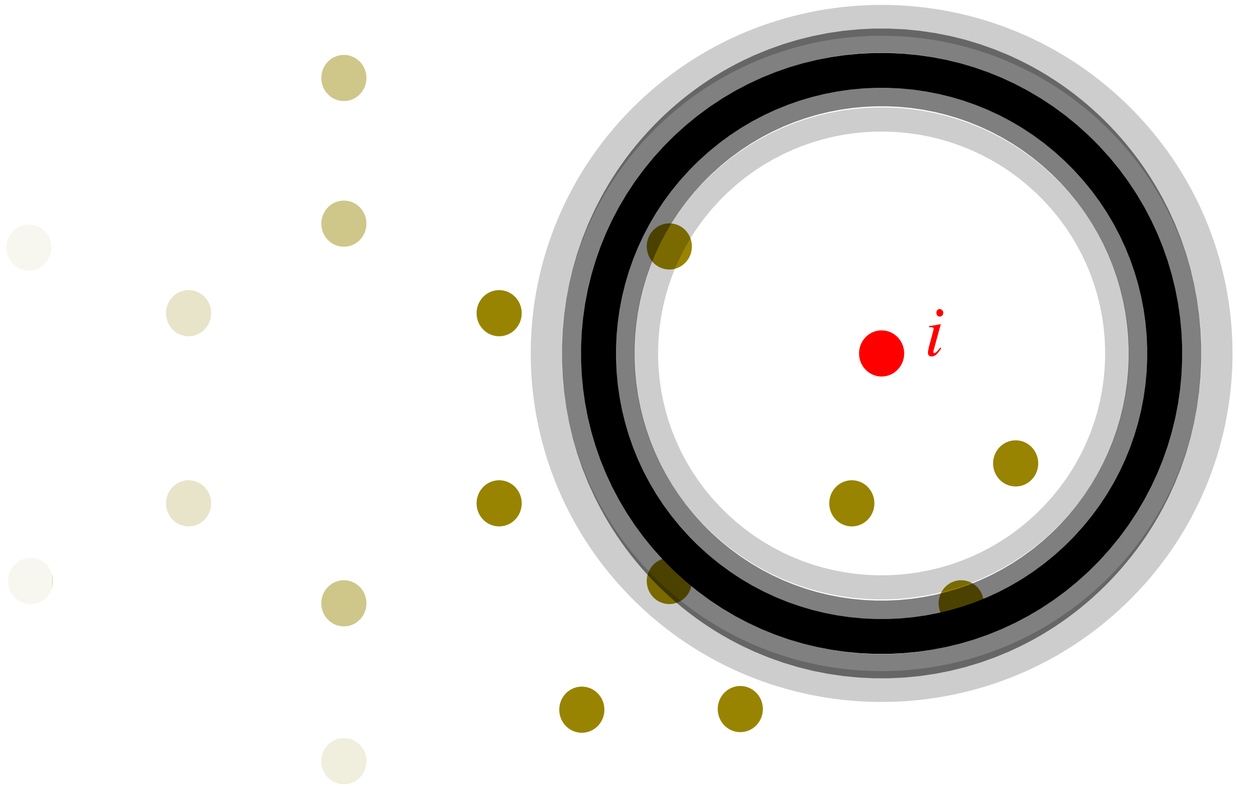
# Review: Behler-Parrinello Neural Networks (BPNNs)






*AEVs : “Symmetry Functions”*

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$




*AEVs : “Symmetry Functions”*

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

**Remember: this is all a funny ploy to have constant length inputs that describe the atomic environment**

Paper 1.

# Atom-centered symmetry functions for constructing high-dimensional neural network potentials

Cite as: J. Chem. Phys. **134**, 074106 (2011); https://doi.org/10.1063/1.3553717
Submitted: 08 December 2010 • Accepted: 21 January 2011 • Published Online: 16 February 2011

**Jörg Behler**

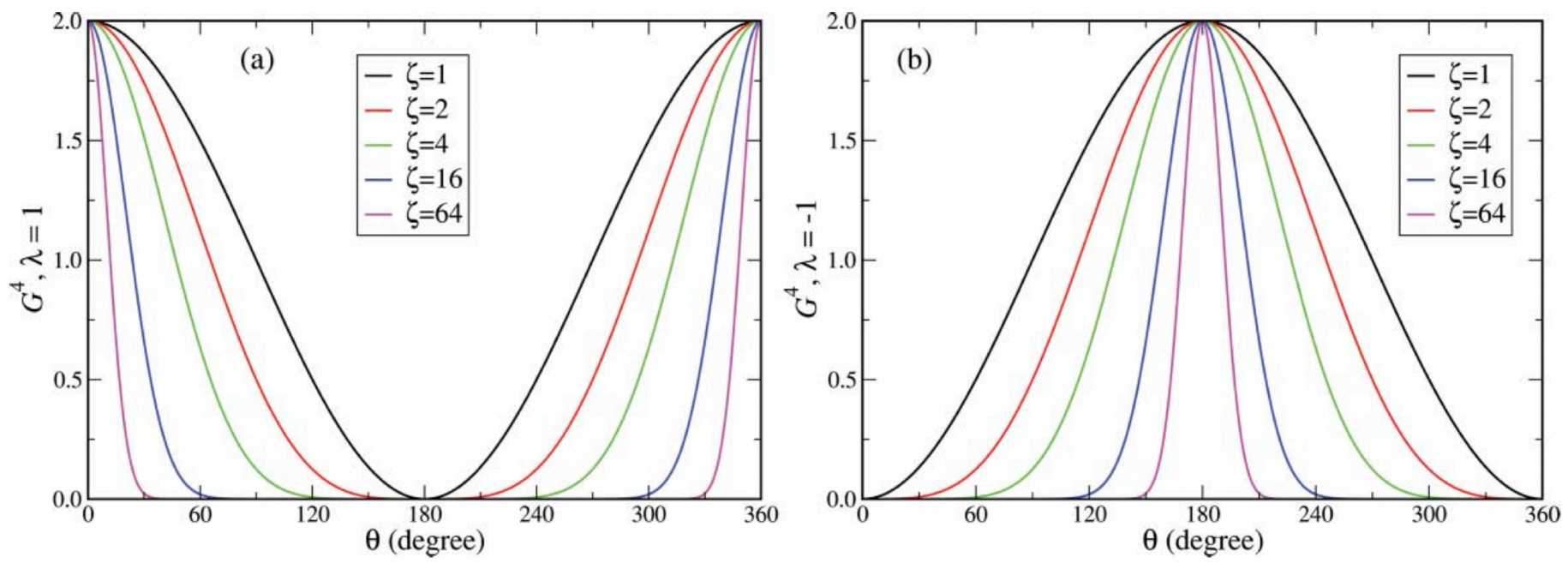
Paper 1.

