دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

Introduction to Machine Learning: Applications in Bioinformatics

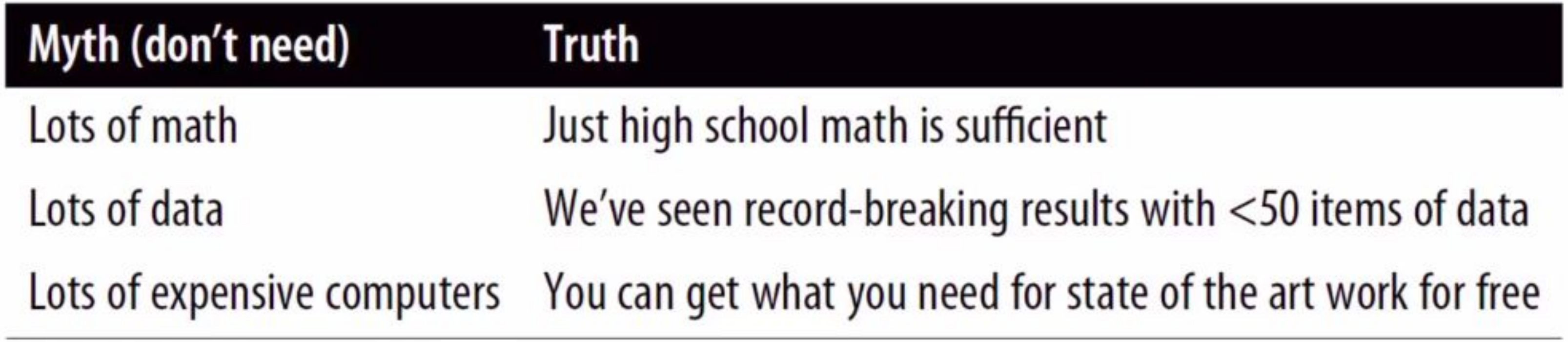
Armin Behjati
2020

* Some slides are inspired by Jeremy Howard

Outline:

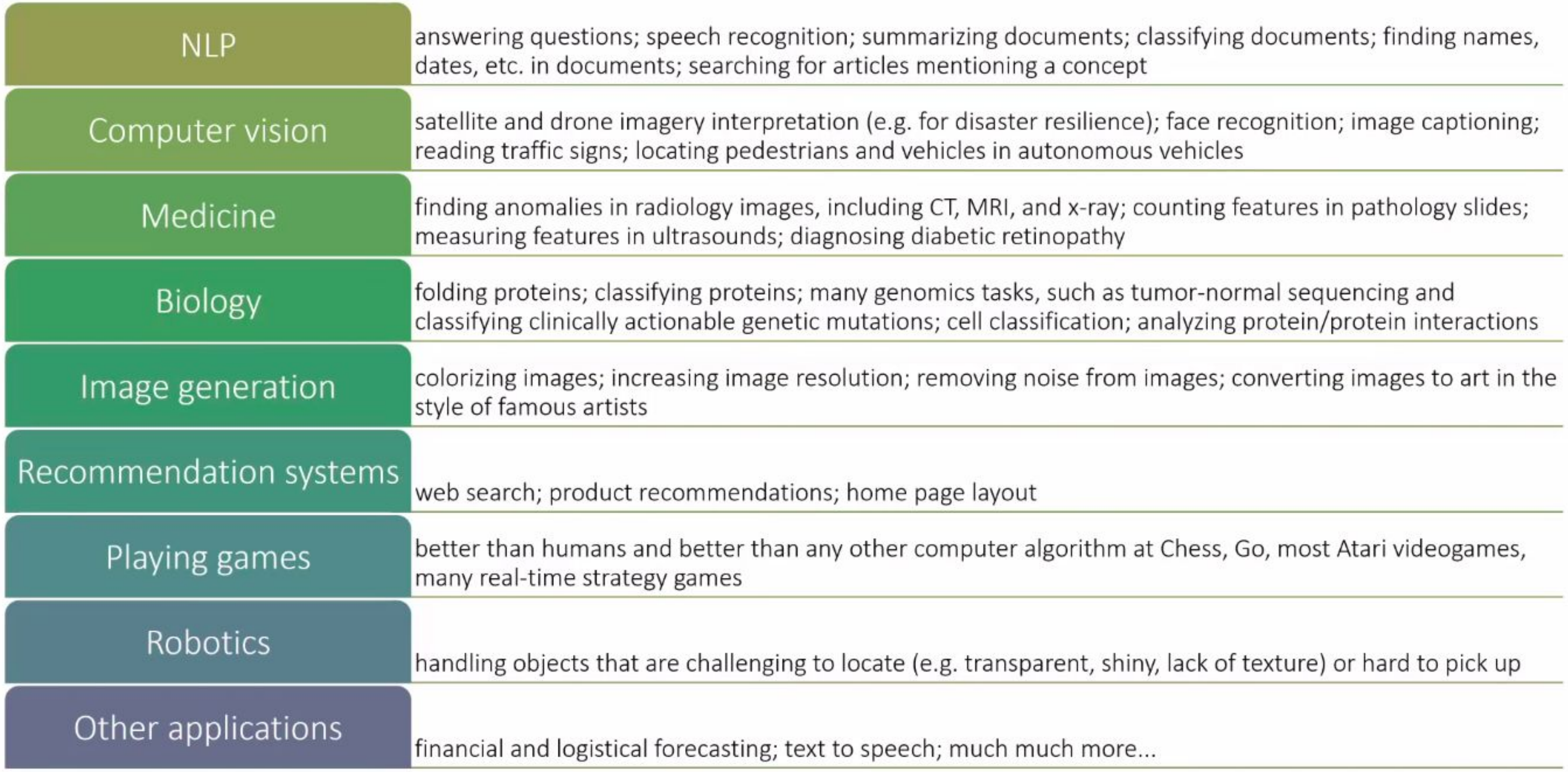
- Why deep learning
- Code demo
- A brief history of neural networks
- Modern neural network
- Protein secondary structure prediction problem
- NLP introduction
- Code Demo

