Mapping Poverty with Neural Networks

Jean, Burke, Xie, Davis, Lobell, & Ermon (2016)

Graham Joncas (戈雷)

17210680479@fudan.edu.cn

https://github.com/gjoncas

School of Economics
Fudan University

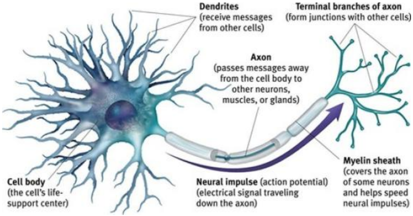
Topics in Development Economics
April 4, 2018

Outline

Introduction to Neural Nets

- One of the most commonly used machine learning algorithms
- Many applications in economics – used since White (1988)
- Avoid unrealistic assumptions about data-generating process
- Make non-linear statistics accessible to non-statisticians
- ‘Deep learning’ is just neural nets with lots of hidden layers

Neural Networks & The Brain



- Dendrites: accept inputs
- Axons: transmit output
- If combined inputs meet a threshold, sends an output

"The cerebral cortex contains about 10^{11} neurons, which is approximately the number of stars in the milky way. [...]

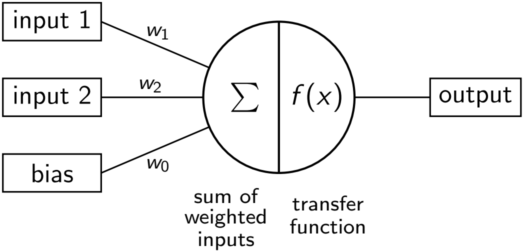
Each neuron is connected to 10^3 to 10^4 other neurons. In total, the human brain contains approximately 10^{14} to 10^{15} interconnections."

— Jain, Mao, Mohiuddin (1996: 33)

Brain vs. Computer

	Brain	Computer
Number of Processing Units	$\approx 10^{11}$	$\approx 10^{9}$
Type of Processing Units	Neurons	Transistors
Form of Calculation	Massively Parallel	Generally Serial
Data Storage	Associative	Address-based
Response Time	$\approx 10^{-3}$s	$\approx 10^{-9}$s
Processing Speed	Very Variable	Fixed
Potential Processing Speed	$\approx 10^{13}$ FLOPS	$\approx 10^{18}$ FLOPS
Real Processing Speed	$\approx 10^{12}$ FLOPS	$\approx 10^{10}$ FLOPS
Resilience	Very High	Almost None
Power Consumption per Day	20W	300W

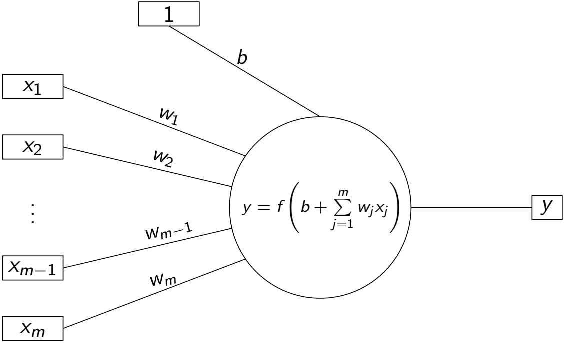
Abstract Model



- Simplest is *perceptron*: binary input and output, one neuron
- Inputs (x_i) have weights (w_i); bias is like β_0 (where $x_0 = 1$)
- Neuron sums weighted inputs ($s = \sum w_i x_i$)
- Transforms sum into output according to threshold θ:

$$f(s) = \begin{cases} 1, & \text{if } s > \theta \\ 0, & \text{otherwise} \end{cases}$$

Machine Learning



- Supervised learning: given correct outputs, learn w_i for inputs
- *Training set* $\rightarrow$ find weights; *test set* $\rightarrow$ see if results still hold
- Supervised $\rightarrow$ pattern classification; unsupervised $\rightarrow$ clustering

Linear Separability: OR vs. XOR

With linear regression, we want to draw a line that best fits all data points (minimizes $\sum u_i^2$).

