Soooooo, here we are: Neural networks

**There are more content in every course. Need dedication.

Week 1: Neural Networks, Tensorflow implementation, Python implementation, Vectorization

Neural networks and decision trees: Some of the most widely used machine learning algorithms. You get to use them.

Also: **Practical advice about ML system building**. You have decisions to make. How do you make them?

If you wanna succeed, you need to start with right decisions.

INFERENCE: Download parameters of an NN that someone trained before, and use the NN with those parameters.

(Quick summary about the way biological brain handles thinking)

Resurgence?

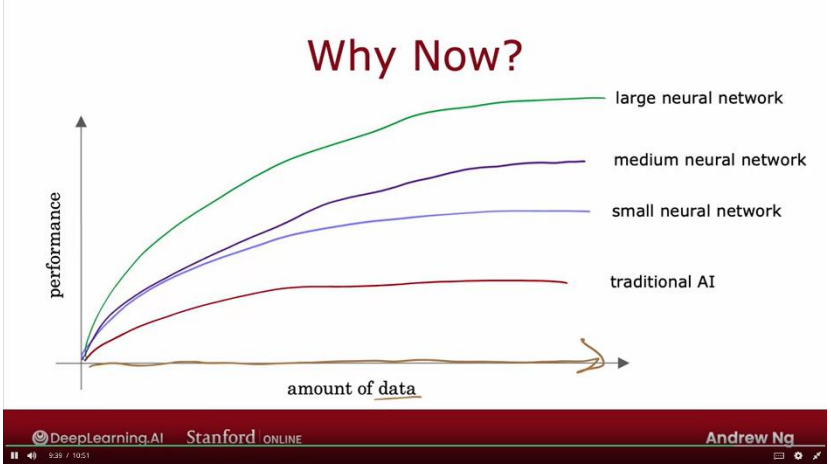
They revolutionized area after area. One of the first was **speech recognition**, then **computer vision**. New few years, **NLP** became a thing. Now it's everywhere from health to recommendations.

Now NN's have nothing similar to biological brain, but early stages aimed to produce algorithms that mimic the way our brain works.

**A neuron only takes one or more inputs, and gives out one or more outputs. These input's are numbers and outputs are computed values. They are derived from inputs with an activation function.

**Idea of NN was present in the past as well. Why did it become popular now??

Son onyıllardaki dijitalleşme, her şeyden elde edilen veriyi çook büyük miktarlarda yaptı. Her şeyin kaydı artık dijital mesela ve çok fazla dijital veri var her yerde. Ve:

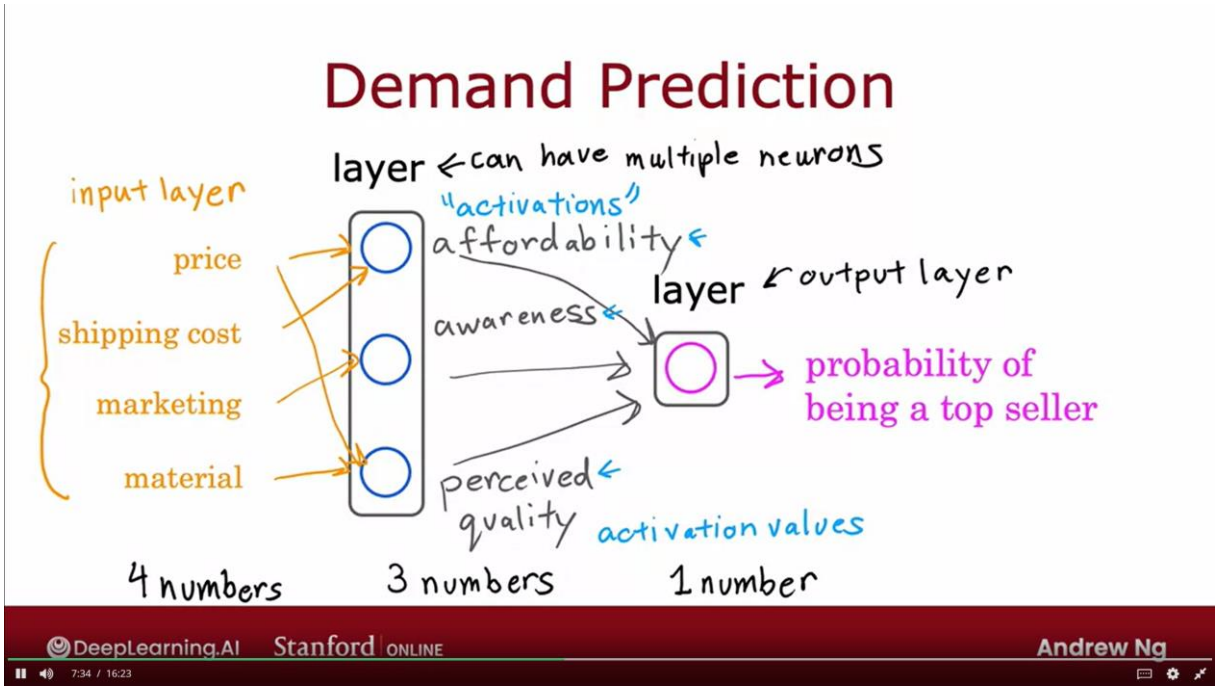


EK: İşlemcilerin ve GPU'ların son yıllarda çok gelişmesinin de yardımı oldu.

