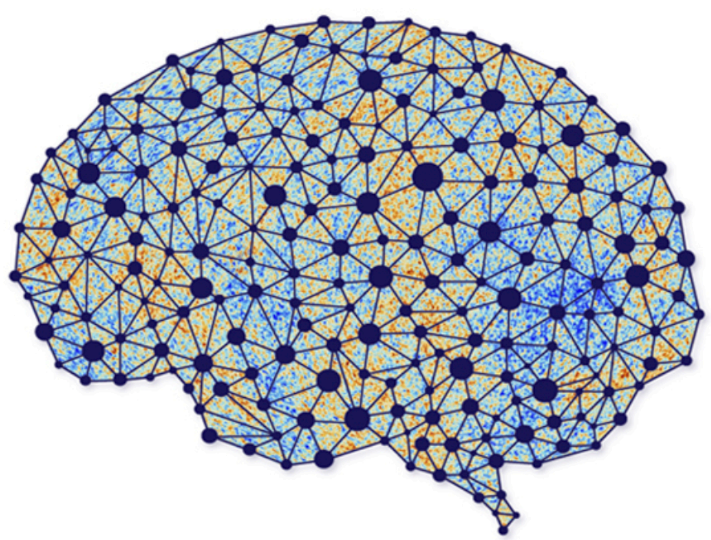
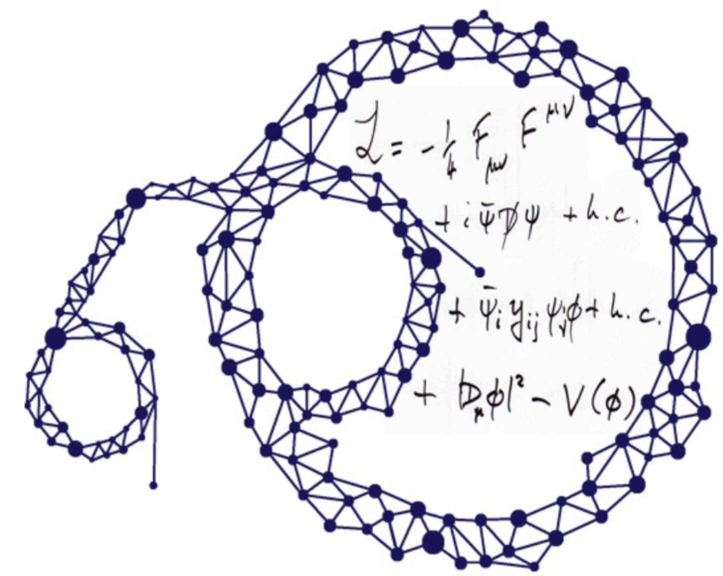
Physics 361 - Machine Learning in Physics

Lecture 6 – Basics of Machine Learning

Feb. 8th 2024



AI
∩
Universe



Moritz Münchmeyer

Unit 2: Machine Learning Basics

2.2 Neural Network Basics (Supervised Learning)

Supervised machine Learning

We want to learn a complicated non-linear function to map input $\vec{x}$ to output $\vec{y}$.

$$\vec{x} \longrightarrow \boxed{\text{e.g. neural } f_\theta(\vec{x}) \text{ network}} \longrightarrow \vec{y}$$

θ: model parameters = weights

We want to find the model parameters by minimizing some loss function on our training data.

E.g. $$L^{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left(y_i - f_\theta(\vec{x}_i)\right)^2$$

A very simple neural network

- We need some way to specify the function $f_\theta(\vec{x})$. It turns out that a very simple „architecture" work very well in practice:

  Linear transformations combined with element-wise non-linearities

- A linear transformation / linear layer* is given by $\vec{y} = \vec{f}(\vec{x}) = W\vec{x} + \vec{b}$

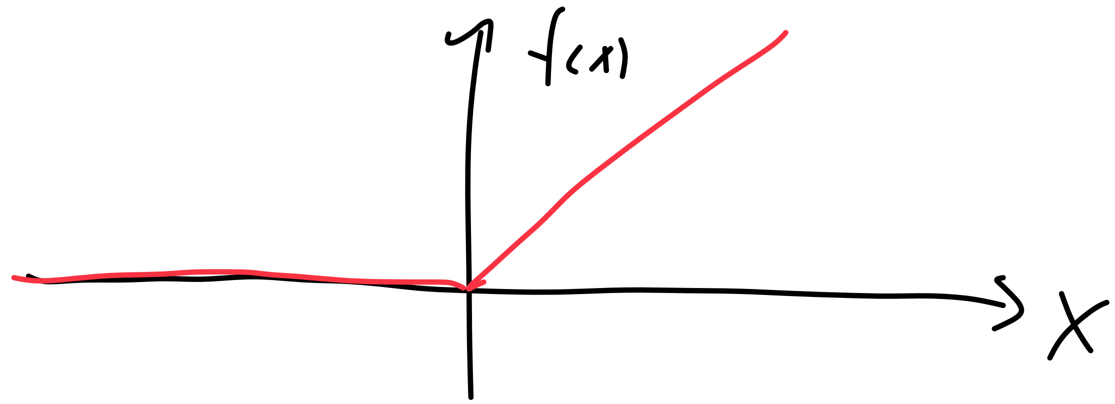
  $\vec{x}$: N_{dim}
  $\vec{y}$: M_{dim}

