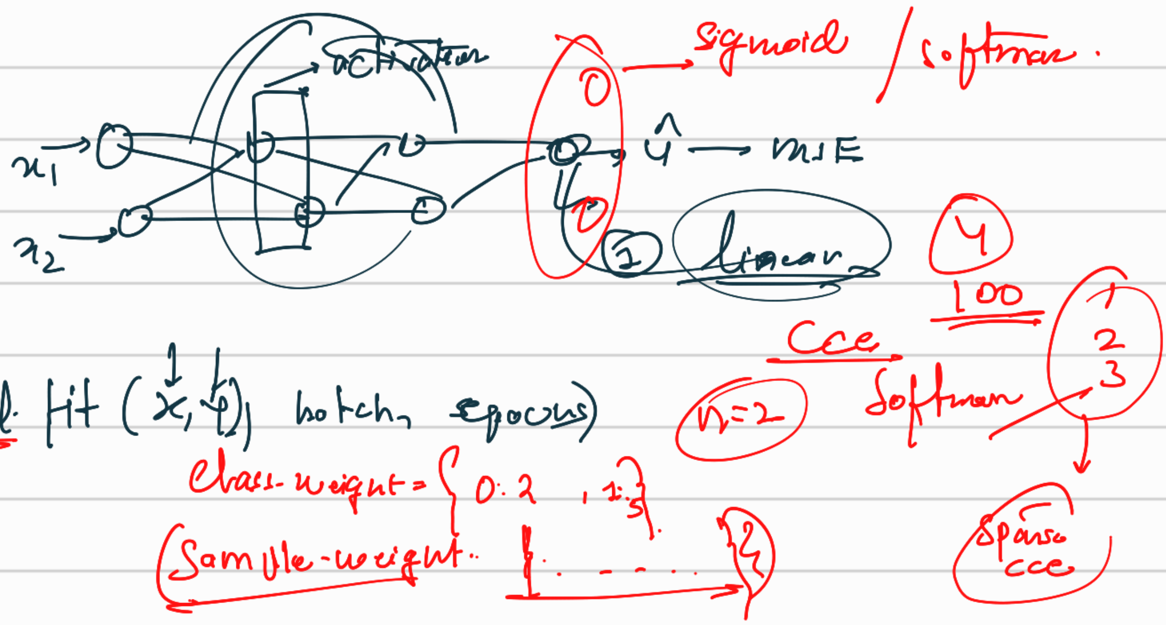
Practical aspects of designing NN

- → Revision
- → NN Tuner Keras
- → NN for optimization problems



model.fit(x, y, batch, epochs)

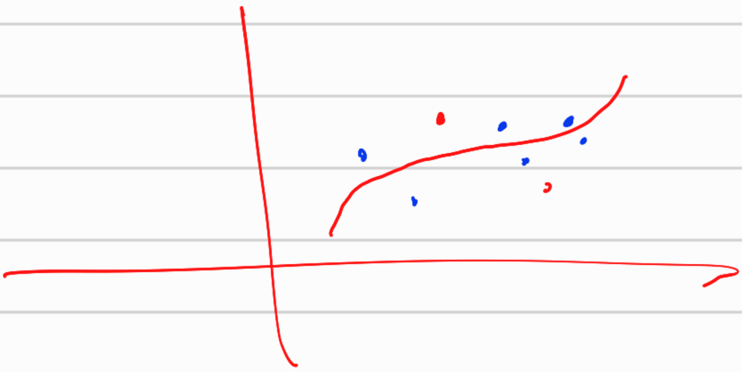
class-weight = {0: 2 , 1: 5}

Sample-weight: ...

