Rapport TP 1

INF8225

par: Kacem Khaled

1. Partie I

Le but de la partie 1 de ce TP est de se familiariser avec les réseaux Bayésiens et de la librairie Numpy.

Rappel des formules utilisées dans les démonstrations mathématiques:

Réseaux Bayesiens

$$Pr(A_1, A_2, .., A_N) = \prod_{i=1}^{N} Pr(A_i|Parents(Ai)) \qquad (1)$$

Marginalisation

$$Pr(X1) = \sum_{\{X\}\setminus X1} Pr(X_1, X_2, \ldots, X_N) \qquad (2)$$

Conditionnement

$$Pr(A|B) = \frac{Pr(A, B)}{Pr(B)} \qquad (3)$$

Pour chaque question, on faira le développement jusqu'à aboutir à une expression qui peut être écrite directement en Python

a) Calcul de : Pr(H=1)

Selon (1):

$$Pr(P, W, A, H) = Pr(W|P).Pr(P).Pr(H|A, P).Pr(A) \qquad (4)$$

Selon (2):

$$Pr(H = 1) = \sum_{P} \sum_{A} \sum_{W} Pr(H = 1, P, A, W) \qquad (5)$$

Selon (4) et (5):

$$Pr(H = 1) = \sum_{P} \sum_{A} \sum_{W} Pr(W|P).Pr(P).Pr(H = 1|A, P).Pr(A) \qquad (5)$$

---

b) Calcul de Pr(H=1|W=1)

Selon (3):

$$Pr(H = 1|W = 1) = \frac{Pr(H = 1, W = 1)}{Pr(W = 1)} \qquad (6)$$

Selon (2):

$$Pr(W = 1) = \sum_{P} \sum_{A} \sum_{H} Pr(H, P, A, W = 1) \qquad (7)$$

$$Pr(H = 1, W = 1) = \sum_{P} \sum_{A} Pr(H = 1, P, A, W = 1) \qquad (8)$$

Selon (6), (7) et (8):

$$Pr(H = 1|W = 1) = \frac{\sum_P \sum_A Pr(H = 1, P, A, W = 1)}{\sum_P \sum_A \sum_H Pr(H, P, A, W = 1)} \qquad (9)$$

Selon (4) et (9):

$$Pr(H = 1|W = 1) = \frac{\sum_P \sum_A Pr(W = 1|P).Pr(P).Pr(H = 1|A, P).Pr(A)}{\sum_P \sum_A \sum_H Pr(W = 1|P).Pr(P).Pr(H|A, P).Pr(A)} \qquad (10)$$

---

c) Calcul de Pr(H=1|W=0)

Par analogie avec (10):

$$Pr(H = 1|W = 0) = \frac{\sum_P \sum_A Pr(W = 0|P).Pr(P).Pr(H = 1|A, P).Pr(A)}{\sum_P \sum_A \sum_H Pr(W = 0|P).Pr(P).Pr(H|A, P).Pr(A)} \qquad (11)$$

---

d) Calcul de Pr(H=1|P=0,W=1)

Selon (3):

$$Pr(H=1|P=0,W=1)=\frac{Pr(H=1,P=0,W=1)}{Pr(P=0,W=1)} \qquad (12)$$

Selon (2):

$$Pr(P=0,W=1)=\sum_A\sum_H Pr(H,P=0,A,W=1) \qquad (13)$$

$$Pr(H=1,P=0,W=1)=\sum_A Pr(H=1,P=0,A,W=1) \qquad (14)$$

Selon (12), (13) et (14):

$$Pr(H=1|P=0,W=1)=\frac{\sum_A Pr(H=1,P=0,A,W=1)}{\sum_A\sum_H Pr(H,P=0,A,W=1)} \qquad (15)$$

Selon (4) et (15):

$$Pr(H=1|P=0,W=1)=\frac{\sum_A Pr(W=1|P=0).Pr(P=0).Pr(H=1|A,P=0).Pr(A)}{\sum_A\sum_H Pr(W=1|P=0).Pr(P=0).Pr(H|A,P=0).Pr(A)} \qquad (16)$$

---

e) Calcul de Pr(W=1|H=1)

Selon (3):

$$Pr(W=1|H=1)=\frac{Pr(W=1,H=1)}{Pr(H=1)} \qquad (17)$$

Selon (2):

$$Pr(H=1)=\sum_P\sum_A\sum_W Pr(H=1,P,A,W) \qquad (18)$$

$$Pr(W=1,H=1)=\sum_P\sum_A Pr(H=1,P,A,W=1) \qquad (19)$$

Selon (17), (18) et (19):

$$Pr(H=1|W=1)=\frac{\sum_P\sum_A Pr(H=1,P,A,W=1)}{\sum_P\sum_A\sum_W Pr(H=1,P,A,W)} \qquad (20)$$

Selon (4) et (9):

$$Pr(W=1|H=1)=\frac{\sum_P\sum_A Pr(W=1|P).Pr(P).Pr(H=1|A,P).Pr(A)}{\sum_P\sum_A\sum_H Pr(W|P).Pr(P).Pr(H=1|A,P).Pr(A)} \qquad (21)$$

---

f) Calcul de Pr(W=1|H=1,A=1)

Selon (3):

$$Pr(W=1|H=1,A=1)=\frac{Pr(W=1,H=1,A=1)}{Pr(H=1,A=1)} \qquad (22)$$

Selon (2):

$$Pr(H=1,A=1)=\sum_P\sum_W Pr(H=1,P,A=1,W) \qquad (23)$$

$$Pr(W=1,H=1,A=1)=\sum_P Pr(H=1,P,A=1,W=1) \qquad (24)$$

Selon (22), (23) et (24):

$$Pr(W=1|H=1,A=1)=\frac{\sum_P Pr(H=1,P,A=1,W=1)}{\sum_P\sum_W Pr(H=1,P,A=1,W)} \qquad (25)$$

Selon (4) et (25):

$$Pr(W=1|H=1,A=1)=\frac{\sum_P Pr(W=1|P).Pr(P).Pr(H=1|A=1,P).Pr(A=1)}{\sum_P\sum_W Pr(W|P).Pr(P).Pr(H=1|A=1,P).Pr(A=1)} \qquad (26)$$

In [40]:

```python
import numpy as np
import matplotlib.pyplot as plt

# les arrays sont batis avec les dimensions suivantes:
# pluie, arroseur, watson, holmes
# et chaque dimension: faux, vrai

prob_pluie = np.array([0.8, 0.2]).reshape(2, 1, 1, 1)
print("Pr(Pluie)={}\n".format(np.squeeze(prob_pluie)))
prob_arroseur = np.array([0.9, 0.1]).reshape(1, 2, 1, 1)
print("Pr(Arroseur)={}\n".format(np.squeeze(prob_arroseur)))
watson = np.array([[0.8, 0.2], [0, 1]]).reshape(2, 1, 2, 1)
print("Pr(Watson|Pluie)={}\n".format(np.squeeze(watson)))
holmes = np.array([[1, 0],[0.1, 0.9],[0,1], [0, 1]]).reshape(2, 2, 1, 2)
print("Pr(Holmes|Pluie,arroseur)={}\n".format(np.squeeze(holmes)))
# prob watson mouille - pluie
print("Pr(W = 1|P = 0) = {}\n".format(np.squeeze(watson[0,:,1,:])))
# prob gazon watson mouille
print("Pr(W = 1) = {:.3f}\n".format(((watson * prob_pluie).sum(0).squeeze()[1])))
# prob gazon holmes mouille si arroseur - pluie
print("Pr(H = 1|A = 1, P = 0) = {}\n".format(holmes[0,1,0,1]))
# prob gazon holmes mouille
print("a) Pr(H = 1) = {:.3f}\n".format(
    (prob_pluie * prob_arroseur * watson * holmes)[:,:,:,1].sum()
))
print("b) Pr(H = 1|W = 1) = {:.3f}\n".format(
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,1,1].sum()) /
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,1,:].sum())
))
print("c) Pr(H = 1|W = 0) = {:.3f}\n".format(
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,0,1].sum()) /
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,0,:].sum())
))
print("d) Pr(H = 1|P = 0, W = 1) = {:.3f}\n".format(
    ((prob_pluie * prob_arroseur * watson * holmes)[0,:,1,1].sum()) /
    ((prob_pluie * prob_arroseur * watson * holmes)[0,:,1,:].sum())
))
print("e) Pr(W = 1|H = 1) = {:.3f}\n".format(
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,1,1].sum()) /
    ((prob_pluie * prob_arroseur * watson * holmes)[:,:,:,1].sum())
))
print("f) Pr(W = 1|H = 1, A = 1) = {:.3f}\n".format(
    ((prob_pluie * prob_arroseur * watson * holmes)[:,1,1,1].sum()) /
    ((prob_pluie * prob_arroseur * watson * holmes)[:,1,:,1].sum())
))
```

```
Pr(Pluie)=[0.8 0.2]

Pr(Arroseur)=[0.9 0.1]

Pr(Watson|Pluie)=[[0.8 0.2]
 [0.  1. ]]

Pr(Holmes|Pluie,arroseur)=[[[1.  0. ]
  [0.1 0.9]]

 [[0.  1. ]
  [0.  1. ]]]

Pr(W = 1|P = 0) = 0.2

Pr(W = 1) = 0.360

Pr(H = 1|A = 1, P = 0) = 0.9

a) Pr(H = 1) = 0.272

b) Pr(H = 1|W = 1) = 0.596

c) Pr(H = 1|W = 0) = 0.090

d) Pr(H = 1|P = 0, W = 1) = 0.090

e) Pr(W = 1|H = 1) = 0.788

f) Pr(W = 1|H = 1, A = 1) = 0.374
```

