Introduction to Deep Learning

Jason Mancuso

pandata

Professional Background

John Carroll UNIVERSITY

B.S. in Mathematics where I developed interest in machine learning and neural networks

Cleveland Clinic

Undergraduate research in machine learning for bio-informatics under Dr. Elena Manilich

Data scientist at Pandata, providing data science solutions to organizations across Northeast Ohio

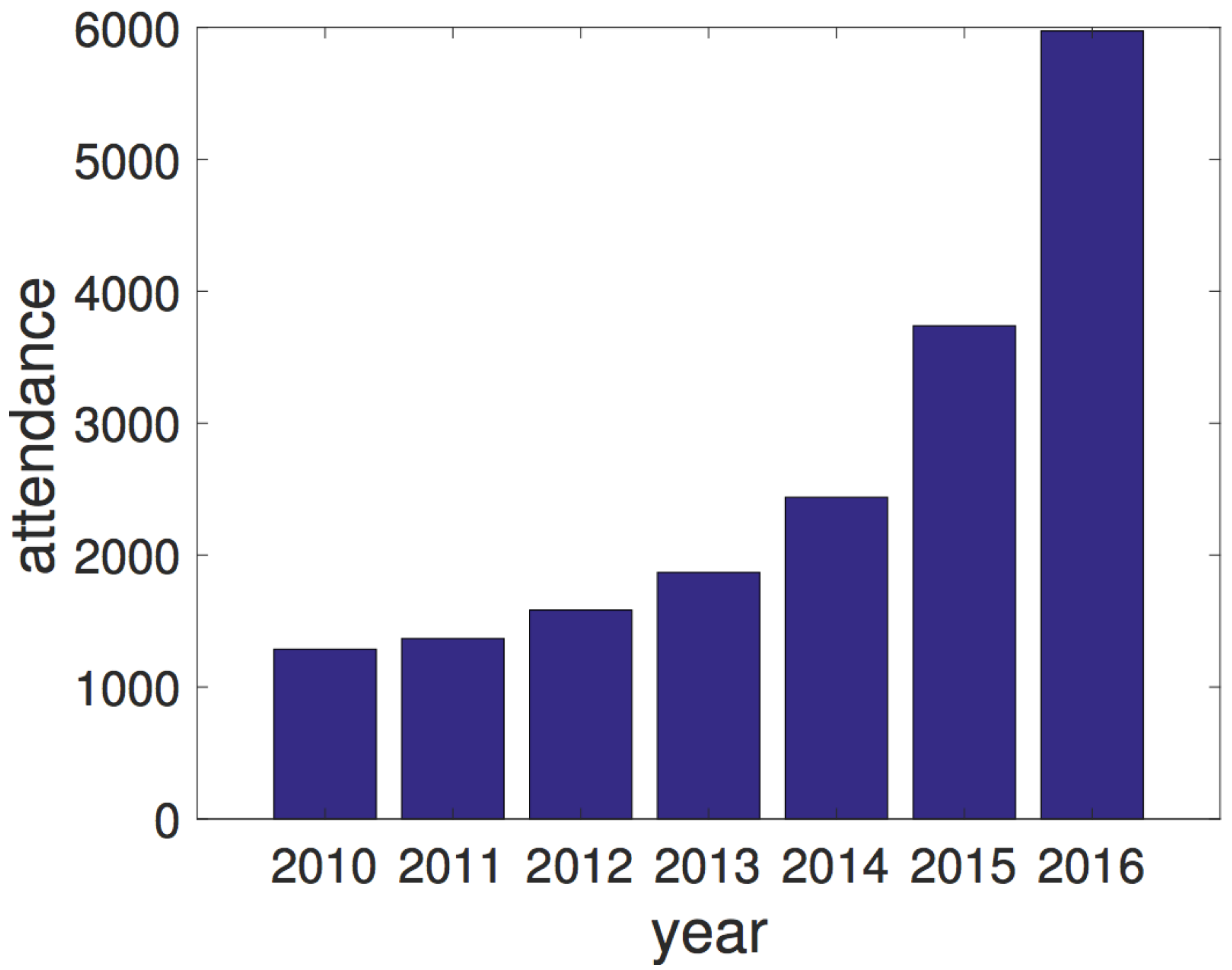
pandata

Mentor for the Artificial Intelligence Nanodegree since February 2017

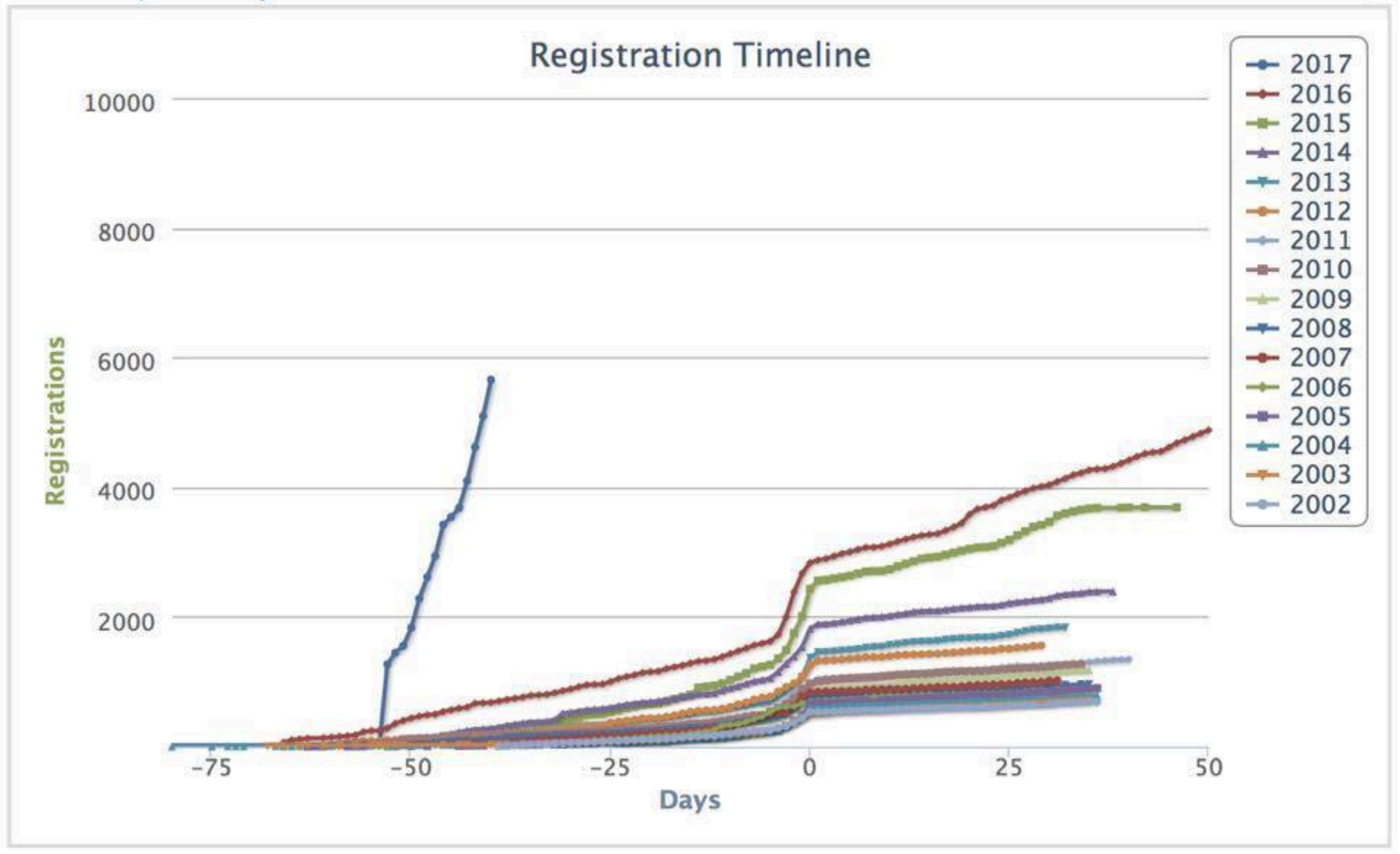
UDACITY

Conference on Neural Information Processing Systems (NIPS)

Attendance



Registration



* AnswerOn

Deep Learning

Using deep neural networks (or more precisely “distributed representations”) to perform machine learning

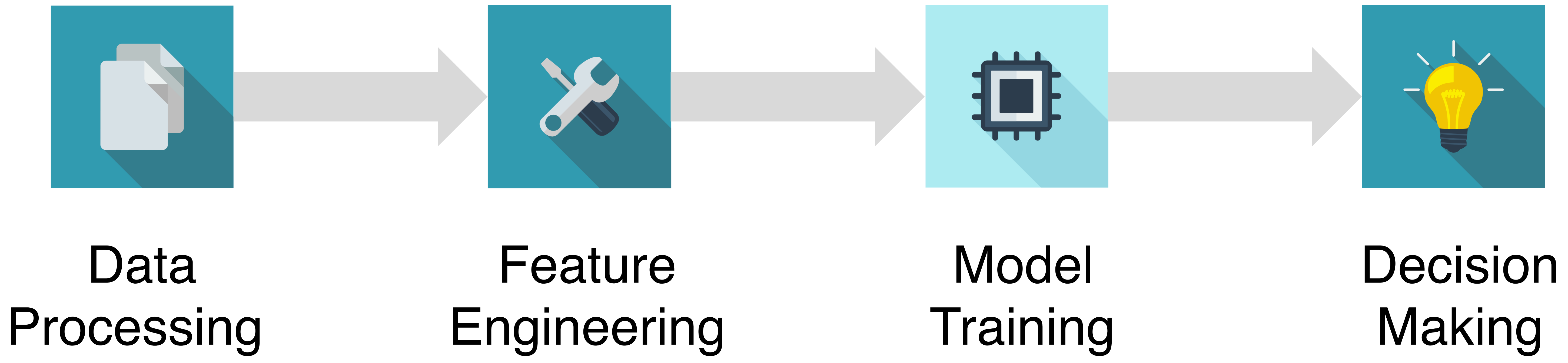
The fundamental innovation is around *representation*

- What the machine learns from directly
- Derived from raw data; it’s the data after it’s been transformed

