USDA

United States Department of Agriculture

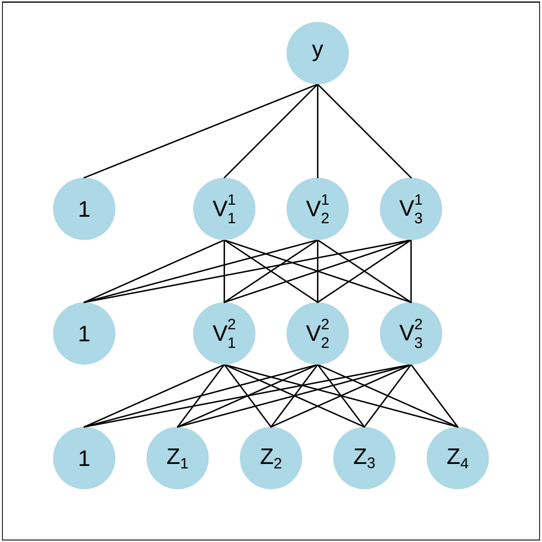
Semiparametric neural networks

Andrew Crane-Droesch

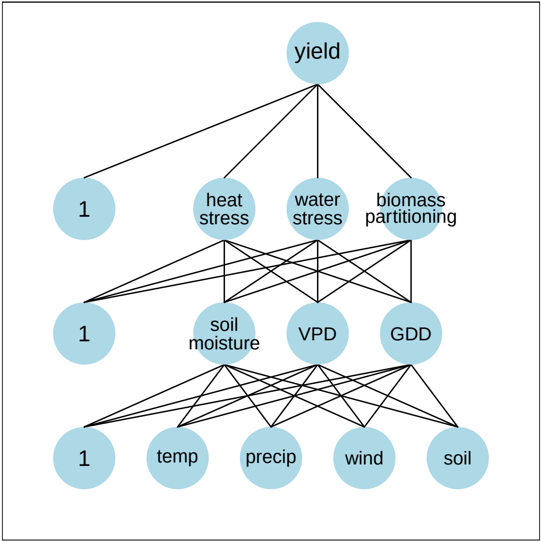
The Workshop in Environmental Economics and Data Science,
Portland OR, March 2019

This research was supported by the intramural research program of the U.S. Department of Agriculture, Economic Research Service.

Neural networks



Representation Learning



Neural networks

$$
\begin{aligned}
y &= \gamma^1 + \boldsymbol{V}^1\Gamma^1 + \epsilon \\
\boldsymbol{V}^1 &= a\left(\gamma^2 + \boldsymbol{V}^2\boldsymbol{\Gamma}^2\right) \\
\boldsymbol{V}^2 &= a\left(\gamma^3 + \boldsymbol{V}^3\boldsymbol{\Gamma}^3\right) \\
&\vdots \\
\boldsymbol{V}^L &= a\left(\gamma^L + \boldsymbol{Z}\boldsymbol{\Gamma}^L\right)
\end{aligned}
$$

- y – a (continuous) outcome
- ϵ – additive error
- $\boldsymbol{Z}$ – data
- $\boldsymbol{V}^l$ – “nodes”: derived variables
- $a()$ – the “activation function”. Maps the real line to some subset of it. Modern nets use variants of the ReLU: $a(x) = max(0, x)$
- Dimension of $\boldsymbol{\Gamma}^{1:L}$ controls number of nodes per layer

Semiparametric neural nets

- The top layer of a neural net is an OLS regression in derived variables $\boldsymbol{V}$

