Advanced Machine Learning

Halim BENHABILES, Ph.D
Associate professor
DSLS Team, H&E Dept., JUNIA ISEN Lille
CNRS UMR-8520 IEMN
Tél. : 33 (0)3 20 30 40 27
halim.benhabiles@junia.com
halim.benhabiles@iemn.fr

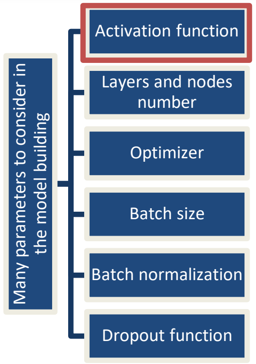
JUNIA ISEN

iemn
Institut d'Electronique, de Microélectronique et de Nanotechnologie
UMR CNRS 8520

Improving the MLP with Keras – TensorFlow core

- Many parameters to consider in the model building
  - Activation function
  - Layers and nodes number
  - Optimizer
  - Batch size
  - Batch normalization
  - Dropout function

Improving the MLP with Keras – TensorFlow core



- Activation function $\sigma(x)$ for **hidden layers**
  - Linear (identity) → x
  - Exponential→ e^x
  - Elu (exponential linear unit) → $\begin{cases} x & if\ \ x > 0 \\ \alpha(e^x - 1) & else \end{cases}$
  - Selu (scaled Elu) → $scale * Elu(x)$
  - Relu (rectified linear unit) → $\max(x, 0)$
  - Sigmoid → $\frac{1}{1+e^{-x}}$
  - Hard_sigmoid → $\begin{cases} 0 & if\ x < -2.5 \\ 1 & if\ \ x > 2.5 \\ 0.2x + 0.5 & if\ \ -2.5 \leq x \leq 2.5 \end{cases}$
  - Tanh → $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
  - Softplus → $\log(e^x + 1)$
  - Softsign → $\frac{x}{|x|+1}$

Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- Batch size
- Batch normalization
- Dropout function

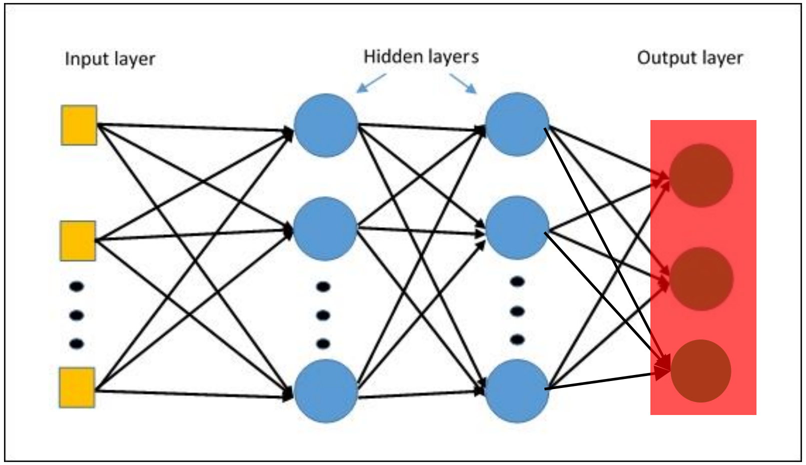
- Activation function $\sigma(x)$ for **hidden layers**
  - Linear (identity) → x
  - Exponential→ e^x
  - Elu (exponential linear unit) → $\begin{cases} x & if \;\; x > 0 \\ \alpha(e^x - 1) & else \end{cases}$
  - Selu (scaled Elu) → $scale * Elu(x)$
  - Relu (rectified linear unit) → $\max(x, 0)$
  - Sigmoid → $\frac{1}{1+e^{-x}}$
  - Hard_sigmoid → $\begin{cases} 0 & if \; x < -2.5 \\ 1 & if \;\; x > 2.5 \\ 0.2x + 0.5 & if \;\; -2.5 \leq x \leq 2.5 \end{cases}$
  - Tanh → $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
  - Softplus → $\log(e^x + 1)$
  - Softsign → $\frac{x}{|x|+1}$

