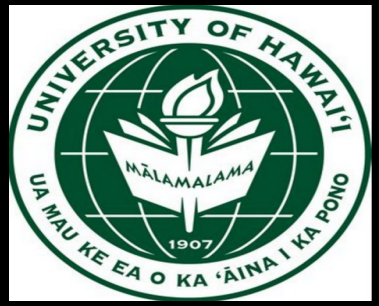


Machine Learning for New Physics in B → K* ll Decays

Belle II Summer Workshop
Ames, IA
08/03/2022

Shawn Dubey
sdubey@hawaii.edu

Introduction

- FCNC forbidden at tree-level in SM
- Sensitive to NP/BSM physics
- May be fruitful to study b → s processes
- A. Sibidanov has implemented improved EvtGen MC model for B → K*ll
  - Improved SM calculations in OPE
  - BSM physics from generic d-6 operators can be set by user in EvtGen decay file by setting $\delta C_i = C_i(\text{eff}) - C_i(\text{SM})$
- Use machine learning (ML) and likelihood-free inference (LFI) methods to try and extract NP information in B → K*ll MC sample
  - Use ML and template fitting to try and determine NP parameters; not simply background suppression
  - Proof-of-concept with signal MC; does not yet include detector response simulation

Introduction

$$\mathcal{M} = \frac{G_F \alpha}{\sqrt{2}\pi} V_{tb} V_{ts}^* \Big\{ \Big[\langle K^* | \bar{s} \gamma^\mu (C_9^{\text{eff}} P_L + C_9' P_R) b | \bar{B} \rangle - \frac{2 m_b}{q^2} \langle K^* | \bar{s} i \sigma^{\mu\nu} q_\nu (C_7^{\text{eff}} P_R + C_7' P_L) b | \bar{B} \rangle \Big] (\bar{\ell} \gamma_\mu \ell) + \langle K^* | \bar{s} \gamma^\mu (C_{10} P_L + C_{10}' P_R) b | \bar{B} \rangle (\bar{\ell} \gamma_\mu \gamma_5 \ell) \Big\},$$

$$\mathcal{O}_S^{(\prime)} = (\bar{s} P_{R(L)} b)(\bar{\mu}\mu) \quad \text{and} \quad \mathcal{O}_P^{(\prime)} = (\bar{s} P_{R(L)} b)(\bar{\mu}\gamma_5 \mu)$$

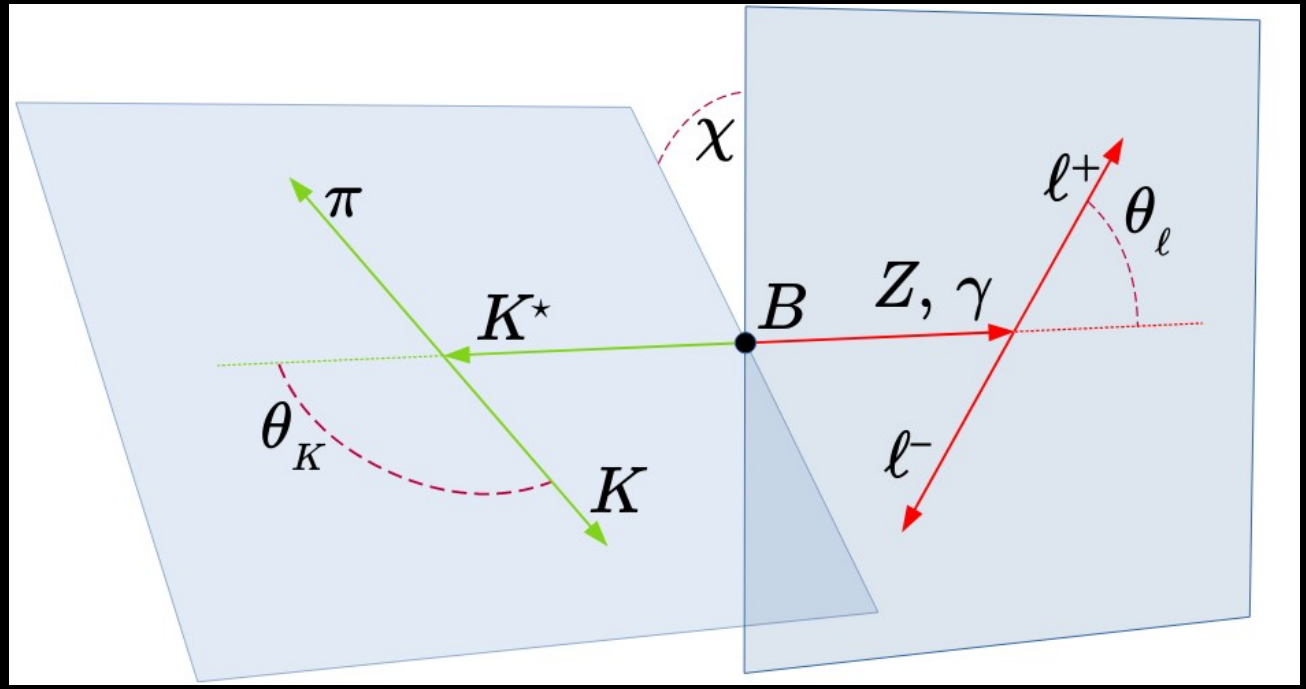
Introduction

$$\mathcal{M} = \frac{G_F \alpha}{\sqrt{2}\pi} V_{tb} V_{ts}^* \Big\{ \Big[\langle K^* | \bar{s} \gamma^\mu (C_9^{\text{eff}} P_L + C_9' P_R) b | \bar{B} \rangle - \frac{2 m_b}{q^2} \langle K^* | \bar{s} i \sigma^{\mu\nu} q_\nu (C_7^{\text{eff}} P_R + C_7' P_L) b | \bar{B} \rangle \Big] (\bar{\ell} \gamma_\mu \ell) + \langle K^* | \bar{s} \gamma^\mu (C_{10} P_L + C_{10}' P_R) b | \bar{B} \rangle (\bar{\ell} \gamma_\mu \gamma_5 \ell) \Big\},$$