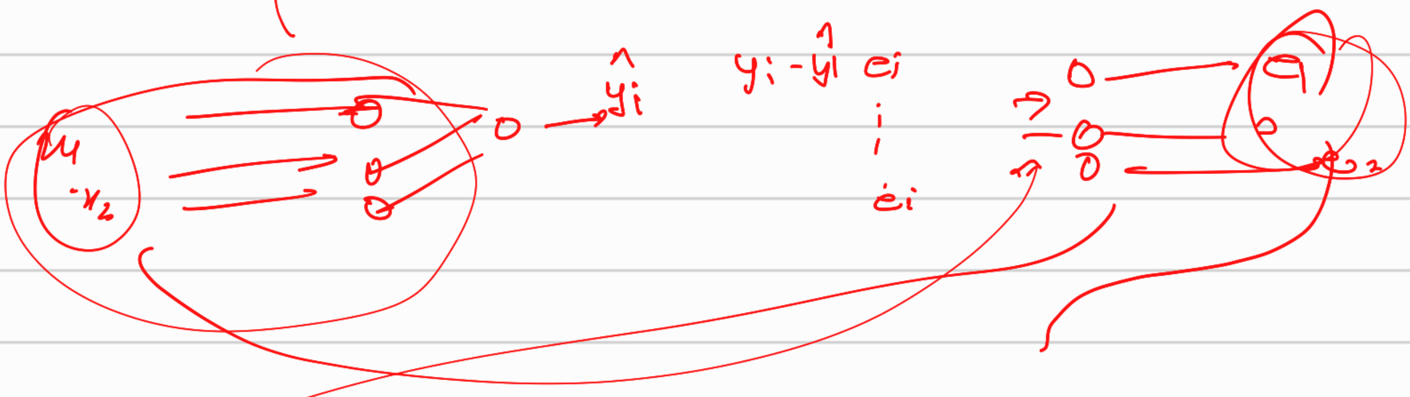
{0: 50   1: 100}

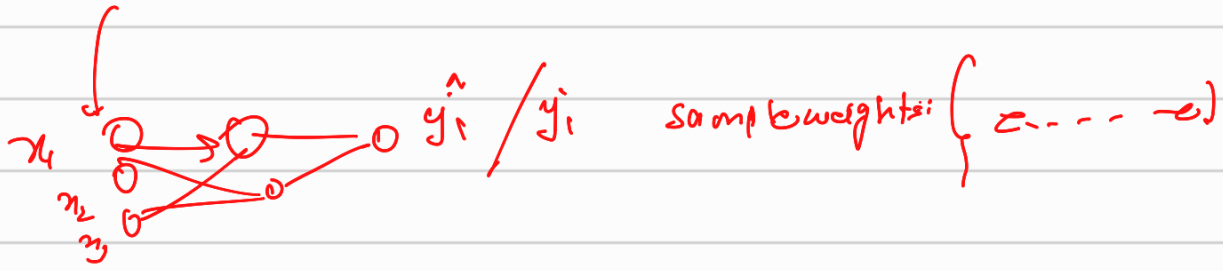
$$\frac{100}{150} \times y \times \log \hat{y} + (1-y)\frac{50}{150} \times \log(1-\hat{y})$$

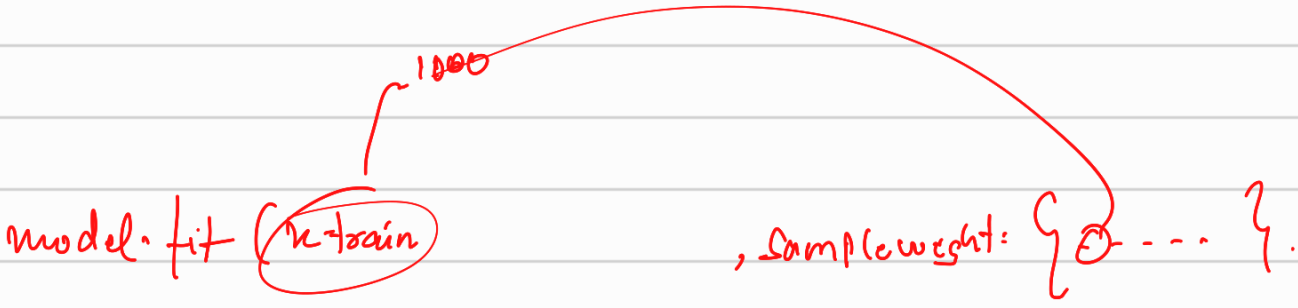
$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



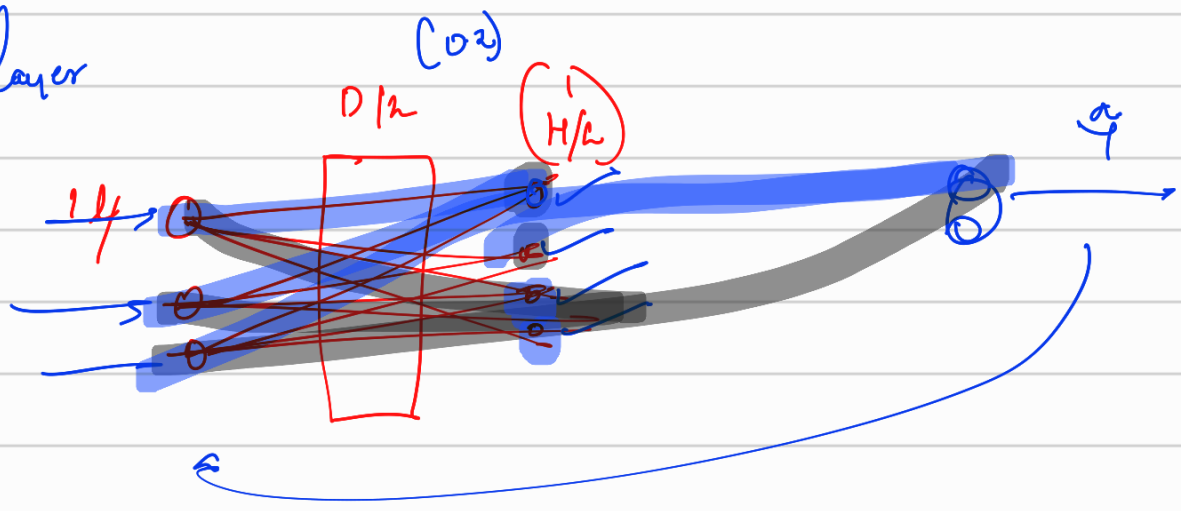
Sample weight: $\{ (y_i - \hat{y}_i)^2 \ . \}$







