Hyperparameter tunning

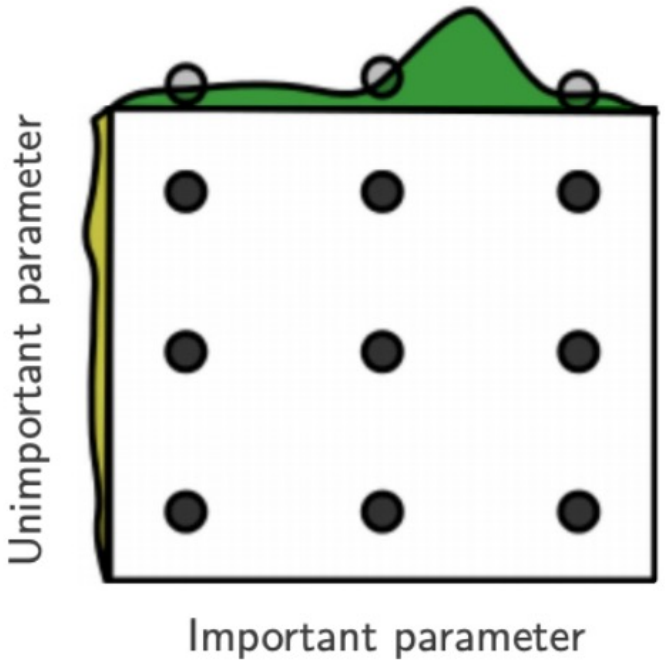
Qingrun Zhang

Parameters

- **Trainable parameters:**
  - weights in neural networks. User does not interfere.
- **Hyper-parameters**
  - Learning rate α,
  - the momentum term β
  - for the Adam Optimization Algorithm$\beta_1,\ \beta_2,\ \varepsilon$
  - The number of layers,
  - the number of hidden units for each layers,
  - Learning rate decay,
  - The mini-batch size.

Hyper parameter tuning process

- Grid search: try all possible combinations of parameter values. Use GridSearchCV from Scikit Learn.
  - Pros: Can get the best hyper-parameter
  - Cons: Computational expensive
- Random Search: randomly select a pre-definec number of combination of hyper-parameters. Use RandomizedSearchCV from Scikit Learn.
- Bayesian Optimization: under scikit-optimize (skopt) library. skopt.gp_minimize



https://towardsdatascience.com/3-ways-to-tune-hyperparameters-of-machine-learning-models-with-python-cda64b62e0ac

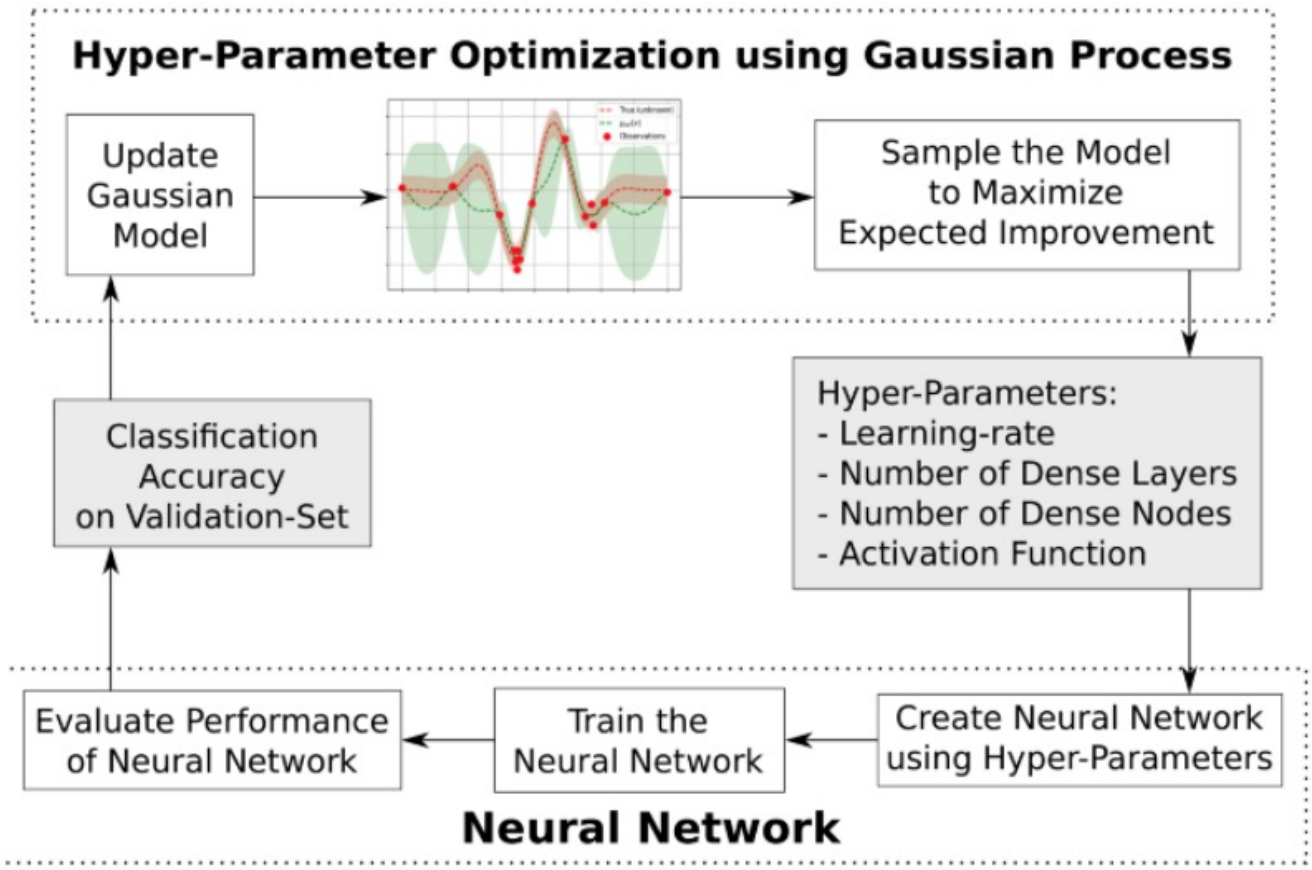
```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC()
>>> clf = GridSearchCV(svc, parameters)
>>> clf.fit(iris.data, iris.target)
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
>>> sorted(clf.cv_results_.keys())
['mean_fit_time', 'mean_score_time', 'mean_test_score',...
