

04 05 2018

fusing design thinking with learning from data



Michael Kai Petersen Senior Scientist PhD mkpe@eriksholm.com



Article
McKinsey Quarterly
January 2018

Why digital strategies fail

By Jacques Bughin, Tanguy Catlin, Martin Hirt, and Paul Willmott



Most digital strategies don't reflect how digital is changing economic fundamentals, industry dynamics, or what it means to compete. Companies should watch out for five pitfalls.

The processing power of today's smartphones are several thousand times greater than that of the computers that landed a man on the moon in 1969. These devices connect the majority of the human population, and they're only ten years old.¹

MOST POPULAR

1. An executive's guide to AI
Interactive
2. Why digital strategies fail
Article - McKinsey Quarterly

How can this be, at a moment when virtually every company in the world is worried about its digital future? In other words, *why* are so many digital strategies failing? The answer has to do with the magnitude of the [disruptive economic force digital has become](#) and its incompatibility with traditional economic, strategic, and operating models. This article unpacks five issues that, in our experience, are particularly problematic. We hope they will awaken a sense of urgency and point toward how to do better. (For more on how companies are redefining their digital strategies, see “[Responding to digital threats](#).”)

Pitfall 1: Fuzzy definitions

When we talk with leaders about what they mean by digital, some view it as the upgraded term for what their IT function does. Others focus on digital marketing or sales. But very few have a broad, holistic view of what digital really means. We view digital as the nearly instant, free, and flawless ability to connect people, devices, and physical objects anywhere. By 2025, some 20 billion devices will be connected, nearly three times the world population. Over the past two years, such devices have churned out 90 percent of the data ever produced. Mining this data greatly enhances the power of analytics, which leads directly to dramatically higher levels of automation—both of processes and, ultimately, of decisions. All this gives birth to brand-new business models.² Think about the opportunities that telematics have created for the insurance industry. Connected cars collect real-time information about a customer’s driving behavior. The data allow insurers to price the risk associated with a driver automatically and more accurately, creating an opportunity to offer direct, pay-as-you-go coverage and bypassing today’s agents.

Lacking a clear definition of digital, companies struggle to connect digital strategy to their business, [leaving them adrift in the fast-churning waters of digital adoption and change](#). What’s happened with the smartphone over the past ten years should haunt you—and no industry will be immune.

Pitfall 2: Misunderstanding the economics of digital

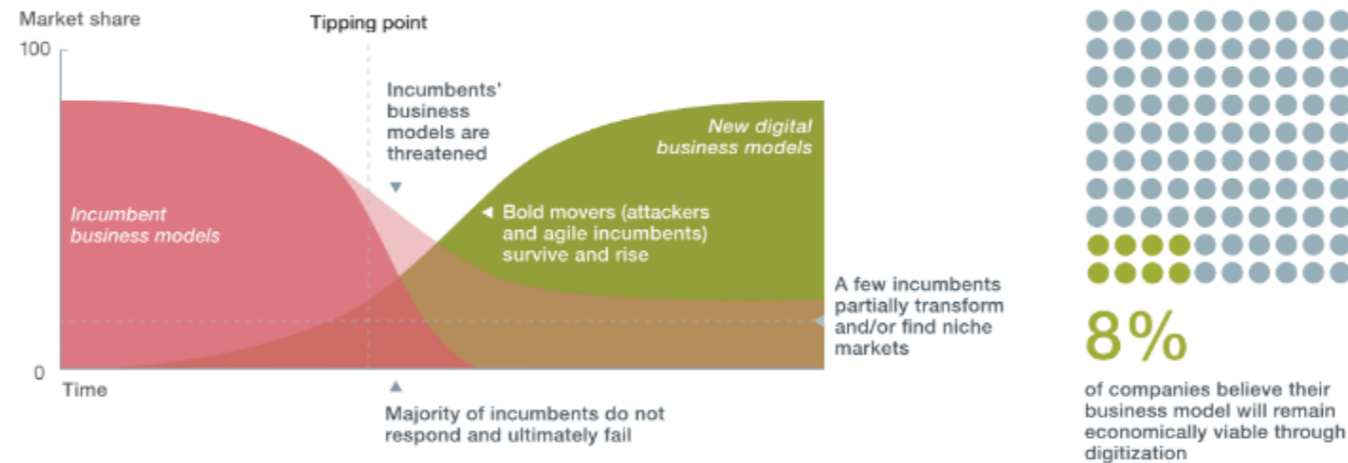
Many of us learned a set of core economic principles years ago and saw the power of their application early and often in our careers. (For more on the changing economics of digital competition, see the infographic below.) This built intuition—which often clashes with the new economic realities of digital competition. Consider these three:

Infographic

Don't underestimate how digital disrupts the nature of competition.

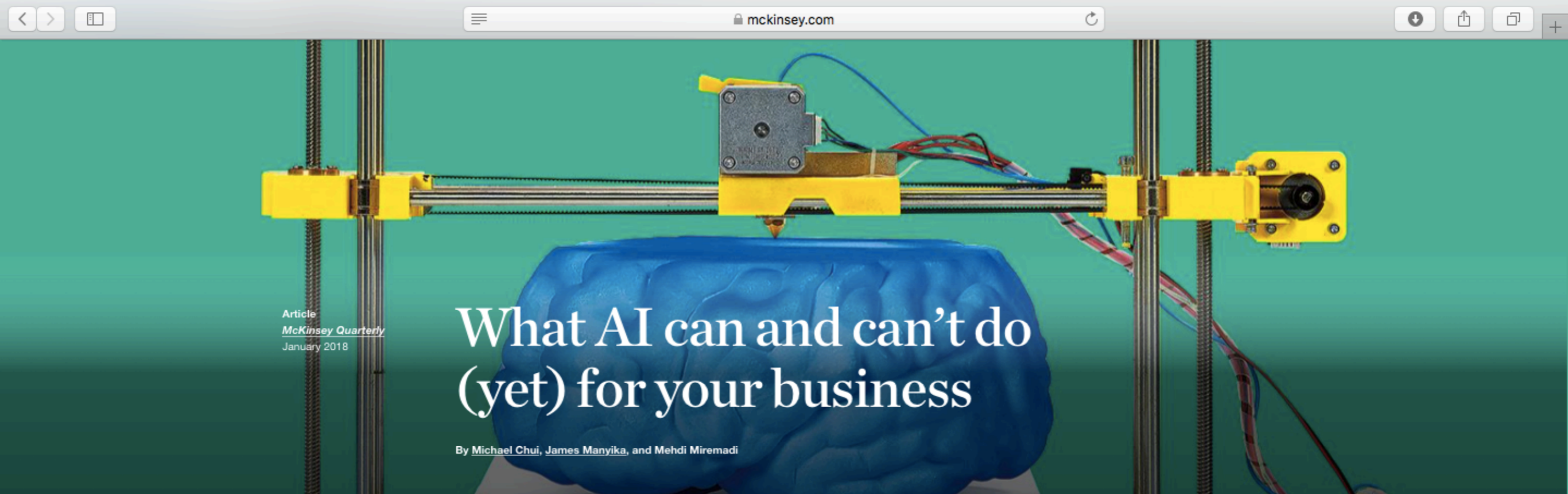


Disruption is always dangerous, but digital disruptions are happening faster than ever.