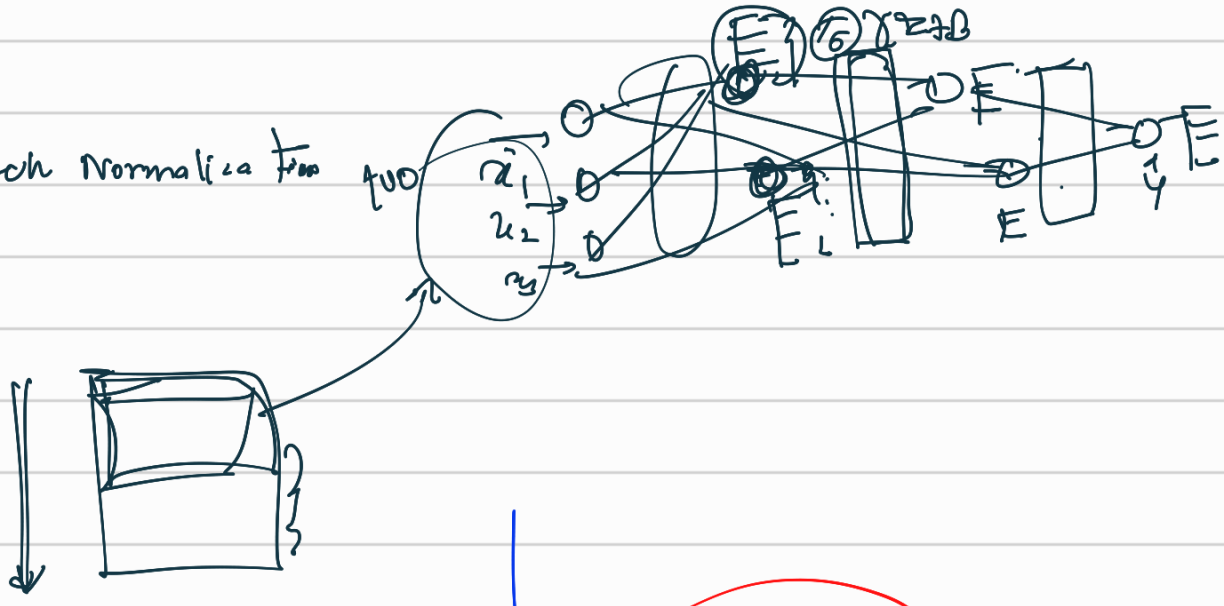
Dropout: layer

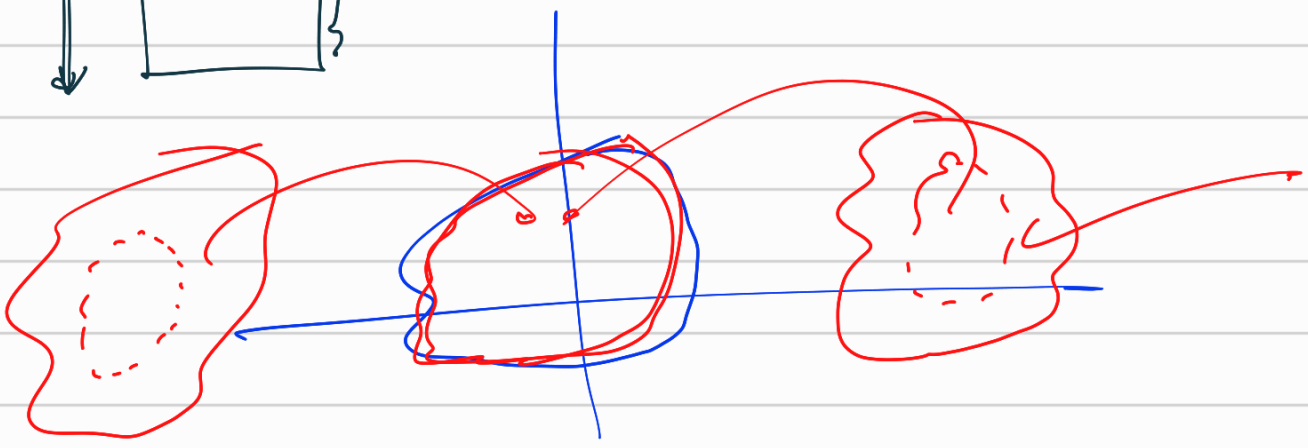


Where you have noise in data and you are building a DNN.