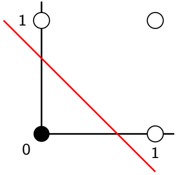
Neural nets: want to draw a line that separates different classes

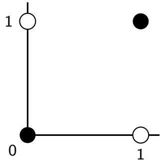
Classes X_0 and X_1 are linearly separable if $\exists \vec{w}, k$ such that:

- For all $x \in X_0$, $\sum_{i=1}^{n} w_i x_i > k$
- For all $x \in X_1$, $\sum_{i=1}^{n} w_i x_i < k$

$\vee$	1	0
1	1	1
0	1	0

$\veebar$	1	0
1	0	1
0	1	0





Learning with Neural Nets

Widrow-Hoff Learning Rule

$\vec{w}_{i+1} = \vec{w}_i + \eta(z_i - y_i)\vec{x}_i$

Learning the classification for logical OR ($\vee$)

Use input vectors as training set, including $x_0 = 1$:

$$\vec{x}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \vec{x}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \vec{x}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \vec{x}_4 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Also given correct classification: $z_1 = z_2 = z_3 = 1$, $z_4 = 0$

Perceptron classifies object as 1 iff $w_0x_0 + w_1x_1 + w_2x_2 > 0$, else 0

Use starting vector $\vec{w}_1 = (0, 0, 0)'$, set learning rate $\eta = 1$

Learning Logical OR ($\vee$)

1. $\vec{w}_1$ gives $y_1 = 0 \neq z_1$. So let's change the weights:

$$\vec{w}_2 = \vec{w}_1 + (z_1 - y_1)\vec{x}_1$$
$$\vec{w}_2 = (0,0,0)' + (1-0)(1,1,0)' = (1,1,0)'$$

2. $\vec{x}_2$ is correctly classified with the weight vector $\vec{w}_2$.

$$1 \times 1 + 1 \times 0 + 0 \times 1 = 1 > 0 \ \rightarrow \ y_2 = z_2 = 1$$

3. $\vec{x}_3$ is correctly classified with the weight vector $\vec{w}_2$.

$$1 \times 1 + 1 \times 1 + 0 \times 1 = 2 > 0 \ \rightarrow \ y_3 = z_3 = 1$$

Learning Logical OR ($\vee$)

4. For $\vec{x}_4$ is $\vec{w}_2'\vec{x}_4 = 1$, so we have to change the weights again:

$$\vec{w}_3 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + (0-1)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

5. $\vec{x}_1$ is now used as input and is correctly classified.
6. Since $\vec{w}_3'\vec{x}_2 = 0 \neq z_2$:

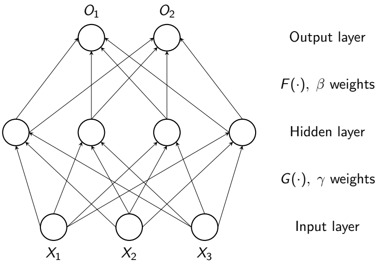
$$\vec{w}_4 = (0,1,0)' + (1-0)(1,0,1)' = (1,1,1)'$$

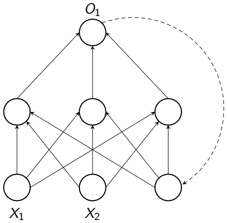
7. Since $\vec{w}_4'\vec{x}_3 = 3 > 0$, $\vec{x}_3$ is correctly classified.
8. $\vec{x}_4$ is incorrectly classified as 1, so that

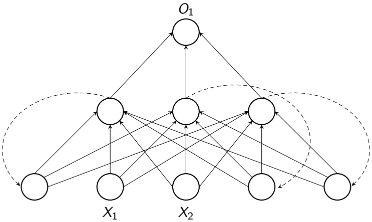
$$\vec{w}_5 = (1,1,1)' + (0-1)(1,0,0)' = (0,1,1)'$$

