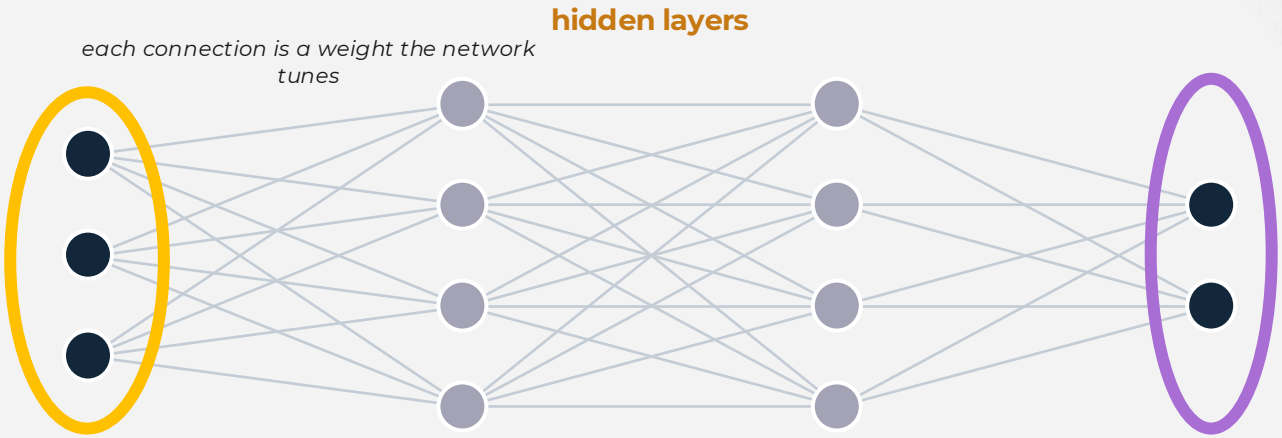
Toy Example: Model Comparison

Konstantina Sokratous

For parameter estimation

Inputs:

Simulated data



input
(data x)

output
(e.g. θ)

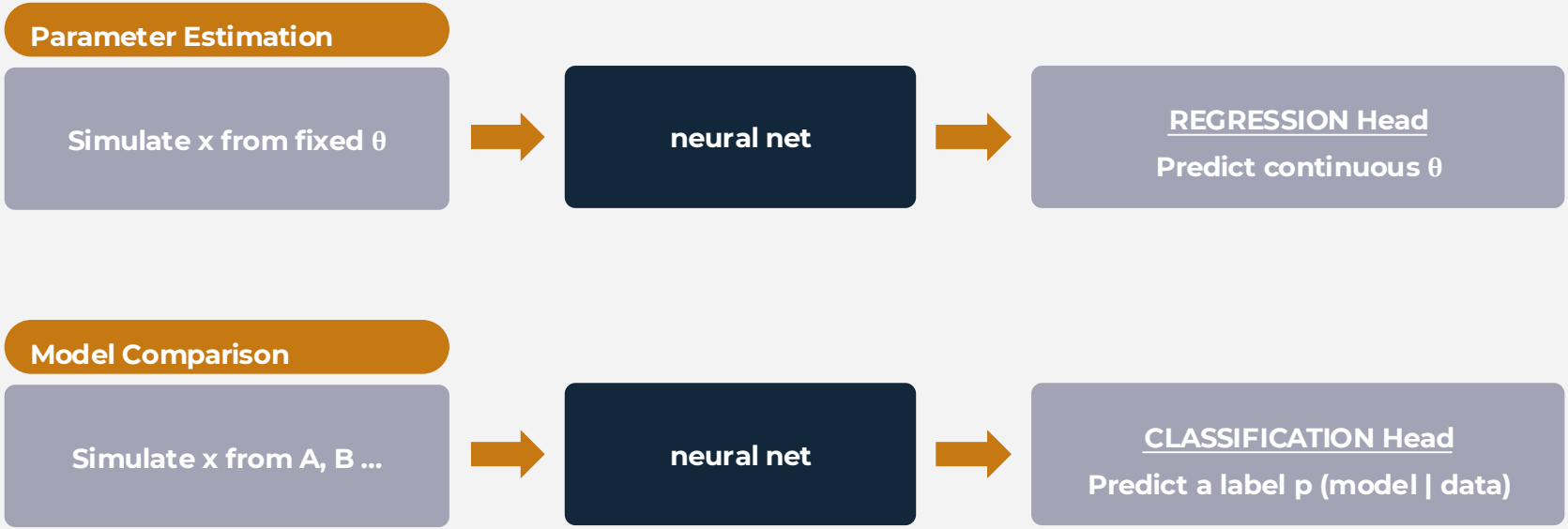
Outputs:
best-fit model parameters (what generated the data?)