$$y = \gamma + \boldsymbol{V}\Gamma + \epsilon$$

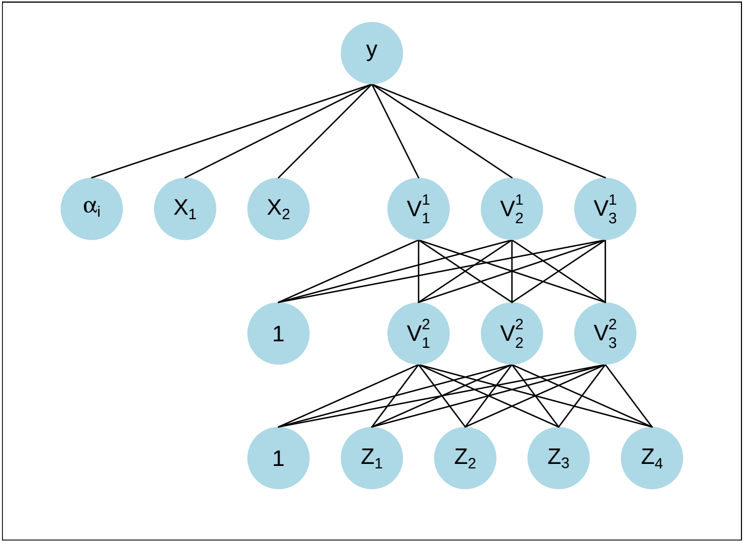
- It is simple to add linear terms to the model, where a linear-in-parameters relationship is known to be appropriate:

$$y = \gamma + \boldsymbol{X}\beta + \boldsymbol{V}\Gamma + \epsilon$$

- Likewise, panel structure can be accounted-for by adding unit-specific intercepts at the top level:

$$y_{it} = \alpha_i + \boldsymbol{X}_{it}\beta + \boldsymbol{V}_{it}\Gamma + \epsilon_{it}$$

Semiparametric and panel neural nets



Why might you want to do this?

- **Statistical efficiency**
- When the goal is to do prediction, but you're working in a topic area in which people have been doing inference and understand the data-generating process somewhat
- When a purely nonparametric model has a hard time representing specific kinds of structure:
  - Longitudinal structure
  - Secular trends
  - Response heterogeneity by specific, known features
- (With caveats) for certain sorts of causal inference tasks
  - 2SLS first stages
  - For predictive models that need a few interpretable marginal effects (maybe)

The method

- It has been implemented in the R package `panelNNET`, and a paper on the method has been published in *Environmental Research Letters*

IOPscience Journals Books Publishing Support Login Search IOPs

Environmental Research Letters

ACCEPTED MANUSCRIPT • OPEN ACCESS

Machine learning methods for crop yield prediction and climate change impact assessment in agriculture

Andrew Crane-Droesch[1]

Accepted Manuscript online 14 September 2018 • Not subject to copyright in the USA. Contribution of United States Department of Agriculture

- It achieves state-of the art predictive skill in its domain, outperforming fully-nonparametric neural nets as well as parametric statistical models

Environmental Research Letters

ACCEPTED MANUSCRIPT • OPEN ACCESS

Machine learning methods for crop yield prediction and climate change impact assessment in agriculture

Andrew Crane-Droesch[1]

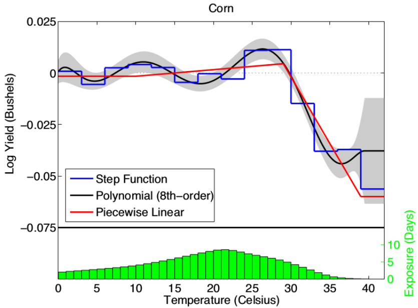
Accepted Manuscript online 14 September 2018 •

Model	Bagged	$\widehat{\mathrm{MSE}}_{oob}$
Parametric	no	367.9
Semiparametric neural net	no	292.8
Parametric	yes	334.4
Fully-nonparametric neural net	yes	638.6
Semiparametric neural net	yes	251.5

Baseline parametric yield model

$$y_{it} = \alpha_i + \sum_r \mathrm{GDD}_{rit}\beta_r + \boldsymbol{X}_{it}\boldsymbol{\beta} + \epsilon_{it}$$

- Pioneered by Schlenker & Roberts (2009)