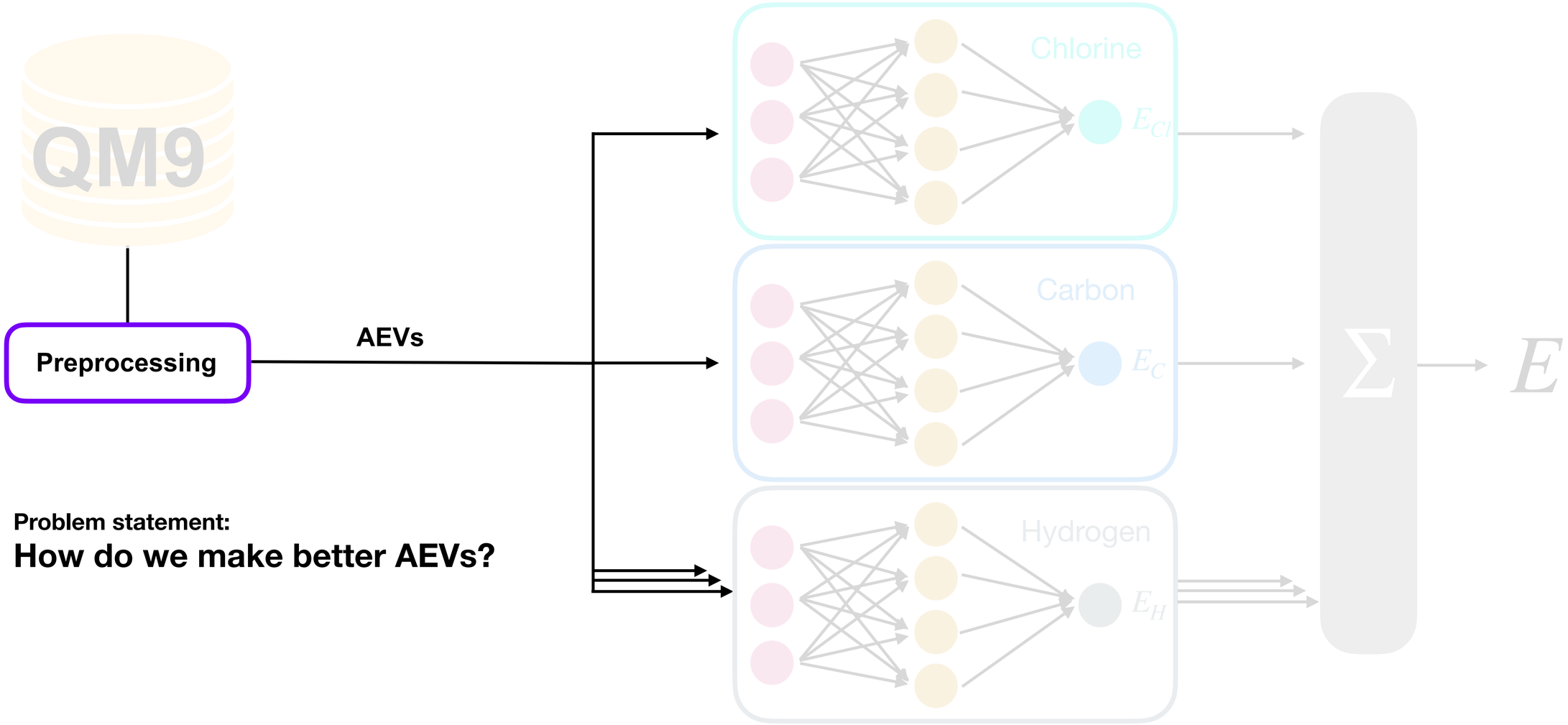
# Atom-centered symmetry functions for constructing high-dimensional neural network potentials

Cite as: J. Chem. Phys. **134**, 074106 (2011); https://doi.org/10.1063/1.3553717
Submitted: 08 December 2010 • Accepted: 21 January 2011 • Published Online: 16 February 2011

**Jörg Behler**

**Introduces *angular* resolution to the AEV as well.**
**Anyone implement these?**






**Problem statement:**

## How do we make better AEVs?

# How do we make better AEVs?

*AEVs : “Local Coulomb Vector”*

$$\overrightarrow{x_i} = \{\frac{Z_j}{r_{ij}} \forall r_{ij} < r_c\}$$

| Pros | Cons |
| --- | --- |
| • Exploits locality<br>• Efficient to compute<br>• Parameter-free | |

# How do we make better AEVs?

*AEVs : “Local Coulomb Vector”*

$$\overrightarrow{x_i} = \{\frac{Z_j}{r_{ij}} \forall r_{ij} < r_c\}$$

| Pros | Cons |
| --- | --- |
| • Exploits locality<br>• Efficient to compute<br>• Parameter-free | • Cannot see beyond a cutoff radius<br>• Non-constant number of neighbors — discontinuous |

# How do we make better AEVs?

*AEVs : “Symmetry Functions”*

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

| Pros | Cons |
| --- | --- |
| • Constant length for all atoms $i$<br>• Smooth & continuous<br>• Exploits locality<br>• Efficient to compute | |