Demand Prediction

Şu ana kadar gördüğümüz logistic regression süreçlerini tek bir nörona indirgediğini düşün. Nöron input alıp a'yı verir (aktivasyon fonksiyonunun çıktısı).

Bu nöronları al, yan yana üst üste koy → artık yapay sinir ağın var.



Böyle bir örnek yapımız var.

**Burada hangi nöronun neyi input aldığını biz belirledik ama büyük bir yapı oluştururken buna elle karar veremezsin. ÇÖZÜM: Her nöron sonraki katmandaki her nörona bağlı. Ağ, önemli olanı öğrenip önemsiz olanın katsayısını azaltmalı, azaltır. Her katmanı bir sonrakine vektör veriyor olarak düşünebilirsin.

Neden "hidden layer"? Veri setinde girdi ve çıktı var ama aradaki değerler, türetilmiş şeyler vs yok.

**Input layer'ı görme: Ortadaki parametrelerle bir "logistic regression" problemine döndü olay. Sadece orijinal özellikler yerine türetilmiş olanları kullanıyorsun. Olay bu şekilde bakabilirsin. Kendi feature'larını öğrenip onlarla tahmin yapıyor sistem. SİNİR AĞI, FEATURE'LARI ELLE AYARLAMAK YERİNE KENDİ KENDİNE BULUR, AĞIRLIK VERİR, AYARLAR.

**SİNİR AĞINI EĞİTİRKEN BU FEATURE'LARI SEN DEĞİL, O KENDİSİ AYARLIYOR. İYİ YAPAN DA BU.

SORU: HİPERPARAMETRELERİ NASIL AYARLIYORUZ? Cevabı ileride.

Bu soru, sinir ağı mimarisinin bir problemi. Cevabı kursun ileri bölümlerinde.

INPUT LAYER is referred as **LAYER 0**.

Example: Recognizing Images

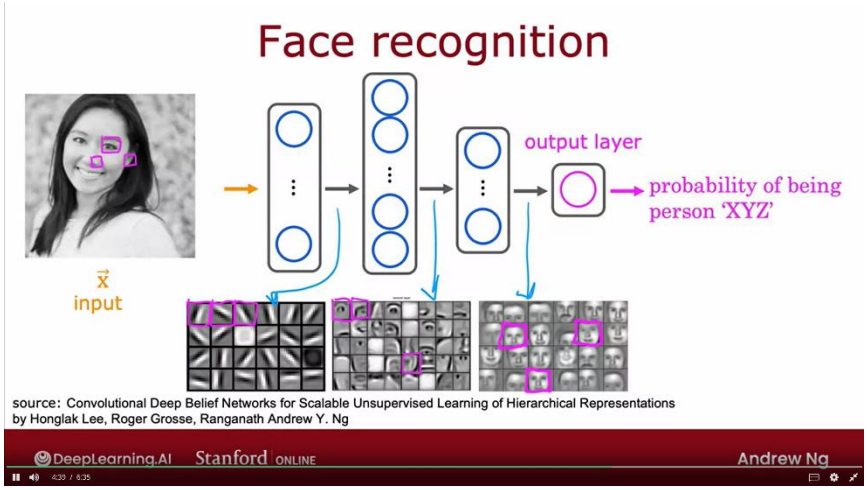
Input: Resim (1000*1000 pikseller) Output: Identity, gender etc.

Bir milyon pikselin var. Onlara bakıp insanı tanıyabilir misin? Nasıl?

Çözüm: inputu işlk katmana ver, o katman feature çıkarsın. Devam, devam, deavam → Çıktı katmanı

Sisteme çok fazla insan resmi gösterirsin.

Örnek: ilk katmanlardaki nöronlar basit çizgiler arıyor. İkinci katmanda mesela, daha göze benzeyen, burna, kulağa benzeyen bir yapıyı arar hale geliyor. Üçüncüde yüzün genel şekline bakıyor.

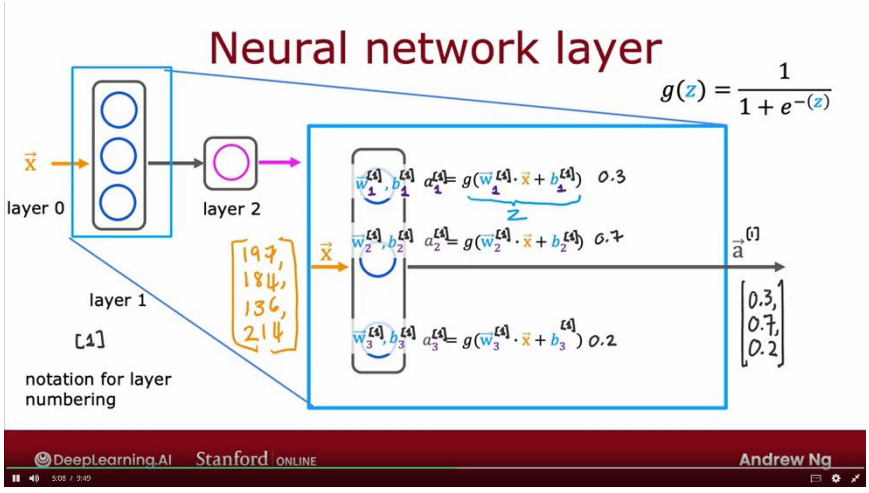


Aynı şekilde araba tanıma yaparsan: Basit çizgiler, tekerlek, kapı gibi belirli bölgeler, aracın genel hatları… Bunları sırayla arar. Bilgisayarlı görüde bunu uyguluyoruz.