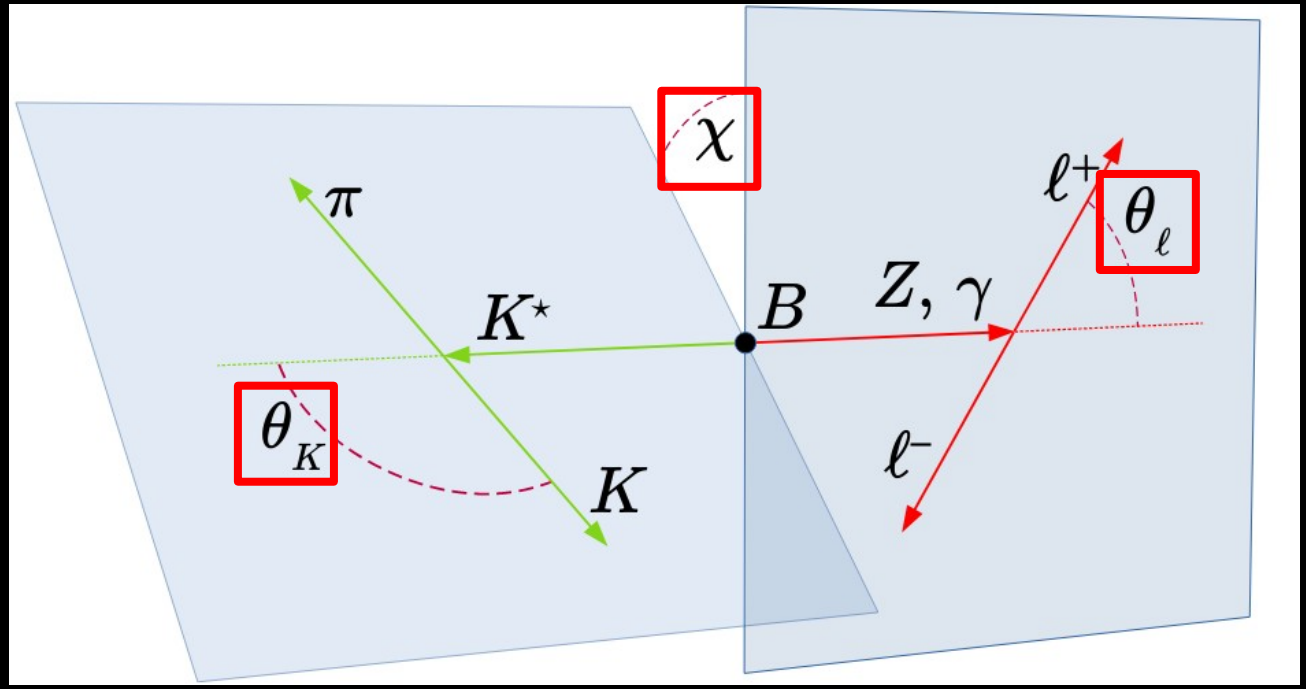
$$\mathcal{O}_S^{(\prime)} = (\bar{s} P_{R(L)} b)(\bar{\mu}\mu) \quad \text{and} \quad \mathcal{O}_P^{(\prime)} = (\bar{s} P_{R(L)} b)(\bar{\mu}\gamma_5 \mu)$$

C's are Wilson Coefficients (WC); contain information about short-distance effects.

Introduction



Introduction



Introduction

- Question: Can machine learning (ML) be used to distinguish between data that exhibit signatures due to SM or NP?

Introduction

- Question: Can machine learning (ML) be used to distinguish between data that exhibit signatures due to SM or NP?
  - Use minimal set of variables to train a neural network (NN)

Introduction

- Question: Can machine learning (ML) be used to distinguish between data that exhibit signatures due to SM or NP?
  - Use minimal set of variables to train a neural network (NN)
  - Use NN output distributions as templates perform binned template fitting to estimate “best fit” value for the δC_9 WC

Machine Learning

Machine Learning

- “[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.”
  - Arthur Samuel, 1959; taken from “Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow” by A. Geron

Neural Network

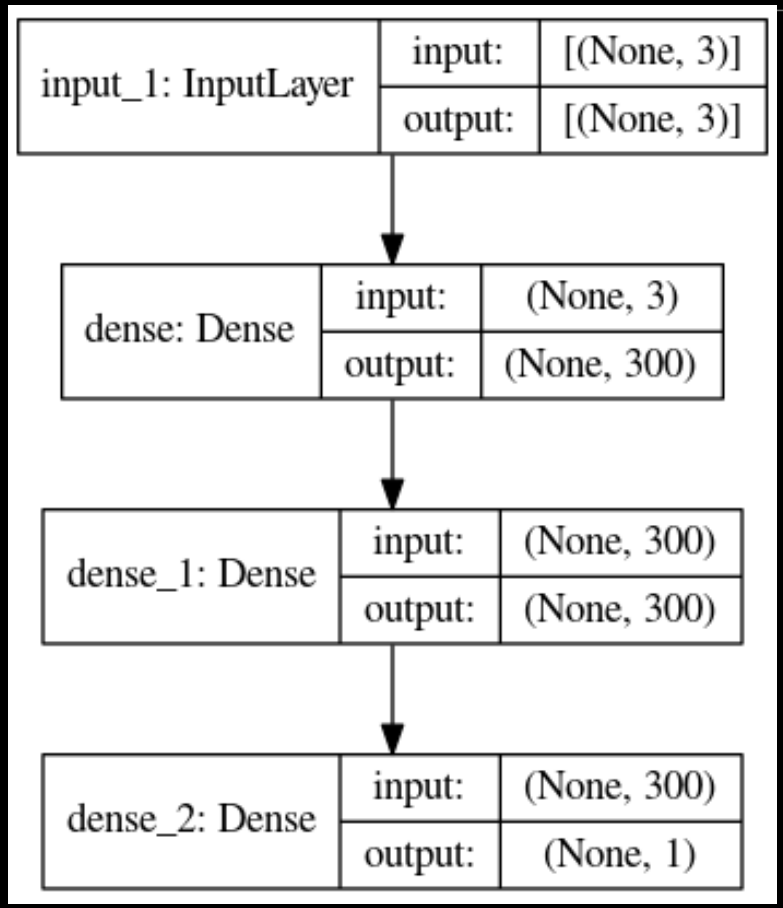
- Artificial Neural Networks (ANNs) are machine learning models that try to learn by modeling the connected neurons in the brain
  - Detail of ANNs outside of scope
  - See “Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow” by A. Geron to learn more