# How do we make better AEVs?

*AEVs : “Symmetry Functions”*

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

| Pros | Cons |
| --- | --- |
| • Constant length for all atoms $i$<br>• Smooth & continuous<br>• Exploits locality<br>• Efficient to compute | • Cannot see beyond a cutoff radius<br>• Some parameters $(\#k, \eta_k, R_k)$ |

# How do we make better AEVs?

*AEVs : "Symmetry Functions"*

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

| Pros | Cons |
| --- | --- |
| • Constant length for all atoms $i$ | • Cannot see beyond a cutoff radius |
| • Smooth & continuous | • Some parameters $(\#k, \eta_k, R_k)$ |
| • Exploits locality | • **Inflexible : How do we know a bunch of Gaussians is the best description of the environment?** |
| • Efficient to compute | |

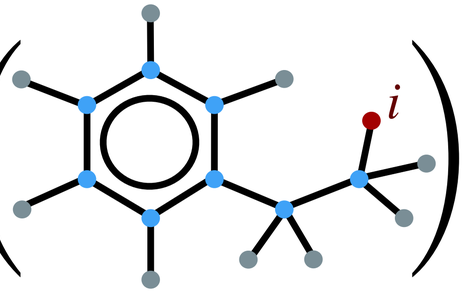
## How do we make better AEVs?

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

**Inflexible**

**Neural networks can express arbitrary functions, can we use those?**

$$\overrightarrow{x_i} = NN\left( \right)$$

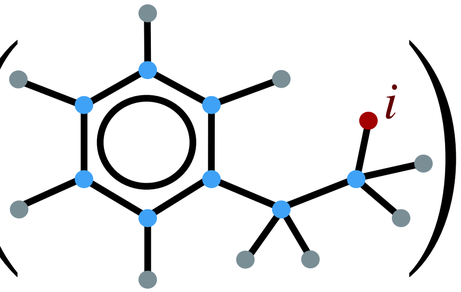


## How do we make better AEVs?

$$\overrightarrow{x_i} = \{ \sum_j e^{-\eta_k (r_{ij} - R_k)^2} \}_k$$

**Inflexible**

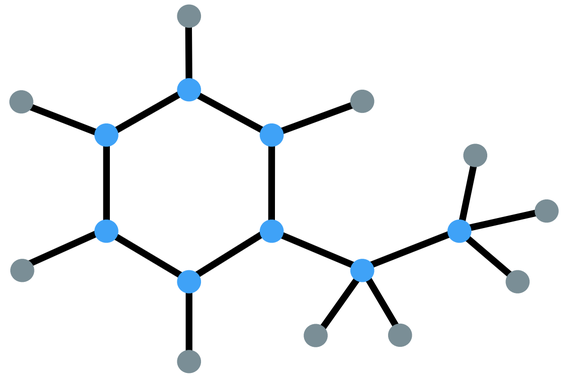
$$\overrightarrow{x_i} = NN\left( \right)$$




***Message-passing neural networks (MPNNs) create AEVs directly from coordinates***

# Molecules as graphs

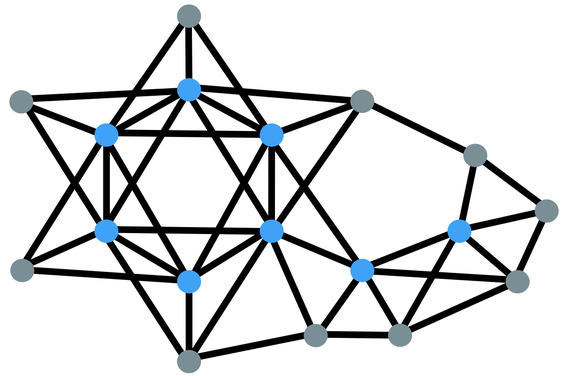
***A molecule can be represented by a graph $\mathscr{G}$ with nodes at nuclei and edges between nearby nuclei:***



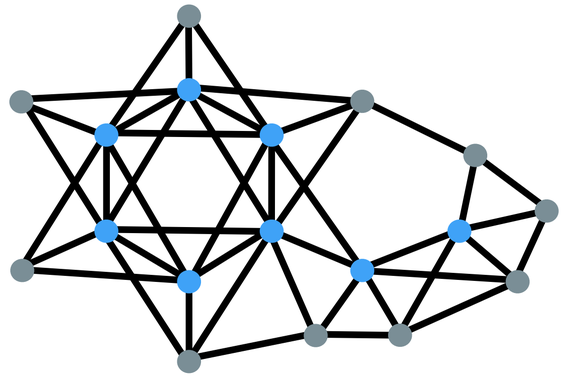
***This “line structure” is a version of a molecular graph, where edges are chemical bonds***