→ small shifts in heat have severe impacts on yields

Semiparametric Specification

$$
\begin{aligned}
y_{it} &= \alpha_i + \sum_r \mathrm{GDD}_{rit}\beta_r + \boldsymbol{X}_{it}\boldsymbol{\beta} + \boldsymbol{V}_{it}\boldsymbol{\Gamma} + \epsilon_{it} && \boldsymbol{\Gamma}\text{: } 100 \times 1 \\
\boldsymbol{V}_{it}^1 &= a\left(\gamma^2 + \boldsymbol{V}_{it}^2\boldsymbol{\Gamma}^2\right) && \boldsymbol{\Gamma}^2\text{: } 100 \times 100 \\
&\vdots \\
\boldsymbol{V}_{it}^{10} &= a\left(\gamma^{10} + \boldsymbol{W}_{it}\boldsymbol{\Gamma}^{10}\right) && \boldsymbol{\Gamma}^{10}\text{: many } \times 100 \\
\boldsymbol{W}_{it} &= \left[\boldsymbol{Z}^{fixed}, \mathcal{C}\left(\boldsymbol{Z}^{daily}\right)\right]
\end{aligned}
$$

- *Identical* to parametric model, with addition of 100-node neural network layer
- 10 layers, 50 nodes each. **MANY parameters (>>N!)** Regularization is extremely important.
- $\mathcal{C}()$ represents a 1D **convolutional** layer
  - In this application, convolutional layers constrain the nonlinear interactions of daily weather to adjacent days

Training: backpropagation

Typical loss function for continuous-variable prediction problems is the L2-penalized squared error loss:

$$\underset{\boldsymbol{\theta}}{argmin}\left((y-\hat{y})^2+\lambda\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{\theta}\right)$$

where $\boldsymbol{\theta} \equiv vec(\Gamma^1, \Gamma^2, ..., \Gamma^L)$ and λ is a tunable hyperparameter controlling model complexity

Other hyperparameters include dropout rate, batch size, learning rate, etc.

No closed-form solution! Instead, training is done by (mini-batch) gradient descent

Goal is a parameter set ($\boldsymbol{\theta}$) that predicts yields of years that were withheld from the training sample

The OLS trick

Loss function can be recast as

$$\underset{\boldsymbol{\theta}}{argmin} \, (y - \hat{y})^2 \text{ s.t. } \hat{\boldsymbol{\theta}}^{\boldsymbol{T}} \hat{\boldsymbol{\theta}} \leq c$$

- λ thus implies a "budget" for deviation from zero within of $\hat{\boldsymbol{\theta}}$
- Because gradient descent is inexact, top-level parameters of neural net are not those that minimize loss function

$$\underset{\Psi}{min} \, (\boldsymbol{y} - \boldsymbol{W}\Psi)^T (\boldsymbol{y} - \boldsymbol{W}\Psi) + \tilde{\lambda}\Psi^T\Psi$$

- where
  - $\boldsymbol{W} \equiv [X, V]$
  - Ψ indicates the portion of $\hat{\boldsymbol{\theta}}$ corresponding to the top level of the network
  - $\tilde{\lambda} > \lambda$ is the penalty corresponding to the "budget" that is "left over" after fitting the lower level parameters which generate $\boldsymbol{V}$

The OLS trick

Can calculate the implicit $\tilde{\lambda}$ for the top level of the neural network by minimizing

$$\underset{\tilde{\lambda}}{min}\left(\mathcal{B}^T\mathcal{B} - \Psi^{mT}\Psi^m\right)^2$$

where

$$\mathcal{B} = (\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{W} + \tilde{\lambda}I)^{-1}\boldsymbol{W}^T\boldsymbol{y}$$

- Replacing Ψ^m with $\mathcal{B}$ ensures that the sum of the squared parameters at the top level of the network remains unchanged
- The (top level of the) penalized loss function reaches its minimum subject to that constraint.
- Alternatively, separate λ's can be specified for each layer, and the OLS trick can pick the optimal value of λ for the top layer as often as desired.