Neural Network: Model

- Use the Keras API to build a NN

```
# define neural network model
def build_model(n_hidden = 2, n_neurons = 300, learning_rate = 1e-3, input_shape=[3]):
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))

    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="selu"))

    model.add(keras.layers.Dense(1, activation="sigmoid"))
    optimizer = keras.optimizers.Nadam(learning_rate=learning_rate)
    model.compile(loss="binary_crossentropy", optimizer=optimizer, metrics=["accuracy"])

    return model
```

Neural Network: Model



Neural Network: Inputs

- Ideally use a minimal set of training variables
  - Similar methods used in LHCb analyses but use many variables (p_T, E, m, angular info, etc.) for NN training
  - Want to reduce the training problem to only angular information and q^2
- Proof of concept: use A_{FB} and S5, angular asymmetry variables, q^2, and nothing else, to train NN

Neural Network: Inputs

$$A_{\rm FB}(q^2) = \frac{\left[\left(\int_0^1 - \int_{-1}^0\right) \mathrm{d}\cos\theta_\ell\right] \mathrm{d}(\Gamma - \bar{\Gamma})}{\int_{-1}^1 \mathrm{d}\cos\theta_\ell \, \mathrm{d}(\Gamma + \bar{\Gamma})}$$

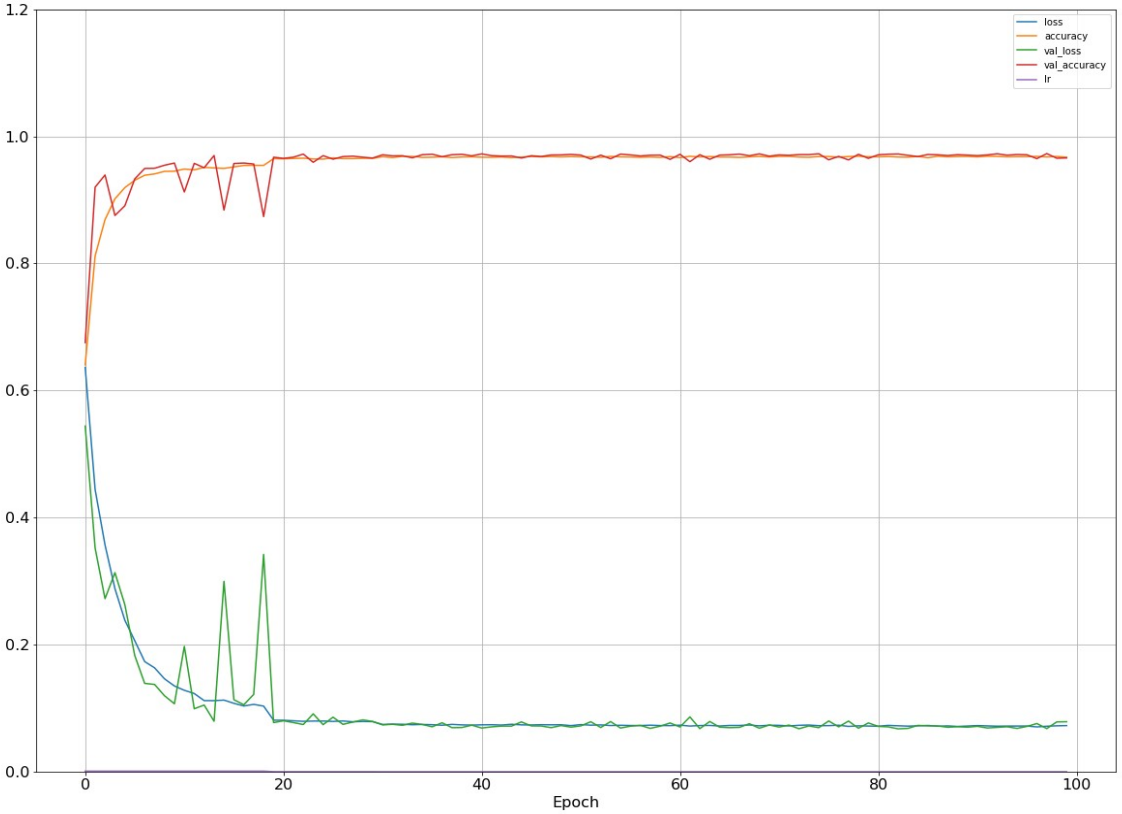
$$S_5(q^2) = \frac{4}{3} \frac{\left[\int_{-\pi/2}^{\pi/2} - \int_{\pi/2}^{3\pi/2}\right] \mathrm{d}\chi \left[\int_0^1 - \int_{-1}^0\right] \mathrm{d}\cos\theta_K \int_{-1}^1 \mathrm{d}\cos\theta_\ell \quad \mathrm{d}(\Gamma - \bar{\Gamma})}{\int_0^{2\pi} \mathrm{d}\chi \int_{-1}^1 \mathrm{d}\cos\theta_K \int_{-1}^1 \mathrm{d}\cos\theta_\ell \; \mathrm{d}(\Gamma + \bar{\Gamma})}$$

See backup.

