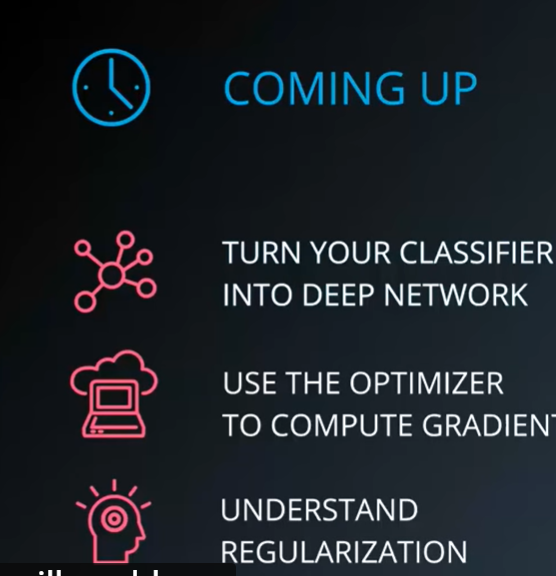
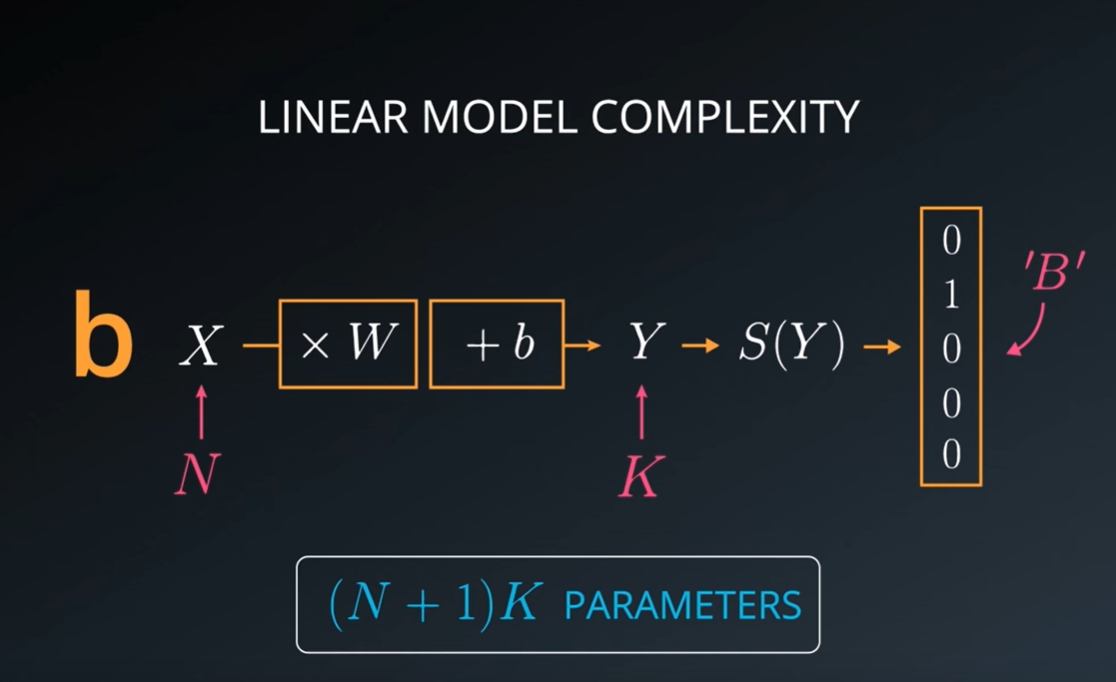
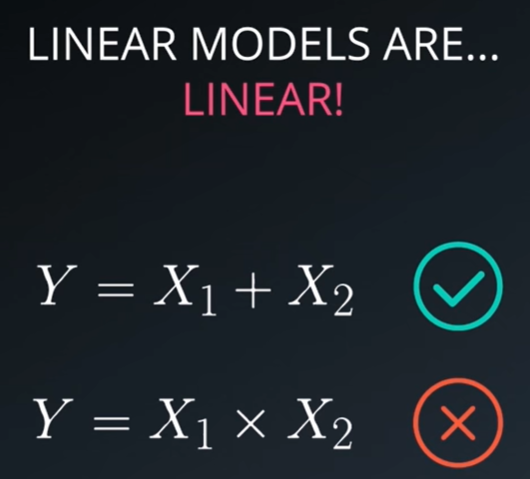
Deep Neural Network



Number of parameters:



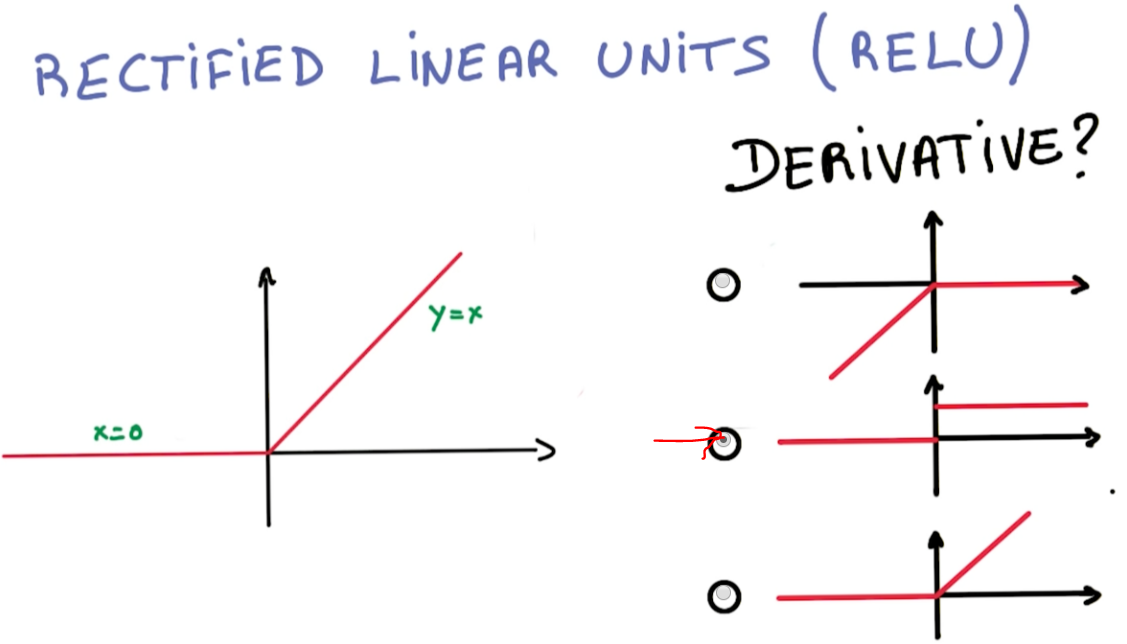
* Daca ai N inputs si K Outputs. Atunci vei avea (N + 1)K parametrii sa ii folosesti



* Linear model functioneaza bine cand faci suma, dar la produs NU merge.
* Este ieftin sa folosesti linear model.
* Dar SCOPUL nostrum este sa introducem NON-LINIARITIES