What about multiple models?

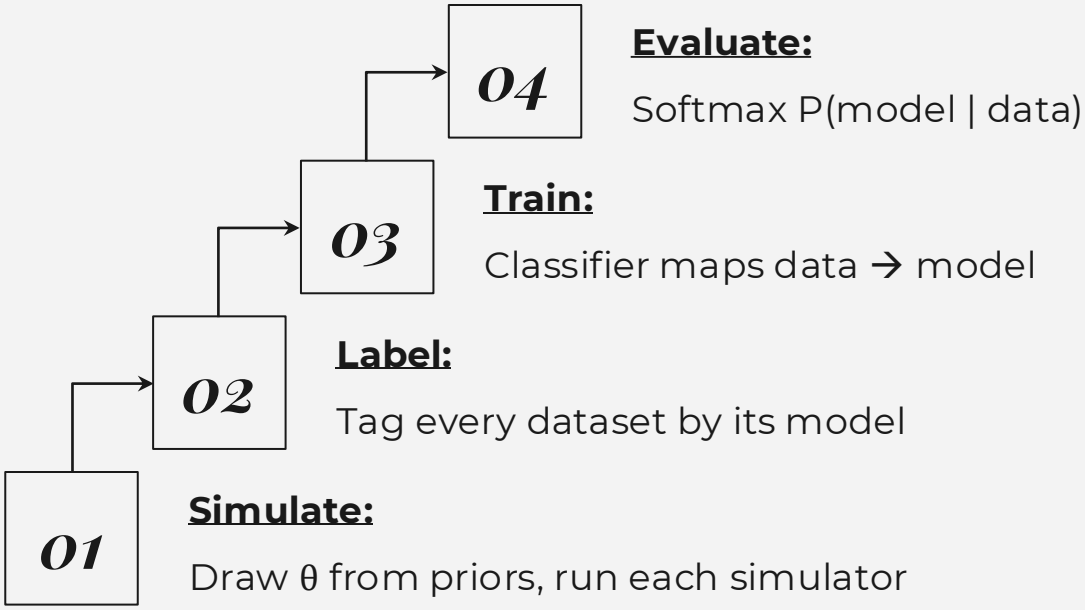
01

P(model | data)

Model Comparison


Parameter Estimation
Simulate x from fixed θ
neural net
REGRESSION Head
Predict continuous θ
Model Comparison
Simulate x from A, B ...
neural net
CLASSIFICATION Head
Predict a label p (model | data)

Main Steps



Evaluate:

Softmax P(model | data)

Train:

Classifier maps data → model

Label:

Tag every dataset by its model

Simulate:

Draw θ from priors, run each simulator

02

Example Application

Intertemporal Choice

The Task – Kirby's Monetary Choice Questionnaire

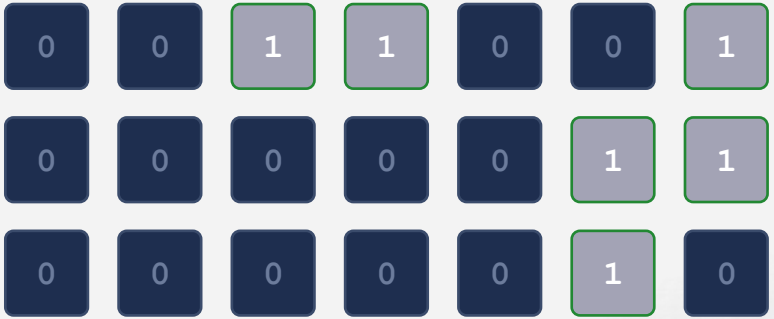
- 27 fixed questions. Each pits a **smaller-sooner** reward against a **larger-later** one

Would you rather have $55 dollars today or $75 dollars in 61 days?