# Molecules as graphs

***Now, forget bonds and use only distances to judge proximity:***

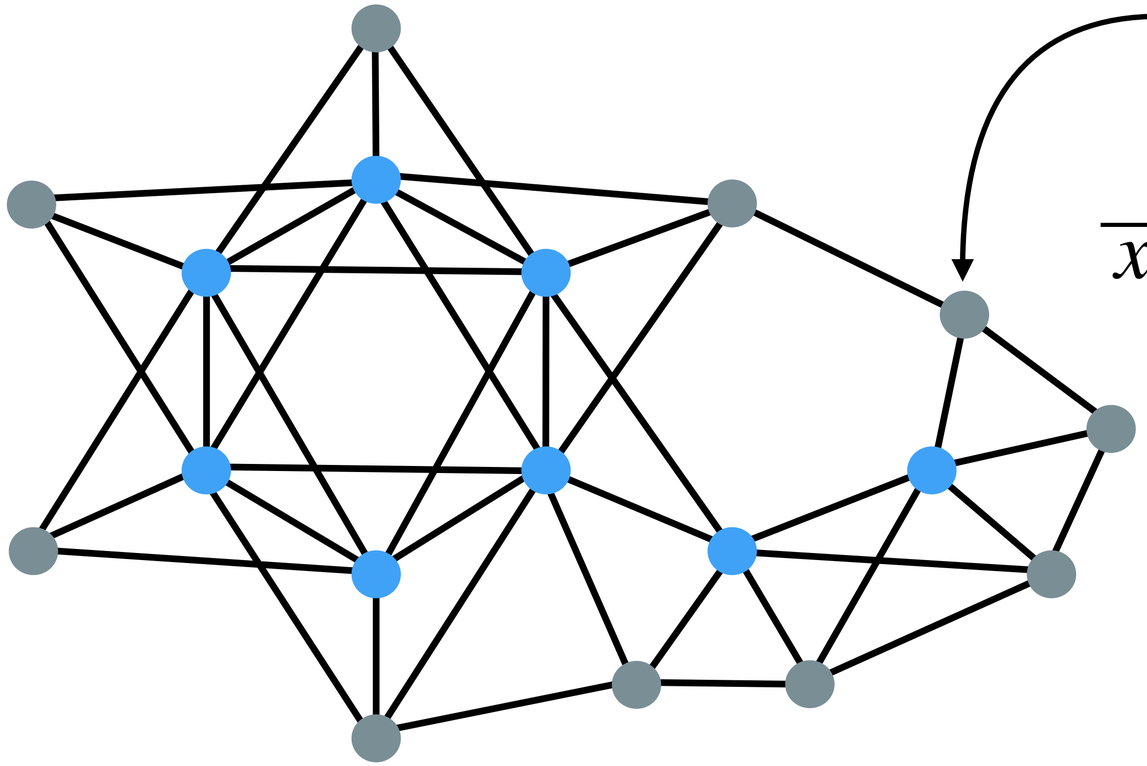


# Molecules as graphs

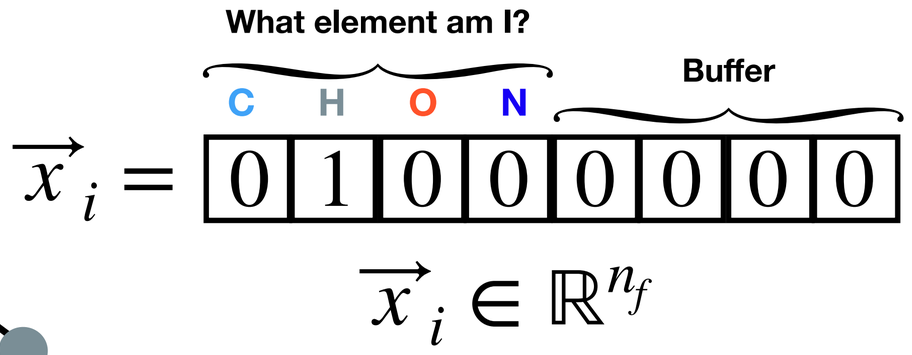


***By performing simple operations on this graph, we can build more informative AEVs.***

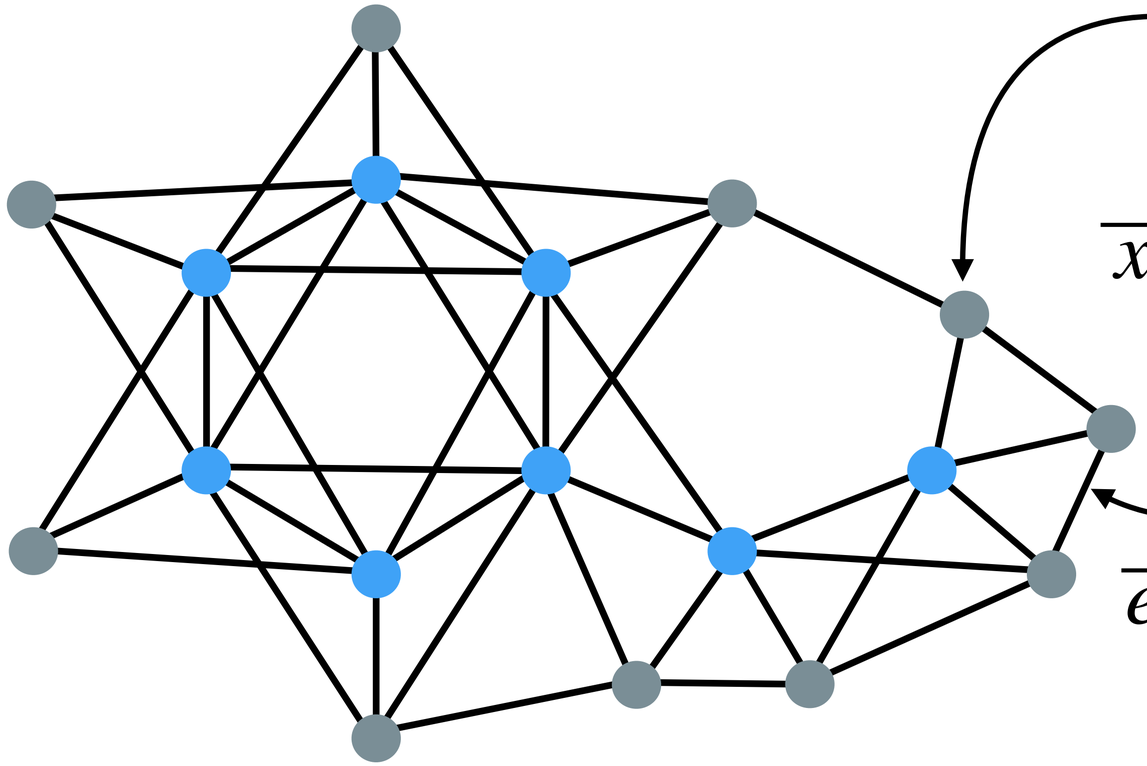
# Message-passing neural networks:
# 1. Initialization



Give each node an initial AEV:



# Message-passing neural networks:
# 1. Initialization



Give each node an initial AEV:

**What element am I?** **Buffer**

C H O N

$$\vec{x}_i = \boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}$$

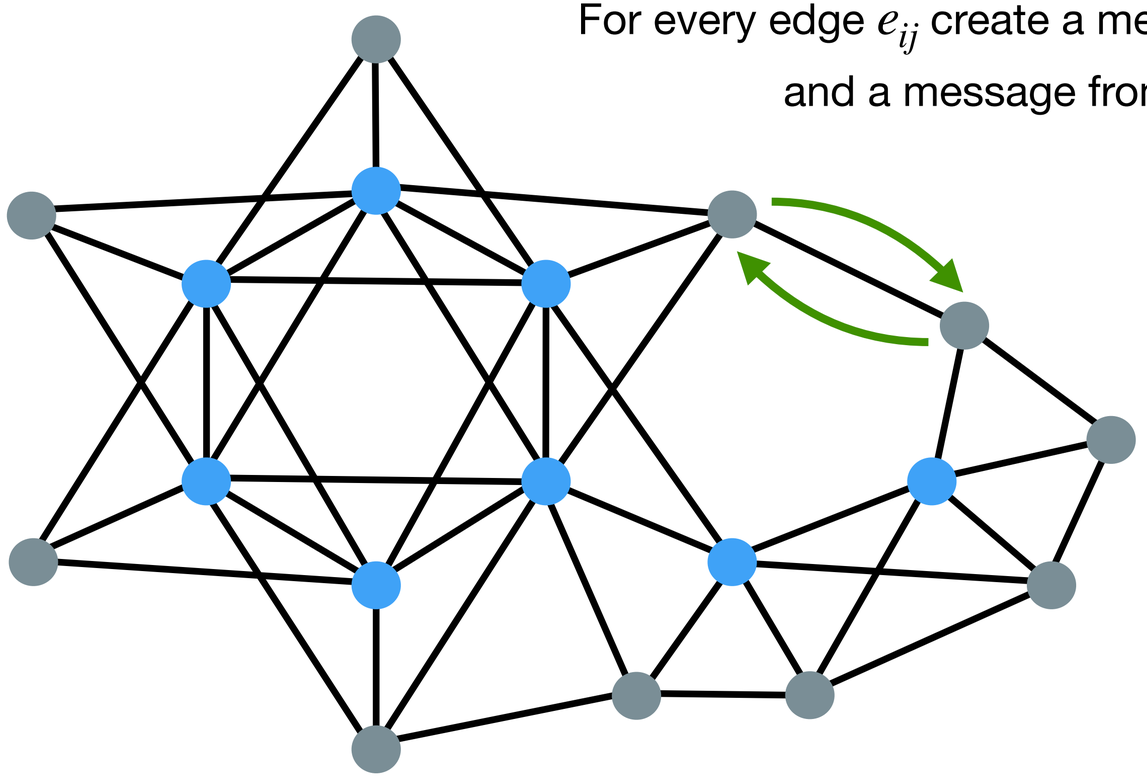
$$\vec{x}_i \in \mathbb{R}^{n_f}$$

Describe each edge by the Euclidean distance:

$$\vec{e}_{ij} = \boxed{r_{ij}} \quad \text{or} \quad \{exp(-\eta_k(r_{ij} - R_k)^2)\}_k \quad \textit{or similar}$$

$$\vec{e}_{ij} \in \mathbb{R}^{n_e}$$

# Message-passing neural networks:
# 2. Message generation



For every edge $e_{ij}$ create a message from $i$ to $j$, $\vec{m}_{ij}$

and a message from $j$ to $i$, $\vec{m}_{ji}$

$$\vec{m}_{ij} = f_m(\vec{x_i}, \vec{x_j}, \vec{e_{ij}})$$

# Message-passing neural networks:
# 2. Message generation

For every edge $e_{ij}$ create a message from $i$ to $j$, $\vec{m}_{ij}$
and a message from $j$ to $i$, $\vec{m}_{ji}$



$$\vec{m}_{ij} = f_m(\vec{x}_i, \vec{x}_j, \vec{e}_{ij})$$

*What do we mean by $f$?*

**Almost anything!**

- **Vector concatenation**
- **Outer product**
- **Matrix multiplication**
- **A neural network ***

# Message-passing neural networks:
# 2. Message generation

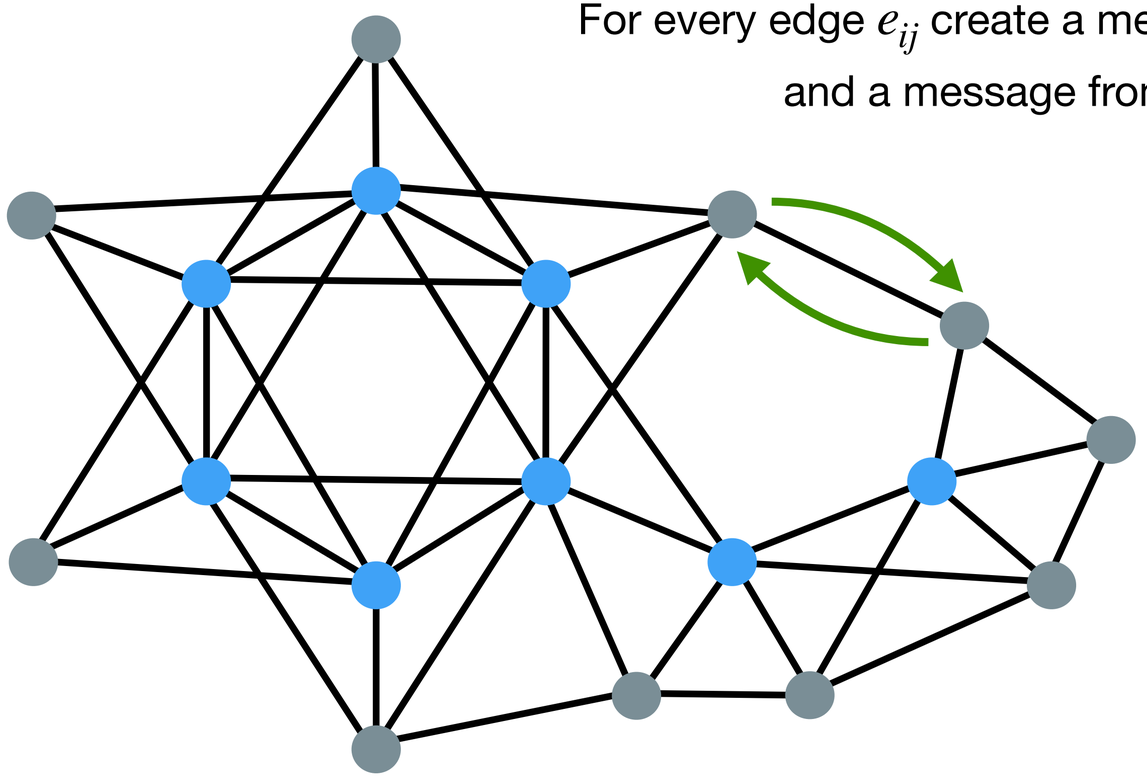
For every edge $e_{ij}$ create a message from $i$ to $j$, $\vec{m}_{ij}$
and a message from $j$ to $i$, $\vec{m}_{ji}$