2. Partie II

2.1 La régression logistique et le calcul du gradient

In [0]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from sklearn import datasets
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()
X = digits.data
```

Nous utilisons l'astuce de redéfinir

$$X_{bias_{(N*(L+1))}} = \begin{bmatrix} & 1 \\ & 1 \\ X_{(N*L)} & .. \\ & 1 \end{bmatrix}$$

- N: nombre des exemples
- L: nombre des features
- K: nombre des classes

In [0]:
```
X_bias = np.concatenate((X,np.ones((X.shape[0], 1))), axis=1)
```

In [43]:
```
# Visualisation des dimensions N x (L+1)
X_bias.shape
```

Out[43]: (1797, 65)

In [0]:
```
y = digits.target
y_one_hot = np.zeros((y.shape[0], len(np.unique(y))))
y_one_hot[np.arange(y.shape[0]), y] = 1  # one hot target or shape NxK
```

In [45]:
```
# Visualisation des dimensions N x K
y_one_hot.shape
```

Out[45]: (1797, 10)

Splitting dataset

In [0]:
```
X_train, X_test, y_train, y_test = train_test_split(X_bias, y_one_hot, test_size=0.3, random_state=42)
X_test, X_validation, y_test, y_validation = train_test_split(X_test, y_test, test_size=0.5, random_sta
te=42)
```

Softmax

$$\hat{p_x}^k = softmax(z)_k = \frac{exp(z_k)}{\sum_{1<j<K} exp(z_j)}$$

Fonction de coût Log loss

Soit la fonction de coût log loss (ou cross entropy):

$$L(\Theta) = \frac{-1}{N} \sum_{1<i<N} \sum_{1<k<K} y_k^i log(\hat{y}_{pred_k}^i)$$

avec:

- **K** le nombre de classes
- **N** le nombre d'exemples dans les données
- $\hat{y}_{pred_k}^i$ la probabilité que l'exemple i soit de la classe k
- y_k^i vaut 1 si la classe cible de l'exemple i est k, 0 sinon

Gradient de la fonction de coût

Sous **forme matricielle**, on peut écrire le **gradient de L par rapport à** Θ:

$$\Delta L(\Theta) = \frac{1}{N}(\hat{y}_{pred} - \tilde{y})^T * X_{bias}$$

Mise à jour des poids

Quand le gradient a été calculé, il faut mettre à jour les poids avec ces gradients.

$$\Theta = \Theta - lr. \Delta L(\Theta)$$

avec:

- Θ la matrice de poids, avec $\Theta = [W \quad b]$
- lr le taux d'apprentissage
- $\Delta L(\Theta)$ le gradient de $L(\Theta)$ selon Θ

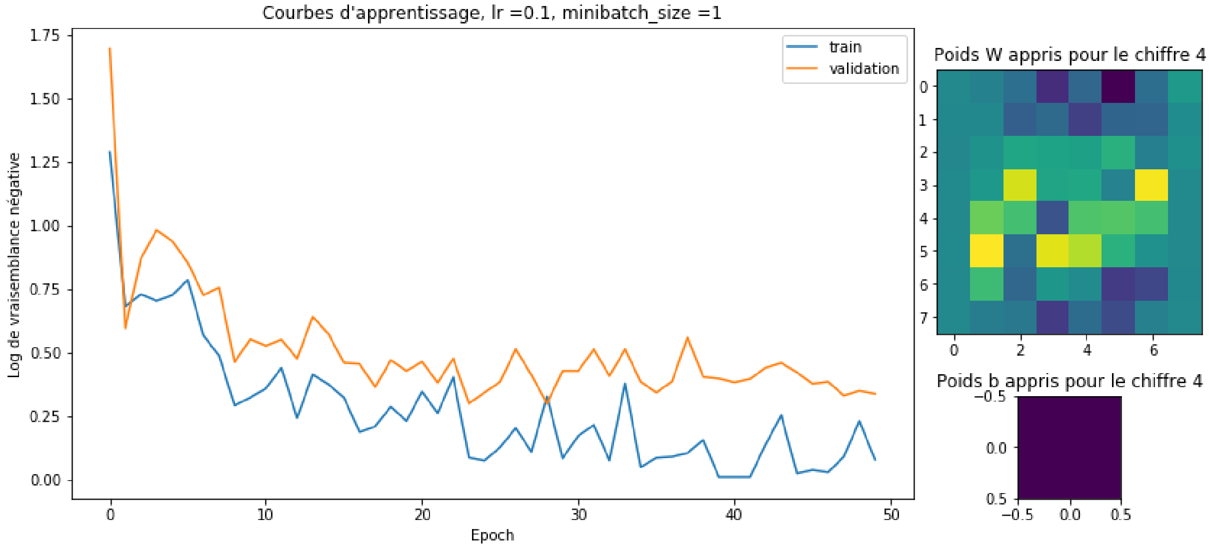
In [0]:

```
def softmax(x):
    expZ = np.exp(x-np.max(x, axis=1, keepdims=True))
    return np.divide(expZ, np.sum(expZ, axis=1, keepdims=True))

def get_accuracy(X, y, W):
    return np.sum(np.argmax(softmax(X @ W.T), axis=1) == np.argmax(y, axis=1)) / len(y)


def get_grads(y, y_pred, X):
    return (1/len(y)) * np.dot((y_pred - y).T, X)


def get_loss(y, y_pred):
    eps=1.0e-5
    y_pred = np.clip(y_pred, eps, 1 - eps) # to prevent dividing by zero
    return (-1 / len(y)) * np.sum(y * np.log(y_pred))

def fit(X_train,y_train,lr,minibatch_size):
    # generate matrix of weights
    #W = np.random.normal(0, 0.01, (len(np.unique(y)), X.shape[1]))  # weights of shape KxL
    Theta = np.random.normal(0, 0.01, (len(np.unique(y)), X.shape[1]+1))  # weights of shape KxL+1

    #best_W = None
    best_theta = None
    best_accuracy = 0
    nb_epochs = 50
    losses_train = []
    losses_val = []
    accuracies = []
    for epoch in range(nb_epochs):
        loss = 0
        accuracy = 0

        for i in range(0, X_train.shape[0], minibatch_size):
            X_train_mini = X_train[i:i + minibatch_size] if (i + minibatch_size < X_train.shape[0]) \
                else X_train[i:X_train.shape[0]]
            y_train_mini = y_train[i:i + minibatch_size] if (i + minibatch_size < X_train.shape[0]) \
                else y_train[i:X_train.shape[0]]
            # feedforward
            y_pred = softmax(np.dot(X_train_mini, Theta.T)) # (y_train_mini * Theta * X_train_mini.T)
            Theta = Theta - lr * get_grads(y_train_mini, y_pred, X_train_mini)