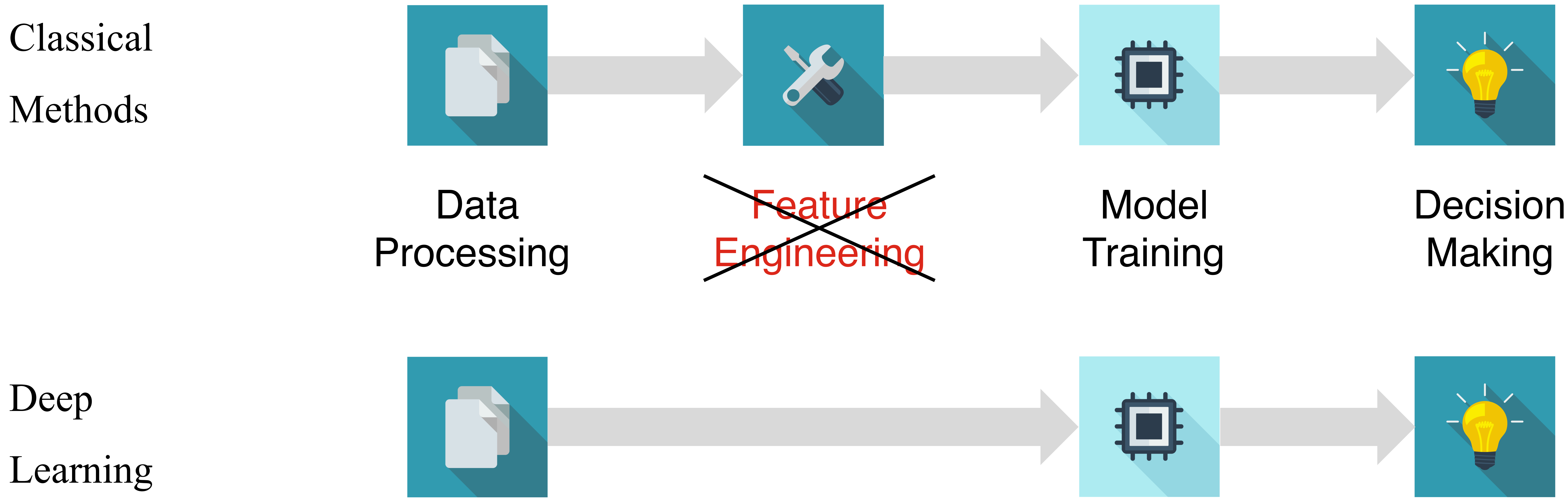
Machine Learning Workflow

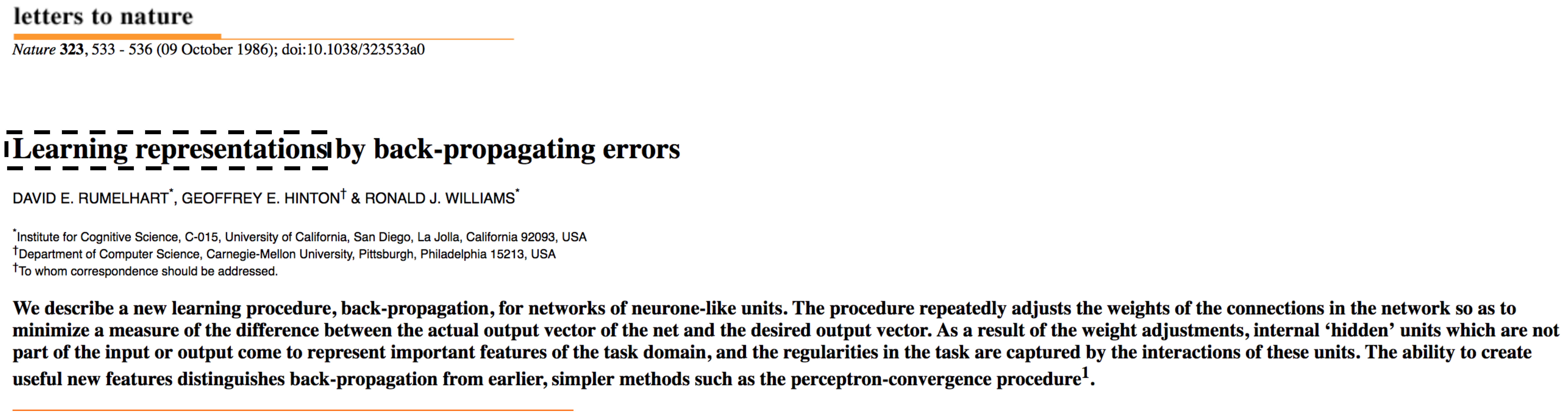
Classical

Methods



Machine Learning Workflow





letters to nature

Nature **323**, 533 - 536 (09 October 1986); doi:10.1038/323533a0

Learning representations by back-propagating errors

DAVID E. RUMELHART[*], GEOFFREY E. HINTON[†] & RONALD J. WILLIAMS[*]

[*]Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
[†]Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA
[†]To whom correspondence should be addressed.

