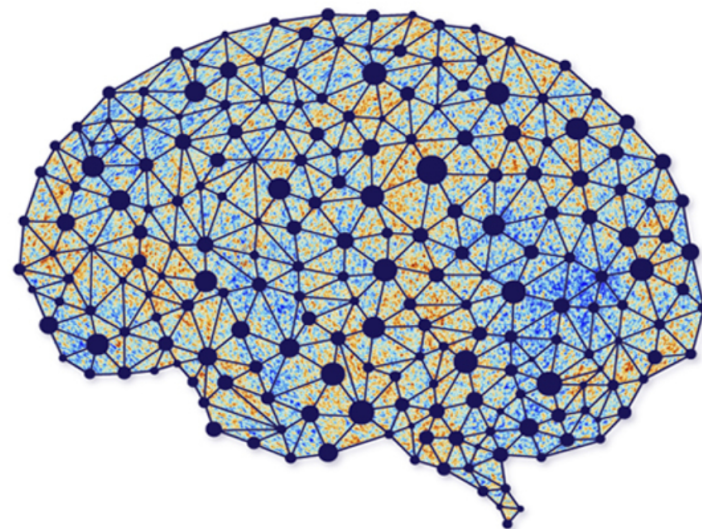


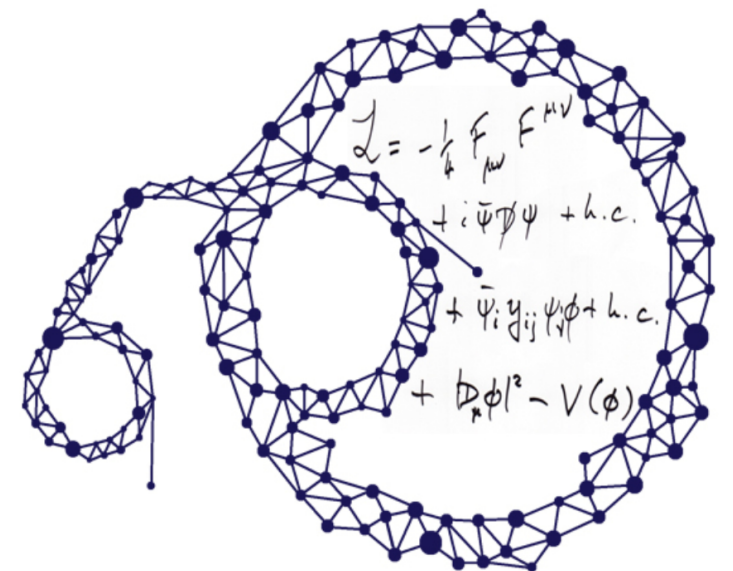
Physics 361 - Machine Learning in Physics

Lecture 5 – Basics of Machine Learning

Feb. 3rd 2025



AI
∩
Universe



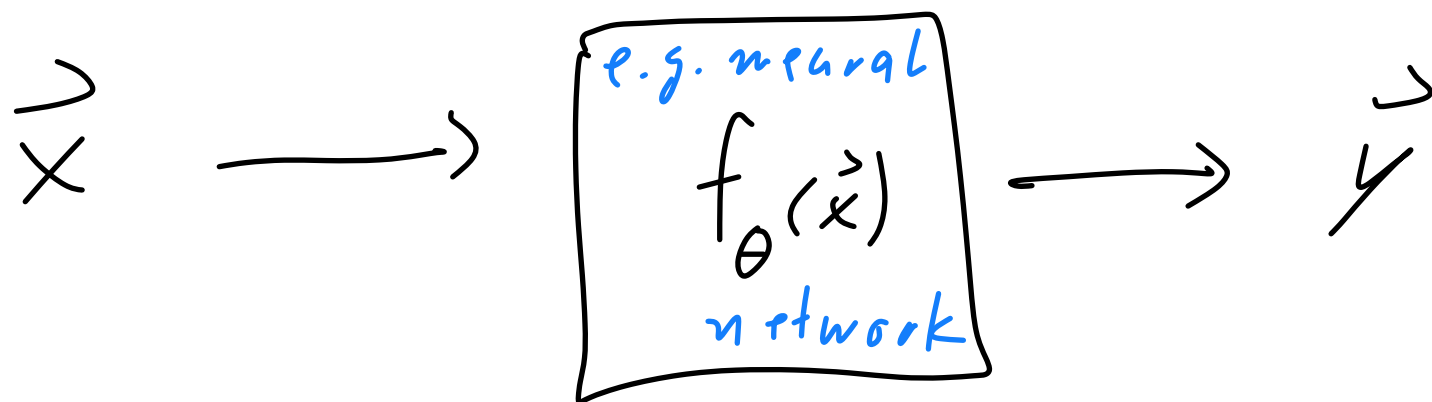
Moritz Münchmeyer

Unit 2: Machine Learning Basics

2.2 Neural Network Basics (cont.)

Supervised machine Learning

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



θ : model parameters = weights

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

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

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" works very well in practice:

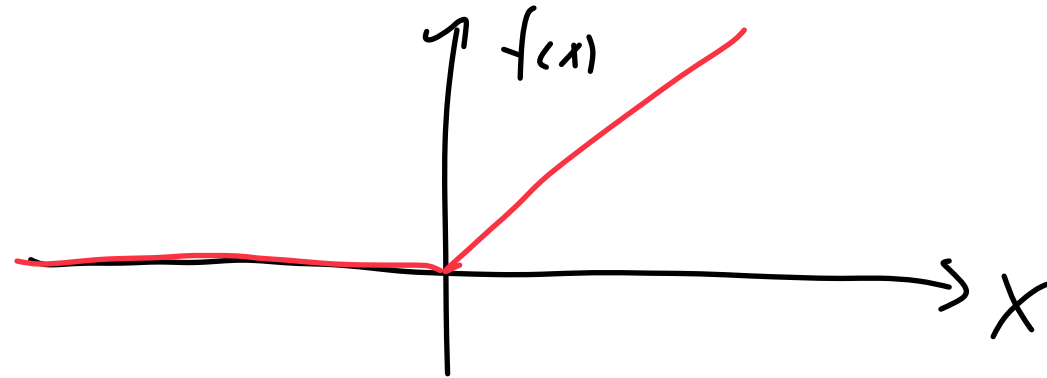
Linear transformations combined with element-wise non-linearities

- A Linear transformation / Linear Layer* is given by $\vec{y} = f(\vec{x}) = W\vec{x} + \vec{b}$
 $\vec{x} : N \text{ dim}$
 $\vec{y} : M \text{ dim}$
 $\uparrow$ matrix of weights
 $\uparrow$ bias

- The most common non-linearity / activation function is the ReLU "rectified Linear unit".

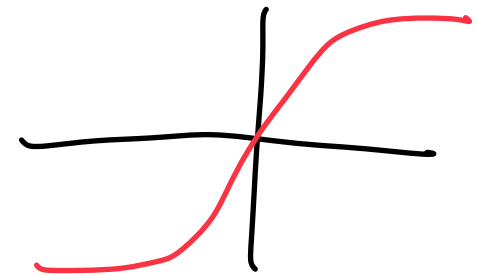
* fully connected Linear Layer

- ReLU is a 1-dimensional function

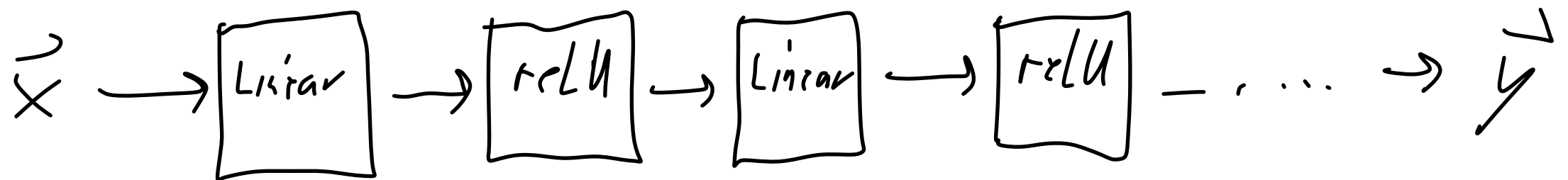


Piecewise linear function.