- 1 participant → 27 binary choices
- The choice data is our input, they go straight into the network

0 = chose sooner 1 = chose later

`shape: (n_participants, 27)`



Direct Difference

$$d = w\,(LL^{u} - SS^{u}) - (1 - w)(t_{LL}^{v} - t_{SS}^{v})$$

parameters: u, v, w

Hyperbolic

$$V = \frac{A}{1 + kt}$$

parameters: k, m

Hyperboloid

$$V = \frac{A}{1 + kt^{s}}$$

parameters: k, s, m

STEP 1 OF 4 · THE CODE

Each model is a simulator (parameters in, choices out)

- params holds one parameter draw (one simulated subject) vectorized over thousands at once
- Compute the value difference d between the two options for all 27 items
- A logistic choice rule turns d into a probability

```
def r_hyperbolic(params, stim):
    """Hyperbolic discounting + logistic choice. params: k, m."""
    k, m = params[:, [0]], params[:, [1]]
    SS, LL, tSS, tLL = unpack_stim(stim)

    d = LL / (1 + k*tLL) - SS / (1 + k*tSS)      # value difference
    p = sigmoid(d / m)                           # choice probability
    return (rng.random(p.shape) < p).astype(np.float32)
```

STEP 1 OF 4 · THE CODE

Draw the priors

- For each model we draw its parameters from priors, not fixed values
- Data from the hyperbolic model means data from the whole family of plausible hyperbolic subjects
- The classifier learns to recognize a model despite parameter variation

```
def draw_priors(n):
    # Direct difference:  u, v, w
    w = sigmoid(rng.normal(0, 1.5, n))
    u, v = rng.normal(0.7, 0.3, n), rng.normal(0.7, 0.3, n)

    # Hyperbolic:  k, m       Hyperboloid:  k, s, m
    k_c = np.exp(rng.normal(-5, 3, n))
    m_c = np.exp(rng.normal(1,  5, n))
    ...
    return dd, hyp, hbd      # one parameter draw per simulated subject
```

STEP 2 OF 4 · THE CODE

Stack simulations and label by its generating model

- Simulate n subjects from each model and concatenate: X has shape (3n, 27)
- The label y is just 0, 1, or 2 (index in python) which simulator produced each row
- That's our entire training set

```python
def simulate_dataset(n_per_model):
    dd, hyp, hbd = draw_priors(n_per_model)

    X = np.concatenate([                # 27 choices per subject
        r_direct_difference(dd,  MCQ_STIM),
        r_hyperbolic(       hyp, MCQ_STIM),
        r_hyperboloid(      hbd, MCQ_STIM),
    ])
    y = np.repeat([0, 1, 2], n_per_model)   # the generating-model label
    return X, y                             # X:(3n, 27)   y:(3n,)
```

