An ANFIS framework for PyTorch

James F. Power
Maynooth University
Co. Kildare, Ireland

Michael Ryan,
O.C.E. Technology Ltd.,
Dublin, Ireland.

John Yan,
Orbita Aerospace Co.,
Zhuhai, China.

ANFIS

The **Adaptive Network-Based Fuzzy Inference System** (ANFIS) architecture is a long-established and popular approach to implementing fuzzy systems [1].

Features:

- **Layered architecture** to model a fuzzy system with layers for membership functions and rules.
- **Learnable parameters** control the shape of the membership functions (e.g. Gaussian) and the coefficients of the TSK-style rule consequents.
- **Hybrid learning** combines LSE to fit the rule coefficients with gradient descent to learn the membership functions.

Principal Contributions

By implementing ANFIS in PyTorch we allow for fuzzy systems to be specified, trained and deployed in a deep learning environment.

Benefits:

- Many features now common in deep learning, such as experimenting with mini-batching, learning-rate optimisation algorithms and different loss functions, are now available to ANFIS systems.
- It is now easier to design fuzzy-neural systems that combine elements of fuzzy and deep learning, or systems that involve multiple layers of fuzzy reasoning.

Connectivity

- **Import** data from NumPy and from SciKit Learn, e.g. sklearn.datasets. Also, add to sklearn pipelines (via skorch)
- **Export** fuzzy systems via Py4JFML JFML implements the *IEEE Standard for Fuzzy Markup Language* and has import/export for other file formats.

... and clustering too

Fuzzy C-Means clustering using gradient descent [2] also fits this framework.

Process: Pick a set of c cluster centres, then

- The *forward* pass assigns points to clusters, with membership based on distance to cluster centre.
- The *backward* pass re-calculates centroids, then moves cluster centres towards these.

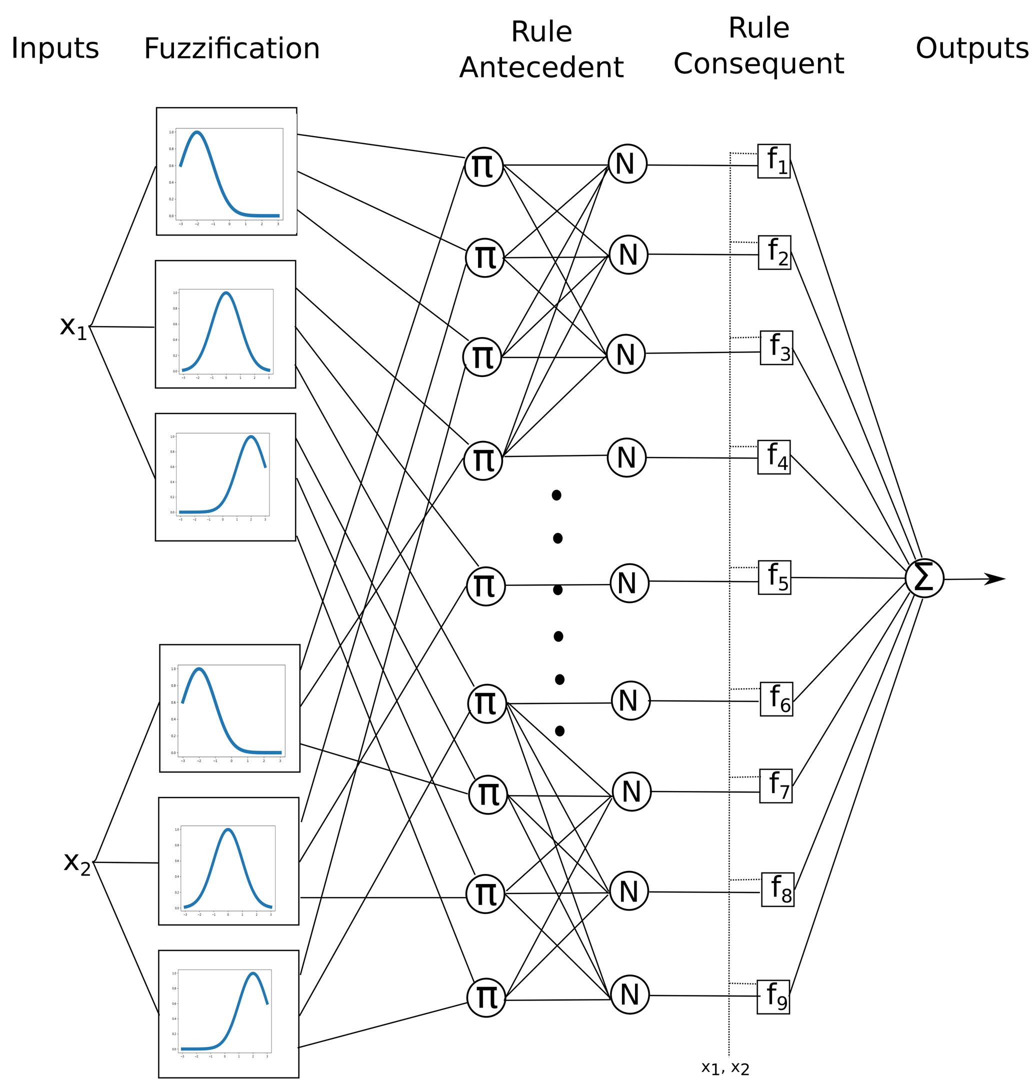
Advantage: can split large data sets into mini-batches.