Source: McKinsey Digital Global Survey, 2016 and 2017; McKinsey analysis

McKinsey&Company



Article
McKinsey Quarterly
January 2018

What AI can and can't do (yet) for your business

By Michael Chui, James Manyika, and Mehdi Miremadi



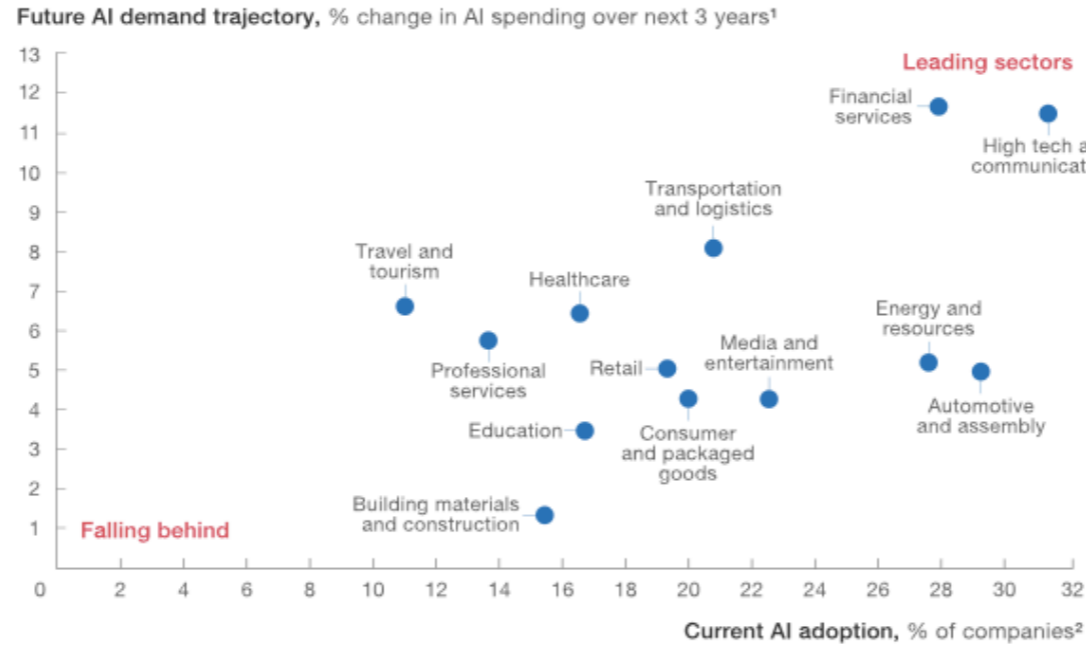
Artificial intelligence is a moving target. Here's how to take better aim.

Artificial intelligence (AI) seems to be everywhere. We experience it at home and on our phones. Before we know it—if entrepreneurs and business innovators are to be believed—AI will be in just about every product and service we buy and use. In addition, its [application to business problem solving](#) is growing in leaps and bounds. And at the same time, concerns about AI's implications are rising: we worry about the impact of AI-enabled automation on the workplace, employment, and society.

A reality sometimes lost amid both the fears and the headline triumphs, such as Alexa, Siri, and AlphaGo, is that the AI technologies themselves—namely, machine learning and its subset, deep learning—have plenty of limitations that will still require considerable effort to overcome. This is an article about those limitations, aimed at helping executives better understand what may be

MOST POPULAR

1. [Leading with inner agility](#)
Article - McKinsey Quarterly
2. [An executive's guide to AI](#)
Interactive
3. [Microsoft's next act](#)
Podcast - McKinsey Quarterly
4. [The fairness factor in performance management](#)



¹Estimated average, weighted by company size; demand trajectory based on midpoint of range selected by survey respondent.

²Adopting 1 or more AI technologies at scale or in business core; weighted by company size.

Source: McKinsey Global Institute AI adoption and use survey; McKinsey Global Institute analysis

McKinsey&Company

Executives hoping to narrow the gap must be able to address AI in an informed way. In other words, they need to understand not just where AI can boost innovation, insight, and decision making; lead to revenue growth; and capture of efficiencies—but also where AI *can't* yet provide value. What's more, they must appreciate the relationship and distinctions between technical constraints and organizational ones, such as cultural barriers; a dearth of personnel capable of building business-ready, AI-powered applications; and the “last mile” challenge of embedding AI in products and processes. If you want to become a leader who understands some of the critical technical challenges slowing AI's advance and is prepared to exploit promising developments that could overcome those limitations and potentially bend the trajectory of AI—read on.

Limitation 1: Data labeling

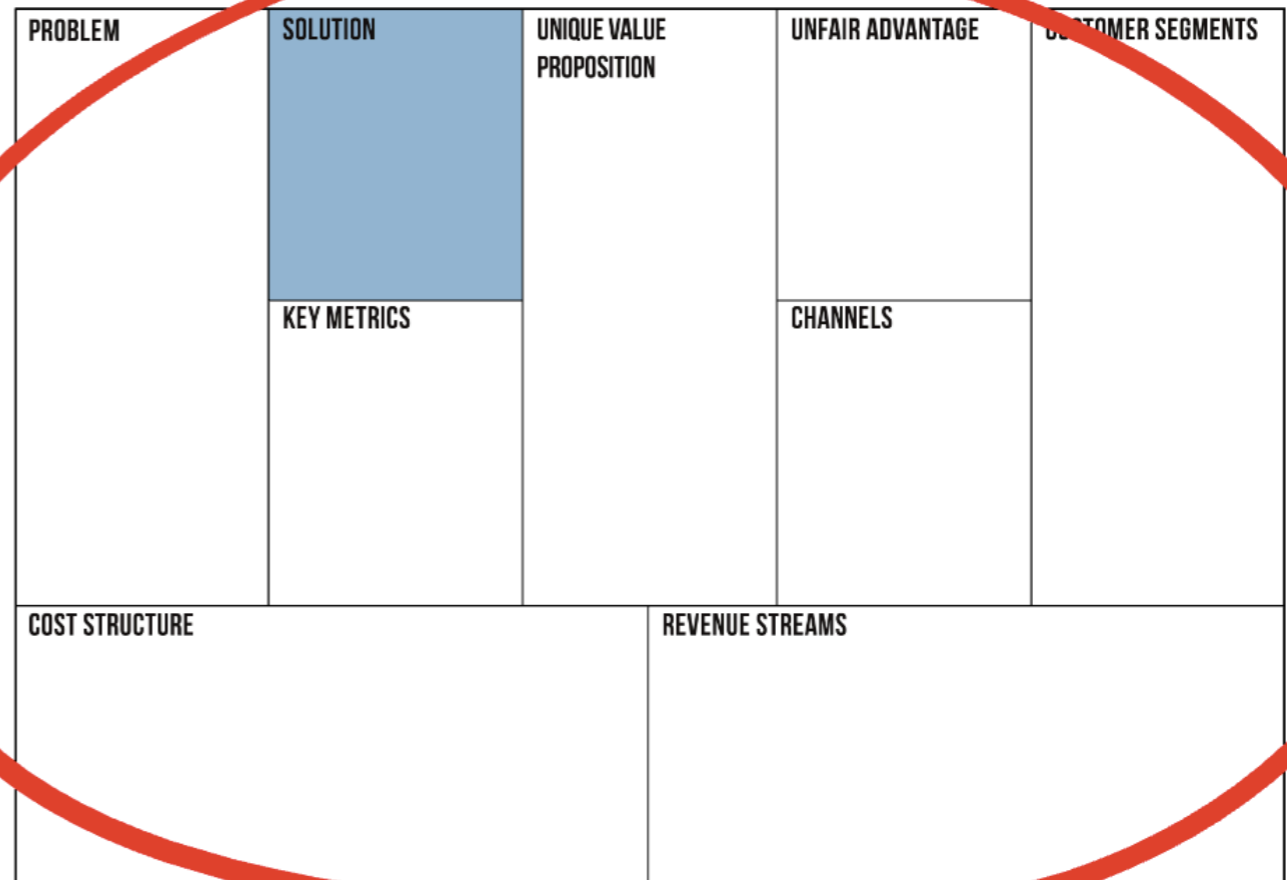
Most current AI models are trained through “supervised learning.” This means that humans must label and categorize the underlying data, which can be a sizable and error-prone chore. For example, companies developing self-driving-car technologies are hiring hundreds of people to manually annotate hours of video feeds from prototype vehicles to help train these systems. At the same time, promising new techniques are emerging, such as in-stream supervision (demonstrated by Eric Horvitz and his colleagues at Microsoft Research), in which data can be labeled in the course of natural usage.² Unsupervised or semisupervised approaches reduce the need for large, labeled data sets. Two promising techniques are reinforcement learning and generative adversarial networks.