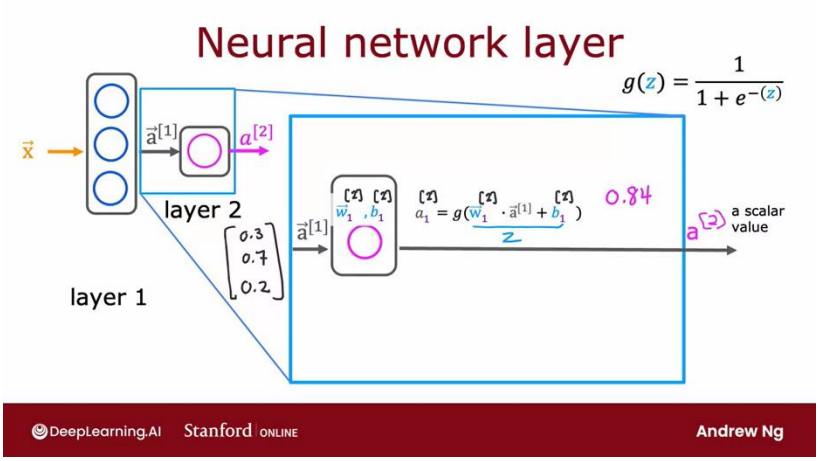
İleride el yazısıyla rakam tanıma uygulayacağız.

Şu anki konu: KATMANLARI ANLAMA VE OLUŞTURMA

Bir katmandaki her nöronun içinde, önceki haftada çalıştırdığın aktivasyon fonksiyonu (sigmoid mesela) çalışıyor. Her nöronun inputları w ve b (nörona has), çıktıları sigmoid fonksiyonunun değeri. Sonra bunu vektör halinde bir sonraki nörona aktarır.



Yanda 1. katman için hesaplamalar var.



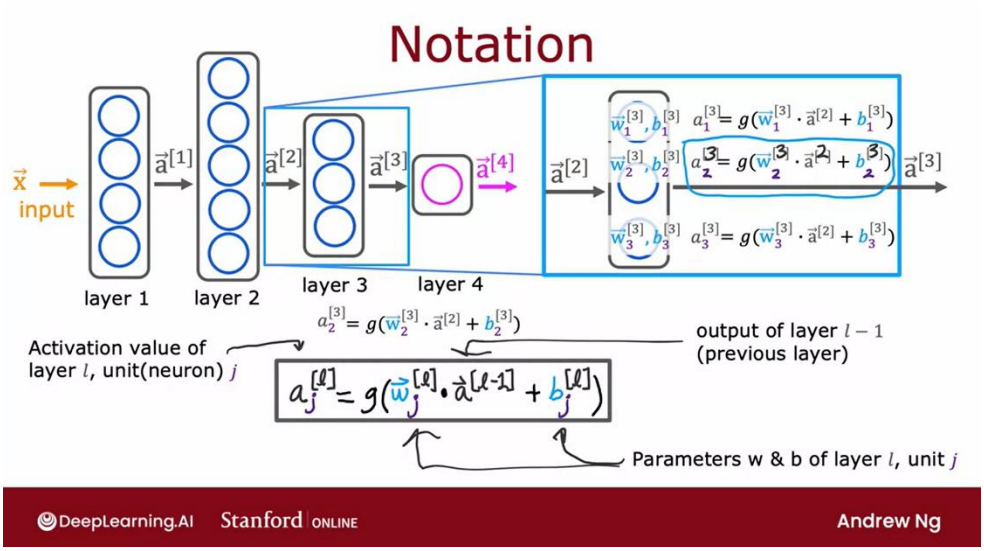
Çıkış değerini buldun. Şimdi binary sınıflandırma yapıyorsan 0.5'e karşı durumuna göre etiket koyabilirsin.