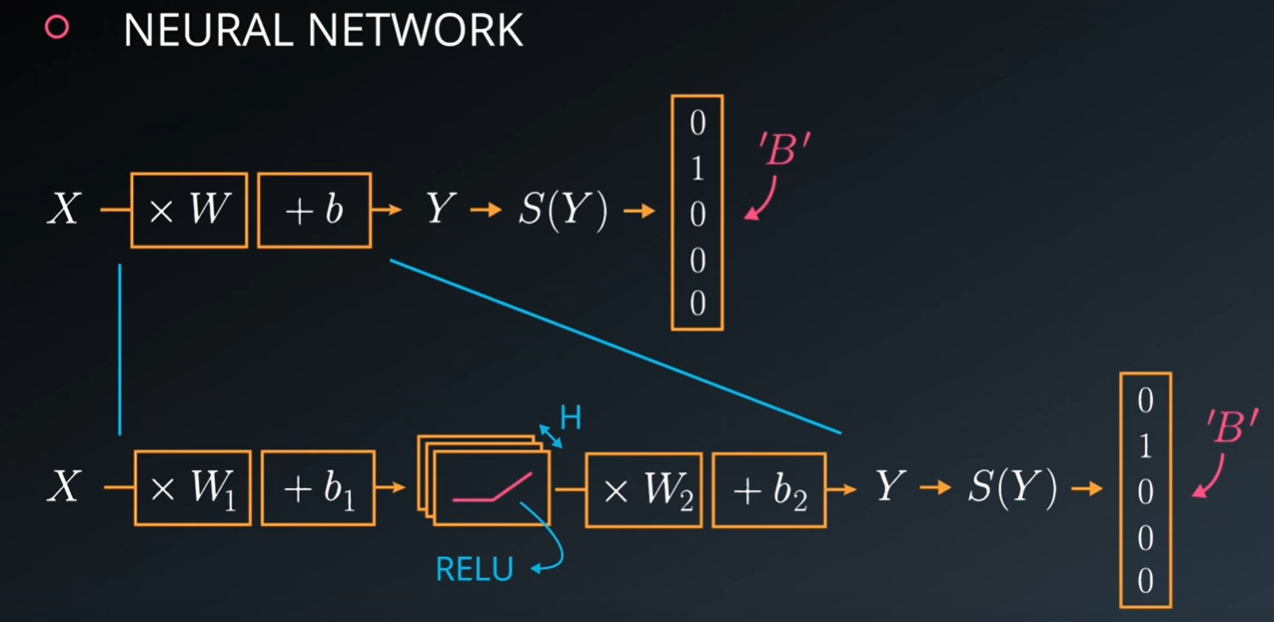
Link: <https://www.youtube.com/watch?v=12AYOYDrpfQ>

**Non-Liniar Functions**



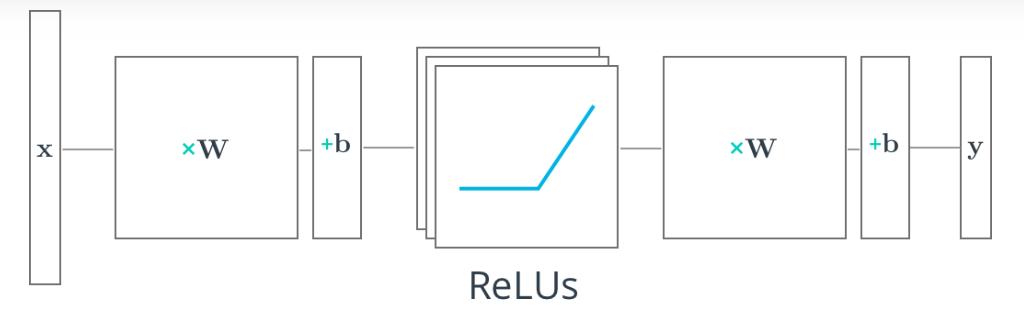
RELU = Reactified Liniar Units (este cea mai lazy)

* Acesta este linear atunci cand y = x, si 0 oriunde in alta parte
* In figura vei observa si derivata



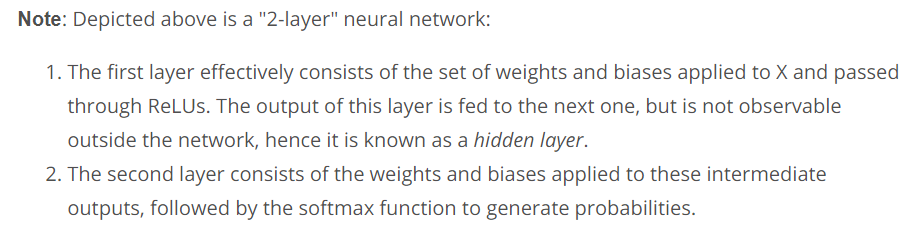
* Acum dorim sa luam logistic regression si sa il transformam intr-un non-liniar function.
* Ca sa realizez asta:
  + In loc sa am o singura matrice cu ajutorul careia inmultim, ca si clasificatorul nostru, vom insera un RELU Fix la mijloc si vom avea 2 matrice, unu care vine de la input si altu care e legat de output.
  + Vom avea un alt parametru care putem face tunning pe el si anume “H”, ce va corespunde cu numarul de unitati RELU ce il avem in clasificatorul nostru. Il putem face cat de mare dorim.
  + H = cate layere ai in RELU

2-Layer Neural Network

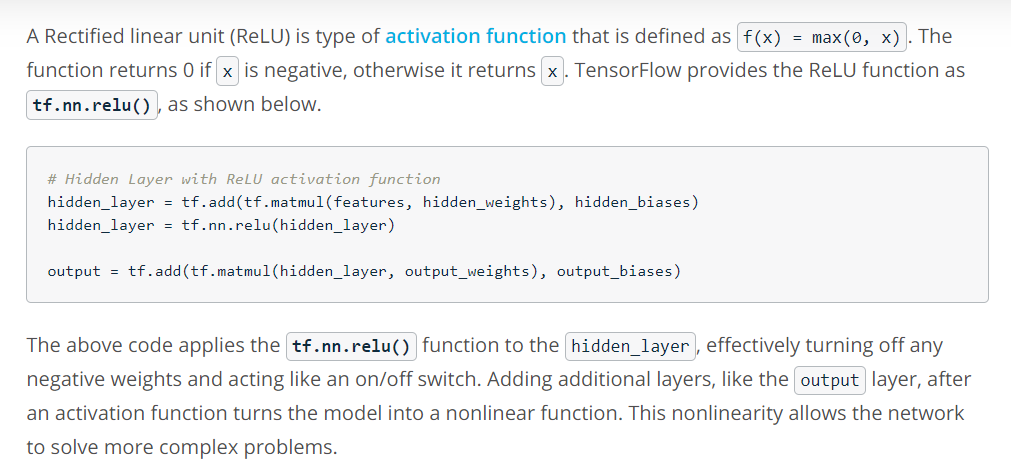


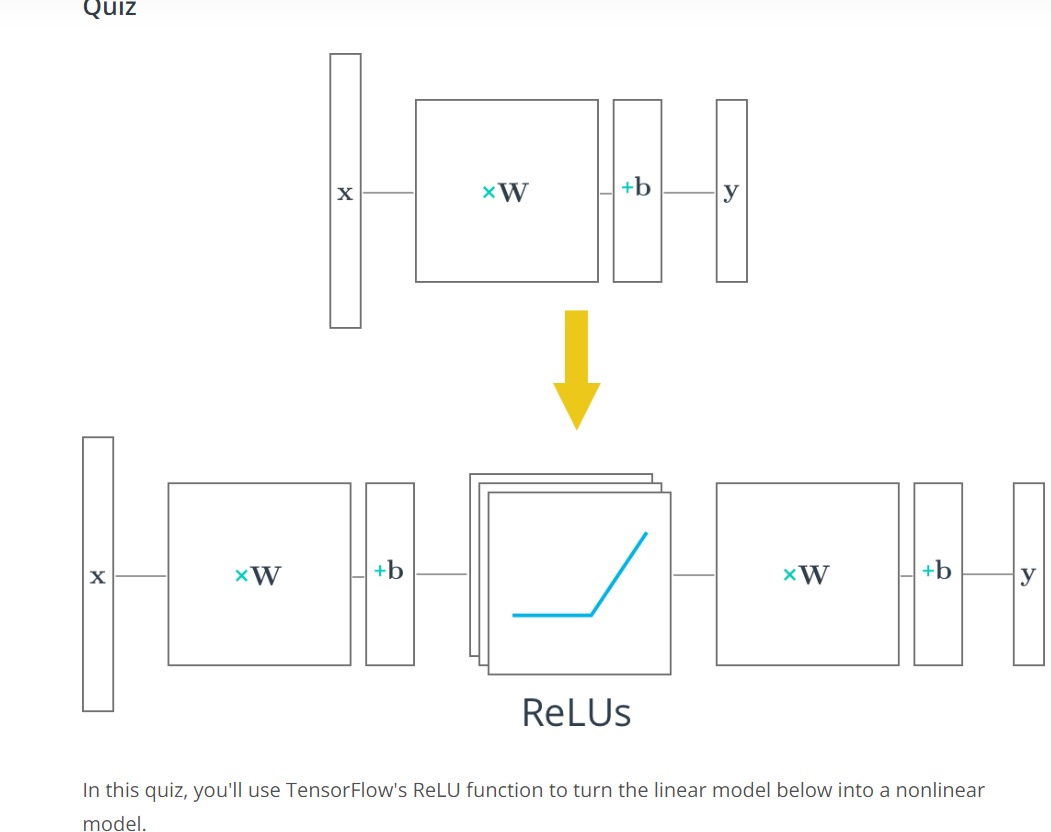
Se urmareste sa construim un multilayer neural network cu Tensorflow. Vom adauga:

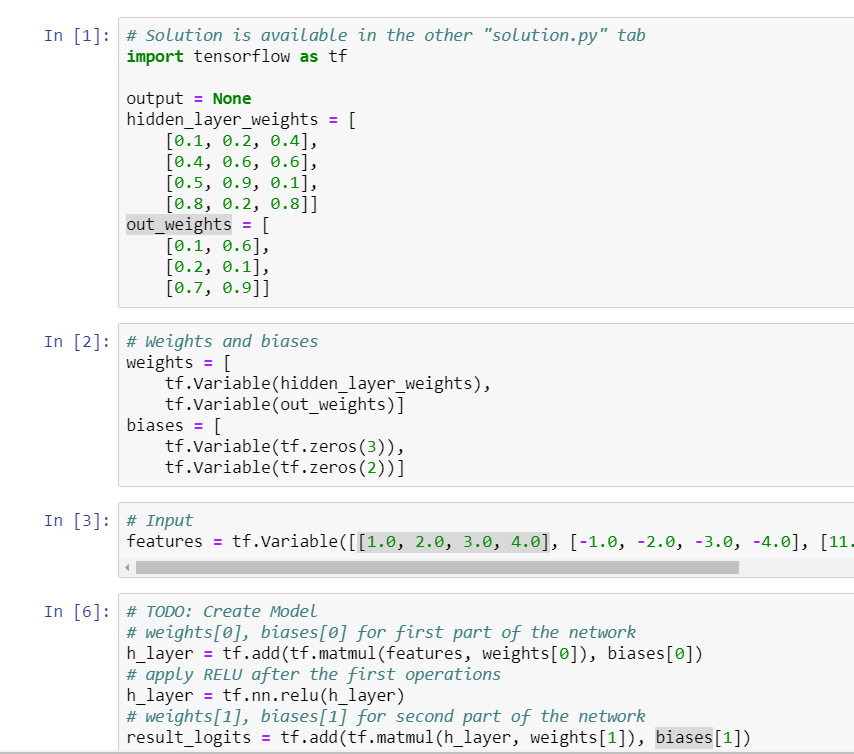
* Hidden layer = permite retelei sa modeleze functii mult mai complexe
* Non-Liniar activation asupra hidden layer = ne permite sa modelam non-liniar function.



Tensorflow RELU







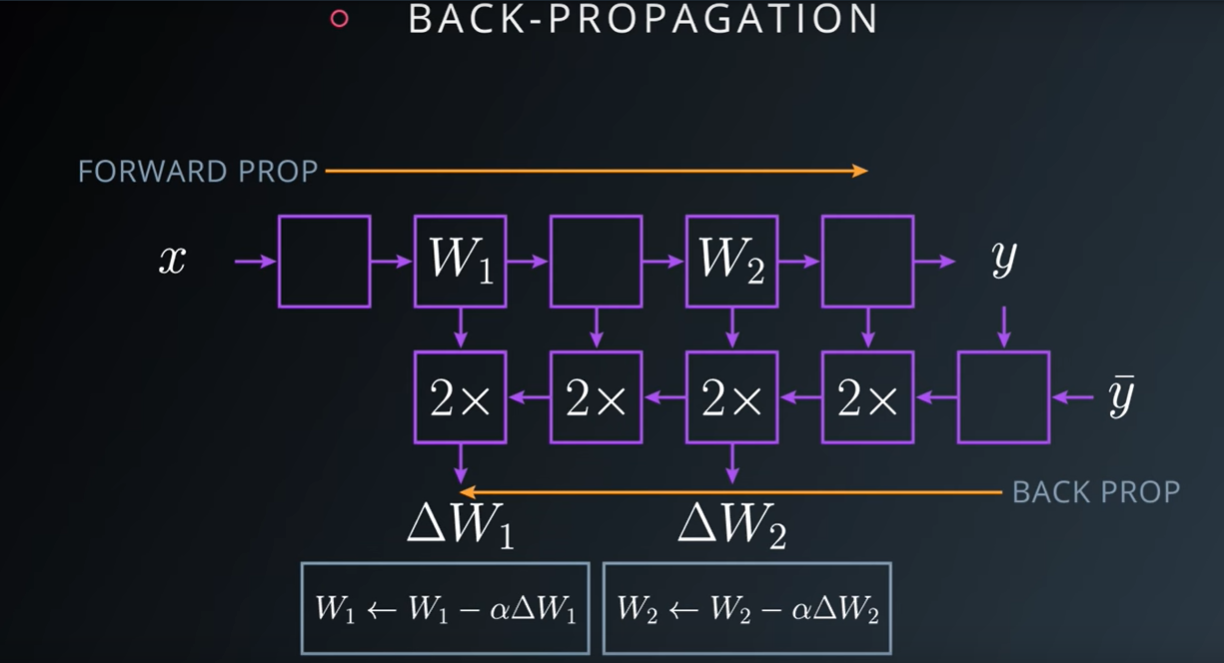


Chain Rule



* Ne spune ca exista o alta forma de reprezentare, a derivatelor unei functii, graphic adik.
* Link: <https://www.youtube.com/watch?v=DxOg_olir0k>

**BackPropagation**



**BACK-Propagation** = Ca sa creezi derivatele, se construieste un alt graf (layer2), ce este in sens invers (datele se transmit invers) si produce gradient. El poate fii derivat, complet individual din operatiile individuale din reteaua noastra.

* El face derivatele functiilor complexe, foarte eficiente, atata timp cat functia este realizata din blocuri simple cu derivate simple.

FORWARD PROP = modelul ce merge normal

BACK PROP = cel ce vine in sens invers

Deci pentru a rula Stocastic Gradient Descent, pentru fiecare batch micut de data, din training set, vom rula FORWARD PROP, iar apoi BACK PROP. Si asta, ne va da, gradient pentru fiecare weight din modelul nostru.