Reinforcement learning. This unsupervised technique allows algorithms to learn tasks simply by trial and error. The methodology hearkens to a “carrot and stick” approach: for every attempt an algorithm makes at performing a task, it receives a “reward” (such as a higher score) if the behavior is successful or a “punishment” if it isn’t. With repetition, performance improves, in many cases surpassing human capabilities—so long as the learning environment is representative of the real world.

Reinforcement learning has famously been used in training computers to play games—most recently, in conjunction with deep-learning techniques. In May 2017, for example, it helped the AI system AlphaGo to defeat world champion Ke Jie in the game of Go. In another example, Microsoft has fielded decision services that draw on reinforcement learning and adapt to user preferences. The potential application of reinforcement learning cuts across many business arenas. Possibilities include an AI-driven trading portfolio that acquires or loses points for gains or losses in value, respectively; a product-recommendation engine that receives points for every recommendation-driven sale; and truck-routing software that receives a reward for on-time deliveries or reducing fuel consumption.

Reinforcement learning can also help AI transcend the natural and social limitations of human labeling by developing previously unimagined solutions and

WDH course module one
build a common vocabulary
to enable learning from data
highlight WDH potential
based on Harvard Business
Review and McKinsey cases



Your "business model" is the product

Lean Canvas is adapted from The Business Model Canvas (<http://www.businessmodelgeneration.com>) and is licensed under the Creative Commons Attribution-Share Alike 3.0 Un-ported License.

O'REILLY



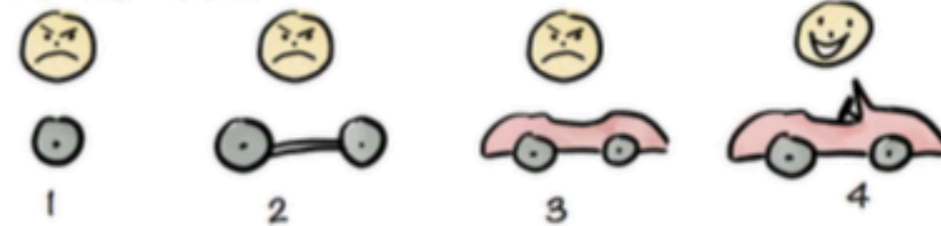
User Story Mapping

DISCOVER THE WHOLE STORY,
BUILD THE RIGHT PRODUCT

Jeff Patton
with Peter Economy
Forewords by Martin Fowler,
Alan Cooper, and Marty Cagan

learned then if his big guess was right. You'll need to trust me on this, but it wouldn't have been—because it rarely is.

Not like this....



This is a simple visualization made by my friend Henrik Kniberg. It beautifully illustrates a broken release strategy where at every release I get something I can't use, until the last release when I get something I can.

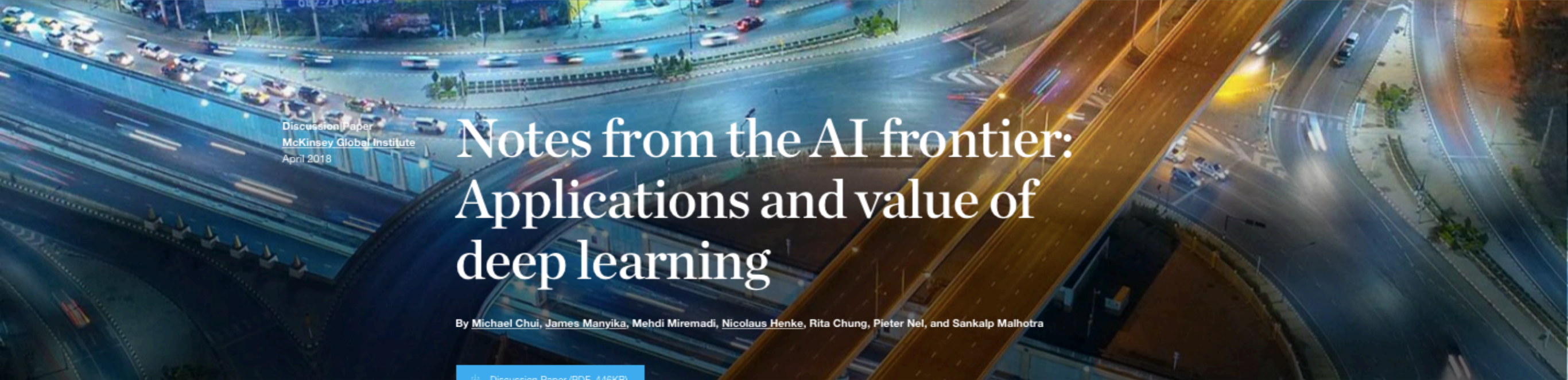
Henrik suggests this alternative strategy:

Like this!



If I plan my releases this way, in each release I deliver something people can actually use. Now, in this silly transportation example, if my goal is to travel a long distance and carry some stuff with me, and you gave me a skateboard, I might feel a bit frustrated. I'd let you know how difficult it was to travel long distances with that thing—although it was fun to goof around with it in the driveway. If your goal was to leave me delighted, you might feel bad about that. But your real goal was to learn, which you did. So that's good. You learned I wanted to travel farther, and if you picked up on it, you also learned I valued having fun.

In Henrik's progression, things start picking up at around the bicycle release because I can actually use it as adequate transportation. And, at about motorcycle level, I can *really* see this working for me—and I'm having fun too. That could be minimum and viable for me. If I



Discussion Paper
McKinsey Global Institute
April 2018

Notes from the AI frontier: Applications and value of deep learning

By Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra

[Discussion Paper \(PDF-446KB\)](#)



An analysis of more than 400 use cases across 19 industries and nine business functions highlights the broad use and significant economic potential of advanced AI techniques.

Artificial intelligence (AI) stands out as a transformational technology of our digital age—and its practical application throughout the economy is growing apace. For this briefing, *Notes from the AI frontier: Insights from hundreds of use cases* (PDF-446KB), we mapped both traditional analytics and newer “deep learning” techniques and the problems they can solve to more than 400 specific use cases in companies and organizations. Drawing on McKinsey Global Institute research and the applied experience with AI of McKinsey Analytics, we assess both the practical applications and the economic potential of advanced AI techniques across industries and business functions. Our findings highlight the substantial potential of applying deep learning techniques to use cases across the economy, but we also see some continuing limitations and obstacles—along with future opportunities as the technologies continue their advance. Ultimately, the value of AI is not to be found in the models themselves, but in companies’ abilities to harness them.