        # compute the loss on the train set
        loss = get_loss(y_train, softmax(np.dot(X_train, Theta.T)))
        losses_train.append(loss)

        # compute the loss on the validation set
        loss = get_loss(y_validation,softmax(np.dot(X_validation, Theta.T)))
        losses_val.append(loss)

        # compute the accuracy on the validation set
        accuracy = get_accuracy(X_validation,y_validation,Theta)
        accuracies.append(accuracy)

        if accuracy > best_accuracy:
            # select the best parameters based on the validation accuracy
            best_accuracy = accuracy
            best_theta = Theta

    accuracy_on_unseen_data = get_accuracy(X_test, y_test, best_theta) # get_accuracy(X_test, y_test, b
est_W)
    print("Accuracy on validation data:{:.10f}\n".format(best_accuracy))
    print("Accuracy on unseen data:{:.10f}\n".format(accuracy_on_unseen_data)) # 0.897506925208
    return losses_train,losses_val, best_theta, best_accuracy, accuracy_on_unseen_data
```

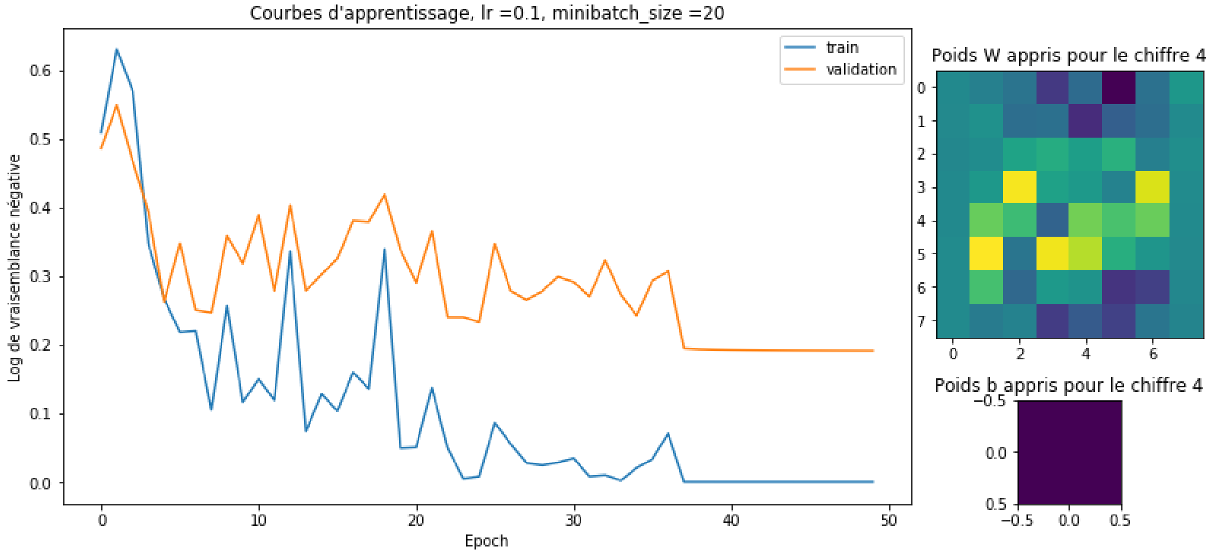
In [48]:

```
#lr = 0.001
#minibatch_size = len(y) // 20
test_nb = 0

chosen_accuracy = 0
chosen_test_accuracy = 0

lrs = [0.1, 0.01, 0.001]
minibatch_sizes = [1, 20, len(y) // 20,  200, 1000]
for lr in lrs:
  for minibatch_size in minibatch_sizes:
    test_nb += 1
    print("Test Number #{},\t lr: {},\t minibatch_size: {}\n".format(test_nb,lr,minibatch_size))
    plt.figure(lrs.index(lr),figsize = (14,6))
    gridspec.GridSpec(4,8)
    losses_train, losses_val, best_theta, best_accuracy, accuracy_on_unseen_data = fit(X_train,y_train,
lr,minibatch_size)

    if best_accuracy > chosen_accuracy:
      chosen_accuracy = best_accuracy
      chosen_test_accuracy = accuracy_on_unseen_data
      chosen_test = test_nb
      chosen_lr = lr
      chosen_minibatch_size = minibatch_size
      chosen_losses_train = losses_train
      chosen_losses_val = losses_val

    best_W = best_theta[:,:64]
    best_b = best_theta[:,64:]

    ax1 = plt.subplot2grid((4,8), (0,0), colspan=6, rowspan=4)
    ax1.plot(losses_train, label="train")
    ax1.plot(losses_val, label="validation")
    ax1.set_title('Courbes d\'apprentissage, lr ='+str(lr)+', minibatch_size ='+str(minibatch_size))
    ax1.set_ylabel('Log de vraisemblance négative')
    ax1.set_xlabel('Epoch')
    ax1.legend(loc='best')

    ax2 = plt.subplot2grid((4,8), (0,6),colspan=2, rowspan=3)
    ax2.imshow(best_W[4, :].reshape(8, 8))
    ax2.set_title('Poids W appris pour le chiffre 4')

    ax3 = plt.subplot2grid((4,8), (3,6),colspan=2, rowspan=1)
    ax3.imshow(best_b[4, :].reshape(1,1))
    ax3.set_title('Poids b appris pour le chiffre 4')

    plt.show()
print('\n\nChosen Test #{} based on the best accuracy on validation data\n\
Accuracy on Validation data= {:.10f}\n\
Accuracy on Test data= {:.10f}\n\
Chosen learning rate = {}\n\
Chosen minibatch size = {}\n'.format(chosen_test,chosen_accuracy,chosen_test_accuracy,chosen_lr,chosen_
minibatch_size))
```

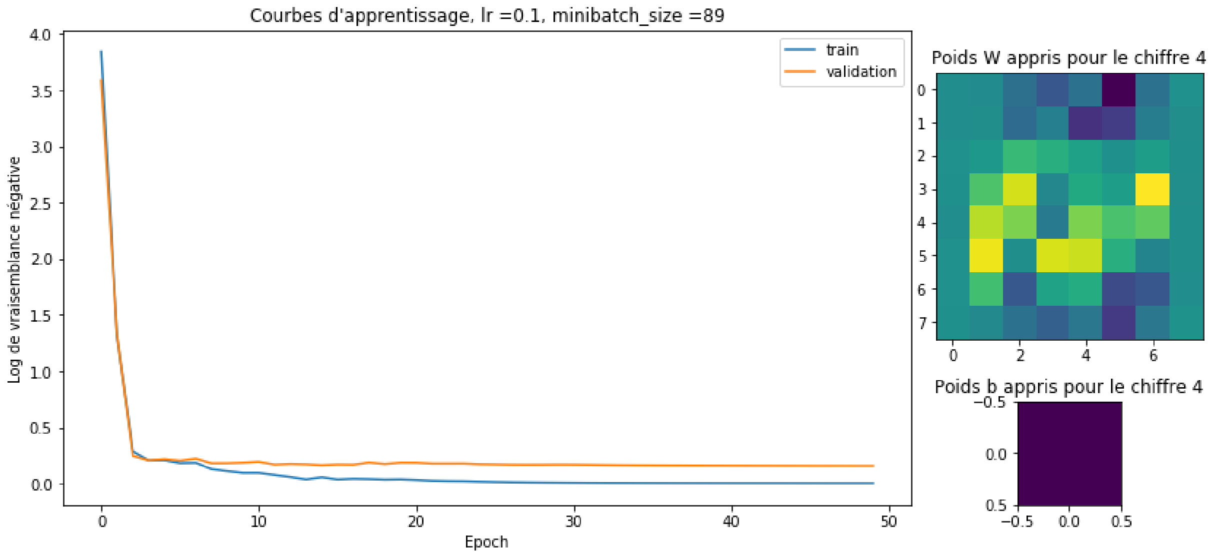
```
Test Number #1,  lr: 0.1,        minibatch_size: 1

