In [1]:

```
import os
from pathlib import Path

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

import torch

BATCH_SIZE = 64
CUDA = torch.cuda.is_available()
LR = 0.1
EPOCHS= 200
```

In [2]:

```
!rm -rf PoS-Tagging
!git clone https://github.com/Janluke0/PoS-Tagging/
os.chdir('PoS-Tagging')
out_dir = Path('/kaggle/working/')
out_dir.mkdir(exist_ok=True)
```

```
Cloning into 'PoS-Tagging'...
remote: Enumerating objects: 77, done.
remote: Counting objects: 100% (77/77), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 77 (delta 27), reused 71 (delta 21), pack-reused 0
Unpacking objects: 100% (77/77), 658.28 KiB | 2.08 MiB/s, done.
```

In [3]:

```
from model.lstm import LSTMTagger
from model import train_model
from dataset import TWITADS
```

In [4]:

```
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence
from transformers import AutoTokenizer
```

In [5]:

```
def tokenize_and_align_labels(tokenizer, tokens, tags, tag_mode='all'):
    tokens = list(tokens)
    tokenized_inputs = tokenizer(tokens, truncation=True, is_split_into_words=True)

    word_ids = tokenized_inputs.word_ids(batch_index=0)  # Map tokens to their respectiv
e word.
    previous_word_idx = None
    label_ids = []
    if tag_mode=='first' or tag_mode =='all':
        for word_idx in word_ids:                            # Set the special tokens to
-100
            if tag_mode=='first':
                if word_idx is None:
                    label_ids.append(-100)
                elif word_idx != previous_word_idx:              # Only label the first
token of a given word.
                    label_ids.append(tags[word_idx])
                    previous_word_idx = word_idx
                else:
                    label_ids.append(-100)
            elif tag_mode=='all':
                if word_idx is None:
                    label_ids.append(-100)
                else:
                    label_ids.append(tags[word_idx])

        tokenized_inputs["labels"] = label_ids
    elif tag_mode == 'last':
        for word_idx in word_ids[::-1]:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:              # Only label the first token of a given word.
                label_ids.append(tags[word_idx])
                previous_word_idx = word_idx
            else:
                label_ids.append(-100)

        tokenized_inputs["labels"] = label_ids[::-1]

    return torch.tensor(tokenized_inputs['input_ids']), torch.tensor(tokenized_inputs['labels'])
```

In [6]:

```
def collate_fn(batch):
    tokens, tags = zip(*batch)
    return pad_sequence(tokens, batch_first=True), pad_sequence(tags, padding_value=-100, batch_first=True)


tknzr = AutoTokenizer.from_pretrained("dbmdz/bert-base-italian-cased")

def mk_dl(tag_mode, ds_names=['train', 'test']):
    def transformer(tkns, tags):
        return tokenize_and_align_labels(tknzr,tkns,tags,tag_mode)
    word_tokenizer = lambda w: [w]
    ds_train = TWITADS(ds_names[0], word_tokenizer,
                       transform=transformer, tag_mode=tag_mode)
    ds_test = TWITADS(ds_names[1], word_tokenizer,
                      transform=transformer, tag_mode=tag_mode)
    return (
        ds_train.n_tags,
        DataLoader(ds_train, shuffle=True,
                   batch_size=BATCH_SIZE, collate_fn=collate_fn),
        DataLoader(ds_test, shuffle=True,
                   batch_size=BATCH_SIZE, collate_fn=collate_fn)
    )
```

In [7]:

```
N_TOKENS = tknzr.vocab_size
DROPOUT = 0.1
def mk_from_key(key, ds_names=['train', 'test']):
    is_bi, l_layers, hid_dim, o_layers, special_tkns, tg_mode = key.split('_')

    is_bi, l_layers, hid_dim, o_layers = is_bi == 'bi', int(
        l_layers), int(hid_dim), int(o_layers)
    #special_tkns is ignored with this tokenizer(btw ot bery useful)
    n_tags, dl_tr, dl_te = mk_dl(tg_mode, ds_names)
    m = LSTMTagger(
        N_TOKENS,
        n_tags,
        hidden_dim=hid_dim,
        dropout=DROPOUT,
        lstm_layers=l_layers,
        bidirectional=is_bi,
        output_layers=o_layers
    )

    return m, dl_tr, dl_te
```