 'param_C', 'param_kernel', 'params',...
 'rank_test_score', 'split0_test_score',...
 'split2_test_score', ...
 'std_fit_time', 'std_score_time', 'std_test_score']
```

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

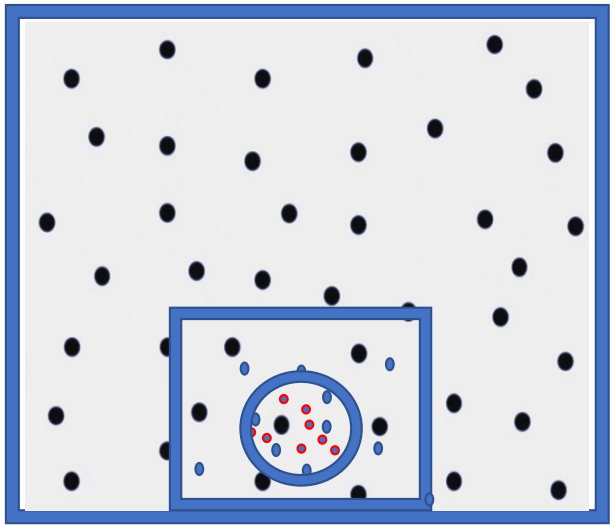
```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.model_selection import RandomizedSearchCV
>>> from scipy.stats import uniform
>>> iris = load_iris()
>>> logistic = LogisticRegression(solver='saga', tol=1e-2, max_iter=200,
...                               random_state=0)
>>> distributions = dict(C=uniform(loc=0, scale=4),
...                      penalty=['l2', 'l1'])
>>> clf = RandomizedSearchCV(logistic, distributions, random_state=0)