Regularization: model.add Dense(,)

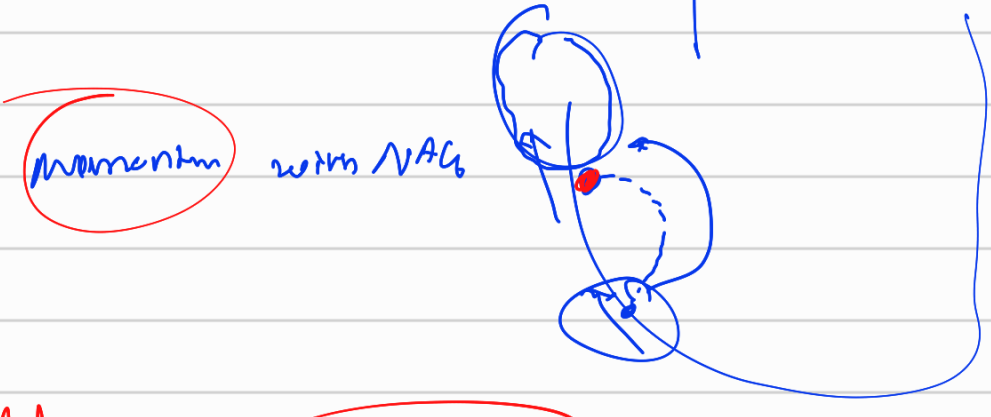
Batch Normalization





optimizers Momentum

momentum with NAG



Adagrad

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \nabla w_t$$

$$v_t = v_{t-1} + (\nabla w_t)^2$$

Rmsprop

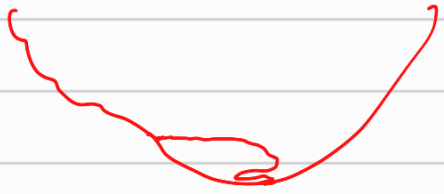
$$v_t = \beta v_{t-1} + (1-\beta)(\nabla w_t)^2$$