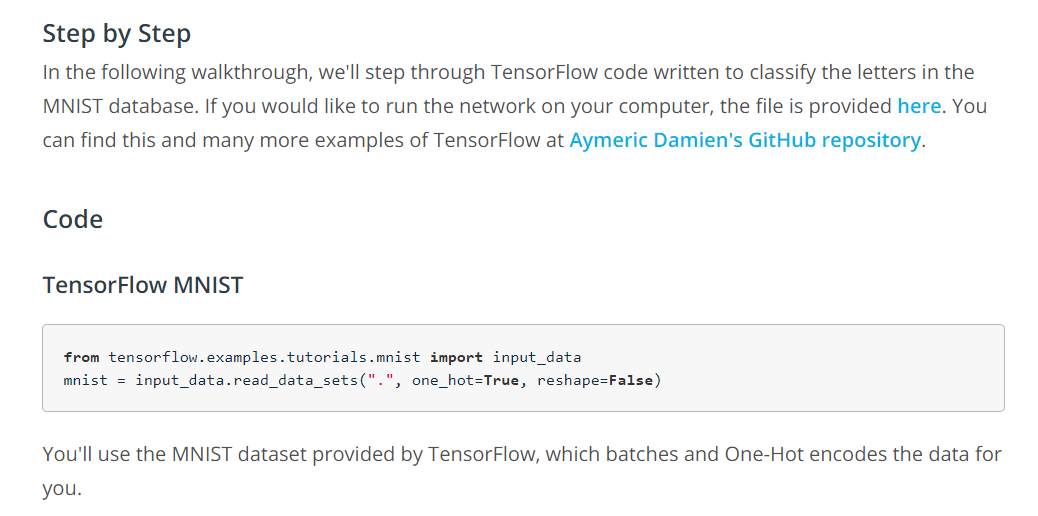
Apoi vom aplica acesti gradient cu learning rate la weight-urile originale si apoi le vom updata. Acest pas se va repeta, pana cand vom avea un model foarte foarte bun.

Ce trebuie retinut, e ca la Back-Propagation are nevoie de 2xori mai multa memorie decat la FORWARD PROP.

Asta este foarte important, atunci cand vrem sa dimensionam modelul si sa il potrivim in memoria noastra.

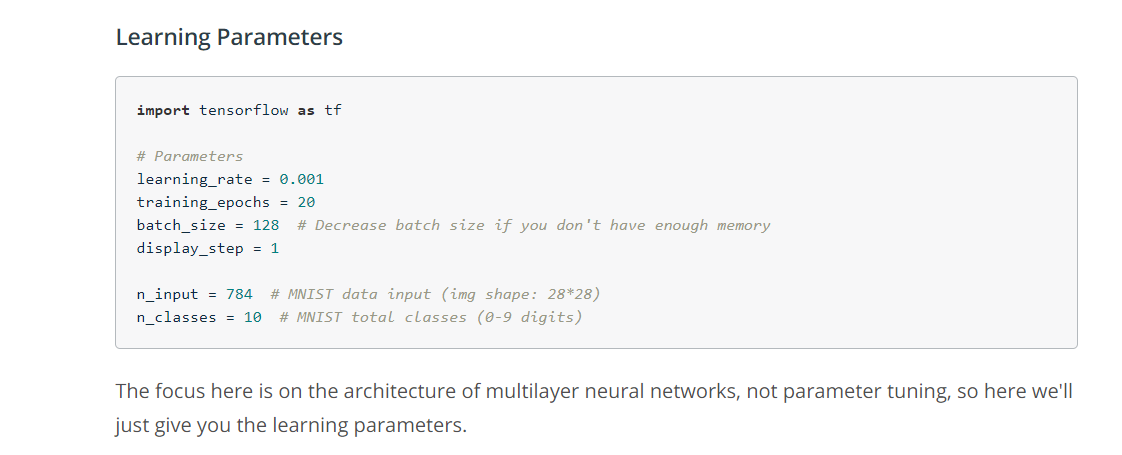
**Deep Neural Network in Tensorflow**

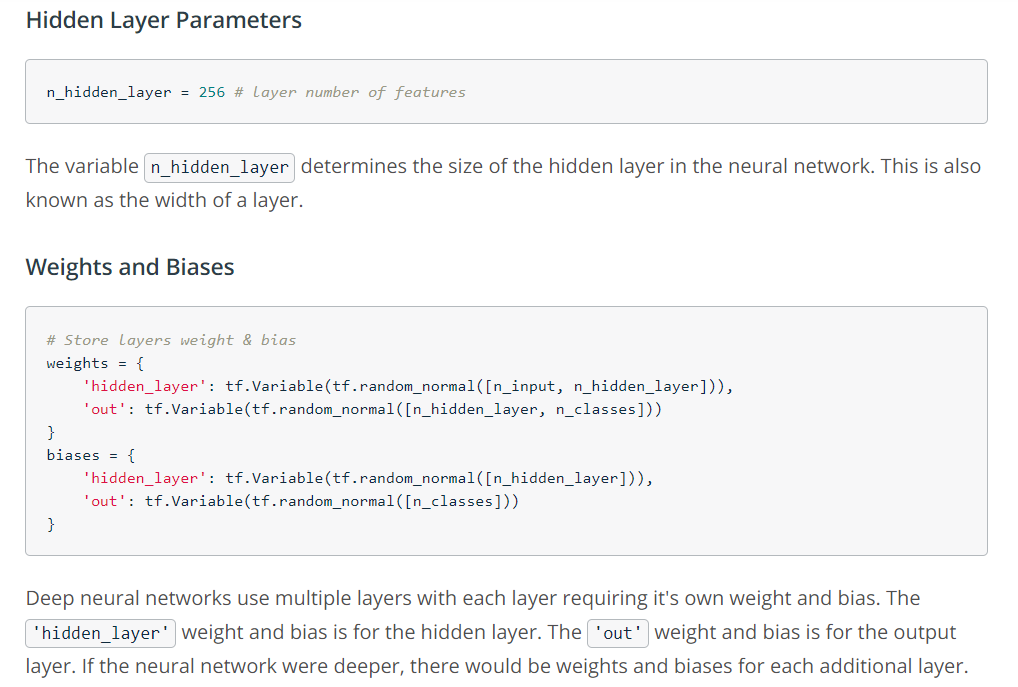
* In sectiunea asta vrem sa vedem cum folosim un logistic classifier pentru a construi un deep neural network.