What you don’t need, to do deep learning



Myth (don’t need)	Truth
Lots of math	Just high school math is sufficient
Lots of data	We’ve seen record-breaking results with <50 items of data
Lots of expensive computers	You can get what you need for state of the art work for free

Where is Deep Learning the best-known approach?



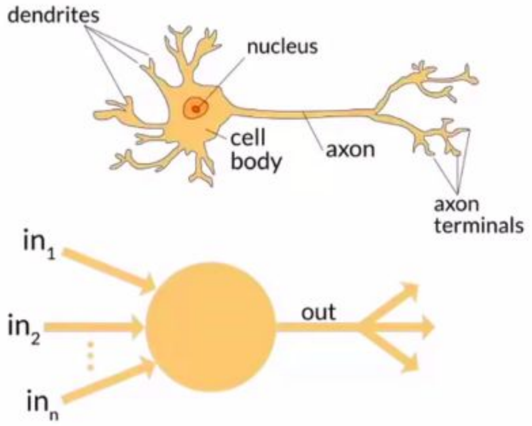
NLP	answering questions; speech recognition; summarizing documents; classifying documents; finding names, dates, etc. in documents; searching for articles mentioning a concept
Computer vision	satellite and drone imagery interpretation (e.g. for disaster resilience); face recognition; image captioning; reading traffic signs; locating pedestrians and vehicles in autonomous vehicles
Medicine	finding anomalies in radiology images, including CT, MRI, and x-ray; counting features in pathology slides; measuring features in ultrasounds; diagnosing diabetic retinopathy
Biology	folding proteins; classifying proteins; many genomics tasks, such as tumor-normal sequencing and classifying clinically actionable genetic mutations; cell classification; analyzing protein/protein interactions
Image generation	colorizing images; increasing image resolution; removing noise from images; converting images to art in the style of famous artists
Recommendation systems	web search; product recommendations; home page layout
Playing games	better than humans and better than any other computer algorithm at Chess, Go, most Atari videogames, many real-time strategy games
Robotics	handling objects that are challenging to locate (e.g. transparent, shiny, lack of texture) or hard to pick up
Other applications	financial and logistical forecasting; text to speech; much much more...