$$\vec{m}_{ij} = f_m(\vec{x}_i, \vec{x}_j, \vec{e}_{ij})$$

*$f$ can even be parameterized, since we can optimize the parameters with backpropagation*

# Message-passing neural networks:
# 2. Message generation



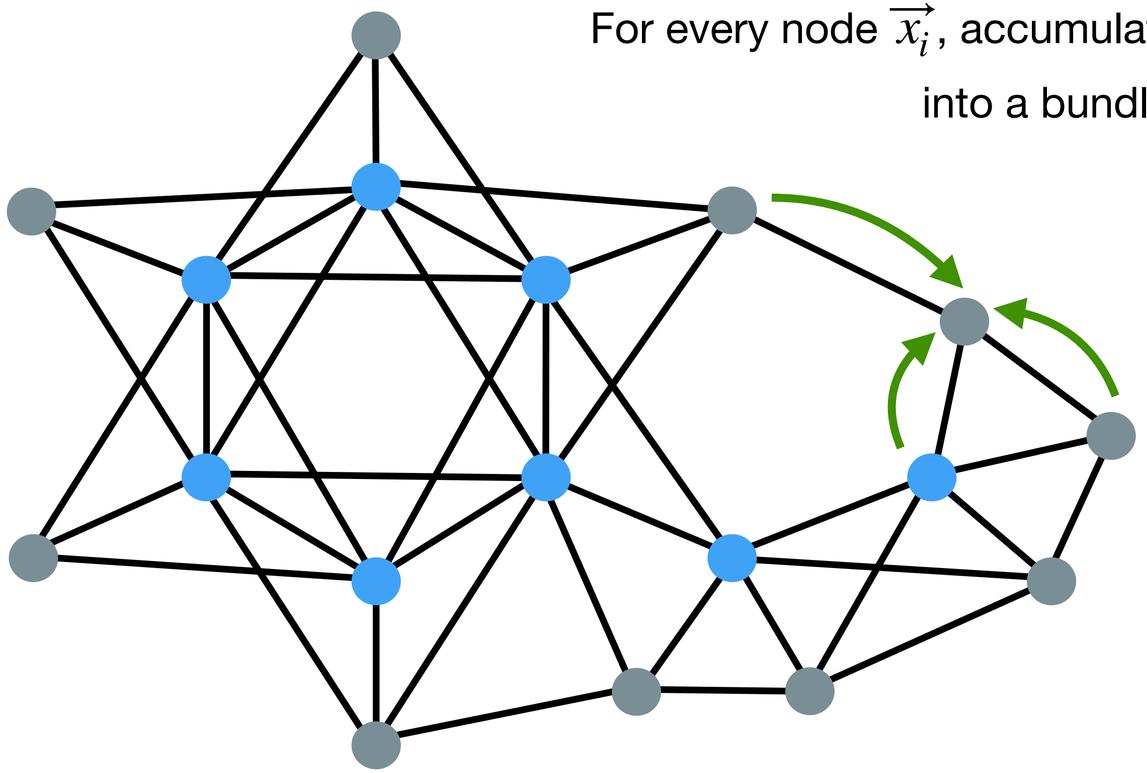
For every edge $e_{ij}$ create a message from $i$ to $j$, $\vec{m}_{ij}$

and a message from $j$ to $i$, $\vec{m}_{ji}$

$$\vec{m}_{ij} = f_m(\vec{x_i}, \vec{x_j}, \vec{e_{ij}})$$

$$f_m : \mathbb{R}^{2n_f+n_e} \rightarrow \mathbb{R}^{n_m}$$

# Message-passing neural networks:
# 3. Message accumulation



For every node $\vec{x}_i$, accumulate all messages $\{\vec{m}_{ji}\}$ into a bundle $\vec{M}_i$