>>> search = clf.fit(iris.data, iris.target)
>>> search.best_params_
{'C': 2..., 'penalty': 'l1'}
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

Bayesian Optimization

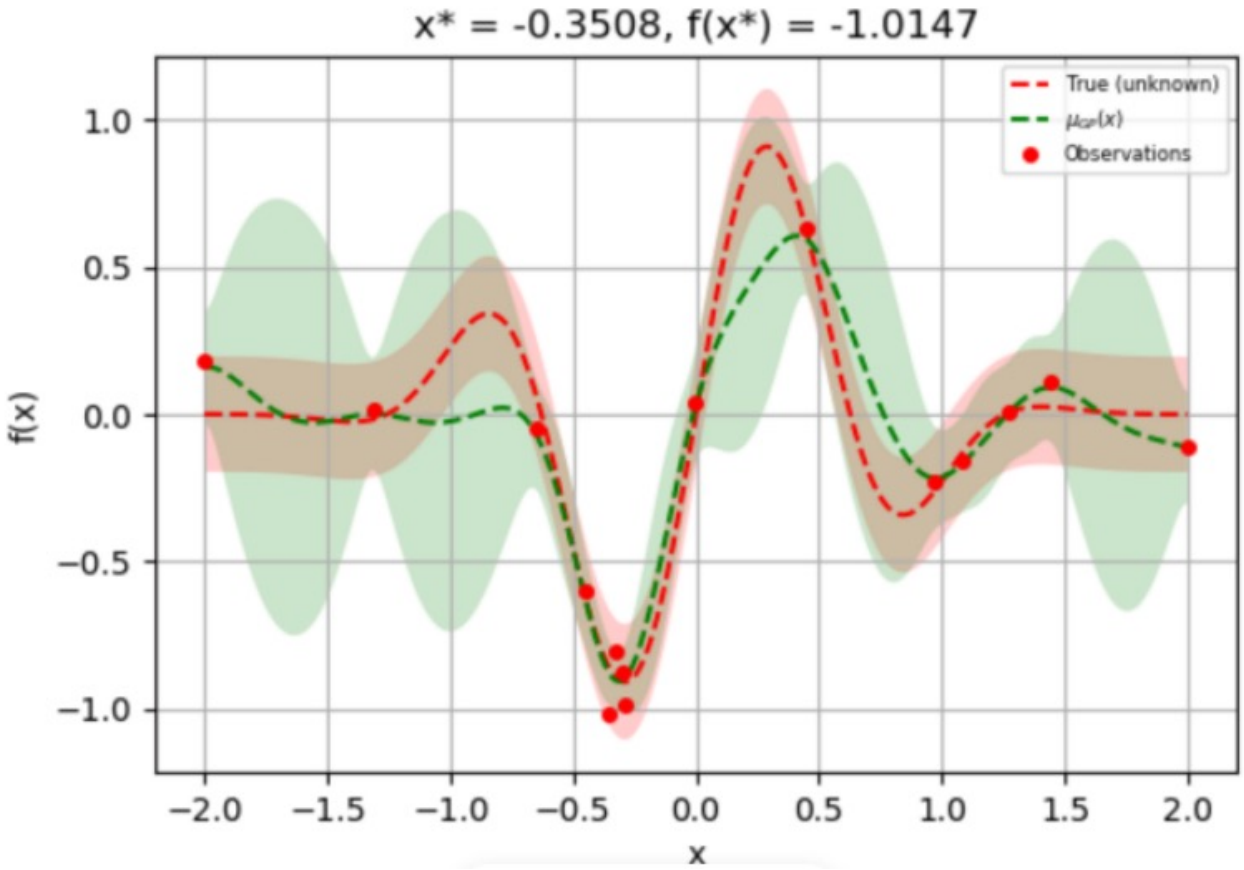


Coarse to fine



https://towardsdatascience.com/3-different-ways-to-tune-hyperparameters-interactive-python-code-87548d7f2365

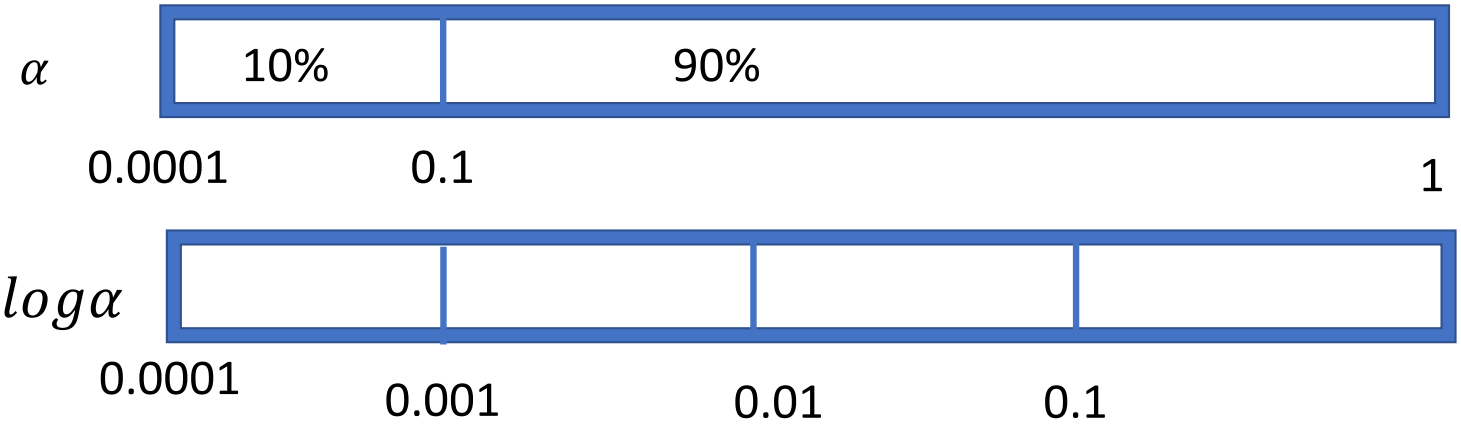
Bayesian Optimization



https://towardsdatascience.com/bayesian-hyper-parameter-optimization-neural-networks-tensorflow-facies-prediction-example-f9c48d21f795

Hyper-parameters Scale

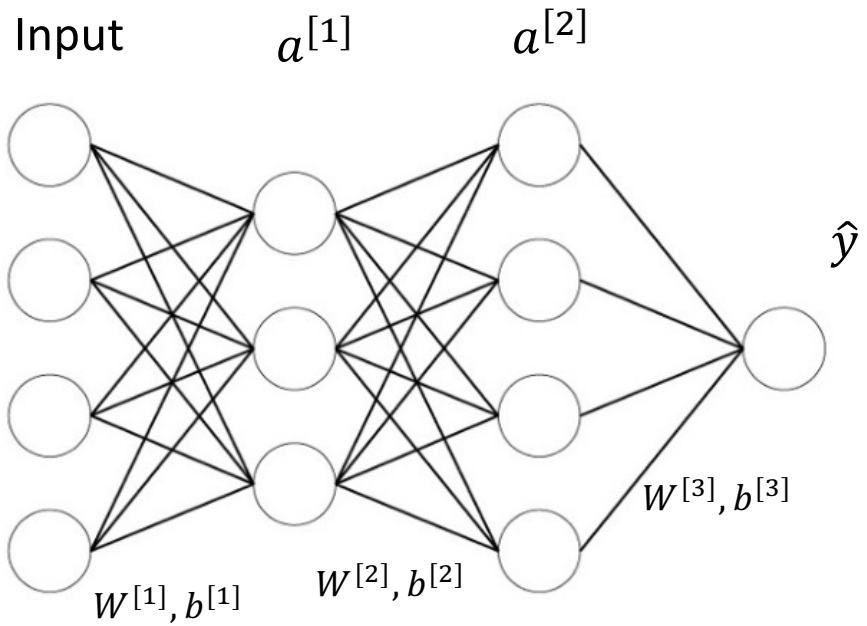
- Random sampling doesn't mean uniform sampling at random.