Fixed and random effects

- Heterogeneous intercepts can be implemented in Keras by concatenating a tensor of dummies, and penalizing their weights if random effects are desired
    - Recall that ridge regression on cross-sectional dummies is equivalent to a random effects model (Harville 1977, 1978)
    - Keras/Tensorflow can handle estimation of thousands of dummy effects without issue on a standard laptop
- For the fixed-effects model (i.e.: where α_i is unpenalized), the closed form solution is faster and more precise:

$$\hat{\alpha} = (y_{it} - \bar{y}_i) - \left(\boldsymbol{X}_{it} - \bar{\boldsymbol{X}}_i\right)\hat{\beta} - \left(\boldsymbol{V}_{it} - \bar{\boldsymbol{V}}_i\right)\hat{\Gamma}$$

    - Like the OLS trick, these can be computed every few iterations and inserted into the model, short-circuiting gradient descent, and speeding convergence.

Implementation in Keras/Tensorflow/R

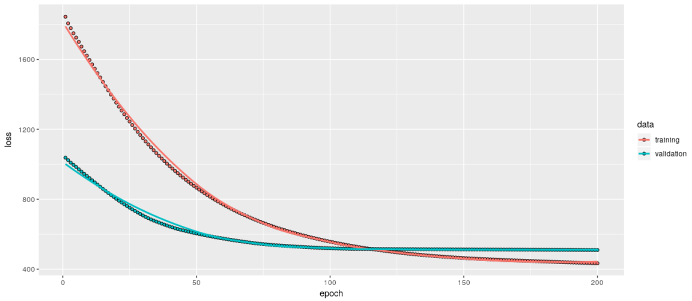
- Keras is a high-level API for various deep learning libraries
- Tensorflow is Google's deep learning library. Keras calls it "under the hood," and can run with others (Torch, Theano, CTNK, etc).
- RStudio has recently implemented a wrapper for Keras. It seems to just call Python from R.
- Most of the user community works in Python – more resources and support in that ecosystem

See `https://github.com/cranedroesch/SNN_python/blob/master/SNN_python.ipynb` for a fixed-effects implementation in python

Fitting a basic neural net in Keras through R

```r
library(keras)
library(dplyr)
X <- mtcars[,-1] %>% as.matrix
y <- mtcars[,1] %>% as.matrix
inp <- layer_input(shape = ncol(X))
# note mutability! not common in R
layers <- inp %>%
  layer_dense(units = 3, activation = "relu") %>%
  layer_dense(units = 1, activation = "linear")
model <- keras_model(inputs = inp, outputs = layers)
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_nadam()
)
```


```
history <- fit(model,
               x = X,
               y = y,
               verbose = 1,
               epochs = 200,
               validation_split = .2
)
plot(history)
```



Representing parametric structure in Keras

Say I wanted to include a parametric term in weight and cylinders, and let the rest get squashed into a single term

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

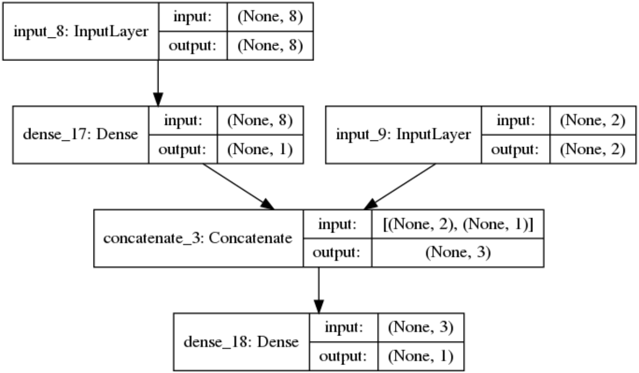
Representing parametric structure in Keras

```
"%ni%" <- Negate("%in%")
Xp <- mtcars[,c("wt", "cyl")] %>% as.matrix
Xnp <- mtcars[,colnames(mtcars) %ni% c("mpg", "cyl", "wt")] %>%
  as.matrix
npinp <- layer_input(shape = ncol(Xnp))
pinp <- layer_input(shape = ncol(Xp))
nplayers <- npinp %>%
  layer_dense(units = 3, activation = "relu")
merged <- layer_concatenate(list(pinp, nplayers)) %>%
  layer_dense(units = 1, activation = "linear")

model <- keras_model(inputs = c(pinp, npinp), outputs = merged)
model %>% compile(
  loss = ’mean_squared_error’,
  optimizer = optimizer_nadam()
)
```