Accuracy on validation data:0.9740740741

Accuracy on unseen data:0.9555555556
```



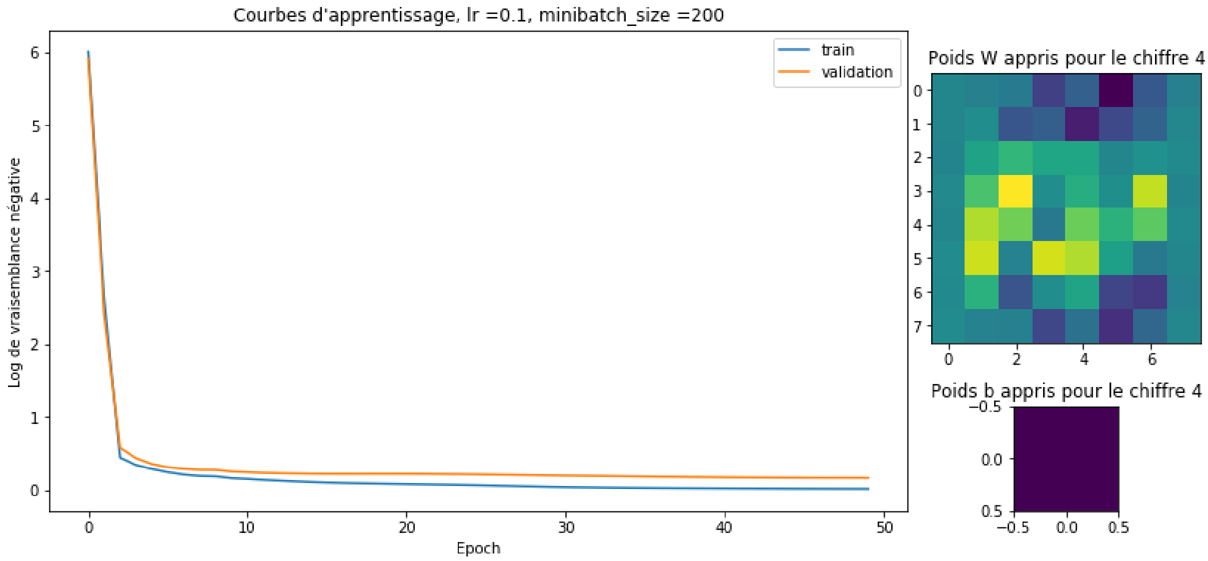

```
Test Number #2,  lr: 0.1,        minibatch_size: 20

Accuracy on validation data:0.9740740741

Accuracy on unseen data:0.9592592593
```



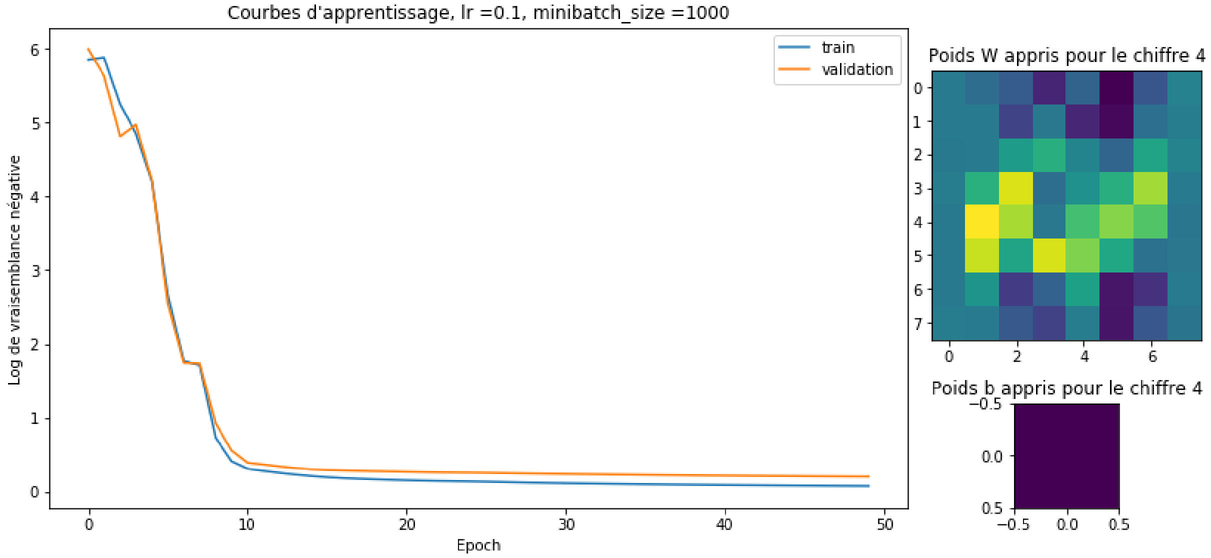

```
Test Number #3,  lr: 0.1,        minibatch_size: 89

Accuracy on validation data:0.9740740741

Accuracy on unseen data:0.9518518519
```




```
Test Number #4,  lr: 0.1,        minibatch_size: 200

Accuracy on validation data:0.9666666667

Accuracy on unseen data:0.9629629630
```



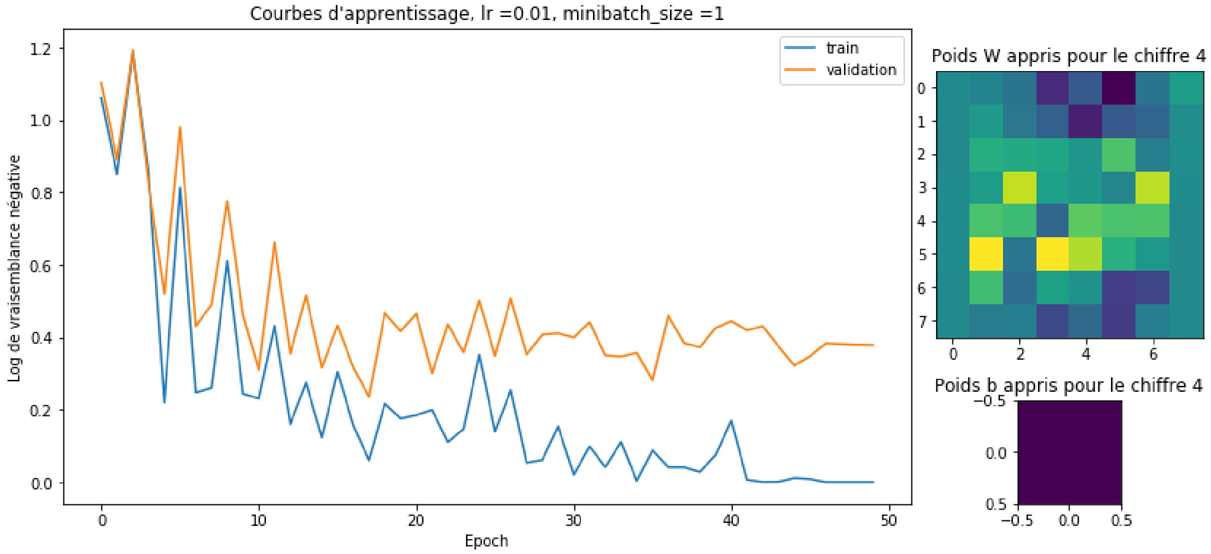