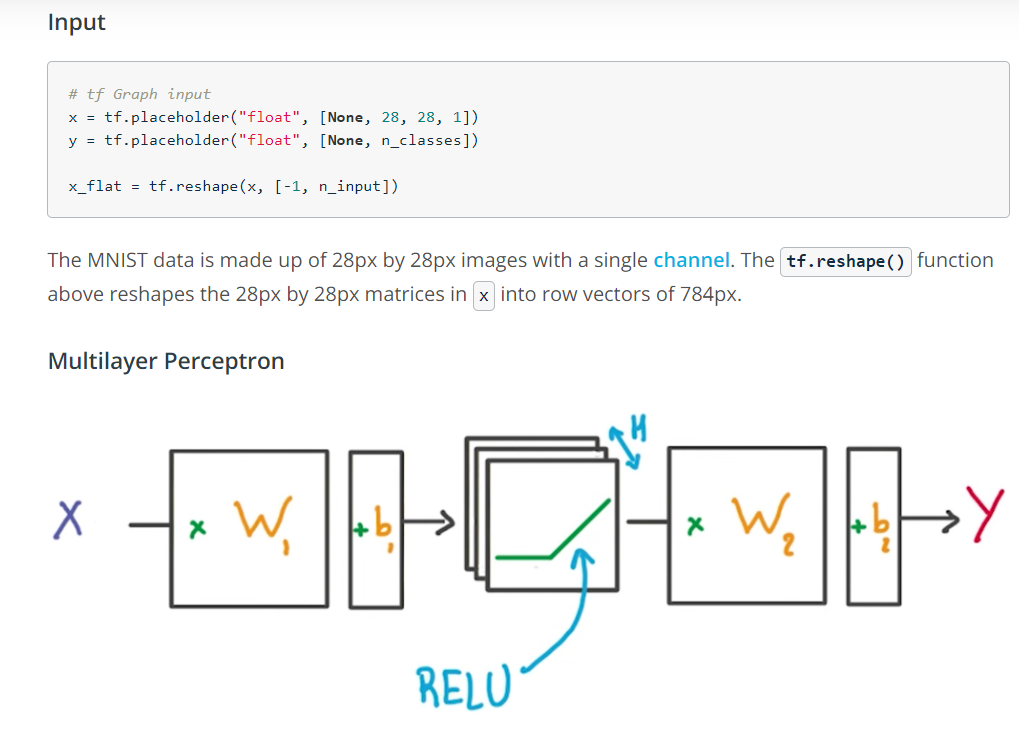


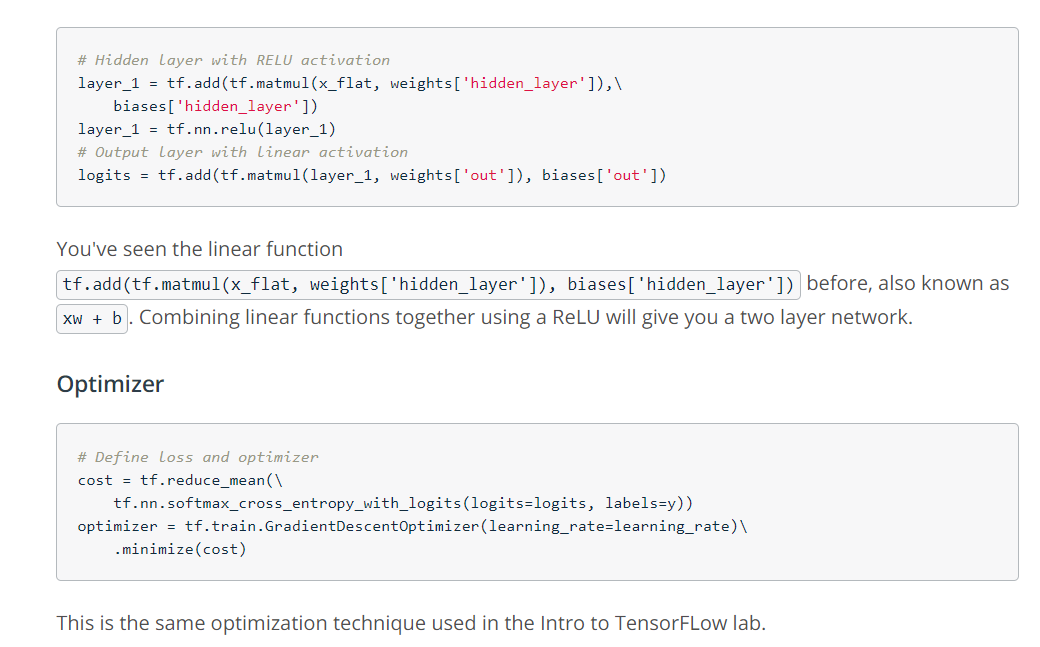
Linkurile: <https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/neural_network_raw.ipynb>

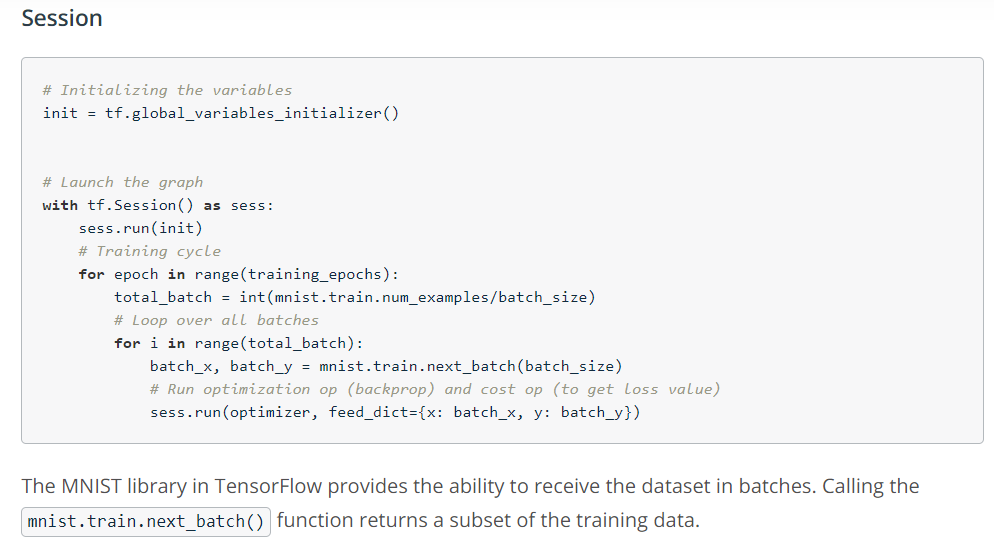
Mai multe exemple tensorflow: <https://github.com/aymericdamien/TensorFlow-Examples>