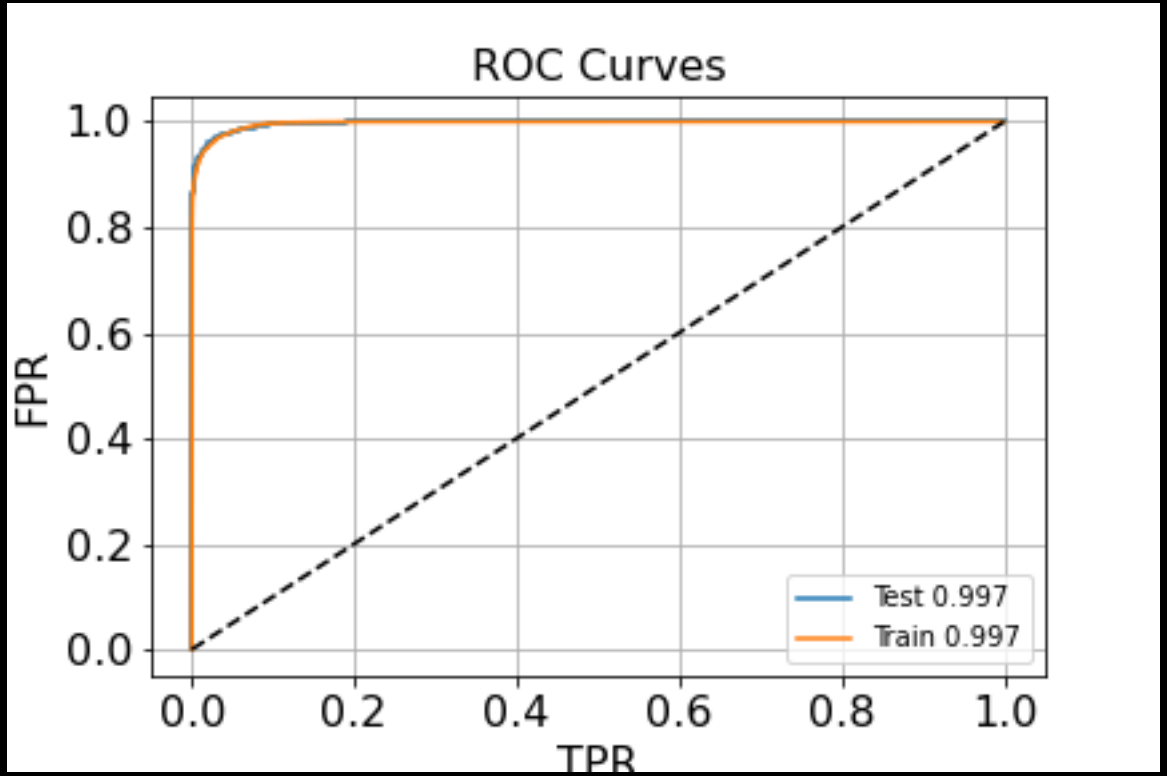
Neural Network: Training

- Training the NN with these inputs
- Balanced training set
- Reserve 20% for test set
- Use 20% for validation during training
- ReduceLROnPlateau
  - Monitor the val_loss and reduce LR when no improvement
- Training proceeds well, do not see signs of overtraining
  - Avoid over training by using larger MC data samples