```
Test Number #5,  lr: 0.1,        minibatch_size: 1000

Accuracy on validation data:0.9666666667

Accuracy on unseen data:0.9481481481
```



Test Number #6, lr: 0.01,   minibatch_size: 1

Accuracy on validation data:0.9740740741

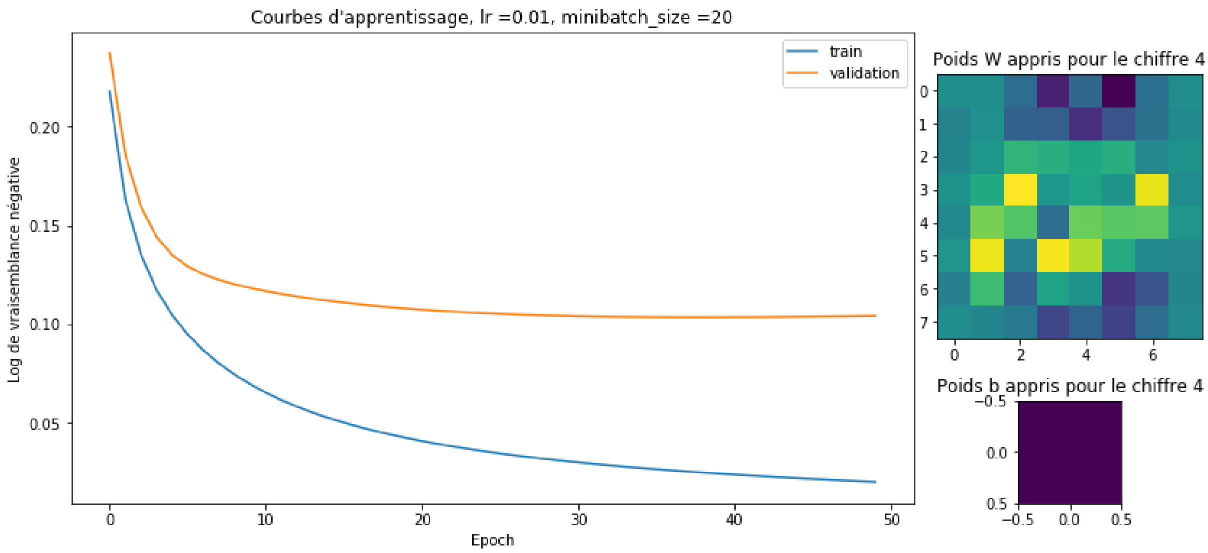
Accuracy on unseen data:0.9703703704



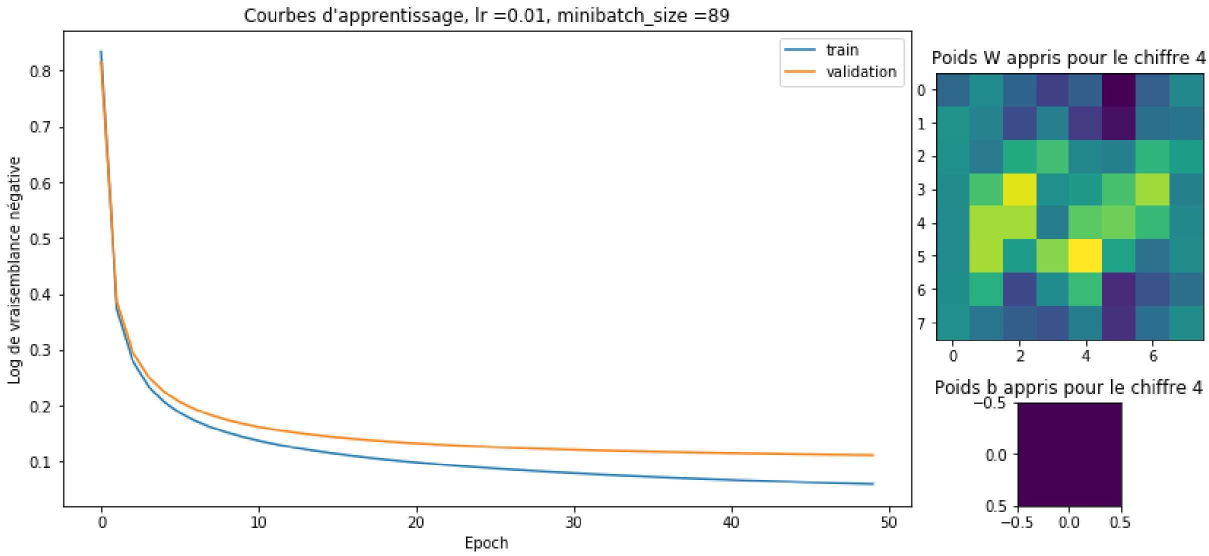
Test Number #7, lr: 0.01,   minibatch_size: 20

Accuracy on validation data:0.9703703704

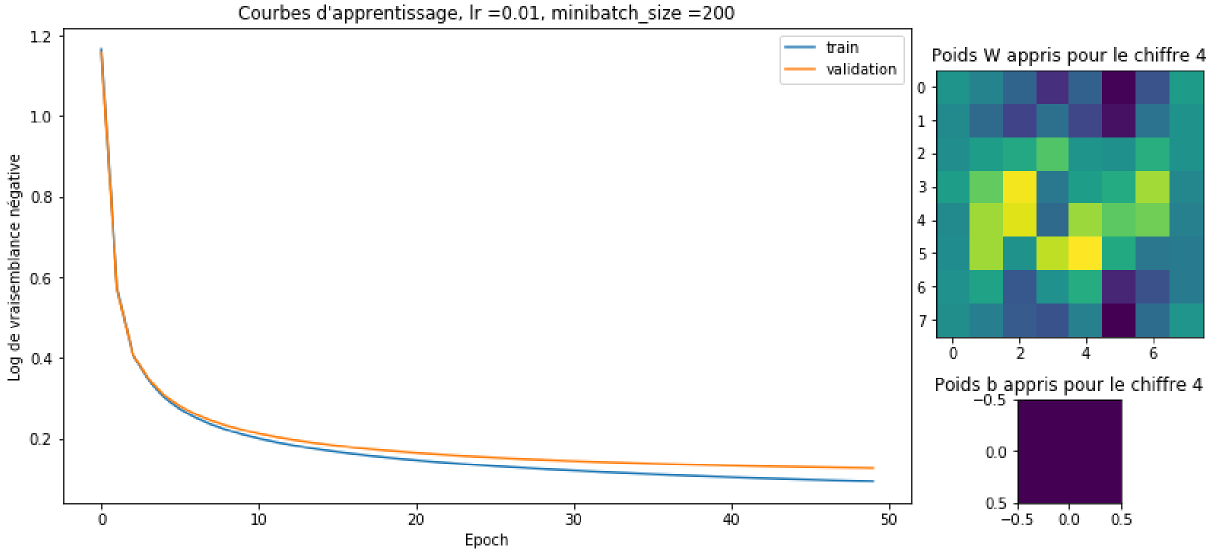
Accuracy on unseen data:0.9555555556



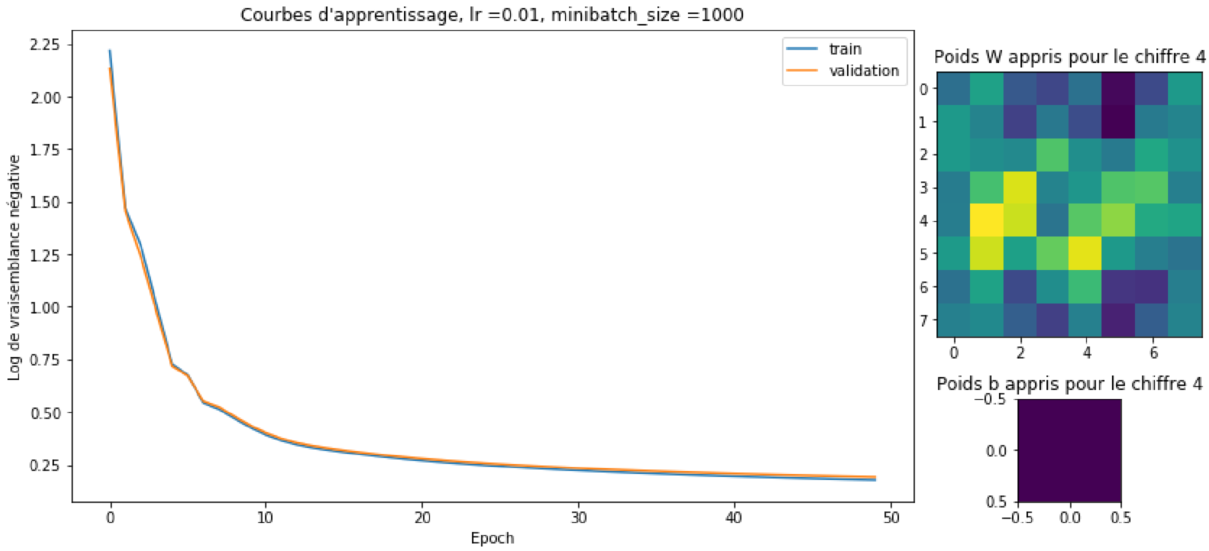