Our perceptron has learned the correct weights for OR: $(0,1,1)'$

Neural Nets – Feed-forward vs. Recurrent Learning

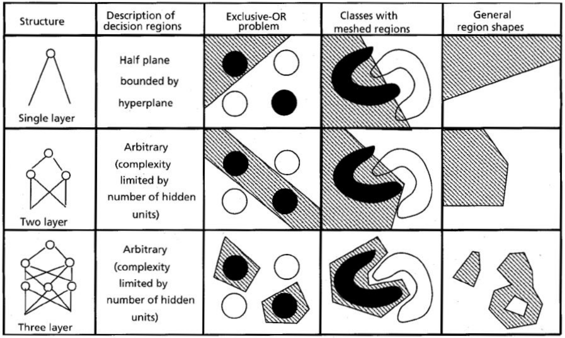






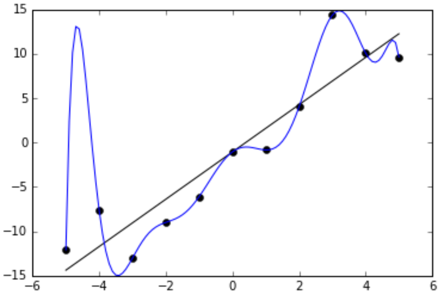
- ‘Hidden layers’ are nodes not connected to input or output
- Feed-forward: only goes forward
- Recurrent: feedback into inputs

Hyperplane Separation



- Adding hidden layers lets us draw more complex shapes
- General case: search for hyperplane to separate each class
- But: adding too many hidden layers just memorizes the data

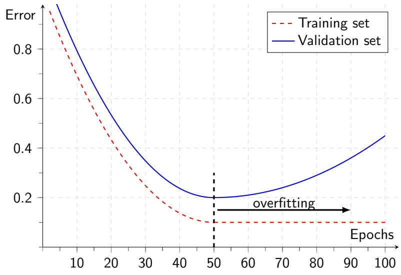
Overfitting



More complexity does not mean better prediction

- Blue function is 10th-degree polynomial with exact fit
- Black function is OLS regression line
- New data likely closer to regression line than fitted line

Dealing with Overfitting



- Training set: infer weights
- Validation set: check if $\vec{w}$ holds outside training data

Trade-off between bias & variance:

- model bias - systematic error in learning underlying relations among variables
- model variance - how well a model generalizes elsewhere

Regression may misrepresent true functional form (high bias), but generalize well (low variance)

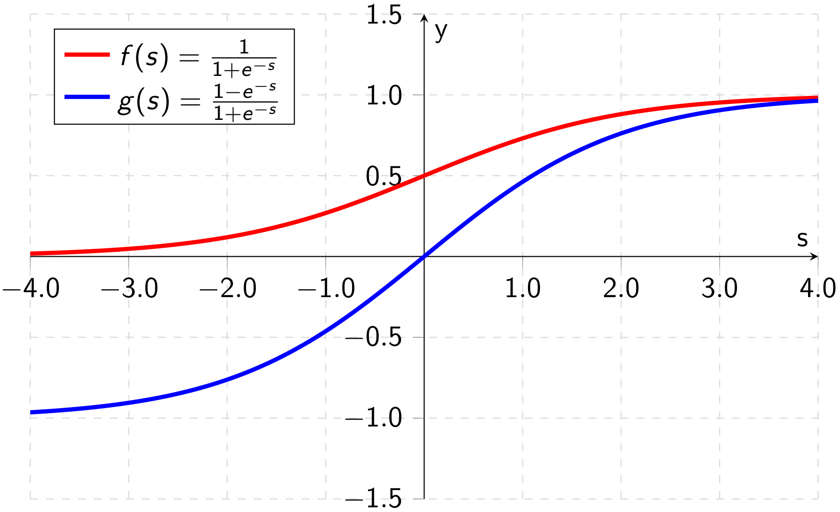
Neural net may depend too much on data (low bias), generalize poorly (high variance)

Analogies with Regression

Linear Regression	Neural Net
$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n + \varepsilon$	$y = f\left(b + \sum_{i=1}^{n} w_i \cdot x_i\right)$

Neural Net model	Statistical equivalent
Single linear perceptron	Linear regression
Simple linear perceptron	Multivariate multiple linear regression
Simple nonlinear perceptron	Logistic regression
Adaline network	Linear discriminant function
Functional link network	Polynomial regression
Multilayer perceptron	Simple nonlinear or multivariate multiple nonlinear regression

Activation Functions



Backpropagation Algorithm

Based on Widrow-Hoff learning rule, extended for multiple neurons

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}, \text{ where } \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

1. Start with random weights, forward-propagate to get output
2. Get error as difference from true answer: $E_i = (F_i(x, w) - z_i)^2$
3. Get partial derivatives of error w.r.t. each w_{ij}, add up for total
   → Use sigmoid function because derivative is easy
4. Update weights by equation above, repeat until E minimized