Adam: Rms prop + mom

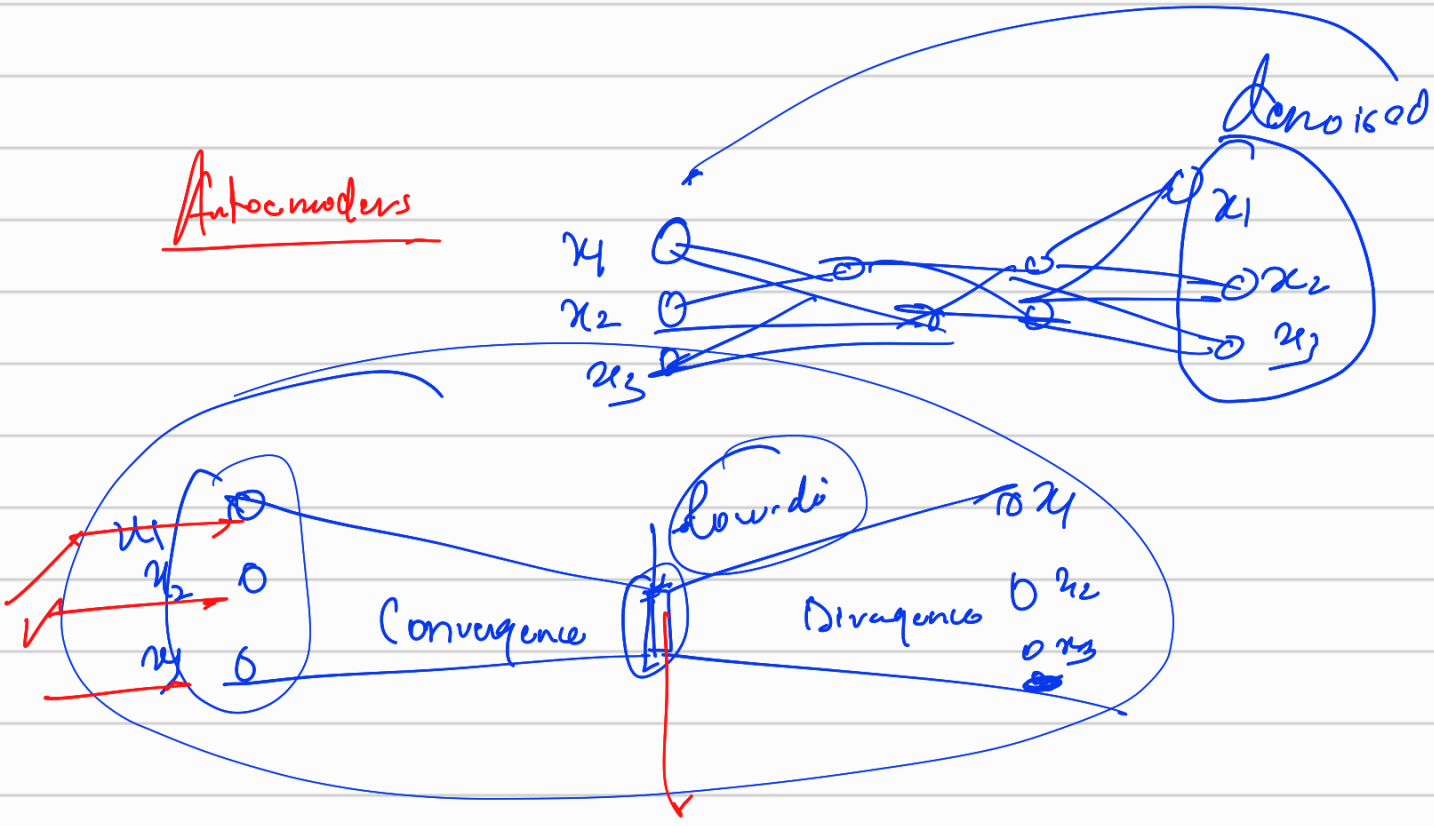
Callbacks : early stopping

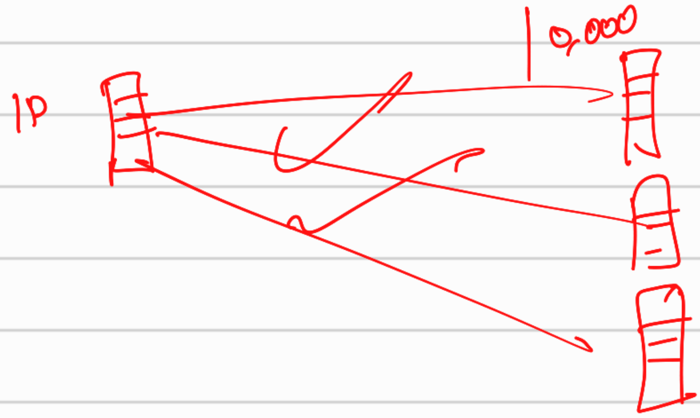
mc: { path; val-loss }

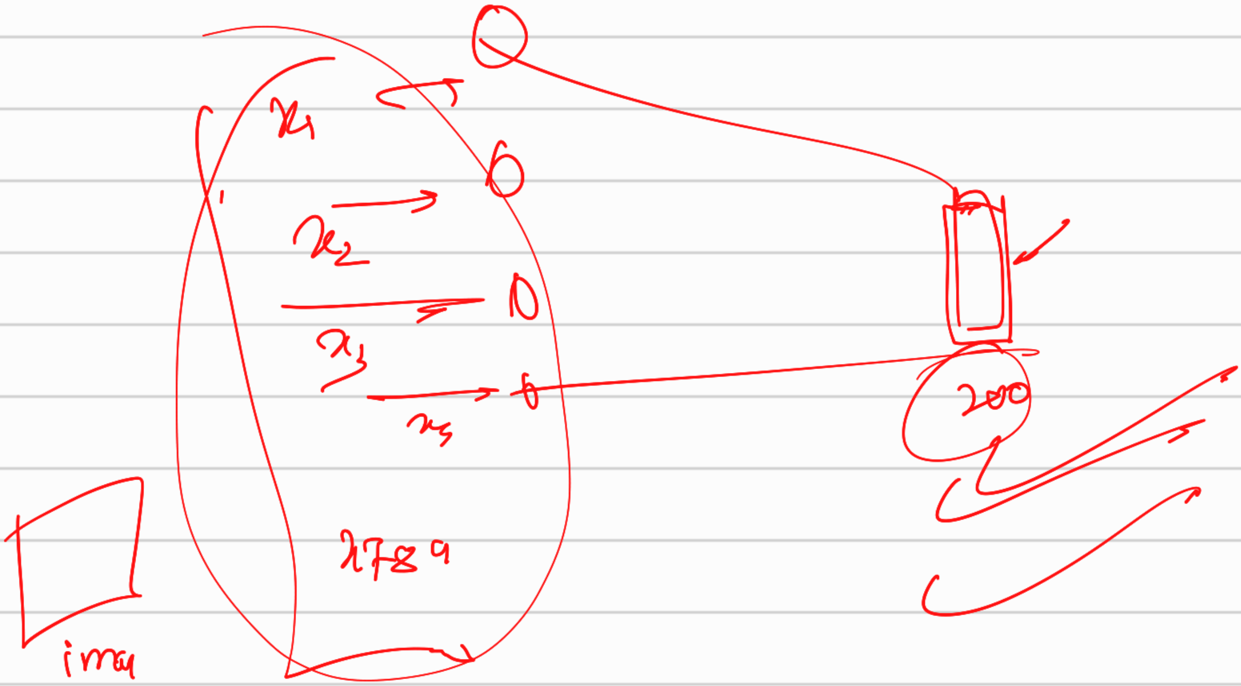
LR



Autoencoders