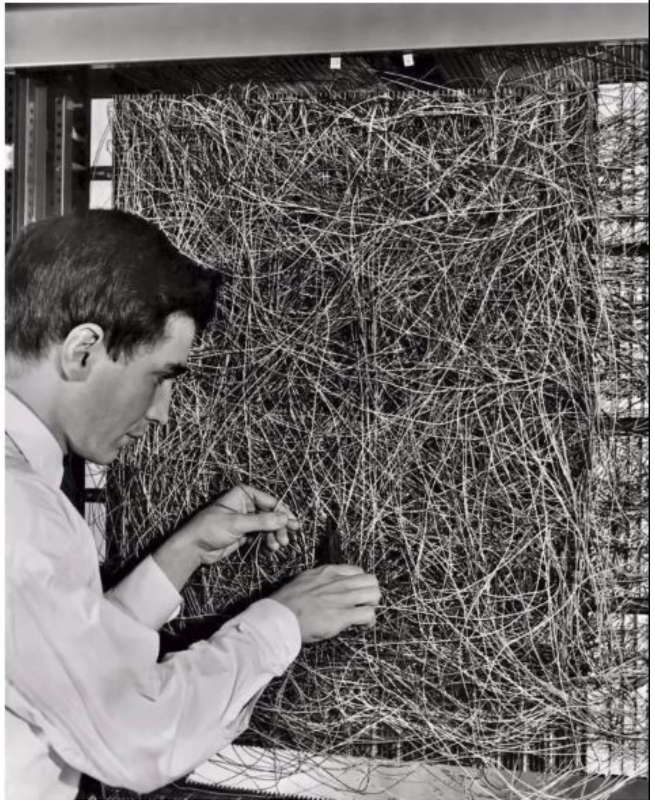
Code demo

Neural networks: a brief history

In 1943 Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, teamed up to develop a mathematical model of an artificial neuron. They declared that:

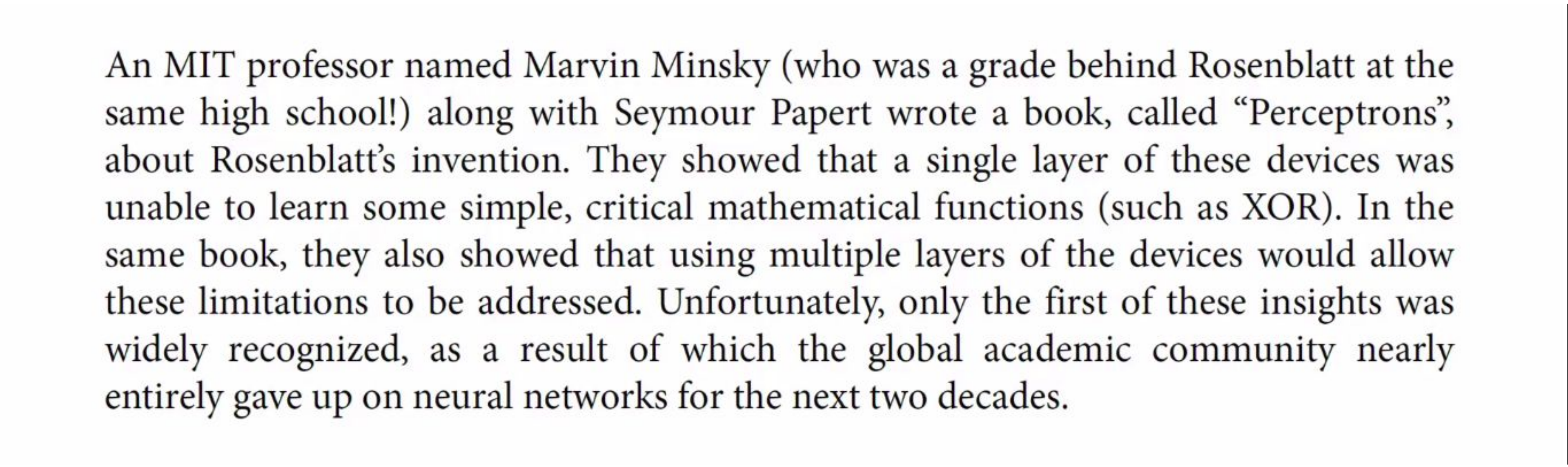
> *Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms.* (Pitts and McCulloch; A Logical Calculus of the Ideas Immanent in Nervous Activity)