  W: matrix of weights
  $\vec{b}$: bias

- The most common non-linearity / activation function is the ReLU „rectified linear unit"

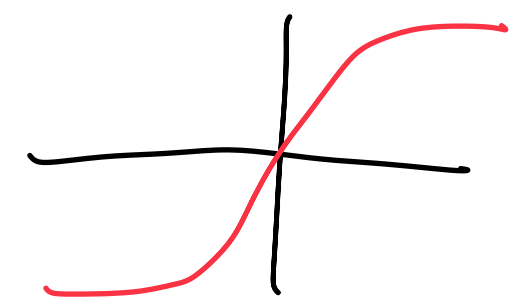
* fully connected linear layer

- ReLU is a 1-dimensional function



Piecewise linear function.

Alteratives : tanh , sigmoid



- A multilayer perceptron (MLP) is a „deep neural network" made out of a stack of the 2 building blocks.

$$\vec{x} \longrightarrow \boxed{\text{Linear}} \longrightarrow \boxed{\text{reLU}} \longrightarrow \boxed{\text{Linear}} \longrightarrow \boxed{\text{reLU}} \text{ -- . . . . } \longrightarrow \vec{y}$$

The combination of ReLU and Linear Layer can be written as

$$f^i(x) = \underbrace{\max(0,}_{\text{ReLU}} W^i_j x^j + b^i)$$

component notation

next Layer uses $f(x)$ as x input.

If we stack these layers:

$$\max\left(0, W^i_{n,j} \max\left(0, W^j_{n-1,k} \max\left(0, \cdots\right) + b^j_{n-1}\right) + b^i_n\right)$$

Now we have a function that we can „fit" to the training data.

Loss for classification

For classification we want the NN output to be the probabilities of the various classes.

$$\vec{x} \longrightarrow \boxed{\text{MLP}} \longrightarrow y_j \xrightarrow{\text{softmax}} P_j(\vec{x})$$

e.g image

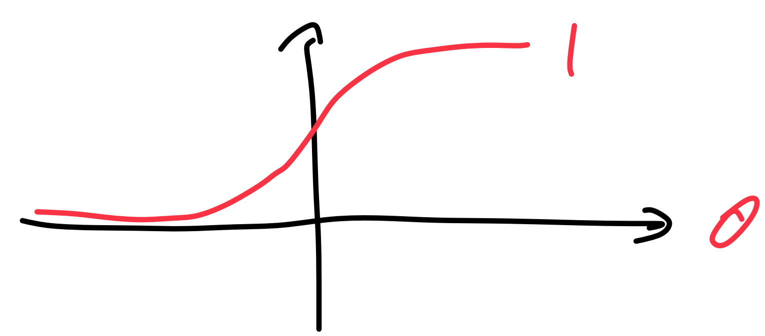
„logits"

probability for $\vec{x}$ to be in class j.

A probability distribution need to have only positive probabilities and they must sum to 1.

To achieve this we need the softmax function

$$P_j(y_j) = \frac{e^{y_j}}{\sum_i e^{y_i}}$$



The typical loss function for classification is the negative log likelihood = cross-entropy:

predicted by NN

$$L(\theta) = -\sum_{i=1}^{N_{examples}} \sum_{j=1}^{N_{classes}} y_j^i \log p_j(\vec{x}_i)$$

parameters of NN: W, b

$$y^i = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

one-hot vector which is 1 for the right class.

Optimization of the parameters

- The last element we need is the „optimization procedure" to find good weights θ^* from minimizing the loss function
- Almost all algorithms used in practice to do this are versions of stochastic gradient descent.

$$\theta_{n+1} = \theta_n - \varepsilon \nabla_\theta L(\theta_n ; X)$$

new parameters at step $n+1$

current parameters

learning rate

gradient of the loss function

- stochastic means we evaluate the gradient at each step using only a subset of the training data, called the „minibatch"

In practice, to calculate this gradient we use "auto-differentiation".