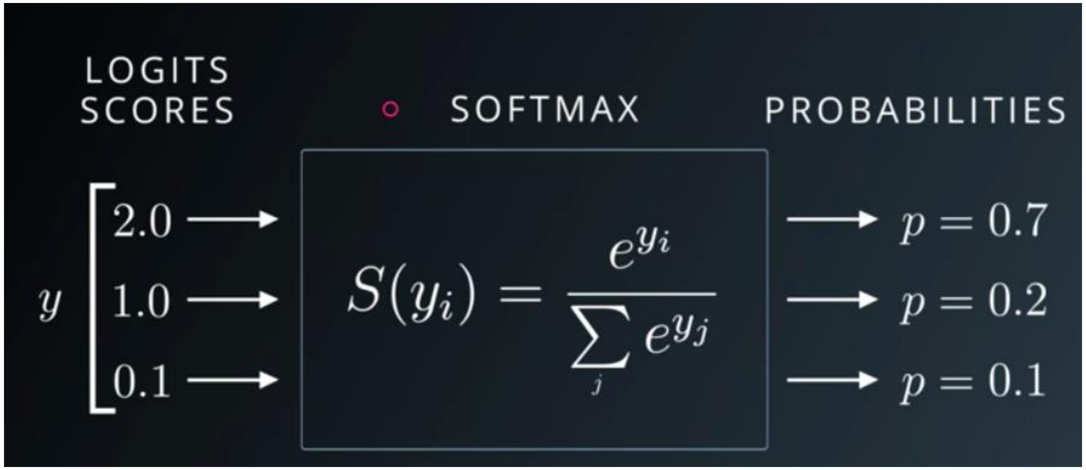
Most used → relu, elu, sigmoid

Improving the MLP with Keras – TensorFlow core

- Softmax function for **output layer → Transform ouput values into probabilities with sum=1 over the the layer**

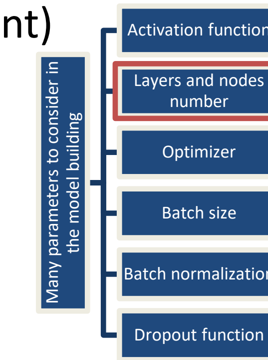
Many parameters to consider in the model building
- **Activation function**
- Layers and nodes number
- Optimizer
- Batch size
- Batch normalization
- Dropout function