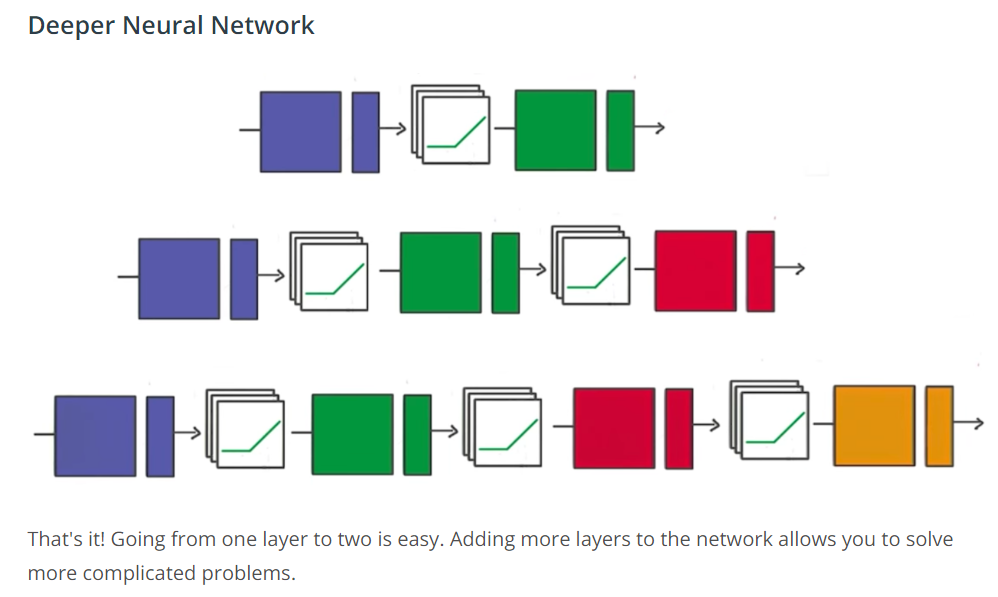








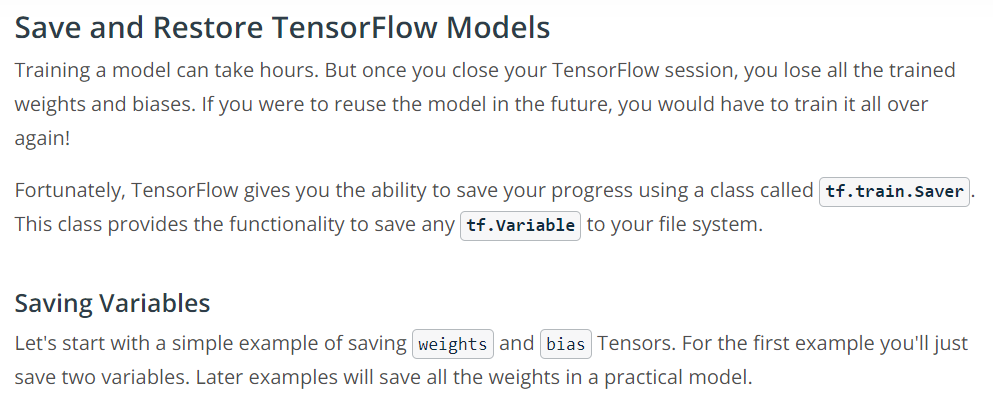


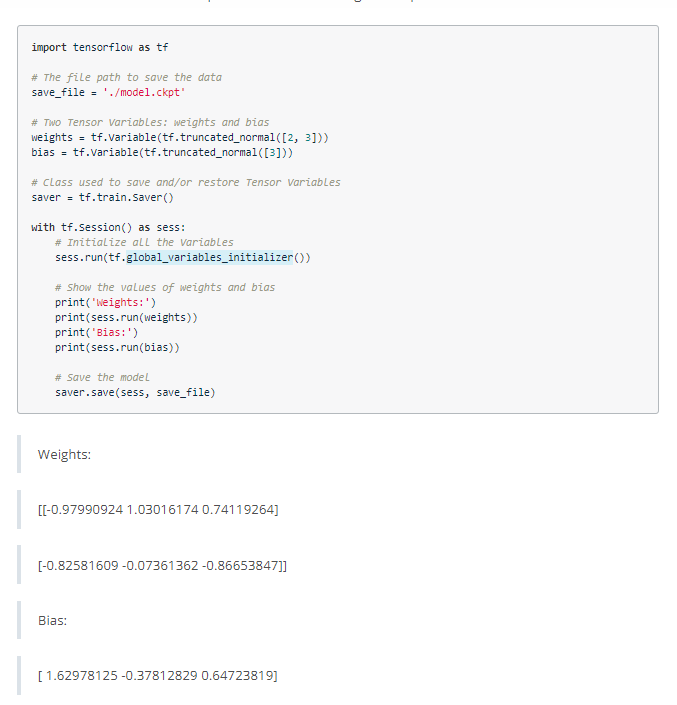


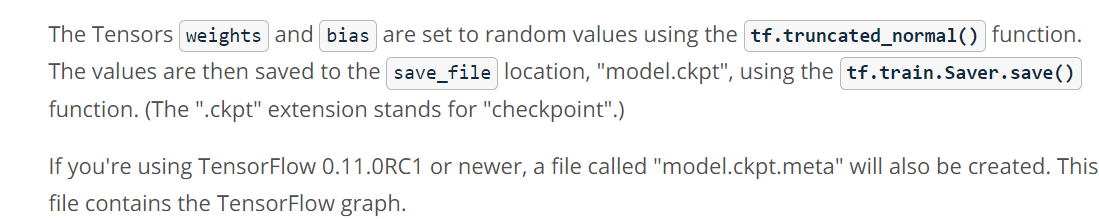
**Training a Deep Learning Network**

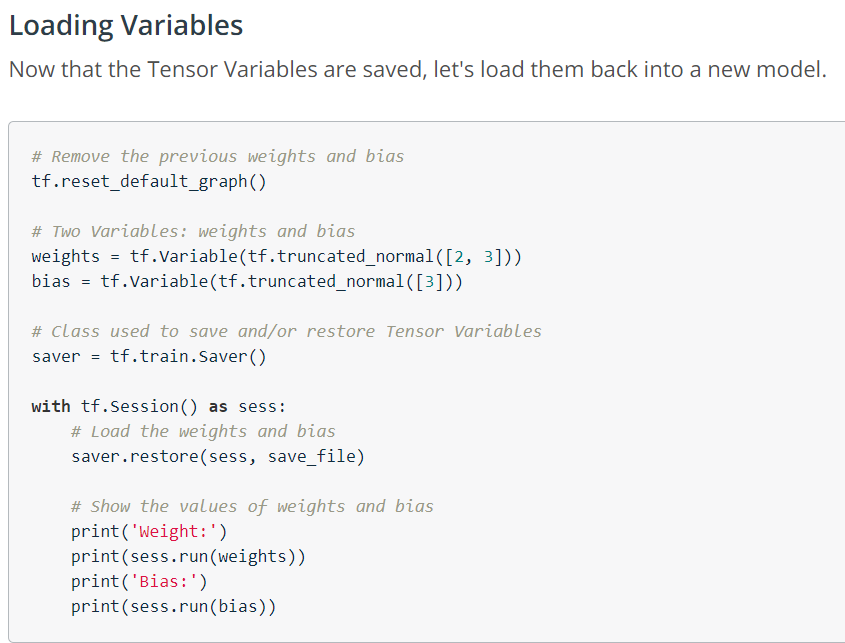
Link: <https://www.youtube.com/watch?v=CsB7yUtMJyk>