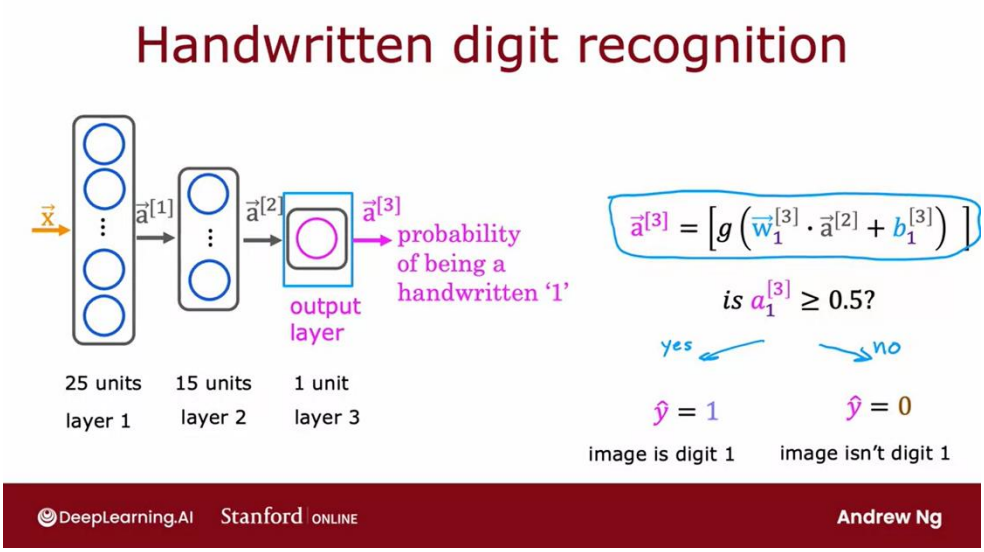
THAT IS HOW A NEURAL NETWORK WORKS. GET, COMPUTE, PASS TO NEXT LAYER.

More complex neural networks

Basic Principle: Every layer passes it's parameters to the next layer.

g(w . a + b) → w: kendi parametreleri, a: önceki katmandan gelen vektör





All we have is a chain of computations. This standard process is called **Forward propagation.** Later we will learn about **"Backpropagation"**, which is used for learning.

İlk katmanlarda çok nöron, ilerledikçe daha az nöron → Tipik bir yaklaşım

***Everything we have done so far, can also be done while implementing linear regression.**

TENSORFLOW AND KERAS

Way to define an NN layer is shown below:

- `linear_layer = tf.keras.layers.Dense(units=1, activation = 'linear',)`

*Tensor: Fancy name for an array.

In the first lab, we initialized a "**Linear layer, with one neuron"** for house price prediction

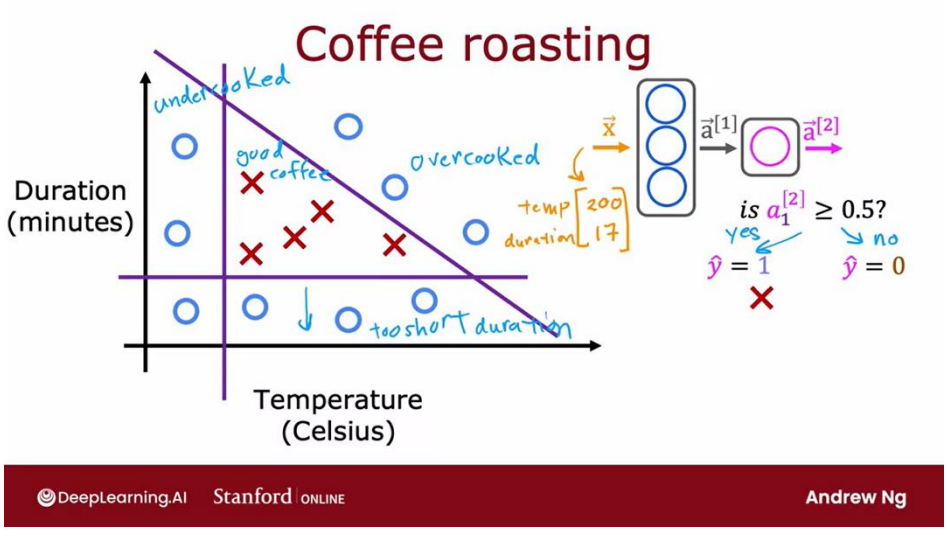
In Tensorflow, we'll most likely be using multi layer multi neuron models. To create them, use Sequential and inside send a list of tf.keras.layers.Dense()

```
model = Sequential(
[tf.keras.layers.Dense(1, input_dim=1,  activation = 'sigmoid', name='L1')]
)
```

Note: There are other types than Dense.

TENSORFLOW IMPLEMENTATION

How to code inference in tensorflow?



This is what we are going to optimize.

NOTES FROM LABS

Normalize Data

Fitting the weights to the data (back-propagation, covered in next week's lectures) will proceed more quickly if the data is normalized. This is the same procedure you used in Course 1 where features in the data are each normalized to have a similar range. The procedure below uses a Keras normalization layer. It has the following steps:

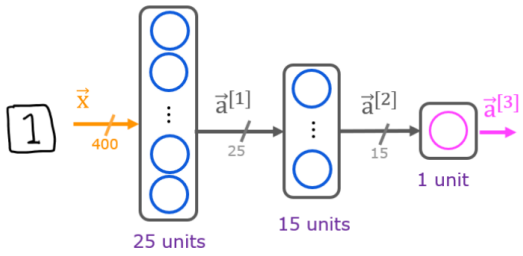
- create a "Normalization Layer". Note, as applied here, this is not a layer in your model.
- 'adapt' the data. This learns the mean and variance of the data set and saves the values internally.
- normalize the data.
  It is important to apply normalization to any future data that utilizes the learned model.

```
norm_l = tf.keras.layers.Normalization(axis=-1)
norm_l.adapt(X)  # learns mean, variance
Xn = norm_l(X)
```

2.3 Model representation

The neural network you will use in this assignment is shown in the figure below.