```
Test Number #8,  lr: 0.01,       minibatch_size: 89

Accuracy on validation data:0.9666666667

Accuracy on unseen data:0.9518518519
```



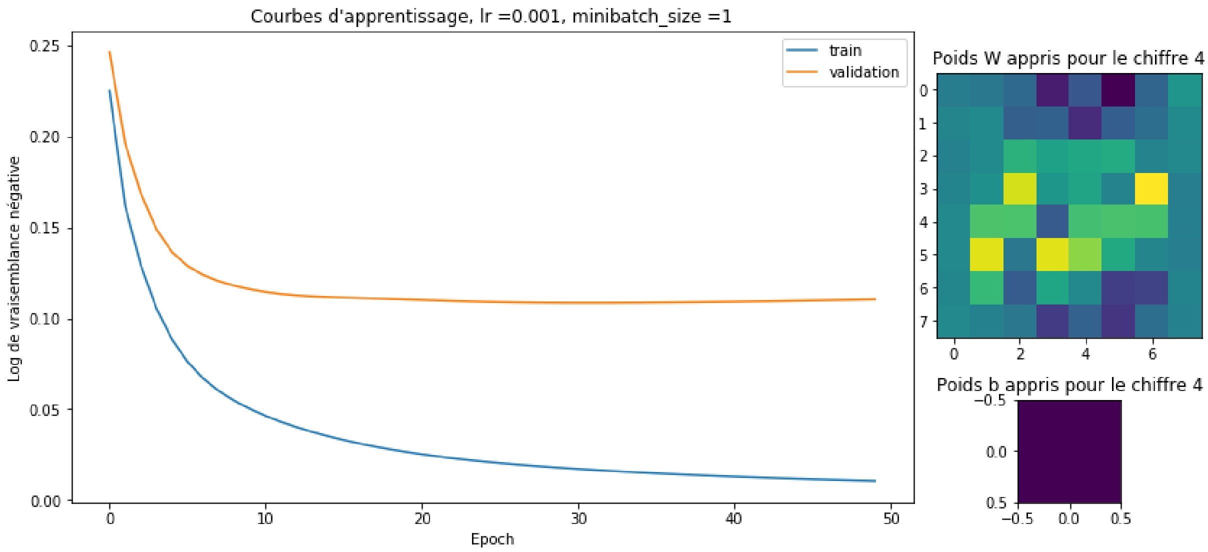

```
Test Number #9,  lr: 0.01,       minibatch_size: 200

Accuracy on validation data:0.9666666667

Accuracy on unseen data:0.9592592593
```



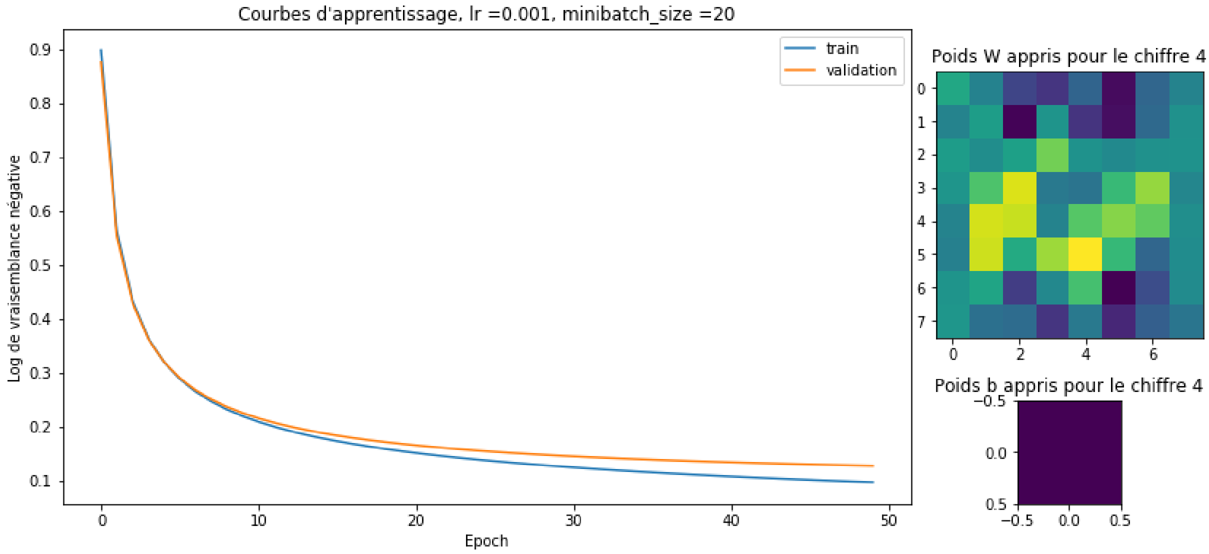

```
Test Number #10,          lr: 0.01,       minibatch_size: 1000

Accuracy on validation data:0.9666666667

Accuracy on unseen data:0.9518518519
```



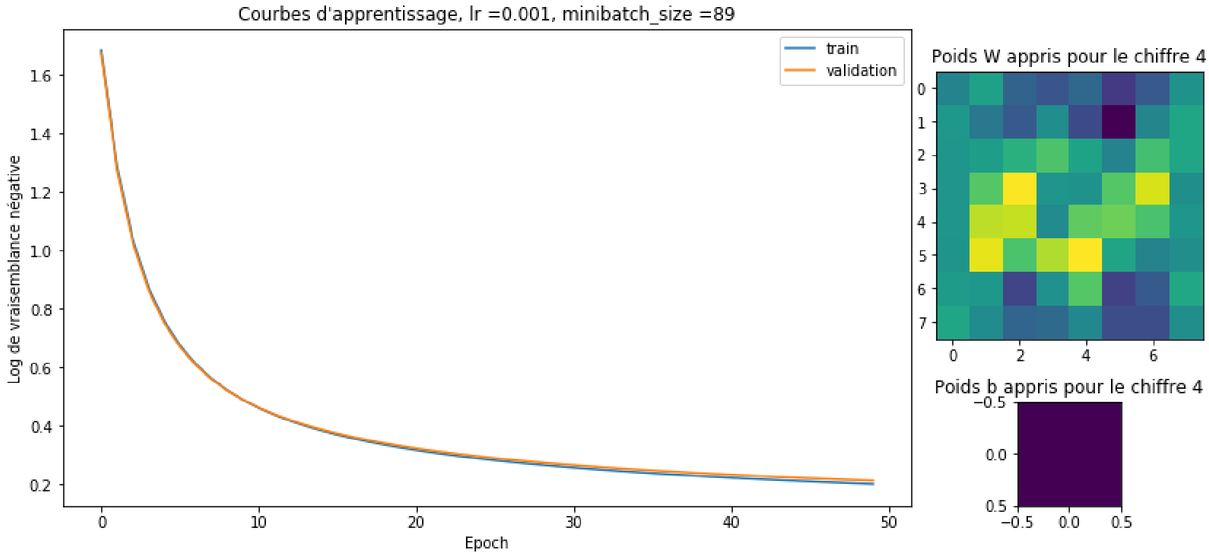