MOST POPULAR

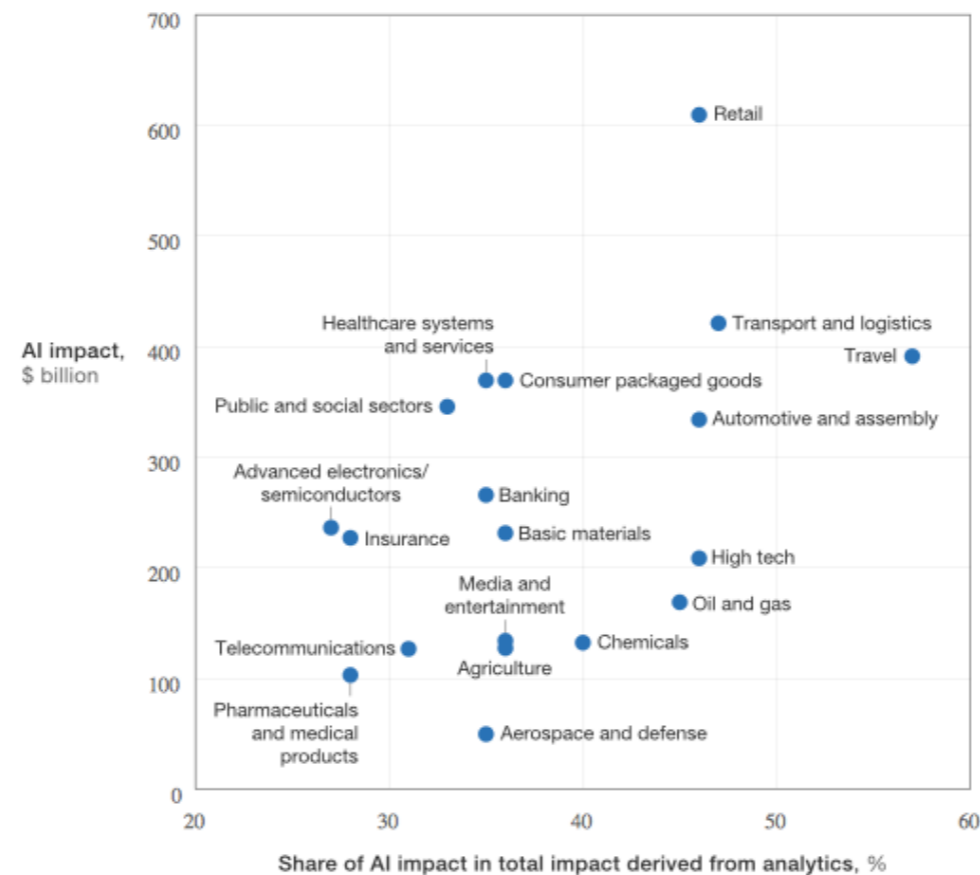
- [1. Leading with inner agility](#)
Article - McKinsey Quarterly
- [2. An executive’s guide to AI](#)
Interactive
- [3. Microsoft’s next act](#)
Podcast - McKinsey Quarterly
- [4. The fairness factor in performance management](#)
Article - McKinsey Quarterly

Sizing the potential value of AI

We estimate that the AI techniques we cite in this briefing together have the potential to create between \$3.5 trillion and \$5.8 trillion in value annually across nine business functions in 19 industries. This constitutes about 40 percent of the overall \$9.5 trillion to \$15.4 trillion annual impact that could potentially be enabled by all analytical techniques (Exhibit 4).

Exhibit 4

Artificial intelligence (AI) has the potential to create value across sectors.



McKinsey&Company | Source: McKinsey Global Institute analysis

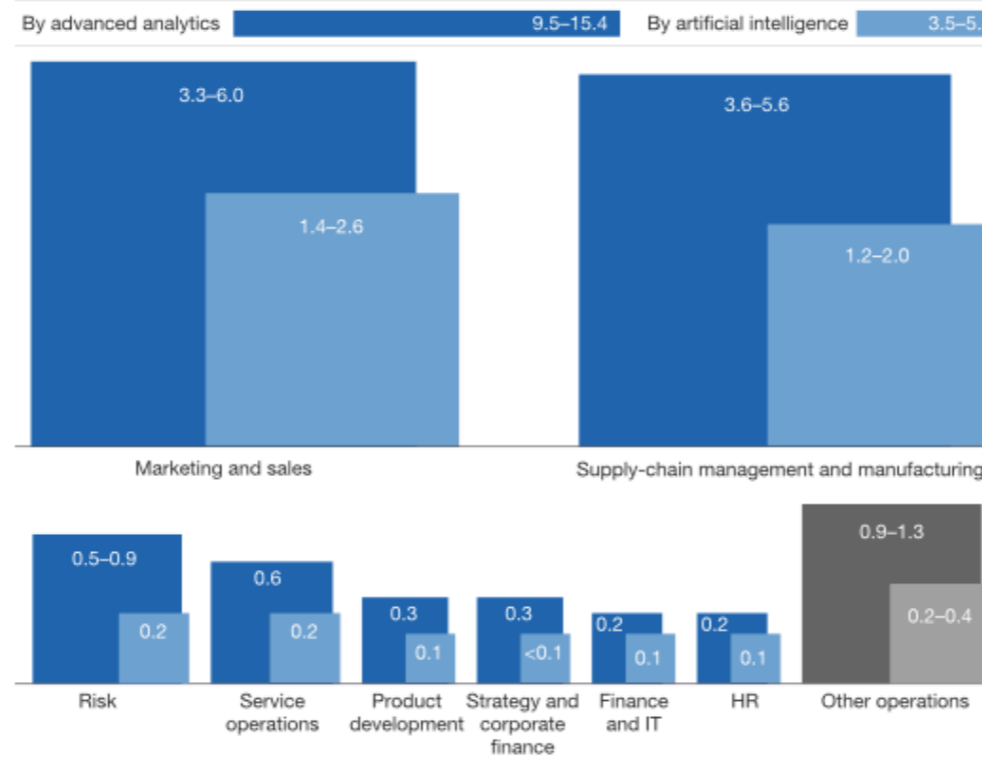
deep learning adds value
by 30 to 128 % above
traditional data analytics
requires labeled data sets
5000 samples per category
10M to outperform humans ?

Consumer industries such as retail and high tech will tend to see more potential from marketing and sales AI applications because frequent and digital interactions between business and customers generate larger data sets for AI techniques to tap into. E-commerce platforms, in particular, stand to benefit. This is because of the ease with which these platforms collect customer information such as click data or time spent on a web page and can then customize promotions, prices, and products for each customer dynamically and in real time.

Exhibit 5

Artificial intelligence's impact is likely to be most substantial in marketing and sales as well as supply-chain management and manufacturing, based on our use cases.

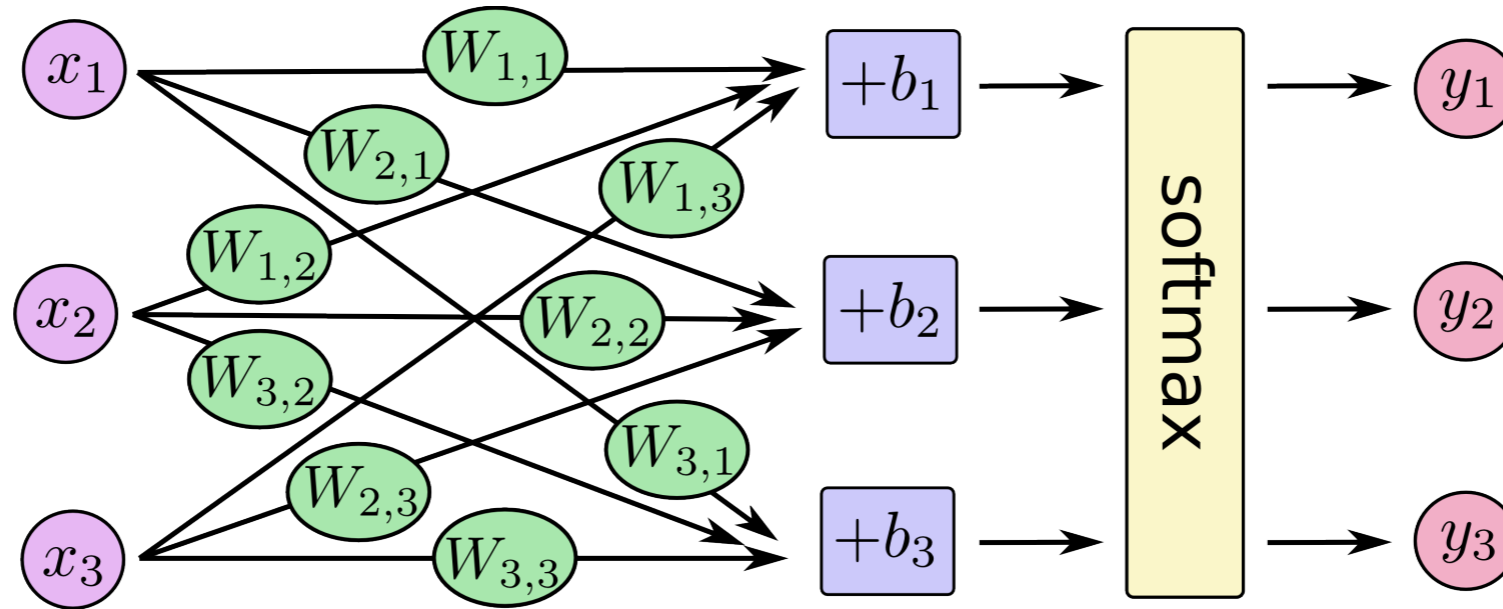
Value unlocked, \$ trillion



Note: Figures may not sum to 100%, because of rounding.