- This has three dense layers with sigmoid activations.
  - Recall that our inputs are pixel values of digit images.
  - Since the images are of size 20×20, this gives us 400 inputs



- The parameters have dimensions that are sized for a neural network with 25 units in layer 1, 15 units in layer 2 and 1 output unit in layer 3.
  - Recall that the dimensions of these parameters are determined as follows:
    - If network has s_{in} units in a layer and s_{out} units in the next layer, then
      - W will be of dimension $s_{in} \times s_{out}$.
      - b will a vector with s_{out} elements
  - Therefore, the shapes of `W`, and `b`, are
    - layer1: The shape of `W1` is (400, 25) and the shape of `b1` is (25,)
    - layer2: The shape of `W2` is (25, 15) and the shape of `b2` is: (15,)
    - layer3: The shape of `W3` is (15, 1) and the shape of `b3` is: (1,)

> **Note:** The bias vector `b` could be represented as a 1-D (n,) or 2-D (1,n) array. Tensorflow utilizes a 1-D representation and this lab will maintain that convention.

from pra. lab

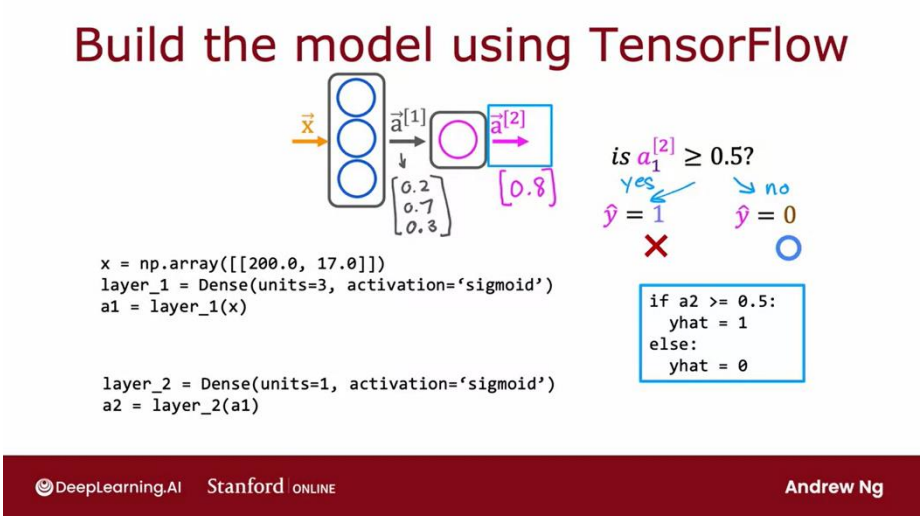
- **INSTEAD OF DEALING WITH ARRAYS, YOU CAN USE MATRIX MULTPLICATION TO HANDLE AN ENTIRE LAYERS COMPUTATION. THEN TAKE SIGMOID OF THE FINAL MATRIX. EZ.**

2.8 NumPy Broadcasting Tutorial (Optional)

In the last example, $\mathbf{Z} = \mathbf{XW} + \mathbf{b}$ utilized NumPy broadcasting to expand the vector $\mathbf{b}$. If you are not familiar with NumPy Broadcasting, this short tutorial is provided.

$\mathbf{XW}$ is a matrix-matrix operation with dimensions $(m, j_1)(j_1, j_2)$ which results in a matrix with dimension (m, j_2). To that, we add a vector $\mathbf{b}$ with dimension $(1, j_2)$. $\mathbf{b}$ must be expanded to be a (m, j_2) matrix for this element-wise operation to make sense. This expansion is accomplished for you by NumPy broadcasting.

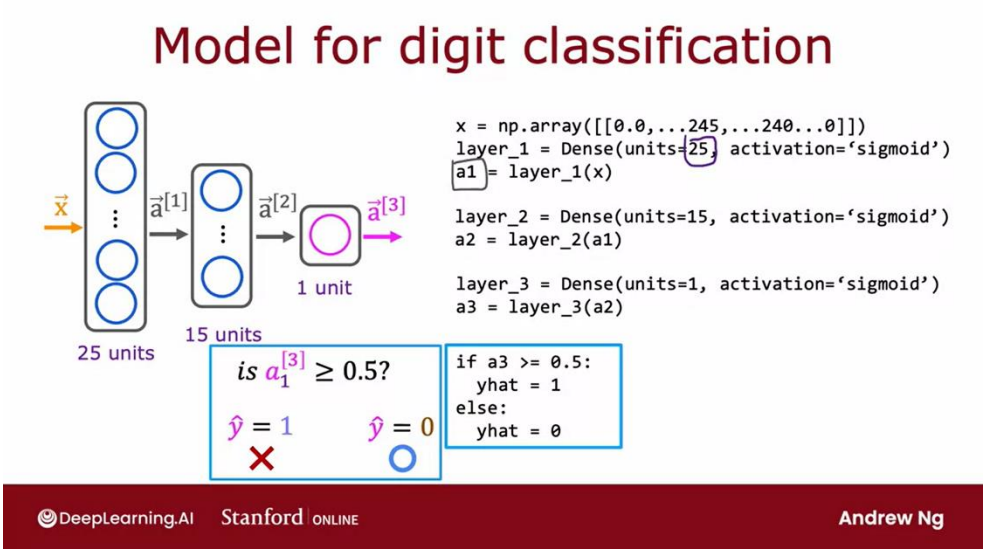
- **PRACTICE LAB MENTIONS IMPORTANT DETAILS. EXAMINE AGAIN LATER.**



Layer after layer,

1) initialize it
2) compute next

Let's revisit hand written digit classification problem in code:



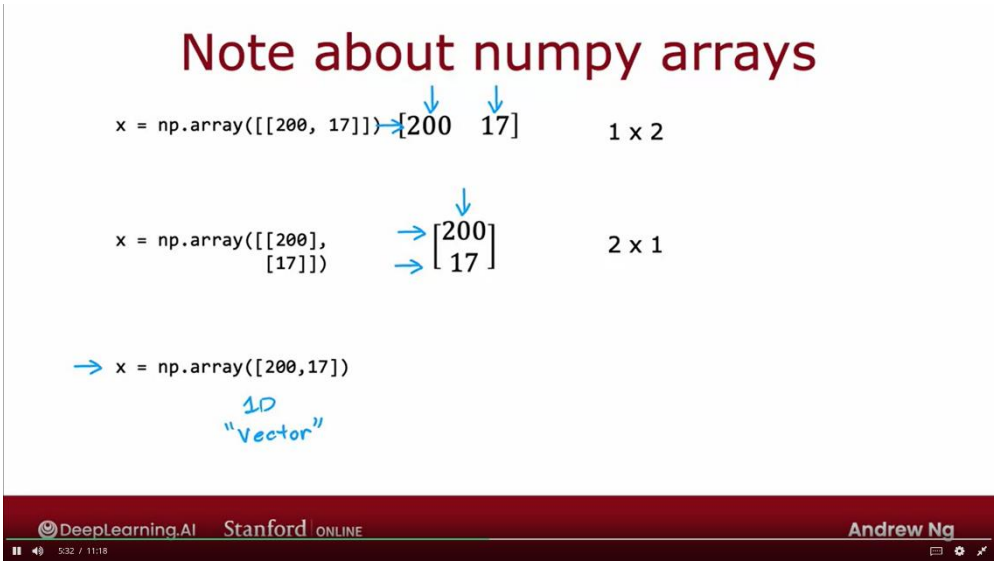
IMPORTANT: Doc of "DENSE"

Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bia is True). These are all attributes of Dense

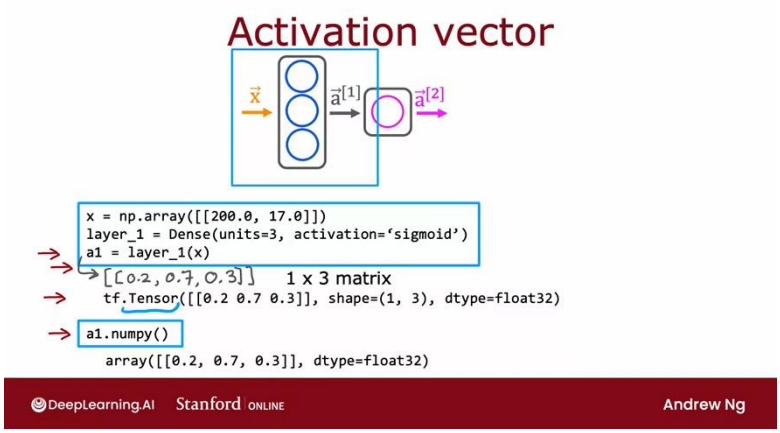
NEXT: **HOW TENSORFLOW (and numpy) HANDLES DATA**

There are small differences, inconsistencies even.

**np.array'ler matris olarak tutulur, matrisin de her satırı [] arasında olur. Matris mantığıyla tek satırlık bir dizin olsa bile şu şekilde yazılır: [[1, 2, 3]]. [[1, 2, 3], [4, 5, 6]] (bu da 2x3)

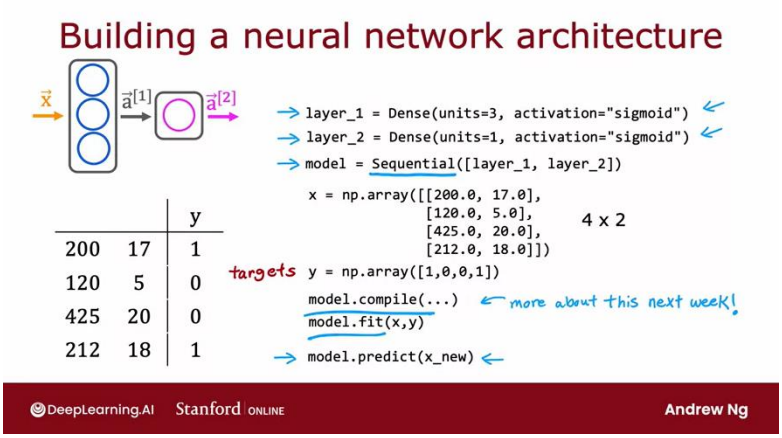


İki [] → 2 boyutlu bir matris, vektör Bir [] → 1 boyutlu dizi



*TENSOR: Matris, hesaplamalar için verimli.