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

How it Works

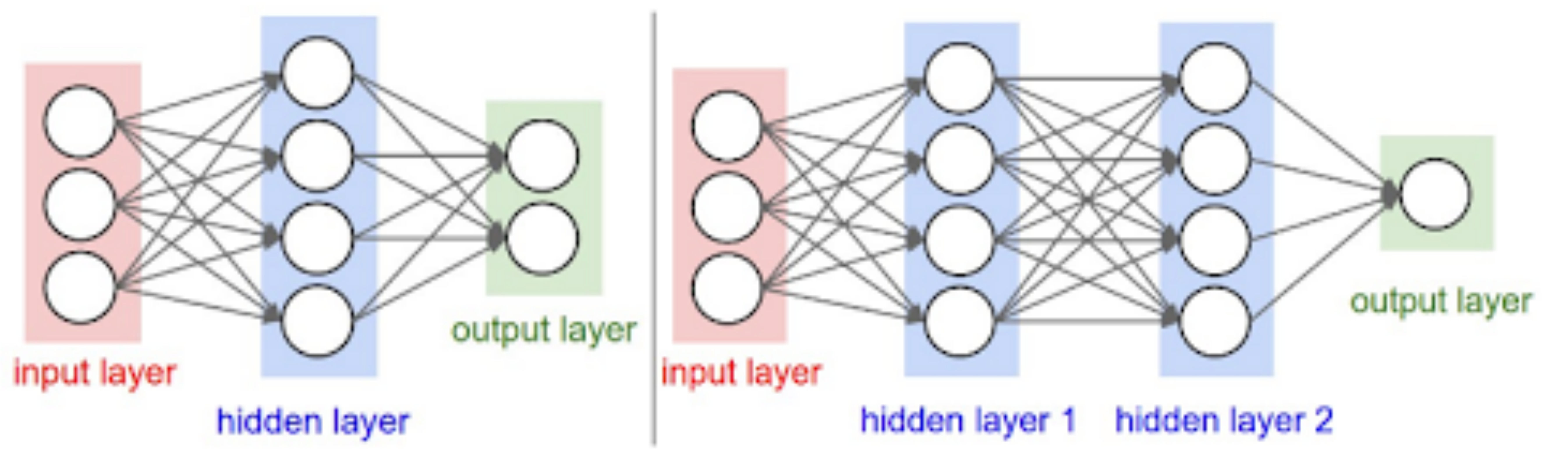
Supervised learning on generic data

- Given data *X* with labels *y*, find a function *f* such that *y = f(X)*

How it Works

Supervised learning on generic data

- Given data *X* with labels *y*, find a function *f* such that *y = f(X)*

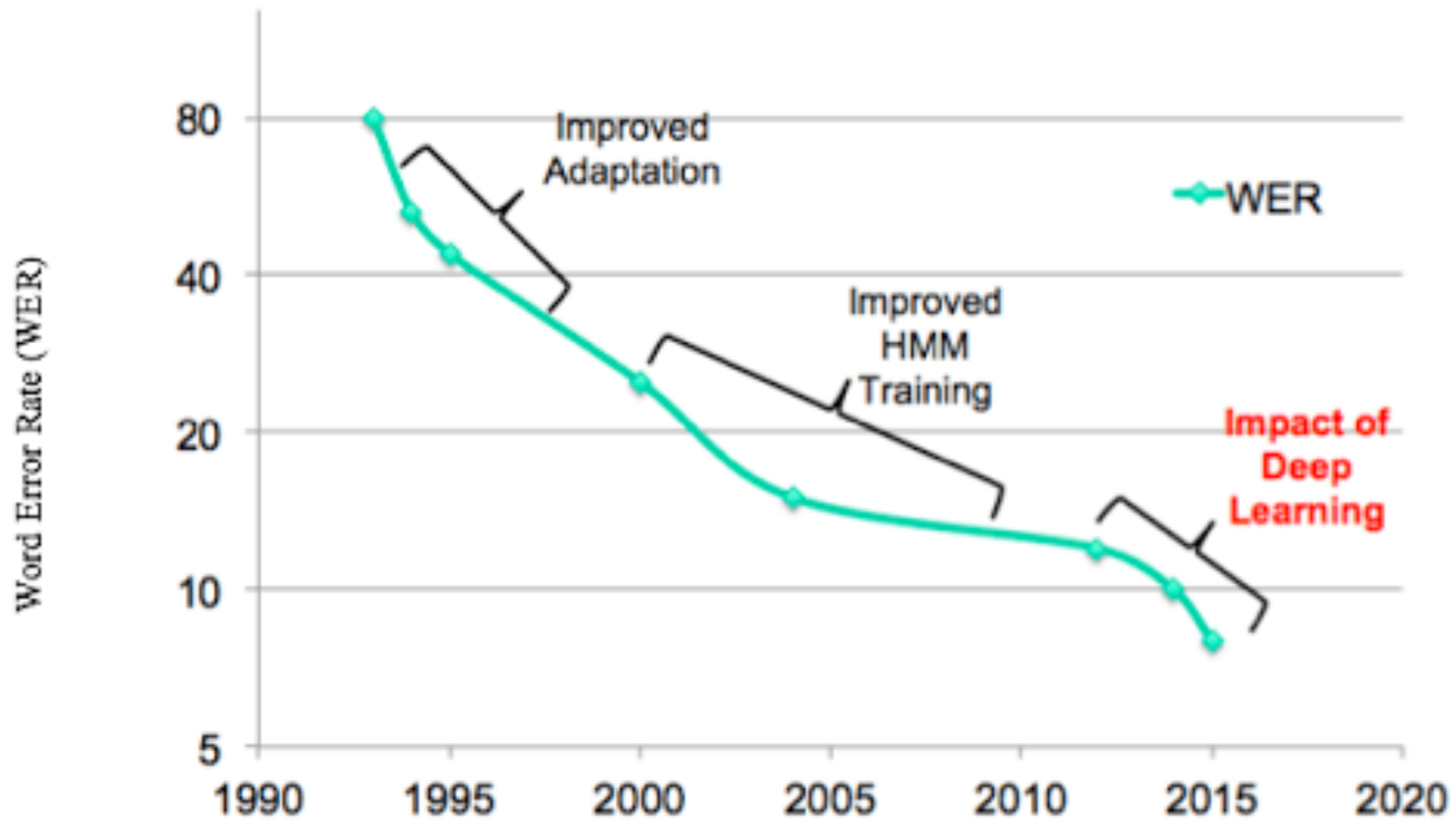


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

cs231n – Andrej Karpathy

ConvNetJS by Andrej Karpathy

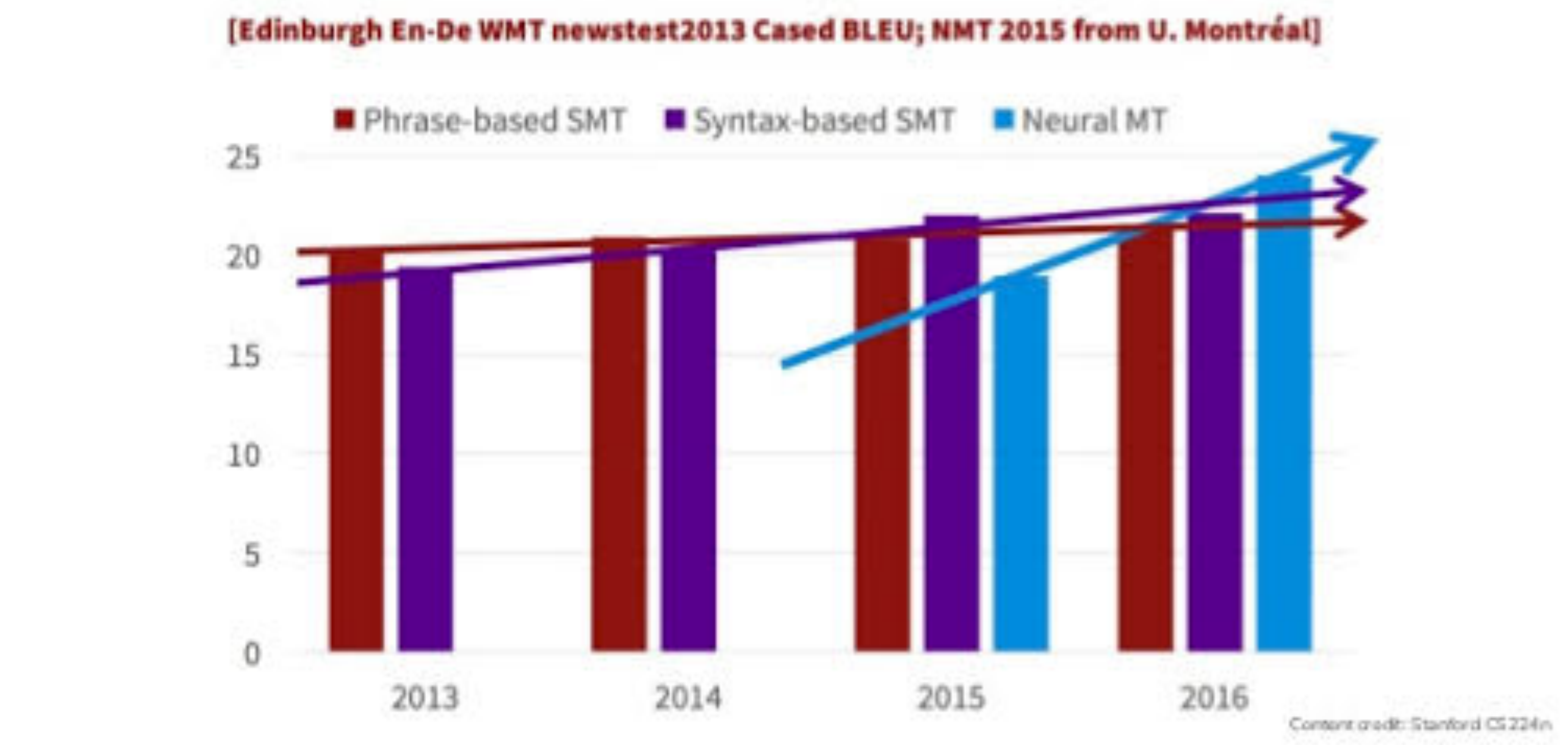
Wait, this works? – Speech Recognition



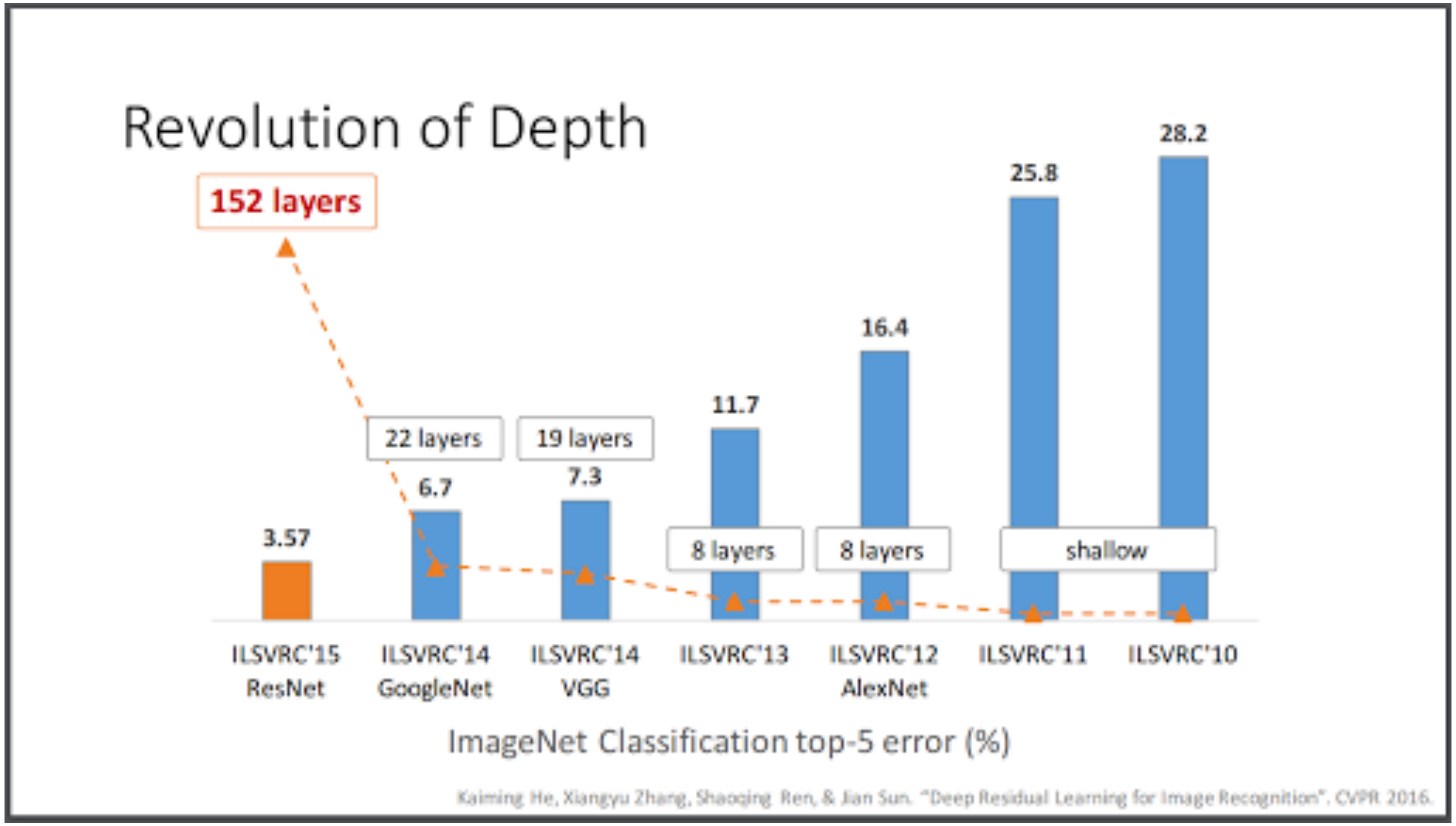
Ming Hsieh, USC

Wait, this works? – Language Translation