software 2.0 paradigm shift
instead of writing lines of
code defining behaviors
neural networks learn
behaviors by recognizing
patterns in data

0	4	1	9	2	1	3	1	4	3
5	3	6	1	7	2	8	6	9	4
0	9	1	1	2	4	3	2	7	3
8	6	9	0	5	6	0	7	6	1
8	7	9	3	9	8	5	9	3	3
0	7	4	9	8	0	9	4	1	4
4	6	0	4	5	6	1	0	0	1
7	1	6	3	0	2	1	1	7	9
0	2	6	7	8	3	9	0	4	6
7	4	6	8	0	7	8	3	1	5



$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$y = \text{softmax}(\text{evidence})$$

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

[Getting Started](#)[Getting Started With TensorFlow](#)[MNIST For ML Beginners](#)[Deep MNIST for Experts](#)[TensorFlow Mechanics 101](#)[tf.contrib.learn Quickstart](#)[Building Input Functions with tf.contrib.learn](#)[Logging and Monitoring Basics with tf.contrib.learn](#)[TensorBoard: Visualizing Learning](#)[TensorBoard: Embedding Visualization](#)[TensorBoard: Graph Visualization](#)

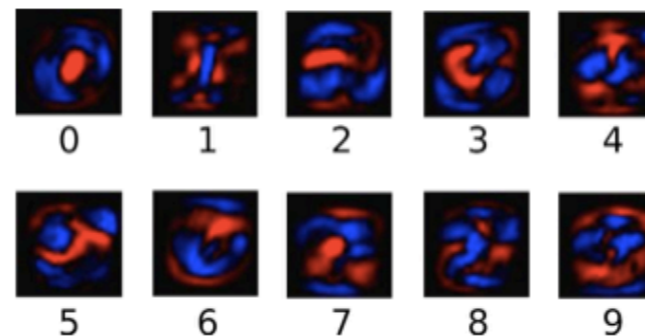
We know that every image in MNIST is of a handwritten digit between zero and nine. So there are only ten possible things that a given image can be. We want to be able to look at an image and give the probabilities for it being each digit. For example, our model might look at a picture of a nine and be 80% sure it's a nine, but give a 5% chance to it being an eight (because of the top loop) and a bit of probability to all the others because it isn't 100% sure.

This is a classic case where a softmax regression is a natural, simple model. If you want to assign probabilities to an object being one of several different things, softmax is the thing to do, because softmax gives us a list of values between 0 and 1 that add up to 1. Even later on, when we train more sophisticated models, the final step will be a layer of softmax.

A softmax regression has two steps: first we add up the evidence of our input being in certain classes, and then we convert that evidence into probabilities.

To tally up the evidence that a given image is in a particular class, we do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.

The following diagram shows the weights one model learned for each of these classes. Red represents negative weights, while blue represents positive weights.



We also add some extra evidence called a bias. Basically, we want to be able to say that some things are more likely independent of the input. The result is that the evidence for a class i given an input x is:

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

```
In [1]: # import images and labels for training a softmax regression model to recognize MNIST digits;
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

/anaconda3/envs/tensorflow/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compiletime version 3.5 of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.6
  return f(*args, **kwargs)

Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
In [2]: # create a softmax regression model by adding up evidence of pixel intensities and turning output into probabilities
# x image input y_labels output, W learned positive/negative weights for class, b class bias independent of input, non
import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

```
In [3]: # train the model and define loss function to minimize cross entropy based on gradient descent
# y_ predicted probability, y true distribution, loading 100 examples for each iteration
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

```
In [4]: # evaluate the accuracy of the model
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))

0.9186
```

In []:

WDH course module two

build a convolutional network on a laptop which learns from data
showcase WDH prototypes
related to business intelligence algorithms and rethinking R&D

Tutorials

Using GPUs

Image Recognition

How to Retrain Inception's Final Layer for New Categories

[A Guide to TF Layers: Building a Convolutional Neural Network](#)

Convolutional Neural Networks

Vector Representations of Words

Recurrent Neural Networks

Sequence-to-Sequence Models

Large-scale Linear Models with TensorFlow

TensorFlow Linear Model Tutorial

TensorFlow Wide & Deep Learning Tutorial

Mandelbrot Set

Partial Differential Equations

Intro to Convolutional Neural Networks

Convolutional neural networks (CNNs) are the current state-of-the-art model architecture for image classification tasks. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification. CNNs contains three components:

- **Convolutional layers**, which apply a specified number of convolution filters to the image. For each subregion, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a [ReLU activation function](#) to the output to introduce nonlinearities into the model.
- **Pooling layers**, which [downsample the image data](#) extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts subregions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values.
- **Dense (fully connected) layers**, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

Typically, a CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification. The final dense layer in a CNN contains a single node for each target class in the model (all the possible classes the model may predict), with a [softmax](#) activation function to generate a value between 0–1 for each node (the sum of all these softmax values is equal to 1). We can interpret the softmax values for a given image as relative measurements of how likely it is that the image falls into each target class.

★ **Note:** For a more comprehensive walkthrough of CNN architecture, see Stanford University's [Convolutional Neural Networks for Visual Recognition course materials](#).

Building the CNN MNIST Classifier

Let's build a model to classify the images in the MNIST dataset using the following CNN architecture:

1. **Convolutional Layer #1:** Applies 32 5x5 filters (extracting 5x5-pixel subregions), with ReLU activation function
2. **Pooling Layer #1:** Performs max pooling with a 2x2 filter and stride of 2 (which specifies that pooled regions do not overlap)
3. **Convolutional Layer #2:** Applies 64 5x5 filters, with ReLU activation function
4. **Pooling Layer #2:** Again, performs max pooling with a 2x2 filter and stride of 2
5. **Dense Layer #1:** 1,024 neurons, with dropout regularization rate of 0.4 (probability of 0.4 that any given element will be dropped during training)
6. **Dense Layer #2 (Logits Layer):** 10 neurons, one for each digit target class (0–9).

Contents

Getting Started

[Intro to Convolutional Neural Networks](#)

Building the CNN MNIST Classifier

Input Layer

Convolutional Layer #1

Pooling Layer #1

Convolutional Layer #2 and Pooling Layer #2

Dense Layer

Logits Layer

Calculate Loss

Configure the Training Op

Generate Predictions

Training and Evaluating the CNN MNIST Classifier

Load Training and Test Data

Create the Estimator

Set Up a Logging Hook

Train the Model

Evaluate the Model

Run the Model

Additional Resources



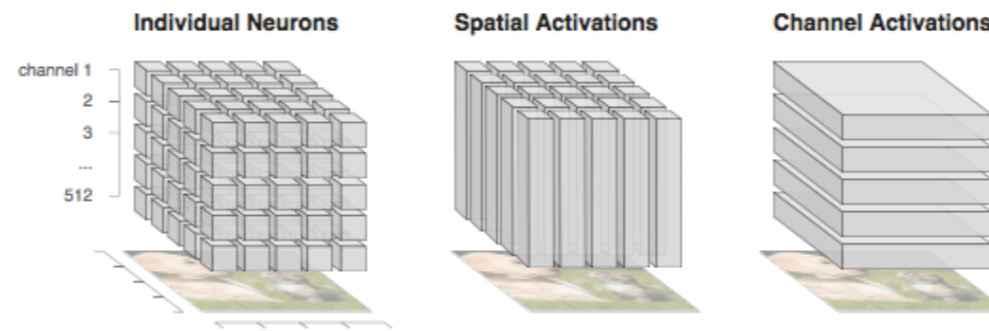
CHOOSE AN INPUT IMAGE



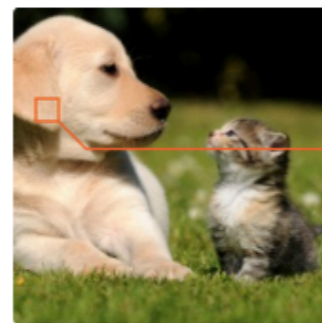
Making Sense of Hidden Layers

Much of the recent work on interpretability is concerned with a neural network's input and output layers. Arguably, this focus is due to the clear meaning these layers have: in computer vision, the input layer represents values for the red, green, and blue color channels for every pixel in the input image, while the output layer consists of class labels and their associated probabilities.