Alternatives: tanh, sigmoid



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



The combination of ReLU and Linear Layer
can be written as

$$f^i(x) = \max(0, W_{ij}^i x^j + b^i)$$

ReLU

component notation

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

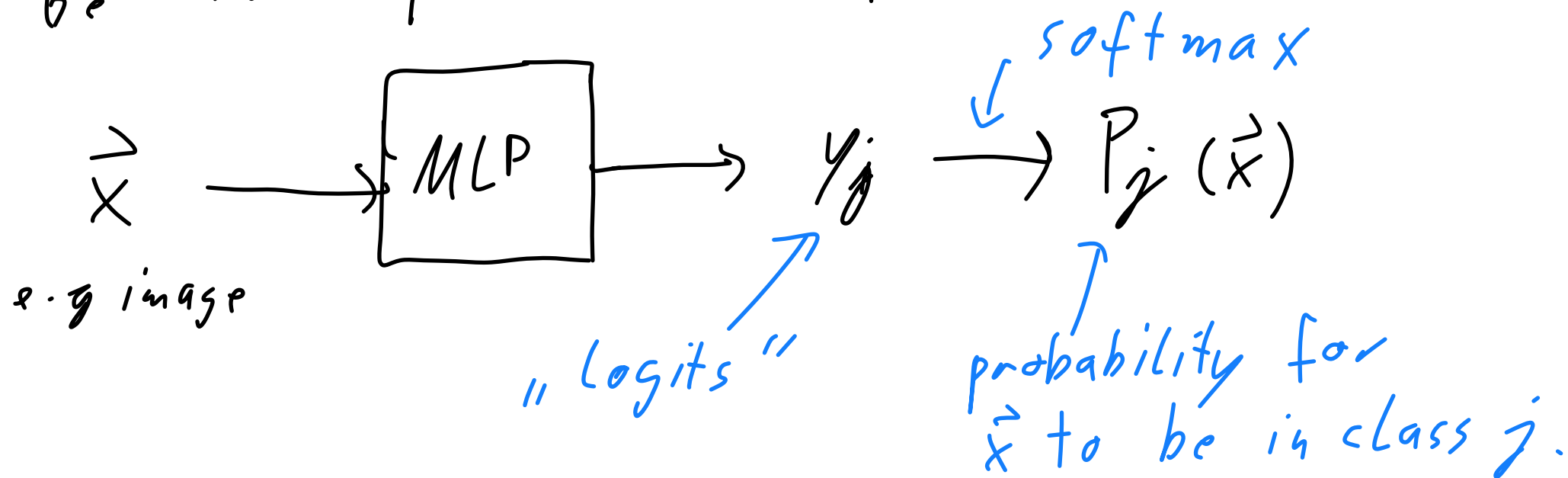
If we stack these layers:

$$\max(0, W_{n,j}^i \max(0, W_{n-1,k}^j \max(0, \dots) + b_{n-1}^j) + b_n^i)$$

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

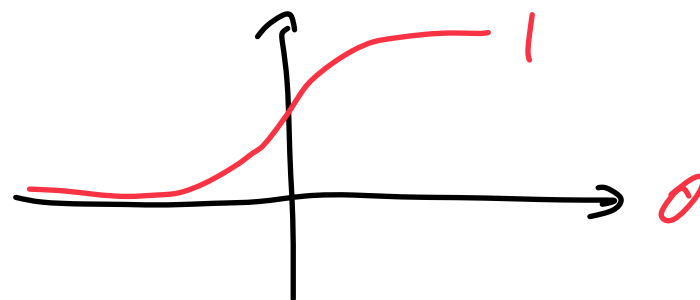
Loss for classification

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



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:

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

$\uparrow$
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.

predicted by NN

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**.

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

Diagram illustrating the stochastic gradient descent update rule:

- θ_{n+1} : new parameters at step $n+1$
- θ_n : current parameters
- $\sum$: learning rate
- ∇_{θ} : gradient of the loss function
- $L(\theta_n; x)$: loss function evaluated at θ_n for a specific data point x

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

Algorithmic differentiation

For gradient descent we need the gradient.

Two approaches that generally don't work well:

- symbolic differentiation.
gives you a (very long) symbolic expression. Not very scalable.
- Finite differences

$$f'(x) = \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

With finite numerical precision this is very noisy, especially when f is very large or very small.

- Instead: Use Autodifferentiation

Auto-differentiation

- Split function $X \rightarrow Y$ into elementary building blocks. "Computation graph"
- Differentiate the building blocks analytically.
- Evaluate the building block derivatives numerically.
- Propagate through the function using the chain rule ("backpropagation algorithm")

$$f(g_i(\theta_j)) \longrightarrow \nabla_{\theta_j} f = \sum_i \frac{\partial f}{\partial g_i} \times \frac{\partial g_i}{\partial \theta_j}$$

This is what frameworks like **PyTorch** implement. We'll skip the details for now since we don't immediately need them.

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.

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.