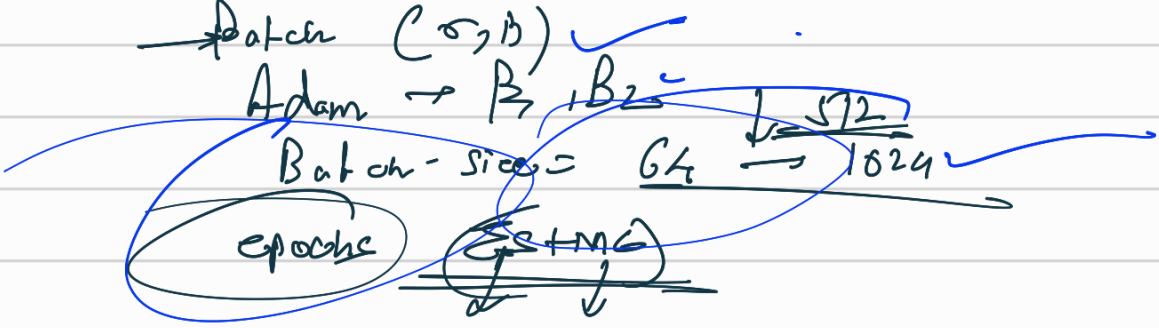
video
Gr
Audi

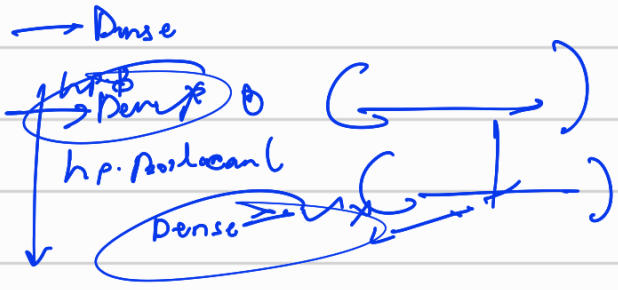
Keras Tuner

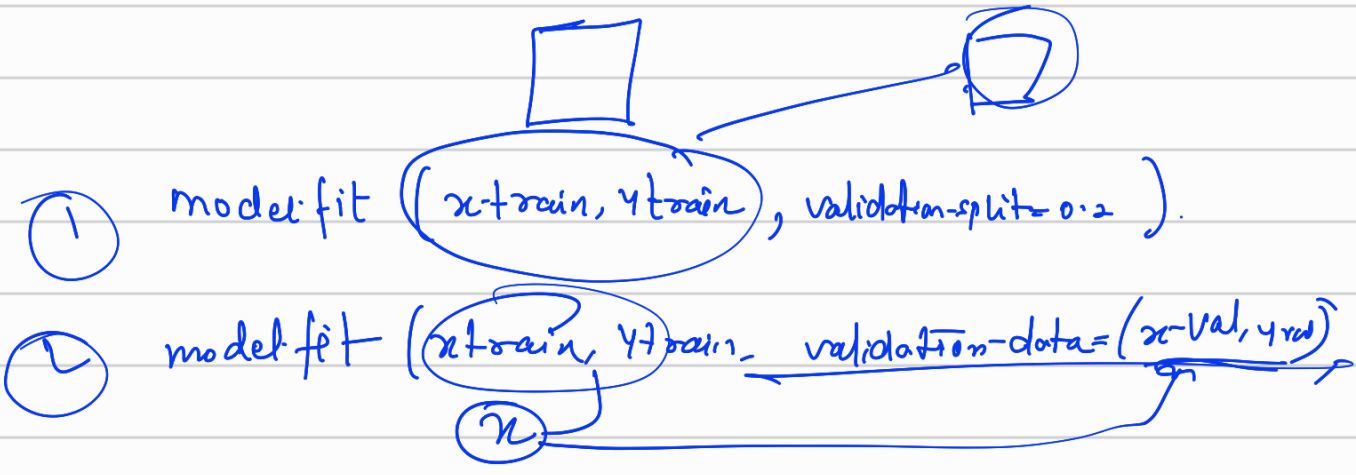
- No of layers
- number of neurons
- learning rate
- regularization
- Dropout → ?