However, the power of neural networks lies in their hidden layers — at every layer, the network discovers a new representation of the input. In computer vision, we use neural networks that run the same feature detectors at every position in the image. We can think of each layer's learned representation as a three-dimensional cube. Each cell in the cube is an *activation*, or the amount a neuron fires. The x- and y-axes correspond to positions in the image, and the z-axis is the channel (or detector) being run.



The cube of activations that a neural network for computer vision develops at each hidden layer. Different slices of the cube allow us to target the activations of individual neurons, spatial positions, or channels.



Making sense of these activations is hard because we usually work with them as abstract vectors:

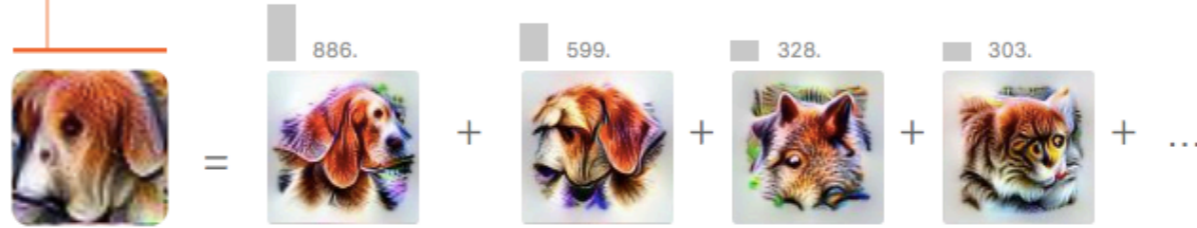
$$a_{4,1} = [0, 0, 0, 25.2, 164.1, 0, 42.7, 4.51, 115.0, 51.3, \dots]$$

With feature visualization, however, we can transform this abstract vector into a more meaningful "semantic dictionary".

CHOOSE AN INPUT IMAGE



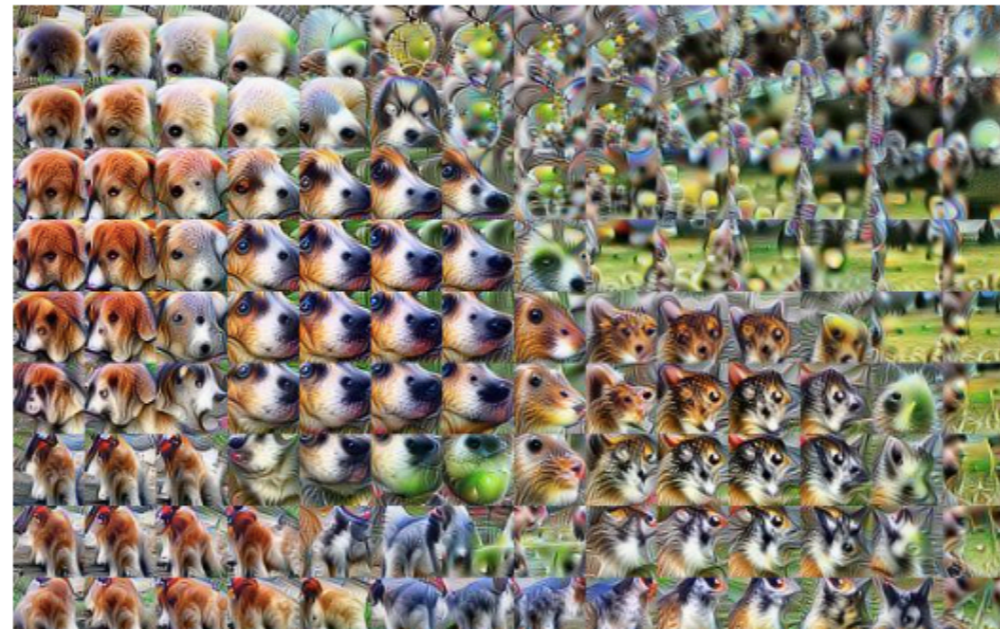
Semantic dictionaries give us a fine-grained look at an activation: what does each single neuron detect? Building off this representation, we can also consider an activation vector as a whole. Instead of visualizing individual neurons, we can instead visualize the *combination* of neurons that fire at a given spatial location. (Concretely, we optimize the image to maximize the dot product of its activations with the original activation vector.)



Activation Vector

Channels

Applying this technique to all the activation vectors allows us to not only see what the network detects at each position, but also what the network understands of the input image as a whole.



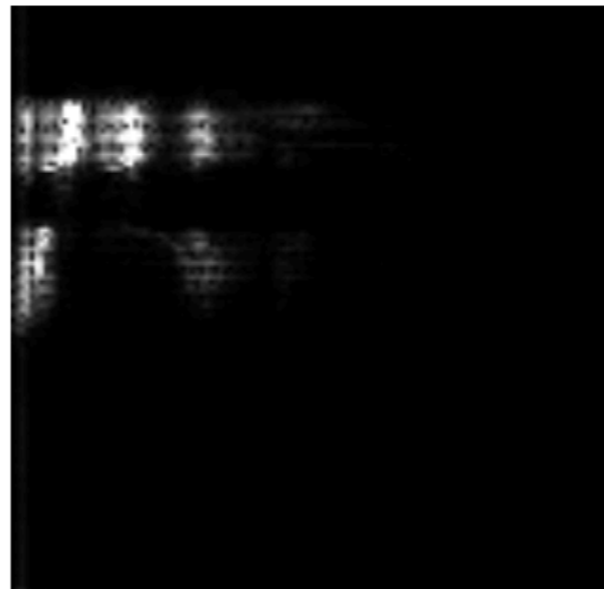
- Tutorials
 - Images
 - MNIST
 - Image Recognition
 - Image Retraining
 - Convolutional Neural Networks
 - Sequences
 - Text Classification
 - Recurrent Neural Networks
 - Neural Machine Translation
 - Drawing Classification
 - [Simple Audio Recognition](#)
 - Data Representation
 - Linear Models
 - Wide & Deep Learning
 - Vector Representations of Words
 - Kernel Methods
 - Non-ML
 - Mandelbrot Set
 - Partial Differential Equations

The architecture used in this tutorial is based on some described in the paper [Convolutional Neural Networks for Small-footprint Keyword Spotting](#). It was chosen because it's comparatively simple, quick to train, and easy to understand, rather than being state of the art. There are lots of different approaches to building neural network models to work with audio, including [recurrent networks](#) or [dilated \(atrous\) convolutions](#). This tutorial is based on the kind of convolutional network that will feel very familiar to anyone who's worked with image recognition. That may seem surprising at first though, since audio is inherently a one-dimensional continuous signal across time, not a 2D spatial problem.