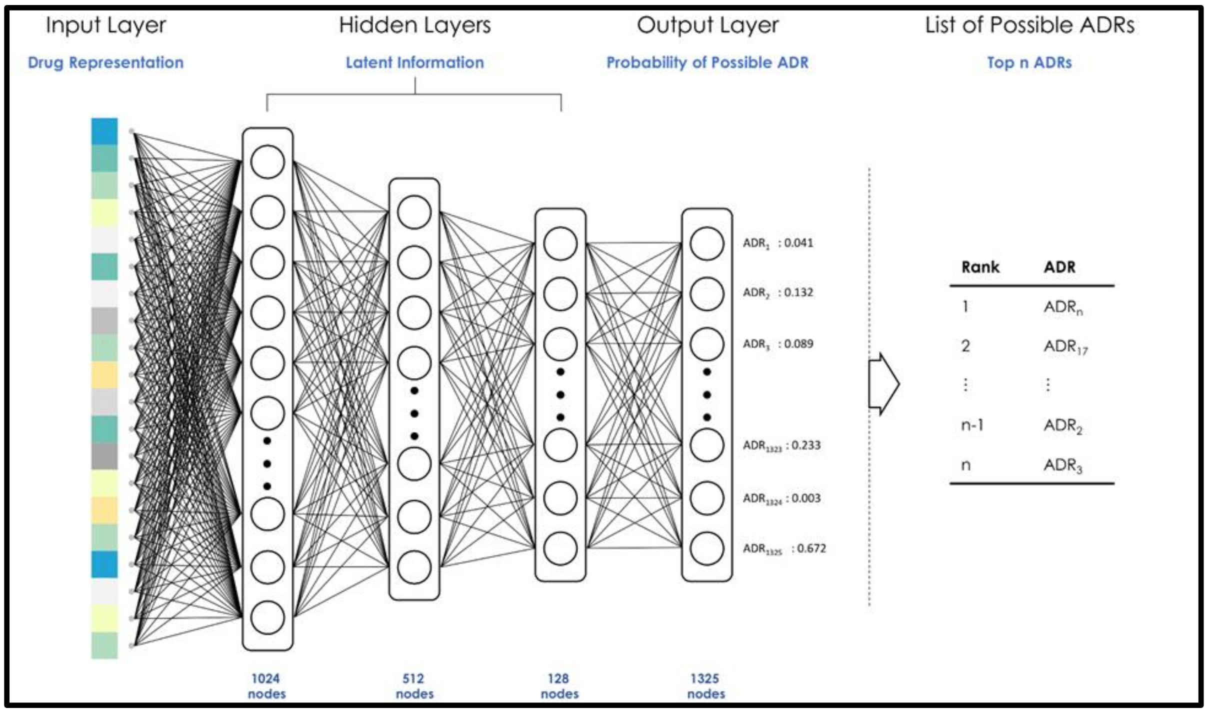


https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

Improving the MLP with Keras – TensorFlow core

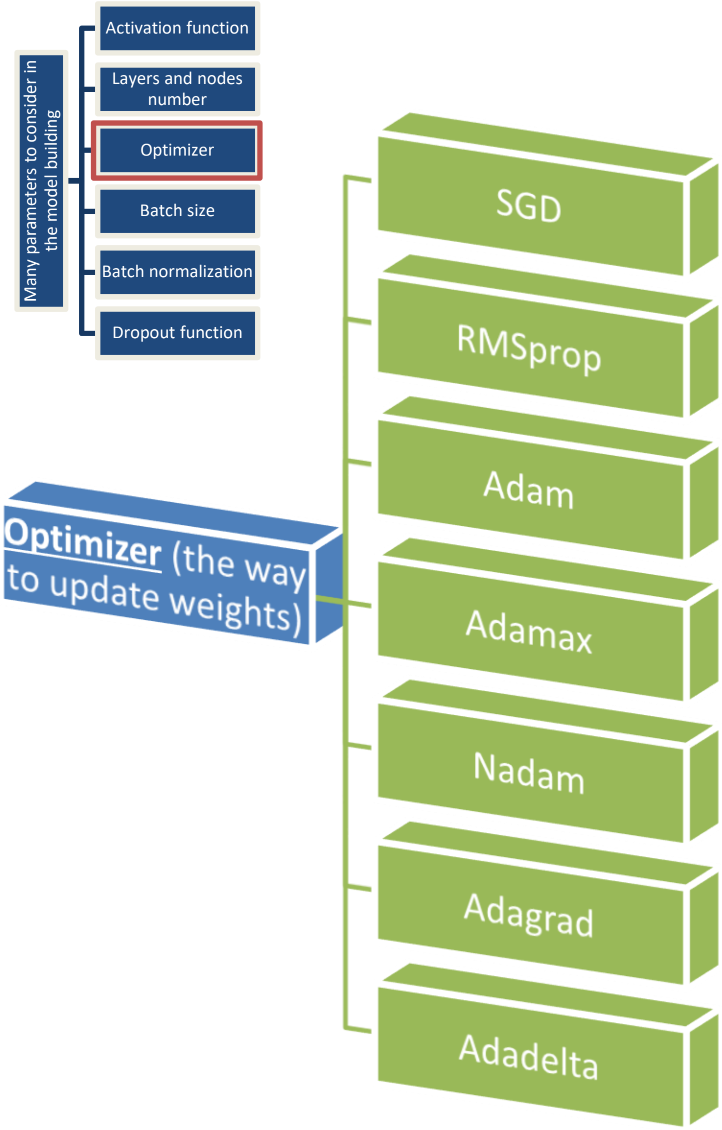


- Setting number of layers and nodes / layer (no exact answer – problem dependent)
  - Input layer → identical to the shape of your data
  - Output layer → number of classes to fix for **classification problem**
  - Hidden layers
    - Large number of layers for non linear data (time and memory limitation)
    - The first hidden layer smaller than the input layer
    - Decreasing size of layers until to reach the output (very often power of 2)



Detecting Potential Adverse Drug Reactions Using a Deep Neural Network Model - https://www.jmir.org/2019/2/e11016/pdf