We'll discuss optimization and auto-differen-tiation in more detail soon.

Unit 2: Machine Learning Basics

2.4 Training our first model

Train an MLP on SUSY data

- We will use the simulated collider data. **The goal will be to discriminate SUSY (supersymmetry) events from non-SUSY events.** The data is from the paper **https://www.nature.com/articles/ncomms5308.**

- We use a modified version of the code from https://physics.bu.edu/~pankajm/MLnotebooks.html which was written for the review https://arxiv.org/abs/1803.08823 **A high-bias, low-variance introduction to Machine Learning for physicists.**

- We will use **python** with the **pytorch** framework (it does the auto-differentiation). Pytorch is the most widely used tool in ML research (followed probably by JAX now).

- We will use **Google Colab** to run the python code. Colab provides free computational resources, including GPUs, with a nice interface based on python's Jupyter.

- I **uploaded a basic python tutorial to canvas in files/python/python tutorial** (from a previous class, Physics 249). You will have to self study some python if you don't have prior experience.

Physics Background

- Using the dataset from the UC Irvine ML repository produced by MC simulations to contain events with 2 leptons (electrons or muons)
- These events with 2 leptons with large p_T can occur in SUSY models or within the SM.
- 18 kinematic variables (“features”) are recorded for each event.
- We train a MLP classifier to classify the events into SUSY or SM background.