```
Test Number #11,          lr: 0.001,      minibatch_size: 1

Accuracy on validation data:0.9740740741

Accuracy on unseen data:0.9481481481
```



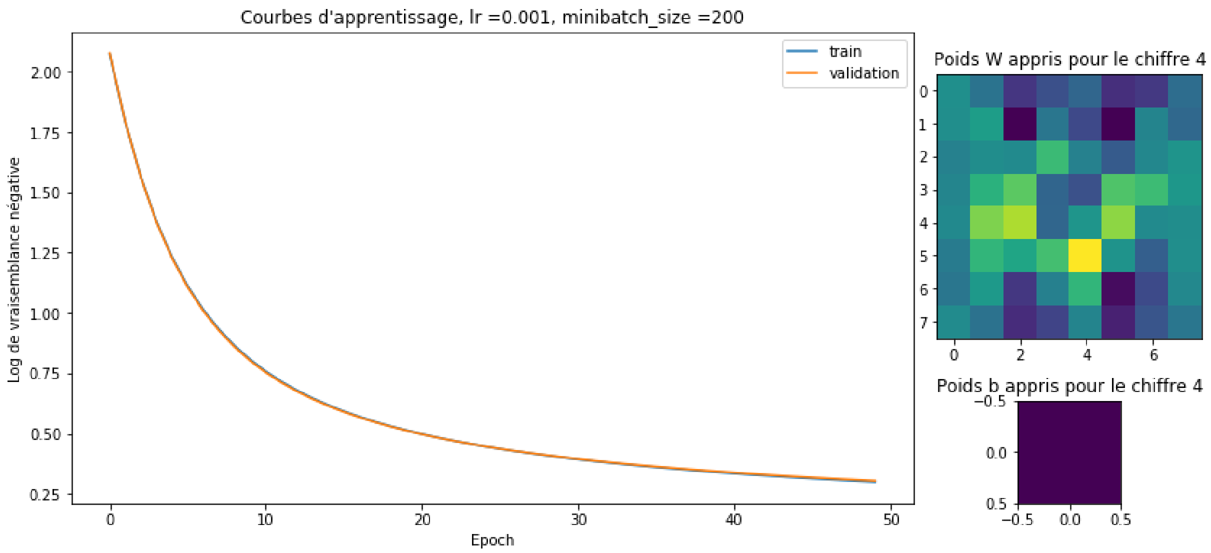
```
Test Number #12,         lr: 0.001,      minibatch_size: 20

Accuracy on validation data:0.9703703704

Accuracy on unseen data:0.9444444444
```



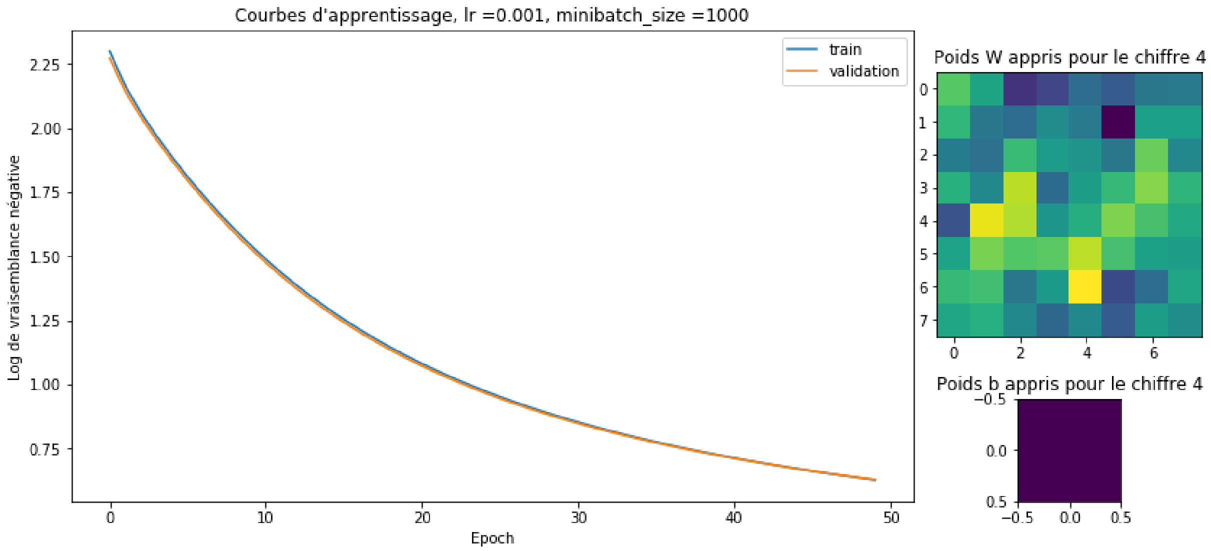

```
Test Number #13,         lr: 0.001,      minibatch_size: 89

Accuracy on validation data:0.9629629630

Accuracy on unseen data:0.9518518519
```




```
Test Number #14,         lr: 0.001,      minibatch_size: 200

Accuracy on validation data:0.9481481481

Accuracy on unseen data:0.9222222222
```




```
Test Number #15,         lr: 0.001,      minibatch_size: 1000

Accuracy on validation data:0.9037037037

Accuracy on unseen data:0.9185185185
```




```
Chosen Test #1 based on the best accuracy on validation data
Accuracy on Validation data= 0.9740740741
Accuracy on Test data= 0.9555555556
Chosen learning rate = 0.1
Chosen minibatch size = 1
```

Comparaison

A travers les courbes générées des différents taux d'apprentissages et tailles des minibatch on peut conclure sur les effets de ces variations:

- Un taux d'aprentissage controle la vitesse avec laquelle le modèle apprend. Lorsque le taux d'apprentissage est trop important, l'aprentissage est plus rapide, mais ça peut augmenter par inadvertance plutôt que diminuer l'erreur d'apprentissage. Lorsque le taux d'apprentissage est trop faible, l'apprentissage est plus lent.
- Pour des tailles de minibatch faibles, on observe des fluctuations sur les courbes parce que nous faisons la moyenne d'un petit nombre d'exemples à la fois. A partir des résultats on peut conclure que choisir un nombre faible de taille de minibatch améliore la précision. Et comme Yann LeCun a dit sur Twitter en 2018: "*Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends dont let friends use minibatches larger than 32.*"

b) Implémentation de Adam

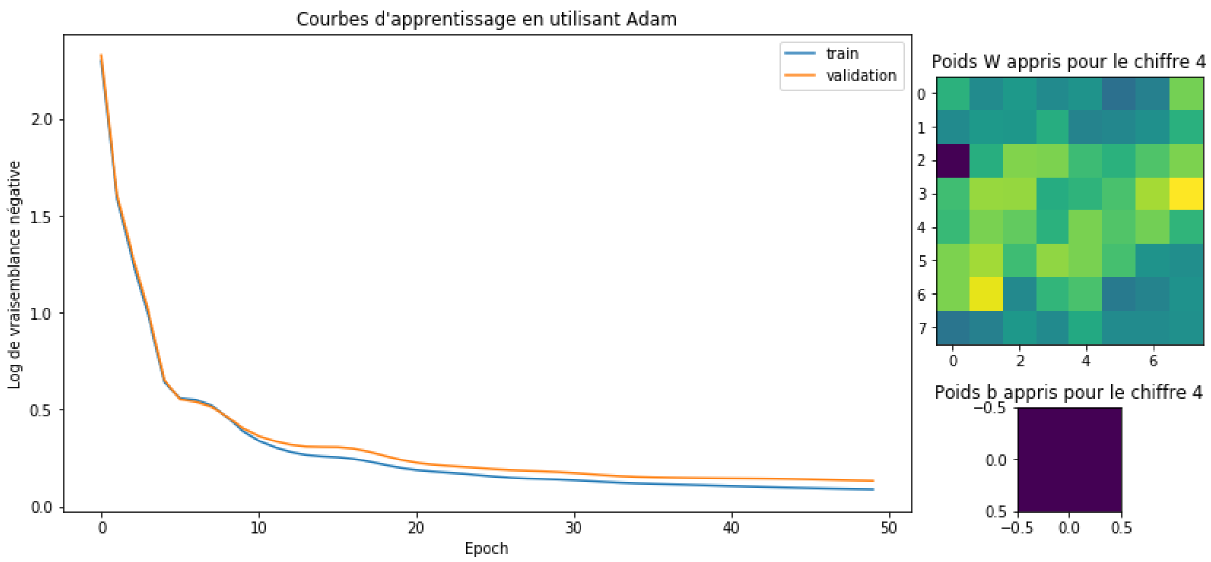
In [0]:

```python
# ADAM
def fit_ADAM(X_train,y_train):
  # initialiser les paramètres de l'algorithme avec celles mentionnées dans le PAPIER
  alpha = 0.01
  beta_1 = 0.9
  beta_2 = 0.999
  epsilon = 1e-8

  m_t = 0
  v_t = 0
  t = 0

  Theta = np.random.normal(0, 0.01, (len(np.unique(y)), X.shape[1]+1))  # weights of shape KxL+1

  best_theta = None
  best_accuracy = 0
  nb_epochs = 50
  losses_train = []
  losses_val = []
  accuracies = []

  for epoch in range(nb_epochs):
      loss = 0
      accuracy = 0

      t+=1
      y_pred = softmax(np.dot(X_train, Theta.T))
      # calcul du gradient de la fonction objective
      g_t = get_grads(y_train, y_pred, X_train)
      # mise à jour des moyennes mobiles du gradient
      m_t = beta_1*m_t + (1-beta_1)* g_t
      # mise à jour des moyennes mobiles du carré du gradient
      v_t = beta_2*v_t + (1-beta_2)* (g_t * g_t)
      # correction des moyennes mobiles
      m_t_c = m_t/(1-(beta_1**t))
      v_t_c = v_t/(1-(beta_2**t))
      # mise à jour des poids Theta
      Theta = Theta - (alpha*m_t_c)/(np.sqrt(v_t_c)+epsilon)