Improving the MLP with Keras – TensorFlow core



Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- Batch size
- Batch normalization
- Dropout function

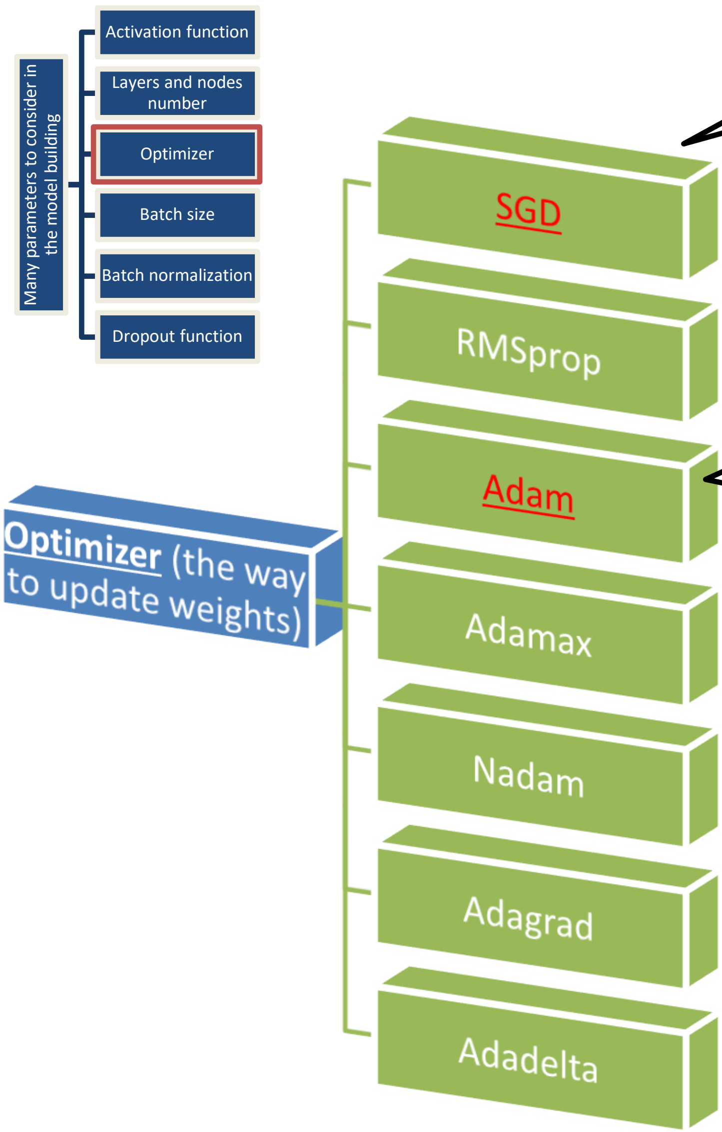
Optimizer (the way to update weights)

- SGD
- RMSprop
- Adam
- Adamax
- Nadam
- Adagrad
- Adadelta

Stochastic Gradient Descent

$$w_{ij} = w_{ij} - \eta * \delta_j * x_i$$

Improving the MLP with Keras – TensorFlow core



Stochastic Gradient Descent

$$w_{ij} = w_{ij} - \eta * \delta_j * x_i$$

Adaptive Moment Optimization

$$w_{ij} = w_{ij} - \eta\left(\frac{m_t}{\sqrt{v_t + \epsilon}}\right), \text{ where}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) * \delta_t,\ \beta_1 = 0.9,\ m_0 = 0$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * \delta_t^2,\ \beta_2 = 0.999,\ v_0 = 0$$

Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- Batch size
- Batch normalization
- Dropout function

Optimizer (the way to update weights)

- SGD
- RMSprop
- Adam
- Adamax
- Nadam
- Adagrad
- Adadelta

Stochastic Gradient Descent

$$w_{ij} = w_{ij} - \eta * \delta_j * x_i$$

Root Mean Square propagation

$$w_{ij} = w_{ij} - \left(\frac{\eta}{\sqrt{v_t + \epsilon}} * \delta_j\right), \text{ where}$$

$$v_t = \beta v_{t-1} + (1 - \beta) * \delta_t^2,\ \beta = 0.9,\ v_0 = 0$$

Adaptive Moment Optimization

$$w_{ij} = w_{ij} - \eta\left(\frac{m_t}{\sqrt{v_t + \epsilon}}\right), \text{ where}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) * \delta_t,\ \beta_1 = 0.9,\ m_0 = 0$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * \delta_t^2,\ \beta_2 = 0.999,\ v_0 = 0$$