STEP 3 OF 4 · THE CODE

Build NN and train

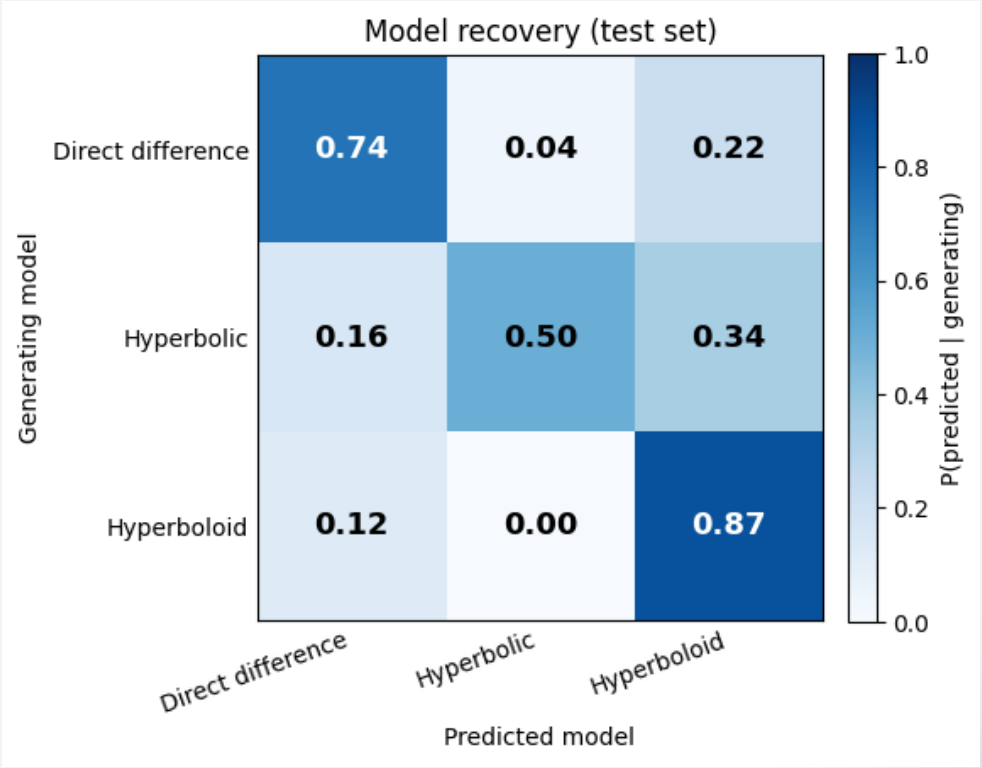
- 3 hidden layers (100 → 50 → 25) with tanh, nothing exotic, final layer outputs raw logits for the 3 classes
- CrossEntropyLoss applies log-softmax internally
- Cross-entropy is minimised by the true conditional probabilities, so at the optimum the softmax outputs approximate P(model | data)

```
class ModelComparisonNet(nn.Module):
    def __init__(self, n_inputs, n_outputs=3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_inputs, 100), nn.Tanh(),
            nn.Linear(100, 50),       nn.Tanh(),
            nn.Linear(50,  25),       nn.Tanh(),
            nn.Linear(25,  n_outputs),   # -> logits, no softmax here
        )

    def posterior(self, x):              # only at prediction time
        return torch.softmax(self.net(x), dim=-1)
```

```
loss_fn = nn.CrossEntropyLoss()      # a proper scoring rule
opt = torch.optim.Adam(model.parameters(), lr=1e-3)

for epoch in range(epochs):
    for idx in batches(n, size=1000):
        opt.zero_grad()
        loss = loss_fn(model(X[idx]), y[idx])   # logits vs. labels
        loss.backward()
        opt.step()
# minimizing cross-entropy  ->  outputs approximate P(model | data)
```