In [8]:

```
def do_train(key):
    model, dl_tr, dl_val = mk_from_key(key,['resampled_train','resampled_validation'])
    loss, acc = train_model(model,dl_tr,dl_val,cuda=CUDA,lr=LR,epochs=EPOCHS,show_plots=
False)
    torch.save(model.state_dict(),out_dir/f"{key}.pth")
    with (out_dir/f"{key}.csv").open("w+") as f:
        f.write(",".join(map(str,loss)))
        f.write("\n")
        f.write(",".join(map(str,acc)))
        f.write("\n")
    return model, loss, acc
```

In [9]:

```
top_1 = {
    'accuracy': 'mono_1_32_1_bow_last',
    'alpha': 'bi_2_128_1__last',
    'combined': 'bi_1_64_1__last',
    'explained': 'bi_1_32_1__last',
    'f1': 'mono_1_128_1_#ow_last'
}
top_1_all = {
    'accuracy': 'mono_1_64_1__all',
    'alpha': 'mono_2_16_1_eow_all',
    'combined': 'mono_1_128_1_eow_all',
    'explained': 'mono_2_32_1__all',
    'f1': 'mono_1_128_1_eow_all'
}
```

Top all tagging modes

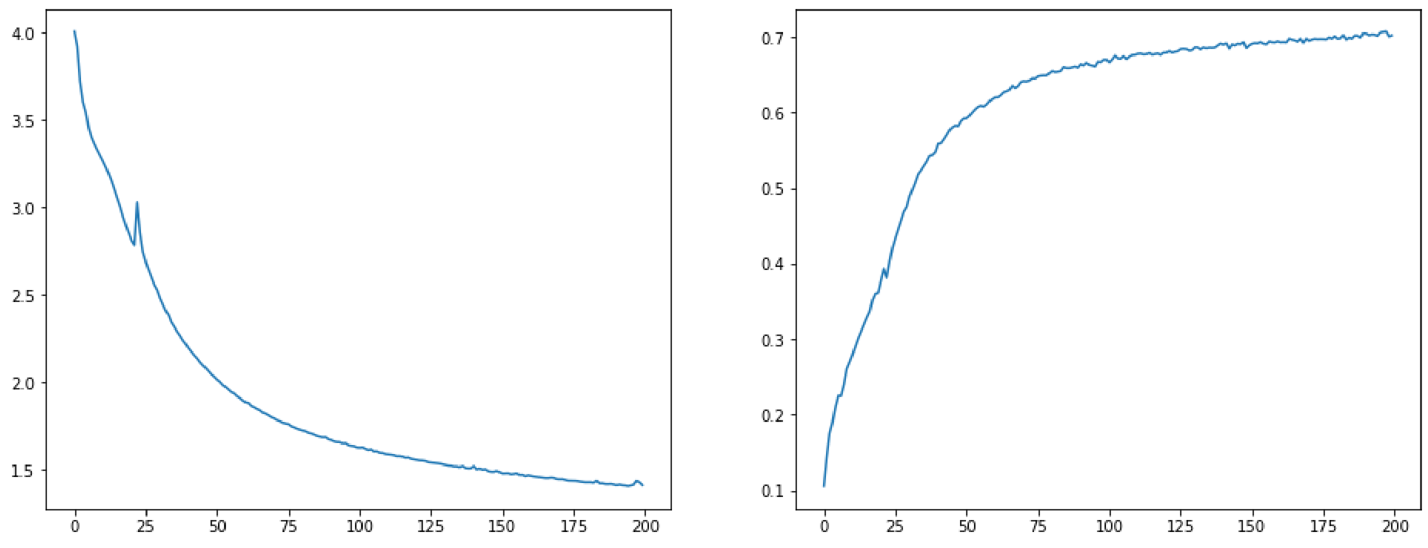
accuracy

In [10]:

```
_,loss,acc = do_train(top_1['accuracy'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x7fa5424499d0>]
```