Related Work: Fuzzy/Python

Elsewhere on `github.com`:

- `scikit-fuzzy/scikit-fuzzy`: Fuzzy logic toolbox for SciPy (no ANFIS)
- `twmeggs/anfis`: bare-bones implementation of ANFIS (manual derivatives) via NumPy
- `cmencar/py4jfml`: A Python wrapper for the Java APIs in JFML

The ANFIS structure



ANFIS in PyTorch

Each ANFIS layer (and each membership function) is implemented as a PyTorch Module or function.

- **Specify** the fuzzy system by describing the input and output variables and (optionally) membership functions.
- **Train** the system by providing test data; the ANFIS system learns the rule consequents, and adjusts membership functions.
- **Deploy** the system as a PyTorch module, Python class, or export to other formats.

ANFIS Implementation

```
# Container for the 5 layers of the ANFIS net.
class AnfisNet(torch.nn.Module):
  def __init__(self, description, invardefs,
                outvarnames, hybrid=True):
    ... Code omitted ...
    if hybrid:  # Uses LSE for coefficients
      cl = ConsLayer(num_in, num_rules, num_out)
    else:  # Uses gradient descent for coeffs
      cl = GDConsLayer(num_in, num_rules, num_out)
    self.layer = torch.nn.ModuleDict(OrderedDict([
      ('fuzzify', FuzzifyLayer(mfdefs, varnames)),
      ('rules', AntecedentLayer(mfdefs)),
      # normalisation layer is just a function.
      ('consequent', cl),
      ('sum', WeightedSumLayer()),
      ]))

def forward(self, x):
  ''' Forward pass: predict y values from x. '''
  fuzzified = self.layer['fuzzify'](x)
  raw_weights = self.layer['rules'](fuzzified)
  weights = F.normalize(raw_weights, p=1, dim=1)
  rule_tsk = self.layer['consequent'](x)
  y_pred = self.layer['sum'](weights, rule_tsk)
  return y_pred
```

- Since PyTorch tensors are used throughout, the back-propagation pass is calculated automatically.
- The training code is standard PyTorch boilerplate:

```
for e in range(num_epochs):
  # Process each mini-batch in turn:
  for x, y_actual in data:
    y_pred = model(x)   # Forward pass thru ANFIS
    loss = compare(y_pred, y_actual)  # Calc loss
    optimizer.zero_grad()    # Zero the gradients
    loss.backward()   # Backprop grads thru ANFIS
    optimizer.step()   # Update ANFIS parameters
```

PyTorch

PyTorch is an open-source deep learning platform for Python, featuring:

- Tensor computing (like NumPy arrays)
- Automatic differentiation on recorded tensor operations
- Dynamically-configurable graphs to back-propagate gradients
- Libraries for neural nets, optimisers, loss functions

`https://pytorch.org`

Example: the inverted pendulum

Learn a fuzzy controller for the classic inverted pendulum example with back-propagation through time [3]

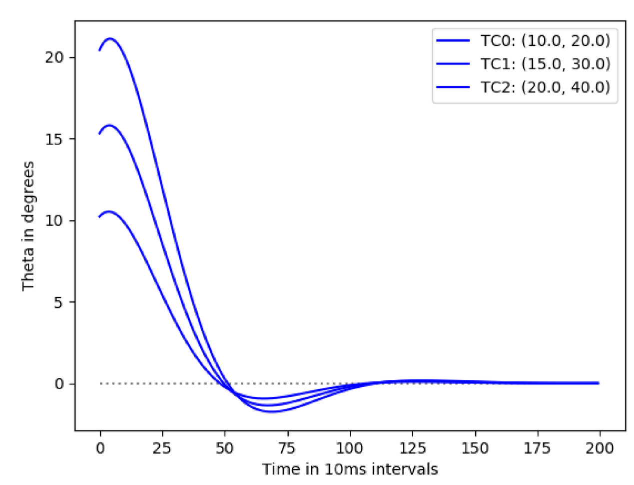
- Specify the ANFIS controller with two inputs (θ and $d\theta$) and two Bell membership functions per input:

```
theta = ('theta', OrderedDict([
  ('small', BellMembFunc(20, 2, -20)),
  ('large', BellMembFunc(20, 2, -20)),
  ]))
dtheta = ('dtheta', OrderedDict([
  ('small', BellMembFunc(20, 2, -50)),
  ('large', BellMembFunc(20, 2, -50)),
  ]))
anfis = AnfisNet('Pendulum Controller',
        invardefs=[theta, dtheta],
        outvarnames=['force'], hybrid=False)
```

- Define a PyTorch module that dynamically builds 100 ANFIS controllers in sequence, with shared parameters:

```
def forward(self, x):
  self.pendulum.state = x
  for i in range(100):
    # Forward pass thru ANFIS controller:
    force = anfis(self.pendulum.state)
    # Physical model: map force to new state:
    self.pendulum.take_step(force)
```

- Train the system (two inputs, 15 epochs) with a loss function that minimises θ and force.
- Test: θ reduces to zero after 100 time steps:



References

[1] J. R. Jang. ANFIS: adaptive-network-based fuzzy inference system. *IEEE Trans. Systems, Man, and Cybernetics*, 23(3):665–685, 1993.

[2] Y. Wang, L. Chen, and J. Mei. Stochastic gradient descent based fuzzy clustering for large data. In *IEEE International Conference on Fuzzy Systems, FUZZ-IEEE*, pages 2511–2518, July 6-11 2014.

[3] J. R. Jang. Self-learning fuzzy controllers based on temporal backpropagation. *IEEE Trans. Neural Networks*, 3(5):714–723, 1992.

Download

Our framework for ANFIS in PyTorch with these and other examples is open-source and available from
`https://github.com/jfpower/anfis-pytorch`