MT Performance



Wait, this works? – Image Recognition



ICML Tutorial -- Kaiming He, FAIR/Microsoft Research

Wait, this works? – Reinforcement Learning

Youtube

Why Deep Learning?

Recall supervised learning:

- Given data *X* with labels *y,* find a function *f* such that *y = f(X)*

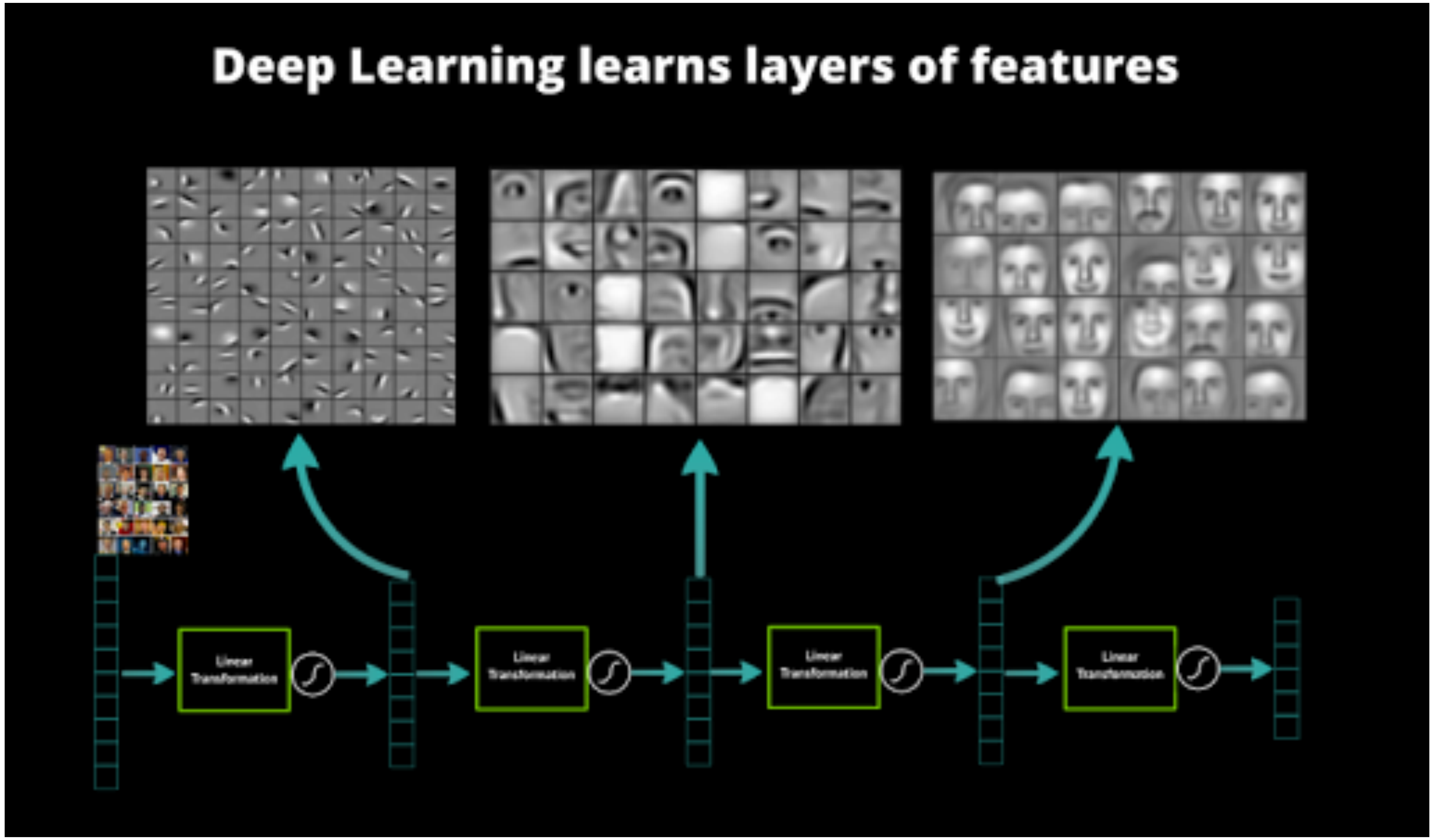
Universal Approximation Theorem: *any* continuous function can be (theoretically) approximated by a neural net with one hidden layer.

- This does not guarantee we can find the right network with the data and optimization algorithm we have

In practice, deeper networks show better performance with a few tricks*

- Intuitively, deeper layers facilitate higher-levels of abstraction, allowing for more complex representations

*batch/layer normalization or SELUs, ReLUs, advanced optimizers, dropout, skip connections, custom architectures like Inception module/stochastic depth, etc.



Dallin Akagi

Why Representation Learning?

- Handcrafted features often require that the representation be human-interpretable
- Computers use different representations than humans do, and we’re not good at guessing which representation will be useful
- Learned representations can be reused for different tasks (*transfer learning*)
- More theoretical justifications presented in Bengio et al.