alpha

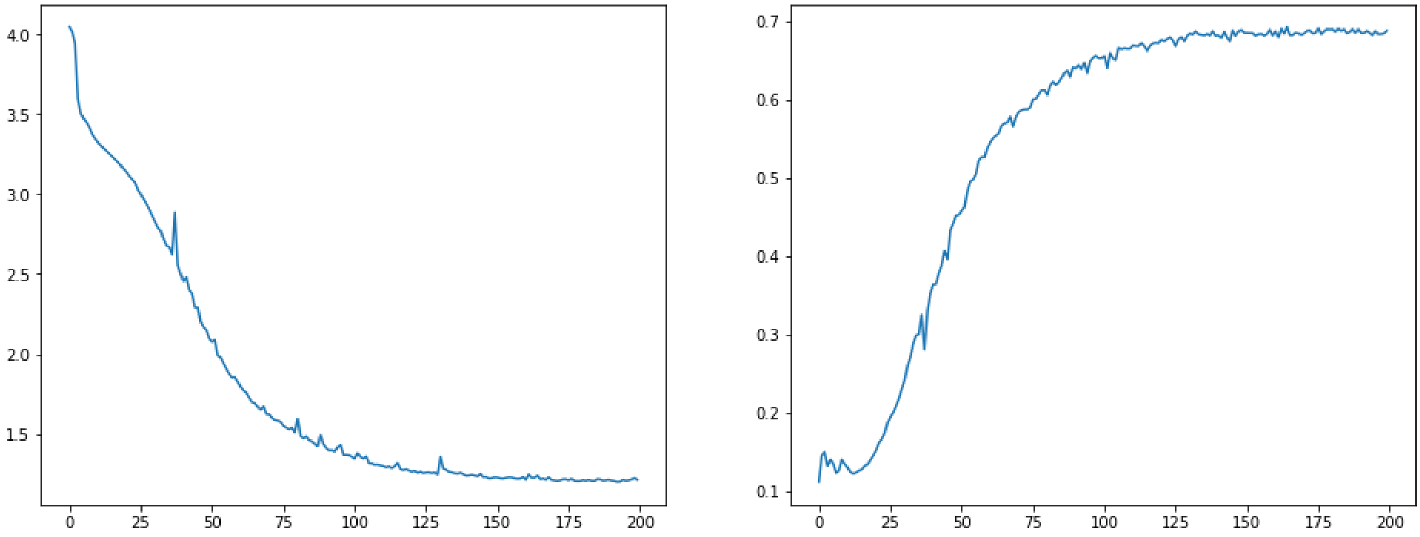
alpha

In [11]:

```
_,loss,acc = do_train(top_1['alpha'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x7fa5135b9110>]
```