**Save and Restore Tensorflow Models**

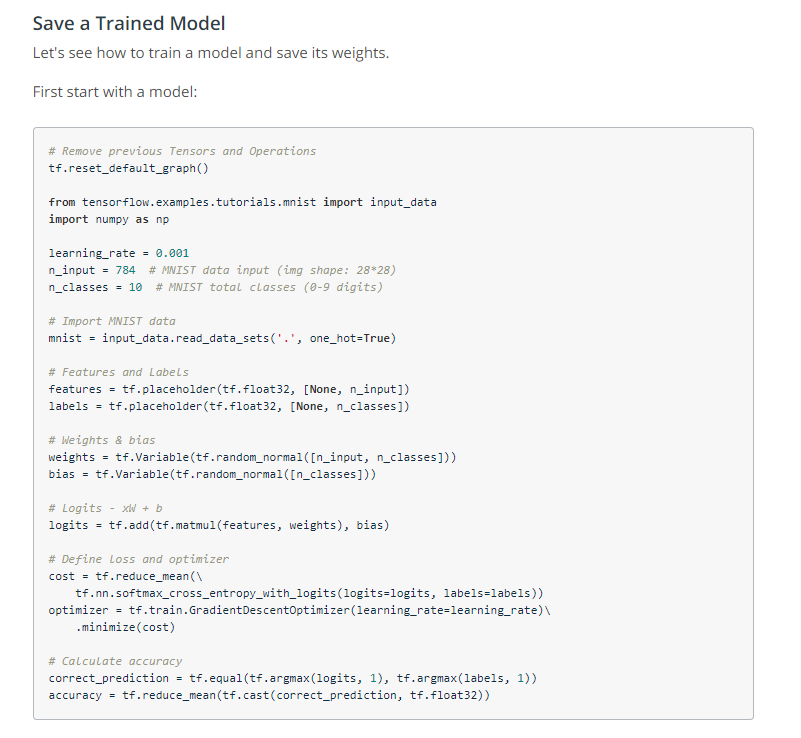




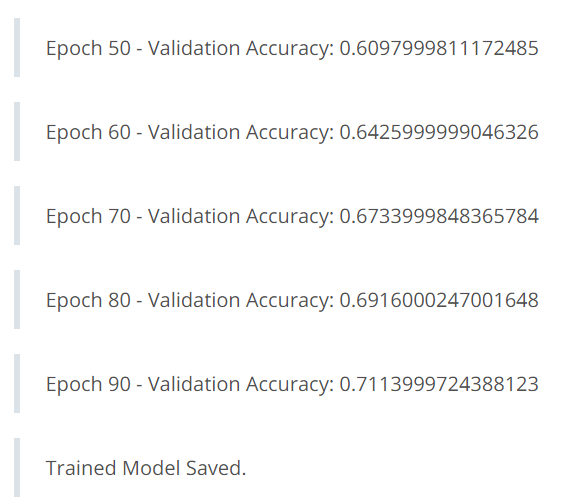


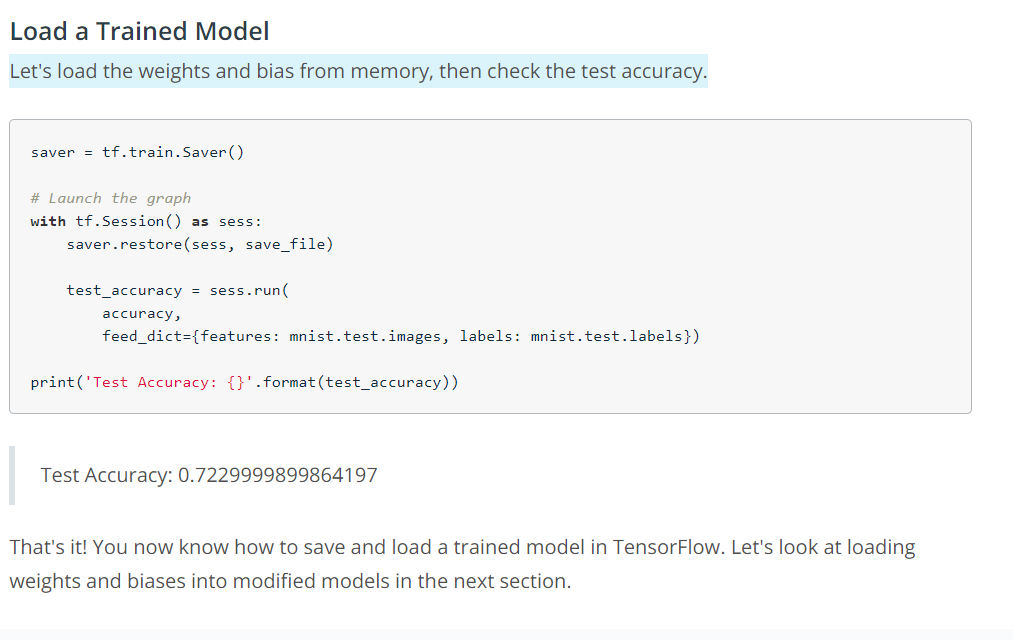




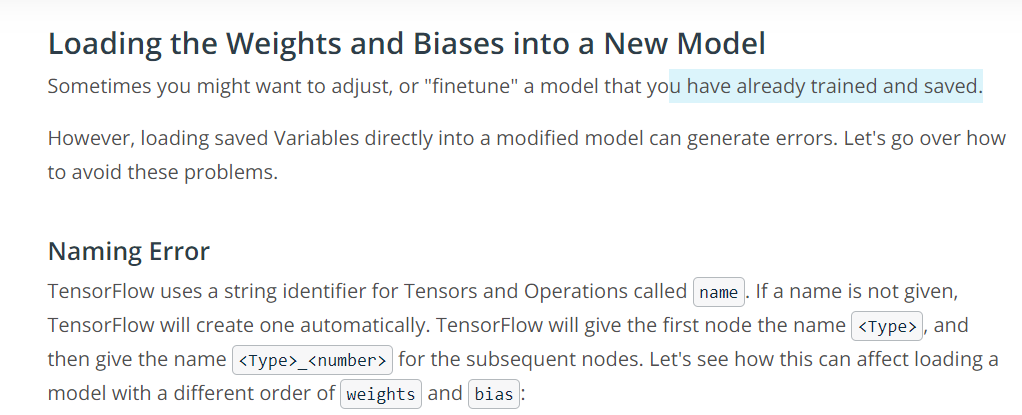




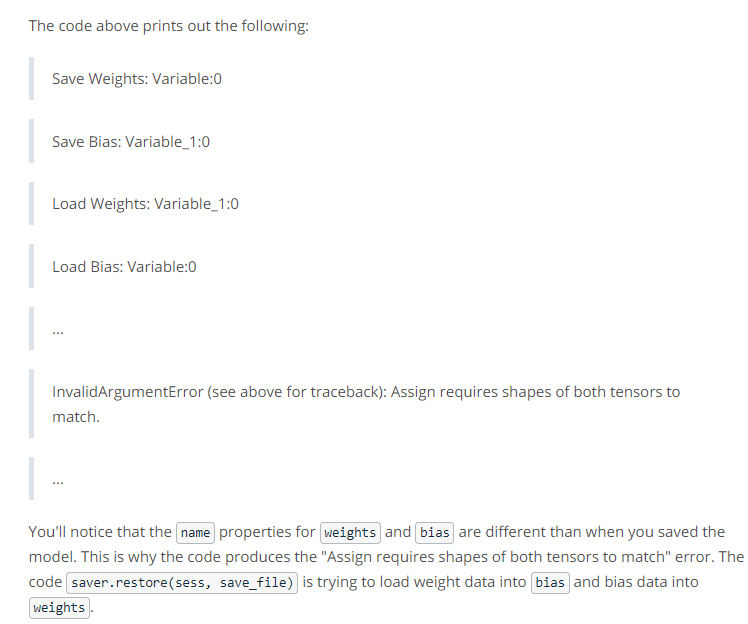




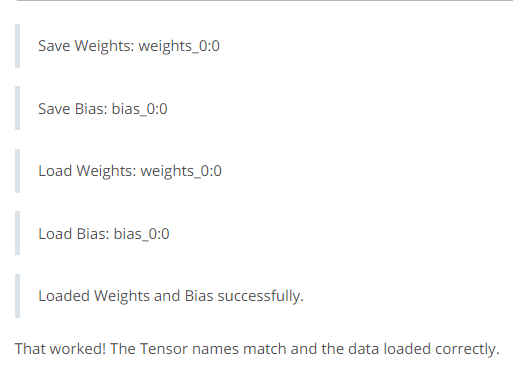
**FineTuning**











Regularization

Link: <https://www.youtube.com/watch?v=pECnr-5F3_Q>

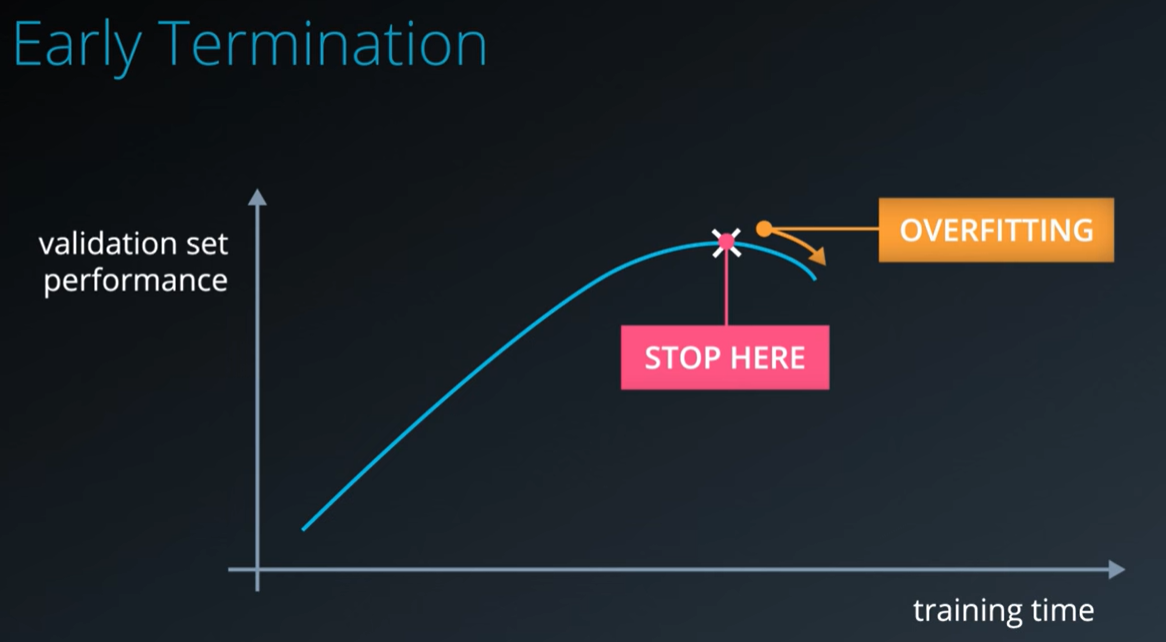
Morala: este ca avem nevoie de un dataset foarte mare