Convolutional Neural Nets – Computer Vision

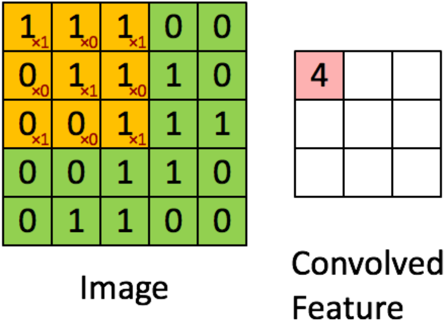
What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What Computers See

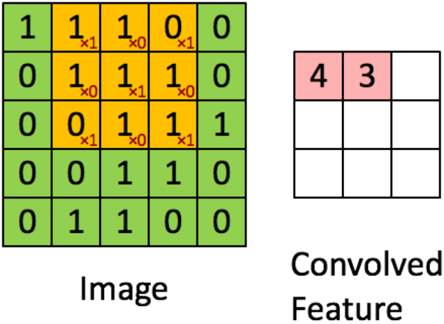
- Need numerical inputs → convert image to numbers
- Convert into matrix of RGB values (0–255) for each pixel
- Now we can perform matrix operations on images

Convolution



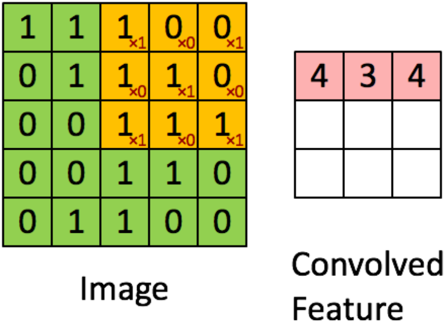
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

Convolution



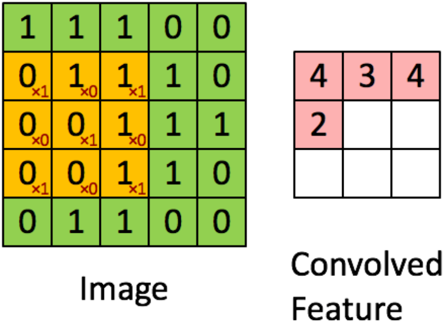
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

Convolution



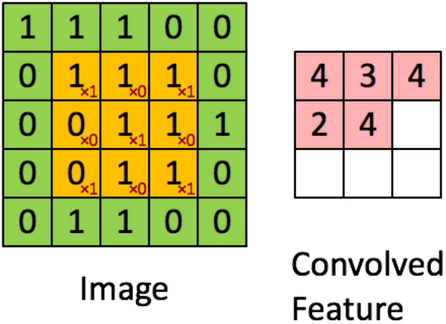
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

Convolution



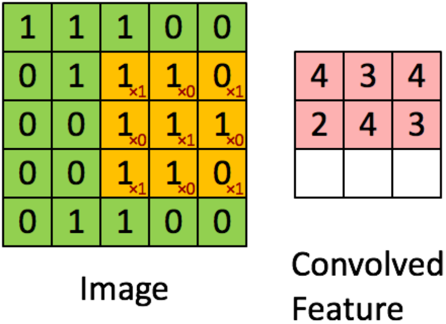
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow ‘filter’ multiplies element-wise at each position
- Output matrix is ‘feature map’, can identify edges, curves, etc.

Convolution



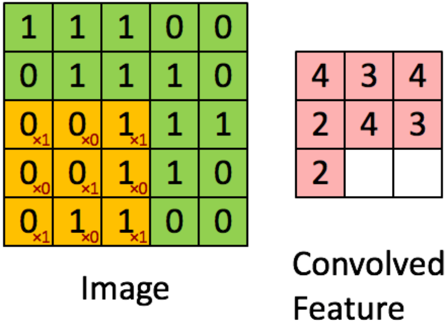
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow ‘filter’ multiplies element-wise at each position
- Output matrix is ‘feature map’, can identify edges, curves, etc.