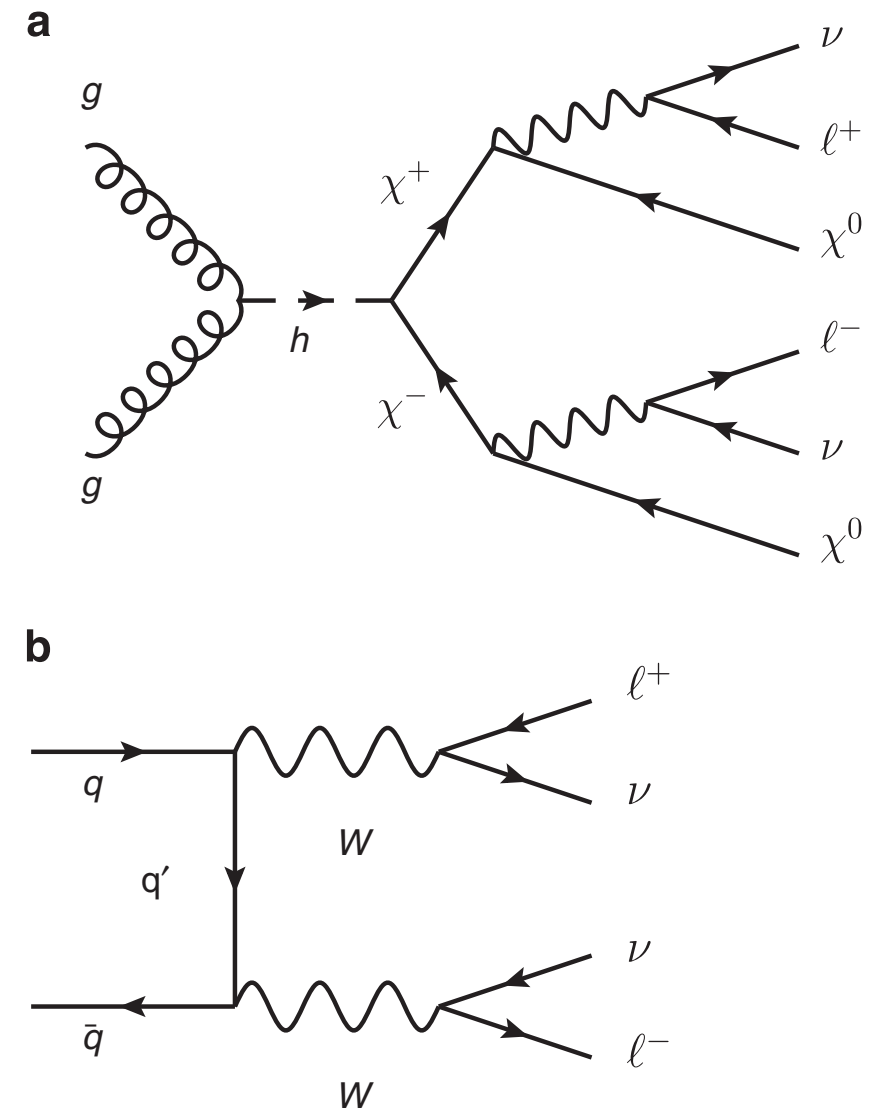
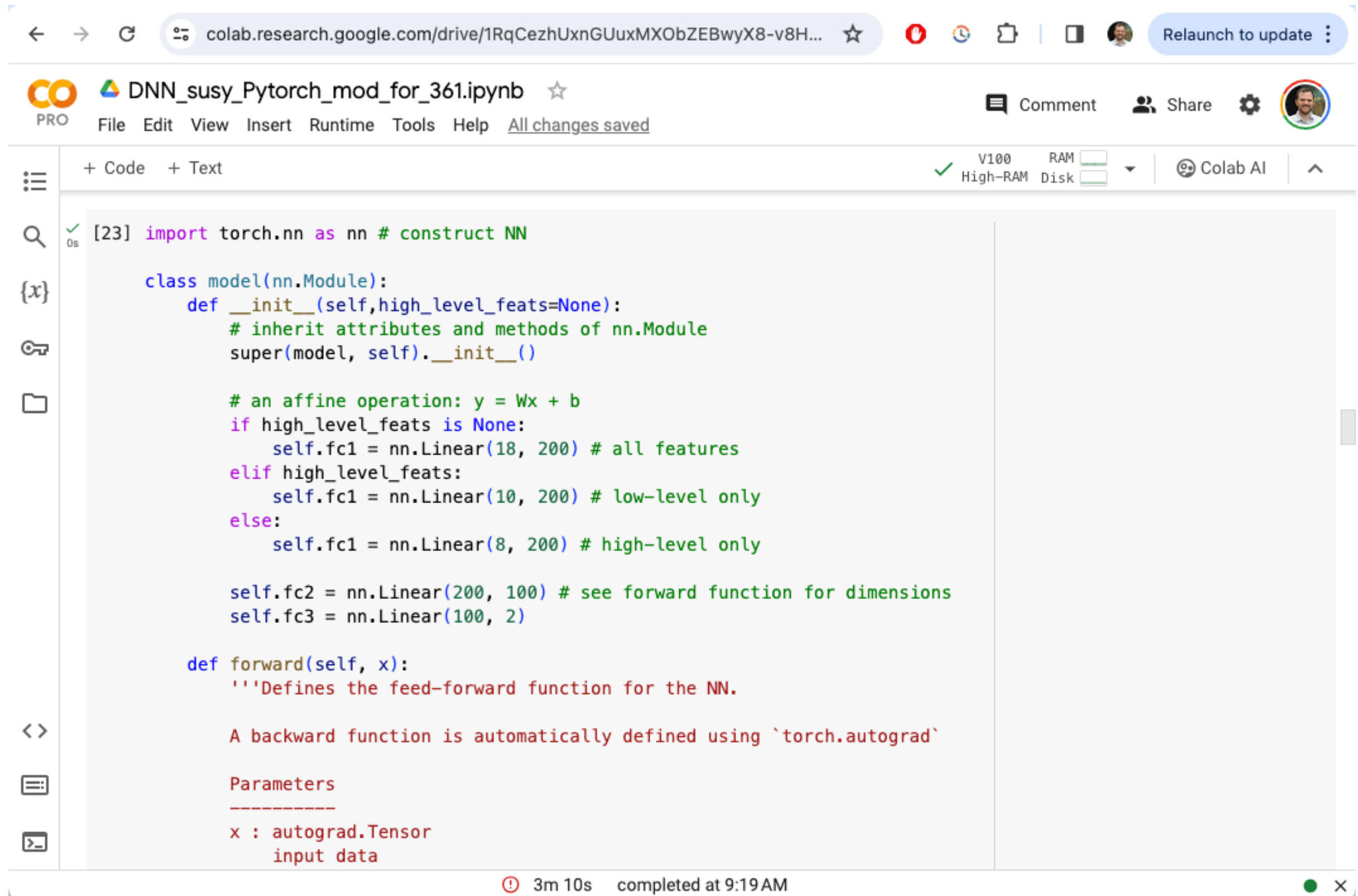