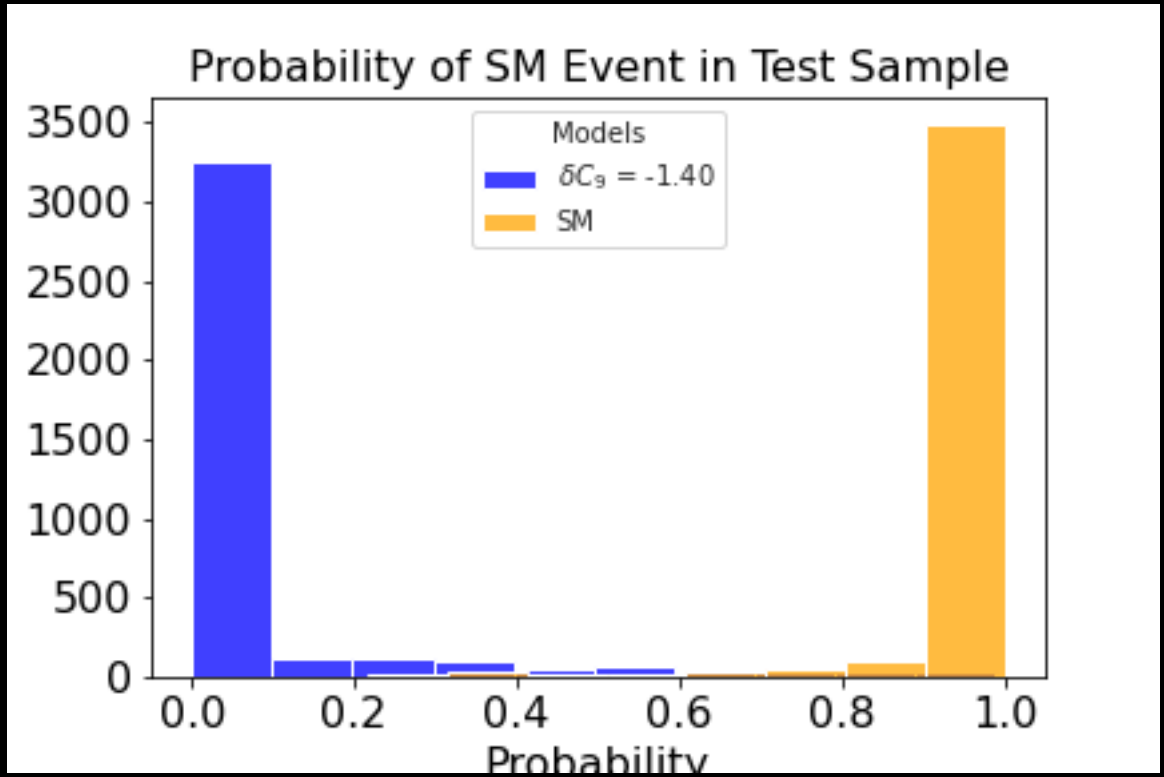
Neural Network: Training



Neural Network: Training



Neural Network: Training



Results

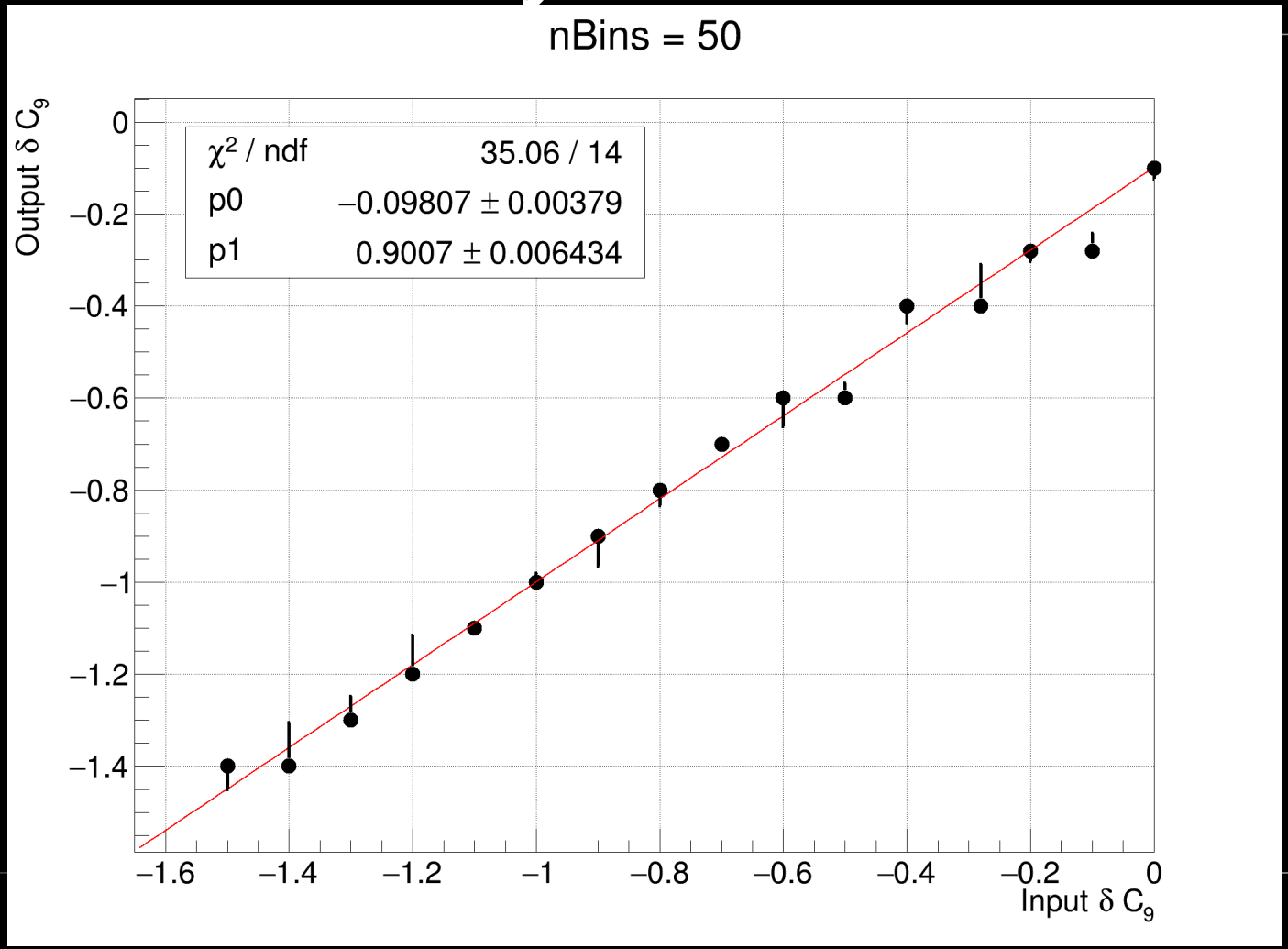
Results: Overview

- Used binned template fitting to fit the NN output distribution
  - Distribution contains all information necessary: q^2 and angular information
- Use pyhf for template fitting
  - Generate several high-statistics MC samples for different δC_9 values for templates and “data”
- Fitting gives NLL values → obtain curve without explicit form of LH (LFI) → δC_9 value with min NLL is the “best fit” value

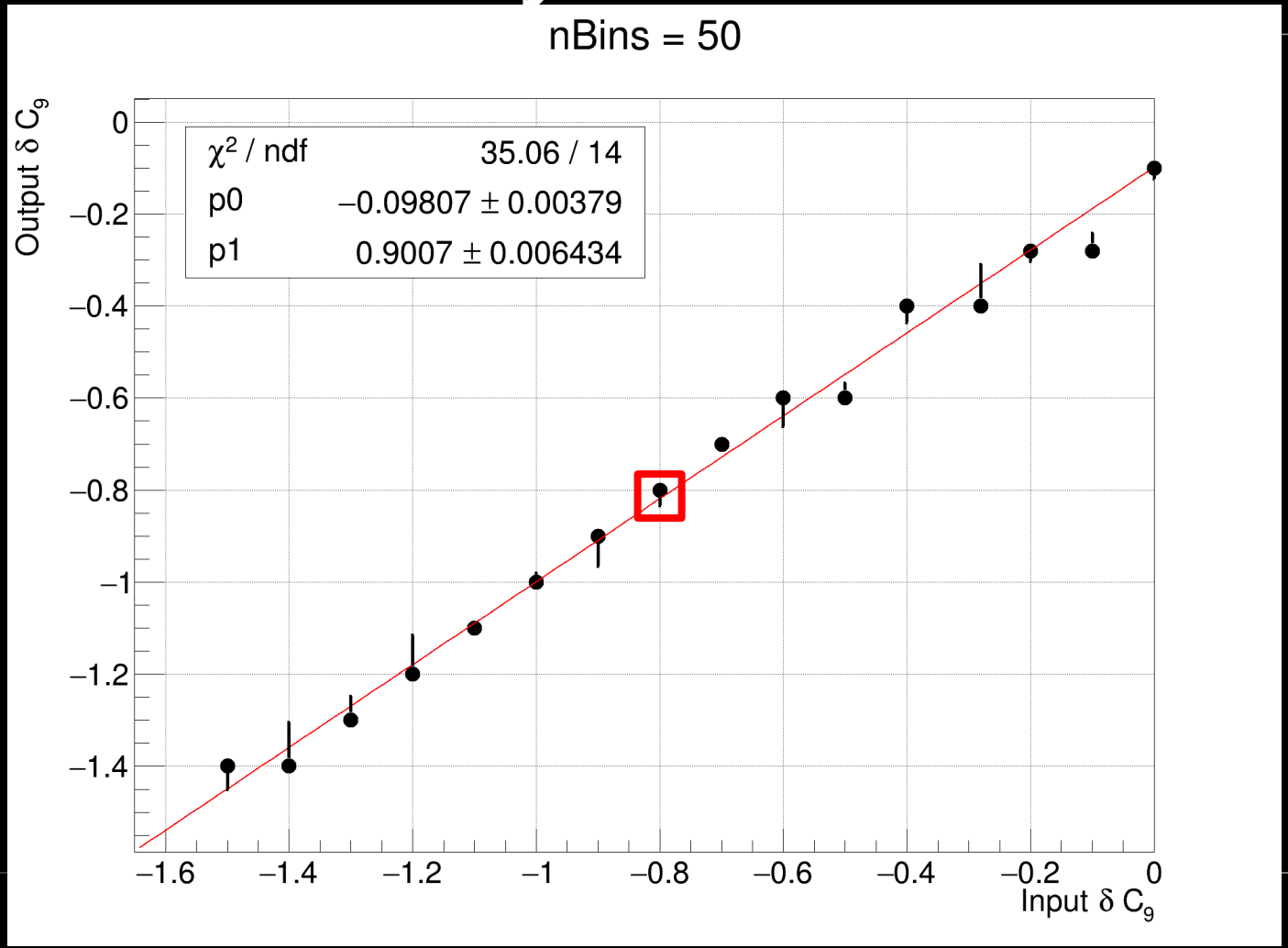
Results: pyhf Implementation

```
def template_fitter(template, data, EPSILON):

    # Signal model; Pad true zeros to avoid error with evaluating Poisson(x|0)
    epsilon = EPSILON

    template_aug = np.where(template==0,epsilon,template)
    data_aug = np.where(data==0,epsilon,data)

    #fitting model
    samples = [
    {
        "name": "sample1",
        "data": list(template_aug),
        "modifiers": [
            {"name": "mu1", "type": "normfactor", "data" : None}
        ],
    },
    ]

    spec = {"channels" : [{"name" : "singlechannel", "samples" : samples}]}

    # info: the `poi_name=None` is nescessary here since we don't want to do a hypothesis test
    model = pyhf.Model(spec, poi_name=None)

    # fit
    mu1 = pyhf.infer.mle.fit(list(data_aug), model)

    # get nll
    nll = negative_loglikelihood_pyhf(model, mu1, data_aug)

    return nll
```