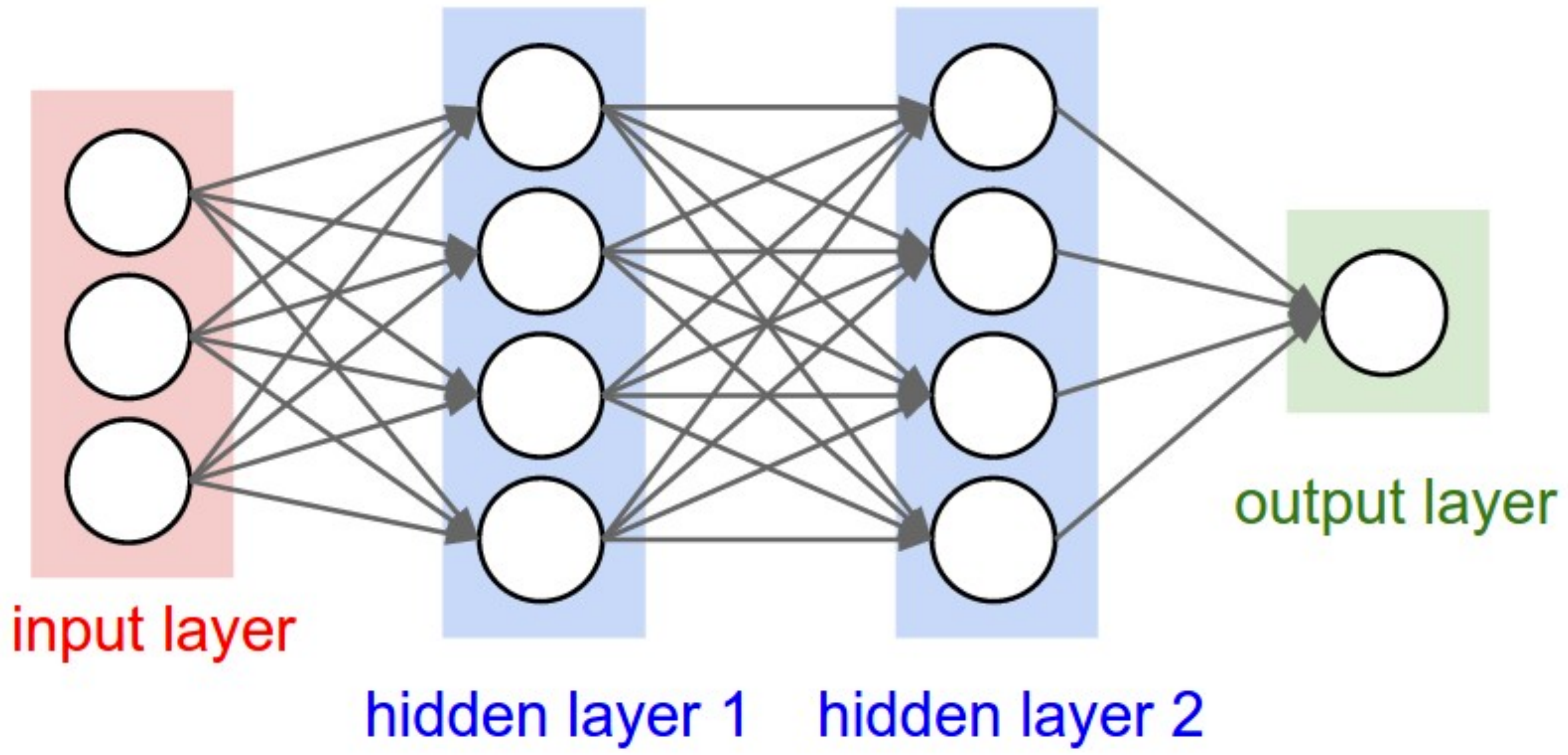
How to Train a Neural Network

- Two main steps
  - Choosing the architecture
  - Tuning the architecture

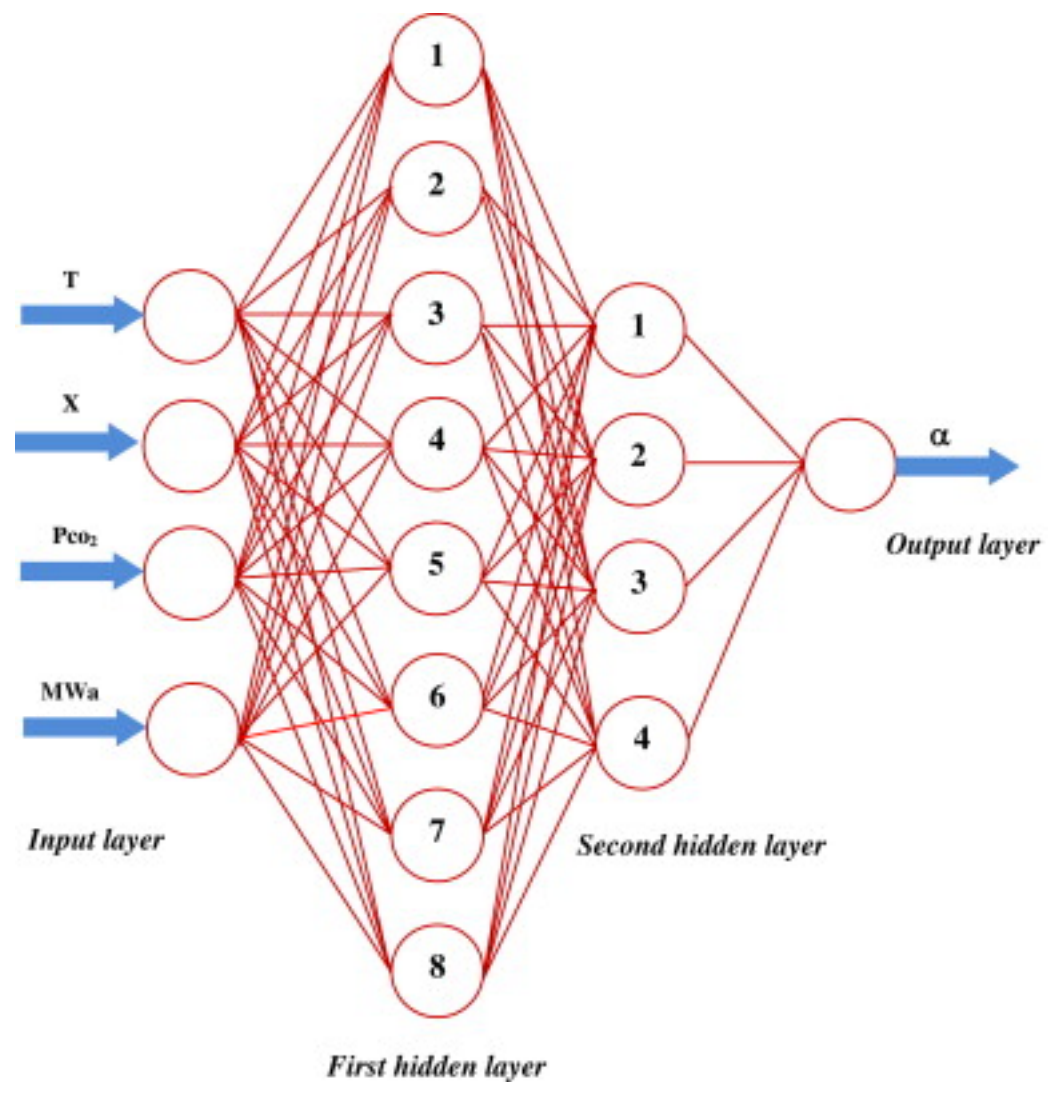
- Each step can be considered “hyperparameter tuning”
  - This can be automated, but hand-tuning is often faster/finds a better solution

Layer Structure

Block



Diavolo



Hamzehie et al.