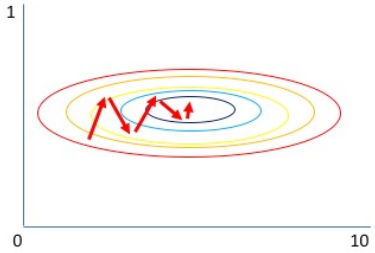


- $log\alpha$

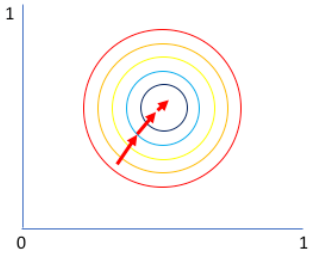
- r = -4 * np.random.rand() # -4 <= r <= 0, uniformly at random
- alpha = np.exp(10, r) # 10e-4 <= alpha <= 1.0

Batch normalization





Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

$$\mu = \frac{1}{m}\sum_i Z^{(i)}$$

$$\sigma = \frac{1}{m}\sum_i \left(Z^{(i)} - \mu\right)^2$$

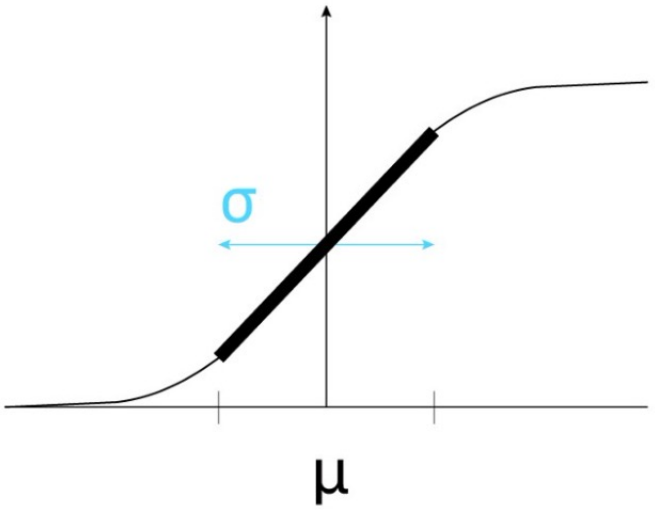
$$Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z^{(i)}_{norm} + \beta$$

$\gamma\ and\ \beta$ are learnable parameter, which allows the hidden value have other means and variances

Sergey Ioffe and Christian Szegedy, 2015. Proceedings of MLR

Why do we use $\gamma \; and \; \beta$?

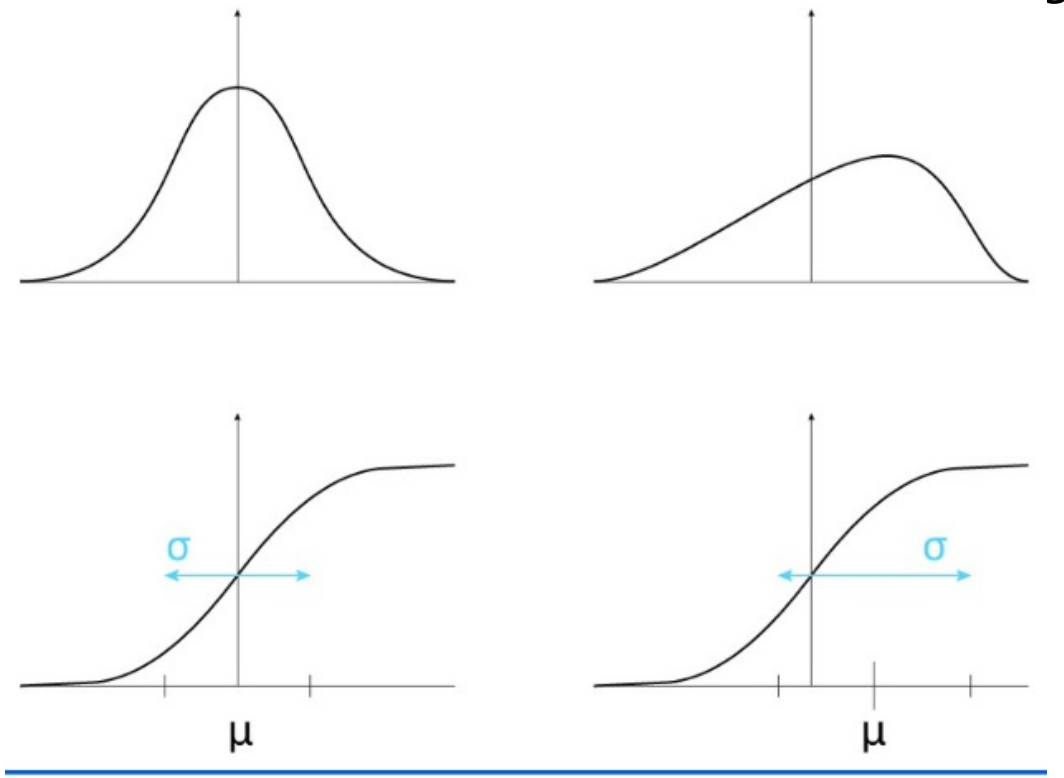


For example: Sigmoid activation function:

- If input values lie between 0 and 1, the nonlinear function would only work in its linear regime.