We solve that issue by defining a window of time we believe our spoken words should fit into, and converting the audio signal in that window into an image. This is done by grouping the incoming audio samples into short segments, just a few milliseconds long, and calculating the strength of the frequencies across a set of bands. Each set of frequency strengths from a segment is treated as a vector of numbers, and those vectors are arranged in time order to form a two-dimensional array. This array of values can then be treated like a single-channel image, and is known as a [spectrogram](#). If you want to view what kind of image an audio sample produces, you can run the `wav_to_spectrogram` tool:

```
bazel run tensorflow/examples/wav_to_spectrogram:wav_to_spectrogram -- \
--input_wav=/tmp/speech_dataset/happy/ab00c4b2_nohash_0.wav \
--output_png=/tmp/spectrogram.png
```

If you open up `/tmp/spectrogram.png` you should see something like this:



Because of TensorFlow's memory order, time in this image is increasing from top to bottom, with frequencies going from left to right, unlike the usual convention for spectrograms where time is left to right. You should be able to see a couple of distinct parts, with the first syllable "Ha" distinct from "ppy".

Because the human ear is more sensitive to some frequencies than others, it's been traditional in speech recognition to do further processing to this representation to turn it into a set of [Mel-Frequency Cepstral Coefficients](#), or MFCCs for

- Contents
- Preparation
- Training
- Confusion Matrix
- Validation
- Tensorboard
- Training Finished
- Running the Model in an Android App
- [How does this Model Work?](#)
- Streaming Accuracy
- RecognizeCommands
- Advanced Training
 - Custom Training Data
 - Unknown Class
 - Background Noise
 - Silence
 - Time Shifting
- Customizing the Model

Tutorials

Using GPUs

Image Recognition

How to Retrain Inception's Final Layer for New Categories

A Guide to TF Layers: Building a Convolutional Neural Network

Convolutional Neural Networks

Vector Representations of Words

Recurrent Neural Networks

Sequence-to-Sequence Models

Large-scale Linear Models with TensorFlow

TensorFlow Linear Model Tutorial

TensorFlow Wide & Deep Learning Tutorial

Mandelbrot Set

Partial Differential Equations

We can visualize the learned vectors by projecting them down to 2 dimensions using for instance something like the [t-SNE dimensionality reduction technique](#). When we inspect these visualizations it becomes apparent that the vectors capture some general, and in fact quite useful, semantic information about words and their relationships to one another. It was very interesting when we first discovered that certain directions in the induced vector space specialize towards certain semantic relationships, e.g. *male-female*, *verb tense* and even *country-capital* relationships between words, as illustrated in the figure below (see also for example [Mikolov et al., 2013](#)).



This explains why these vectors are also useful as features for many canonical NLP prediction tasks, such as part-of-speech tagging or named entity recognition (see for example the original work by [Collobert et al., 2011 \(pdf\)](#), or follow-up work by [Turian et al., 2010](#)).

But for now, let's just use them to draw pretty pictures!

Building the Graph

This is all about embeddings, so let's define our embedding matrix. This is just a big random matrix to start. We'll initialize the values to be uniform in the unit cube.

```
embeddings = tf.Variable(
    tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0))
```

The noise-contrastive estimation loss is defined in terms of a logistic regression model. For this, we need to define the weights and biases for each word in the vocabulary (also called the `output weights` as opposed to the `input embeddings`). So let's define that.

```
nce_weights = tf.Variable(
    tf.truncated_normal([vocabulary_size, embedding_size],
                       stddev=1.0 / math.sqrt(embedding_size)))
nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
```

Contents

Highlights

Motivation: Why Learn Word Embeddings?

Scaling up with Noise-Contrastive Training

The Skip-gram Model

Building the Graph

Training the Model

Visualizing the Learned Embeddings

Evaluating Embeddings: Analogical Reasoning

Optimizing the Implementation

Conclusion



Tutorials

Using GPUs

Image Recognition

How to Retrain Inception's Final Layer for New Categories

A Guide to TF Layers: Building a Convolutional Neural Network

Convolutional Neural Networks

Vector Representations of Words

Recurrent Neural Networks

[Sequence-to-Sequence Models](#)

Large-scale Linear Models with TensorFlow

TensorFlow Linear Model Tutorial

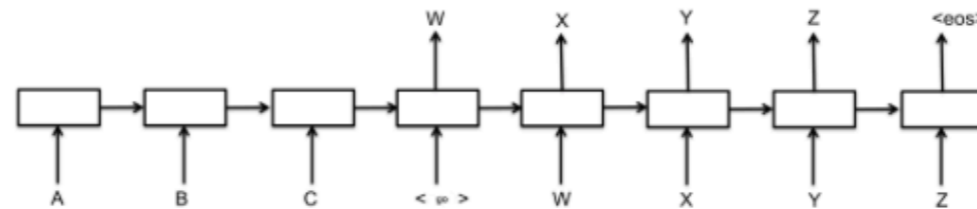
TensorFlow Wide & Deep Learning Tutorial

Mandelbrot Set

Partial Differential Equations

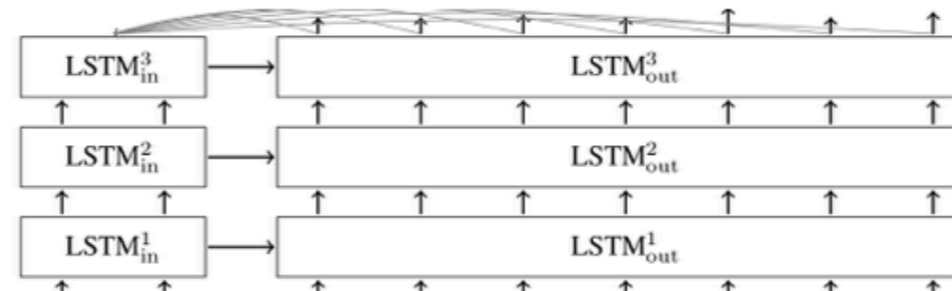
Sequence-to-sequence basics

A basic sequence-to-sequence model, as introduced in [Cho et al., 2014 \(pdf\)](#), consists of two recurrent neural networks (RNNs): an *encoder* that processes the input and a *decoder* that generates the output. This basic architecture is depicted below.



Each box in the picture above represents a cell of the RNN, most commonly a GRU cell or an LSTM cell (see the [RNN Tutorial](#) for an explanation of those). Encoder and decoder can share weights or, as is more common, use a different set of parameters. Multi-layer cells have been successfully used in sequence-to-sequence models too, e.g. for translation [Sutskever et al., 2014 \(pdf\)](#).

In the basic model depicted above, every input has to be encoded into a fixed-size state vector, as that is the only thing passed to the decoder. To allow the decoder more direct access to the input, an *attention* mechanism was introduced in [Bahdanau et al., 2014 \(pdf\)](#). We will not go into the details of the attention mechanism (see the paper); suffice it to say that it allows the decoder to peek into the input at every decoding step. A multi-layer sequence-to-sequence network with LSTM cells and attention mechanism in the decoder looks like this.



TensorFlow seq2seq library

As you can see above, there are many different sequence-to-sequence models. Each of these models can use different RNN cells, but all of them accept encoder inputs and decoder inputs. This motivates the interfaces in the TensorFlow seq2seq library (`tensorflow/tensorflow/python/ops/seq2seq.py`). The basic RNN encoder-decoder sequence-to-sequence model works as follows.

```
outputs, states = basic_rnn_seq2seq(encoder_inputs, decoder_inputs, cell)
```

Contents

[Sequence-to-sequence basics](#)

TensorFlow seq2seq library

Neural translation model

Sampled softmax and output projection

Bucketing and padding

Let's run it

What next?