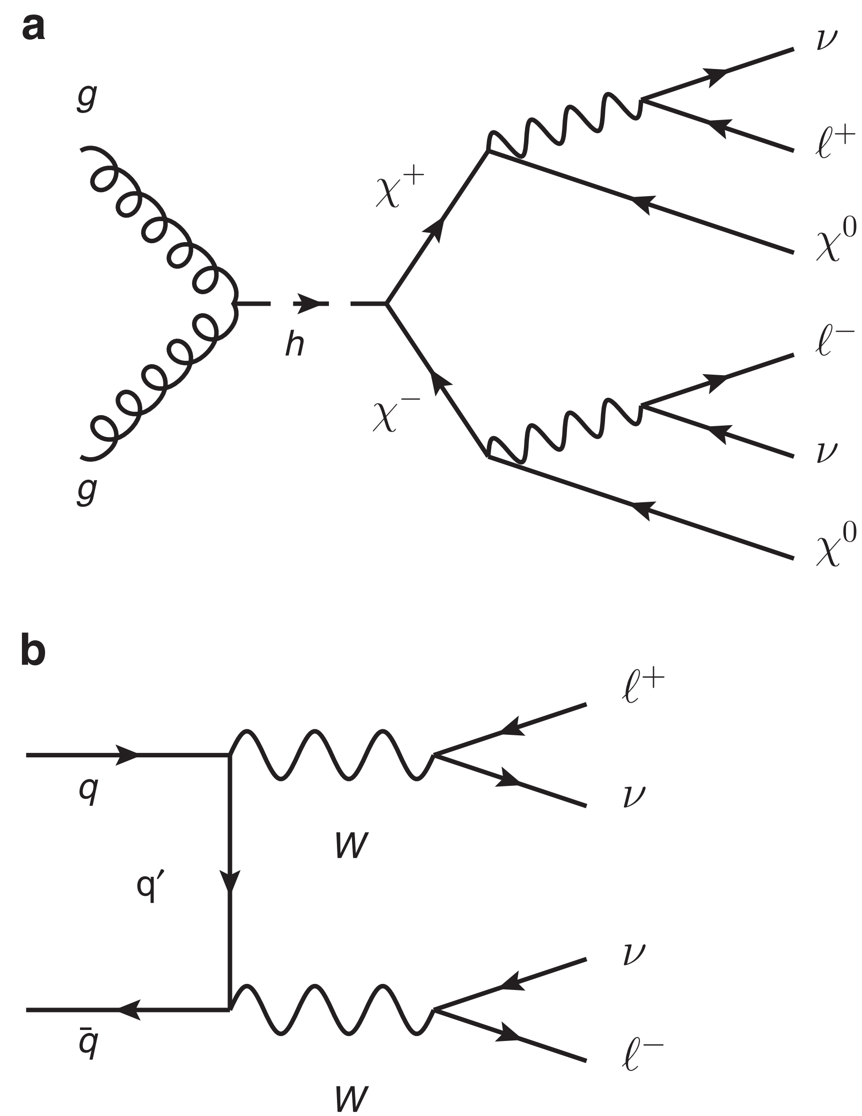
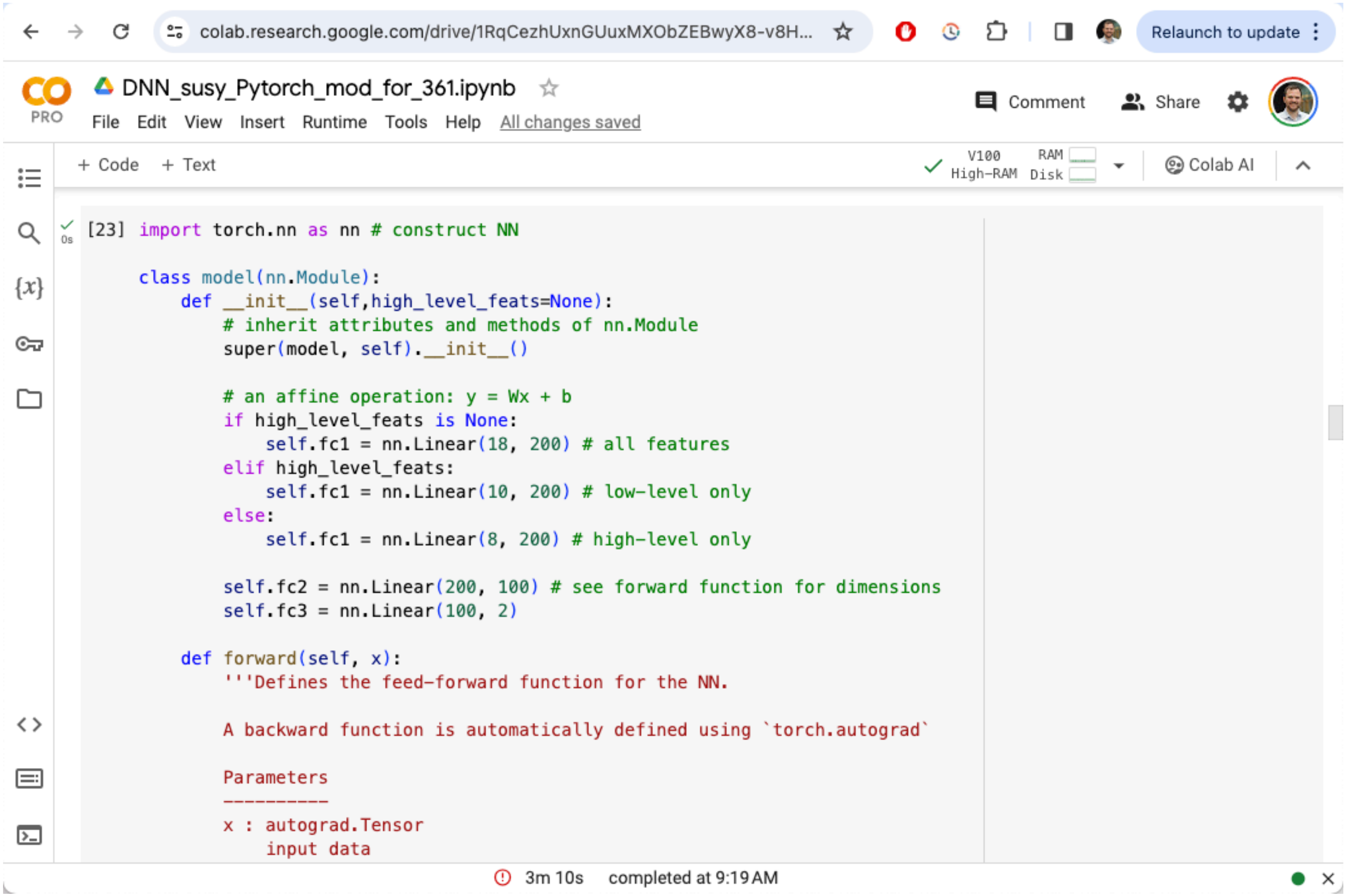


Figure 4 | Diagrams for SUSY benchmark. Example diagrams describing the signal process involving hypothetical supersymmetric particles $\chi^{\pm}$ and χ^0 along with charged leptons $\ell^{\pm}$ and neutrinos ν (**a**) and the background process involving W bosons (**b**). In both cases, the resulting observed particles are two charged leptons, as neutrinos and χ^0 escape undetected.

Baldi et al, Nature Communications, Volume 5, Article number: 4308 (2014)

The rest of this section will be presented in Colab. I will upload the Colab notebook on the course page, and you can download it from there, upload it to Colab, and run it yourself.



Course logistics

- **Reading for this lecture:**
  - **For example:** Deeplearningbook.org chapter 6.

- **Problem set**: First problem set to appear tomorrow.