Loss Functions

Regression

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

No output activation
(usually)

Classification

$$\begin{tikzcd} \hat{\mathbf{y}} \arrow[dr, bend left] & & \mathbf{y} \arrow[dl, bend right] \\ \begin{bmatrix}0.1\\0.5\\0.4\end{bmatrix} & D(\hat{\mathbf{y}},\mathbf{y}) = -\sum_j y_j \ln \hat{y}_j & \begin{bmatrix}0\\1\\0\end{bmatrix} \end{tikzcd}$$

Softmax or sigmoid
output activation

Three Modern Architectures

- The Standard
- The Residual Network
- The Self-Normalizing Network

The Standard

- Activation functions: ReLU -- f(x) = max(0,x)
- Batch normalization (batchnorm)
  - Normalize the inputs to each layer over the examples in each batch
  - Extra mean and standard deviation parameters that are learned over the course of training
  - Prevents internal covariate shift
  - Slight regularization effect (introduces slight bias in optimization, but slightly reduces variance)
- Dropout
  - During training, randomly drop each neuron with probability p
  - At test time, multiply the weights by p
  - Prevents co-adaptation of neurons, a common cause of overfitting
  - Maximal regularization with $p = 0.5$
  - $p = 0.2$ is a good first choice with batchnorm

The Residual Network

- Composed of many layers of residual blocks
- Each block usually consists of two or three ReLU + BN + Dropout layers, with a skip connection from the input to the output
- Skip connection allows gradients from later in the network to flow backwards more easily, unimpeded by intermediate networks
- Avoids vanishing gradient problem that affects very deep networks
- Tons of layers: common variants include 40, 60, 110, 152

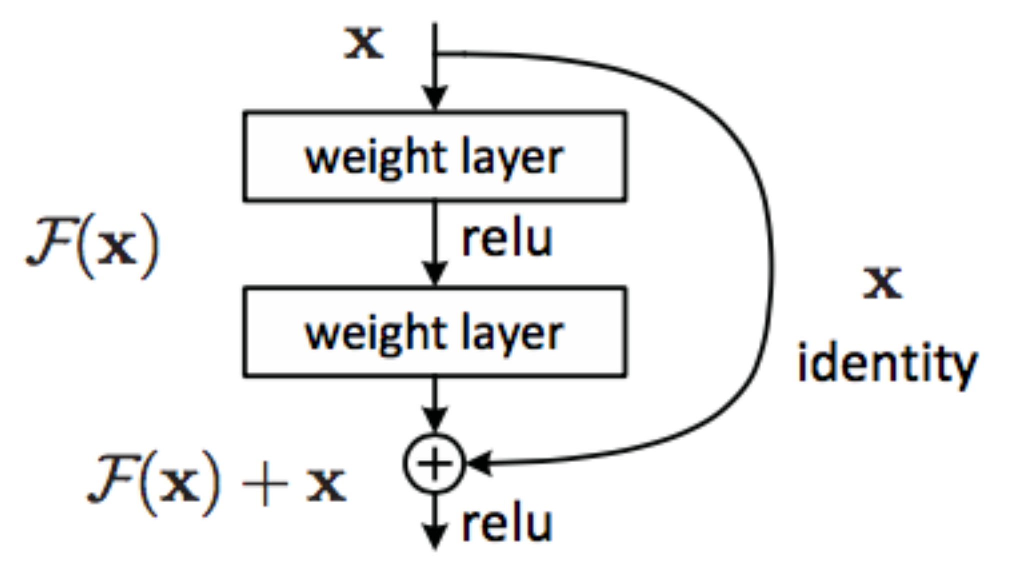


Figure 2. Residual learning: a building block.

StackExchange

The Self-Normalizing Network

- Allows for creation of much deeper feedforward networks
  - What batch norm is supposed to do, but stable
- Special activation function: SELU
- Requires custom initialization (lecun_normal in Keras)
- Also requires special version of dropout (Alpha Dropout)
- Alpha Dropout: randomly set input to −λα
  - Limit of selu(*x*) as *x* goes to –inf
- *p* = 0.05 or 0.1 are good defaults

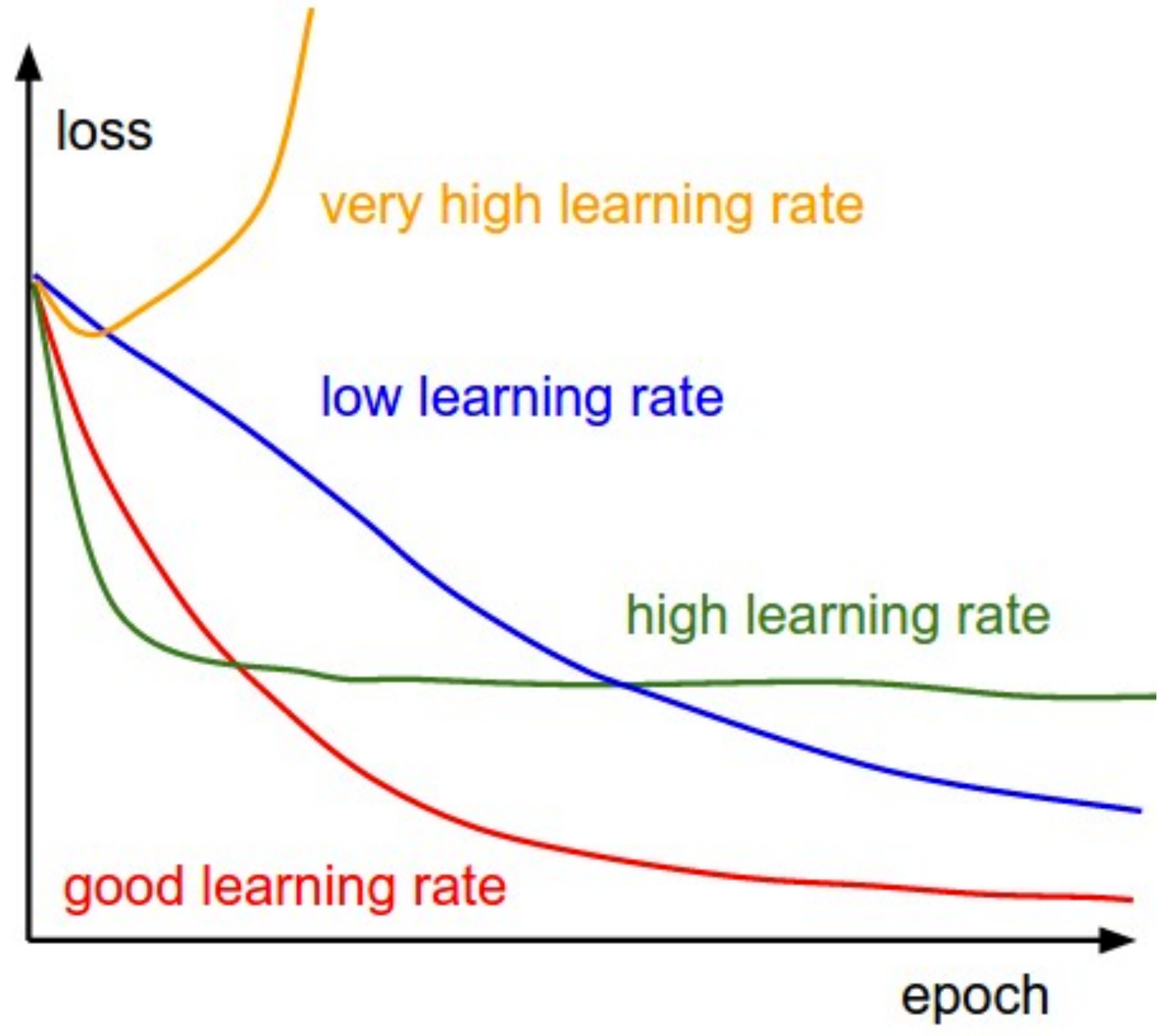
$$\mathrm{selu}(x) \;=\; \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^{x} - \alpha & \text{if } x \leqslant 0 \end{cases} .$$