$$\vec{M}_i = \sum_j \vec{m}_{ji}$$

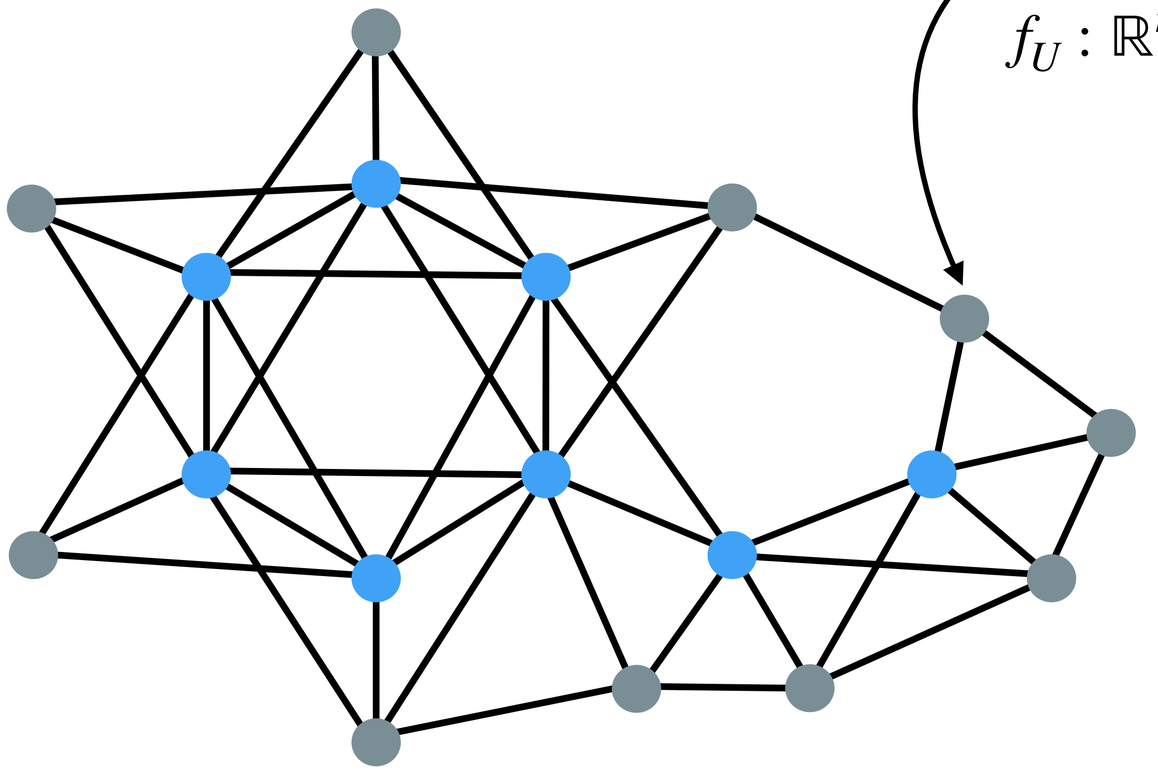
$$\vec{M}_i \in \mathbb{R}^{n_m}$$

# Message-passing neural networks:
# 4. Node update

Update node states $\vec{x_i}$ based on message bundle $\vec{M}_i$

$$\vec{x_i} = f_U(\vec{x_i}, \vec{M}_i)$$

$$f_U : \mathbb{R}^{n_f + n_m} \rightarrow \mathbb{R}^{n_f}$$

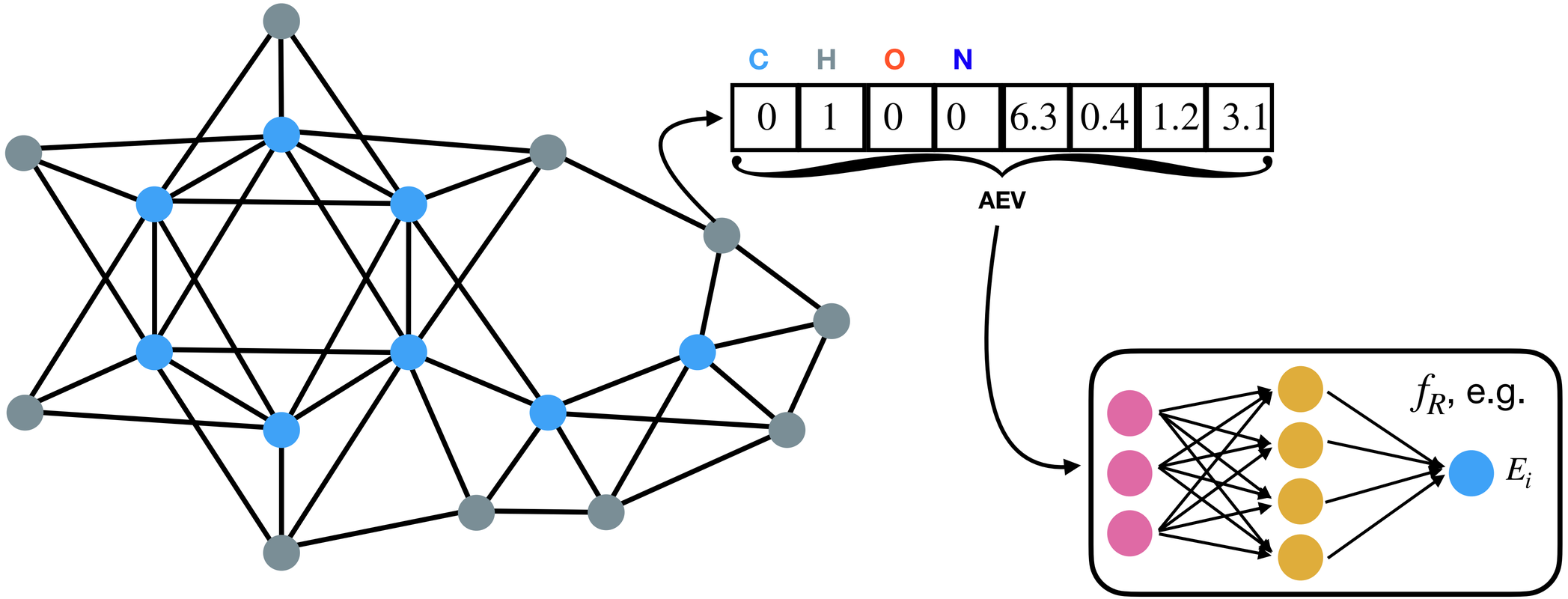


# Message-passing neural networks:
# 5. Readout (get atomic energies)

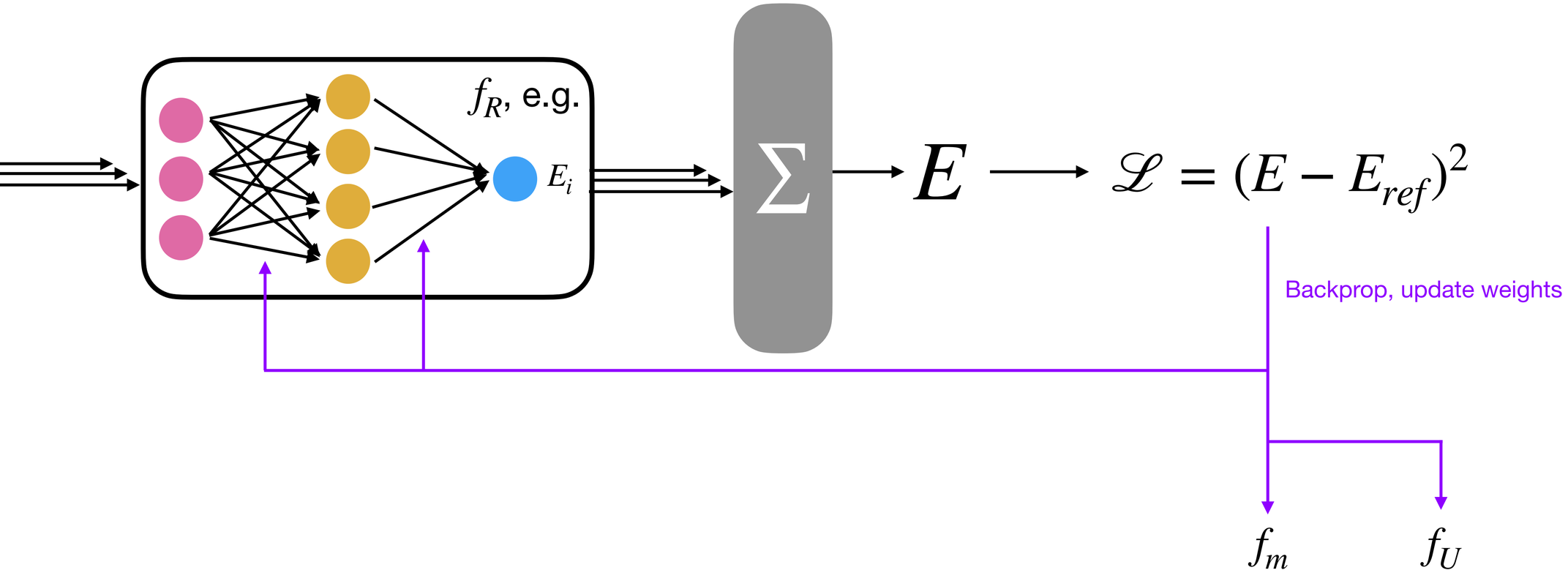
$$E_i = f_R(\overrightarrow{x_i})$$

$$f_R : \mathbb{R}^{n_f} \rightarrow \mathbb{R}^1$$

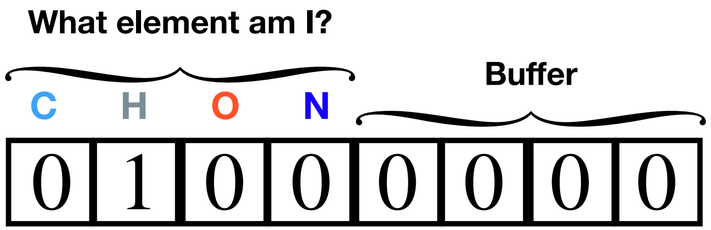
Pass each final AEV into an atomic function called a *"readout"*



# Message-passing neural networks:
# 6. Get loss, update weights



# MPNN Blueprint




1. Initialize features $\vec{x}_i^{t=0}$ and edges $\vec{e}_{ij}$ — $\boxed{r_{ij}}$ or $\{exp(-\eta_k(r_{ij}-R_k)^2)\}_k$ or similar

2. Create a "message" $\vec{m}_{ji}^t$ from each atom $j$ to $i$ — $\vec{m}_{ji}^t = f_m(\vec{x}_j^{t-1}, \vec{x}_i^{t-1}, \vec{e}_{ij})$