NN BUILDING



Katmanları öncede olduğu gibi tanımlıyoruz, sonra da "Sequential" ile birbirine bağlı katmanlar oluşturuyoruz. **You can do the same inside Sequential by giving every layer a unique name. (we'll be using this approach)**

Then compile and fit.

If new x comes, just make the prediction.

ANOTHER EXAMPLE:

```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[e..., 245, ..., 17], [o..., 200, ..., 184])
y = np. array([1,0])
model. fit(x,y)
model. predict(x_new)
```

***From coffe roasting lab:

"Fitting the weights to the data (back-propagation, covered in next week's lectures) will proceed more quickly if the data is normalized. This is the same procedure you used in Course 1 where features in the data are each normalized to have a similar range."

- create a "Normalization Layer". Note, as applied here, this is not a layer in your model.
- 'adapt' the data. This learns the mean and variance of the data set and saves the values internally.
- normalize the data.

```
norm_l = tf.keras.layers.Normalization(axis=-1)
norm_l.adapt(X)  # learns mean, variance
Xn = norm_l(X)
```

Let's build the "Coffee Roasting Network" described in lecture. There are two layers with sigmoid activations as shown below:

```
tf.random.set_seed(1234)  # applied to achieve consistent results
model = Sequential(
    [
        tf.keras.Input(shape=(2,)),
        Dense(3, activation='sigmoid', name = 'layer1'),
        Dense(1, activation='sigmoid', name = 'layer2')
     ]
)
```

> **Note 1:** The `tf.keras.Input(shape=(2,))` specifies the expected shape of the input. This allows Tensorflow to size the weights and bias parameters at this point. This is useful when exploring Tensorflow models. This statement can be omitted in practice and Tensorflow will size the network parameters when the input data is specified in the `model.fit` statement.
> **Note 2:** Including the sigmoid activation in the final layer is not considered best practice. It would instead be accounted for in the loss which improves numerical stability. This will be described in more detail in a later lab.

- Use "model.summary()" for information about the network.
- The model.compile statement defines a loss function and specifies a compile optimization.
- "model.fit()" uses GD to fit weights to the data.

- WHEN YOU NORMALIZE WHILE TRAINING, NORMALIZE YOUR TRAINING DATA AS WELL, BEFORE YOU MAKE A PREDICTION.

```
yhat = np.zeros_like(predictions)
for i in range(len(predictions)):
    if predictions[i] >= 0.5:
        yhat[i] = 1
    else:
        yhat[i] = 0
print(f"decisions = \n{yhat}")
```

```
decisions =
[[1.]
 [0.]]
```

This can be accomplished more succinctly:

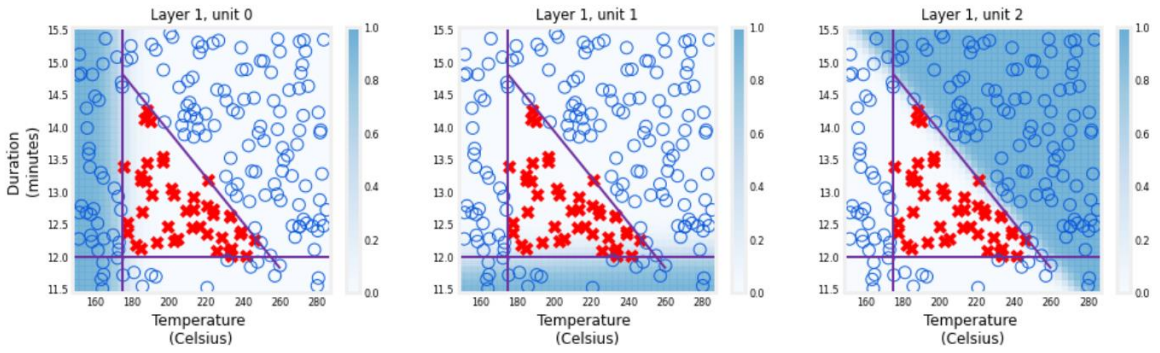
```
yhat = (predictions >= 0.5).astype(int)
print(f"decisions = \n{yhat}")
```

```
decisions =
[[1]
 [0]]
```

***Note for better writing.

Note: “astype” is unique to numpy arrays.

```
plt_layer(X,Y.reshape(-1,),W1,b1,norm_l)
```