Convolution



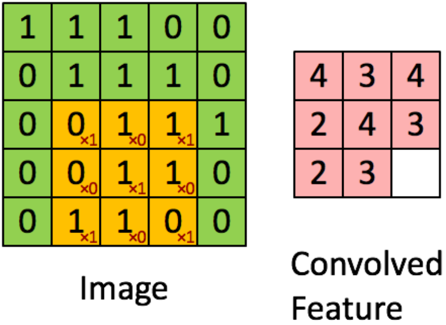
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

Convolution



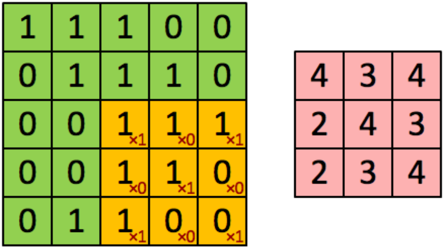
- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow ‘filter’ multiplies element-wise at each position
- Output matrix is ‘feature map’, can identify edges, curves, etc.

Convolution



- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

Convolution



Image

Convolved Feature

- Green matrix represents matrix of image (each pixel 0 or 1)
- Yellow 'filter' multiplies element-wise at each position
- Output matrix is 'feature map', can identify edges, curves, etc.

复旦大学
Fudan University

Filters & Matrices

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Different filter matrices identify different types of features

- e.g. edges, curves, colors, sharpening, blurring

Convolution operation captures local dependencies in image

- translation invariant: output strength depends not on where features are located, but only that they're present
- e.g. CNN will recognize cats, even if in different positions

Jean et al. (2016) – Motivation

- Very few surveys we can use to construct poverty measures
- World Bank's Living Standards Measurement Study: 39/59 African countries had fewer than two surveys, 14/59 had none
- Demographic & Health Survey: 20/59 African countries had none, 19 more only had one — cost is $\sim$1 million USD each
- Surveys resisted by governments (to hide poor performance)

- Nightlights: hard to distinguish between poor, densely populated areas and wealthy, sparsely populated areas
- Mobile phone data relies on "disparate proprietary datasets"

Jean et al. (2016) – Structure

1. Pre-train neural net on ImageNet (database of labeled images)
2. Train neural net to predict nightlight intensities (DMSP data) based on daytime satellite imagery (Google Maps)

   "the model learns to 'summarize'...high-dimensional input daytime images as a lower-dimensional set of image features ...predictive of variation in nightlights" (2016: 791)

3. Train ridge regressions using cluster-level values from surveys (Y) + image features extracted from daylight imagery (X)

   $\rightarrow$ helps prevent overfitting on small training set

Pre-training on ImageNet

Recall: Backpropagation Algorithm

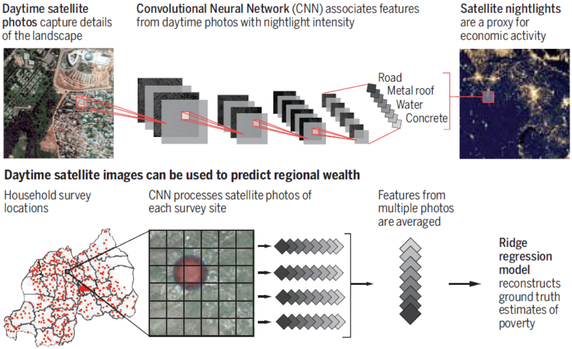
$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}, \text{ where } \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Pre-training as 'prior distribution'

Before, we used starting weights $\vec{w}_1$, which were just random.

- ImageNet: database of labeled images from 1000 categories
- Through classifying images, "the model learns to identify low-level image features such as edges & corners" (2016: 791)
- Further layers learn high-level features, e.g. textures & objects
- Scale, centering differ from satellite images — but good start

Transfer Learning



- CNN model has over 55 million parameters, 8 layers
- Large size due to pixels 400×400 ($= 160{,}000$), etc.
- Problem: surveys covers a few hundred locations, deep learning requires much more data $\rightarrow$ transfer learning

Data

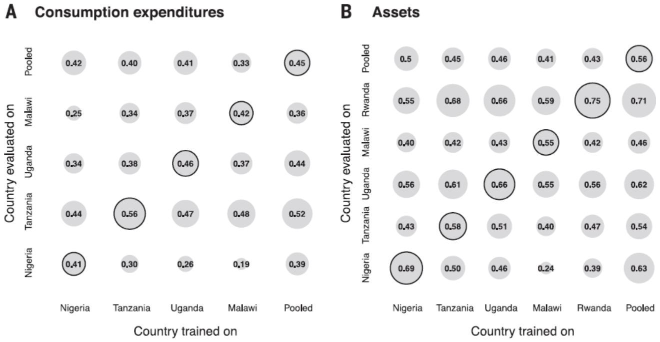
- World Bank's LSMS $\rightarrow$ index of consumption expenditure
- DHS $\rightarrow$ wealth index, e.g. bikes, TVs, housing material
- Noise added to locations due to privacy ($\pm$5km) $\rightarrow$ clusters
- Countries: Nigeria, Tanzania, Uganda, Malawi, Rwanda