- $\gamma \; and \; \beta$ allow the optimizer to define optimum mean (using β) and standard deviation (using γ)

Why do we use γ and β ?

- γ allows to adjust the standard deviation ;
- β allows to adjust the bias, shifting the curve on the right or on the left side.



How to add batch norm to a network

$$x \xrightarrow{w^{[1]},\, b^{[1]}} z^{[1]} \xrightarrow{\gamma^{[1]},\beta^{[1]}BN} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]},\, b^{[2]}} z^{[2]} \xrightarrow{\gamma^{[2]},\beta^{[2]}BN} \tilde{z}^{[2]} \ldots$$

Parameters: $w^{[1]}, b^{[1]}, \gamma^{[1]}, \beta^{[1]}$……

```
my_seq = tf.keras.Sequential([tf.keras.layers.Conv2D(1, (1, 1),
                                                     input_shape=(
                                                         None, None, 3)),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Conv2D(2, 1,
                                                     padding='same'),
                              tf.keras.layers.BatchNormalization(),
                              tf.keras.layers.Conv2D(3, (1, 1)),
                              tf.keras.layers.BatchNormalization()])
my_seq(tf.zeros([1, 2, 3, 3]))
```

Batch normalization

BN-Baseline: Same as Inception with Batch Normalization before each nonlinearity.

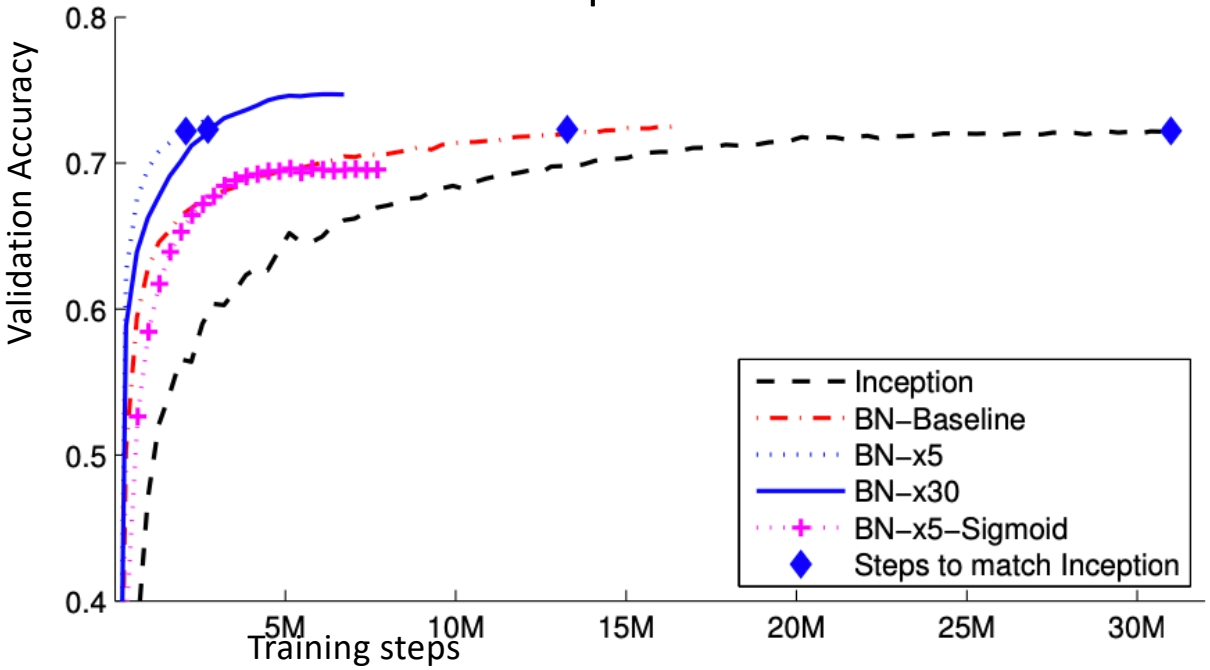


Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

https://proceedings.mlr.press/v37/ioffe15.pdf

- Adding BN layers leads to faster and better convergence.