Figure 4 | Diagrams for SUSY benchmark. Example diagrams describing the signal process involving hypothetical supersymmetric particles $\chi^\pm$ and χ^0 along with charged leptons $\ell^\pm$ and neutrinos ν (a) and the background process involving W bosons (b). In both cases, the resulting observed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



The screenshot shows a Google Colab notebook titled "DNN_susy_Pytorch_mod_for_361.ipynb". The interface includes a top navigation bar with the Colab logo, file management options (File, Edit, View, Insert, Runtime, Tools, Help), and a status bar indicating "All changes saved". On the right, there are options to "Comment", "Share", and "Relaunch to update". Below the navigation bar, the notebook content is displayed in a code editor. The code defines a PyTorch neural network model named "model" that inherits from "nn.Module". The model's initialization method "__init__" takes an optional parameter "high_level_feats" and sets up three fully connected layers: "fc1", "fc2", and "fc3". The "forward" method is also defined, which takes an input "x" and returns the output. The notebook shows the execution of cell [23], which successfully imports the necessary modules and defines the model. The status bar at the bottom indicates that the notebook was completed at 9:19 AM.

```
[23] import torch.nn as nn # construct NN

class model(nn.Module):
    def __init__(self, high_level_feats=None):
        # inherit attributes and methods of nn.Module
        super(model, self).__init__()

        # an affine operation: y = Wx + b
        if high_level_feats is None:
            self.fc1 = nn.Linear(18, 200) # all features
        elif high_level_feats:
            self.fc1 = nn.Linear(10, 200) # low-level only
        else:
            self.fc1 = nn.Linear(8, 200) # high-level only

        self.fc2 = nn.Linear(200, 100) # see forward function for dimensions
        self.fc3 = nn.Linear(100, 2)

    def forward(self, x):
        '''Defines the feed-forward function for the NN.

        A backward function is automatically defined using `torch.autograd`

        Parameters
        -----
        x : autograd.Tensor
            input data
```

3m 10s completed at 9:19 AM

Course logistics

- **Reading for this lecture:**
 - **For example:** [Deeplearningbook.org](https://deeplearningbook.org) chapter 5.
- **Problem set:** Second problem set coming out today or tomorrow.