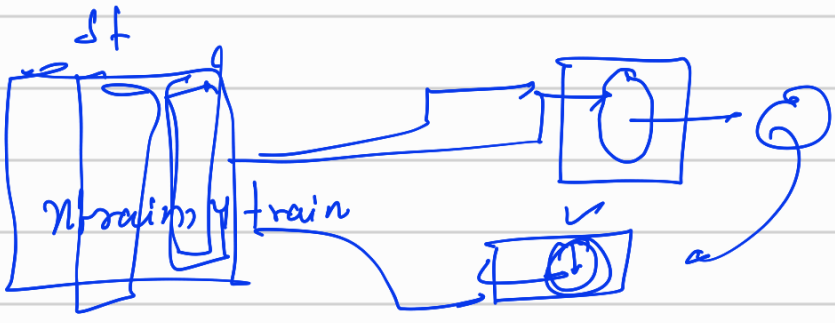
- GsearchCV
- RsearchCV (DT/RF .....)
- Keras Tuner
- (AutoKeras)
- Scikit wrapper (Keras)

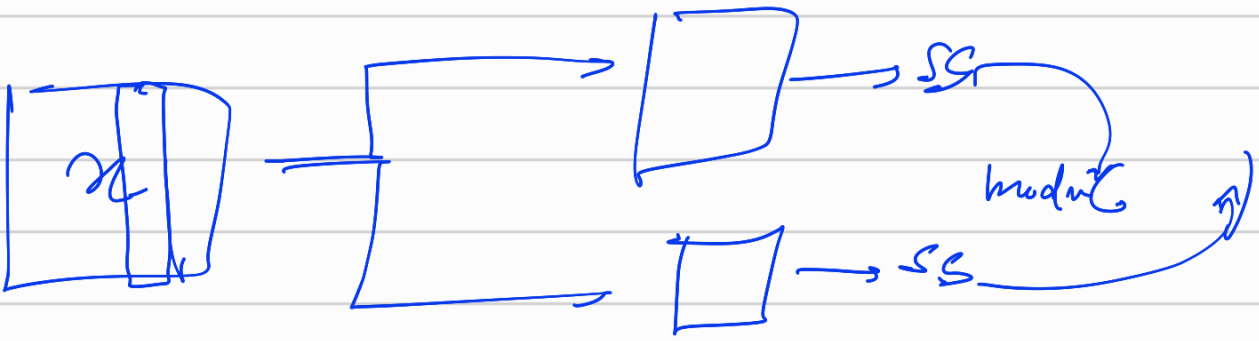
Scikit


Batch (σ, β)
Adam → β1, β2
Batch-size = 64 → 1024
epochs


Dense
hp.Boolean(
Dense

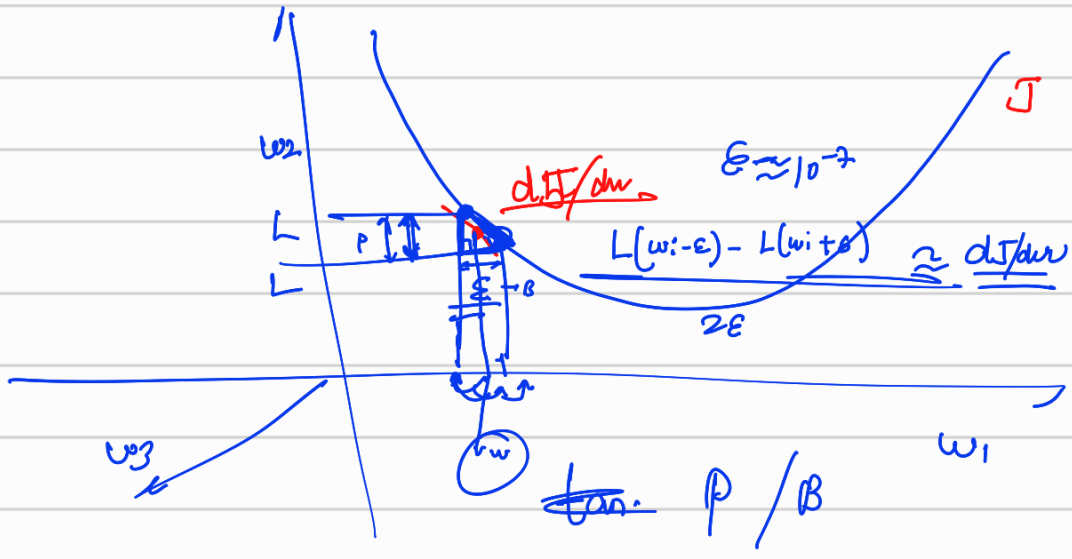

model.fit(x_train, y_train, validation_split=0.2)
model.fit(x_train, y_train, validation_data=(x_val, y_val))


df
x_train
y_train




```
model = Sequential()
model.add(Dense)
model.add(Dense)
model.add(Dense)
model.add(Dense)
```

```
for i in range(n):
    model.add(Dense())
```