- Adding BN layer allows to use higher learning rate (LR) without compromising convergence .
- The sigmoid-based model reached competitive results with ReLU-based models

Batch norm with Mini-batch

- $x^{\{1\}} \xrightarrow{w^{[1]},\, b^{[1]}} z^{[1]} \xrightarrow{\gamma^{[1]},\beta^{[1]}BN} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]},\, b^{[2]}} z^{[2]} \xrightarrow{\gamma^{[2]},\beta^{[2]}BN} \tilde{z}^{[2]} \ldots$
- $x^{\{2\}} \xrightarrow{w^{[1]},\, b^{[1]}} z^{[1]} \xrightarrow{\gamma^{[1]},\beta^{[1]}BN} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]},\, b^{[2]}} z^{[2]} \xrightarrow{\gamma^{[2]},\beta^{[2]}BN} \tilde{z}^{[2]} \ldots$

….

Parameters: $w^{[1]}$, ~~$b^{[1]}$~~, $\gamma^{[1]}, \beta^{[1]}$……

$$z^{[l]} = w^{[l]} a^{[l-1]} + \cancel{b^{[l]}}$$

$z^{[l]} \in \mathbb{R}\,(n^{[l]}, 1)$

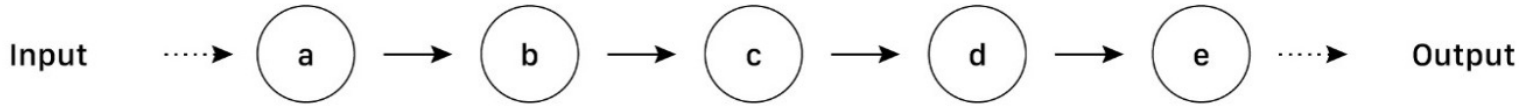
$$z^{[l]} = w^{[l]} a^{[l-1]} \rightarrow z^{[l]}_{norm}$$

What's the dimension for $\gamma^{[l]}, \beta^{[l]}$?

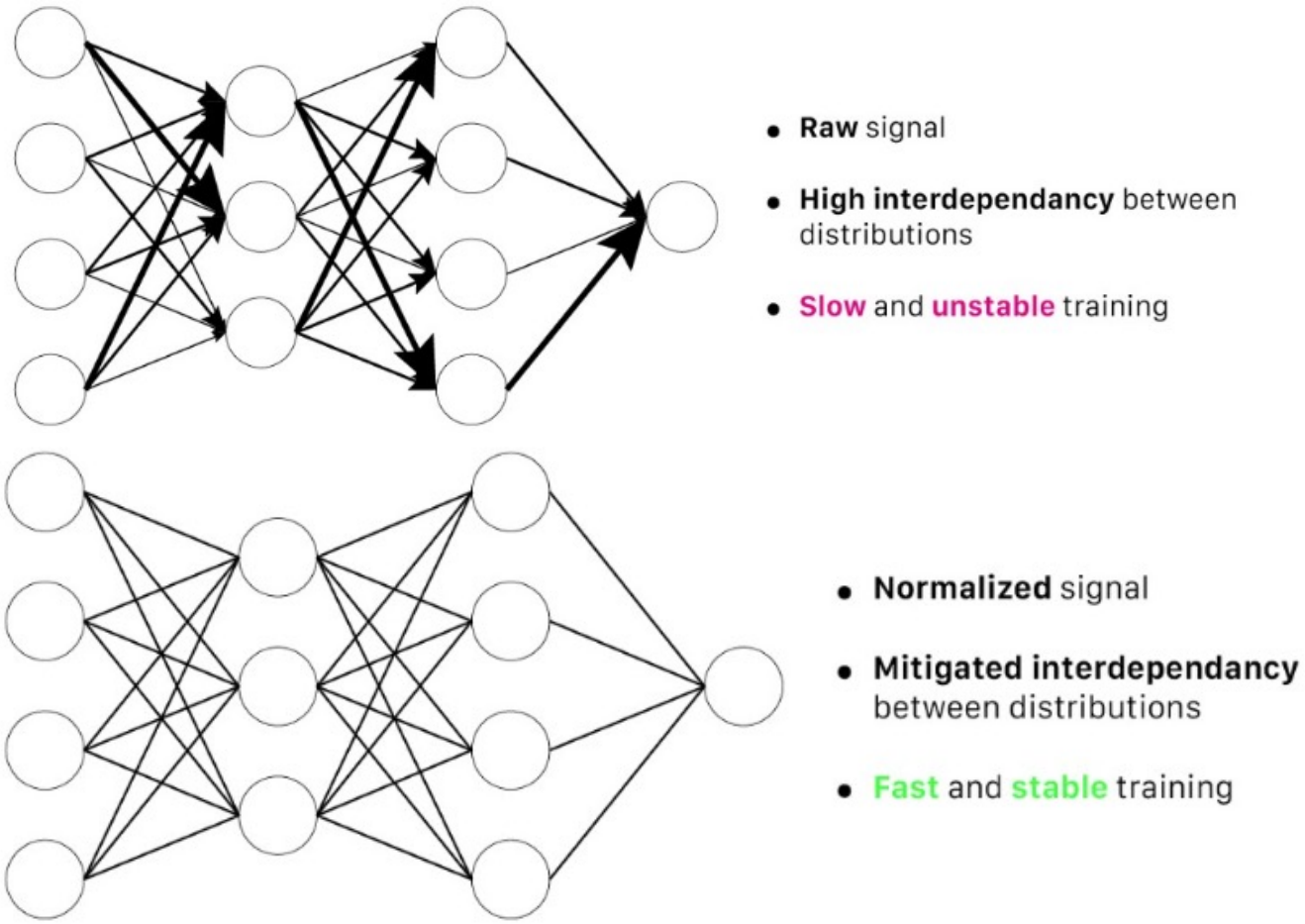
$$\tilde{z}^{[l]} = \gamma^{[l]}\, z^{[l]}_{norm} + \beta^{[l]}$$

Why does BN work?