Variants of Adam

Adaptive Subgradient optimization

$$w_{ij} = w_{ij} - \left(\frac{\eta}{\sqrt{\Delta_t^2 + \epsilon}} * \delta_j\right), \text{ where}$$

$$\Delta_t^2 = \sum_{\tau=1}^{t} \delta_\tau^2$$

Variant of Adagrad

Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- **Batch size**
- Batch normalization
- Dropout function

Batch size

Number of examples to propagate before updating weights. Generally power of 2: from 8→128

1

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - Take an example (X,c) from S ← **1 example**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * (c - \sigma_i)$

    $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

Weights update

Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- **Batch size**
- Batch normalization
- Dropout function

Batch size

Number of examples to propagate before updating weights. Generally power of 2: from 8→128

1

All - N

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - Take an example (X,c) from S ← **1 example**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * (c - \sigma_i)$

    $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - ~~Take an example~~ (X,c) from S ← **N examples**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * (c - \sigma_i)$

    $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1 - \sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

Weights update

Improving the MLP with Keras – TensorFlow core

Many parameters to consider in the model building

- Activation function
- Layers and nodes number
- Optimizer
- **Batch size**
- Batch normalization
- Dropout function

Batch size

Number of examples to propagate before updating weights. Generally power of 2: from 8→128

1

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - Take an example (X,c) from S ← **1 example**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * (c-\sigma_i)$

  $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

All - N

- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - ~~Take an example~~ (X,c) from S ← **N examples**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * (c-\sigma_i)$

  $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

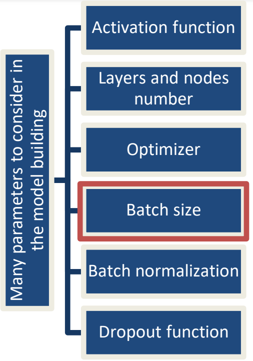
1< M <N

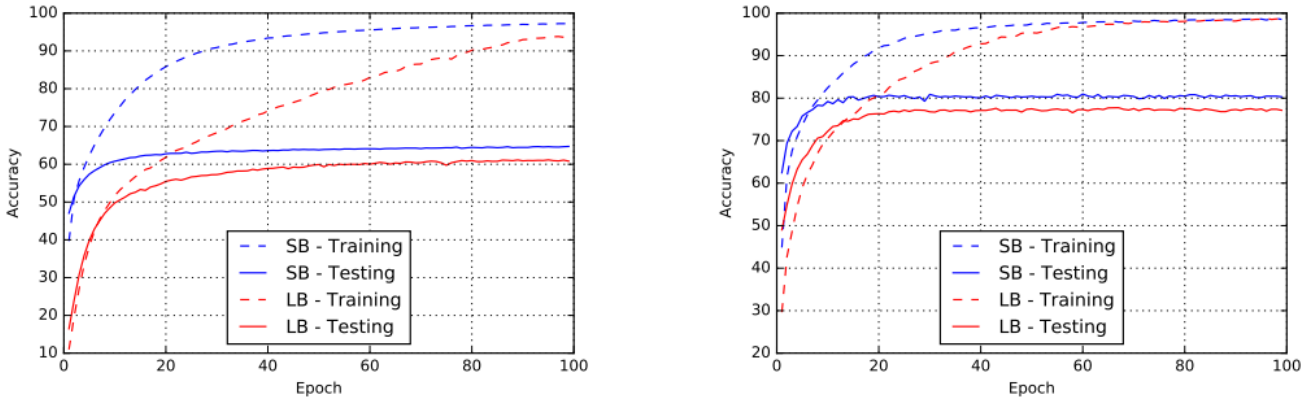
- Set randomly weights w_{ij} in [-0.5, 0.5]
- Repeat
  - ~~Take an example~~ (X,c) from S ← **M examples**
  - Calculate the outputs σ
  - For each output neuron i
    - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * (c-\sigma_i)$