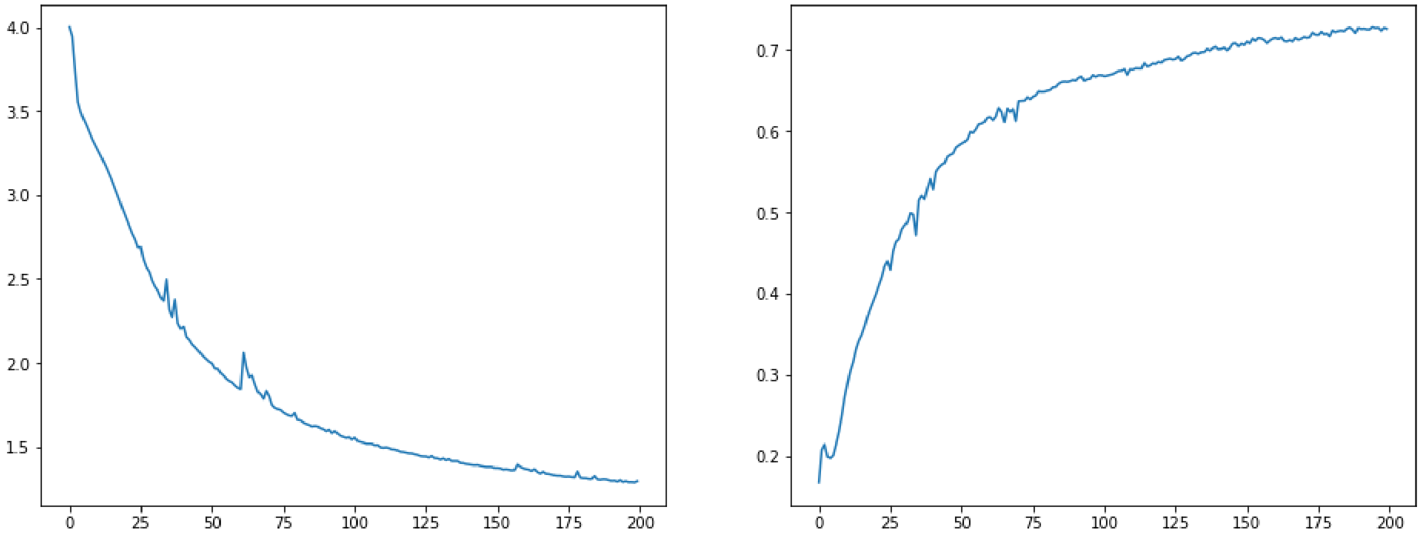
combined

In [12]:

```
_,loss,acc = do_train(top_1['combined'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[12]:

```
[<matplotlib.lines.Line2D at 0x7fa4f83992d0>]
```



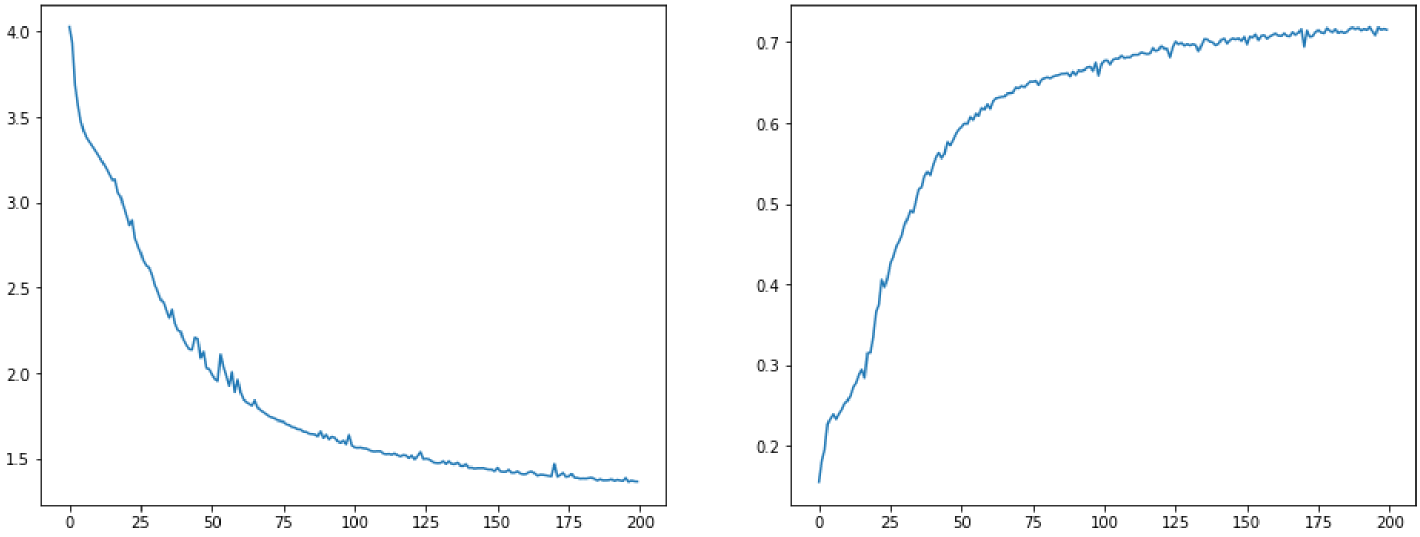
f1

In [13]:

```
_,loss,acc  = do_train(top_1['f1'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7fa4f07780d0>]
```



Top only tag all (tokens) mode

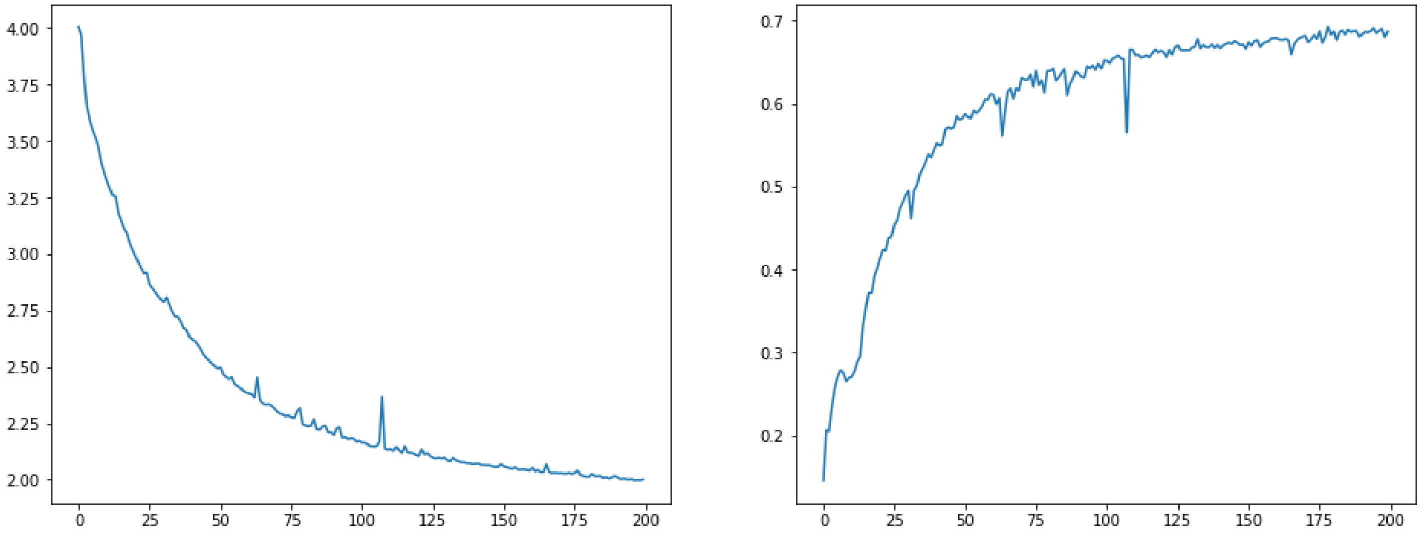
accuracy

In [14]:

```
_,loss,acc = do_train(top_1_all['accuracy'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x7fa478c36650>]
```



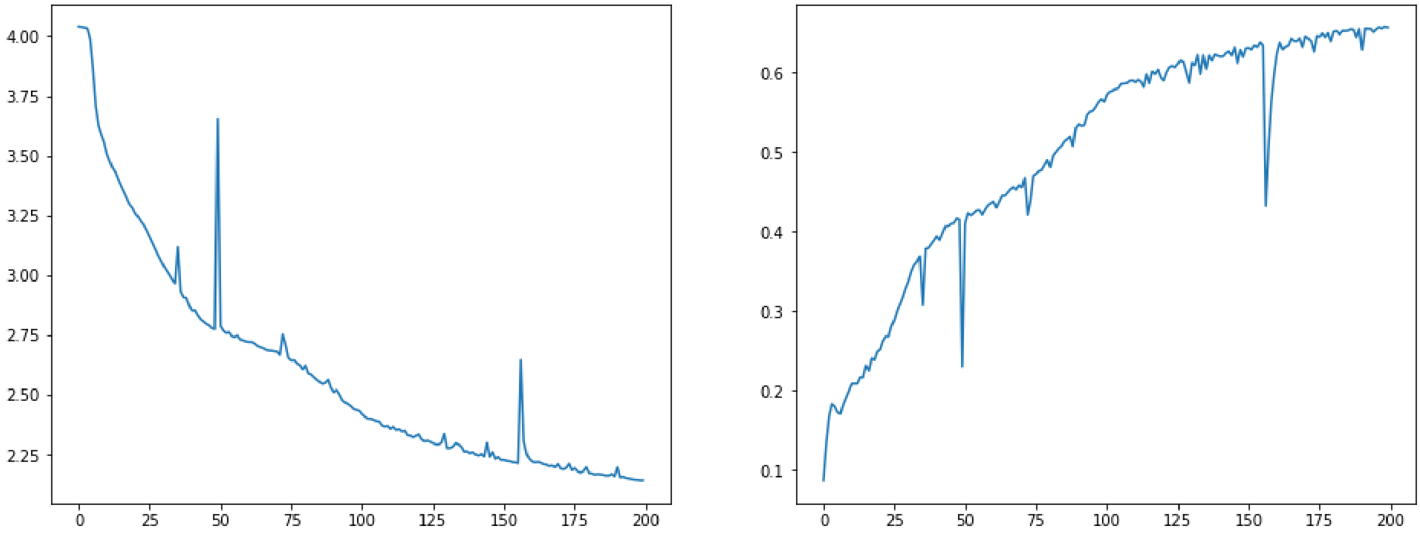
alpha

In [15]:

```
_,loss,acc = do_train(top_1_all['alpha'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[15]:

```
[<matplotlib.lines.Line2D at 0x7fa39106b610>]
```