- Hypothesis 1: BN reduce of internal covariate shift (ICS) 
  - Training distributions are statistically too different from testing. E.g. classification of cat or not, training data are black cat, testing are colored cat
  - **It is wrong, in practice there is no correlation** between ICS and training performances
- Hypothesis 2: BN mitigates the interdependency between layers during training.



  - Grad(a)=grad(b)*grad(c)* grad(d)* grad(e)
  - Without BN, if all gradients are large, grad(a) will be very large. Or if all gradients are small, grad(a) will be negligible



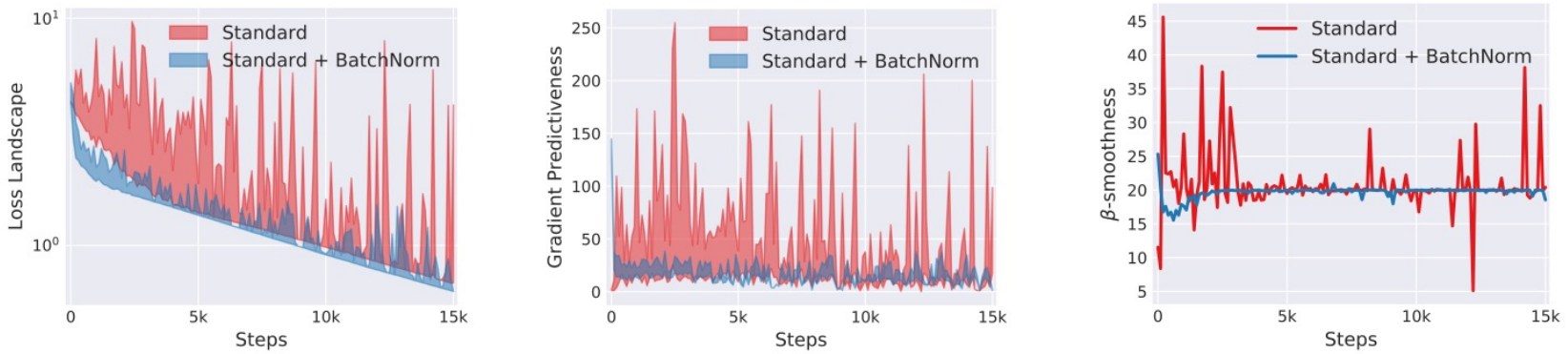
Multilayer Perceptron (MLP) without batch normalization (BN) | Credit : author - Design : Lou HD

How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

NIPS 2018

- Hypothesis 3: BN make the optimization landscape smoother, make the training easier



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

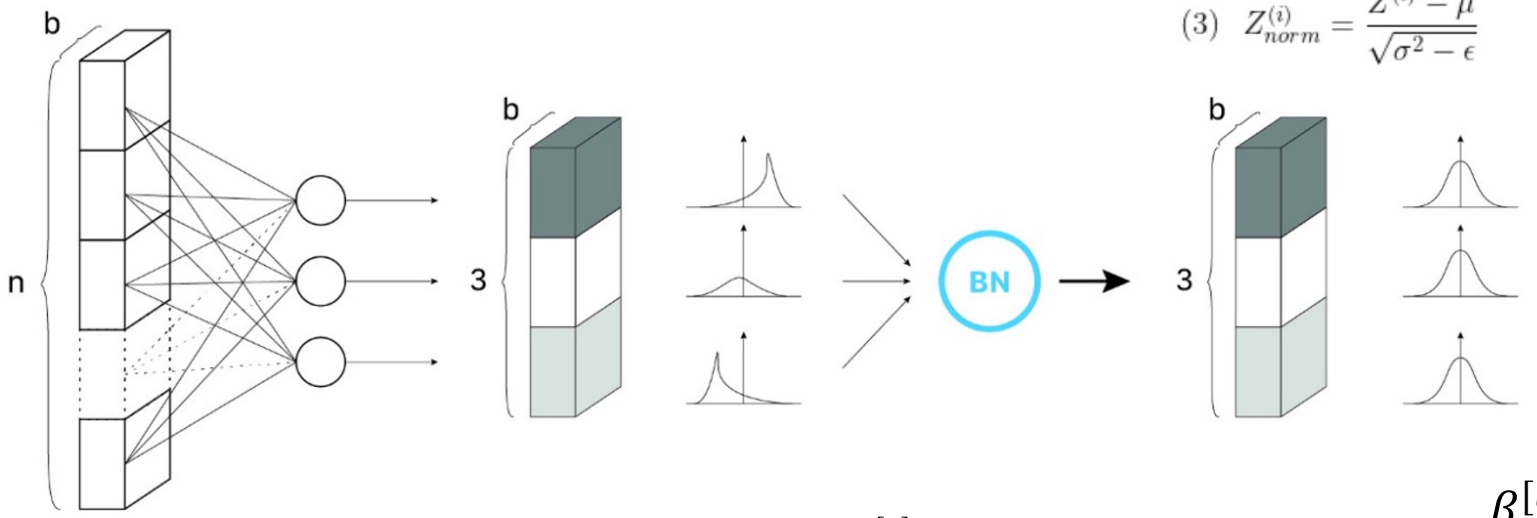
Figure 4: Analysis of the optimization landscape of VGG networks.