  $\sigma'(x)$ the derivative of sigmoid log in this example
  - For each hidden layer (q-1 to 1) [backpropagation]
    - For each hidden neuron i of current layer
      - $\delta_i = \underbrace{\sigma_i * (1-\sigma_i)}_{\sigma'(x)} * \sum_{k \in succ(i)} \delta_k * w_{ik}$
  - For each weight w_{ij} [forwardpropagation]
    - $w_{ij} = w_{ij} - \eta * \delta_j * x_i$
- EndRepeat

Weights update

Improving the MLP with Keras – TensorFlow core





(a) F_2 (b) C_1

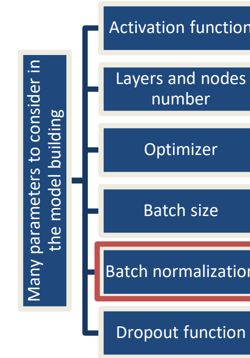
Figure 2: Convergence trajectories of training and testing accuracy for SB and LB methods

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

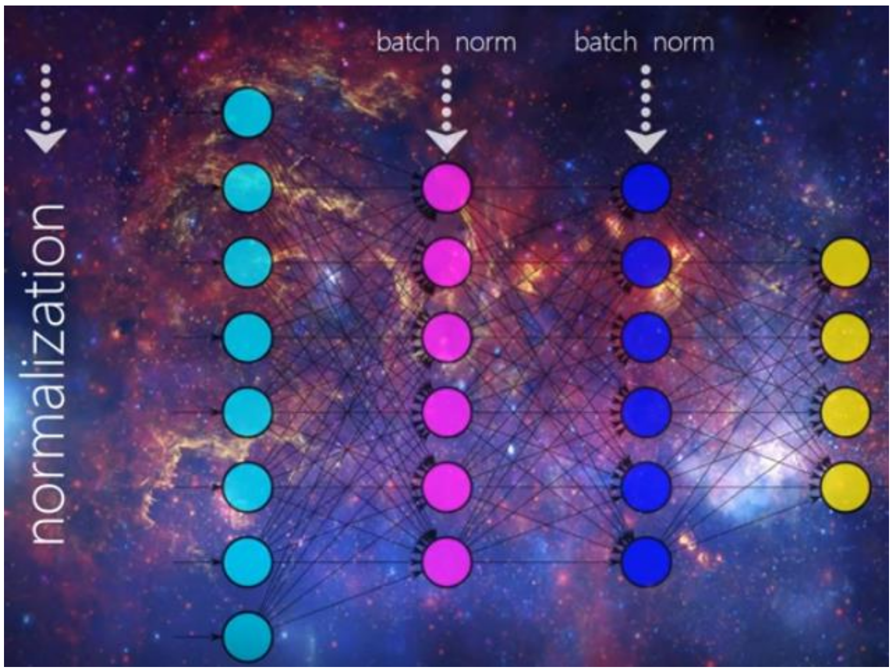
	Training Accuracy		Testing Accuracy	
Network Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99 \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

https://stats.stackexchange.com/questions/164876/what-is-the-trade-off-between-batch-size-and-number-of-iterations-to-train-a-neu

Improving the MLP with Keras – TensorFlow core



- **Batch normalization**
  - **Comes after each dense hidden layer**



https://www.youtube.com/watch?v=dXB-KQYkzNU

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

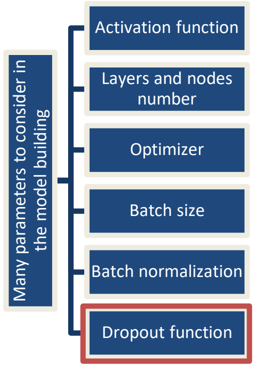
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