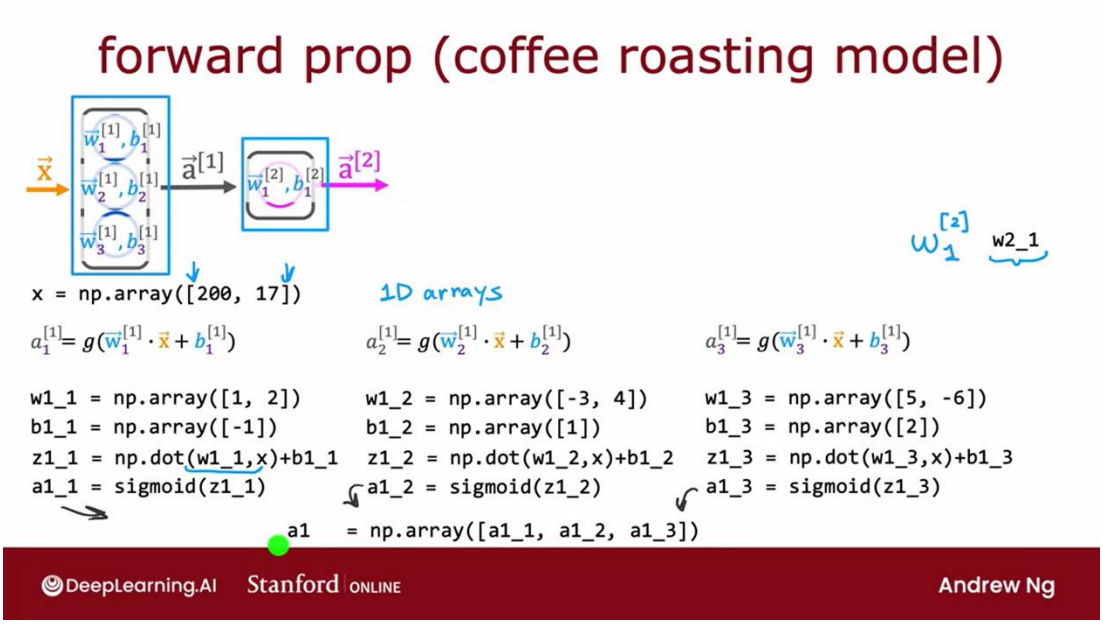


The shading shows that each unit is responsible for a different "bad roast" region. unit 0 has larger values when the temperature is too low. unit 1 has larger values when the duration is too short and unit 2 has larger values for bad combinations of time/temp. It is worth noting that the network learned these functions on its own through the process of gradient descent. They are very much the same sort of functions a person might choose to make the same decisions.

Basically, every neuron has a region. Problem is split into different sub-problems and each neuron handles a different region.

IMPLEMENTATION IN PYTHON (from scratch**)**

Following labs will include these codes.



a1 is a vector. Compute it first, then a2, then a3… Method of cumputation is the same for all of ‘em.

Instead of hardcoding every single neuron: Just pack it inside a function. This part just tells about it’s python code. For every layer, get a_in and compute a_out. Inside a layer, compute a_out[i] for every neuron.

*__Even when using an advanced framework:__ Ability to debug is valuable.

“For Fun”: Is there a path to AGI?

Everythin in this part is quoted directly from Andrew NG:

“Progress towards ANI does not mean we’re closing AGI. (Is it true? If yer, how much?)

Is there any hope? Brain is amazingly adaptable to a wide range of inputs. Can the same brain tissue feedl touch, sound, see etc. If so, **CAN AN ALGORITHM DO IT? That’s the question.**

Avoid overhyping.

“

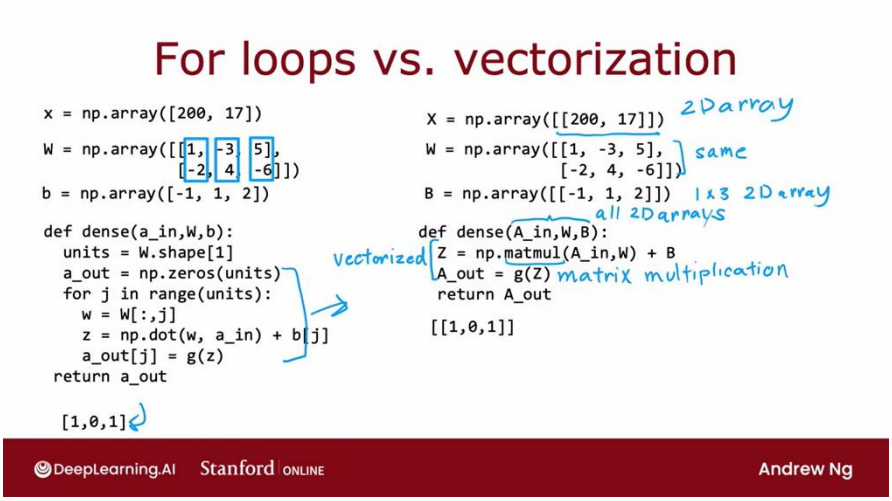
VECTORIZATION

We need to do very large matrix operations. We desperately need them do implement deep learning.

MAIN IDEA: For loops are “meh”, vectorization is “wow”.

**Can use np.matmul to multiple matrixes.

IMPORTANT: Examine codes below. In the vectorized approach, every np.array, including x and b, are 2D MATRIXES, EVEN THOUGH THEY ARE 1X3 or sth.



Next videos just tell about linear algebra, everything is already known to me.

“”””Vectorized approach runs amazingly faster.””””

That’s why matrix operations are so crucial. Can’t wait days unless you really have to.