Evaluate

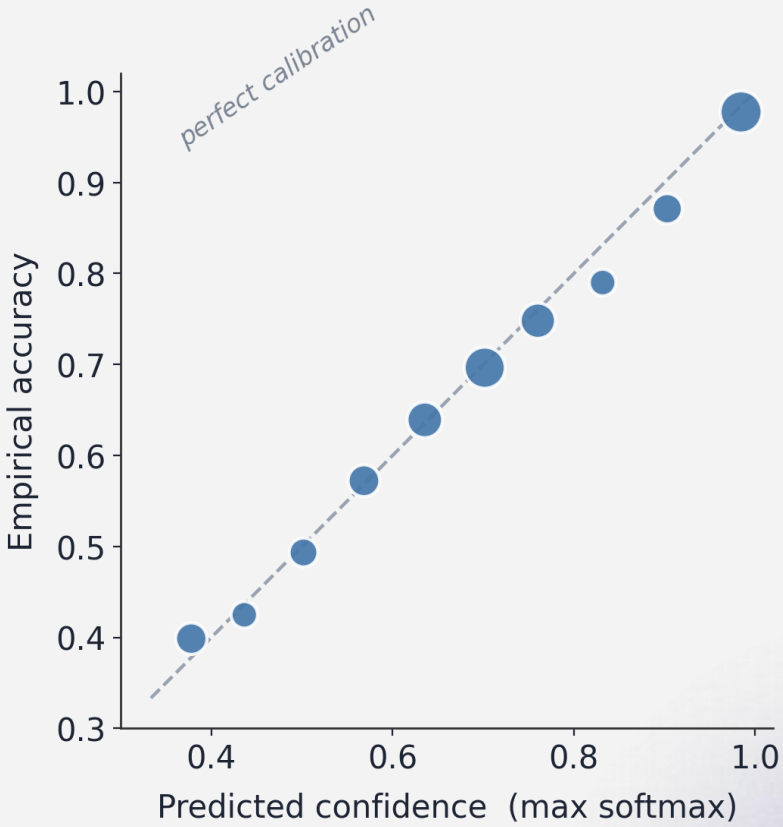
- **Rows**: generating model, **columns**: model selected by network
- Direct difference and hyperboloid recover cleanly (0.73, 0.84)
- Hyperbolic leaks a third of its mass into hyperboloid because it IS hyperboloid at s = 1



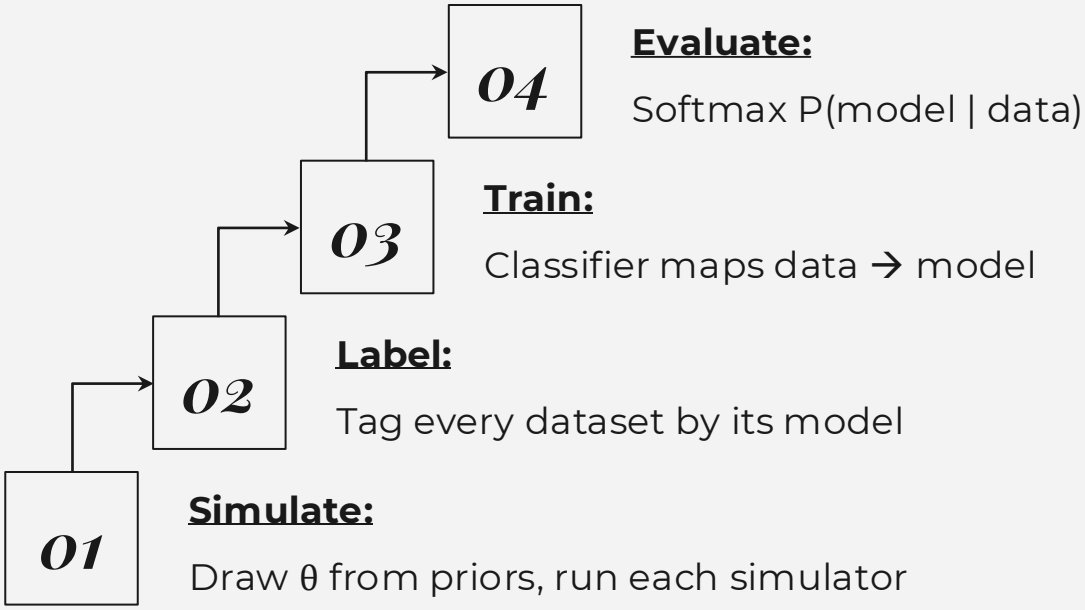
Softmax outputs honest probabilities

When the network says 70% confident, it's right about 70% of the time.

- Points sit on the diagonal across the whole confidence range.
- So a probability of 0.9 can be trusted as evidence strength



What to remember



Thanks!

Do you have any questions?

ksokratous@missouri.edu

lazyneuron.com