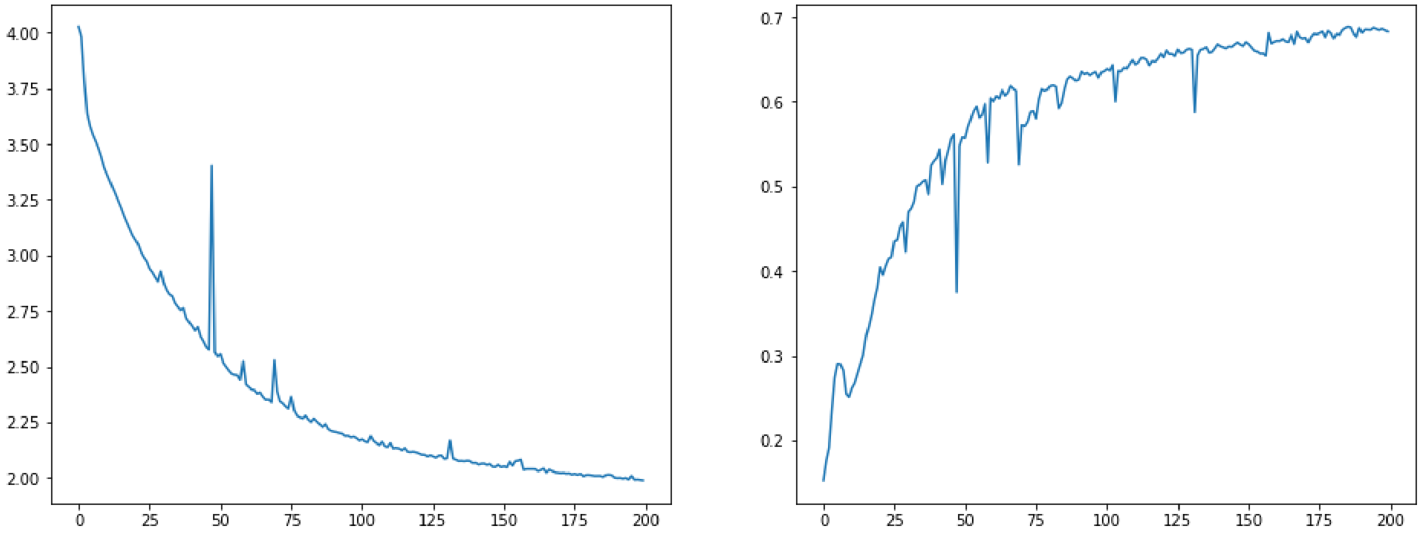
combined

In [16]:

```
_,loss,acc = do_train(top_1_all['combined'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x7fa38f467e90>]
```



f1

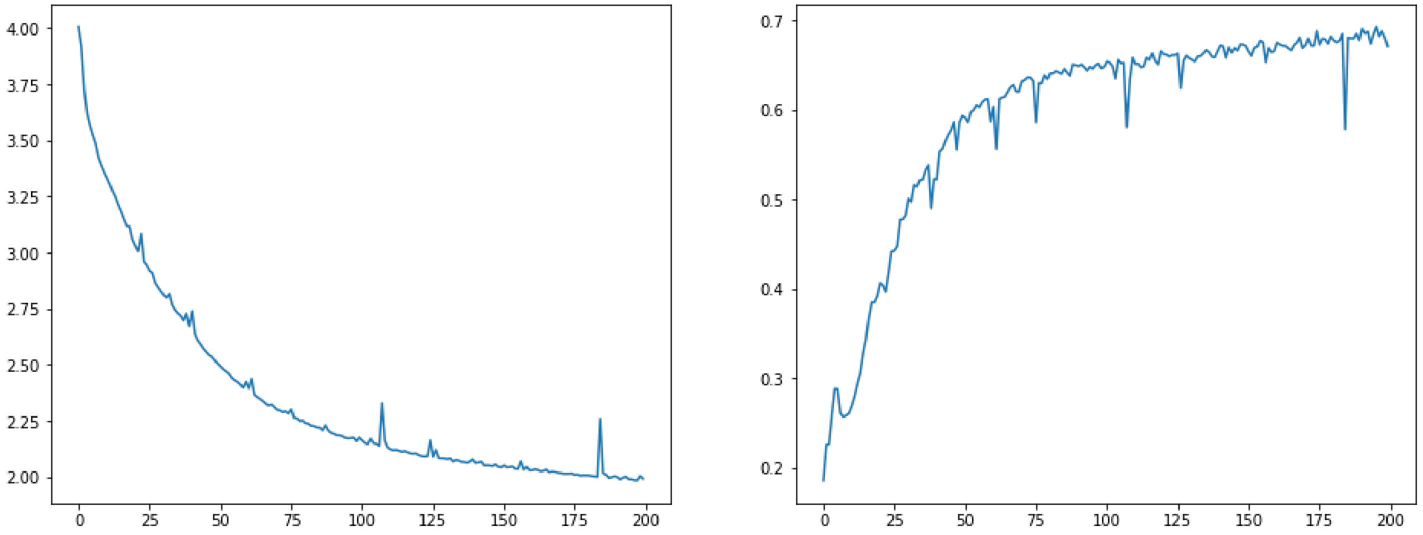
```
''
```

In [17]:

```
_,loss,acc = do_train(top_1_all['f1'])
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(loss)
plt.subplot(122)
plt.plot(acc)
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x7fa5403ad4d0>]
```


4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
0
25
50
75
100
125
150
175
200
0.7
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
125
150
175
200