"we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control". Frank Rosenblatt

The first AI winter, Thanks to misinterpretation of theory



An MIT professor named Marvin Minsky (who was a grade behind Rosenblatt at the same high school!) along with Seymour Papert wrote a book, called “Perceptrons”, about Rosenblatt’s invention. They showed that a single layer of these devices was unable to learn some simple, critical mathematical functions (such as XOR). In the same book, they also showed that using multiple layers of the devices would allow these limitations to be addressed. Unfortunately, only the first of these insights was widely recognized, as a result of which the global academic community nearly entirely gave up on neural networks for the next two decades.

The age of deep learning is here

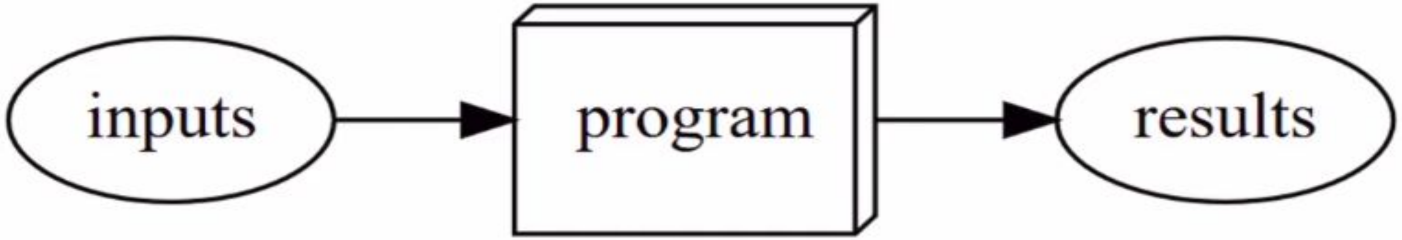
In the 1980's most models were built with a **second layer of neurons**, thus avoiding the problem that had been identified by Minsky. However, again a misunderstanding of the theoretical issues held back the field. **In theory, adding just one extra layer of neurons was enough** to allow any mathematical model to be approximated with these neural networks, **but in practice such networks were often too big and slow** to be useful.

Researchers showed 30 years ago that **to get practical good performance you need to use even more layers of neurons**. Neural networks are now finally living up to their potential, thanks to the understanding to use more layers as well as improved ability to do so thanks to improvements in computer hardware, increases in data availability, and algorithmic.

We now have "a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control".

Machine learning is, like regular programming, a way to get computers to complete a specific task. But how do you use regular programming to recognize cats and dogs?

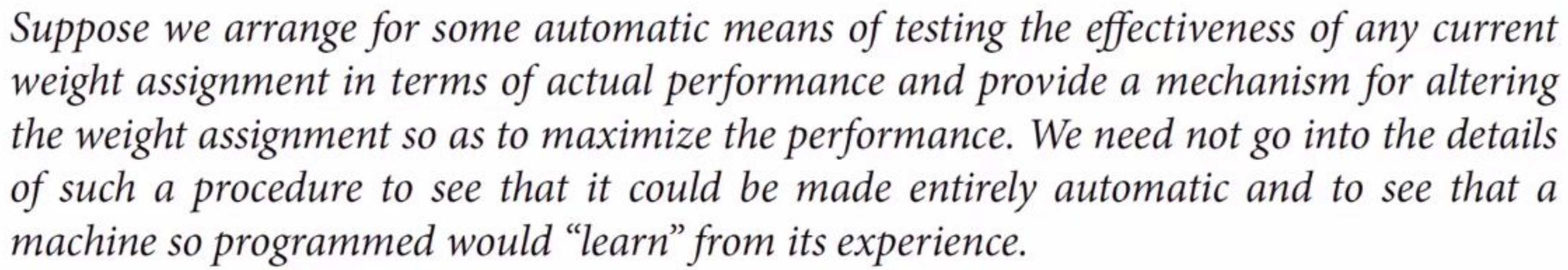
Normally, it's easy enough for us to write down the steps to complete a task when we're writing a program. In general, we write a function that looks something like this:



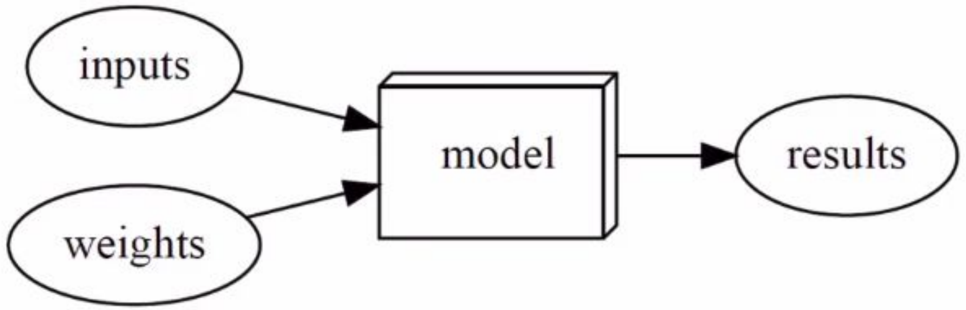
Right back at the dawn of computing, in 1949, an IBM researcher named Arthur Samuel started working on a different way to get computers to complete tasks, which he called *machine learning*. In his classic 1962 essay *Artificial Intelligence: A Frontier of Automation*, he wrote:

> *Programming a computer for such computations is, at best, a difficult task, not primarily because of any inherent complexity in the computer itself but, rather, because of the need to spell out every minute step of the process in the most exasperating detail. Computers, as any programmer will tell you, are giant morons, not giant brains.*

His basic idea was this: instead of telling the computer the exact steps required to solve a problem, show it examples of the problem to solve and let it figure out how to solve it itself. This turned out to be very effective.
Here is how he described his idea:



Suppose we arrange for some automatic means of testing the effectiveness of any current weight assignment in terms of actual performance and provide a mechanism for altering the weight assignment so as to maximize the performance. We need not go into the details of such a procedure to see that it could be made entirely automatic and to see that a machine so programmed would "learn" from its experience.



Also note that once the model is trained – that is, once we've chosen our final, best, favorite weight assignment – then we can think of the weights as being *part of the model*, since we're not varying them any more.

Therefore actually *using* a model after it's trained looks like Figure 1-7.

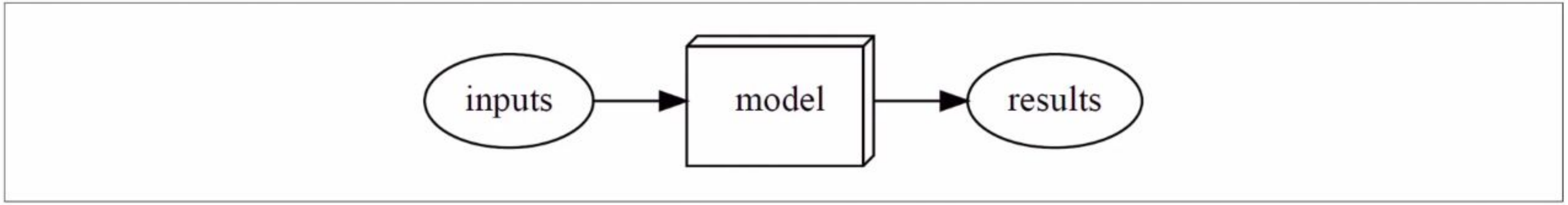


Figure 1-7. Using a trained model as a program

Bring on the neural networks!

It's not at all obvious what the model might look like for an image recognition program, or for understanding text, or for many other interesting problems we might imagine.

What we would like is some kind of function that is so flexible that it could be used to solve any given problem, just by varying its weights. Amazingly enough, this function actually exists! It's the neural network.

A mathematical proof called the *universal approximation theorem* shows that this function can solve any problem to any level of accuracy, in theory.

Bring on SGD!