Explore Professionals



Oticon Opn™

Delivers a natural, OpenSound experience





Applets


Services


New Applet


Filter Applets


 Sync your Amazon Alexa to-dos with your reminders


On works with 


 If maker Event "High", then set a program


On works with 


 If maker Event "Medium", then set a program

On works with 


 If maker Event "Low", then set a program


On works with 

 If maker Event "Pinna", then set a program

On works with 

 31 Read out any upcoming events in my google calendar using Oticon OPNs text to speech

On works with 

 ON If battery is low, then send a notification

 Log when enter/exit Oticon

 Log when enter/exit DTU skylab

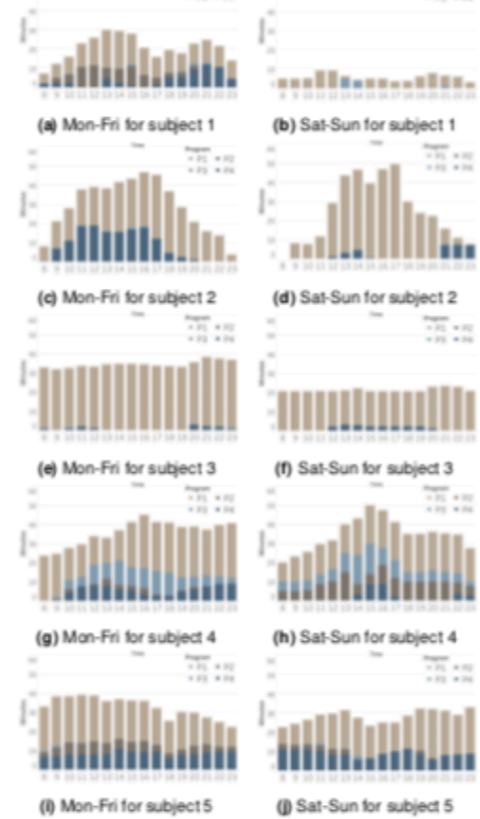


Figure 2: Program usage over time, from 8AM to 12AM. P1 is beige, P2 is brown, P3 is light blue and P4 is dark blue. The left hand columns represents usage over weekdays, and the right ones represents usages in weekends.

Volume and program interactions

An additional parameter to investigate when modeling the behavioral patterns are the volume change interactions. The volume interaction can be interpreted as a fine tuning of the desired auditory scene, by increasing or decreasing the intensity, thus zooming in or out of an auditory scene. In Figure 3 a comparison of the 5 test subjects and their usage of volume with respect to program can be observed. The light to dark blue colors reflect decreasing volume, while the yellow to orange gradients reflect an increase in gain. It can be observed that most subjects decrease the volume in P1 during the weekend. Subject 4 prefers to primarily reduce the volume, in contrast with Subject 5 which prefers to mostly increase the volume. In these cases we hypothesize that the gain settings of the devices might need to be adjusted. Subject 1 adjusts the volume both up and down from Monday through Friday, whereas the volume is only decreased during weekends.

While the above user interaction over a 10 week period can be inferred directly from the program change and volume adjustment, we subsequently in follow-up audiological sessions with the subjects found that the behavioral patterns were aligned with the aggregated program usage history data continuously collected over 4 months by the devices. Subsequently we interviewed the test subjects to determine what defined their program and volume preferences. The P1 program was preferred in most listening scenarios because it allows the users to selectively shift their attention omnidirectionally to any sound sources. However, when encountering more challenging acoustical environments, the three alternative program settings were selected, whether the aim was to enhance speech intelligibility, attenuate ambient sounds or remove background noise. Additionally



users increased or reduced the perceived loudness of these settings by continuously adjusting the volume.

Perspectives

These results indicate that the users predominantly preferred to combine volume adjustments with settings providing an open frontal focus coupled with a natural attenuation of ambient sounds in 74% of the usage time. This differ from earlier studies reporting that an omnidirectional focus was only chosen in respectively 37% [11] and 33% [1] of listening scenarios. In contrast to earlier studies using simulated sound environments [2] our findings are based on the actual acoustic environments encountered by users over several weeks of usage. It is difficult to compare these studies, as the data generated in our study represent snapshots of user intents triggered by the changing auditory context throughout daily life. When compared to earlier studies, the quality of sound enabled by recent advances in digital signal processing provided by the state of the art



Figure 2: Time series data combining the contextual soundscape data captured from the HA (green gradient) with the corresponding interactions related to the user selected programs (yellow-red gradient) for subject 1 (top) and 2 (bottom).

The resulting four soundscape clusters were labeled according to the proportion of samples with different ground-truth labels within each cluster (Figure 1) while ambiguities were solved by examination of the cluster centroids. The first cluster mainly captured the 'quiet' class which is also validated by the cluster centroid having very low values of sound pressure level and noise floor. Thus, the environments assigned to this cluster will be represented as 'quiet'. The second cluster captured both 'speech in noise' and 'noise' classes which suggests that the numerical representations of these environments are similar. For simplicity, we label them as 'speech in noise'. The third and fourth cluster both captured mainly 'speech in quiet' with a small addition of other classes. As the third cluster captured samples with much higher sound pressure level and signal to noise ratio, it will be labeled as 'clear speech', while the fourth cluster with attributes of the samples closer to mean will be represented as 'normal speech'.

RESULTS

We refer to the user's selected volume and program choice as user preferences, and to the corresponding auditory environment as the context. Juxtaposing user preferences and the context allows us to learn which HA settings are selected in specific listening scenarios. To facilitate interpretation we assign each cluster a color from white to green gradient, in which increasing darkness correspond to increased noise in the context (quiet → clean speech → normal speech → speech in noise). Likewise, we assign each program a color from yellow to red gradient. Lighter colors define programs with an omnidirectional focus and added brightness. Darker colors indicate increasing attenuation of noise. This coloring scheme will apply throughout the paper.

Contextual user preferences

Figure 2 shows the user preference and context changes for both subjects, plotted across the hours of the day over the weeks constituting the full experimental period. Subject 1 most frequently selects programs which provide an omnidi-

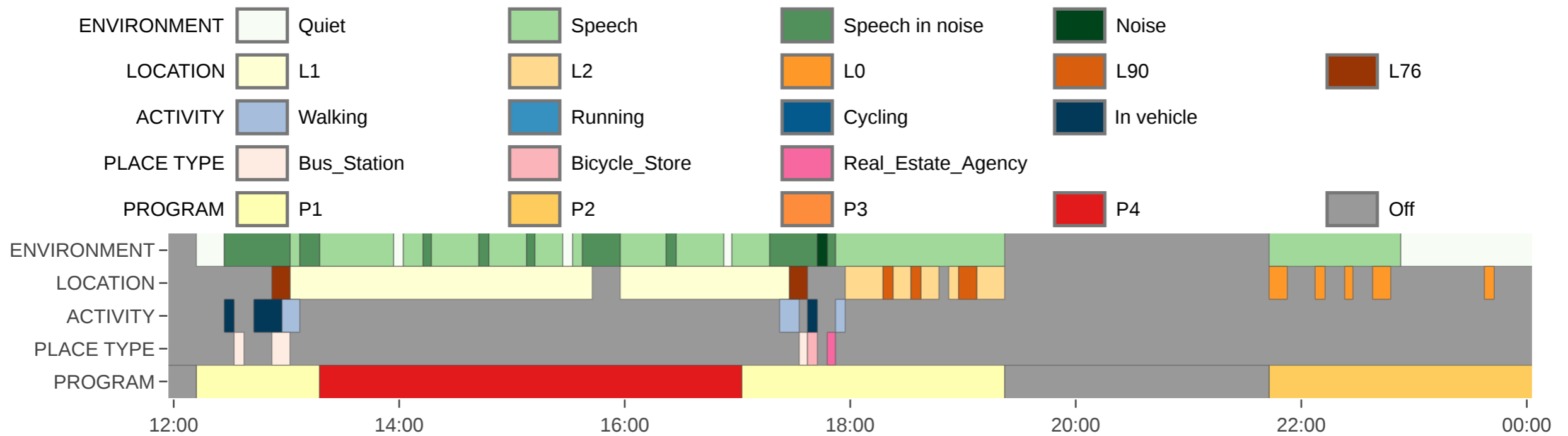


Figure 1: Example of user setting preferences (four programs - P1-P4) juxtaposed with different types of context captured in a continuous manner for a period of 12 hours. Environment is obtained through HA' in-built environment classification, location is represented as cluster membership based on HDBSCAN clustering, activity was estimated by Google's Activity Recognition API and place type was queried using Google Places API.

thanks ..
questions ?