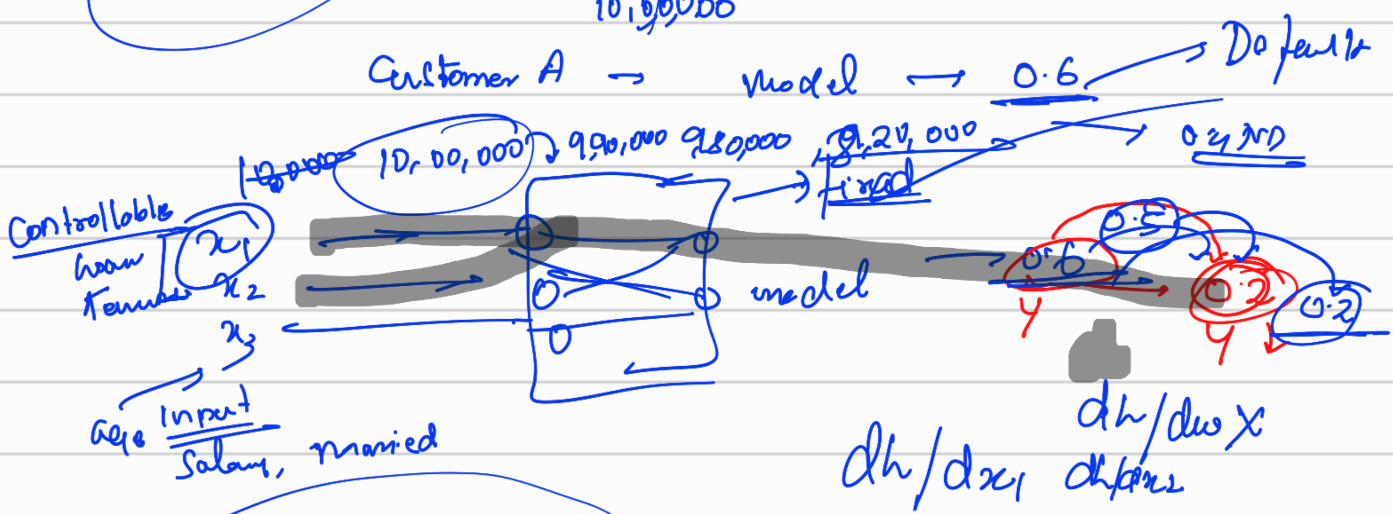
NN as optimization problem

Bank

Loan
Salary
Cscore
EMI
Net

→ model → 0/1

ND ← → D



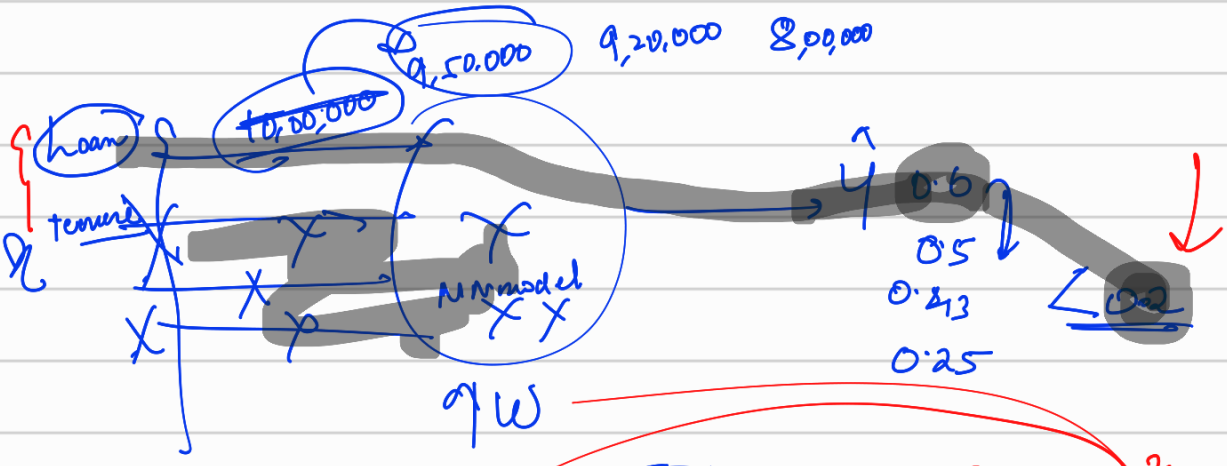
Prediction

$h \sim \hat{y} \sim f(w, x)$

Optimization

$h \sim \hat{y} \sim f(w, x)$

Prediction solved



$$\frac{d(\hat{y})}{dx_1} + \lambda_1 (\text{in} - 10)^2 + \lambda (\text{tenure} - 7)^2$$

Non-linear constrained optimization