Tuning the Network

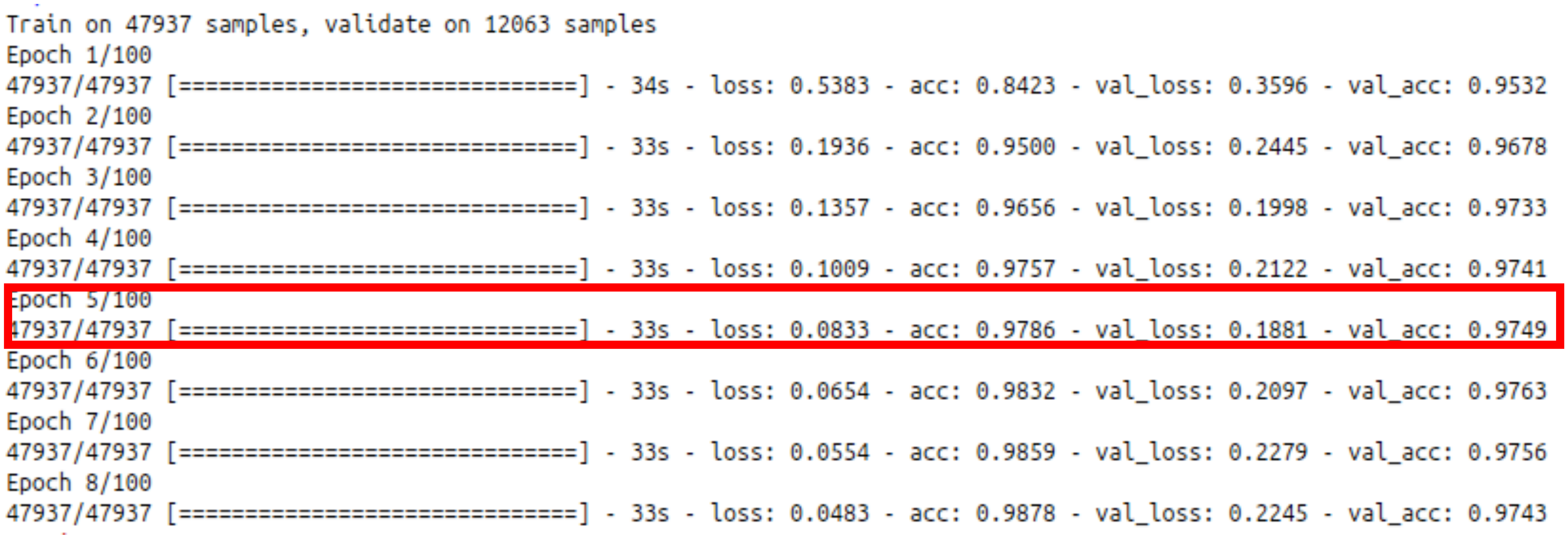
- Size of the layers (capacity): Start with slightly more parameters than you have training samples
- Start without any regularization (no batchnorm, no dropout)
- Tune learning rate (next slide)
- Increase size of network until you notice overfitting
- Add batchnorm, then increase capacity a bit more until you overfit again
- Add dropout (more effective than L1/L2 or elastic net due to efficiency and convergence considerations)
- If tuned well, training error should be better than validation error, but still comparable
  - Not enough regularization, you'll overfit
  - Too few parameters or too much regularization, you'll underfit

Tuning the Learning Rate

- By far, the most important hyperparameter to tune
- Start with 0.001, then double or halve as many times as needed
- Use Adam, RMSProp, or SGD with momentum (Nesterov momentum usually works even better)



Demo-ish



```
Train on 47937 samples, validate on 12063 samples
Epoch 1/100
47937/47937 [==============================] - 34s - loss: 0.5383 - acc: 0.8423 - val_loss: 0.3596 - val_acc: 0.9532
Epoch 2/100
47937/47937 [==============================] - 33s - loss: 0.1936 - acc: 0.9500 - val_loss: 0.2445 - val_acc: 0.9678
Epoch 3/100
47937/47937 [==============================] - 33s - loss: 0.1357 - acc: 0.9656 - val_loss: 0.1998 - val_acc: 0.9733
Epoch 4/100
47937/47937 [==============================] - 33s - loss: 0.1009 - acc: 0.9757 - val_loss: 0.2122 - val_acc: 0.9741
Epoch 5/100
47937/47937 [==============================] - 33s - loss: 0.0833 - acc: 0.9786 - val_loss: 0.1881 - val_acc: 0.9749
Epoch 6/100
47937/47937 [==============================] - 33s - loss: 0.0654 - acc: 0.9832 - val_loss: 0.2097 - val_acc: 0.9763
Epoch 7/100
47937/47937 [==============================] - 33s - loss: 0.0554 - acc: 0.9859 - val_loss: 0.2279 - val_acc: 0.9756
Epoch 8/100
47937/47937 [==============================] - 33s - loss: 0.0483 - acc: 0.9878 - val_loss: 0.2245 - val_acc: 0.9743
```

Thank you!

Questions?

@jvmancuso

jason@pandata.co

github.com/jvmancuso

Disadvantages of Deep Learning

Hard to train! Model architecture and tuning depends on dataset (dark art)

- Current research directions on evolving network architecture and tuning with hyperparameter optimization/reinforcement learning (all part of the AutoML research trend)

Long to train! Requires GPU to train efficiently and development can be complex

- Open-source libraries* abstract away implementation details and interface with GPUs (even in the cloud!)
- Distributed training algorithms exist and are an open area of research

Trade interpretability for performance

- Inner workings of the model are not interpretable to humans

* **PyTorch**, Keras/Tensorflow, Theano, Caffe, etc.

Broader Limitations

Sample efficiency! Easy tasks require very little data, but complex tasks require much more

- There is data everywhere; open research directions into one-/few-shot learning

Optimization! Often we cannot find the best network with current methods, but one that works "well enough"

- More sophisticated optimization routines, especially "learning to learn" research direction

Intelligence! Can deep learning really lead to superintelligence, or is this just pattern recognition?

- It'll probably be useful in the final solution, but we'll need a paradigm shift before that happens.
- See Karpathy's recent talk for more.