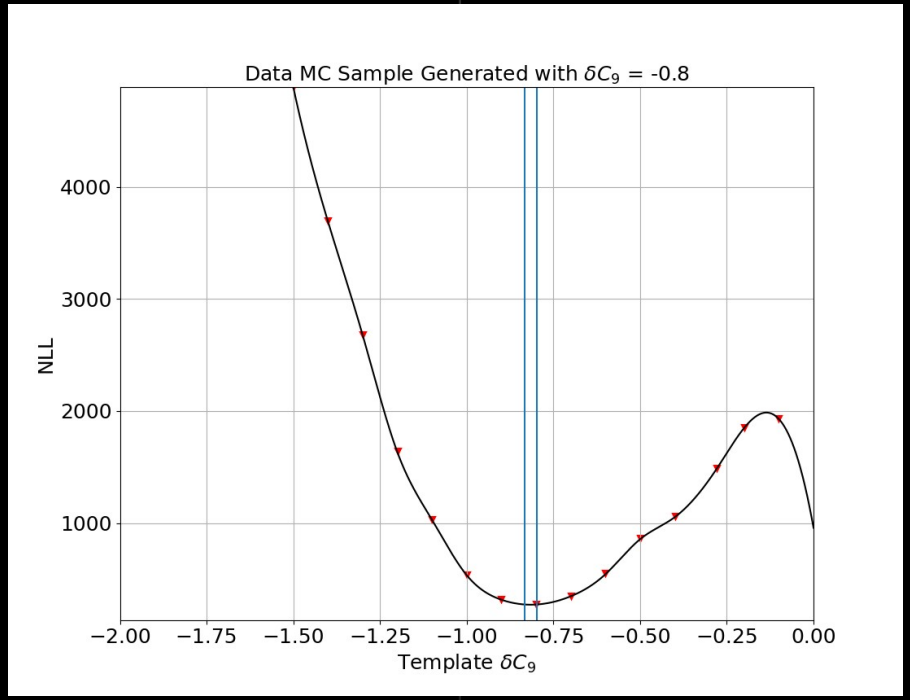
Results: Linearity Test



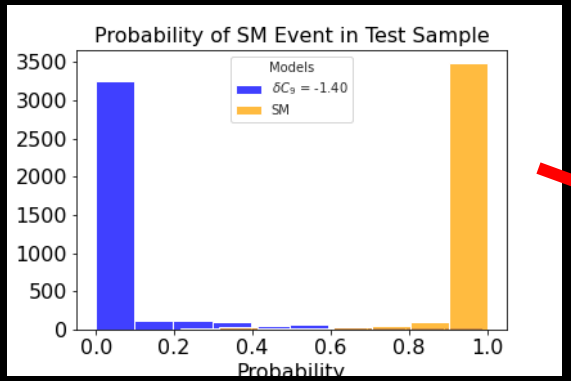
Results: Linearity Test



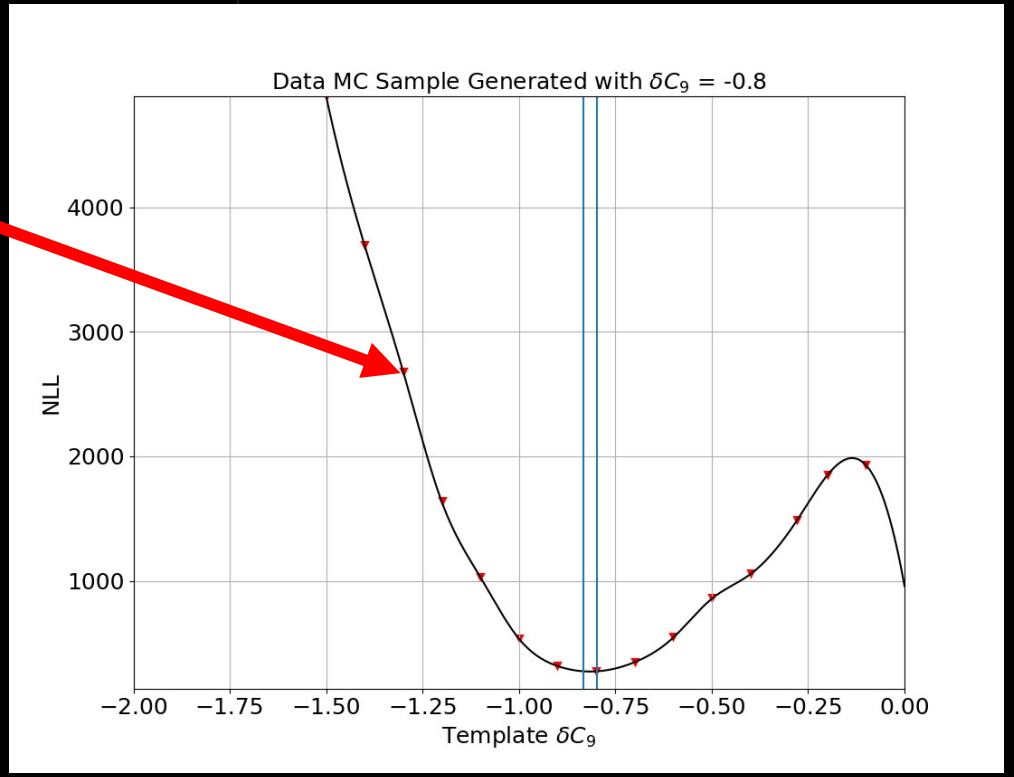
Results: Linearity Test



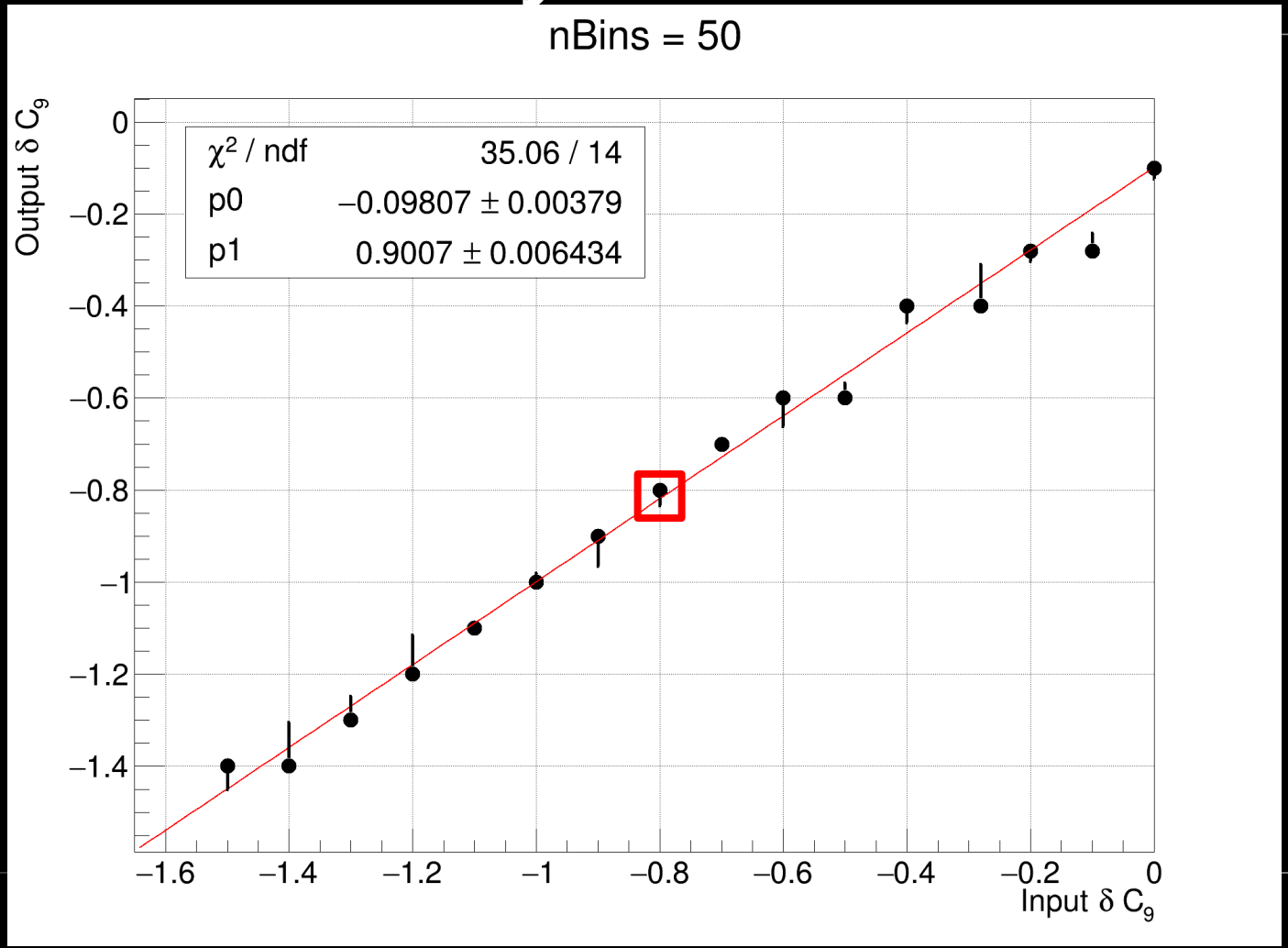
Results: Linearity Test



(e.g.)