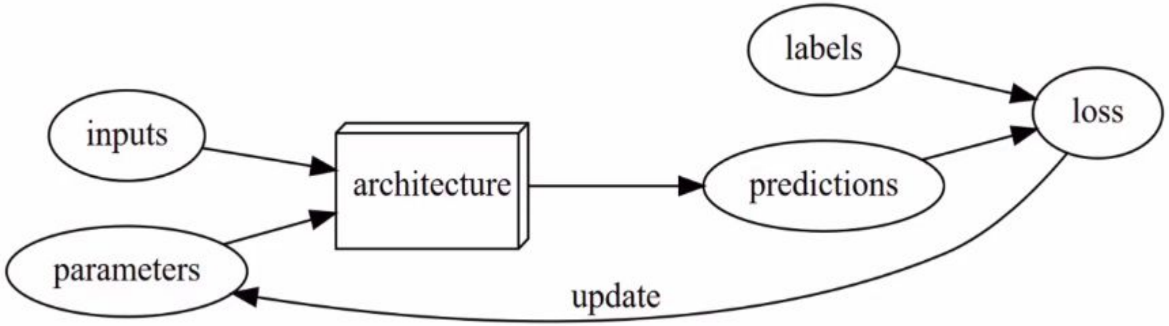
The fact that neural networks are so flexible means that, in practice, they are often a suitable kind of model, and you can focus your effort on the process of training them, that is, of finding good weight assignments.

But what about that process? What we'd like here as well is a completely general way to update the weights of a neural network, to make it improve at any given task. Conveniently, this exists!

This is called *stochastic gradient descent* (SGD)

Samuel was working in the 1960s, but terminology has changed. Here is the modern deep learning terminology for all the pieces we have discussed:

- The functional form of the *model* is called its *architecture* (but be careful--sometimes people use *model* as a synonym of *architecture*, so this can get confusing) ;
- The *weights* are called *parameters* ;
- The *predictions* are calculated from the *independent variables*, which is the *data* not including the *labels* ;
- The *results* of the model are called *predictions* ;
- The measure of *performance* is called the *loss*;
- The loss depends not only on the predictions, but also the correct *labels* (also known as *targets* or *dependent variable*), e.g. "dog" or "cat"



Limitations inherent to machine learning

From this picture we can now see some fundamental things about training a deep learning model:

- A model cannot be created without data ;
- A model can only learn to operate on the patterns seen in the input data used to train it ;
- This learning approach only creates *predictions*, not recommended *actions* ;
- It's not enough to just have examples of input data; we need *labels* for that data too (e.g. pictures of dogs and cats aren't enough to train a model; we need a label for each one, saying which ones are dogs, and which are cats)

Consider how a model interacts with its environment

This can create feedback loops, such as:

- A *predictive policing* model is created based on where arrests have been made in the past. In practice, this is not actually predicting crime, but rather predicting arrests, and is therefore partially simply reflecting biases in existing policing processes;
- Law enforcement officers then might use that model to decide where to focus their police activity, resulting in increased arrests in those areas;
- These additional arrests would then feed back to re-training future versions of the model;
- This is a *positive feedback loop*, where the more the model is used, the more biased the data becomes, making the model even more biased, and so forth

Stochastic Gradient Descent (SGD)

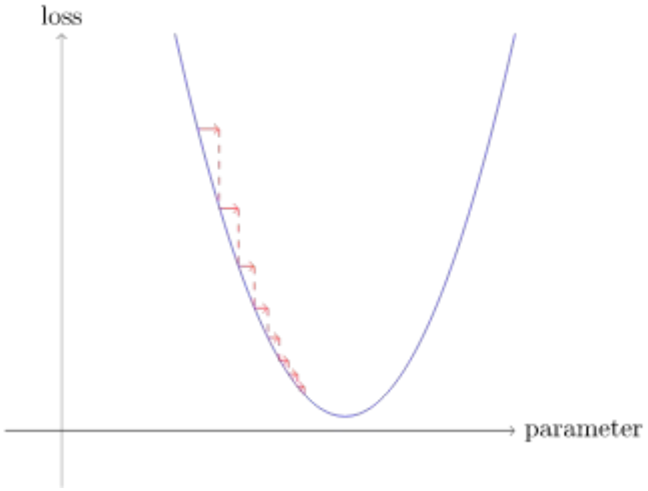
here are the steps that we are going to require, to create a machine learning classifier:

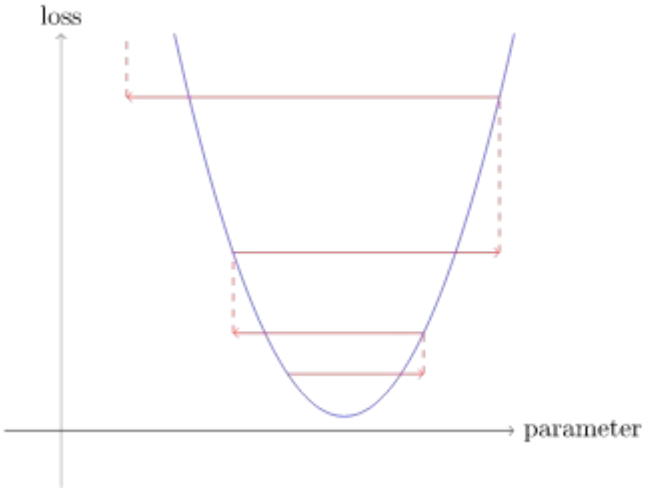
1. *Initialize* the weights.
2. For each example, use these weights to *predict* a label.
3. Based on these predictions, calculate how good the model is (its *loss*).
4. Calculate the *gradient*, which measures for each weight, how changing that weight would change the loss.
5. *Change* all the weights based on that calculation.
6. Go back to the step 2, and *repeat* the process.
7. Iterate until you decide to *stop* the training process (for instance, because the model is good enough or you don't want to wait any longer).

Stepping With a Learning Rate:

Deciding how to change our parameters based on the values of the gradients is an important part of the deep learning process. Nearly all approaches start with the basic idea of multiplying the gradient by some small number, called the *learning rate* (LR). Once you've picked a learning rate, you can adjust your parameters using this simple function:

```
w -= gradient(w) * lr
```





Protein Secondary Structure Prediction

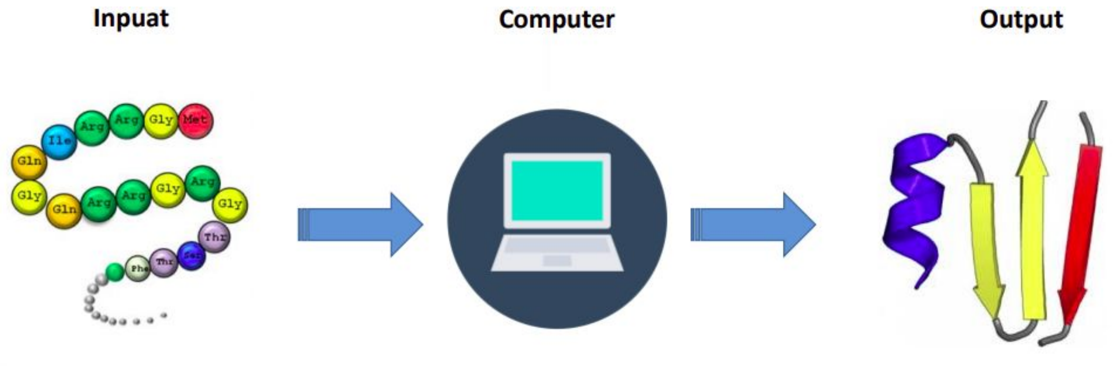


*image from Bahram Mohammadpour

Language Model:

What we call a language model is a model that has been trained to guess what the next word in a text is (having read the ones before). This kind of task is called *self-supervised learning*: we do not need to give labels to our model, just feed it lots and lots of texts. It has a process to automatically get labels from the data, and this task isn't trivial: to properly guess the next word in a sentence, the model will have to develop an understanding of the language.

Transfer learning:

Even if our language model knows the basics of the language we are using in the task (e.g., our pretrained model is in English), it helps to get used to the style of the corpus we are targeting. It may be more informal language, or more technical, with new words to learn or different ways of composing sentences.

Example details in the code Demo

Codes and slides are available at:

github.com/armheb/Bio_pre_workshop_2020

Thanks!

armin.beh96@gmail.com

bioinformatics.aut.ac.ir