Representing parametric structure in Keras

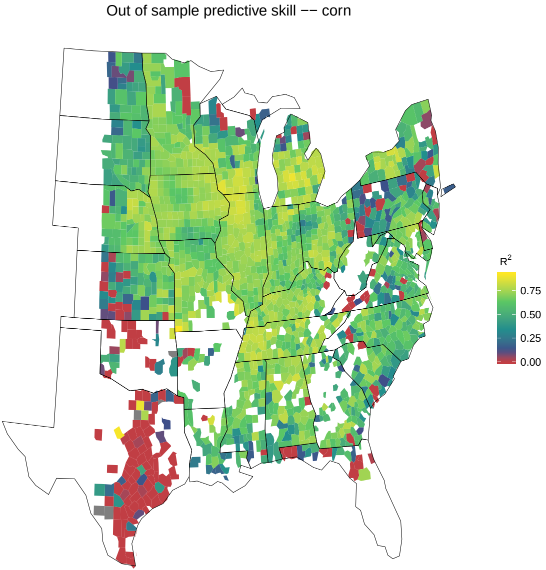
```r
library(kerasR)
plot_model(model, to_file = "model.png", show_shapes = T,
           show_layer_names = TRUE)
```



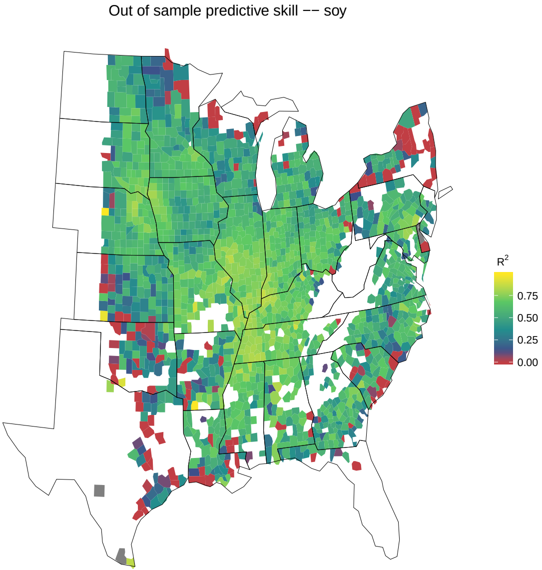
Application: dryland yield prediction

- Maize and soy yield data from the US east of 100W Longitude
- Historical weather data from METDATA (Abatzoglou 2013)
  - Daily observations of min/max temperature, min/max relative humidity, precipitation, wind speed, insolation
  - 4km resolution
  - Aggregated to counties and weighted by agricultural area
- Model is a semiparametric neural net with the following parametric terms
  - random (not fixed) effects
  - GDD variables
  - penalized cubic spline in total annual rainfall, time, and insolation
- Fit in Keras/R, developed to study SRM geoengineering

Model fit – Corn

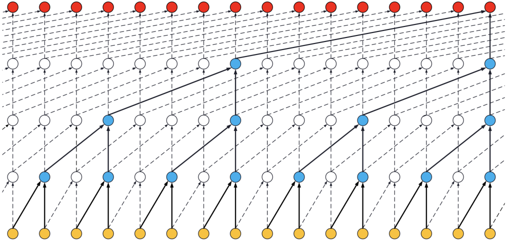


Model fit – Soy



Future work

- Moving beyond feed-forward neural nets: adding parametric time-series structure to Temporal Convolutional Networks



  - Initializing the parametric component using `GETS/IIS`
  - Applying it to spatio-temporal forecasting problems (adaptation?)
- Explicitly modeling response heterogeneity through interactive, multi-headed architectures

Big picture

- Keras and Tensorflow are powerful and well-made
- Neural nets are general: workhorse econometric models are nested within them (with more or less hacking)
- They're a complement to the work that empirical economists have always done

Resources

- See here for a Python/Keras implementation of a SNN: `github.com/cranedroesch/SNN_python/blob/master/SNN_python.ipynb`
- Code for replicating the ERL paper: `github.com/cranedroesch/ML_yield_ERL`
- panelNNET source: `github.com/cranedroesch/panelNNET`