Results: Linearity Test



Results: Linearity Test

- Good start
- Small(ish) but non-zero bias
- Some type of pathology in the asymmetric errors
- But method seems to point to being able to extract NP information; will continue to iterate to reduce the bias

Conclusion

- Used a NN trained with angular info and q^2 to implement a LFI
  - Useful since it can extract NP information from all higher dim data without needing to perform a complicated fit
- Linearity indicates this may be a reasonable method to use to extract NP information at Belle II
  - Implemented in some LHC analyses but not Belle(II)
- Hope to improve this then move on to simply using angular information, binned in q^2, rather than variables like A_{FB} and S5
- Done in collaboration with T.E. Browder (U. Hawaii Manoa), S. Kohani (U. Hawaii Manoa), R. Mandal (IIT Gandhinagar), S. Sandilya (IIT Hyderabad), A. Sibidanov (U. Hawaii Manoa), R. Sinha (IMSc, U. Hawaii Manoa), and S.E. Vahsen (U. Hawaii Manoa)
- A similar program is planned for B → D* l nu, with another group

Backup

Backup: Machine Learning Jargon

- Hyperparameter
  - parameter of learning algorithm (A. Geron)
    - Examples
      - learning rate
      - number of hidden layers
      - batch size

Backup: Machine Learning Jargon

- Activation function
  - A function that computes the output of a layer of neurons
    - Different functions for different purposes
      - Ones used in this study
        - Scaled Exponential Linear Unit (SELU)
        - Sigmoid

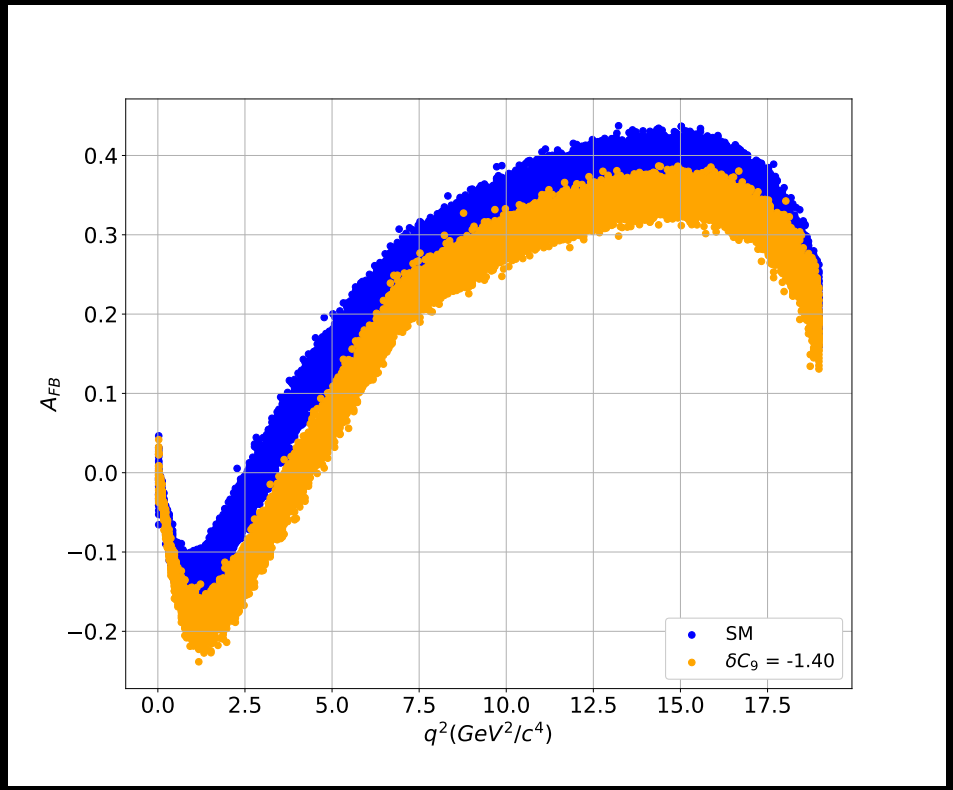
Backup: Machine Learning Jargon

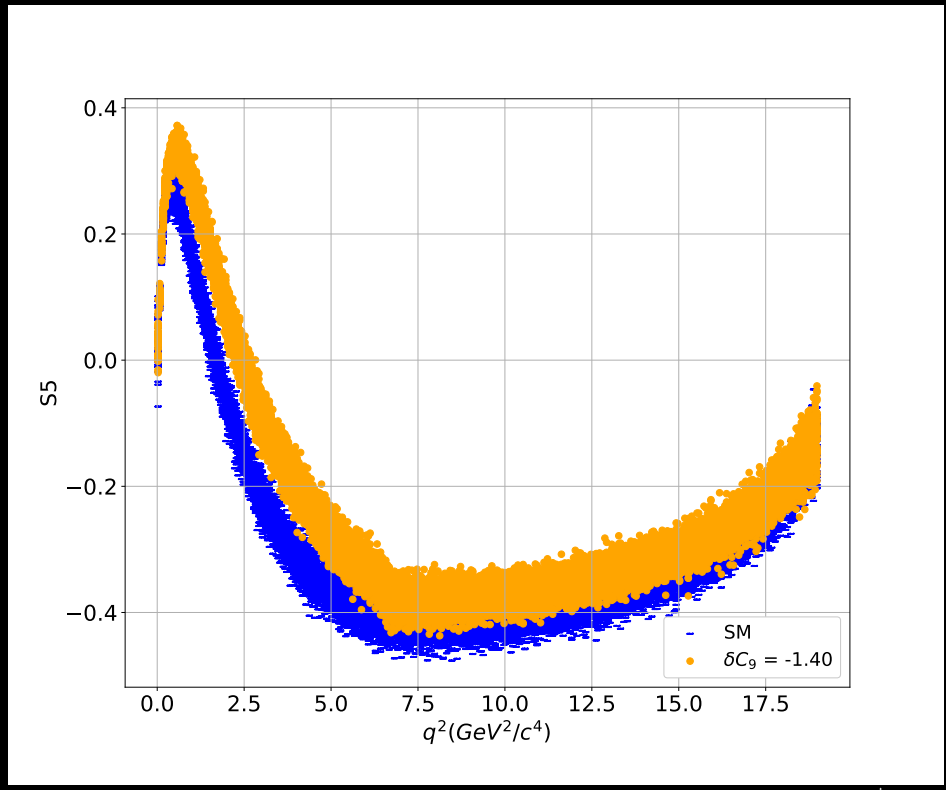
- Loss
  - A function that compares the output of the NN to the desired output
    - Used (by gradient descent) to train a model
    - Many functions for many purposes
      - Binary cross-entropy used in this study
- Metric
  - Used to evaluate a model
    - Many types of metrics
      - accuracy used in this study

Backup: Machine Learning Jargon

- Optimizer
  - “Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.”
    - https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
  - Use the Nadam optimizer in this study
    - See A. Geron, Ch. 11 for definition

Neural Network: Inputs





Widths are due to MC statistical fluctuations