3. Accumulate all messages received at each atom $i$ — $\vec{M}_i^t = \sum_j \vec{m}_{ji}^t$

4. Update atomic features $\vec{x}_i^{t+=1}$ based on messages — $\vec{x}_i^t += f_U(\vec{M}_i^t, \vec{x}_i^{t-1})$

Repeat $T$ times ($T \approx 2-5$)

5. Readout atomic predictions — $y_i = f_R(\vec{x}^T)$

6. Compute loss, optimize the three functions that define the MPNN: $f_m$, $f_U$, and $f_R$

## Molecular mechanics

**Inflexible AEV**

$$\overrightarrow{x_i} = \{r_{ij}\}, \{\theta_{ijk}\}_{bonded}, \{\phi_{ijkl}\}_{bonded}$$

**Inflexible energy function**

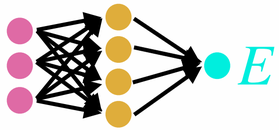
$$E = \sum^{bonded} E_{ij}^{bond} + E_{ijk}^{angle} + E_{ijkl}^{torsion} + \sum^{non-bonded} E_{ij}^{elst} + E_{ij}^{vdW}$$

## Behler-Parrinello neural network (BPNN)

**Inflexible AEV**

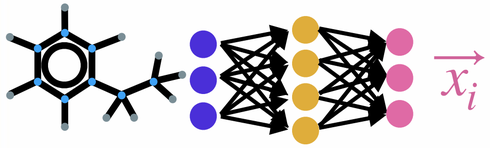
$$\overrightarrow{x_i} = \{\sum_j e^{-\eta_k(r_{ij}-R_k)^2}\}_k$$

**Flexible energy function**




## Message-passing neural network (MPNN)

**Flexible AEV**




**Flexible energy function**