      # compute the loss on the train set
      loss = get_loss(y_train, softmax(np.dot(X_train, Theta.T)))
      losses_train.append(loss)
      # compute the loss on the validation set
      loss = get_loss(y_validation,softmax(np.dot(X_validation, Theta.T)))
      losses_val.append(loss)
      # compute the accuracy on the validation set
      accuracy = get_accuracy(X_validation,y_validation,Theta)
      accuracies.append(accuracy)
      if accuracy > best_accuracy:
          # select the best parameters based on the validation accuracy
          best_accuracy = accuracy
          #best_W = W
          best_theta = Theta
  accuracy_on_unseen_data = get_accuracy(X_test, y_test, best_theta)
  print("Accuracy on validation data:{:.10f}\n".format(best_accuracy))
  print("Accuracy on unseen data:{:.10f}\n".format(accuracy_on_unseen_data))
  return losses_train,losses_val, best_theta, accuracy_on_unseen_data
```

Visualisation des résultats de ADAM

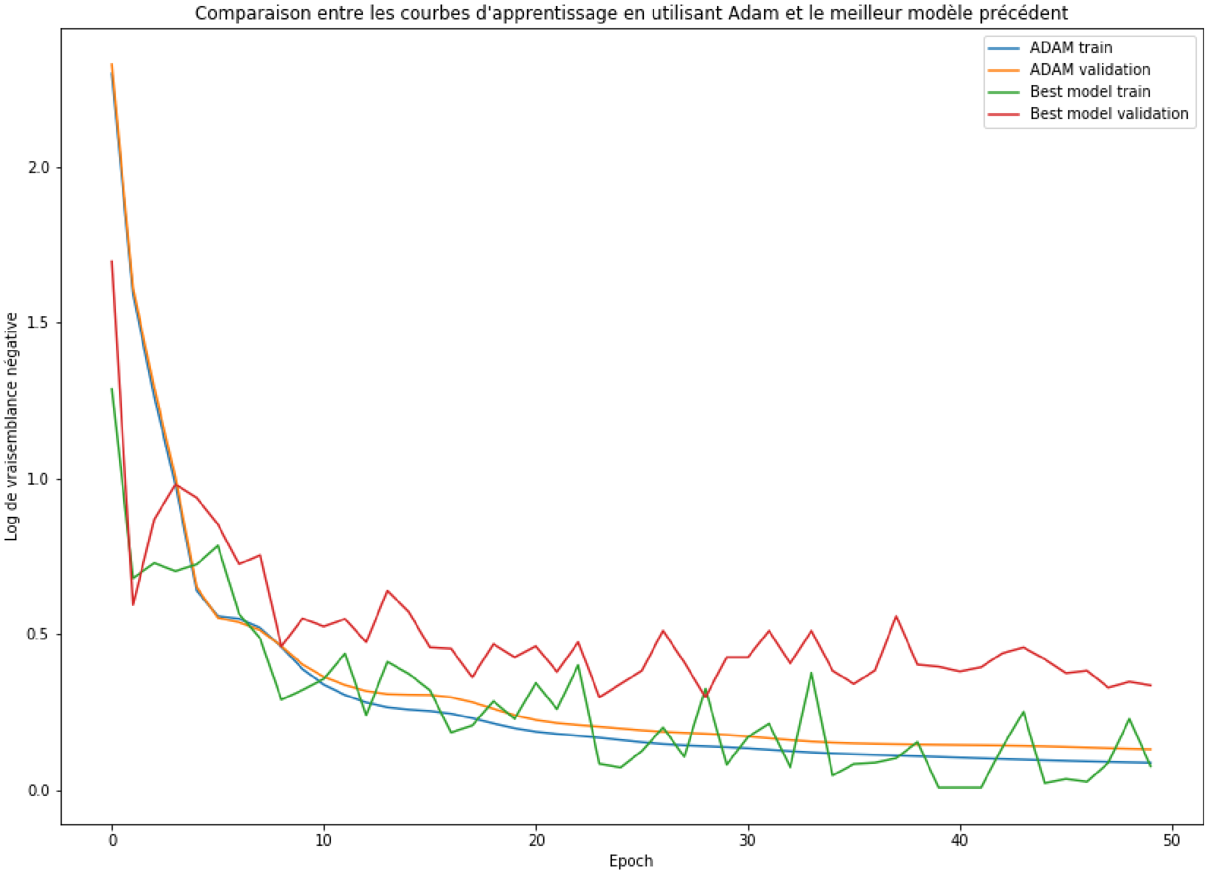
In [54]:
```
plt.figure(figsize = (14,6))
gridspec.GridSpec(4,8)
losses_train_ADAM, losses_val_ADAM, best_theta, accuracy_on_unseen_data_ADAM = fit_ADAM(X_train,y_train
)

best_W = best_theta[:,:64]
best_b = best_theta[:,64:]


ax1 = plt.subplot2grid((4,8), (0,0), colspan=6, rowspan=4)
ax1.plot(losses_train_ADAM, label="train")
ax1.plot(losses_val_ADAM, label="validation")
ax1.set_title('Courbes d\'apprentissage en utilisant Adam')
ax1.set_ylabel('Log de vraisemblance négative')
ax1.set_xlabel('Epoch')
ax1.legend(loc='best')


ax2 = plt.subplot2grid((4,8), (0,6),colspan=2, rowspan=3)
ax2.imshow(best_W[4, :].reshape(8, 8))
ax2.set_title('Poids W appris pour le chiffre 4')


ax3 = plt.subplot2grid((4,8), (3,6),colspan=2, rowspan=1)
ax3.imshow(best_b[4, :].reshape(1,1))
ax3.set_title('Poids b appris pour le chiffre 4')

plt.show()
```

```
Accuracy on validation data:0.9703703704

Accuracy on unseen data:0.9481481481
```



Visualisation des courbes de notre meuilleur modèle choisi et les résultats de ADAM sur le même graphe

In [57]:
```
plt.figure(figsize = (14,10))
gridspec.GridSpec(6,8)

ax1 = plt.subplot2grid((6,8), (0,0), colspan=8, rowspan=6)
ax1.plot(losses_train_ADAM, label="ADAM train")
ax1.plot(losses_val_ADAM, label="ADAM validation")
ax1.plot(chosen_losses_train, label="Best model train")
ax1.plot(chosen_losses_val, label="Best model validation")
ax1.set_title('Comparaison entre les courbes d\'apprentissage en utilisant \
Adam et le meilleur modèle précédent')
ax1.set_ylabel('Log de vraisemblance négative')
ax1.set_xlabel('Epoch')
ax1.legend(loc='best')
plt.show()


print("ADAM : Accuracy on unseen data: {:.10f}\n".format(accuracy_on_unseen_data_ADAM))
print("Selected Model : Accuracy on unseen data: {:.10f}\n".format(chosen_test_accuracy))
```




```
ADAM : Accuracy on unseen data: 0.9481481481

Selected Model : Accuracy on unseen data: 0.9555555556
```

Les courbes obtenues de l'implémentation de l'algorithme d'optimisation ADAM sont lisses et montre que la courbe de validation suit la courbe d'apprentissage avec une petite erreur.

Bien que l'implémentation de l'algorithme d'optimisation ADAM a montré des bons résultats, elle nous donne une précision un peu plus pire que le modèle choisi de minibatch gradient descent (avec learning rate = 0.1 et minibatch size = 1). En effet, récemment plusieurs chercheurs dans la communauté ont fait cette observation que parfois dans certains cas ADAM a une pauvre généralisation en se comparant avec SGD et minibatch gradient decent.