Link2: <https://www.youtube.com/watch?v=QcJBhbuCl5g>

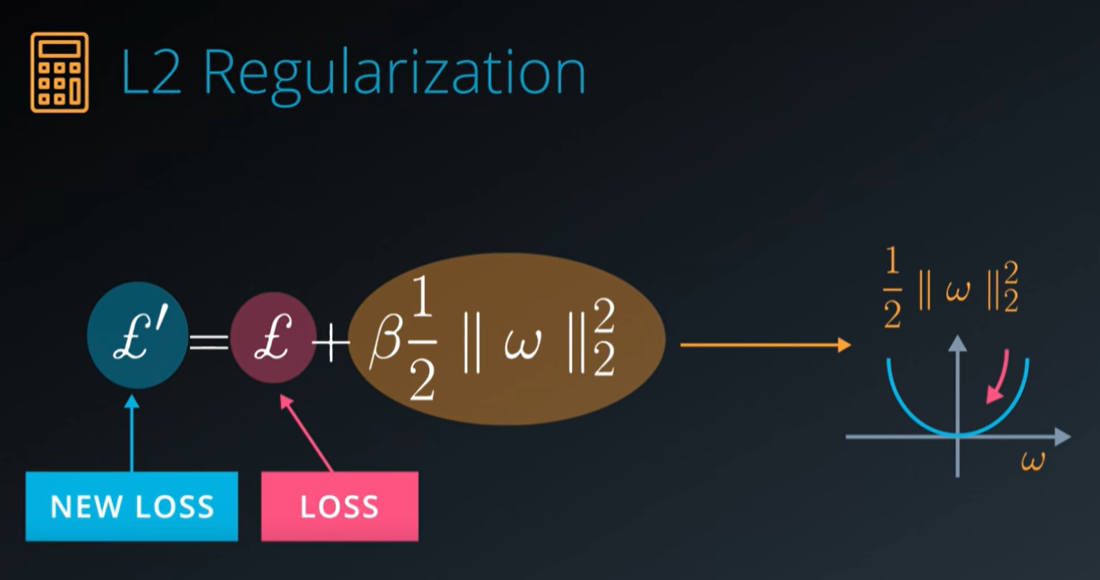
* Trb gasite metode de a combate OverFitting

Metode:

1. Este sa ne uitam la performanta setului nostrum de validare si sa oprim trainuirea atunci cand performanta la validare incepe sa scada. (“**Early termination**”)
2. Aplicam **Regularization**



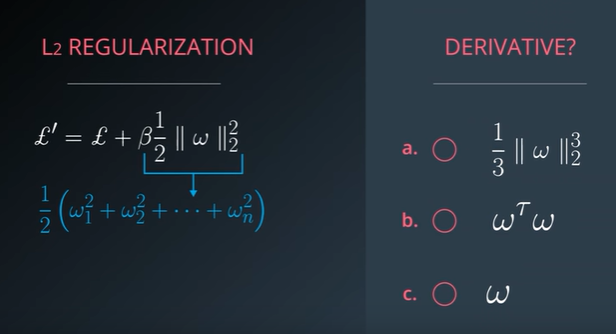
**Regularization** = metoda ce presupune aplicarea de constrangeri artificiale in reteaua noastra, care reduc implicit numarul de parametrii.



In Deep Learning numim asta: **L2 Regularization**

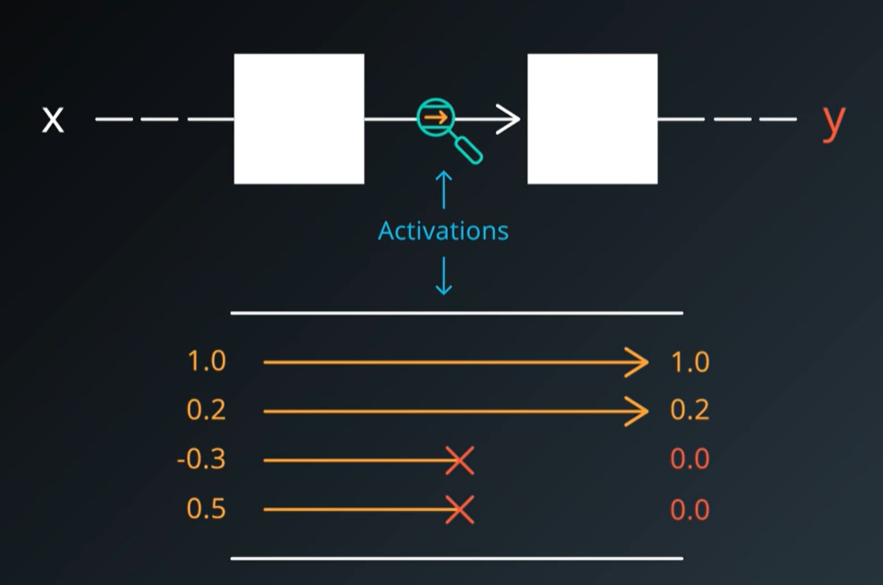
* Idea este sa adaugam un alt termen la **loss**, ce va penalize weight-urile foarte mari. Prin adaugarea normei L2 al weight-urilor la **loss** multiplicate de o constanta.

L2Regularization Link: <https://www.youtube.com/watch?v=E0eEW6V0_sA>



* Vrea sa zica, ca smekeria aia de acolo, este suma patratelor weighturilor noastre.

**DROPOUT**



**Link:** <https://www.youtube.com/watch?v=6DcImJS8uV8>

* Este o alta tehnica importanta pentru Regularization

**Activations** = sunt valorile ce vin de la un layer la altul.

* Algoritmul asta de DROPOUT presupune ca: in momentul in care transmit date de la un layer la altul (activations), iei random jumatate din toate valorile astea si le setezi cu 0.

Ideea: “Your network can never rely on any given activation to be present”

* Pare inefficient, dar in practica face lucrurile mai “robust” si previne overfitting.

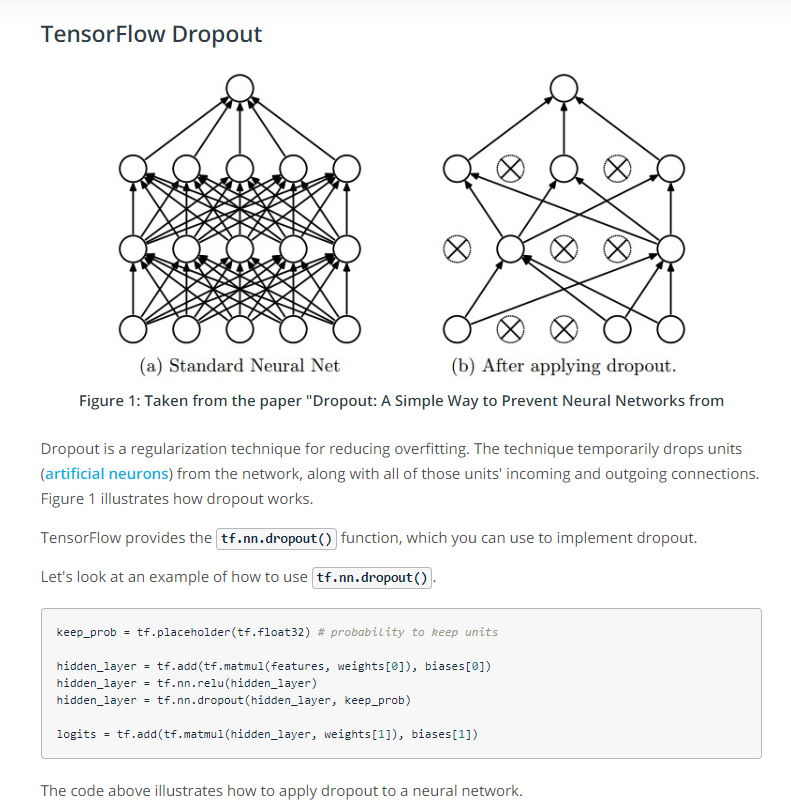
Daca DROPOUT nu functioneaza, sigur ar trebui sa folosesc o retea mai mare.



Link: <https://www.youtube.com/watch?v=8nG8zzJMbZw>

* Cand vrem sa evaluam, calculam media la toate yt adik E(yt)
* Dar schema e, ca in momentul cand faci average, inmultesti activationurile care nu sunt facute de tine 0, cu 2

**Tensorflow DROPOUT**



Artificial neuron: <https://en.wikipedia.org/wiki/Artificial_neuron>