- Nightlights: 400×400-pixel at zoom level 16 ($\sim$1km^2)
- Based on histogram, divide nightlight intensity into 3 classes
- Daylight: for each cluster, up to 100 images; 10km^2 area

Model learns meaningful features ('filters') on its own

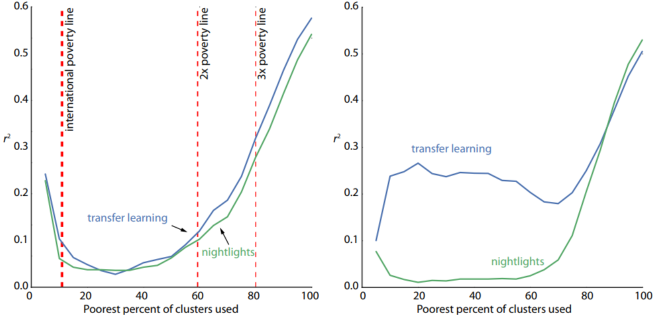
复旦大学 Fudan University

Ridge Regressions



- Pick 100 most useful image features, use as X's in regression
- Benefit: coefficients are easy to interpret across countries
- Model generalization: β's same across countries $\rightarrow$ low MSE
- Feature gen: same X's relevant (β's significant) $\rightarrow$ high R^2

Transfer Learning vs. Nightlights



- Better at predicting assets (right) than consumption (left)
- R^2 around 37-55% for consumption, 55-75% for asset wealth
- Pooled model (trained across all countries) has R^2 of 44-59%

Extensions: Multi-Resolution Images

- New dataset: Visible Infrared Imaging Radiometer Suite (VIIRS)
- Use images with zoom level 14, 16, 18 $\rightarrow$ different features
- Low-res images generalize better across countries (same β's)
- High-res images capture more variation (high R^2, diff β's)

Conclusions

"Why should the financial services industry, where mere dollars are at stake, be using more advanced technologies than the aid industry, where human life is at stake?" ∼Sendhil Mullainathan

- Machine learning methods far cheaper than official surveys
- Use cutting-edge tools from computer science (deep learning, computer vision) to solve problems in development economics
- Applies out-of-country → use for countries without surveys
- Policy: ensure that resources reach those with greatest need
- Can perhaps be extended for other UN Development Indexes

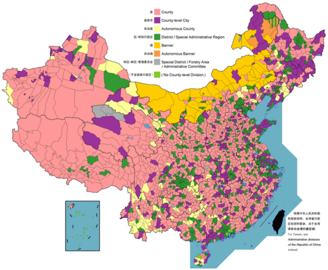
复旦大学 Fudan University

Machine Learning & ‘Patadata

‘Patadata - take metadata output from one model, use as data for other model

- Train neural net on satellite data
- Predict location of borders
- Find exceptions: where neural net says borders *should* be, but aren't

$$
\begin{tikzcd}
 & B \arrow[dr, "g"] & \\
A \arrow[ur, "f"] & & C
\end{tikzcd}
$$



Discrepancy must be due to unobservable factors — institutions, history, culture

References

- Jean, N., Burke, M., Xie, M., Davis, W., Lobell, D., Ermon, S. (2016). "Combining satellite imagery and machine learning to predict poverty." *Science* 353(6301), pp. 790-4 & S14-32
- Kim, J., Xie, M., Jean, N., Ermon, S. (2016). "Incorporating Spatial Context and Fine-grained Detail from Satellite Imagery to Predict Poverty." Working Paper. Retrieved from `http://cs.stanford.edu/~eix/multi_resolution.pdf`
- Xie, M., Jean, N., Burke, M., Lobell, D., Ermon, S. (2015). "Transfer Learning from Deep Features for Remote Sensing and Poverty." *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 1-11