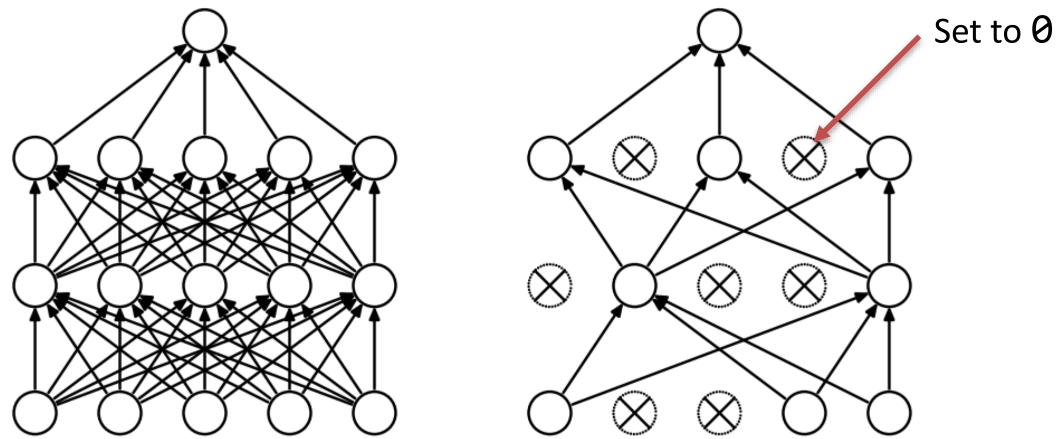
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift - https://arxiv.org/pdf/1502.03167v3.pdf

Improving the MLP with Keras – TensorFlow core



- **Dropout function**
  - **Comes after batch normalization**



(a) Standard Neural Net (b) After applying dropout.

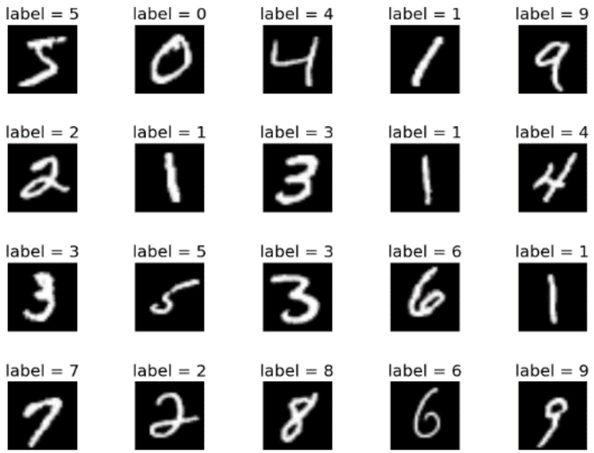
Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Decreasing probability p through layers. From 0.8- 0.95 → 0.1

Dropout: A Simple Way to Prevent Neural Networks from Overfitting - http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Lab session Keras – TensorFlow core

- Build an MLP to solve the hand written digits problem
  - Use MNIST dataset



MNIST database of handwritten digits

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

Usage:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

https://keras.io/datasets/#mnist-database-of-handwritten-digits

Lab session Keras – TensorFlow core

- Build an MLP to solve the hand written digits problem
  - Use MNIST dataset

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.utils import to_categorical

# Load data
mnist = tf.keras.datasets.mnist

# the data, split between train and validation sets
(x_train, y_train), (x_validation, y_validation) = mnist.load_data()

# convert images into one dimension from 28x28 pixels
x_train = x_train.reshape(60000, 784)
x_validation = x_validation.reshape(10000, 784)
x_train = x_train.astype('float32')
x_validation = x_validation.astype('float32')

# normalize into [0,1]
x_train /= 255
x_validation /= 255
print('train samples', x_train.shape)
print('test samples', x_validation.shape)

print('train label samples', y_train.shape)
print('test label samples', y_validation.shape)
```

Lab session Keras – TensorFlow core

- Save your best model during training and keep history of training metrics

```
from tensorflow.keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='model.hdf5', monitor='val_loss',  verbose=1, save_best_only=True)