a) The variation (shaded region) in loss. b) L2 changes in the gradient. C) The "effective" β-smoothness, refers to the maximum difference (in L2-norm) in gradient over distance moved in that direction.

Batch normalization

$$(1)\ \mu = \frac{1}{n}\sum_i Z^{(i)} \qquad (2)\ \sigma^2 = \frac{1}{n}\sum_i (Z^{(i)} - \mu)^2$$

$$(3)\ Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \epsilon}} \qquad (4)\ \breve{Z} = \gamma * Z^{(i)}_{norm} + \beta$$



$$X \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow[\gamma^{[1]}\beta^{[1]}]{BN} \tilde{Z}^{[1]} \xrightarrow{g^{[1]}(\tilde{Z}^{[1]})} a^{[1]} \xrightarrow{W^{[2]},b^{[2]}} Z^{[2]} \ldots\ldots$$

$$\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$$

tf.nn.batch_normalization(x, mean, variance, offset, scale, variance_epsilon, name=None)
scale γ to it, as well as an offset β

https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338

Working with mini-batches

- $X^{\{1\}} \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow{\gamma^{[1]}\beta^{[1]}} \tilde{Z}^{[1]} \xrightarrow{g^{[1]}(\tilde{Z}^{[1]})} a^{[1]} \xrightarrow{W^{[2]},b^{[2]}} Z^{[2]} \ldots\ldots$

$X^{\{2\}} \xrightarrow{W^{[1]},b^{[1]}} Z^{[1]} \xrightarrow{\gamma^{[1]}\beta^{[1]}} \tilde{Z}^{[1]} \xrightarrow{g^{[1]}(\tilde{Z}^{[1]})} a^{[1]} \xrightarrow{W^{[2]},b^{[2]}} Z^{[2]} \ldots\ldots$

$X^{\{t\}} \longrightarrow Z^{[1]} \longrightarrow \tilde{Z}^{[1]} \longrightarrow a^{[1]} \longrightarrow Z^{[2]} \ldots\ldots$

Parameters: $w^{[l]}, \cancel{b^{[l]}}, \beta^{[l]}, \gamma^{[l]}$ $\qquad z^{[l]} = w^{[l]}a^{[l-1]} + \cancel{b^{[l]}}$

Each mini-batch is scaled by the mean and variance computed on that mini-batch This add some noise to the value $z^{[l]}$ with in. that minibatch. BN adds noise to each hidden layer's activations. Similar to dropout, BN has slight regularization effect. Larger minibatch size will reducing regularization effect.

BN at test time

Training time

For each mini-batch

$$\mu = \frac{1}{b}\sum_i Z^{(i)}$$

$$\sigma = \frac{1}{b}\sum_i \left(Z^{(i)} - \mu\right)^2$$

$$Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z^{(i)}_{norm} + \beta$$

Testing time, $\mu \; and \; \sigma^2$ are estimated using a exponentially weighted average across the mini batches in the training time

$$X^{\{1\}}, X^{\{2\}}, X^{\{3\}} \ldots$$

$$\mu^{\{1\}[l]}, \mu^{\{2\}[l]}, \mu^{\{3\}[l]} \ldots \rightarrow \mu^{[l]}$$

$$\sigma^{2\{1\}[l]}, \sigma^{2\{2\}[l]}, \sigma^{2\{3\}[l]} \ldots \rightarrow \sigma^{2[l]}$$

$$z_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z} = \gamma z_{norm} + \beta$$

Example code

- https://colab.research.google.com/drive/1nMgrmfBNdTiD0MLf-ReSJfPUajUhUvn_#scrollTo=95UPdhbH5leA
- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/keras_tuner.ipynb

Project 1 part c

- Add batch norm to your model build to classify beans
- Use grid search, random search and Bayesian Optimization to tune the hyper parameters