# compile the model with adam
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# train model with fit for 10 epochs
history = model.fit(x_train, y_train, validation_data = (x_validation, y_validation),epochs=10,callbacks = [checkpointer])
```

Lab session Keras – TensorFlow core

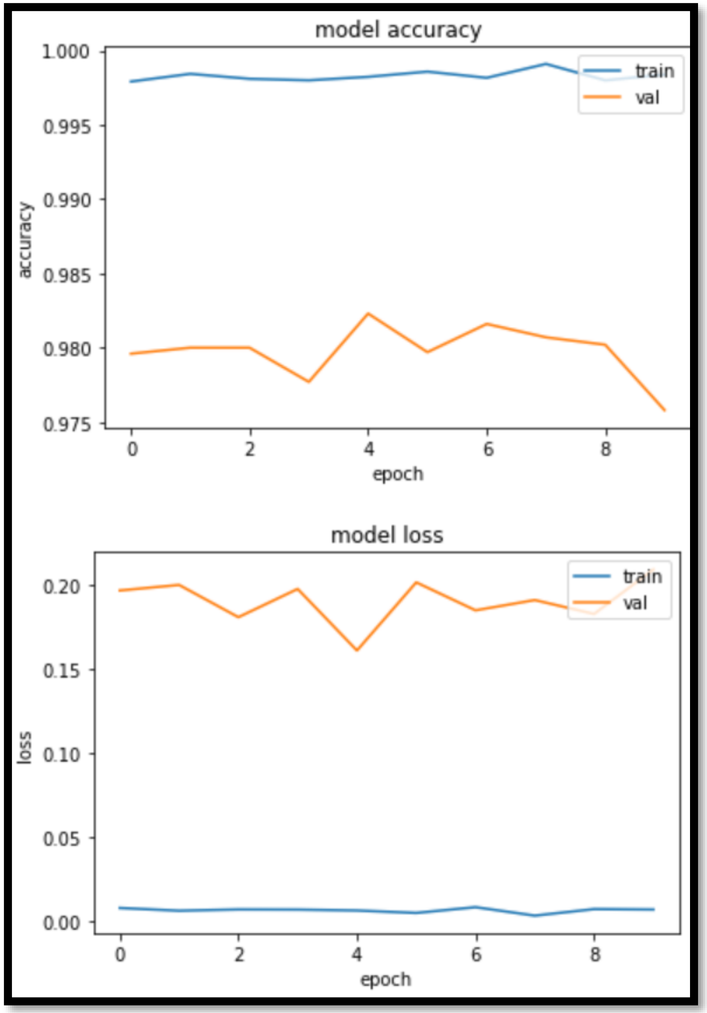
- Using the categorical cross entropy loss 'categorical_crossentropy’
  https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class
  - Transform your label data in one_hot representation

```
# if we want to use categorical_crossentropy loss instead of sparse we have to convert the "y" labels
from tensorflow.keras.utils import to_categorical

y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_validation = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

Lab session Keras – TensorFlow core

- Show training and validation curves

```
from matplotlib import pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```



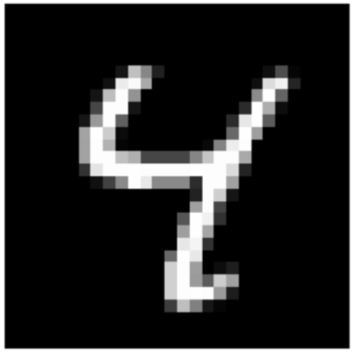
Lab session Keras – TensorFlow core

- Load model and test it on an image

```
from tensorflow.keras.models import load_model

model = load_model('model.hdf5')
```

```
import cv2
from matplotlib import pyplot as plt


image = x_validation[6].reshape(28,28)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')

datapoint = x_validation[6].reshape(1,784)
predict_prob = model.predict(datapoint)
print('I am confindent around {:.2f}% that this image corresponds to digit {}'.format(np.amax(predict_prob)*100,
                                                                                     np.argmax(predict_prob)))
```

```
I am confindent around 100.00% that this image corresponds to digit 4
```



Lab session Keras – TensorFlow core

- Give the performance of your MLP model (same architecture) by
  - Using the categorical cross entropy loss 'categorical_crossentropy’ https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class
  - Testing three batch sizes: 16, 32, 64
  - Testing two optimizers: SGD, ADAM
  - with/without batch normalization and dropout
  - Show training and validation accuracy/loss curves for each scenario
- If your machine is not equipped with a GPU
  - Use the online Google Colaboratory platform. In the ''Edit'' option activate a GPU https://colab.research.google.com/notebooks/intro.ipynb?hl=en